ОКПД2 26.51.70

ТН ВЭД 9032 89 000 0

Многофункциональный универсальный контроллер «Saturn-PLC»

Руководство по эксплуатации

Часть 9. Режим «С-Скрипт»

ЕСАН.426469.019РЭ8

Редакция от 11.01.2023

©МНПП САТУРН, 2023

Подп. и дата	
Инв.№ дубл.	
Взам. инв.№	
Подп. и дата	
Инв.№ подл.	

СОДЕРЖАНИЕ

1. Введение 16

2. Язык С-Скрипт 16

2.1. Типы данных 16

2.2. Переменные 17

2.3. Размещение переменных в памяти 21

2.4. Квалификатор const 22

2.5. Инициализация переменных 23

2.6. Константы 23

2.7. Операции 25

2.8. Выражения 36

2.9. Структура программы 37

2.10. Комментарии 38

2.11. Операторы 39

2.11.1. Логические значениия ИСТИНА (True) и ЛОЖЬ (False) в языке С 39

2.11.2. Условные операторы 39

2.11.3. Оператор '?' 41

2.11.4. Оператор выбора - switch 42

2.11.5. Оператор цикла for 44

2.11.6. Оператор цикла while 48

2.11.7. Оператор цикла do-while 50

2.11.8. Операторы перехода 50

2.11.9. Оператор-выражение 53

2.11.10. Блок операторов 53

2.12. Массивы и строки 53

2.12.1. Одномерные массивы 54
2.12.2. Размещение в памяти 55
2.12.3. Создание указателя на массив 57
2.12.4. Передача одномерного массива в функцию 57
2.12.5. Строки 58
2.12.6. Двухмерные массивы 60
2.12.7. Индексация указателей 60
2.12.8. Инициализация массивов 60
2.13. Указатели 62
2.13.1. Что такое указатели 62
2.13.2. Указательные переменные 62
2.13.3. Операции для работы с указателями 63
2.13.4. Указательные выражения 63
2.13.5. Указатели и массивы 68
2.13.6. Инициализация указателей 70
2.13.7. Трудности при работе с указателями 70
2.14. Функции 72
2.14.1. Область действия функции 73
2.14.2. Аргументы функции 73
2.14.3. Оператор return 77
2.14.4. Рекурсия 81
2.14.5. Прототип функции 82
2.14.6. Объявление списков параметров переменной длины 83
2.15. Препроцессор 85
2.15.1. Директива #define 85
2.15.2. Директива #error 87
2.15.3. Директива #include 87
2.15.4. Директивы условной компиляции 88

2.15.5. Директива #undef ... 91
2.15.6. Использование defined ... 91
2.15.7. Директива #pragma ... 92
2.15.8. Операторы препроцессора # и ## ... 94
2.16. Средство typedef ... 94
2.17. Использование sizeof ... 95
2.18. Перечисления enum ... 96
2.19. Объединения union ... 97
2.20. printf ... 98
2.21. sprintf ... 101
2.22. string.h ... 102
2.22.1. memcpy ... 103
2.22.2. memmove ... 104
2.22.3. memchr ... 105
2.22.4. memcmp ... 106
2.22.5. memset ... 107
2.22.6. strcat ... 107
2.22.7. strchr ... 108
2.22.8. strcmp ... 109
2.22.9. strcpy ... 109
2.22.10. strcspn ... 110
2.22.11. strlen ... 110
2.22.12. strncat ... 111
2.22.13. strncmp ... 112
2.22.14. strncpy ... 112
2.22.15. strpbrk ... 113
2.22.16. strrchr ... 113
2.22.17. strspn ... 114

2.22.18. strstr ... 114
2.22.19. strtok ... 115
2.22.20. atoi ... 116
2.23. ctype.h ... 117
2.23.1. isalnum ... 117
2.23.2. isalpha ... 118
2.23.3. isblank ... 119
2.23.4. iscntrl ... 119
2.23.5. isdigit ... 120
2.23.6. isgraph ... 121
2.23.7. islower ... 121
2.23.8. isprint ... 122
2.23.9. ispunct ... 123
2.23.10. isspace ... 123
2.23.11. isupper ... 124
2.23.12. isxdigit ... 125
2.23.13. toupper ... 125
2.23.14. tolower ... 126
2.24. math.h ... 126
2.24.1. sin ... 127
2.24.2. cos ... 128
2.24.3. tan ... 129
2.24.4. asin ... 130
2.24.5. acos ... 131
2.24.6. atan ... 132
2.24.7. sinh ... 133
2.24.8. cosh ... 134
2.24.9. tanh ... 135

2.24.10. pow ... 137
2.24.11. log ... 137
2.24.12. log10 ... 138
2.24.13. exp ... 139
2.24.14. sqrt ... 141
2.25. limits.h ... 142
3. CAT-500 ... 142
3.1. Входы и выходы CAT-500 ... 142
3.1.1. Входы CAT-500 ... 143
3.1.2. Выходы CAT-500 ... 145
3.2. Системное меню ... 146
3.2.1. Пункты меню ... 146
3.2.2. Вызов системного меню ... 154
3.2.3. Названия сигналов ... 155
3.3. Дисплей ... 159
3.3.1. BigFont ... 160
3.3.2. BrushColor ... 161
3.3.3. ClrScr ... 162
3.3.4. DrawPicture ... 163
3.3.5. Ellipse ... 164
3.3.6. EllipsePen ... 165
3.3.7. FontColor ... 166
3.3.8. GotoXY ... 167
3.3.9. LineTo ... 168
3.3.10. PenColor ... 169
3.3.11. PenWidth ... 170
3.3.12. printf ... 171
3.3.13. PrintInt ... 175

3.3.14. PrintIntLen 176
3.3.15. putchar 177
3.3.16. puts 178
3.3.17. RGB 179
3.3.18. Rectangle 180
3.3.19. RectangleBrush 181
3.3.20. RectanglePen 181
3.3.21. SetPixel 182
3.3.22. ShowScr 183
3.3.23. Цвет 184
3.3.24. Константы цветов 184
3.3.25. Размеры дисплея 185
3.4. Время 185
3.4.1. GetTickCount 186
3.4.2. GetTime 187
3.4.3. GetDate 188
3.4.4. GetDateTime 189
3.4.5. SecOfTime 190
3.4.6. MinOfTime 192
3.4.7. HourOfTime 193
3.4.8. DayOfTime 194
3.4.9. MonthOfTime 195
3.4.10. YearOfTime 197
3.4.11. WeekDayOfTime 198
3.4.12. YearDayOfTime 199
3.4.13. MkTime 201
3.4.14. TimeOfSunrise 202
3.4.15. TimeOfSunset 204

3.4.16. Производственный календарь ... 206
3.4.16.1. Программа "PlcCalendar" ... 209
3.5. Таймеры ... 217
3.5.1. StartTimer ... 218
3.5.2. StopTimer ... 220
3.6. Клавиатура ... 222
3.6.1. EnableKeyboard ... 222
3.6.2. GetKey ... 223
3.6.3. PressKey ... 225
3.7. Звуковой сигнал ... 227
3.7.1. Beep ... 227
3.8. Энергонезависимая память EPROM ... 229
3.8.1. ReadEprom ... 230
3.8.2. WriteEprom ... 231
3.9. Энергонезависимая память BACKUPRAM ... 231
3.9.1. BackupRamRead8 ... 233
3.9.2. BackupRamWrite8 ... 233
3.9.3. BackupRamRead16 ... 234
3.9.4. BackupRamWrite16 ... 234
3.9.5. BackupRamRead32 ... 235
3.9.6. BackupRamWrite32 ... 236
3.9.7. BackupRamRead ... 237
3.9.8. BackupRamWrite ... 237
3.10. Дополнительная память EXTRAM ... 238
3.10.1. GetExtRamSize ... 239
3.10.2. ReadExtRamByte ... 240
3.10.3. ReadExtRamWord ... 241
3.10.4. ReadExtRamUint ... 242

3.10.5. WriteExtRamByte ... 243
3.10.6. WriteExtRamWord ... 245
3.10.7. WriteExtRamUint ... 246
3.10.8. ReadExtRam ... 247
3.10.9. WriteExtRam ... 248
3.11. Случайное число ... 250
3.11.1. rand ... 250
3.11.2. srand ... 251
3.12. TCP клиент ... 252
3.12.1. TcpOpen ... 253
3.12.2. TcpClose ... 255
3.12.3. TcpConnected ... 256
3.12.4. TcpState ... 256
3.12.5. TcpWrite ... 257
3.12.6. TcpReadLength ... 257
3.12.7. TcpRead ... 257
3.12.8. TcpClear ... 258
3.12.9. TcpLost ... 258
3.12.10. Пример ... 258
3.13. MODBUS TCP ... 264
3.13.1. MODBUS TCP Master ... 264
3.13.1.1. ModbusTcpSlave ... 267
3.13.1.2. ModbusTcpTimeout ... 267
3.13.1.3. ModbusTcpOpen ... 267
3.13.1.4. ModbusTcpConnected ... 269
3.13.1.5. ModbusTcpClose ... 269
3.13.1.6. ModbusTcpResult ... 269
3.13.1.7. ModbusTcpReadCoils ... 271

3.13.1.8. ModbusTcpReadDiscreteInputs 273
3.13.1.9. ModbusTcpReadHoldingRegisters 275
3.13.1.10. ModbusTcpReadInputRegisters 277
3.13.1.11. ModbusTcpWriteSingleCoil 278
3.13.1.12. ModbusTcpWriteHoldingRegister 279
3.13.1.13. ModbusTcpWriteRegisters 281
3.13.1.14. ModbusTcpWriteCoils 282
3.13.2. MODBUS TCP Slave 284
3.13.2.1. StartModbusTcpSlave 286
3.13.2.2. StopModbusTcpSlave 287
3.13.2.3. ModbusTcpCount 288
3.14. Интерфейс RS-485 288
3.14.1. Базовые функции и переменные 288
3.14.1.1. SerialOpen 289
3.14.1.2. SerialClose 290
3.14.1.3. SerialWrite 291
3.14.1.4. SerialReadLength 292
3.14.1.5. SerialRead 293
3.14.1.6. SerialIsOpen 295
3.14.2. MODBUS RTU Master 295
3.14.2.1. ModbusSlave 298
3.14.2.2. ModbusTimeout 300
3.14.2.3. ModbusResult 301
3.14.2.4. ModbusReadCoils 303
3.14.2.5. ModbusReadDiscreteInputs 305
3.14.2.6. ModbusReadHoldingRegisters 307
3.14.2.7. ModbusReadInputRegisters 309
3.14.2.8. ModbusReadStatus 310

3.14.2.9. ModbusWriteSingleCoil 310
3.14.2.10. ModbusWriteHoldingRegister 312
3.14.2.11. ModbusWriteRegisters 313
3.14.2.12. ModbusWriteCoils 315
3.14.2.13. ModbusCrc16 315
3.14.3. MODBUS RTU Slave 316
3.14.3.1. StartModbusSlave 316
3.14.3.2. StopModbusSlave 318
3.14.3.3. ModbusCount 320
3.15. Клиент LanMon сервера 320
3.15.1. Сервер LanMon 321
3.15.1.1. Установка сервера 324
3.15.1.2. Вызов окна приложения сервера 324
3.15.1.3. Настройка авторизации 325
3.15.1.4. Добавление каналов 328
3.15.1.5. Просмотр каналов 330
3.15.1.6. Настройка номера порта сервера 331
3.15.2. LmChanActivity 332
3.15.3. LmChanNum 335
3.15.4. LmChanRegister 337
3.15.5. LmClose 340
3.15.6. LmConnected 341
3.15.7. LmGetCtrlName 342
3.15.8. LmGetCtrlValue 345
3.15.9. LmOpen 348
3.15.10. LmSendByName 349
3.15.11. LmSendByNum 351
3.15.12. LmState 353

3.15.13. Пример ... 354

3.16. Сетевые переменные ... 361

3.16.1. NetVarsPacketCount ... 362

3.16.2. NetVarsStart ... 362

3.16.3. NetVarsStop ... 366

3.16.4. NetVarsSend ... 368

3.16.5. Просмотр сетевых переменных ... 370

3.16.6. Настройки сетевых переменных ... 371

3.17. Клиент сервера Postgre SQL ... 373

3.17.1. PgCell ... 376

3.17.2. PgCellAsInteger ... 378

3.17.3. PgClose ... 381

3.17.4. PgColCount ... 383

3.17.5. PgColName ... 385

3.17.6. PgConnected ... 388

3.17.7. PgDataLen ... 389

3.17.8. PgErrorCode ... 392

3.17.9. PgErrorText ... 401

3.17.10. PgOpen ... 404

3.17.11. PgRowCount ... 410

3.17.12. PgSql ... 412

3.17.13. PgState ... 418

3.17.14. PgTimeout ... 420

3.17.15. Отладка работы с Postgre SQL ... 422

3.18. HTTP сервер ... 424

3.18.1. Протокол HTTP ... 425

3.18.2. Данные ответа по умолчанию ... 426

3.18.3. Использование браузера для проверки ... 428

3.18.4. HttpDataCount ... 429
3.18.5. StartHttpData ... 432
3.18.6. StopHttpData ... 433
3.18.7. JSON ... 433
3.18.8. HTML ... 435
3.18.9. TEXT ... 438
3.18.10. Пользовательский ... 439
3.19. Дополнительные функции ... 442
3.19.1. GetModbusFloat ... 443
3.19.2. GetFloat ... 444
3.19.3. GetDouble ... 444
3.19.4. CreateFloat ... 445
3.19.5. CreateDouble ... 446
3.19.6. abs ... 447
3.20. Ethernet ... 447
3.20.1. EthernetRestart ... 447
3.20.2. EthernetLink ... 448
3.21. Параметры САТ-500 ... 449
3.21.1. SatSerNumber ... 449
3.21.2. SatIp ... 451
3.21.3. SatMask ... 451
3.21.4. SatGateway ... 452
3.21.5. SatDns ... 453
3.21.6. SatDhcp ... 454
3.22. Ошибки ... 455
3.22.1. ADCErrors ... 456
3.23. Обновление программы через Интернет ... 457
3.24. БКД-ПК и САТ-500 ... 459

3.25. Регистры ... 462

4. Среда разработки ... 462

4.1. Исходный текст и исполняемая программа ... 464

4.1.1. Экспорт программы ... 464

4.1.2. Запись программы в CAT-500 из файла ... 465

4.1.3. Чтение программы из CAT-500 в файл ... 468

4.1.4. Информация о файле программы ... 469

4.2. Основное меню ... 471

4.3. Редактор ... 476

4.3.1. Предложение кода ... 477

4.3.2. Вставка/удаление комментариев ... 478

4.3.3. Автозавершение текста ... 479

4.3.4. Поиск фигурной скобки ... 483

4.3.5. Поиск в текущем окне редактирования ... 483

4.3.6. Быстрый поиск ... 484

4.3.7. Поиск в файлах ... 484

4.3.8. Замена текста ... 485

4.3.9. Отменить/вернуть последнее действие ... 486

4.3.10. "Горячие" клавиши ... 487

4.4. Компиляция ... 488

4.5. Отладка в имитаторе ... 492

4.5.1. Окно RAM ... 498

4.5.2. Окно ROM ... 499

4.5.3. Окно BACKUPRAM ... 500

4.5.4. Окно EPROM ... 501

4.5.5. Вкладка "Переменные" ... 503

4.5.6. Вкладка "Функции" ... 505

4.5.7. Вкладка "Дерево функций" ... 508

4.5.8. Вкладка "Файлы проекта" 509

4.5.9. Вкладка "Вычислитель" 510

4.5.10. Дополнительные вкладки 511

4.6. Имитатор CAT-500 512

4.7. Настройка программы 516

4.8. Отладка в CAT-500 524

4.9. Управление CAT-500 531

4.10. Обновление программы компилятора 534

4.11. Удаление С-скрипт программы 535

1. ВВЕДЕНИЕ

"**С-Скрипт**" является скриптовым языком, предназначенным для создания собственных приложений для контроллера CAT-500, работающего в режиме "C-Script".
Синтаксис языка практически совпадает с классическим C89 с некоторыми ограничениями:

- Отсутствуют плавающие типы *float* и *double*
- Отсутствует возможность создавать собственные типы (структуры *struct)*
- Определение *enum* может быть только неименованным (не может быть типом)
- Объединение *union* может быть только неименованным (не может быть типом)
- Отсутствуют указатели на функции
- Отсутствуют многомерные массивы
- Не используются спецификаторы *static, register, auto, volatile*
- Отсутствуют операторы препроцессора # и ##
- Ограниченная поддержка функций стандартной библиотеки C

Компилятор является скриптовым, т.к. он не создаёт машинный код для непосредственного исполнения процессором. Вместо этого он создаёт код в виде команд псевдоассемблера для виртуальной машины. Контроллер CAT-500 в режиме "C-Script" имеет в своём составе виртуальную машину, которая умеет исполнять код псевдоассемблера, созданный компилятором.

В разделе *Язык С-Скрипт* можно подробно познакомится с синтаксисом языка "**С-Скрипт**", в разделе *CAT-500* приведено описания расширений для взаимодействия программы с контроллером CAT-500. А в разделе *Среда разработки* можно ознакомиться с процессом разработки собственного приложения для CAT-500.

*Имеется несколько разновидностей контроллеров CAT-500. Все эти контроллеры выпускаются под единым названием торговым названием **Saturn-PLC**. Контроллеры отличаются используемым типом процессора, и могут иметь непринципиальные аппаратные отличия.*
*Все контроллеры серии **Saturn-PLC** в среде разработки "С-Скрипт" называются CAT-500.*

2. ЯЗЫК С-СКРИПТ

Язык С-Скрипт является подмножеством стандартного языка C89.

2.1. ТИПЫ ДАННЫХ

Стандарт C определяет пять фундаментальных типов данных: char — символьные данные, int — целые, float — с плавающей точкой, double — двойной точности, void — без значения. На основе этих типов формируются другие типы данных.

Компилятор "С-Скрипт" не поддерживает плавающие типы float и double.

Базовые типы данных (кроме void) могут иметь дополнительные спецификаторы, предшествующие им в тексте программы. Спецификатор типа так изменяет значение базового типа, чтобы он более точно соответствовал своему назначению в программе. Список спецификаторов типов:

```
unsigned
short
```

Базовые типы ***int*** и ***char*** могут быть модифицированы одним из этих спецификаторов. Тип ***char*** модифицируется с помощью ***unsigned***. В таблице ниже приведены все допустимые комбинации типов данных с их минимальным диапазоном значений и типичным размером.

Объект типа ***char*** всегда занимает один байт. Объект ***int*** занимает четыре байта. Дополнительно имеется тип ***short int***, объект которого занимает два байта.

Таблица 1 - Все типы данных, поддерживаемые Компилятором "С-Скрипт"

Тип	***Размер в битах***	***Допустимый диапазон значений***
char	8	от -127 до 127
unsigned char	8	от 0 до 255
short int	16	от -32767 до 32767
unsigned short int	16	от 0 до 65535
int	32	от -2 147 483 647 до 2 147 483 647
unsigned int	32	от 0 до 4 294 967 295

Тип ***void*** служит для объявления функции, не возвращающей значения, или для создания универсального указателя.
Целые числа со знаком и без знака отличаются интерпретацией старшего бита числа. Если целое объявлено со знаком, компилятор считает, что старший бит содержит знак числа. Если в старшем бите записан 0, число считается положительным, а если 1 - отрицательным.
Отрицательные числа представлены в двоичном дополнительном коде. Это значит, что для отрицательного числа все биты, кроме старшего, инвертируются, к полученному числу добавляется 1, а старший бит устанавливается в 1.
Целые числа со знаком используются почти во всех алгоритмах, но абсолютная величина наибольшего из них составляет примерно только половину максимального целого без знака. Например, знаковое целое число 32767 (***short int***) в двоичном коде имеет вид
01111111 11111111
Если в старший бит записать 1, то оно будет интерпретироваться как -1. Однако если полученную запись рассматривать как представление числа, объявленного как ***unsigned short int***, то оно будет интерпретироваться как 65535.
Если спецификатор типа записать сам по себе (без следующего за ним базового типа), то предполагается, что он модифицирует тип ***int***. Таким образом, следующие спецификаторы типов эквивалентны:

Тип	***Эквивалентный тип***
short	short int
unsigned	unsigned int

Хотя базовый тип ***int*** и предполагается по умолчанию, его, тем не менее, обычно указывают явно.

2.2. ПЕРЕМЕННЫЕ

Переменная - это именованный участок памяти, в котором хранится значение, которое может быть изменено программой. Все переменные перед их использованием должны быть объявлены. Общая форма объявления имеет такой вид:

тип список_переменных;

Здесь *тип* означает один из базовых или объявленных программистом типов (если необходимо — с одним или несколькими спецификаторами), а *список_переменных* состоит из одного или более идентификаторов, разделенных запятыми. Ниже приведены примеры объявлений:

```
char var;
```

```
int i,j,l;
short int si;
unsigned int ui;
```

Необходимо помнить, что в С имя переменной никогда не определяет ее тип.

Имена переменных

В языке С имена переменных, функций, меток и т.п. называются идентификаторами. Длина идентификатора (количество символов, из которых состоит идентификатор) является натуральным числом, обычно идентификатор представляет собой последовательность из одного или нескольких символов. Первый символ должен быть буквой или символом подчеркивания, последующие символы должны быть буквами, цифрами или символами подчеркивания.
Ниже приведены примеры правильных и неправильных записей идентификаторов:

Правильные	***Неправильные***
count	1count
test23	hi!here
High_Balance	high...balance

Верхние и нижние регистры символов рассматриваются как различные. Следовательно, count, Count и COUNT — это три разных идентификатора.
Идентификатор не может совпадать с ключевым словом С или с именем библиотечной функции или с именем функции пользователя.

Компилятор "С-Скрипт" не поддерживает имена переменных длиной более 31 символа.

Где объявляются переменные

Объявление переменных может быть расположено в трех местах: внутри функции, в определении параметров функции и вне всех функций. Это - места объявлений соответственно локальных, формальных параметров функций и глобальных переменных.

Локальные переменные

Переменные, объявленные внутри функций, называются локальными переменными. Иногда их называют динамическими переменными или стековыми из-за расположения переменных.
Локальную переменную можно использовать только внутри блока, в котором она объявлена. Иными словами, локальная переменная невидима за пределами своего блока. (Блок программы - это описания и инструкции, объединенные в одну конструкцию путем заключения их в фигурные скобки).
Локальные переменные существуют только во время выполнения программного блока, в котором они объявлены, создаются они при входе в блок, а разрушаются при выходе из него. Более того, переменная, объявленная в одном блоке, не имеет никакого отношения к переменной с тем же именем, объявленной в другом блоке.
Чаще всего блоком программы, в котором объявлены локальные переменные, является функция. Рассмотрим, например, следующие две функции:

```
void func1(void)
{
  int x;
  x = 10;
}

void func2(void)
{
  int x;
  x = -199;
}
```

Целая переменная x объявлена дважды: один раз в ***func1()*** и второй - в ***func2()***. При этом переменная x в одной функции никак не связана и никак не влияет на переменную с тем же именем в другой функции. Это происходит потому, что локальная переменная видима только внутри блока, в котором она объявлена, за пределами этого блока она невидима.
Из соображений удобства и в силу устоявшейся традиции все локальные переменные функции чаще всего объявляются в самом начале функции, сразу после открывающейся фигурной скобки. Однако можно объявить локальную переменную и внутри блока программы (блок функции - это частный случай блока программы). Например:

```
void f(void)
{
  int t;

  t=GetValt();
  if(t==1)
  {
    char s[80];  /* эта переменная создаётся только
                    при входе в этот блок */
    /* некоторые операторы ... */
  }

  /* здесь переменная s невидима */
}
```

В этом примере локальная переменная s создаётся при входе в блок ***if*** и разрушается при выходе из него. Следовательно, переменная ***s*** видима только внутри блока ***if*** и не может быть использована ни в каких других местах, даже если они находятся внутри функции, содержащей этот блок.
Объявление переменных внутри блока программы помогает избежать нежелательных побочных эффектов. Переменная не существует вне блока, в котором она объявлена, следовательно, "посторонний" участок программы не сможет случайно изменить её значение.
Локальная переменная может быть объявлена в любом месте внутри блока до её первого использования.
Так как локальные переменные создаются и уничтожаются при каждом входе и выходе из блока, их значение теряется каждый раз, когда программа выходит из блока. Это необходимо учитывать при вызове функции. Локальная переменная создаётся при входе в функцию и разрушается при выходе из неё. Это значит, что локальная переменная не сохраняет своё значение в период между вызовами.
По умолчанию локальные переменные хранятся в стеке. Стек - динамически изменяющаяся область памяти. Вот почему в общем случае локальные переменные не сохраняют своё значение в период между вызовами функций.
Локальные переменные можно инициализировать каким-либо заранее заданным значением. Это значение будет присвоено переменной каждый раз при входе в тот блок программы, в котором она объявлена. Например, следующая программа напечатает число 10 пять раз:

```
#include <sat500.h>

void f(void)
{
    int j = 10;
    printf("%d ", j);
    j++;  /* этот оператор не влияет на результат */
}

int main(void)
{
    ClrScr();
    int i;
```

```
    for(i=0; i<5; i++)  f();
    ShowScr();
    while(1);
}
```

Формальные параметры функции

Если функция имеет аргументы, значит должны быть объявлены переменные, которые примут их значения. Эти переменные называются *формальными параметрами* функции. Внутри функции они фигурируют как обычные локальные переменные. Как показано в следующем фрагменте программы, они объявляются после имени функции внутри круглых скобок:

```
/* Возвращает 1, если в строке s содержится символ c, в противном случае воз-
вращает 0 */
int is_in(char * s,char c)
{
    while(*s)
    if(*s==c) return 1;
    else s++;
    return 0;
}
```

Функция ***is_in()*** имеет два параметра: ***s*** и ***c***, она возвращает ***1***, если символ, записанный в переменной ***c***, входит в строку ***s***, в противном случае она возвращает ***0***.
Внутри функции формальные параметры ничем не отличаются от обычных локальных переменных, единственное их отличие состоит в том, что при входе в функцию они получают значения аргументов. Можно, например, присваивать параметру какое-либо значение или использовать его в выражении. Необходимо помнить, что, как и локальные переменные, формальнее параметры тоже являются динамическими переменными и, следовательно, разрушаются при выходе из функции.

Глобальные переменные

В отличие от локальных, *глобальные переменные* видимы, и могут использоваться в любом месте программы. Они сохраняют своё значение на протяжении всей работы программы. Чтобы создать глобальную переменную, её необходимо объявить за пределами функции. Глобальная переменная может быть использована в любом выражении, независимо от того, в каком блоке это выражение используется.
В следующем примере переменная ***count*** объявлена вне каких бы то ни было функций. Её объявление расположено перед ***main()***, однако, оно может находиться в любом месте перед первым использованием этой переменной, но только не внутри функции. Объявлять глобальные переменные рекомендуется в верхней части программы.

```
#include <sat500.h>
int count;  /* глобальная переменная count */

void func1(void);
void func2(void);

void main(void)
{
    count = 100;
    ClrScr();
    func1();
    ShowScr();
    while(1);
}

void func1(void)
{
    int temp;
```

```
    temp = count;
    func2();
    printf("count равно %d", count); /* напечатает 100 */
}
void func2(void)
{
    int count;
    for(count=1; count<10; count++)
    putchar('.');
}
```

Внимательно посмотрите на эту программу. Обратите внимание на то, что ни в ***func1()***, ни в ***func2()*** нет объявления переменной ***count***, однако они обе могут ее использовать. В ***func2()*** эта возможность не реализуется, так как в ней объявлена локальная переменная с тем же именем. Когда внутри ***func2()*** происходит обращение к переменной ***count***, то это будет обращение к локальной, а не глобальной переменной. Таким образом, выполняется следующее правило: если локальная и глобальная переменные имеют одно и то же имя, то при обращении к ней внутри блока, в котором объявлена локальная переменная, происходит ссылка на локальную переменную, а на глобальную переменную это никак не влияет.
Глобальные переменные хранятся в отдельной фиксированной области памяти, созданной компилятором специально для этого. Глобальные переменные используются в тех случаях, когда разные функции программы используют одни и те же данные. Однако рекомендуется избегать излишнего использования глобальных переменных, потому что они занимают память в течение всего времени выполнения программы, а не только тогда, когда они необходимы. Кроме того, и это ещё более важно, использование глобальной переменной делает функцию менее универсальной, потому что в этом случае функция использует нечто, определённое вне её. К тому же большое количество глобальных переменных легко приводит к ошибкам в программе из-за нежелательных побочных эффектов. При увеличении размера программы серьёзной проблемой становится случайное изменение значения переменной где-то в другой части программы, а когда глобальных переменных много, предотвратить это очень трудно.

2.3. РАЗМЕЩЕНИЕ ПЕРЕМЕННЫХ В ПАМЯТИ

Глобальные переменные по умолчанию располагаются в RAM памяти (оперативное запоминающее устройство).
Однако при необходимости можно разместить переменную в ROM памяти (память только для чтения). В этом случае значение переменной будет невозможно изменить. В контроллере CAT-500 в качестве ROM используется FLASH память микроконтроллера.
Для размещения переменной памяти в неизменяемой памяти следует использовать модификатор const следующим образом:

```
const тип имя_переменной=значение;
```

Например создание целочисленной переменной со значением 2020 будет выглядеть следующим образом:

```
const int myvar=2020;
```

Вместо использования таких переменных лучше использовать директиву #define, что приведёт к экономии ресурсов контроллера:

```
#define myvar 2020
```

Модификатор const обычно используется для хранения в программе некоторых неизменяемых данных - таблиц данных или массивов строковых данных.

Выравнивание

Компилятор C-Script не выполняет выравнивание переменных и массивов данных. Другими словами всегда используется побайтовое выравнивание.
Ещё одной особенностью компилятора является то, что относительный порядок расположения переменных точно СООТВЕТСТВЕТ порядку их объявления.
Например, если в како-либо месте программы сделано такое объявление:

```
char Ch;
short Sh;
int In;
```

то эти три переменные будут расположены одна за другой в порядке объявления и займут 7 байтов. Фактически три переменные можно рассматривать как неименованную структуру с выравниванием 1.

2.4. КВАЛИФИКАТОР CONST

Переменная, к которой в объявлении (декларации) применен квалификатор ***const***, не может изменять свое значение. Ее можно только инициализировать, то есть присвоить ей значение в начале выполнения программы. Компилятор помещает переменную этого типа в постоянное запоминающее устройство (ROM, read-only memory или FLASH память). Например, в объявлении

```
const int a=10;
```

создается переменная с именем ***a***, причем ей присваивается начальное значение 10, которое в дальнейшем в программе изменить никак нельзя. Переменную, к которой в объявлении применен квалификатор ***const***, можно использовать в различных выражениях. Однако свое значение она может получить только в результате инициализации.
Квалификатор ***const*** часто используется для того, чтобы предотвратить изменение функцией объекта, на который указывает аргумент функции. Без него при передаче в функцию указателя эта функция может изменить объект, на который он указывает. Однако если в объявлении параметра-указателя применен квалификатор ***const***, функция не сможет изменить этот объект. В следующем примере функция ***sp_to_dash()*** печатает минус вместо каждого пробела в строке, передаваемой ей как аргумент. То есть строка "тестовый пример" будет напечатана как "тестовый-пример". Применение квалификатора ***const*** в объявлении параметра функции гарантирует, что внутри функций объект, на который указывает параметр функции, не будет изменен.

```
#include <sat500.h>

void sp_to_dash(const char * str);
void main(void)
{
    ClrScr();
    sp_to_dash("тестовый пример");
    ShowScr();
}

void sp_to_dash(const char *str)
{
    while(*str)
    {
        if(*str== ' ') printf("%c", '-');
        else printf("%c", *str);
        str++;
```

```
    }
}
```

Квалификатор ***const*** используется в объявлениях параметров многих функций стандартной библиотеки. Если функция стандартной библиотеки не предназначена для изменения аргумента, то практически всегда в объявлении указателя на аргумент применяется квалификатор ***const***. Программист тоже может применять квалификатор ***const*** для того, чтобы гарантировать сохранность объекта.

2.5. ИНИЦИАЛИЗАЦИЯ ПЕРЕМЕННЫХ

При объявлении переменной она может быть инициализирована. Для этого нужно после ее объявления поставить знак равенства и константу, т.е. общая форма инициализации имеет следующий вид:

```
тип имя_переменной = константа;
```

Несколько примеров инициализации переменных:

```
char ch = 'a';
int first = 0;
char * ptr=&a;
char Msg[]="Press a key...";
char * Texts[]=
{
    "Текст1",
    "Текст2",
    "Текст3",
    ""
};
unsigned int Massiv[3]={2018,2019,2020};
unsigned short us[]={1,2,4,8};
```

Глобальные переменные инициализируются только один раз в начале работы программы. А локальные переменные инициализируются каждый раз при входе в блок, в котором они объявлены. Неинициализированные локальные переменные до первого присвоения имеют произвольное значение.
Неинициализированные глобальные переменные в начале работы программы автоматически обнуляются.

2.6. КОНСТАНТЫ

Константа — это фиксированное значение, которое не может быть изменено программой. Константа может относиться к любому базовому типу. Способ представления константы определяется ее типом. Константы также называются литералами.
Символьные константы заключаются в одинарные кавычки. Например, ***'a'*** и ***'%'*** — это символьные константы.
Целые константы определяются как числа без дробной части. Например, ***10*** и ***-100*** — это целые константы.
Для целых типов суффикс ***U*** означает unsigned, а ***L*** - long (int). Тип суффикса не зависит от регистра.
Несколько примеров:

Тип данных	*Примеры констант*			
int	1	123	21000	-243
long int	35000L	-34L		
unsigned int	10000U	987u	40000U	

Шестнадцатиричные константы

Иногда удобнее использовать не десятичную, а шестнадцатеричную систему.
Позиционная система счисления с основанием 16 называется ***шестнадцатеричной***. В ней используются 16 символов: цифры от 0 до 9 и символы от A до F, обозначающие цифры от 10 до 15. Например, запись 10 в шестнадцатеричной системе обозначает то же число, что и 16 в десятичной системе. Эти системы счисления используются довольно часто, поэтому в C целые константы можно определять не только в десятичной, но и в шестнадцатеричной системе счисления. Шестнадцатеричная константа начинается с 0x, например:

```
int hex = 0x80;      /* 128 в десятичной системе */
int val = 0xFFFF;    /* 65535 в десятичной системе */
```

Строковые константы

Язык C поддерживает ещё один тип констант, а именно - строковые. Строка - это последовательность символов, заключённых в двойные кавычки. Например, "тест" - это строка. В термине "строковая константа" слово "строковая" не означает строковый предопределённый тип данных, такого в C нет, здесь это всего лишь прилагательное.
Не следует путать понятия строки и символа. Символьная константа заключается в одинарные кавычки, например, ***'a'***. Соответственно запись ***"a"*** означает строку, состоящую из одного символа.

Специальные символьные константы

Чтобы представить большинство символьных констант, достаточно заключить соответствующий символ в одинарные кавычки. Но некоторые символы, например, символ возврата каретки, требуют специального представления. В языке C определены специальные символьные константы, приведённые в таблице ниже. Иногда их называют ESC-последовательностями, управляющими последовательностями и символами с обратным слэшем. Управляющие последовательности можно использовать вместо ASCII-кодов для обеспечения лучшей переносимости программы.
В следующем примере программа выводит символ новой строки (т.е. переходит на новую строку), выводит символ табуляции (т.е. переходит на первую позицию табуляции) и, наконец, выводит строку Простой тест.

```
#include <sat500.h>

void main(void)
{
    ClrScr();
    printf("\n\tПростой тест.");
    ShowScr();
    while(1);
}
```

Таблица 2 - Специальные символы константы

Код	***Назначение***
\b	Удаление предыдущего символа
\f	Подача бумаги
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция

\"	Двойные кавычки
\'	Одинарная кавычка
\\	Обратный слэш
\v	Вертикальная табуляция
\a	Сигнал
\xN	Шестнадцатеричная константа (N - шестнадцатеричное представление)

2.7. ОПЕРАЦИИ

Язык С содержит большое количество встроенных операций. Их роль в С значительно больше, чем в других языках программирования. Существует четыре основных класса операций: *арифметические, логические, поразрядные и операции сравнения*. Кроме них, есть также некоторые специальные операторы, например, *оператор присваивания*.

Оператор присваивания

Оператор присваивания может присутствовать в любом выражении языка С. Этим С отличается от большинства других языков программирования (Pascal, BASIC и FORTRAN), в которых присваивание возможно только в отдельном операторе. Общая форма оператора присваивания:

```
имя_переменной=выражение;
```

Выражение может быть просто константой или сколь угодно сложным выражением. В отличие от Pascal или Modula-2, в которых для присваивания используется знак ***":="***, в языке С оператором присваивания служит единственный знак присваивания ***"="***. Адресатом (получателем), т.е. левой частью оператора присваивания должен быть объект, способный получить значение, например, переменная.

Преобразование типов при присваиваниях

Если в операции встречаются переменные разных типов, происходит *преобразование типов*. В операторе присваивания действует простое правило: значение выражения в правой части преобразуется к типу объекта в левой части.

```
char ch;
short int si;
int i;
unsigned int ui;
void func(void)
{
    ch = i;    /* 1-я строка */
    si = i;    /* 2-я строка */
    ui = ch;   /* 3-я строка */
    i = ch;    /* 4-я строка */
}
```

В 1-й строке этого примера старшие двоичные разряды целой переменной ***i*** отбрасываются, а в ***ch*** заносятся младшие 8 бит. Если значение i лежит в интервале от -128 до 128, то ***ch*** и ***i*** будут идентичны и потери информации не произойдет. В противном случае в ***ch*** будут занесены только младшие разряды переменной ***i***.
Во 2-й строке в ***si*** будет записаны два младших байта переменной ***i***. Если значение ***i*** лежит в интервале от -32768 до 32767, то ***si*** и ***i*** будут идентичны и потери информации не произойдет. В противном случае в **si** будут занесены только два младших байта переменной ***i***.
В 3-й строке беззнаковой переменной ***ui*** будет присвоено значение ***ch*** - значение **ch** будет помещено в младший байт переменной ***ui***, а старшие три байта будут обнулены.

В 4-й строке знаковой переменной ***i*** будет присвоено значение ***ch*** - значение **ch**. Старшие три байта ***i*** будут расширены знаковым битом переменной ***ch***.

В таблице 3 приведены варианты потери информации при некоторых преобразованиях. Некоторые компиляторы при преобразовании char в int считают переменную char положительной независимо от ее значения. Другие компиляторы считают переменную char отрицательной, если она больше 127. Поэтому для обеспечения переносимости программы необходимо использовать переменные типа char для хранения символов, а переменные типа signed char и int (целый) — для хранения чисел.

Таблица 3 - Результат некоторых преобразований типов

Тип адресата	*Тип выражения*	*Потеря информации*
signed char	char	Если значение > 127, то результат отрицательный
char	short int	Старшие 6 бит
char	int (32-разрядный)	Старшие 24 бит
short int	int (32-разрядный)	Старшие 16 бит

Множественные присваивания

В одном операторе присваивания можно присвоить одно и то же значение многим переменным. Для этого используется оператор *множественного присваивания*, например:

```
x = y = z = 0;
```

Следует отметить, что в практике программирования этот прием используется очень часто. *Множественное присваивание* - присваивание одного и того же значения нескольким переменным. Под множественным присваиванием также подразумевается конструкция языка программирования, позволяющая присвоить одно и то же значение нескольким переменным одновременно.

Составное присваивание

Составное присваивание - это разновидность оператора присваивания, в которой запись сокращается и становится более удобной в написании. Например, оператор

```
x = x+10;
```

можно записать как

```
x += 10;
```

Оператор "+=" сообщает компилятору, что к переменной x нужно прибавить 10.
"Составные" операторы присваивания существуют для всех бинарных операций (то есть операций, имеющих два операнда). Любой оператор вида

```
переменная = переменная оператор выражение;
```

можно записать как

```
переменная оператор = выражение;
```

Еще один пример:

```
x = x-100;
```

означает то же самое, что и

```
x -= 100;
```

Составное присваивание значительно компактнее, чем соответствующее простое присваивание, поэтому его иногда называют *стенографическим* (*shorthand*) присваиванием. В программах на С этот оператор широко используется, поэтому необходимо хорошо его усвоить.
Варианты оператора присваивания, в которых используется такая запись, называются "сокращенными" или "укороченными". Что касается терминологии, то необходимо отметить также следующее обстоятельство. Хотя термины присваивание и оператор присваивания часто могут рассматриваться как синонимы, составное присваивание не является составным оператором! (Под составным оператором в языке С подразумевают блок.)

Арифметические операции

В таблице 4 приведены арифметические операции С. Операции +, -, * и / работают так же, как и в большинстве других языков программирования. Их можно применять почти ко всем встроенным типам данных. Если операция / применяется к целому или символьному типам, то остаток от деления отбрасывается. Например, результатом операции 5/2 является 2.

Таблица 4 - Арифметические операции

Оператор	***Операция***
-	Вычитание, так же унарный минус
+	Сложение
*	Умножение
/	Деление
%	Остаток от деления
--	Декремент, или уменьшение
++	Инкремент, или увеличение

Оператор деления по модулю % в С работает так же, как и в других языках, его результатом является остаток от целочисленного деления. Применение оператора % иллюстрируется следующим примером:

```
int x, y;

x = 5;
y = 2;

printf("%d ", x/y);   /* напечатает 2 */
printf("%d ", x%y);   /* напечатает 1, остаток от целочисленного деления */

x = 1;
y = 2;

printf("%d %d", x/y, x%y); /*  напечатает 0 1 */
```

Последняя строка программы напечатает 0 1 потому, что при целочисленном делении остаток отбрасывается и здесь результат будет 0, а сам остаток равен 1.
Унарный минус умножает операнд на -1, то есть меняет его знак на противоположный.

Операции увеличения (инкремента) и уменьшения (декремента)

В языке С есть два полезных оператора, значительно упрощающие широко распространенные операции. Это инкремент **++** и декремент **--**. Оператор **++** увеличивает значение операнда на 1, а **--** уменьшает на 1. Иными словами,

```
x = x+1;
```

можно записать как

```
++x;
```

Аналогично оператор

```
x = x-1;
```

равносилен оператору

```
x--;
```

Как инкремент, так и декремент могут предшествовать операнду (префиксная форма) или следовать за ним (постфиксная форма). Например

```
x = x+1;
```

можно записать как в виде

```
++x;
```

так и в виде

```
x++;
```

Однако префиксная и постфиксная формы отличаются при использовании их в выражениях. Если оператор инкремента или декремента предшествует операнду, то сама операция выполняется до использования результата в выражении. Если же оператор следует за операндом, то в выражении значение операнда используется до выполнения операции инкремента или декремента. То есть для выражения эта операция как бы не существует, она выполняется только для операнда. Например,

```
x = 10;
y = ++x;
```

присваивает y значение 11. Однако если написать

```
x = 10;
y = x++;
```

то переменной y будет присвоено значение 10. В обоих случаях x присвоено значение 11, разница только в том, когда именно это случилось, до или после присваивания значения переменной y. Компилятор C генерируют для инкремента и декремента очень быстрый, эффективный объектный код, значительно лучший, чем для соответствующих операторов присваивания. Поэтому везде, где это возможно, рекомендуется использовать инкремент и декремент.
Приоритет выполнения арифметических операторов следующий:

Наивысший	++ --
	- (унарный минус)
	* / %
Наинизший	+ -

Операции с одинаковым приоритетом выполняются слева направо. Используя круглые скобки, можно изменить порядок вычислений. В языке C круглые скобки интерпретируются компилятором так же, как и в любом другом языке программирования: они как бы придают операции (или последовательности операций) наивысший приоритет.
На жаргоне программистов -- называют декремент, а ++ называют инкремент.

Операции сравнения и логические операции

Операции *сравнения* — это операции, в которых значения двух переменных сравниваются друг с другом. Логические же операции реализуют средствами языка С операции формальной логики. Между логическими операциями и операциями сравнения существует тесная связь: результаты операций сравнения часто являются операндами логических операций.
В операциях сравнения и логических операциях в качестве операндов и результатов операций используются значения ИСТИНА (true) и ЛОЖЬ (false). В языке С значение ИСТИНА представляется любым числом, отличным от нуля. Значение ЛОЖЬ представляется нулем. Результатом операции сравнения или логической операции являются ИСТИНА (true, 1) или ЛОЖЬ (false, 0).
Значение ИСТИНА представлено любым отличным от нуля числом, а ЛОЖЬ - нулем.
В таблице 5 приведён полный список операций сравнения и логических операций.

Таблица 5 - Истинность логических операций

p	***q***	***p && q***	***p || q***	***!p***
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Как операции сравнения, так и логические операции имеют низший приоритет по сравнению с арифметическими. То есть, выражение 10>1+12 интерпретируется как 10>(1+12). Результат, конечно, равен ЛОЖЬ.
В одном выражении можно использовать несколько операций:

```
10>5 && !(10<9) || 3<4
```

В этом случае результатом будет ИСТИНА.
В языке С не определена операция "исключающего ИЛИ" (exclusive OR, или XOR). Однако с помощью логических операторов несложно написать функцию, выполняющую эту операцию. Результатом операции "исключающее ИЛИ" является ИСТИНА, если и только если один из операндов (но не оба) имеют значение ИСТИНА. В следующем примере функция ***xor()*** возвращает результат операции "исключающее ИЛИ", а операндами служат аргументы функции:

```
#include <sat500.h>

int xor(int a, int b);

void main(void)
{
    ClrScr();
    printf("%d", xor(1, 0));
    printf("%d", xor(1, 1));
    printf("%d", xor(0, 1));
    printf("%d", xor(0, 0));
    ShowScr();
    while(1);
}

/* Выполнение логической операции исключающее ИЛИ над двумя аргументами. */
int xor(int a, int b)
{
  return (a || b) && !(a && b);
}
```

Таблица 6 - Операции сравнения и логические операции

Операторы сравнения	
Оператор	***Операция***
>	Больше чем
>=	Больше или равно
<	Меньше чем
<=	Меньше или равно
==	Равно
!=	Не равно
Логические операции	
Оператор	**Операция**
&&	И
||	ИЛИ
!	НЕ, отрицание

Ниже приведен приоритет логических операций:

Наивысший	!
	> >= < <=
	== !=
	&&
Наинизший	||

Как и в арифметических выражениях, для изменения порядка выполнения операций сравнения и логических операций можно использовать круглые скобки. Например, выражение:

```
!0 && 0 || 0
```

равно ЛОЖЬ. Однако, если добавить скобки как показано ниже, то результатом будет ИСТИНА:

```
!(0 && 0) || 0
```

Необходимо помнить, что результатом любой операции сравнения или логической операции есть 0 или 1. Поэтому следующий фрагмент программы является правильным и в результате его выполнения будет напечатано 1.

```
int x;

x = 100;
printf("%d", x>10);
```

Поразрядные операции

В отличие от многих других языков программирования, в С определен полный набор поразрядных операций (Называются также битовыми, побитовыми и логическими операциями). Это обусловлено тем, что С был задуман как язык, призванный во многих приложениях заменить ассемблер, который способен оперировать битами данных. Поразрядные операции - это тестирование (проверка), сдвиг или присвоение значений отдельным битам данных.

В таблице 7 приведён полный список знаков поразрядных операций, выполняемых над отдельными разрядами (битами) операндов.

Таблица 7 - Поразрядные операции

Оператор	***Операция***
&	И
|	ИЛИ

^	исключающее ИЛИ
~	НЕ (отрицание, дополнение к 1)
>>	Сдвиг вправо
<<	Сдвиг влево

Таблицы истинности логических операций и поразрядных операций И, ИЛИ, НЕ совпадают. Отличие лишь в том, что поразрядные операции выполняются над отдельными разрядами (битами) операндов. Операция "исключающее ИЛИ" имеет следующую таблицу истинности:

p	***q***	***p ^ q***
0	0	0
1	0	1
1	1	0
0	1	1

Как показано в таблице, результат операции "исключающее ИЛИ" равен ИСТИНА если и только если один из операндов равен 1, иначе результат будет равен ЛОЖЬ.
Поразрядные операции используются для маскирования определенных битов, например, бита контроля четности. (Этот бит служит для проверки правильности остальных битов в байте. Чаще всего это бит старшего разряда в каждом байте.)
Операция И может быть использована для очищения бита. Иными словами, для гашения бита используется следующее свойство операции И: если бит одного из операндов равен 0, то соответствующий бит результата будет равен 0 независимо от состояния этого бита во втором операнде. Например, следующая функция обнуляет бит контроля четности (старший бит):

```
char get_char(char ch)
{
    return(ch & 127);
}
```

Бит контроля четности, находящийся в 8-м разряде байта, обнуляется с помощью операции ***И***. При этом в качестве второго операнда выбирается число, имеющее 1 в разрядах от 1 до 7, и 0 в 8-м разряде. Именно таким числом и является 127, поскольку все биты двоичного представления числа 127, кроме старшего, равны 1. В силу указанного свойства операции ***И*** операция ***ch & 127*** оставляет все биты, кроме старшего, без изменения, а старший обнуляет:
Бит контроля четности

```
  |
  V
  1100 0001     переменная ch содержит символ 'A' с битом четности
  0111 1111     двоичное представление числа 127
& ---------     поразрядная операция И
  0100 0001     символ 'A' с обнуленным битом контроля четности
```

Поразрядная операция ***ИЛИ***, являющаяся двойственной операции ***И***, применяется для установки необходимых битов в 1. В следующем примере выполняется операция 128 | 3:

```
  |
  V
  1000 0000     двоичное представление числа 128
  0000 0011     двоичное представление числа 3
| ---------     поразрядная операция ИЛИ
  1000 0011     результат
```

Операция исключающего ***ИЛИ (XOR)*** устанавливает бит результата в 1, если соответствующие биты операндов различны. В следующем примере выполняется операция ***127 ^ 120***:

```
  |
  V
  0111 1111     двоичное представление числа 127
  0111 1000     двоичное представление числа 120
^ ---------     поразрядная операция XOR
  0000 0111     результат
```

Необходимо помнить, что результат логической операции всегда равен 0 или 1. В то же время результатом поразрядной операции может быть любое значение, которое, как видно из предыдущих примеров, не обязательно равно 0 или 1.
Поразрядные операторы сдвига >> и << сдвигают все биты переменной вправо или влево. Общая форма оператора сдвига вправо:

переменная >> количество_разрядов

Общая форма оператора сдвига влево:

переменная << количество_разрядов

Во время сдвига битов в один конец числа, другой конец заполняется нулями. Но если число типа int отрицательно, то при сдвиге вправо левый конец заполняется единицами, так что знак числа сохраняется. Необходимо отметить различие между сдвигом и циклическим сдвигом. При циклическом сдвиге биты, сдвигаемые за пределы операнда, появляются на другом конце операнда. А при сдвиге вышедшие за границу биты теряются.
Побитовые операторы сдвига могут быстро умножать и делить целые числа. Как показано в табл. 2.7, сдвиг на один бит вправо делит число на 2, а на один бит влево — умножает на 2. Следующая программа иллюстрирует применение операторов сдвига:

```
/* Пример применения операторов сдвига. */
#include <sat500.h>
void main(void)
{
    unsigned int i;
    int j;
    ClrScr();
    i = 1;
    /* сдвиг влево */
    for(j=0; j<4; j++)
    {
        i = i << 1;  /* сдвиг i влево на 1 разраяд, что равносильно умножению
на 2 */
        printf("Сдвиг влево на %d разр.: %d\n", j, i);
    }
    /* сдвиг вправо */
    for(j=0; j<4; j++)
    {
        i = i >> 1;  /* сдвиг i вправо на 1 разраяд, что равносильно делению
на 2 */
        printf("Сдвиг вправо на %d разр.: %d\n", j, i);
    }
    ShowScr();
    while(1);
}
```

Таблица 8 - Умножение и деление операторами сдвига

unsigned char x	x после операции	значение x
x = 7	0000 0111	7
x = x << 1	0000 1110	14

x = x << 3	0111 0000	112
x = x << 2	1100 0000	192
x = x >> 1	0110 0000	96
x = x >> 2	0001 1000	24

Каждый сдвиг влево умножает на 2. Потеря информации произошла после операции x << 2 в результате сдвига за левую границу.
Каждый сдвиг вправо делит на 2. Сдвиг вправо потерянную информацию не восстановил

Поразрядная операция отрицания (дополнения) **~** инвертирует состояние каждого бита операнда. То есть, 0 преобразует в 1, а 1 - в 0.
Поразрядные операции часто используются в процедурах кодирования. Проделав с массивом байтов некоторые поразрядные операции, его можно сделать нечитаемым. Простейший способ сделать это - применить операцию отрицания к каждому биту:
Исходный байт 0010100
После 1-го отрицания 1101011
После 2-го отрицания 0010100
Обратите внимание, при последовательном применении 2-х отрицаний результатом всегда будет исходное число. Таким образом, 1-е отрицание кодирует состояние байта, а 2-е — декодирует.
В следующем примере оператор отрицания используется в функции шифрования символа:

```
/* Простейшая процедура шифрования. */
char encode(char ch)
{
  return(~ch); /* оперция отрицания */
}

Конечно, взломать такой шифр не представляет труда.
```

Операция ?

В языке С определен мощный и удобный оператор, который часто можно использовать вместо оператора вида ***if-then-else***. Речь идет о тернарном операторе ***?***, общий вид которого следующий:

```
Выражение1 ? Выражение2 : Выражение3;
```

Обратите внимание на использование двоеточия. Оператор ***?*** работает следующим образом: сначала вычисляется Выражение1, если оно истинно, то вычисляется Выражение2 и его значение присваивается всему выражению; если Выражение1 ложно, то вычисляется Выражение3 и всему выражению присваивается его значение. В примере

```
x = 10;
y = x>9 ? 100 : 200;
```

переменной y будет присвоено значение 100. Если бы x было меньше 9, то переменной y было бы присвоено значение 200. Эту же процедуру можно написать, используя оператор if-else:

```
x = 10;
if(x>9) y = 100;
else y = 200;
```

Операция получения адреса (&) и раскрытия ссылки (*)

Указатель — это адрес объекта в памяти. Переменная типа "*указатель*" (или просто *переменная-указатель*) — это специально объявленная переменная, в которой хранится указатель на переменную определённого типа. В языке С указатели служат мощнейшим средством создания программ и широко используются для самых разных целей. Например, с их помощью можно быстро обратиться к элементам массива или дать функции возможность модифицировать свои аргументы.

Указатели широко используются для связи элементов в списках, в двоичных деревьях и в других динамических структурах данных. В данном разделе коротко рассматриваются два оператора, использующиеся для работы с указателями.
Первый из них — оператор ***&***, это унарный оператор, возвращающий адрес операнда в памяти. (Унарной операцией называется операция, имеющая только один операнд.) Например, оператор

```
m = &count;
```

записывает в переменную ***m*** адрес переменной ***count***. Этот адрес представляет собой адрес ячейки памяти компьютера, в которой размещена переменная. Адрес и значение переменной — совершенно разные понятия. Выражение "*&переменная*" означает "*адрес переменной*". Следовательно, инструкция ***m = &scount;*** означает: "Переменной m присвоить адрес, по которому расположена переменная count;".
Допустим, переменная ***count*** расположена в памяти в ячейке с адресом 2000, а её значение равно 100. Тогда в предыдущем примере переменной m будет присвоено значение 2000.
Второй рассматриваемый оператор ***** является двойственным (дополняющим) по отношению к &. Оператор * является унарным оператором, он возвращает значение объекта, расположенного по указанному адресу. Операндом для ***** служит адрес объекта (переменной). Например, если переменная m содержит адрес переменной count, то оператор

```
q = *m;
```

записывает значение переменной ***count*** в переменную ***q***. В нашем примере переменная ***q*** получит значение 100, потому что по адресу 2000 записано число 100, причём этот адрес записан в переменной ***m***. Выражение "* адрес" означает "по адресу". Наш фрагмент программы можно прочесть как "*q получает значение, расположенное по адресу m*".
К сожалению, символ операции раскрытия ссылки совпадает с символом операции умножения, а символ операции получения адреса - с символом операции поразрядного ***И***. Необходимо помнить, что эти операторы не имеют никакого отношения друг к другу. Операторы ***** и ***&*** имеют более высокий приоритет, чем любая арифметическая операция, кроме унарного минуса, имеющего такой же приоритет.
Если переменная является указателем, то в объявлении перед её именем нужно поставить символ *****, он сообщит компилятору о том, что это указатель на переменную данного типа. Например, объявление указателя на переменную типа char записывается так:

```
char *ch;
```

Необходимо понимать, что ***ch*** - это не переменная типа ***char***, а указатель на переменную данного типа, это совершенно разные вещи. Тип данных, на который указывает указатель (в данном случае это ***char***), называется базовым типом указателя. Сам указатель является переменной, содержащей адрес объекта базового типа. Компилятор учтёт размер указателя в архитектуре компьютера и выделит для него необходимое количество байтов, чтобы в указатель поместился адрес. Базовый тип указателя определяет тип объекта, хранящегося по этому адресу.
В одном операторе объявления можно одновременно объявить и указатель, и переменную, не являющуюся указателем. Например, оператор

```
int x, *y, count;
```

объявляет ***x*** и ***count*** как переменные целого типа, а ***y*** - как указатель на переменную целого типа.
В следующей программе операторы ***** и ***&*** используются для записи значения 10 в переменную ***target***. Программа выведет значение 10 на экран.

```
#include <sat500.h>

void main(void)
{
    int target, source;
```

```
    int * m;

    source = 10;
    m = &source;
    target = *m;
    ClrScr();
    printf("%d", target);
    ShowScr();
    while(1);
}
```

Оператор **&** называется также оператором получения (взятия) адреса.
Оператор ***** называется также оператором косвенности, оператором раскрытия ссылки и оператором разыменования адреса.

Операция определения размера sizeof

Унарная операция ***sizeof***, выполняемая во время компиляции программы, позволяет определить длину операнда в байтах. Например, компилятор для чисел типа int отводит 4 байта, а для чисел типа short int 2, то следующая программа напечатает 4 2.

```
short int si;

printf("%d", sizeof(int));
printf("%d ", sizeof(si));
```

Необходимо помнить, что для вычисления размера типа переменной имя типа должно быть заключено в круглые скобки. Имя переменной заключать в скобки не обязательно, но ошибки в этом не будет.
В языке С определяется (с помощью спецификатора класса памяти ***typedef***) специальный тип ***size_t***, соответствующий целому числу без знака. Результат операции ***sizeof*** имеет тип ***size_t***. Но практически его можно использовать везде, где допустимо использование целого числа без знака.
Оператор ***sizeof*** очень полезен для улучшения переносимости программ, так как переносимость существенно зависит от размеров встроенных типов данных. Для примера рассмотрим программу, работающую с базой данных, в которой необходимо хранить шесть целых чисел в одной записи. Если эта программа предназначена для работы на многих компьютерах, ни в коем случае нельзя полагаться на то, что размер целого числа на всех компьютерах будет один и тот же.
И в заключение: оператор ***sizeof*** выполняется во время трансляции, его результат в программе рассматривается как константа.

Оператор [ ] и ()

Круглые скобки являются оператором, повышающим приоритет выполнения операций, которые в них заключены.
Квадратные скобки служат для индексации массива (массивы подробно рассматриваются в отдельном разделе). Если в программе определён массив, то выражение в квадратных скобках представляет собой индекс массива. Например, в программе

```
#include <sat500.h>

char s[80];

void main(void)
{
    ClrScr();
    s[3] = 'X';
    printf("%c", s[3]);
    ShowScr();
    while(1);
```

```
}
```

значение 'X' сначала присваивается четвёртому элементу массива (в С элементы массива нумеруются с нуля), затем этот элемент выводится на экран.

Сводка приоритетов операций

В таблице 9 приведены приоритеты всех операций, определённых в С. Необходимо помнить, что все операторы, кроме унарных и "?", связывают (присоединяют, ассоциируют) свои операнды слева направо. Унарные операторы (*, &, -) и "?" связывают (присоединяют, ассоциируют) свои операнды справа налево.

Таблица 9 - Приоритеты операций в языке С

Наивысший	( ) [ ] -> . ! ~ ++ -- - (type) * & sizeof
	* / %
	+ -
	<< >>
	< <= > >=
	== !=
	&
	^
	|
	&&
	||
	?:
	= += -= *= /= и т.д.
Наинизший	,

2.8. ВЫРАЖЕНИЯ

Выражения состоят из операторов, констант, функций и переменных. В языке С выражением является любая правильная последовательность этих элементов. Большинство выражений в языке С по форме очень похожи на алгебраические, часто их и пишут, руководствуясь правилами алгебры. Однако здесь необходимо быть внимательным и учитывать специфику выражений в языке С.

Порядок вычислений

Порядок вычисления подвыражений в выражениях языка С не определен. Компилятор может самостоятельно перестроить выражение с целью создания оптимального объектного кода. Это значит, что программист не может полагаться на определенную последовательность вычисления подвыражений. Например, при вычислении выражения

```
x = f1() + f2();
```

нет никаких гарантий того, что функция ***f1()*** будет вызвана перед вызовом ***f2()***.

Преобразования типов в выражениях

Если в выражении встречаются переменные и константы разных типов, они преобразуются к одному типу. Компилятор преобразует "меньший" тип в "больший".
Этот процесс называется *продвижением типов (type promotion)*. Все переменные типов ***char*** и ***short int*** автоматически продвигаются в ***int***. Это называется целочисленным расширением. Целочисленное расширение может также завершиться преобразованием в *unsigned int*.

Кроме того, действует следующее правило: если один из операндов имеет тип ***int***, а второй - ***unsigned int***, то оба операнда преобразуются в ***unsigned long***.
После выполнения приведенных выше преобразований оба операнда относятся к одному и тому же типу, к этому типу относится и результат операции.

Явное преобразование типов: операция приведения типов

Программист может "принудительно" преобразовать значение выражения к нужному ему типу, используя операцию приведения типов. Общая форма оператора явного приведения типа:

```
(тип) выражение
```

Здесь *тип* — это любой поддерживаемый тип данных. Например, следующая запись преобразует значение выражения ***x/2*** к типу ***char***:

```
(char)x/2
```

Явное преобразование типа — это операция. Оператор приведения типа является унарным и имеет тот же приоритет, что и остальные унарные операторы.
Иногда приведение типов может быть весьма полезным.

Пробелы и круглые скобки

Для повышения удобочитаемости программы при записи выражений можно использовать пробелы и символы табуляции. Например, следующие два оператора эквивалентны:

```
x=10/y~(127/x);

x = 10 / y ~(127/x);
```

Лишние скобки, если они не изменяют приоритет операций, не приводят к ошибке и не замедляют вычисление выражения. Дополнительные скобки часто используют для прояснения порядка вычислений. В следующем примере 2-я строка читается значительно легче:

```
x = y/3-34*temp+127;

x = (y/3) - (34*temp) + 127;
```

2.9. СТРУКТУРА ПРОГРАММЫ

Любая программа на языке С состоит из одной и более функций, одна из которых должна иметь имя main, и именно ей передается управление при старте. Функция – это коллективное имя для некоторой группы описаний и операторов, заключенных в фигурные скобки { } и являющихся телом функции. В общем виде программа на языке С имеет следующую структуру:

```
#<Директивы препроцессора>

<прототипы используемых функций>

void main(void)
{
      <тело функции main>
}
function1( )
{
      <тело функции function1>
}
```

```
function2( )
{
      <тело функции function2>
}
…
```

2.10. КОММЕНТАРИИ

В стандарте С комментарий начинается с символов /* и заканчивается символами */. Между звездочкой и слешем не должно быть никаких пробелов. Любой текст, расположенный между начальными и конечными символами комментария, компилятором игнорируется. Например, следующая программа только выведет на экран *Привет*:

```
#include <sat500.h>

void main(void)
{
    ClrScr();
    printf("Привет");
    /* printf("всем"); */
    ShowScr();
    while(1);
}
```

Комментарий такого вида называется *многострочным комментарием* (multiline comment), потому что его текст может располагаться в нескольких строках. Например,

```
/* это
   многострочный
   комментарий */
```

Комментарии могут находиться в любом месте программы, за исключением середины ключевого слова или идентификатора. Приведенный ниже комментарий правильный:

```
x = 10+ /* прибавлять числа */5;
```

а комментарий

```
swi/* такое работать не будет */tch(c) { ...
```

не является допустимым, потому что комментарий не может разрывать ключевое слово. Впрочем, комментарии обычно не следует размещать и в середине выражений, потому что так труднее разобраться и с выражениями, и с самими комментариями.
Многострочные комментарии не могут быть вложенными. То есть в одном комментарии не может находиться другой. Например, при компиляции следующего фрагмента кода будет обнаружена ошибка:

```
/* Это внешний комментарий
  x = y/a;
  /* а это внутренний комментарий, обнаружив который,
     компилятор выдаст сообщение об ошибке */
*/
```

Однострочные комментарии

Такой комментарий начинается с символов // и заканчивается в конце строки. Например:

```
// это однострочный комментарий
```

Однострочные комментарии особенно полезны тогда, когда нужны краткие, не более чем в одну строку пояснения.
Однострочный комментарий может находиться внутри многострочного комментария. Например, следующий комментарий является вполне допустимым:

```
/* это // проверка вложенных комментариев */
```

Комментарии должны находиться там, где требуется объяснить работу кода. Например, в начале всех функций, за исключением самых очевидных, должен быть комментарий, который сообщает, что именно делает функция, как она вызывается и что возвращает.

2.11. ОПЕРАТОРЫ

Оператор - это часть программы, которая может быть выполнена отдельно. Это означает, что оператор определяет некоторое действие. В языке С существуют следующие группы операторов:

- Условные операторы
- Операторы цикла
- Операторы безусловного перехода
- Метки
- Операторы-выражения
- Блоки

К условным относятся операторы ***if*** и ***switch***. Иногда их также называют операторами условного перехода. Операторы цикла - это ***while***, ***for*** и ***do-while***. К операторам безусловного перехода относятся ***break***, ***continue***, ***goto*** и ***return***. К меткам относятся операторы ***case***, ***default*** (рассматриваются в разделе "Оператор выбора - ***switch***") и собственно метки (рассматриваются в разделе "Оператор ***goto***"). Операторы-выражения — это операторы, состоящие из допустимых выражений. Блок представляет собой фрагмент текста программы, обрамленный фигурными скобками {}. Блок иногда называют составным оператором.

2.11.1. ЛОГИЧЕСКИЕ ЗНАЧЕНИИЯ ИСТИНА (TRUE) И ЛОЖЬ (FALSE) В ЯЗЫКЕ С

Так как во многих операторах применяются условные выражения, их рассмотрение начнём с краткой характеристики значений **ИСТИНА** и **ЛОЖЬ**.
При выполнении многих операторов языка С вычисляются значения условных выражений и в зависимости от полученного значения выбирается та или иная ветвь вычислительного процесса. Условное выражение может принимать одно из двух значений: **ИСТИНА** или **ЛОЖЬ**.
В языке С значение **ИСТИНА** представлено любым ненулевым значением, включая отрицательные числа.
Значение **ЛОЖЬ** всегда представлено нулём. Такое представление логических значений **ИСТИНА** и **ЛОЖЬ** позволяет весьма эффективно программировать многие процедуры.
В заголовочном файле *<sat500.h>* определены константы true и false следующим образом:

```
#define true 1
#define false 0
```

2.11.2. УСЛОВНЫЕ ОПЕРАТОРЫ

В языке С существуют два условных оператора: if и switch. При определенных обстоятельствах оператор ? является альтернативой оператора if.

Оператор if

Общая форма оператора ***if*** следующая:

```
if (выражение) оператор;
else оператор;
```

Здесь оператор может быть только одним оператором, блоком операторов или отсутствовать (пустой оператор). Фраза ***else*** может вообще отсутствовать.
Если выражение истинно (т.е. принимает любое значение, отличное от нуля), то выполняется оператор или блок операторов, следующий за ***if***. В противном случае выполняется оператор (или блок операторов), следующий за ***else*** (если эта фраза присутствует). Необходимо помнить, что выполняется или оператор, связанный с ***if***, или с ***else***, но оба - никогда!
Условное выражение, входящее в ***if***, должно иметь скалярный результат. Это значит, что результатом должно быть целое число, символ или указатель , но им не может быть массив.
У начинающих программистов иногда возникают трудности в связи с тем, что в условном (управляющем) выражении операторов ***if*** или ***?*** могут стоять любые операторы, причем это не обязательно операторы отношения или логические (как в языках Basic или Pascal). В языке С значением результата управляющего выражения являются ИСТИНА или ЛОЖЬ, однако тип результата может быть любым скалярным типом. Считается, что любой ненулевой результат представляет значение ИСТИНА, а нулевой — ЛОЖЬ.

Вложенные условные операторы if

Оператор ***if*** является вложенным, если он вложен, т.е. находится внутри другого оператора ***if*** или ***else***. В практике программирования вложенные условные операторы используются довольно часто. Во вложенном условном операторе фраза ***else*** всегда ассоциирована с ближайшим ***if*** в том же блоке, если этот ***if*** не ассоциирован с другой фразой ***else***. Например:

```
if(i)
{
      if(j) statement 1;
      if(k) statement 2; /* этот if */
      else  statement 3; /* ассоциирован с этим else */
}
else statement 4; /* ассоциирован с if(i) */
```

Последняя фраза ***else*** не ассоциирована с if(j) потому, что она находится в другом блоке. Эта фраза ***else*** ассоциирована с ***if(i)***. Внутренняя фраза ***else*** ассоциирована с ***if(k)***, потому что этот ***if*** - ближайший.
Уровень вложенности условных операторов может быть любым,однако на практике необходимость в глубине вложенности, большей, чем несколько уровней, возникает довольно редко, так как увеличение глубины вложенности быстро запутывает программу и делает ее нечитаемой.

Лестница if-else-if

В программах часто используется конструкция, которую называют *лестницей* ***if-else-if*** (называется также *структурой выбора* или *конструкцией условного перехода*.).
Общая форма лестницы имеет вид

```
if (выражение) оператор;
else
  if (выражение) оператор;
  else
    if (выражение) оператор;
    .
    .
```

```
    .
    else оператор;
```

Работает эта конструкция следующим образом. Условные выражения операторов ***if*** вычисляются сверху вниз. После выполнения некоторого условия, т.е. когда встретится выражение, принимающее значение ИСТИНА, выполняется ассоциированный с этим выражением оператор, а оставшаяся часть лестницы пропускается. Если все условия ложны, то выполняется оператор в последней фразе ***else***, а если последняя фраза ***else*** отсутствует, то в этом случае не выполняется ни один оператор.
Недостаток предыдущей записи лестницы состоит в том, что с ростом глубины вложенности увеличивается количество отступов в строке. Это становится неудобным с технической точки зрения. Поэтому *лестницу* ***if-else-if*** обычно записывают так:

```
if (выражение)
  оператор;
else if (выражение)
  оператор;
else if (выражение)
  оператор;
    .
    .
    .
else
  оператор;
```

2.11.3. ОПЕРАТОР '?'

Оператор ***?*** можно использовать вместо оператора ***if-else***, записанного в форме

```
if (условие) переменная = выражение;
else переменная = выражение;
```

Оператор ***?*** является тернарным, потому что он имеет три операнда. Его общая форма следующая:

```
Выражение1 ? Выражение2 : Выражение3;
```

Обратите внимание на использование и расположение двоеточия.
Результат операции *?* определяется следующим образом. Сначала вычисляется *Выражение1*. Если оно имеет значение ИСТИНА, вычисляется *Выражение2* и его значение становится результатом операции ***?***. Если *Выражение1* имеет значение ЛОЖЬ, вычисляется *Выражение3* и его значение становится результатом операции ?.
Например:

```
x = 10;
y = x>9 ? 100 : 200;
```

В этом примере переменной ***y*** присваивается значение 100. Если бы ***x*** было меньше 9, то переменная y получила бы значение 200. То же самое можно записать, используя оператор ***if-else***:

```
x = 10;
if(x>9) y = 100;
else y = 200;
```

Оператор ***?*** можно использовать вместо ***if-else*** не только в операторе присваивания. Как известно, все функции (за исключением имеющих тип результата ***void***) возвращают значение. Следовательно, в операторе ***?*** можно использовать вызовы функций. Когда в выражении встречается вызов функции, она выполняется, а возвращаемое ею значение используется при вычислении выражения. Это значит, что можно выполнить одну или несколько функций путем размещения их вызовов в выражениях оператора ***?*** в качестве операндов.

2.11.4. ОПЕРАТОР ВЫБОРА - SWITCH

Оператор выбора ***switch*** (часто его называют переключателем) предназначен для выбора ветви вычислительного процесса исходя из значения управляющего выражения. (При этом значение управляющего выражения сравнивается со значениями в списке целых или символьных констант. Если будет найдено совпадение, то выполнится ассоциированный с совпавшей константой оператор.) Общая форма оператора ***switch*** следующая:

```
switch (выражение)
{
  case постоянная1:
    последовательность операторов
    break;
  case постоянная2:
    последовательность операторов
    break;
  case постоянная3:
    последовательность операторов
    break;
  default:
    последовательность операторов;
}
```

Значение *выражения* оператора ***switch*** должно быть таким, чтобы его можно было выразить целым числом. Это означает, что в управляющем выражении можно использовать переменные целого или символьного типа. Значение управляющего выражения по очереди сравнивается с постоянными в операторах ***case***. Если значение управляющего выражения совпадет с какой-то из постоянных, управление передается на соответствующую метку ***case*** и выполняется последовательность операторов до оператора ***break***. Если оператор ***break*** отсутствует, выполнение последовательности операторов продолжается до тех пор, пока не встретится ***break*** (в другой метке) или не кончится тело оператора ***switch*** (т.е. блок, следующий за ***switch***). Оператор ***default*** выполняется в том случае, когда значение управляющего выражения не совпало ни с одной постоянной. Оператор **i** также может отсутствовать. В этом случае при отсутствии совпадений не выполняется ни один оператор. Количество операторов case практически не ограничено. Ели вы пишете программы вручную, большое количество операторов вам никогда не понадобится.
Оператор ***case*** - это метка, однако он не может быть использован сам по себе, вне оператора ***switch***.
Оператор ***break*** — это один из операторов безусловного перехода. Он может применяться не только в операторе ***switch***, но и в циклах, (см. раздел "Операторы цикла"). Когда в теле оператора ***switch*** встречается оператор ***break***, программа выходит из оператора ***switch*** и выполняет оператор, следующий за фигурной скобкой ***}*** оператора ***switch***.
Об операторе switch очень важно помнить следующее:

- Оператор ***switch*** отличается от ***if*** тем, что в нем управляющее выражение проверяется только на равенство с постоянными, в то время как в ***if*** проверяется любой вид отношения или логического выражения.
- В одном и том же операторе ***switch*** никакие два оператора ***case*** не могут иметь равных постоянных. Конечно, если один ***switch*** вложен в другой, в их операторах ***case*** могут быть совпадающие постоянные.

- Если в управляющем выражении оператора ***switch*** встречаются символьные константы, они автоматически преобразуются к целому типу по принятым в языке С правилам приведения типов.

Оператор ***switch*** часто используется для обработки команд с клавиатуры, например, при выборе пунктов меню.
С точки зрения синтаксиса, присутствие операторов ***break*** внутри ***switch*** не обязательно. Они прерывают выполнение последовательности операторов, ассоциированных с данной константой. Если оператор ***break*** отсутствует, то выполняется следующий оператор ***case***, пока не встретится очередной ***break***, или не будет достигнут конец тела оператора ***switch***. Например, в функции inp_handler для упрощения программы несколько операторов ***break*** опущено, поэтому выполняются сразу несколько операторов case:

```
/* Обработка значения i */
void inp_handler(int i)
{
    int flag;
    flag = -1;

    switch(i)
    {
        case 1:  /* Эти case эти общую */
        case 2:  /* последовательность операторов. */
        case 3:
              flag = 0;
              break;
        case 4:
              flag = 1;
        case 5:
              error(flag);
              break;
        default:
              process(i);
              break;
    }
}
```

Приведенный пример иллюстрирует следующие две особенности оператора ***switch()***.
Во-первых, оператор ***case*** может не иметь ассоциированной с ним последовательности операторов. Тогда управление переходит к следующему ***case***. В этом примере три первых ***case*** вызывают выполнение одной и той же последовательности операторов, а именно:

```
flag = 0;
break;
```

Во-вторых, если оператор ***break*** отсутствует, то выполняется последовательность операторов следующего ***case***. Если ***i*** равно 4, то переменной ***flag*** присваивается значение 1 и, поскольку ***break*** отсутствует, выполнение продолжается и вызывается ***error(flag)***. Если ***i*** равно 5, то ***error()*** будет вызвана со значением переменной ***flag***, равным -1, а не 1.
То, что при отсутствии ***break*** операторы ***case*** выполняются вместе, позволяет избежать ненужного дублирования операторов.

Вложенные операторы switch

Оператор ***switch*** может находиться в теле внешнего по отношению к нему оператора ***switch***. Операторы ***case*** внутреннего и внешнего ***switch*** могут иметь одинаковые константы, в этом случае они не конфликтуют между собой. Например, следующий фрагмент программы вполне работоспособен:

```
switch(x)
{
```

```
  case 1:
        switch(y)
        {
            case 0: printf("Деление на нуль.\n");
                    break;
            case 1: process(x,y);
                    break;
        }
        break;
  case 2:
    .
    .
    .
```

2.11.5. ОПЕРАТОР ЦИКЛА FOR

В языке С, как и в других языках программирования, операторы цикла служат для многократного выполнения последовательности операторов до тех пор, пока выполняется некоторое условие. Условие может быть установленным заранее (как в операторе ***for***) или меняться при выполнении тела цикла (как в ***while*** или ***do-while***).

Цикл for

Во всех процедурных языках программирования циклы ***for*** очень похожи. Однако в С этот цикл особенно гибкий и мощный. Общая форма оператора ***for*** следующая:

```
for (инициализация; условие; приращение) оператор;
```

Цикл ***for*** может иметь большое количество вариаций. В наиболее общем виде принцип его работы следующий. *Инициализация* - это присваивание начального значения переменной, которая называется *параметром цикла*. *Условие* представляет собой условное выражение, определяющее, следует ли выполнять оператор цикла (часто его называют телом цикла) в очередной раз. Оператор *приращение* осуществляет изменение параметра цикла при каждой итерации. Эти три оператора (они называются также *секциями оператора* ***for***) обязательно разделяются точкой с запятой. Цикл ***for*** выполняется, если выражение условие принимает значение ИСТИНА. Если оно хотя бы один раз примет значение ЛОЖЬ, то программа выходит из цикла и выполняется оператор, следующий за телом цикла ***for***.
В следующем примере в цикле for выводятся на экран числа от 1 до 10:

```
#include <sat500.h>

void main(void)
{
    int i;
    ClrScr();
    for(i=1; i <= 10; i++) printf("%d ", i);
    ShowScr();
    while(1);
}
```

В этом примере параметр цикла ***i*** инициализирован числом 1, а затем при каждой итерации сравнивается с числом 10. Пока переменная x меньше 10, вызывается функция printf() и цикл повторяется. При этом ***i*** увеличивается на 1 и опять проверяется условие цикла x <= 10. Процесс повторяется, пока переменная x не станет больше 10. После этого процесс выходит из цикла, а управление передается оператору, следующему за ним. В этом примере параметром цикла является переменная ***i***, при каждой итерации она изменяется и проверяется в секции условия цикла.
В следующем примере в цикле ***for*** выполняется блок операторов:

```
for(x=100; x != 65; x -= 5)
{
    z = x*x;
    printf("Квадрат %d равен %d", x, z);
}
```

Операции возведения переменной ***x*** в квадрат и вызова функции ***printf()*** повторяются, пока ***x*** не примет значение 65. Обратите внимание на то, что здесь параметр цикла уменьшается, он инициализирован числом 100 и уменьшается на 5 при каждой итерации.
В операторе ***for*** условие цикла всегда проверяется перед началом итерации. Это значит, что операторы цикла могут не выполняться ни разу, если перед первой итерацией условие примет значение ЛОЖЬ. Например, в следующем фрагменте программы

```
x = 10;
for(y=10; y!=x; ++y) printf("%d", y);
printf("%d", y);  /* Это единственный printf() который будет выполнен */
```

цикл не выполнится ни разу, потому что при входе в цикл значения переменных ***x*** и ***y*** равны. Поэтому условие цикла принимает значение ЛОЖЬ, а тело цикла и оператор приращение не выполняются. Переменная ***y*** остается равной 10, единственный результат работы этой программы - вывод на экран числа 10 в результате вызова функции ***printf()***, расположенной вне цикла.

Варианты цикла for

В предыдущем разделе рассмотрена наиболее общая форма цикла ***for***. Однако в языке С допускаются некоторые его варианты, позволяющие во многих случаях увеличить мощность и гибкость программы.
Один из распространенных способов усиления мощности цикла ***for*** - применение оператора "запятая" для создания двух параметров цикла. Оператор "запятая" связывает несколько выражений, заставляя их выполняться вместе . В следующем примере обе переменные (***x*** и ***y***) являются параметрами цикла ***for*** и обе инициализируются в этом цикле:

```
for(x=0, y=0; x+y<10; ++x)
{
    y = Data[x];
    y = y - '0'; /* Вычитание из y ASCII-кода нуля */
    ...
}
```

Здесь запятая разделяет два оператора инициализации. При каждой итерации значение переменной ***x*** увеличивается, а значение y вводится из массива. Для выполнения итерации как ***x***, так и ***y*** должны иметь определенное значение. Несмотря на то что значение ***y*** вводится из массива, оно должно быть инициализировано таким образом, чтобы выполнилось условие цикла при первой итерации. Если y не инициализировать, то оно может случайно оказаться таким, что условие цикла примет значение ЛОЖЬ, тело цикла не будет выполнено ни разу.
Следующий пример демонстрирует использование двух параметров цикла. Функция ***converge()*** копирует содержимое одной строки в другую, начиная с обоих концов строки и кончая в ее середине.

```
/* Демонстрация использования 2-х параметров цикла. */

#include <sat500.h>

void converge(char * targ, char * src);
void main(void)
{
    char target[80] = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
    ClrScr();
    converge(target, "Это проверка функции converge().");
```

```
    printf("Строка-результат: %s\n", target);
    ShowScr();
    while(1);
}

/* Эта функция копирует содержимое одной строки в
   другую, начиная с обоих концов и сходясь посередине. */
void converge(char *targ, char *src)
{
    int i, j;

    printf("%s\n", targ);
    for(i=0, j=strlen(src); i<=j; i++, j--)
    {
        targ[i] = src[i];
        targ[j] = src[j];
        printf("%s\n", targ);
    }
}
```

Программа выводит на экран следующее:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
ЭXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
ЭтXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.
ЭтоXXXXXXXXXXXXXXXXXXXXXXXXXXX).
Это XXXXXXXXXXXXXXXXXXXXXXXXX().
Это пXXXXXXXXXXXXXXXXXXXXXXXXe().
Это прXXXXXXXXXXXXXXXXXXXXXXXge().
Это проXXXXXXXXXXXXXXXXXXXXXrge().
Это провXXXXXXXXXXXXXXXXXXXerge().
Это провеXXXXXXXXXXXXXXXXXverge().
Это проверXXXXXXXXXXXXXXXnverge().
Это проверKXXXXXXXXXXXXXonverge().
Это проверкаXXXXXXXXXXconverge().
Это проверка XXXXXXX converge().
Это проверка фXXXXXXи converge().
Это проверка фуXXXXии converge().
Это проверка фунXXции converge().
Это проверка функции converge().
Строка-результат: Это проверка функции converge().
```

В функции ***convergence()*** цикл ***for*** использует два параметра цикла (***i*** и ***j***) для индексации строки с противоположных концов. Параметр ***i*** в цикле увеличивается, а ***j*** — уменьшается. Итерации прекращаются, когда ***i*** становится больше ***j***. Это обеспечивает копирование всех символов.
Проверка параметра цикла на соответствие некоторому условию не обязательна. Условие может быть любым логическим оператором или оператором отношения. Это значит, что условие выполнения цикла может состоять из нескольких условий, или операторов отношения.
Следует помнить, что каждая из трех секций оператора ***for*** может быть любым синтаксически правильным выражением. Эти выражения не всегда каким-либо образом отображают назначение секции.
Другая интересная особенность цикла ***for*** состоит в том, что его секции могут быть вообще пустыми, присутствие в них какого-либо выражения не обязательно.
Инициализацию параметра цикла ***for*** можно сделать за пределами этого цикла, но, конечно, до него. Это особенно уместно, если начальное значение параметра цикла вычисляется достаточно сложно, например:

```
gets(s);  /* читает строку в s */
if(*s) x = strlen(s); /* вычисление длины строки */
```

```
else x = 10;

for( ; x<10; )
{
    printf("%d", x);
    ++x;
}
```

В этом примере секция инициализации оставлена пустой, а переменная x инициализируется до входа в цикл.

Бесконечный цикл

Для создания бесконечного цикла можно использовать любой оператор цикла, но чаще всего для этого выбирают оператор for. Так как в операторе for может отсутствовать любая секция, бесконечный цикл проще всего сделать, оставив пустыми все секции. Это хорошо показано в следующем примере:

```
for( ; ; ) printf("Этот цикл крутится бесконечно.\n");
```

Если условие цикла ***for*** отсутствует, то предполагается, что его значение - ИСТИНА. В оператор ***for*** можно добавить выражения инициализации и приращения, хотя обычно для создания бесконечного цикла используют конструкцию ***for(; ;)***.
Фактически конструкция ***for(; ;)*** не гарантирует бесконечность итераций, потому что в нем может встретиться оператор ***break***, вызывающий немедленный выход из цикла. (Подробно оператор break рассмотрен в этом разделе далее.) В этом случае выполнение программы продолжается с оператора, следующего за закрывающейся фигурной скобкой цикла ***for***:

```
ch = '\0';

for( ; ; )
{
      ch = GetKey();    /* считывание символа */
      if(ch==KEY_LEFT) break;    /* выход из цикла */
}
printf("Вы нажали ВЛЕВО'");
```

В данном примере цикл выполняется до тех пор, пока пользователь не нажмёт на клавиатуре кнопку ВЛЕВО.

Цикл for без тела цикла

Следует учесть, что оператор может быть пустым. Это значит, что тело цикла ***for*** (или любого другого цикла) также может быть пустым. Такую особенность цикла ***for*** можно использовать для упрощения некоторых программ, а также в циклах, предназначенных для того, чтобы отложить выполнение последующей части программы на некоторое время.
Программисту иногда приходится решать задачу удаления пробелов из входного потока. В следующем примере цикл for удаляет начальные пробелы в строке ***str***:

```
for( ; *str == ' '; str++) ;
```

В этом примере указатель ***str*** переставляется на первый символ, не являющийся пробелом. Цикл не имеет тела, так как в нем нет необходимости.
Иногда возникает необходимость отложить выполнение последующей части программы на определенное время. Это можно сделать с помощью цикла ***for*** следующим образом:

```
for(t=0; t<SOME_VALUE; t++) ;
```

Единственное назначение этого цикла - задержка выполнения последующей части программы. Однако следует иметь в виду, что компилятор может оптимизировать объектный код таким образом,

что пропустит этот цикл вообще, поскольку он не выполняет никаких действий, тогда желаемой задержки выполнения последующей части программы не произойдет.

Объявление переменных внутри цикла

Допускается объявление переменных в секции инициализации цикла for. Объявленная таким образом переменная является локальной переменной цикла и ее область действия распространяется на тело цикла.
Рассмотрим следующий пример:

```
/*
    Здесь переменная i является локальной
    переменной цикла, а j видима вне цикла.
*/
int j;
for(int i = 0; i<10; i++)
  j = i * i;

/* i = 10; Это ошибка, переменная i здесь недоступна! */
```

В данном примере переменная ***i*** объявлена в секции инициализации цикла ***for*** и служит параметром цикла. Вне цикла переменная ***i*** невидима.
Поскольку параметр цикла чаше всего необходим только внутри цикла, его объявление в секции инициализации очень удобно и входит в широкую практику.

2.11.6. ОПЕРАТОР ЦИКЛА WHILE

Общая форма цикла ***while*** имеет следующий вид:

```
while (условие) оператор;
```

Здесь *оператор* (тело цикла) может быть пустым оператором, единственным оператором или блоком. *Условие* (управляющее выражение) может быть любым допустимым в языке выражением. *Условие* считается истинным, если значение выражения не равно нулю, а оператор выполняется, если *условие* принимает значение ИСТИНА. Если *условие* принимает значение ЛОЖЬ, программа выходит из цикла и выполняется следующий за циклом оператор.
В следующем примере выполнение происходит до тех пор, пока пользователь не нажмёт кнопку ВВЕРХ:

```
char wait_for_up(void)
{
    char ch;

    ch = 0;  /* инициализация ch */
    while(ch != KEY_LEFT  || ch ==0)
    {
        ch = GetKey();
        ...
    }
    return ch;
}
```

Переменная ***ch*** является локальной, ее значение при входе в функцию произвольно, поэтому сначала значение ***ch*** инициализируется значением `0`. Условие цикла ***while*** истинно, если **ch** не равно `KEY_LEFT` или ***ch*** равно `0`. Поскольку ***ch*** инициализировано `0`, то условие истинно и цикл начинает выполняться. Условие проверяется при каждом нажатии клавиши пользователем. При нажатии на кнопку отличную от кнопки ВВЕРХ условие становится ложным и выполнение цикла прекращается.

Как и в цикле ***for***, в цикле ***while*** условие проверяется перед началом итерации. Это значит, что если условие ложно, тело цикла не будет выполнено. Благодаря этому нет необходимости вводить в программу отдельное условие перед циклом. Рассмотрим это на примере функции ***pad()***, которая добавляет пробелы в конец строки и делает ее длину равной предварительно заданной величине. Если строка уже имеет необходимую длину, то пробелы не добавляются:

```
#include <sat500.h>

void pad(char *s, int length);

void main(void)
{
    char str[40];
    ClrScr();
    strcpy(str,"это проверка");
    pad(str,30);
    printf("%d", strlen(str));
    ShowScr();
    while(1);
}

/* Добавление пробелов в конец строки. */
void pad(char *s, int length)
{
    int len;
    len = strlen(s); /* определение длины строки */
    while(len<length)
    {
        s[len] = ' '; /* вставка пробелов */
        len++;
    }
    s[len]= '\0'; /* строка должна заканчиваться нулем */
}
```

Аргументами функции ***pad()*** являются ***s*** (указатель на исходную строку) и ***length*** (требуемое количество символов в строке). Если длина строки ***s*** при входе в функцию равна или больше ***length***, то цикл ***while*** не выполняется. В противном случае ***pad()*** добавляет требуемое количество пробелов, а библиотечная функция ***strlen()*** возвращает длину строки.
Если выполнение цикла должно зависеть от нескольких условий, можно создать так называемую управляющую переменную, значения которой присваиваются разными операторами тела цикла. Рассмотрим следующий пример:

```
void func1(void)
{
    int working;
    working = 1; /* т.е. ИСТИНА */
    while(working)
    {
        working = process1();
        if(working)
          working = process2();
        if(working)
          working = process3();
    }
}
```

В этом примере переменная ***working*** является управляющей. Любая из трех функций может возвратить значение 0 и этим прервать выполнение цикла. Тело цикла ***while*** может быть пустым. Например, цикл

```
while((ch=GetKey()) == 0) ;
```

выполняется до тех пор, пока пользователь не нажмёт какую либо кнопку. Напоминаем, что оператор присваивания выполняет две задачи: присваивает значение выражения справа переменной слева и возвращает это значение как своё собственное.

2.11.7. ОПЕРАТОР ЦИКЛА DO-WHILE

В отличие от циклов ***for*** и ***while***, которые проверяют свое условие перед итерацией, ***do-while*** делает это после нее. Поэтому цикл ***do-while*** всегда выполняется как минимум один раз. Общая форма цикла ***do-while*** следующая:

```
do
{
    оператор;
}while (условие);
```

Если *оператор* не является блоком, фигурные скобки не обязательны, но их почти всегда ставят, чтобы оператор достаточно наглядно отделялся от *условия*. Итерации оператора ***do-while*** выполняются, пока *условие* не примет значение ЛОЖЬ.
В следующем примере в цикле ***do-while*** числа считываются из массива, пока не встретится число, меньшее или равное 100:

```
do
{
      num=Data[i++];
} while(num > 100);
```

2.11.8. ОПЕРАТОРЫ ПЕРЕХОДА

В языке С определены четыре оператора перехода: ***return, goto, break*** и ***continue***. Операторы ***return*** и ***goto*** можно использовать в любом месте внутри функции. Операторы ***break*** и ***continue*** можно использовать в любом из операторов цикла. Как указывалось ранее в этом разделе, ***break*** можно также использовать в операторе ***switch***.

Оператор return

Оператор **return** используется для выхода из функции. Отнесение его к категории операторов перехода обусловлено тем, что он заставляет программу перейти в точку вызова функции. Оператор ***return*** может иметь ассоциированное с ним значение, тогда при выполнении данного оператора это значение возвращается в качестве значения функции. В функциях типа ***void*** используется оператор ***return*** без значения.
Если тип функции отличен от ***void***, то ее оператор return обязательно должен иметь значение.
Общая форма оператора **return** следующая:

```
return выражение;
```

Выражение присутствует только в том случае, если функция возвращает значение. Это значение *выражения* становится возвращаемым значением функции.
Внутри функции может присутствовать произвольное количество операторов ***return***. Выход из функции происходит тогда, когда встречается один из них. Закрывающаяся фигурная скобка ***}***

также вызывает выход из функции. Выход программы на нее эквивалентен оператору ***return*** без значения.
Функция, определенная со спецификатором ***void***, не может содержать ***return*** со значением. Так как эта функция не возвращает значения, в ней не может быть оператора ***return***, возвращающего значение.

Оператор goto

Кроме ***goto***, в языке С есть другие операторы управления (например ***break***, ***continue***), поэтому необходимости в применении ***goto*** практически нет. В результате чрезмерного использования операторов ***goto*** программа плохо читается. То есть оператор ***goto*** весьма непопулярен, более того, считается, что в программировании не существует ситуаций, в которых нельзя обойтись без оператора ***goto***. Но в некоторых случаях его применение все же уместно. Иногда, при умелом использовании, этот оператор может оказаться весьма полезным, например, если нужно покинуть глубоко вложенные циклы.
Для оператора **goto** всегда необходима *метка*. *Метка* - это идентификатор с последующим двоеточием. *Метка* должна находится в той же функции, что и ***goto***, переход в другую функцию невозможен. Общая форма оператора ***goto*** следующая:

```
goto метка;
...
метка:
```

Метка может находиться как до, так и после оператора ***goto***. Например, используя оператор ***goto***, можно выполнить цикл от 1 до 10:

```
x = 1;
loop1:
      x++;
      if(x<=100) goto loop1;
```

Оператор break

Оператор ***break*** применяется в двух случаях. Во-первых, в операторе ***switch*** с его помощью прерывается выполнение последовательности ***case*** (см. раздел "Оператор выбора - switch"). В этом случае оператор ***break*** не передает управление за пределы блока. Во-вторых, оператор ***break*** используется для немедленного прекращения выполнения цикла без проверки его условия, в этом случае оператор ***break*** передает управление оператору, следующему после оператора цикла.
Когда внутри цикла встречается оператор ***break***, выполнение цикла безусловно (т.е. без проверки каких-либо условий.) прекращается и управление передается оператору, следующему за ним.
Например, программа

```
#include <sat500.h>

int main(void)
{
    int t;
    ClrScr();
    for(t=0; t<100; t++)
    {
        printf("%d ", t);
        if(t==10) break;
    }
    ShowScr();
    while(1);
}
```

выводит на экран числа от 0 до 10. После этого выполнение цикла прекращается оператором ***break***, условие t < 100 при этом игнорируется.

Оператор ***break*** часто используется в циклах, в которых некоторое событие должно вызвать немедленное прекращение выполнения цикла.
Оператор ***break*** вызывает выход только из внутреннего цикла. Например, программа

```
for(t=0; t<20; ++t)
{
    count = 1;
    for(;;)
    {
        printf("%d ", count);
        count++;
        if(count==10) break;
    }
}
```

20 раз выводит на экран числа от 1 до 9. Оператор ***break*** передаёт управление внешнему циклу ***for***.
Если оператор ***break*** присутствует внутри оператора ***switch***, который вложен в какие-либо циклы, то ***break*** относится только к ***switch***, выход из цикла не происходит.

Оператор continue

Можно сказать, что оператор ***continue*** немного похож на ***break***. Оператор ***break*** вызывает прерывание цикла, а ***continue*** - прерывание текущей итерации цикла и осуществляет переход к следующей итерации. При этом все операторы до конца тела цикла пропускаются. В цикле ***for*** оператор continue вызывает выполнение операторов приращения и проверки условия цикла. В циклах ***while*** и ***do-while*** оператор ***continue*** передаёт управление операторам проверки условий цикла. В следующем примере программа подсчитывает количество пробелов в строке:

```
/* Подсчет количества пробелов */
int space_count(char * s)
{
    int count;
    char * str = s;
    for(count=0; *str; str++)
    {
        if(*str != ' ') continue;
        count++;
    }
    printf("%d пробелов\n", count);
    return count;
}
```

Каждый символ строки сравнивается с пробелом. Если сравниваемый символ не является пробелом, оператор ***continue*** передает управление в конец цикла ***count*** и выполняется следующая итерация. Если символ является пробелом, значение переменной ***count*** увеличивается на 1.
В следующем примере оператор ***continue*** применяется для выхода из цикла ***while*** путем передачи управления на условие цикла:

```
void code(void)
{
    char done, ch;
    done = 0;
    while(!done)
    {
        ch = GetKey();
        if(ch == KEY_RIGHT)
        {
              done = 1;
```

```
                continue;
            }
            if(ch)putchar(ch+'0'); /* печать символа */
        }
}
```

Функция прекращает работу при нажатии на кнопку ВПРАВО.

2.11.9. ОПЕРАТОР-ВЫРАЖЕНИЕ

Выражения были подробно рассмотрены в предыдущем разделе. Но говоря об операторах, будет уместно добавить несколько слов и о выражениях. Любое выражение, которое заканчивается точкой с запятой, является оператором. Рассмотрим следующие примеры:

```
func();  /* вызов функции */

a = b+c; /* оператор присваивания */

b+f();   /* правильный, но "странный" оператор */

;        /* пустой оператор */
```

Первый оператор выполняет вызов функции, второй - присваивание. Третий оператор выглядит странно, но транслятор все же не укажет на ошибку. В этом операторе необходимые действия, видимо, выполняются функцией ***f()***. Последний пример - пустой оператор, не выполняющий никакого действия.

2.11.10. БЛОК ОПЕРАТОРОВ

Блок - это последовательность операторов, заключённых в фигурные скобки и рассматриваются как одна программная единица. Операторы, составляющие блок, логически связаны друг с другом. Иногда блок называют составным оператором. Блок всегда начинается открывающейся фигурной скобкой ***{*** и заканчивается закрывающейся ***}***. Чаще всего блок используется как составная часть какого-либо оператора, выполняющего действие над группой операторов, например, ***if*** или ***for***. Однако блок можно поставить в любом месте, где может находиться оператор, как это показано в следующем примере:

```
#include <sat500.h>

void main(void)
{
    int i;
    ClrScr();
    {
        /* блок операторов */
        i = 120;
        printf("%d", i);
    }
    ShowScr();
}
```

2.12. МАССИВЫ И СТРОКИ

Массив - это набор переменных одного типа, имеющих одно и то же имя. Доступ к конкретному элементу массива осуществляется с помощью индекса. В языке С все массивы располагаются в отдельной непрерывной области памяти. Первый элемент массива располагается по самому меньшему адресу, а последний — по самому большому. Строка представляет собой массив символьных переменных, заканчивающийся специальным нулевым символом, это наиболее распространенный тип массива.
Массивы и указатели тесно связаны. То, что может быть сказано о массивах, чаще всего непосредственно относится и к указателям, и наоборот. В этом разделе рассматриваются массивы, указатели будут подробно рассмотрены в другом разделе.

2.12.1. ОДНОМЕРНЫЕ МАССИВЫ

Общая форма объявления *одномерного массива* имеет следующий вид:

```
тип имя_переменной [размер];
```

Как и другие переменные, массив должен быть объявлен явно, чтобы компилятор выделил для него определенную область памяти (т.е. разместил массив). Здесь тип обозначает базовый тип массива, являющийся типом каждого элемента. Размер задает количество элементов массива. Например, следующий оператор объявляет массив из 100 элементов типа ***int*** под именем ***balance***:

```
int balance[100];
```

Размер массива должен быть указан явно с помощью выражения-константы. Таким образом, в программе размер массива определяется во время компиляции и впоследствии остается неизменным. Доступ к элементу массива осуществляется с помощью имени массива и индекса. Индекс элемента массива помещается в квадратных скобках после имени. Например, оператор

```
balance[3] = 1223;
```

присваивает 3-му элементу массива balance значение 1223.
Индекс первого элемента любого массива в языке С равен нулю. Поэтому оператор

```
char p[10];
```

объявляет массив символов из 10 элементов — от p[0] до p[9].

В следующей программе присваиваются и показываются значения элементов массива x[12] целого типа с индексами от 0 до 11:

```
#include <sat500.h>
void main(void)
{
    int x[12]; //объявление массива 100 целых
    int t;
    ClrScr();
    BigFont=0;
    //присваивание массиву значений от 100 до 111
    for(t=0; t<12; ++t) x[t] = 100+t;
    //вывод на экран содержимого x
    for(t=0; t<12; ++t) printf("x[%d]=%d \n",t, x[t]);
    ShowScr();
    while(1);
}
```

Результат:

Объем памяти, необходимый для хранения массива, непосредственно определяется его *типом* и *размером*. Для одномерного массива количество байтов памяти вычисляется следующим образом:

```
количество_байтов = sizeof(базовый_тип) × длина_массива
```

Во время выполнения программы на С не проверяется ни соблюдение границ массивов, ни их содержимое. В область памяти, занятую массивом, может быть записано что угодно. Программист должен сам, где это необходимо, ввести проверку границ индексов. Следующий пример программы компилируется без ошибки, однако при выполнении происходит нарушение границы массива count и разрушение соседних участков памяти:

```
int count[10], i;

/* здесь нарушена граница массива count */
for(i=0; i<100; i++) count[i] = i;
```

Можно сказать, что *одномерный массив* - это *список*, хранящийся в непрерывной области памяти в порядке индексации. На рисунке ниже показано, как хранится в памяти массив a, начинающийся по адресу 1000 и объявленный как

```
char a[7];
```

Таблица 10 - Массив из семи символов, начинающийся по адресу 1000

Элемент	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Адрес	1000	1001	1002	1003	1004	1005	1006	1007

2.12.2. РАЗМЕЩЕНИЕ В ПАМЯТИ

Глобальные переменные по умолчанию располагаются в RAM памяти (оперативное запоминающее устройство).
Однако при необходимости можно разместить массив в ROM памяти (память только для чтения). В этом случае значения массива будет невозможно изменить. В контроллере CAT-500 в качестве ROM используется FLASH память микроконтроллера.
Для размещения переменной памяти в неизменяемой памяти следует использовать модификатор *const* следующим образом:

```
const тип имя_переменной[]=значение;
```

Например создание создание массива 5 целочисленных неизменяемых переменных с присвоением значением будет выглядеть следующим образом:

```
const int Data[5]={100,200,300,400,500};
```

В const памяти располагаются выводимые на дисплей строки. Например вот таким образом определяется строка, которую можно вывести на дисплей:

```
const char MsgOk[]="Успешно выполнено...\n";
```

В последний символ строки `MsgOk` автоматически добавляется нулевой символ, если он не задан явно. Таким образом массив `MsgOk` превращается в С-строку.
Конечный нулевой символ С-строки также можно задать явно:

```
const char MsgOk[]="Успешно выполнено...\n\0";
```

В этом случае компилятор не будет добавлять лишний нулевой символ.
Модификатор const обычно используется для хранения в программе некоторых неизменяемых данных - таблиц данных или массивов строковых данных.

Также компилятор автоматически создаёт const строковые переменные, которые задаются непосредственно в коде в виде строки в кавычках.
Например спецификатор формата в функции printf может быть задан прямо в вызове функции:

```
printf("Значение: %d",i);
```

Фактически компилятор выполнит вот такие действия:

```
const char c$0[]="Значение: %d";
printf(c$0,i);
```

Во время компиляции строка `"Значение: %d"` будет автоматически помещена в *const* память и ей будет присвоено уникальное имя, которое и буде передано в качестве фактического параметра формата.
Это выполняется для любых вариантов задания строки в двойных кавычках - любая строка в двойных кавычках обрабатывается как автоматически создаваемая *cons char* массив.
Примеры:

```
puts("Результат:\n")
...
char * Msg(void)
{
     return "Сообщение";
}
```

Выравнивание

Компилятор C-Script не выполняет выравнивание переменных и массивов данных. Другими словами всегда используется побайтовое выравнивание.
Ещё одной особенностью компилятора является то, что относительный порядок расположения переменных точно СООТВЕТСТВЕТ порядку их объявления.
Например, если в како-либо месте программы сделано такое объявление:

```
const char MsgOk[]="Успешно выполнено...\n";
const char MsgError[]="Произошла ошибка...\n";
```

то эти два массива будут расположены один за другим в порядке объявления и займут 22+21 байтов.

```
#include <sat500.h>

const char MsgOk[]="Успешно выполнено...\n";
const char MsgError[]="Произошла ошибка...\n";

void main(void)
{
    ClrScr();
    BigFont=0;
    printf("sizeof(MsgOk)=%d\n",sizeof(MsgOk));
    puts("Текст MsgOk:\n");
    puts(MsgOk);
    printf("sizeof(MsgError)=%d\n",sizeof(MsgError));
```

```
    puts("Текст MsgError:\n");
    puts(MsgError);
    ShowScr();
    while(1);
}
```

Результат:

2.12.3. СОЗДАНИЕ УКАЗАТЕЛЯ НА МАССИВ

Указатель на 1-й элемент массива можно создать путем присваивания ему имени массива без индекса. Например, если есть объявление

```
int sample[10];
```

то в качестве указателя на 1-й элемент массива можно использовать имя ***sample***. В следующем фрагменте программы адрес 1-го элемента массива ***sample*** присваивается указателю ***p***:

```
int *p;
int sample[10];

p = sample;
```

В обеих переменных (***p*** и ***sample***) хранится адрес 1-го элемента, отличаются эти переменные только тем, что значение ***sample*** в программе изменять нельзя. Адрес первого элемента можно также получить, используя оператор получения адреса ***&***. Например, выражения ***sample*** и ***&sample[0]*** имеют одно и то же значение. Тем не менее, в профессионально написанных программах вы не встретите выражения ***&sample[0]***.

2.12.4. ПЕРЕДАЧА ОДНОМЕРНОГО МАССИВА В ФУНКЦИЮ

В языке С нельзя передать весь массив как аргумент функции. Однако можно передать указатель на массив, т.е. имя массива без индекса. Например, в представленной программе в ***func1()*** передается указатель на массив *i*:

```
void main(void)
{
      int i[10];

      func1(i);

      /* ... */
}
```

Если в функцию передается указатель на одномерный массив, то в самой функции его можно объявить одним из трех вариантов: как указатель, как массив определенного размера и как массив без определенного размера. Например, чтобы функция ***func1()*** получила доступ к значениям, хранящимся в массиве **i**, она может быть объявлена как

```
void func1(int *x) /* указатель */
{
      /* ... */
}
```

или как

```
void func1(int x[10]) /* массив определенного размера */
{
      /* ... */
}
```

и наконец как

```
void func1(int x[]) /* массив без определенного размера */
{
      /* ... */
}
```

Эти три объявления тождественны, потому что каждое из них сообщает компилятору одно и то же: в функцию будет передан указатель на переменную целого типа. В первом объявлении используется указатель, во втором - стандартное объявление массива. В последнем примере измененная форма объявления массива сообщает компилятору, что в функцию будет передан массив неопределенной длины. Как видно, длина массива не имеет для функции никакого значения, потому что в С проверка границ массива не выполняется. Эту функцию можно объявить даже так:

```
void func1(int x[32])
{
      /* ... */
}
```

И при этом программа будет выполнена правильно, потому что компилятор не создает массив из 32 элементов, а только подготавливает функцию к приему указателя.

2.12.5. СТРОКИ

Одномерный массив наиболее часто применяется в виде строки символов. Строка - это одномерный массив символов, заканчивающийся нулевым символом. В языке С признаком окончания строки (нулевым символом) служит символ '\0'. Таким образом, строка содержит символы, составляющие строку, а также нулевой символ. Это единственный вид строки, определённый в С. Объявляя массив символов, предназначенный для хранения строки, необходимо предусмотреть место для нуля, т.е. указать его размер в объявлении на один символ больше, чем наибольшее предполагаемое количество символов. Например, объявление массива str, предназначенного для хранения строки из 10 символов, должно выглядеть так:

```
char Str[11];
```

Последний байт предназначен для нулевого символа.
Записанная в тексте программы строка символов, заключённых в двойные кавычки, является строковой константой, например,

```
"некоторая строка"
```

В конец строковой константы компилятор автоматически добавляет нулевой символ.

Строковый массив может быть размещён в FLASH памяти контроллера. Для этого следует использовать модификатор const следующим образом:

```
const char StrInFlash[]="Not found...";
```

Длина строки в FLASH памяти может быть до 16383 байтов.

Для задания длинной строки можно воспользоваться следующими способами:
1. Задавать строку в виде нескольких подстрок, разделённых пробелами или переводами строк. Например:

```
const char MyStr[]="Server "
                   "not found..."
                   "Please restart !";
```

Приведёт к созданию одной строки: "Server not found...Please restart !"

2. Использовать символ перевода строки \:
Например:

```
const char MyStr[]="Server \
not found...\
Please restart !";
```

Приведёт к созданию одной строки: "Server not found...Please restart !"

Для обработки строк в С определено много различных библиотечных функций. Чаще всего используются следующие функции:

Имя функции	***Выполняемое действие***
strcpy(s1,s2)	Копирование s2 в s1
strcat(s1,s2)	Конкатенация (присоединение) s2 в конец s1
strlen(s1)	Возвращает длину строки s1
strcmp(s1,s2)	Возвращает 0, если s1 и s2 совпадают, отрицательное значение, если s1<s2 и положительное значение, если s1>s2
strchr(s1,ch)	Возвращает указатель на первое вхождение символа ch в строку s1
strstr(s1,s2)	Возвращает указатель на первое вхождение строки s2 в строку s1

Эти функции объявлены в заголовочном файле *<string.h>*.

Применение библиотечных функций обработки строк иллюстрируется следующим примером:

```
#include <sat500.h>
#include <string.h>

char s1[80]="Алло!";
char s2[]="Алло!";

void main(void)
{
    ClrScr();
    printf("Длина: %d %d\n", strlen(s1), strlen(s2));
    if(!strcmp(s1, s2)) printf("Строки равны\n");
    strcat(s1, s2);
    printf("%s\n", s1);
    strcpy(s1, "Проверка.\n");
    printf(s1);
    if(strchr("Алло", 'л')) printf(" л есть в Алло\n");
    if(strstr("Привет", "ив")) printf(" найдено ив ");
    ShowScr();
    while(1);
}
```

Если эту программу выполнить, то на экран будет выведено следующее:

Следует помнить, что *strcmp()* принимает значение ЛОЖЬ, если строки совпадают (хоть это и несколько нелогично). Поэтому в тесте на совпадение нужно использовать логический оператор отрицания !.

2.12.6. ДВУХМЕРНЫЕ МАССИВЫ

Двухмерные и многомерные массивы компилятором С-Скрипт не поддерживаются.

2.12.7. ИНДЕКСАЦИЯ УКАЗАТЕЛЕЙ

Указатели и массивы тесно связаны друг с другом. Имя массива без индекса - это указатель на первый (начальный) элемент массива. Рассмотрим, например, следующий массив:

```
char p[10];
```

Следующие два выражения идентичны:

```
p
&p[0]
```

Выражение

```
p == &p[0]
```

принимает значение ИСТИНА, потому что адрес 1-го элемента массива - это то же самое, что и адрес массива.
Как уже указывалось, имя массива без индекса представляет собой указатель. И наоборот, указатель можно индексировать как массив. Рассмотрим следующий фрагмент программы:

```
int * p, i[10];
p = i;
p[5] = 100;  /* в присвоении используется индекс */
*(p+5) = 100; /* в присвоении используется адресная арифметика */
```

Оба оператора присваивания заносят число 100 в 6-й элемент массива ***i***. Первый из них индексирует указатель ***p***, во втором применяются правила адресной арифметики. В обоих случаях получается один и тот же результат. (Подробно указатели и адресная арифметика рассматриваются в отдельном разделе *Указатели*)
Правила адресной арифметики требуют явного преобразования указателя на массив в указатель на базовый тип. Указатели используются для обращения к элементам массива потому, что часто операции адресной арифметики выполняются быстрее, чем индексация массива.

2.12.8. ИНИЦИАЛИЗАЦИЯ МАССИВОВ

В языке С массивы при объявлении можно инициализировать. Общая форма инициализации массива аналогична инициализации переменной:

```
тип имя_массива[размер] = {список_значений};
```

Список_значений представляет собой список констант, разделенных запятыми. Типы констант должны быть совместимыми с типом массива. Первая константа присваивается первому элементу

массива, вторая — второму и так далее. После закрывающейся фигурной скобки точка с запятой обязательна.
В следующем примере массив целых из 10 элементов инициализируется числами от 1 до 10:

```
int i[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Здесь элементу i[0] присваивается 1, а i[9] - 10.
Символьные массивы, содержащие строки, можно инициализировать строковыми константами:

```
char имя_массива[размер] = "строка";
```

В следующем примере массив str инициализируется фразой "Язык С":

```
char str[9] = "Язык С";
```

Это объявление можно записать так:

```
char str[9] = {'Я', 'з', 'ы', 'к', ' ', 'С', '\0'};
```

Строка кончается нулевым символом, поэтому при объявлении необходимо задавать размер массива, достаточный для того, чтобы этот символ поместился в нем. В предыдущем примере размер строки задан равным 7, хотя во фразе "Язык С" содержится 6 символов. Если строка инициализируется строковой константой, компилятор автоматически добавляет нулевой символ в конец строки.

Инициализация безразмерных массивов

Предположим, что необходимо создать таблицу сообщений об ошибках, используя инициализацию массивов:

```
char e1[12] = "Ошибка чтения\n";
char e2[13] = "Ошибка записи\n";
char e3[26] = "Нельзя получить значение\n";
```

Для задания размера массива пришлось бы вручную подсчитывать количество символов в каждом сообщении. Однако в языке С есть конструкция, благодаря которой компилятор автоматически определяет необходимую длину строки. Если в операторе инициализации массива не указан размер массива, компилятор создает массив такого размера, что в нем умещаются все инициализирующие элементы. Таким образом создается безразмерный массив. Используя этот метод, предыдущий пример можно записать так:

```
char e1[] = "Ошибка чтения\n";
char e2[] = "Ошибка записи\n";
char e3[] = "Нельзя получить значение\n";
```

Кроме уменьшения трудоемкости, инициализация безразмерных массивов полезна тем, что позволяет изменять длину любого сообщения, не заботясь о соблюдении границ массивов.
Преимущество безразмерного объявления массива состоит в том, что можно изменять длину таблицы, не заботясь о размере массива.

Если строка длинная, то можно использовать наклонную черту следующим образом:

```
char MyStr[]="Это моя \
очень \
длинная строка";
```

Это эквивалентно следующему объявлению:

```
char MyStr[]="Это моя очень длинная строка";
```

Для размещения переменной памяти в неизменяемой памяти (FLASH память в контроллере CAT-500) следует использовать модификатор *const* следующим образом:

```
const тип имя_переменной[]=значение;
```

Более подробно см. раздел *Размещение в памяти*.

2.13. УКАЗАТЕЛИ

Правильное понимание и использование указателей особенно необходимо для составления хороших программ на языке С. Указатели являются средством, с помощью которого функция может изменять значения передаваемых в нее аргументов.
Таким образом, указатели являются весьма мощным средством языка С. Но и весьма опасным. Например, если указатель содержит неправильное значение, программа может потерпеть крах. Указатели весьма опасны еще и потому, что легко ошибиться при их использовании. К тому же ошибки, связанные с неправильными значениями указателей, найти очень трудно.

2.13.1. ЧТО ТАКОЕ УКАЗАТЕЛИ

Указатель - это переменная, значением которой является адрес некоторого объекта (обычно другой переменной) в памяти компьютера. Например, если одна переменная содержит адрес другой переменной, то говорят, что первая переменная указывает (ссылается) на вторую. Это иллюстрируется с помощью рисунка ниже.

2.13.2. УКАЗАТЕЛЬНЫЕ ПЕРЕМЕННЫЕ

Как известно, переменную, являющуюся указателем, нужно соответствующим образом объявить. Объявление указателя состоит из имени базового типа, символа * и имени переменной. Общая форма объявления указателя следующая:

```
тип *имя;
```

Здесь *тип* - это базовый тип указателя, им может быть любой правильный тип. Имя определяет имя переменной-указателя.
Базовый тип указателя определяет тип объекта, на который указатель будет ссылаться. Фактически указатель любого типа может ссылаться на любое место в памяти. Однако выполняемые с указателем операции существенно зависят от его типа. Например, если объявлен указатель типа ***int ****, компилятор предполагает, что любой адрес, на который он ссылается, содержит переменную типа ***int***, хоть это может быть и не так. Следовательно, объявляя указатель, необходимо убедиться, что его тип совместим с типом объекта, на который он будет ссылаться.

2.13.3. ОПЕРАЦИИ ДЛЯ РАБОТЫ С УКАЗАТЕЛЯМИ

Операции для работы с указателями рассматривались ранее. Приведем их обзор. В языке С определены две операции для работы с указателями: ***** и ***&***.
Оператор ***&*** - это унарный оператор, возвращающий адрес своего операнда. (Напомним, что унарный оператор имеет один операнд). Например, оператор

```
m = &count;
```

присваивает переменной ***m*** адрес переменной ***count***. Можно сказать, что адрес - это номер первого байта участка памяти, в котором хранится переменная. Адрес и значение переменной - это совершенно разные понятия. Оператор ***&*** можно представить себе как оператор, возвращающий адрес объекта. Следовательно, предыдущий пример можно прочесть так: "*переменной m присваивается адрес переменной count*".
Предположим, переменная ***count*** хранится в ячейке памяти под номером 2000, а ее значение равно 100. Тогда переменной ***m*** будет присвоено значение 2000.
Вторая операция для работы с указателями (ее знак, т.е. оператор, *****) выполняет действие, обратное по отношению к ***&***. Оператор ***** - это унарный оператор, возвращающий значение переменной, расположенной по указанному адресу. Например, если ***m*** содержит адрес переменной ***count***, то оператор

```
q = *m;
```

присваивает переменной ***q*** значение переменной ***count***. Таким образом, ***q*** получит значение 100, потому что по адресу 2000 расположена переменная ***count***, которая имеет значение 100. Действие оператора ***** можно выразить словами "*значение по адресу*", тогда предыдущий оператор может быть прочитан так: "*q получает значение переменной, расположенной по адресу m*".

2.13.4. УКАЗАТЕЛЬНЫЕ ВЫРАЖЕНИЯ

В общем случае выражения с указателями подчиняются тем же правилам, что и обычные выражения. В этом разделе рассматривается применение указательных выражений в операциях присваивания, преобразования типов, а также в операциях "указательной" арифметики.

Присваивание указателей

Указатель можно использовать в правой части оператора присваивания для присваивания его значения другому указателю. Если оба указателя имеют один и тот же тип, то выполняется простое присваивание, без преобразования типа. В следующем примере

Пример:

```
#include <sat500.h>
char v[12];

void main(void)
{
```

```
    int x = 99;
    int *p1,*p2;
    p1 = &x;
    p2 = p1;
    ClrScr();
    BigFont=false;
    //Указатель p1
    printf("Адрес указателя p1:%d\n", &p1);
    printf("Значение указателя p1:%d\n", p1);
    //Печать значения *p1
    printf("Значение из p1:%d\n",*p1);
    //Указатель p2
    printf("Адрес указателя p2:%d\n", &p2);
    printf("Значение указателя p2:%d\n", p2);
    //Печать значения *p2
    printf("Значение из p1:%d\n",*p2);
    ShowScr();
    while(1);
}
```

После присваивания

```
p1 = &x;
p2 = p1;
```

оба указателя (p1 и p2) ссылаются на x. То есть, оба указателя ссылаются на один и тот же объект. Программа выводит на экран следующее:

Результат:

Допускается присваивание указателя одного типа указателю другого типа. Однако для этого необходимо выполнить явное преобразование типа указателя (операция приведения типов), которая рассматривается далее.

Преобразование типа указателя

Указатель можно преобразовать к другому типу. Эти преобразования бывают двух видов: с использованием указателя типа ***void** ** и без его использования.
В языке C допускается присваивание указателя типа ***void** ** указателю любого другого типа (и наоборот) без явного преобразования типа указателя. Тип указателя ***void** ** используется, если тип объекта неизвестен. Например, использование типа ***void** ** в качестве параметра функции позволяет передавать в функцию указатель на объект любого типа, при этом сообщение об ошибке не генерируется. Также он полезен для ссылки на произвольный участок памяти, независимо от размещенных там объектов.
В отличие от ***void** **, преобразования всех остальных типов указателей должны быть всегда явными (т.е. должна быть указана операция приведения типов). Однако следует учитывать, что преобразование одного типа указателя к другому может вызвать непредсказуемое поведение программы. Например, в следующей программе делается попытка присвоить значение x переменной y посредством указателя p. При компиляции программы сообщение об ошибке не генерируется, однако результат работы программы неверен.

Пример

```
#include <sat500.h>

void main(void)
{
    int x = 1000, y;
```

```
    char  * p;
    ClrScr();
    /* В следующем операторе указателю  p
        (присваивается значение, ссылающееся на int. */
    p = (char *) &x;
    /* Следующий оператор работает не так, как ожидается. */
    y = *p;
    /* Следующий оператор не выведет число  1000. */
    printf("Значение x равно: %d (Это не так!)", y);
    ShowScr();
    while(1);
}
```

Результат:

Обратите внимание на то, что операция приведения типов применяется в операторе присваивания адреса переменной x (он имеет тип int *) указателю p, тип которого char *. Преобразование типа выполнено корректно, однако программа работает не так, как ожидается. Для разъяснения проблемы вспомним, что переменная int занимает в памяти 4 байта, а char 1 байт. Указатель p объявлен как указатель на char переменную (т.е. типа char), поэтому оператор присваивания

```
y = *p;
```

передаст переменной y только 1 байт информации, а не 4 байта, необходимых для int. Несмотря на то, что p ссылается на объект int, оператор присваивания выполнит действие с объектом типа char, потому что p объявлен как указатель на char. Поэтому такое использование указателя p неправильное.
Число 1000 в шестнадцатиричном представлении - это 0x03E8. На печать был выведен только один младший байт числа - 0xE8. Учитывая, что это char переменная, т.е. знаковая, то значение (char)0xE8 как раз и есть -24.
Приведенный пример подтверждает то, что операции с указателями выполняются в зависимости от базового типа указателей. Синтаксически допускается ссылка на объект с типом, отличным от типа указателя, однако при этом указатель будет "думать", что он ссылается на объект своего типа. Таким образом, операции с указателями управляются типом указателя, а не типом объекта, на который он ссылается.
Разрешен еще один тип преобразований: преобразование целого в указатель и наоборот. В этом случае необходимо применить операцию приведения типов (явное преобразование типа). Однако пользоваться этим средством нужно очень осторожно, потому что при этом легко получить непредсказуемое поведение программы. Явное преобразование типа не обязательно, если преобразуется нуль, то есть нулевой указатель.

Адресная арифметика

В языке C допустимы только две арифметические операции над указателями: *суммирование* и *вычитание*. Предположим, текущее значение указателя ***p1*** типа ***int **** равно 2000. Мы также знаем, что переменная типа ***int*** занимает в памяти 4 байта. Тогда после операции увеличения

```
p1++;
```

указатель ***p1*** принимает значение 2004, а не 2001. То есть, при увеличении на 1 указатель ***p1*** будет ссылаться на следующее целое число. Это же справедливо и для операции уменьшения. Например, если ***p1*** равно 2000, то после выполнения оператора

```
p1--;
```

значение ***p1*** будет равно 1996.

Пример

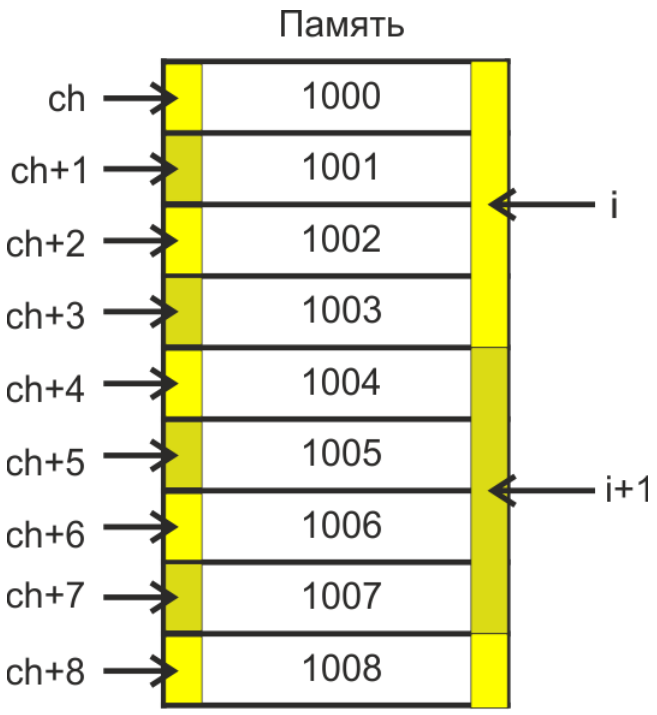
```
#include <sat500.h>

int Var[32];
int * pvar=&Var;

void main(void)
{
    ClrScr();
    BigFont=0;
    GotoXY(0,4);
    printf("Значение pvar равно: %d\n",pvar);
    pvar++;
    printf("Значение pvar равно: %d\n",pvar);
    pvar+=2;
    printf("Значение pvar равно: %d\n",pvar);
    ShowScr();
    while(1);
}
```

Результат:

Операции адресной арифметики подчиняются следующим правилам. После выполнения операции увеличения над указателем, данный указатель будет ссылаться на следующий объект своего базового типа. После выполнения операции уменьшения - на предыдущий объект. Применительно к указателям на ***char***, операций адресной арифметики выполняются как обычные арифметические операции, потому что длина объекта ***char*** всегда равна 1. Для всех указателей адрес увеличивается или уменьшается на величину, равную размеру объекта того типа, на который они указывают. Поэтому указатель всегда ссылается на объект с типом, тождественным базовому типу указателя. Эта концепция иллюстрируется с помощью рисунка ниже.

Рисунок - Пример размещения в памяти переменных ***char*** (слева) и ***int*** (справа)

```
char *ch = (char *) 1000;
int *i = (int *) 1000;
```




```
#include <sat500.h>
void main(void)
{
    char *ch = (char *) 1000;
    int *i = (int *) 1000;
    ClrScr();
    BigFont=0;
    GotoXY(0,4);
    for(int k=0;k<8;k++)
    {
        printf("Значение ch равно: %d\n",ch);
        ch++;
    }
    for(int k=0;k<4;k++)
    {
        printf("Значение i равно: %d\n",i);
        i++;
    }
    ShowScr();
    while(1);
}
```

Результат:

Операции адресной арифметики не ограничены увеличением (инкрементом) и уменьшением (декрементом). Например, к указателям можно добавлять целые числа или вычитать из них целые числа. Выполнение оператора

```
p1 = p1 + 12;
```

"передвигает" указатель p1 на 12 объектов в сторону увеличения адресов.

Кроме суммирования и вычитания указателя и целого, разрешена еще только одна операция адресной арифметики: *можно вычитать два указателя*. Благодаря этому можно определить количество объектов, расположенных между адресами, на которые указывают данные два указателя; правда, при этом считается, что тип объектов совпадает с базовым типом указателей. Все остальные арифметические операции запрещены. А именно: *нельзя делить и умножать указатели*, *суммировать два указателя*, *выполнять над указателями побитовые операции*.

Сравнение указателей

Стандартом С допускается сравнение двух указателей. Например, если объявлены два указателя p и q, то следующий оператор является правильным:

```
if(p < q) printf("p ссылается на меньший адрес, чем q\n");
```

Как правило, сравнение указателей может оказаться полезным, только тогда, когда два указателя ссылаются на общий объект, например, на массив.

2.13.5. УКАЗАТЕЛИ И МАССИВЫ

Понятия указателей и массивов тесно связаны. Рассмотрим следующий фрагмент программы:

```
char str[80], * p1;
p1 = str;
```

Здесь ***p1*** указывает на первый элемент массива ***str***. Обратиться к пятому элементу массива ***str*** можно с помощью любого из двух выражений:

```
str[4]
* (p1+4)
```

Массив начинается с нуля. Поэтому для пятого элемента массива ***str*** нужно использовать индекс 4. Можно также увеличить ***p1*** на 4, тогда он будет указывать на пятый элемент. (Напомним, что имя массива без индекса возвращает адрес первого элемента массива.)
В языке С существуют два метода обращения к элементу массива: адресная арифметика и индексация массива. Стандартная запись массивов с индексами наглядна и удобна в использовании, однако с помощью адресной арифметики иногда удается сократить время доступа к элементам массива. Поэтому адресная арифметика часто используется в программах, где существенную роль играет быстродействие.
В следующем фрагменте программы приведены две версии функции *putstr()*, выводящей строку на экран. В первой версии используется индексация массива, а во второй — адресная арифметика:

```
/* Индексация указателя s как массива. */
void putstr(char *s)
{
    int t;
    for(t=0; s[t]; ++t) putchar(s[t]);
}

/* Использование адресной арифметики. */
void putstr(char *s)
{
    while(*s) putchar(*s++);
}
```

Большинство профессиональных программистов сочтут вторую версию более наглядной и удобной. Для большинства компиляторов она также более быстродействующая. Поэтому в процедурах такого типа приемы адресной арифметики используются довольно часто.

Массивы указателей

Как и объекты любых других типов, указатели могут быть собраны в массив. В следующем операторе объявлен массив из 10 указателей на объекты типа ***int***:

```
int *x[10];
```

Для присвоения, например, адреса переменной ***var*** третьему элементу массива указателей, необходимо написать:

```
x[2] = &var;
```

В результате этой операции, следующее выражение принимает то же значение, что и var:

```
*x[2]
```

Для передачи массива указателей в функцию используется тот же метод, что и для любого другого массива: имя массива без индекса записывается как формальный параметр функции. Например, следующая функция может принять массив x в качестве аргумента:

```
void display_array(int *q[])
{
    int t;

    for(t=0; t<10; t++)
        printf("%d ",*q[t]);
}
```

Необходимо помнить, что ***q*** - это не указатель на целые, а указатель на массив указателей на целые. Поэтому параметр ***q*** нужно объявить как массив указателей на целые. Нельзя объявить q просто как указатель на целые, потому что он представляет собой указатель на указатель.
Массивы указателей часто используются при работе со строками. Например, можно написать функцию, выводящую нужную строку с сообщением об ошибке по индексу ***num***:

```
#include <sat500.h>

char *err[] =
{
    "Нельзя открыть\n",
    "Ошибка при чтении\n",
    "Ошибка при записи\n",
    "Некачественный носитель\n"
};

void syntax_error(int num)
{
    printf("%s", err[num]);
}

void main(void)
{
    ClrScr();
    syntax_error(2);
    ShowScr();
    while(1);
}
```

Результат:

Массив ***err*** содержит указатели на строки с сообщениями об ошибках. Здесь строковые константы в выражении инициализации создают указатели на строки. Аргументом функции ***printf()*** служит один из указателей массива ***err***, который в соответствии с индексом ***num*** указывает на нужную строку с сообщением об ошибке. Например, если в функцию ***syntax_error()*** передается ***num*** со значением 2, то выводится сообщение *Ошибка при записи*.
Массив указателей *err* будет располагаться в RAM, а вот строки на которые ссылается массив будут помещены в *const* память (FLASH).
Чтобы и массив err располагался в *const* памяти следует определить его с модификатором *const*:

```
const char *err[] =
{
    "Нельзя открыть\n",
    "Ошибка при чтении\n",
    "Ошибка при записи\n",
    "Некачественный носитель\n"
};
```

2.13.6. ИНИЦИАЛИЗАЦИЯ УКАЗАТЕЛЕЙ

После объявления нестатического локального указателя до первого присвоения он содержит неопределенное значение. (Глобальные и статические локальные указатели при объявлении неявно инициализируются нулем.) Если попытаться использовать указатель перед присвоением ему нужного значения, то скорее всего он мгновенно разрушит программу. Это очень досадная ошибка.
При работе с указателями большинство программистов придерживаются следующего важного соглашения: указатель, не ссылающийся в текущий момент времени должным образом на конкретный объект, должен содержать нулевое значение. Нуль используется потому, что С гарантирует отсутствие чего-либо по нулевому адресу. Следовательно, если указатель равен нулю, то это значит, во-первых, что он ни на что не ссылается, а во-вторых — что его сейчас нельзя использовать.
Указателю можно задать нулевое значение, присвоив ему 0. Например, следующий оператор инициализирует ***p*** нулем:

```
char *p = 0;
```

Дополнительно к этому во многих заголовочных файлах языка С, определен макрос NULL, являющийся нулевой указательной константой. Поэтому в программах на С часто можно увидеть следующее присваивание:

```
p = NULL;
```

Однако равенство указателя нулю не делает его абсолютно "безопасным". Использование нуля в качестве признака неподготовленности указателя - это только соглашение программистов, но не правило языка С.
Во многих процедурах для повышения эффективности программы можно использовать то, что нулевой указатель заведомо считается неподготовленным для использования. Например, можно использовать нулевой указатель как признак конца массива указателей (по аналогии с нулевым терминатором строки). Процедура, использующая массив указателей, таким образом узнает о конце массива.

2.13.7. ТРУДНОСТИ ПРИ РАБОТЕ С УКАЗАТЕЛЯМИ

Ничто не может доставить больше неприятностей, чем "дикий" указатель! Указатели похожи на обоюдоострое оружие: их возможности огромны, однако обезвредить ошибки в них особенно трудно.

Ошибочный указатель трудно найти потому, что ошибка в самом указателе никак себя не проявляет. Проблемы возникают при попытке обратиться к объекту с помощью этого указателя. Если значение указателя неправильное, то программа с его помощью обращается к произвольной ячейке памяти. При чтении в программу попадают неправильные данные, а при записи искажаются другие данные, хранящиеся в памяти, или портится участок программы, не имеющий никакого отношения к ошибочному указателю. В обоих случаях ошибка может не проявиться вовсе или проявиться позже в форме, никак не указывающей на ее причину.
Поскольку ошибки, связанные с указателями, особенно трудно обезвредить, при работе с указателями следует соблюдать особую осторожность. Рассмотрим некоторые ошибки, наиболее часто возникающие при работе с указателями. Классический пример - неинициализированный указатель:

```
/* Эта программа содержит ошибку. */
void main(void)
{
    int x, *p;
    x = 10;
    *p = x; /* ошибка, p не инициализирован */
    ...
}
```

Эта программа присваивает значение 10 некоторой неизвестной области памяти. Рассмотрим, почему это происходит. Хотя указателю ***p*** не было присвоено никакого значения, но в момент выполнения операции ****p = x*** он имел некоторое (совершенно произвольное!) значение. Поэтому здесь имела место попытка выполнить операцию записи в область памяти, на которую указывал данный указатель. В небольших программах такая ошибка часто остается незамеченной, потому что если программа и данные занимают немного места, то "выстрел наугад" скорее всего будет "промахом". С увеличением размера программы вероятность "попасть" в нее возрастает.
Вторая распространенная ошибка заключается в простом недоразумении при использовании указателя:

```
/* Эта программа содержит ошибку. */

#include <sat500.h>

int main(void)
{
    int x, *p;
    x = 10;
    p = x;
    printf("%d", *p);
    ...
}
```

Вызов *printf()* не выводит на экран значение x, равное 10. Выводится произвольная величина, потому что оператор

```
p = x;
```

записан неправильно. Он присваивает значение 10 указателю, однако указатель должен содержать адрес, а не значение. Правильный оператор выглядит так:

```
p = &x;
```

Ещё одна типичная ошибка происходит иногда при неправильном понимании принципов расположения переменных в памяти. Программисту ничего не известно о том, как используемые им данные располагаются в памяти, будут ли они расположены так же при следующем выполнении программы или как их расположат другие компиляторы. Поэтому сравнивать одни указатели с другими недопустимо. Например, программа

```
char s[80], y[80];
char *p1, *p2;

p1 = s;
p2 = y;
if(p1 < p2) . . .
```

в общем случае неправильна. (В некоторых необычных ситуациях иногда определяют относительное положение переменных, но это делают очень редко.)
Похожая ошибка возникает, когда делается необоснованное предположение о расположении массивов. Иногда, предполагая, что массивы расположены рядом, пытаются обращаться к ним с помощью одного и того же указателя, например:

```
int first[10], second[10];
int *p, t;

p = first;
for(t=0; t<20; ++t)  *p++ = t;
```

Так присваивать значения массивам first и second нельзя. Если компилятор разместит массивы рядом, это может и не привести к неправильному результату. Однако подобная ошибка особенно неприятна тем, что при проверке она может остаться незамеченной, а потом компилятор будет размещать массивы по-другому и программа выполнится неправильно.

2.14. ФУНКЦИИ

В общем виде функция выглядит следующим образом:

```
возвращаемый-тип имя-функции(список параметров)
{
  тело функции
}
```

Возвращаемый-тип определяет тип данного, возвращаемого функцией.
Данное, возвращаемое функцией, называется также результатом. Соответственно, возвращаемый тип часто называется также типом результата.
Функция может возвращать любой тип данных, за исключением массивов.
Список параметров - это список, элементы которого отделяются друг от друга запятыми. Каждый такой элемент состоит из имени переменной и ее типа данных. При вызове функции параметры принимают значения аргументов. Функция может быть и без параметров, тогда их список будет пустым. Такой пустой список можно указать в явном виде, поместив для этого внутри скобок ключевое слово *void*.
В объявлениях (декларациях) переменных можно объявить (декларировать) несколько переменных одного и того же типа, используя для этого список одних только имен, элементы которого отделены друг от друга запятыми. А все параметры функций, наоборот, должны объявляться отдельно, причем для каждого из них надо указывать и тип, и имя. То есть в общем виде список объявлений параметров должен выглядеть следующим образом:

```
f(тип имя_переменной1, тип имя_переменной2,..., тип имя_переменнойN)
```

Вот, например, два объявления параметров функций, первое из которых правильное, а второе - нет:

```
f(int i, int k, int j) /* правильное */
```

```
f(int i, k, float j)   /* неправильное, у переменной k должен быть собственный спецификатор типа */
```

2.14.1. ОБЛАСТЬ ДЕЙСТВИЯ ФУНКЦИИ

В языке правила работы с областями действия - это правила, которые определяют, известен ли фрагменту кода другой фрагмент кода или данных, или имеет ли он доступ к этому другому фрагменту.
Здесь мы более подробно рассмотрим одну специальную область действия - ту, которая определяется функцией.
Каждая функция представляет собой конечный блок кода. Таким образом, она определяет область действия этого блока. Это значит, что код функции является закрытым и недоступным ни для какого выражения из любой другой функции, если только не выполняется вызов содержащей его функции. (Например, нельзя перейти в середину другой функции с помощью *goto*.)
Код, который составляет тело функции, скрыт от остальной части программы, и если он не использует глобальных переменных, то не может воздействовать на другие части программы или, наоборот, подвергаться воздействию с их стороны. Иначе говоря, код и данные, определенные внутри одной функции, без глобальных переменных не могут воздействовать на код и данные внутри другой функции, так как у любых двух разных функций разные области действия.
Переменные, определенные внутри функции, являются локальными. Локальная переменная создается в начале выполнения функции, а при выходе из этой функции она уничтожается. Таким образом, локальная переменная не может сохранять свое значение в промежутках между вызовами функции.
Формальные параметры функции также находятся в ее области действия. Это значит, что параметр доступен внутри всей функции. Параметр создается в начале выполнения функции, и уничтожается при выходе из нее.
Все функции имеют файл в качестве области действия (file scope). Таким образом, функцию нельзя определять внутри другой функции. Поэтому С практически не является языком с блочной структурой.

2.14.2. АРГУМЕНТЫ ФУНКЦИИ

Если функция должна принимать аргументы, то в ее объявлении следует декларировать параметры, которые примут значения этих аргументов. Как видно из объявления следующей функции, объявления параметров стоят после имени функции.

```
/* Возвращает 1, если символ с входит в строку s;
   и 0 в противном случае. */
int is_in(char *s,  char c)
{
    while(*s)
        if(*s==c) return 1;
        else s++;
    return 0;
}
```

Функция *is_in()* имеет два параметра: ***s*** и ***d***. Если символ с входит в строку ***s***, то эта функция возвращает 1, в противном случае она возвращает 0.
Хотя параметры выполняют специальную задачу, — принимают значения аргументов, передаваемых функции, — они все равно ведут себя так, как и другие локальные переменные. Формальным параметрам функции, например, можно присваивать какие-либо значения или использовать эти параметры в каких-либо выражениях.

Вызовы по значению и по ссылке

В языках программирования имеется два способа передачи значений подпрограмме. Первый из них — вызов по значению. При его применении в формальный параметр подпрограммы копируется значение аргумента. В таком случае изменения параметра на аргумент не влияют.
Вторым способом передачи аргументов подпрограмме является вызов по ссылке. При его применении в параметр копируется адрес аргумента. Это значит, что, в отличие от вызова по значению, изменения значения параметра приводят к точно таким же изменениям значения аргумента.
За небольшим количеством исключений, в языке С для передачи аргументов используется вызов по значению. Обычно это означает, что код, находящийся внутри функции, не может изменять значений аргументов, которые использовались при вызове функции.

```
#include <sat500.h>

int sqr(int x);

void main(void)
{
    int t=10;
    ClrScr();
    printf("%d %d", sqr(t), t);
    ShowScr();
    while(1);
}

int sqr(int x)
{
    x = x*x;
    return x;
}
```

В этом примере в параметр ***x*** копируется 10 - значение аргумента для *sqr().* Когда выполняется присваивание *x=x*x,* модифицируется только локальная переменная x. А значение переменной ***t***, использованной в качестве аргумента при вызове *sqr()*, по-прежнему остаётся равным 10. Поэтому выведено будет следующее: 100.10.
Помните, что именно копия значения аргумента передаётся в функцию. А то, что происходит внутри функции, не влияет на значение переменной, которая была использована при вызове в качестве аргумента.

Вызов по ссылке

Хотя в С для передачи параметров применяется вызов по значению, можно создать вызов и по ссылке, передавая не сам аргумент, а указатель на него. Так как функции передаётся адрес аргумента, то её внутренний код в состоянии изменить значение этого аргумента, находящегося, между прочим, за пределами самой функции.
Указатель передаётся функции так, как и любой другой аргумент. Конечно, в таком случае параметр следует декларировать как один из типов указателей. Это можно увидеть на примере функции swap(), которая меняет местами значения двух целых переменных, на которые указывают аргументы этой функции:

```
void swap(int *x, int *y)
{
    int temp;

    temp = *x;  /* сохранить значение по адресу x */
    *x = *y;    /* поместить y в x */
    *y = temp;  /* поместить x в y */
}
```

Функция swap() может выполнять обмен значениями двух переменных, на которые указывают x и y, потому что передаются их адреса, а не значения. Внутри функции, используя стандартные операции с указателями, можно получить доступ к содержимому переменных и провести обмен их значений.
Помните, что swap() (или любую другую функцию, в которой используются параметры в виде указателей) необходимо вызывать вместе с адресами аргументов. Следующая программа показывает, как надо правильно вызывать swap():

```
#include <sat500.h>

void swap(int *x, int *y);

void main(void)
{
    int i, j;
    ClrScr();
    i = 10;
    j = 20;
    printf("i и j перед обменом: %d %d\n", i, j);
    swap(&i, &j); /* передать адреса переменных i и j */
    printf("i и j после обмена: %d %d\n", i, j);
    ShowScr();
    while(1);
}

void swap(int *x, int *y)
{
    int temp;

    temp = *x;  /* сохранить значение по адресу x */
    *x = *y;    /* поместить y в x */
    *y = temp;  /* поместить x в y */
}
```

И вот что вывела эта программа:
i и j перед обменом: 10 20
i и j после обмена: 20 10
В программе переменной ***i*** присваивается значение 10, а переменной ***j*** — значение 20. Затем вызывается функция *swap()* с адресами этих переменных. (Для получения адреса каждой из переменных используется унарный оператор &.) Поэтому в *swap()* передаются адреса переменных ***i*** и ***j***, а не их значения.

Вызов функций с помощью массивов

В настоящем же разделе рассказывается о передаче массивов функциям в качестве аргументов. Этот вопрос рассматривается потому, что эта операция является исключением по отношению к обычной передаче параметров, выполняемой путем вызова по значению.
Когда в качестве аргумента функции используется массив, то функции передаётся его адрес. В этом и состоит исключение по отношению к правилу, которое гласит, что при передаче параметров используется вызов по значению. В случае передачи массива функции её внутренний код работает с реальным содержимым этого массива и вполне может изменить это содержимое. Проанализируйте, например, функцию *print_upper()*, которая печатает свой строковый аргумент на верхнем регистре:

```
#include <sat500.h>
#include <ctype.h>

void print_upper(char *string);
```

```
char s[]="This is a test.";

void main(void)
{
    ClrScr();
    print_upper(s);
    printf("\ns теперь на верхнем регистре: %s", s);
    ShowScr();
    while(1);
}

/* Печатать строку на верхнем регистре. */
void print_upper(char *string)
{
    int t;

    for(t=0; string[t]; ++t)
    {
        string[t] = toupper(string[t]);
        putchar(string[t]);
    }
}
```

Вот что будет выведено в случае фразы "This is a test." (это тест):

```
THIS IS A TEST.
s теперь в верхнем регистре: THIS IS A TEST.
```

После вызова *print_upper()* содержимое массива ***s*** в *main()* переводится в символы верхнего регистра. Если вам это не нужно, программу можно написать следующим образом:

```
#include <sat500.h>
#include <ctype.h>

void print_upper(char *string);

char s[]="This is a test.";

void main(void)
{
    ClrScr();
    print_upper(s);
    printf("\ns не изменялась: %s", s);
    ShowScr();
    while(1);
}

void print_upper(char *string)
{
    int t;
    for(t=0; string[t]; ++t)
        putchar(toupper(string[t]));
}
```

Вот какой на этот раз получится фраза "This is a test.":

```
THIS IS A TEST.
s не изменилась: This is a test.
```

На этот раз содержимое массива не изменилось, потому что внутри *print_upper()* не изменялись его значения.

2.14.3. ОПЕРАТОР RETURN

Оператор *return* имеет два важных применения. Во-первых, он обеспечивает немедленный выход из функции, т.е. заставляет выполняющуюся программу передать управление коду, вызвавшему функцию. Во-вторых, этот оператор можно использовать для того, чтобы возвратить значение. В следующих разделах рассказывается, каким именно образом можно использовать оператор *return*.

Возврат из функции

Функция может завершать выполнение и осуществлять возврат в вызывающую программу двумя способами. Первый способ используется тогда, когда после выполнения последнего оператора в функции встречается закрывающая фигурная скобка (}). (Конечно, это просто жаргон, ведь в настоящем объектном коде фигурной скобки нет!) Например, функция *pr_reverse()* в приведенной ниже программе просто выводит на экран в обратном порядке строку Мне нравится С, а затем возвращает управление вызывающей программе.

```
#include <sat500.h>
#include <string.h>

void pr_reverse(char *s);

void main(void)
{
    ClrScr();
    pr_reverse("Мне нравится С");
    ShowScr();
    while(1);
}

void pr_reverse(char *s)
{
    int t;
    for(t=strlen(s)-1; t>=0; t--) putchar(s[t]);
}
```

Как только строка выведена на экран, функции *pr_reverse()* "делать больше нечего", поэтому она возвращает управление туда, откуда она была вызвана.
Но на практике не так уж много функций используют именно такой способ завершения выполнения. В большинстве функций для завершения выполнения используется оператор *return* - или потому, что необходимо вернуть значение, или чтобы сделать код функции проще и эффективнее.
В функции может быть несколько операторов return. Например, в следующей программе функция *find_substr()* возвращает начальную позицию подстроки в строке или же возвращает -1, если подстрока, наоборот, не найдена. В этой функции для упрощения кодирования используются два оператора *return*.

```
#include <sat500.h>

int find_substr(char *s1, char *s2);

void main(void)
{
    ClrScr();
    if(find_substr("С - это забавно", "is") != -1)
        printf("Подстрока найдена.");
    ShowScr();
    while(1);
```

```
}

/* Вернуть позицию первого, вхождения s2 в s1. */
int find_substr(char *s1, char *s2)
{
    int t;
    char *p, *p2;

    for(t=0; s1[t]; t++)
    {
        p = &s1[t];
        p2 = s2;

        while(*p2 && *p2==*p)
        {
              p++;
              p2++;
        }
        if(!*p2) return t; /* 1-й оператор return */
    }
   return -1; /* 2-й оператор return */
}
```

Возврат значений

Все функции, кроме тех, которые относятся к типу void, возвращают значение. Это значение указывается выражением в операторе return. В функции, тип которой отличен от void, в операторе return необходимо обязательно указать возвращаемое значение. То есть, для какой-либо функции указано, что она возвращает значение, то внутри этой функции у любого оператора return должно быть свое выражение.
Если функция не объявлена как имеющая тип void, она может использоваться как операнд в выражении. Поэтому каждое из следующих выражений является правильным:

```
x = power(y);
if(max(x,y) > 100) printf("больше");
for(ch=getchar(); isdigit(ch); ) ... ;
```

Общепринятое правило гласит, что вызов функции не может находиться в левой части оператора присваивания. Выражение

```
swap(x,y) = 100; /* неправильное выражение */
```

является неправильным. Если компилятор С в какой-либо программе найдет такое выражение, то пометит его как ошибочное и программу компилировать не будет.
В программе можно использовать функции трех видов.
Первый вид — простые вычисления. Эти функции предназначены для выполнения операций над своими аргументами и возвращают полученное в результате этих операций значение. Вычислительная функция является функцией "в чистом виде".
Второй вид включает в себя функции, которые обрабатывают информацию и возвращают значение, которое показывает, успешно ли была выполнена эта обработка.
У функций последнего, третьего вида нет явно возвращаемых значений. В сущности, такие функции являются чисто процедурными и никаких значений выдавать не должны.
Все функции, которые не возвращают значение, должны объявляться как возвращающие значение типа *void*. Объявляя функцию как возвращающую значение типа *void*, вы запрещаете ее применение в выражениях, предотвращая таким образом случайное использование этой функции не по назначению.
Иногда функции, которые, казалось бы, фактически не выдают содержательный результат, все же возвращают какое-то значение. Например, *printf()* возвращает количество выведенных символов. Если бы нашлась такая программа, которая на самом деле проверяла бы это значение, то это

было бы что-то необычное... Другими словами, хотя все функции, за исключением относящихся к типу *void*, возвращают значения, вовсе не нужно стремиться использовать эти значения во что бы то ни стало. Часто при обсуждении значений, возвращаемых функциями, возникает такой довольно распространенный вопрос: "Неужели не обязательно присваивать возвращенное значение какой-либо переменной? Не повиснет ли оно где-нибудь и не приведет ли это в дальнейшем к каким-либо неприятностям?" Отвечая на этот вопрос, повторим, что присваивание отнюдь не является обязательным, причем отсутствие его не станет причиной каких-либо неприятностей. Если возвращаемое значение не входит ни в один из операторов присваивания, то это значение будет просто отброшено. Отметим также, что такое отбрасывание значения встречается очень часто. Проанализируйте следующую программу, в которой используется функция *mul()*:

```
#include <sat500.h>

int mul(int a, int b);

int main(void)
{
    int x, y, z;
    ClrScr();
    x = 10;   y = 20;
    z = mul(x, y);           /* 1 */
    printf("%d", mul(x,y));  /* 2 */
    mul(x, y);               /* 3 */
    ShowScr();
    while(1);
}

int mul(int a, int b)
{
    return a*b;
}
```

В строке 1 значение, возвращаемое функцией *mul()*, присваивается переменной **z**. В строке 2 возвращаемое значение не присваивается, но используется функцией *printf()*. И наконец, в строке 3 возвращаемое значение теряется, потому что не присваивается никакой из переменных и не используется как часть какого-либо выражения.

Возвращаемые указатели

Хотя с функциями, которые возвращают указатели, обращаются так же, как и с любыми другими функциями, все же будет полезно познакомиться с некоторыми основными понятиями и рассмотреть соответствующий пример. Указатели не являются ни целыми, ни целыми без знака. Они являются адресами в памяти и относятся к особому типу данных. Такая особенность указателей определяется тем, что арифметика указателей (адресная арифметика) работает с учетом параметров базового типа. Например, если указателю на целое придать минимальное (ненулевое) приращение, то его текущее значение станет на четыре больше, чем предыдущее (при условии, что целые значения занимают 4 байта). Вообще говоря, каждый раз, когда значение указателя увеличивается (уменьшается) на минимальную величину, то он указывает на последующий (предыдущий) элемент, имеющий базовый тип указателя. Так как размеры разных типов данных могут быть разными, то компилятор должен знать тип данных, на которые может указывать указатель. Поэтому в объявлении функции, которая возвращает указатель, тип возвращаемого указателя должен декларироваться явно. Например, нельзя объявлять возвращаемый тип как *int* *, если возвращается указатель типа *char* *! Иногда (правда, крайне редко!) требуется, чтобы функция возвращала "универсальный" указатель, т.е. указатель, который может указывать на данные любого типа. Тогда тип результата функции следует определить как *void* *.

Чтобы функция могла возвратить указатель, она должна быть объявлена как возвращающая указатель на нужный тип. Например, следующая функция возвращает указатель на первое вхождение символа, присвоенного переменной ***c***, в строку ***s***. Если этого символа в строке нет, то возвращается указатель на символ конца строки ('0').

```
/* Возвращает указатель на первое вхождение c в s. */
char *match(char c, char *s)
{
    while(c!=*s && *s) s++;
    return(s);
}
```

Вот небольшая программа, в которой используется функция match():

```
#include <sat500.h>

char *match(char c, char *s);  /* прототип */

char s[]="Это строка";

int main(void)
{
    char *p,ch;
    ClrScr();
    ch = 'c';
    p = match(ch, s);
    if(*p)  /* символ найден */
        printf("%s ", p);
    else
        printf("Символа нет.");
    ShowScr();
    while(1);
}
```

Эта программа проводит поиск местонахождения символа в строке. При наличии символа в строке переменная p укажет на него, и программа выведет строку, начиная с найденного символа. Если символ в строке не найден, то p укажет на символ конца строки ('0'), причем *p будет представлять логическое значение ЛОЖЬ (false). В таком случае программа выведет сообщение Символа нет.

Функция типа void

Одним из применений ключевого слова ***void*** является явное объявление функций, которые не возвращают значений. Мы уже знаем, что такие функции не могут применяться в выражениях, и указание ключевого слова ***void*** предотвращает их случайное использование не по назначению. Например, функция *print_vertical()* выводит в боковой части экрана свой строчный аргумент по вертикали сверху вниз.

```
void print_vertical(char *str)
{
      while(*str)printf("%c\n", *str++);
}
```

Вот пример использования функции print_vertical():

```
#include <sat500.h>

void print_vertical(char *str);  /* прототип */

void main(void)
{
    ClrScr();
    print_vertical("Привет");
    ShowScr();
```

```
    while(1);
}

void print_vertical(char *str)
{
    while(*str)printf("%c\n", *str++);
}
```

2.14.4. РЕКУРСИЯ

В языке С функция может вызывать сама себя. В этом случае такая функция называется *рекурсивной*. *Рекурсия* - это процесс определения чего-либо на основе самого себя, из-за чего рекурсию еще называют *рекурсивным определением*.
Простым примером рекурсивной функции является factr(), которая вычисляет факториал целого неотрицательного числа. Факториалом числа n (обозначается n!) называется произведение всех целых чисел, от 1 до n включительно (для 0, по определению, факториал равен 1.). Например, 3! — это 1×2×3, или 6. Здесь показаны *factr()* и эквивалентная ей функция, в которой используется итерация:

```
/* рекурсивная функция */
int factr(int n)
{
    int answer;
    if(n==1) return(1);
    answer = factr(n-1)*n; /* рекурсивный вызов */
    return(answer);
}

/* неркурсивная функция */
int fact(int n)
{
    int t, answer;
    answer = 1;
    for(t=1; t<=n; t++)answer=answer*(t);
        return(answer);
}
```

Нерекурсивное вычисление факториала, то есть вычисление с помощью *fact()*, выполняется достаточно просто. В этой функции в теле цикла, выполняющемся для t от 1 до n, вычисленное ранее произведение последовательно умножается на каждое из этих чисел. (Значение факториала для 0 получается, конечно, с помощью оператора присваивания. Значение факториала для 1 также получается умножением не на ранее полученное произведение, а на заранее подготовленное число, тоже равное 1.)
Работа же рекурсивной функции *factr()* чуть более сложная. Когда *factr()* вызывается с аргументом 0, то она сразу возвращает 1. Если же аргумент больше 0, то возвращается произведение *factr(n-1)*n*. Чтобы вычислить значение этого выражения, factr() вызывается с аргументом n-1. Это выполняется до тех пор, пока n не станет равным 0. Когда это произойдет, вызовы функции начнут возвращать вычисленные ими значения факториалов.
При вычислении 2! первый вызов *factr()* влечет за собой второй, теперь уже рекурсивный вызов с аргументом 1, который, в свою очередь, влечет третий, тоже рекурсивный вызов с аргументом 0. Этот вызов возвращает число 1, которое затем умножается на 1, а потом на 2 (первоначальное значение n). Ответ в данном случае равен 2. Попробуйте самостоятельно вычислить 3!. (Вам, возможно, захочется вставить в функцию *factr()* выражения printf(), чтобы видеть уровень каждого вывода, и то, какие будут промежуточные ответы.)
Когда функция вызывает сама себя, новый набор локальных переменных и параметров размещается в памяти в стеке, а код функции выполняется с самого своего начала, причем используются

именно эти новые переменные. При рекурсивном вызове функции новая копия ее кода не создается. Новыми являются только значения, которые использует данная функция. При каждом возвращении из рекурсивного вызова старые локальные переменные и параметры извлекаются из стека, и сразу за рекурсивным вызовом возобновляется работа функции. При использовании рекурсивных функций стек работает подобно "телескопической" трубе, выдвигающейся вперед и складывающейся обратно.
Хотя и кажется, что рекурсия предлагает более высокую эффективность, но на самом деле такое бывает достаточно редко. Использование рекурсии в программах зачастую не очень сильно уменьшают их размер кода и обычно только незначительно увеличивает эффективность использования памяти. Кроме того, рекурсивные версии большинства программ могут выполняться несколько медленнее, чем их итеративные варианты, потому что при рекурсивных вызовах функций расходуются дополнительные ресурсы. Кроме того, большое количество рекурсивных вызовов функции может вызвать переполнение стека. Из-за того, что память для параметров функции и локальных переменных находится в стеке и при каждом новом вызове создается еще один набор этих переменных, то для переменных места в стеке может рано или поздно не хватить. Переполнение стека — вот обычная причина аварийного завершения программы, когда функция утрачивает контроль над рекурсивными обращениями.
Главным преимуществом рекурсивных функций является то, что с их помощью упрощается реализация некоторых алгоритмов, а программа становится понятнее.
В тексте рекурсивной функции обязательно должен быть выполнен условный оператор, например if, который при определенных условиях вызовет завершение функции, т.е. возврат, а не выполнит очередной рекурсивный вызов. Если такого оператора нет, то после вызова функция никогда не сможет завершить работы. Распространенной ошибкой при написании рекурсивных функций как раз и является отсутствие в них условного оператора. При создании программ не отказывайтесь от функции *printf(),* тогда вы сможете увидеть, что происходит на самом деле и сможете прервать выполнение, когда обнаружите ошибку.

2.14.5. ПРОТОТИП ФУНКЦИИ

В современных, правильно написанных программах на языке С каждую функцию перед использованием необходимо объявлять. Обычно это делается с помощью прототипа функции.
Хотя прототипы формально не требуются, но их использование очень желательно.Прототипы дают компилятору возможность тщательнее выполнять проверку типов, подобно тому, как это делается в таких языках как Pascal. Если используются прототипы, то компилятор может обнаружить любые сомнительные преобразования типов аргументов, необходимые при вызове функции, если тип ее параметров отличается от типов аргументов. При этом будут выданы предупреждения обо всех таких сомнительных преобразованиях. Компилятор также обнаружит различия в количестве аргументов, использованных при вызове функции, и в количестве параметров функции.
В общем виде прототип функции должен выглядеть таким образом:

```
тип имя_функции(тип имя_парам1, тип имя_парам2, ..., имя_парамN);
```

Имена параметров дают возможность компилятору при наличии ошибки указать имена.
Следующая программа показывает, насколько ценными являются прототипы функций. В ней выводится сообщение об ошибке, происходящей из-за того, что программа содержит попытку вызова sqr_it() с целым аргументом, в то время как требуется указатель на целое.

```
/* В этой программе используется прототип функции
   чтобы обеспечить тщательную проверку типов. */

void sqr_it(int *i); /* прототип */

int main(void)
{
    int x;
    x = 10;
```

```
    sqr_it(x);  /* несоответствие типов */
    return 0;
}

void sqr_it(int *i)
{
    *i = *i * *i;
}
```

В качестве прототипа функции может также служить её определение, если оно находится в программе до первого вызова этой функции. Вот, например, правильная программа:

```
#include <sat500.h>
/* Это определение будет также служить и
   прототипом внутри этой программы. */
void f(int a, int b)
{
    printf("%d ", a % b);
}

void main(void)
{
    f(10,3);
    ...
}
```

В этом примере специальный прототип не требуется; так как функция *f()* определена ещё до того, как она начинает использоваться в *main()*. Хотя определение функции и может служить её прототипом в малых программах, но в больших такое встречается редко - особенно, когда используется несколько файлов.
Единственная функция, для которой не требуется прототип - это *main()*, так как это первая функция, вызываемая в начале работы программы.
Если функция в языке С не имеет параметров, то в её прототипе внутри списка параметров стоит только ключевое слово ***void***. Вот, например, прототип функции ***f()*** в том виде, в каком он должен быть в программе на языке С:

```
int f(void);
```

Таким образом компилятор узнает, что у функции нет параметров, и любое обращение к ней, в котором имеются аргументы, будет считаться ошибкой. В С++ использование ключевого слова ***void*** внутри пустого списка параметров также разрешено, но считается излишним.
Прототипы функций позволяют "отлавливать" ошибки ещё до запуска программы. Кроме того, они запрещают вызов функций при несовпадении типов (т.е. с неподходящими аргументами) и тем самым помогают проверять правильность программы.

2.14.6. ОБЪЯВЛЕНИЕ СПИСКОВ ПАРАМЕТРОВ ПЕРЕМЕННОЙ ДЛИНЫ

Можно вызвать функцию, которая имеет переменное количество параметров. Самым известным примером является *printf()*. Чтобы сообщить компилятору, что функции будет передано заранее неизвестное количество аргументов, объявление списка ее параметров необходимо закончить многоточием. Например, следующий прототип указывает, что у функции *func()* будет как минимум два целых параметра и после них еще некоторое количество (в том числе и 0) параметров:

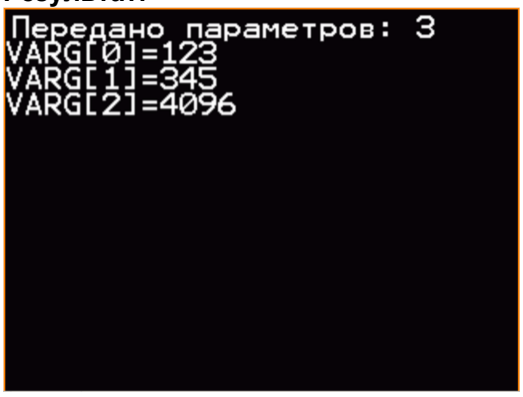
```
int func(int a, int b, ...);
```

Все параметры, заданные вместо многоточия через запятую будут помещены в С-стек программы как целочисленные переменные независимо от их типа.
Доступ к переданным параметрам выполняется через специальный набор переменных с именем _VARG[].
Для доступа к нулевому переданному параметру следует использовать имя _VARG[0].
Для доступа к i-му переданному параметру следует использовать имя _VARG[i].
Внимание! Действительное количество переданных параметров и то как к ним происходит доступ к функции никак не контролируются.
Один из вариантов контроля количества параметров - передать в качестве первого обязательного параметра количество дополнительных параметров.
В примере ниже объявлена функция Func, у которой имеется один параметр, и задан список параметров переменной длины.
Первый параметр предполагает что в нём будет задано количество параметров помещённых в списке переменной длины.
В самой функции все переданные параметры выводятся на экран.
Вызов функции содержит 3 параметра, один из которых является адресом текстовой строки в ROM памяти контроллера (Text).

Пример:

```
#include <sat500.h>

//Текстовая строка в ROM памяти
const char Text[]="Это текст";

//cnt - количество действительно переданных параметров в С-стек
void Func(int cnt,...)
{
    printf("Передано параметров: %d\r\n",cnt);
    for(int i=0;i<cnt;i++)
    {
        printf("VARG[%d]=%d\r\n",i,_VARG[i]);
    }
}
void main(void)
{
    BrushColor=_COLOR_BLACK;     //Цвет кисти - чёрный
    BigFont=0;                   //Обычный шрифт
    ClrScr();                    //Очистить экран чёрным цветом
    Func(3,123,345,Text);        //Передача трёх параметров
    ShowScr();                   //Вывести нарисованное не дисплей
    while(1);                    //Бесконечная пауза
}
```

Результат:



2.15. ПРЕПРОЦЕССОР

Имеются следующие директивы препроцессора:

```
#define     #endif     #ifdef       #line
#elif       #error     #ifndef      #pragma
#else       #if        #include     #undef
```

Как легко заметить, все они начинаются со знака #. Кроме того, каждая директива препроцессора должна занимать отдельную строку. Например, строка

```
#include <sat500.h>   #include <string.h>
```

рассматривается как недопустимая.

2.15.1. ДИРЕКТИВА #DEFINE

Директива *#define* определяет идентификатор и последовательность символов, которая будет подставляться вместо идентификатора каждый раз, когда он встретится в исходном файле. Идентификатор называется *именем макроса*, а сам процесс замены - *макрозаменой* (а также *макрорасширением*, *макрогенерацией* и *макроподстановкой*. Определение макроса часто называют *макроопределением*, а обращение к макросу — *макровызовом* или *макрокомандой*. Впрочем, иногда *макроопределение* также называется *макрокомандой*).
В общем виде директива выглядит таким образом:

```
#define имя_макроса последовательность_символов
```

Обратите внимание, что в этом выражении нет точки с запятой. Между идентификатором и последовательностью символов *последовательность_символов* может быть любое количество пробелов, но признаком конца последовательности символов может быть только разделитель строк. Предположим, например, что вместо значения 1 нужно использовать слово LEFT (левый), а вместо значения 0 — слово RIGHT (правый). Тогда можно сделать следующие объявления с помощью директивы *#define*:

```
#define LEFT 1
#define RIGHT 0
```

В результате компилятор будет подставлять 1 или 0 каждый раз, когда в вашем файле исходного кода встречается идентификатор соответственно LEFT или RIGHT. Например, следующий код выводит на экран 0 1 2:

```
printf("%d %d %d", RIGHT, LEFT, LEFT+1);
```

После определения имя макроса можно использовать в определениях других имен макросов. Вот, например, код, определяющий значения ONE (один), TWO (два) и three (три):

```
#define ONE    1
#define TWO    ONE+ONE
#define THREE  ONE+TWO
```

Макроподстановка - это просто замена какого-либо идентификатора связанной с ним последовательностью символов. Поэтому если требуется определить стандартное сообщение об ошибке, то можно написать примерно следующее:

```
#define E_MS "стандартная ошибка\n"
/* ... */
printf(E_MS);
```

Теперь каждый раз, когда встретится идентификатор E_MS, компилятор будет его заменять строкой "*стандартная ошибка\n*". Для компилятора выражение *printf()* на самом деле будет выглядеть таким образом:

```
printf("стандартная ошибка\n");
```

Если идентификатор находится внутри строки, заключенной в кавычки, то замены не будет. Например, при выполнении кода

```
#define XYZ это проверка
printf("XYZ");
```

вместо сообщения это проверка будет выводиться последовательность символов XYZ.
Если последовательность_символов не помещается в одной строке, то эту последовательность можно продолжить на следующей строке, поместив в конце предыдущей, как показано ниже, обратную косую черту:

```
#define LONG_STRING "это очень длинная \
строка, используемая в качестве примера"
```

Программисты, пишущие программы на языке С, в именах определяемых идентификаторов часто используют буквы верхнего регистра. Если разработчики программ следуют этому правилу, то тот, кто будет читать их программу, с первого взгляда поймет, что будет происходить макрозамена. Кроме того, все директивы *#define* обычно лучше всего помещать в самом начале файла или в отдельном заголовочном файле, а не разбрасывать по всей программе.
Имена макросов часто используются для определения имен так называемых "*магических чисел*" (встречающихся в программе). Например, имеется программа, в которой определяется массив и несколько процедур, получающих доступ к этому массиву. Вместо того чтобы размер массива "зашивать в код" в виде константы, этот размер можно определить с помощью оператора *#define*, а затем использовать это имя макроса везде, где требуется размер массива. Таким образом, если требуется изменить этот размер, то потребуется изменить только соответствующий оператор *#define*, а затем перекомпилировать программу. Рассмотрим, например, фрагмент программы

```
#define MAX_SIZE 100
/* ... */
int balance[MAX_SIZE];
/* ... */
for(i=0; i<MAX_SIZE; i++) printf("%d", balance[i]);
/* ... */
for(i=0; i<MAX_SIZE; i++) x += balance[i];
```

Размер массива balance определяется именем макроса *MAX_SIZE*, и поэтому если этот размер потребуется в будущем изменить, то надо будет изменить только определение *MAX_SIZE*. В результате при перекомпиляции программы все обращения к этому имени макроса, находящиеся после измененного определения, будут автоматически изменены.

Определение макросов с формальными параметрами

У директивы *#define* имеется еще одно большое достоинство: имя макроса может определяться с формальными параметрами. Тогда каждый раз, когда в программе встречается имя макроса, то используемые в его определении формальные параметры заменяются теми аргументами, которые встретились в программе. Такого рода макросы называются макросами с формальными параметрами. Например,

```
#include <sat500.h>

#define ABS(a)  (a) < 0 ? -(a) : (a)

void main(void)
{
    ClrScr();
    printf("модули чисел -1 и 1 равны соответственно %d и %d",
    ABS(-1), ABS(1));
    ShowScr();
    while(1);
}
```

Во время компиляции этой программы вместо формального параметра a из определения макроса будут подставляться значения -1 и 1. Скобки, в которых находится a, позволяют в любом случае сделать правильную замену. Например, если скобки, стоящие вокруг a, удалить, то выражение

```
ABS(10-20)
```

после макрозамены будет преобразовано в

```
10-20 < 0 ? -10-20 : 10-20
```

и может привести к неправильному результату.
Использование вместо настоящих функций макросов с формальными параметрами дает одно существенное преимущество: увеличивается скорость выполнения кода, потому что в таких случаях не надо тратить ресурсы на вызов функций. Однако если у макроса с формальными параметрами очень большие размеры, то тогда из-за дублирования кода увеличение скорости достигается за счет увеличения размеров программы.

2.15.2. ДИРЕКТИВА #ERROR

Директива *#error* заставляет компилятор прекратить компиляцию. Эта директива используется в основном для отладки. В общем виде директива *#error* выглядит таким образом:

```
#error сообщение-об-ошибке
```

сообщение-об-ошибке в двойные кавычки не заключается. Когда встречается директива *#error*, то выводится сообщение об ошибке — возможно, вместе с другой информацией, определяемой компилятором.

2.15.3. ДИРЕКТИВА #INCLUDE

Директива *#include* дает указание компилятору читать еще один исходный файл - в дополнение к тому файлу, в котором находится сама эта директива. Имя исходного файла должно быть заключено в двойные кавычки или в угловые скобки. Например, обе директивы

```
#include "sat500.h"
#include <sat500.h>
```

дают компилятору указание читать и компилировать заголовок для функций САТ-500.
Файлы, имена которых находятся в директивах *#include*, могут в свою очередь содержать другие директивы *#include*. Они называются вложенными директивами *#include*. Количество допустимых уровней вложенности неограничено.

Способ поиска файла зависит от того, заключено ли его имя в двойные кавычки или же в угловые скобки. Если имя заключено в угловые скобки, то поиск файла проводится тем способом, который определен в компиляторе. Часто это означает поиск определенного каталога, специально предназначенного для хранения таких файлов. Если имя заключено в кавычки, то поиск файла проводится другим способом. Во многих компиляторах это означает поиск файла в текущем рабочем каталоге. Если же файл не найден, то поиск повторяется уже так, как будто имя файла заключено в угловые скобки.
Обычно большинство программистов имена стандартных заголовочных файлов заключают в угловые скобки. А использование кавычек обычно приберегается для имен специальных файлов, относящихся к конкретной программе. Впрочем, твердого и простого правила, по которому кавычки требуется использовать именно таким образом, не существует.
В С-программе директиву *#include* можно использовать не только для указания имени файла, содержащего обычный исходный текст программы, но и для указания заголовка. В языке С определен набор стандартных заголовков, содержащих необходимую информацию о различных библиотеках этого языка. Заголовок — это стандартный идентификатор, который может соответствовать имени файла, а может и не соответствовать ему. Таким образом, заголовок является просто абстракцией, которая гарантирует наличие некоторой информации. Однако на практике в языке С заголовки почти всегда являются именами файлов.

2.15.4. ДИРЕКТИВЫ УСЛОВНОЙ КОМПИЛЯЦИИ

Имеется несколько директив, которые дают возможность выборочно компилировать части исходного кода вашей программы. Этот процесс называется условной компиляцией и широко используется фирмами, живущими за счет коммерческого программного обеспечения - теми, которые поставляют и поддерживают многие специальные версии одной программы.

Директивы #if, #else, #elif и #endif

Возможно, самыми распространенными директивами условной компиляции являются *#if, #else, #elif* и *#endif*. Они дают возможность в зависимости от значения константного выражения включать или исключать те или иные части кода.
В общем виде директива *#if* выглядит таким образом:

```
#if константное выражение
    последовательность операторов
#endif
```

Если находящееся за *#if* константное выражение истинно, то компилируется код, который находится между этим выражением и *#endif*. В противном случае этот промежуточный код пропускается. Директива *#endif* обозначает конец блока *#if*. Например,

```
/* Простой пример #if. */
#include <sat500.h>

#define MAX 100

void main(void)
{
    ClrScr();
    #if MAX>99
         printf("Компилирует для массива, размер которого больше 99.\n");
    #endif
    ShowScr();
    while(1);
}
```

Это программа выводит сообщение на экран, потому что *MAX* больше 99. В этом примере показано нечто очень важное. Значение выражения, находящегося за директивой *#if*, должно быть вычислено во время компиляции. Поэтому в этом выражении могут находиться только ранее определенные идентификаторы и константы, - но не переменные.
Директива *#else* работает в основном так, как *else* - ключевое слово языка С: задает альтернативу на тот случай, если не выполнено условие *#if*. Предыдущий пример можно дополнить следующим образом:

```
/* Простой пример #if/#else. */

#include <sat500.h>

#define MAX 10

void main(void)
{
      ClrScr();
      #if MAX>99
            printf("Компилирует для массива, размер которого больше 99.\n");
      #else
            printf("Компилирует для небольшого массива.\n");
      #endif
      ShowScr();
      while(1);
}
```

В этом случае выясняется, что *MAX* меньше 99, поэтому часть кода, относящаяся к *#if*, не компилируется. Однако компилируется альтернативный код, относящийся к *#else*, и откомпилированная программа будет отображать сообщение Компилируется для небольшого массива.
Обратите внимание, что директива *#else* используется для того, чтобы обозначить и конец блока *#if*, и начало блока *#else*. Это естественно, поскольку любой директиве *#if* может соответствовать только одна директива *#endif*.
Директива *#elif* означает "*else if*" и устанавливает для множества вариантов компиляции цепочку *if-else-if*. После *#elif* находится константное выражение. Если это выражение истинно, то компилируется находящийся за ним блок кода, и больше не проверяются никакие другие выражения *#elif*. В противном же случае проверяется следующий блок этой последовательности. В общем виде *#elif* выглядит таким образом:

```
#if выражение
    последовательность операторов
#elif выражение 1
    последовательность операторов
#elif выражение 2
    последовательность операторов
#elif выражение 3
    последовательность операторов
#elif выражение 4
.
.
.
#elif выражение N
    последовательность операторов
#endif
```

Например, в следующем фрагменте для определения знака денежной единицы используется значение ACTIVE_COUNTRY (для какой страны):

```
#define US 0
```

```
#define ENGLAND 1
#define FRANCE 2

#define ACTIVE_COUNTRY US

#if ACTIVE_COUNTRY == US
  char currency[] = "dollar";
#elif ACTIVE_COUNTRY == ENGLAND
  char currency[] = "pound";
#else
  char currency[] = "euro";
#endif
```

При вложенности каждая директива *#endif, #else* или *#elif* относится к ближайшей директиве *#if* или *#elif*. Например, совершенно правильным является следующий фрагмент кода:

```
#if MAX>100
  #if SERIAL_VERSION
    int port=198;
  #elif
    int port=200;
  #endif
#else
  char out_buffer[100];
#endif
```

Директивы #ifdef и #ifndef

Другой способ условной компиляции - это использование директив *#ifdef* и *#ifndef*, которые соответственно означают "*if defined*" (если определено) и "*if not defined*" (если не определено). В общем виде *#ifdef* выглядит таким образом:

```
#ifdef имя_макроса
       последовательность операторов
#endif
```

Блок кода будет компилироваться, если *имя макроса* было определено ранее в операторе *#define*. В общем виде оператор *#ifndef* выглядит таким образом:

```
#ifndef имя_макроса
        последовательность операторов
#endif
```

Блок кода будет компилироваться, если *имя макроса* еще не определено в операторе #define.
И в *#ifdef*, и в *#ifndef* можно использовать оператор *#else* или *#elif*. Например,

```
#include <sat500.h>

#define TED 10


void main(void)
{
      ClrScr();
      #ifdef TED
            printf("Привет, Тед\n");
      #else
```

```
            printf("Привет, кто-нибудь\n");
      #endif
      #ifndef RALPH
            printf("А RALPH не определен, т.ч. Ральфу не повезло.\n");
      #endif

      ShowScr();
      while(1);
}
```

выведет *Привет, Тед, а также А RALPH не определен, т.ч. Ральфу не повезло.*

2.15.5. ДИРЕКТИВА #UNDEF

Директива *#undef* удаляет ранее заданное определение имени макроса, то есть "*аннулирует*" его определение, само имя макроса должно находиться после директивы. В общем виде директива *#undef* выглядит таким образом:

```
#undef имя_макроса
```

Вот как, например, можно использовать эту директиву:

```
#define LEN 100
#define WIDTH 100

char array[LEN];
char vars[WIDTH];

#undef LEN
#undef WIDTH

/* а здесь и LEN и WIDTH уже не определены */
```

И LEN, и WIDTH определены, пока не встретился оператор *#undef*.
Директива *#undef* используется в основном для того, чтобы локализовать имена макросов в тех участках кода, где они нужны.

2.15.6. ИСПОЛЬЗОВАНИЕ DEFINED

Кроме применения *#ifdef*, есть еще второй способ узнать, определено ли имя макроса. Можно использовать директиву *#if* в сочетании с оператором времени компиляции *defined*. В общем виде оператор *defined* выглядит таким образом:

```
defined имя_макроса
```

Если имя_макроса определено, то выражение считается истинным, в противном случае - ложным. Например, чтобы узнать, определено ли имя макроса *MYFILE*, можно использовать одну из двух команд препроцессора:

#if defined MYFILE

или

```
#ifdef MYFILE
```

Можно также задать противоположное условие, поставив ! прямо перед defined. Например, следующий фрагмент компилируется только тогда, когда имя макроса DEBUG не определено:

```
#if !defined DEBUG
  printf("Окончательная версия!\n");
#endif
```

Единственная причина, по которой используется оператор *defined*, состоит в том, что с его помощью в *#elif* можно узнать, определено ли имя макроса.

2.15.7. ДИРЕКТИВА #PRAGMA

Директива *#pragma* - это определяемая реализацией директива, которая позволяет передавать компилятору различные инструкции.
Компилятор С-Скрипт поддерживает некоторые расширенные возможности, задаваемые директивой *#pragma*.
Некоторые директивы задают способы интеграции с аппаратурой контроллера CAT-500.
Для связи с аппаратурой используются понятие аппаратная переменная и аппаратная функция.
Перед использованием их следует описать при помощи директивы *#pragma*.
ВНИМАНИЕ! Вы не можете создать новые аппаратные переменные и функции кроме тех которые описаны в стандартных заголовочных файлах, находящихся в подкаталоге LIB.
Любое изменение данных в этих файлах приведёт к неработоспособности программы. Описания в пунктах `#pragma variable` и `#pragma function` далее приведены как справочные данные.

Задание номера версии программы

```
#pragma version verh.verl

verh - число, задающее старший номер версии (0-65535)
verl - число, задающее младший номер версии (0-65535)
```

ПРИМЕР:

```
#pragma version 2.19
```

Номер версии программы помещается в исполняемый файл программы, который помещается в контроллер CAT-500.
Номер версии можно просмотреть на дисплее контроллера CAT-500.

Описание аппаратной переменной

```
#pragma variable(ID) тип имя
```

ID – уникальный идентификатор аппаратной переменной (не может быть нулевым).

ПРИМЕР:

```
#pragma variable(14) int AI1
```

После объявления можно обращаться к переменной AI1 на чтение и запись

Описание аппаратного массива переменных

```
#pragma variable (ID) тип имя[размер]
```

ПРИМЕР:

```
#pragma variable(1)  char DI[8]
```

После объявления появятся переменные DI[0]…DI[7] и занимаются идентификаторы: 2..9

Описание аппаратной функции

```
#pragma function (ID) прототип функции
```

ПРИМЕР:

```
#pragma function(2)  void GotoXY(short int x,short int y)
```

После объявления можно вызывать функцию.
Идентификаторы переменных и функций могут иметь одинаковые значения.

Описание настроечного параметра (переменная в меню контроллера)

Меню "**Параметры программы/Настройки/***" контроллера CAT-500 состоит из пунктов, заданных в программе при помощи директивы *#pragma menuitem.*
Каждая такая директива создаёт переменную в программе и связывает её с новым подпунктом меню "**Параметры программы/Настройки/***".
Общий вид директивы добавления пункта меню:

```
#pragma menuitem тип имя: "Текст",min,max,value
```

, где :
тип имя - тип и имя создаваемой переменной
"Текст" - текст, отображаемый в меню
min - минимально допустимое значение переменной
max - максимально допустимое значение переменной
value - значение переменной по умолчанию (при первом запуске)

Порядок переменных в меню определяется порядком объявления переменных.
Иерархические пункты меню не поддерживаются.

ПРИМЕР:

```
#pragma menuitem int Period: "Период, мс",10,5000,1000
```

Более подробно описание директивы можно найти в разделе *Пункты меню*

Описание используемой картинки

Для использования картинки в программе её необходимо предварительно описать при помощи директивы:

```
#pragma picture(ID) "имя файла"
```

ID – уникальный идентификатор картинки (не может быть нулевым).
В программе картинку можно будет нарисовать с помощью функции *DrawPicture*.
ПРИМЕР:

```
#pragma picture(1) "Image1.bmp"
#pragma picture(2) "c:\\Tmp\\Bmp\\led_red.bmp"
```

Включение исходного текста программы в файл программы САТ-500

Для включения исходного текста в файл программы необходимо вставить в текст программы следующую директиву:
#pragma source on

Директива должна быть вставлена в основной файл программы и в те файлы, которые должны быть включены в тело программы.
Любой файл, участвующей в компиляции программы и содержащий данную директиву *#pragma source on* будет помещён в тело программы в виде текста.

Включение в программу дополнительной информации

Для включения различной дополнительной информации в программу следует использовать следующую директиву:
#pragma inf("HDR") "TEXT"

Ниже приведена таблица возможных вариантов заголовка HDR:

HDR	TEXT
$or	Наименование организации-разработчика программы
$st	Ссылка на сайт программы
$es	E-mail адрес поддержки программы
$au	Автор программы
$tl	Телефонный номер организации или автора программы
$em	E-mail адрес автора программы
$up	Адрес каталога обновления программы на web-сервере
$rm	Примечание – произвольный текст, заданный автором программы

Директивы pragma с неизвестным значением HDR вызывают ошибки.

ПРИМЕР:
```
#pragma inf("$or") "МНПП Сатурн"
#pragma inf("$es") "info@mnppsaturn.ru"
#pragma inf("$au") "Иванов Иван"
#pragma inf("$rm") "Тестовая программа для изделия СТ20761.34.19"
```

2.15.8. ОПЕРАТОРЫ ПРЕПРОЦЕССОРА # И

Операторы препроцессора # и ## в компиляторе С-Скрипт не поддерживаются.

2.16. СРЕДСТВО TYPEDEF

Новые имена типов данных можно определять, используя ключевое слово *typedef*. На самом деле таким способом новый тип данных не создается, а всего лишь определяется новое имя для уже существующего типа. Этот процесс может помочь сделать машинно-зависимые программы более переносимыми. Если вы для каждого машинно-зависимого типа данных, используемого в вашей программе, определяете данное вами имя, то при компиляции для новой среды придется менять только операторы *typedef*. Такие выражения могут помочь в самодокументировании кода, позволяя давать понятные имена стандартным типам данных. Общий вид декларации *typedef* (оператора ***typedef***) такой:

```
typedef тип новое_имя;
```

где *тип* - это любой тип данных языка С, а *новое_имя* - новое имя этого типа. Новое имя является дополнением к уже существующему, а не его заменой.
Например, для ***char*** можно создать новое имя с помощью

```
typedef char bool;
```

Это выражение даёт компилятору указание считать **`bool`** еще одним именем ***char***. Затем, используя ***bool***, можно создать переменную типа ***char***:

```
bool FlagStop;
```

Теперь имеется переменная с `FlagStop` типа ***bool***, а ***bool*** является еще одним именем типа ***char***. Теперь, когда имя ***bool*** определено, его можно использовать и в другом операторе *typedef*. Например, выражение

```
typedef bool boolean;
```

даёт компилятору указание признавать *boolean* в качестве ещё одного имени *bool*, которое в свою очередь является ещё одним именем *char*.
Использование операторов *typedef* может облегчить чтение кода и его перенос на новую машину. Однако новый физический тип данных таким способом вы не создадите.

Примеры часто употребляемых определений *typedef:*

```
typedef char               int8_t;
typedef unsigned char      uint8_t;
typedef short int          int16_t;
typedef unsigned short int uint16_t;
typedef int                int32_t;
typedef unsigned int       uint32_t;
typedef unsigned char      BYTE;
typedef unsigned short int WORD;
typedef unsigned int       UINT;
typedef unsigned int       DWORD;
```

2.17. ИСПОЛЬЗОВАНИЕ SIZEOF

Оператор *sizeof* подсчитывает размер любой переменной или любого типа и может быть полезен, если в программах требуется свести к минимуму машинно-зависимый код. Перед тем как переходить к дальнейшему изложению, отметим, что определенные типы данных имеют следующие размеры:

Тип	Размер в байтах
char	1
short	2
int	4

Поэтому при выполнении следующего кода на экран будут выведены числа 1, 2 и 4:

```
char ch;
short int s;
int i;

printf("%d", sizeof(ch));
printf("%d", sizeof(s));
printf("%d", sizeof(i));
```

В случае применения к массиву можно получить его размер в байтах.
В следующем примере выведется число 20:

```
short int Vars[10];
printf("%d", sizeof(Vars));
```

2.18. ПЕРЕЧИСЛЕНИЯ ENUM

Перечисление - это набор именованных целых констант. Перечисления довольно часто встречаются в повседневной жизни.

ВНИМАНИЕ:
В компиляторе С-Скрипт перечисление не может быть типом, т.е. допускаются только неименованные перечисления.

Началом объявления перечислимого типа служит ключевое слово *enum*. Перечисление в общем виде выглядит так:

```
enum {список перечисления};
```

Следующий фрагмент кода определяет неименованное перечисление:

```
enum { penny, nickel, dime, quarter, half_dollar, dollar};
```

Главное, что нужно знать для понимания перечислений - каждый их элемент представляет целое число. В таком виде элементы перечислений можно применять везде, где используются целые числа. Каждому элементу дается значение, на единицу большее, чем у его предшественника. Первый элемент перечисления имеет значение 0.
Поэтому объявление приводит к появленью следующих констант:

```
penny           0
nickel          1
dime            2
quarter         3
half_dollar     4
dollar          5
```

Однако для одного или более элементов можно указать значение, используемое как инициализатор. Для этого после перечислителя надо поставить знак равенства, а затем - целое значение. Перечислителям, которые идут после инициализатора, присваиваются значения, большие предшествующего. Например, следующий код присваивает quarter значение 100:

```
enum { penny, nickel, dime, quarter=100, half_dollar, dollar};
```

вот какие значения появились у этих элементов:

```
penny           0
nickel          1
dime            2
quarter         100
half_dollar     101
dollar          102
```

2.19. ОБЪЕДИНЕНИЯ UNION

Объединение - это место в памяти, которое используется для хранения переменных, разных типов. Объединение дает возможность интерпретировать один и тот же набор битов не менее, чем двумя разными способами.

ВНИМАНИЕ:
В компиляторе С-Скрипт объединение не может быть типом, т.е. допускаются только неименованные объединения.

Объявление объединения (начинается с ключевого слова union) и в общем виде выглядит так:

```
union {
  тип имя-члена;
  тип имя-члена;
  тип имя-члена;
  .
  .
  .
};
```

Например:

```
union
{
    int i;
    char ch;
};
```

Это объявление создает две переменные, но обе эти переменные расположены по одному адресу. Таким образом *ch* - это младший байт переменной *i*.
Когда переменная объявляется с ключевым словом *union*, компилятор автоматически выделяет столько памяти, чтобы в ней поместился самый большой член нового объединения. Например, целые значения занимают по 4 байта, для размещения *i* необходимо, чтобы длина этого объединения составляла 4 байта, даже если для *ch* требуется только 1 байт.
Объединения часто используются тогда, когда нужно выполнить специфическое преобразование типов, потому что хранящиеся в объединениях данные можно обозначать совершенно разными

способами. Например, используя объединения, можно манипулировать байтами, составляющими значение типа *int*.

```
union
{
    int i;
    char ch[4];
};
```

Используя массив *ch* можно получить удобный доступ к байтам целого числа *i* и наоборот - заполняя *ch* можно изменить *i*.

2.20. PRINTF

Функция printf позволяет пользователю выводить информацию на дисплей. Для использования функции необходимо подключить файл с описаниями с помощью директивы:

```
#include <sat500.h>
```

Описание функции имеет вид:

```
int printf(char * format, ...);
```

, где *format* - управляющая строка
... - аргументы, выводимые в формате, заданном в управляющей строке

В случае успеха функция printf() возвращает число выведенных символов.

Управляющая_строка состоит из элементов двух видов. Первый из них - это символы, которые предстоит вывести на экран; второй - это спецификаторы преобразования, которые определяют способ вывода стоящих за ними аргументов. Каждый такой спецификатор начинается со знака процента, за которым следует код формата. Аргументов должно быть ровно столько, сколько и спецификаторов, причем спецификаторы преобразования и аргументы должны попарно соответствовать друг другу в направлении слева направо. Например, в результате такого вызова *printf()*

```
printf("Мне нравится язык %c %s", 'C', "и к тому же очень сильно!");
```

Будет выведено:

```
Мне нравится язык C и к тому же очень сильно!
```

В этом примере первому спецификатору преобразования (%c), соответствует символ 'C', а второму (%s), — строка "*и к тому же очень сильно!*".
В функции printf(), как видно из таблицы, имеется основной набор спецификаторов преобразования.

Код	***Формат***
%c	Символ
%d	Десятичное целое со знаком
%s	Строка символов
%u	Десятичное целое без знака
%x	Шестнадцатеричное без знака (буквы на нижнем регистре)
%X	Шестнадцатеричное без знака (буквы на верхнем регистре)
%%	Выводит знак %

Вывод символов

Для вывода отдельного символа используйте %c. В результате соответствующий аргумент будет выведен на экран без изменения.
Для вывода строки используйте %s.

Вывод чисел

Числа в десятичном формате со знаком отображаются с помощью спецификатора преобразования %d.
Для вывода целого значения без знака используйте %u.
Целые числа без знака можно выводить в шестнадцатеричном формате, используя спецификатор преобразования %X или %x. Так как в шестнадцатеричной системе для представления чисел от 10 до 15 используются буквы от A до F, то эти буквы можно выводить на верхнем или на нижнем регистре. Как показано ниже, в первом случае используется спецификатор преобразования %X, а во втором — спецификатор преобразования %x:

```
#include <sat500.h>

void main(void)
{
    unsigned num;
    ClrScr();
    BigFont=0;
    for(num=0; num < 14; num++)
    {
        printf("%d ", num);
        printf("%x ", num);
        printf("%X\n", num);
    }
    ShowScr();
    while(1);
}
```

Вот что вывела эта программа:

Отображение адреса

Для отображения адреса используйте спецификатор преобразования **%X,%x или %d**.

Модификаторы формата

Во многих спецификаторах преобразования можно указать модификаторы, которые слегка меняют их значение. Например, можно указывать минимальную ширину поля, количество десятичных разрядов и выравнивание по левому краю. Модификатор формата помещают между знаком процента и кодом формата. Об этих модификаторах сейчас и пойдет речь.

Модификаторы минимальной ширины поля

Целое число, расположенное между знаком % и кодом формата, играет роль модификатора минимальной ширины поля. Если указан модификатор минимальной ширины поля, то чтобы ширина поля вывода была не меньше указанной минимальной длины, при необходимости вывод будет дополнен пробелами. Если же выводятся строки или числа, которые длиннее указанного минимума, то они все равно будут отображаться полностью. По умолчанию для дополнения используются пробелы. А если для этого надо использовать нули, то перед модификатором ширины поля следует поместить 0. Например, *%05d* означает, что любое число, количество цифр которого меньше пяти, будет дополнено таким количеством нулей, чтобы число состояло из пяти цифр.

Модификатор минимальной ширины поля чаще всего используется при создании таблиц, в которых столбцы должны быть выровнены по вертикали.

Выравнивание вывода

По умолчанию весь вывод выравнивается по правому краю. То есть если ширина поля больше ширины выводимых данных, то эти данные располагаются по правому краю поля. Вывод по левому краю можно назначить принудительно, поместив знак минус прямо за %. Например, *%-8d* означает, что число с будет выровнено по левому краю 8-символьного поля.
В следующей программе показано, как применяется выравнивание по левому краю:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;     //Цвет кисти - чёрный
    ClrScr();                   //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;      //Цвет шрифта - белый
    BigFont=0;                  //Установить обычный шрифт (12x16, 12 пикселей
по горизонтали, 16 пикселей  по вертикали)
    printf(".......................\n");
    printf("по правому краю: %8d\n", 100);
    printf("по левому краю: %-8d\n", 100);
    ShowScr();                  //Вывести нарисованное не дисплей
    while(1);                   //Бесконечная пауза
}
```

И вот что получилось:

Обработка данных других типов

Некоторые модификаторы в вызове функции *printf()* позволяют отображать целые числа типа ***short*** и ***char***. Такие модификаторы можно использовать для следующих спецификаторов типа: d, u и x.

Пример

Ниже приведён пример, который поможет разобраться с форматированным выводом *printf()*.
Пример носит академический характер, т.к. такое большое число строк нельзя одновременно вывести на дисплей CAT-500.

```
#include <sat500.h>

void main(void)
{
    ClrScr();
    char *ptr = "Hello world!";
    char *np = 0;
    int i = 5;
    unsigned int bs = sizeof(int)*8;
    int mi;
    mi = (1 << (bs-1)) + 1;
    printf("%s\n", ptr);                                  // Hello world!
    printf("printf test\n");                              // printf test
    printf("%s is null pointer\n", np);                   // (null) is null
pointer
    printf("%d = 5\n", i);                                // 5 = 5
    printf("%d = - max int\n", mi);                       // -2147483647 = -
max int
```

```
    printf("char %c = 'a'\n", 'a');                          // char a = 'a'
    printf("hex %x = ff\n", 0xff);                           // hex ff = ff
    printf("hex %02x = 00\n", 0);                            // hex 00 = 00
    printf("signed %d = unsigned %u = hex %x\n",-3,-3,-3);  // signed -3 = un-
signed 4294967293 = hex fffffffd
    printf("%d %s(s)%", 0, "message");                       // 0 message(s)
    printf("\n");                                            //
    printf("%d %s(s) with %%\n", 0, "message");              // 0 message(s)
with %
    printf("justif: \"%-10s\"\n", "left");                   // justif: "left "
    printf("justif: \"%10s\"\n", "right");                   // justif: "
right"
    printf(" 3: %04d zero padded\n", 3);                     // 3: 0003 zero
padded
    printf(" 3: %-4d left justif.\n", 3);                    // 3: 3 left jus-
tif.
    printf(" 3: %4d right justif.\n", 3);                    // 3: 3 right jus-
tif.
    printf("-3: %04d zero padded\n", -3);                    // -3: -003 zero
padded
    printf("-3: %-4d left justif.\n", -3);                   // -3: -3 left
justif.
    printf("-3: %4d right justif.\n", -3);                   // -3: -3 right
justif.
    ShowScr();
    while(1);
}
```

2.21. SPRINTF

```
#include <sat500.h>

int sprintf(char * buf,char * format, ...);
```

Функция *sprintf()* идентична *printf()*, за исключением того, что вывод производится в массив, указанный аргументом ***buf***.
По окончании работы функции этот массив будет завершаться символом конца строки (нуль-символом). Подробности рассматриваются в разделе, посвященном описанию функции *printf()*.
Возвращаемая величина равна количеству символов, действительно занесенных в массив.

Безопасность

Функция *sprintf* имеет следующие уязвимости:
1. Размер буфера ***buf***, передаваемого функции должен быть достаточным для печати заданной строки. В случае недостаточности размера буфера происходит разрушение области статических или стековых переменных, что с большой вероятностью нарушает правильность работы программы.
2. Количество заданных аргументов должно совпадать с количеством спецификаторов, заданных в строке ***format***. В случае несоответствия возможно разрушение областей памяти и как следствие ошибочная работа программы.

Пример:

```
#include <sat500.h>
```

```
char Str[64];
const char Text1[]="По правому краю: %8d\n";
const char Text2[]="По левому краю: %-8d\n";

void main(void)
{
    BrushColor=COLOR_BLACK;     //Цвет кисти - чёрный
    ClrScr();                   //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;      //Цвет шрифта - белый
    BigFont=0;                  //Установить обычный шрифт (12x16, 12 пикселей
по горизонтали, 16 пикселей  по вертикали)
    //Вывод текста в строку Str
    sprintf(Str,".......................\n");
    //Вывод строки Str на экран
    puts(Str);
    //Вывод текста и числа в строку Str
    sprintf(Str,Text1,100);
    //Вывод строки Str на экран
    puts(Str);
    //Вывод текста и числа в строку Str
    sprintf(Str,Text2,100);
    //Вывод строки Str на экран
    puts(Str);
    ShowScr();                  //Вывести нарисованное на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:

2.22. STRING.H

string.h - заголовочный файл стандартной библиотеки языка Си, содержащий функции для работы с нуль-терминированными строками и различными функциями работы с памятью.
Строковые функции работают только с набором символов ASCII.

Имя	***Описание***
*void * memcpy (void *dest,const void *src,size_t n);*	Копирует ***n*** байт из области памяти ***src*** в ***dest***, которые не должны пересекаться, в противном случае результат не определён (возможно как правильное копирование, так и нет)
*void * memmove (void *dest,const void *src,size_t n);*	Копирует **n** байт из области памяти **src** в **dest**, которые в отличие от ***memcpy*** могут перекрываться
*void * memchr (const char *s,char c,size_t n);*	Возвращает указатель на первое вхождение значения с среди первых ***n*** байтов ***s*** или ***NULL***, если не найдено
*int memcmp (const char *s1, const char *s2, size_t n);*	Сравнивает первые ***n*** символов в областях памяти ***s1*** и ***s2***
*void * memset (char *buf, int z, size_t);*	Заполняет область ***buf*** памяти одним байтом ***z***

*char * strcat (char *dest, const char *src);*	Дописывает строку ***src*** в конец ***dest***
*char * strchr (const char *s, int c);*	Возвращает адрес символа ***c*** в строке ***s***, начиная с головы, или ***NULL***, если строка ***s*** не содержит символ ***c***
*int strcmp (const char *s1, const char *s2);*	Лексикографическое сравнение строк (возвращает ***0***, если строки одинаковые, положительное, если первая ***s1*** строка больше ***s2***, и отрицательное, если меньше)
*char * strcpy (char *to, const char *from);*	Копирует строку из одного места в другое
*size_t strcspn (const char *s, const char *reject);*	Определяет максимальную длину начальной подстроки, состоящей исключительно из байтов, не перечисленных в ***reject***
*size_t strlen (const char * s);*	Возвращает длину строки ***s***
*char * strncat (char *dest, const char *src, size_t n);*	• Дописывает не более ***n*** начальных символов строки ***src*** (или всю ***src***, если её длина меньше) в конец ***dest***
*int strncmp (const char *s1, const char *s2, size_t n);*	Лексикографическое сравнение первых ***n*** байтов строк ***s1*** и ***s2***
*char * strncpy (char *to, const char *from, size_t n);*	Копирует до ***n*** байт строки из одного места в другое
*char * strpbrk (const char *s, const char *accept);*	Находит первое вхождение любого символа, перечисленного в ***accept***
*char * strrchr (const char *s, int c);*	Возвращает адрес символа ***c*** в строке ***s***, начиная с хвоста, или ***NULL***, если строка ***s*** не содержит символ ***c***
*size_t strspn (const char * s, const char * accept);*	Определяет максимальную длину начальной подстроки, состоящей исключительно из байтов, перечисленных в ***accept***
*char * strstr (const char *str1, const char *str2);*	Находит первое вхождение строки ***str2*** в ***str1***
*char * strtok (char * str1, const char * str2);*	Преобразует строку в последовательность токенов.(Не реентерабельная)
*int atoi (const char * s);*	Функция *atoi* преобразует строку ***s*** в целое значение типа *int*.

2.22.1. MEMCPY

```
#include <string.h>

void *memcpy(void *to, const void *from, size_t count);
```

Функция *memcpy()* копирует ***count*** символов из массива, адресуемого параметром ***from***, в массив, адресуемый параметром ***to***. Если заданные массивы перекрываются, поведение функции *memcopy()* не определено.

Функция *memcpy()* возвращает значение указателя to.

Пример
Данная программа копирует содержимое массива buf1 в массив buf2 и выводит результат на дисплей.

```
#include <sat500.h>
#include <string.h>

#define SIZE 80

void main(void)
{
    char buf1[SIZE],buf2[SIZE];
    ClrScr();
    strcpy(buf1,"Когда, в случае если...");
    memcpy(buf2,buf1,SIZE);
    printf(buf2);
    ShowScr();
    while(1)
    {
    }
}
```

См. также
memmove

2.22.2. MEMMOVE

```
#include <string.h>

void *memmove(void *to, const void *from, size_t count);
```

Функция *memmove()* копирует ***count*** символов из массива, адресуемого параметром ***from***, в массив, адресуемый параметром ***to***. Если заданные массивы перекрываются, процесс копирования проходит корректно, т.е. соответствующее содержимое будет помещено в массив ***to***, но содержимое массива ***from*** при этом изменится.

Функция *memmove()* возвращает значение указателя to.

Пример
Данная программа сдвигает содержимое массива str на 10 позиций в сторону младших адресов и выводит результат на дисплей.

```
#include <sat500.h>
#include <string.h>

#define SIZE 80

void main(void)
{
    char str[SIZE],*p;
    ClrScr();
    strcpy(str,"Когда, в случае если...");
    p = str + 10;
    memmove(str, p, SIZE);
```

```
    printf("результат сдвига: %s", str);
    ShowScr();
    while(1)
    {
    }
}
```

См. также
memcpy

2.22.3. MEMCHR

```
#include <string.h>

void *memchr(const void *buffer, int ch, size_t count);
```

Функция *memchr()* просматривает массив, адресуемый параметром ***buffer***, чтобы отыскать первое вхождение символа ***ch*** в первых ***count*** символах.

Эта функция возвращает указатель на первый из символов ***ch***, входящих в массив ***buffer***, или нулевой указатель, если символ ***ch*** не найден.

Пример
Данная программа выводит на экран сообщение из примера.

```
#include <sat500.h>
#include <string.h>

void main(void)
{
    char *p;
    ClrScr();
    p = memchr("строка из примера",' ',17);
    printf(p);
    ShowScr();
    while(1)
    {
    }
}
```

Еще один пример
Некоторые примеры, приведенные в качестве иллюстрации применения строковых функций, иногда носят "несколько учебный" характер и не всегда могут быть рекомендованы для профессионального программирования. Например, указание константы 17 в операторе

```
p = memchr("строка из примера", ' ', 17);
```

предыдущего примера едва ли может служить хорошим примером для программиста. Подумайте, что будет, если изменить строку? Опять подсчитывать количество символов? Чтобы избежать этого, во многих руководствах предлагается использовать для этой цели функцию strlen.

```
#include <sat500.h>
#include <string.h>

int main(void)
{
    char str[17];
```

```
    char *ptr;
    ClrScr();
    strcpy(str, "This is a string");
    ptr = (char *) memchr(str, 'r', strlen(str));
    if (ptr)
    printf("The character 'r' is at position: %d\n", ptr - str);
    else
    printf("The character was not found\n"); return 0;
    ShowScr();
    while(1)
    {
    }
}
```

В данном примере, правда, константа указана в объявлении массива str. Однако этого можно было легко избежать, проинициализировав массив в объявлении. Правда, остается еще один недостаток: некоторое снижение эффективности кода из-за вызова функции strlen. Но и его можно устранить, если длина строки может быть вычислена при компиляции (как в наших примерах). Действительно, тогда ведь можно воспользоваться операцией sizeof и записать не константу 17, а константное выражение sizeof (str) /sizeof (char).

См. также
memcpy *strlen*

2.22.4. MEMCMP

```
#include <string.h>

int memcmp(const void *buf1, const void *buf2, size_t count);
```

Функция *memcmp()* сравнивает первые count символов массивов, адресуемых параметрами ***buf1*** и ***buf2***.

Функция *memcmp()* возвращает целое значение, которое интерпретируется следующим образом:

Значение	***Результат сравнения***
Меньше нуля	buf1 меньше buf2
Нуль	buf1 равен buf2
Больше нуля	buf1 больше buf2

Пример
Данная программа выдает результат сравнения двух своих аргументов, которые задаются в командной строке.

```
#include <sat500.h>
#include <string.h>

char s1[]="Это строка 1";
char s2[]="Это строка 2";

void main(void)
{
    int outcome, len, l1, l2;
    ClrScr();
    //определение длины более короткой строки
    l1 = strlen(s1);
```

```
    l2 = strlen(s2);
    len = l1 < l2 ? l1:l2;
    outcome = memcmp(s1,s2,len);
    if(!outcome) printf("Равны");
    else if(outcome<0) printf("Первый меньше второго.");
    else printf("Первый больше второго.");
    ShowScr();
    while(1)
    {
    }
}
```

См. также
memchr *memcpy* *strlen*

2.22.5. MEMSET

```
#include <string.h>

void *memset(void *buf, int ch, size_t count);
```

Функция *memset()* копирует младший байт параметра ***ch*** в первые ***count*** символов массива, адресуемого параметром ***buf***.

Функция возвращает значение указателя ***buf***.

Чаще всего функция *memset()* используется для инициализации области памяти некоторым известным значением.

Пример
Данный фрагмент инициализирует первые 100 байтов массива, адресуемого указателем buf, нулями. Затем он помещает символы X в первые 10 байтов этого массива и выводит строку XXXXXXXXXX.

```
memset(buf, '\0', 100);
memset(buf, 'X', 10);
printf(buf);
```

См. также
memcpy *memcmp* *memmove*

2.22.6. STRCAT

```
#include <string.h>

char *strcat(char *str1, const char *str2);
```

Функция *strcat()* присоединяет к строке ***str1*** копию строки ***str2*** и завершает строку ***str1*** нулевым символом. Конечный нуль-символ, первоначально завершающий строку ***str1***, перезаписывается первым символом строки ***str2***. Строка ***str2*** при этом не изменяется. Если заданные массивы перекрываются, поведение функции *strcat()* не определено.

Функция *strcat()* возвращает значение указателя ***str1***.

Помните, что при выполнении операций с массивами символов контроль нарушения их границ не выполняется, поэтому программист должен сам позаботиться о достаточном размере массива ***str1***, позволяющем вместить как его исходное содержимое, так и содержимое массива ***str2***.

Пример

```
#include <sat500.h>
#include <string.h>

void main(void)
{
    char s1[80], s2[80];
    ClrScr();
    strcpy(s1,"привет");
    strcpy(s2,"Всем ");
    strcat(s2,s1);
    printf(s2);
    ShowScr();
    while(1)
    {
    }
}
```

См. также
strcpy *strncat*

2.22.7. STRCHR

```
#include <string.h>

char *strchr(const char *str, int ch);
```

Функция *strchr()* возвращает указатель на первое вхождение младшего байта параметра ***ch*** в строку ***str***.
Если указанный символ не найден, возвращается нулевой указатель.

Пример

```
#include <sat500.h>
#include <string.h>

void main(void)
{
    char *p;

    ClrScr();
    p = strchr("строка из примера", ' ');
    printf(p);
    ShowScr();
    while(1)
    {
    }
}
```

См. также

strpbrk strspn strstr strtok

2.22.8. STRCMP

```
#include <string.h>

int strcmp(const char *str1, const char *str2);
```

Функция *strcmp()* сравнивает в лексикографическом порядке две строки и возвращает целое значение, зависящее следующим образом от результата сравнения.

Значение	***Результат сравнения строк***
Меньше нуля	str1 меньше str2
Нуль	str1 равен str2
Больше нуля	str1 больше str2

Пример

```
#include <sat500.h>
#include <string.h>

const char Text[]="Это строка";
void main(void)
{
    ClrScr();
    if(strcmp(Text,"это строка"))
    {
        puts("Строки не совпали");
    }
    else
    {
        puts("Строки совпали");
    }
    ShowScr();
    while(1)
    {
    }
}
```

См. также
strchr strcpy

2.22.9. STRCPY

```
#include <string.h>

char *strcpy(char *str1, const char *str2);
```

Функция *strcpy()* копирует содержимое строки ***str2*** в строку ***str1***. Параметр ***str2*** должен указывать на строку с завершающим нулевым символом.
Функция *strcpy()* возвращает значение указателя ***str1***.
Если символьные массивы ***str1*** и ***str2*** перекрываются, поведение функции *strcpy()* не определено.

Пример

Следующий фрагмент программы копирует строку привет в строку str.

```
char str[80];
strcpy(str,"привет");
```

См. также
memcpy strchr strcmp strncmp

2.22.10. STRCSPN

```
#include <string.h>

size_t strcspn(const char *str1, const char *str2);
```

Функция *strcspn()* выполняет поиск первого вхождения в строку ***str1*** любого из символов строки ***str2***, и возвращает количество символов до найденного первого вхождения.
Поиск учитывает и завершающий нуль-символов, поэтому, если функция возвращает длину строки ***str1***, это значит, что ни один из символов строки ***str2*** не входит в состав ***str1***.
Функция *strcspn()* возвращает длину начальной подстроки в строке, адресуемой параметром ***str1***, которая не содержит ни одного символа из строки, адресуемой параметром ***str2***.
Другими словами, функция *strcspn()* возвращает индекс первого символа в строке ***str1***, который совпадает с любым из символов в строке ***str2***.

Пример
Следующая программа выводит число 6

```
#include <sat500.h>
#include <string.h>

void main(void)
{
    int len;
    ClrScr();
    len = strcspn("это тест", "сл");
    printf("%d", len);
    ShowScr();
    while(1)
    {
    }
}
```

См. также
strchr strpbrk strstr strtok

2.22.11. STRLEN

```
#include <string.h>

size_t strlen(const char *str);
```

Функция *strlen()* возвращает длину строки, адресуемой параметром ***str***, причем строка должна заканчиваться символом конца строки. Символ конца строки ('0') не учитывается.

Пример
Данный фрагмент программы выводит на экран число 6

```
printf("%d", strlen("привет"));
```

См. также
memcpy *strchr* *strncmp* *strncmp*

2.22.12. STRNCAT

```
#include <string.h>

char *strncat(char *str1, const char *str2, size_t count);
```

Функция *strncat()* присоединяет к строке, адресуемой параметром ***str1***, не более ***count*** символов строки, адресуемой параметром ***str2***, завершая "результирующую" строку ***str1*** нулевым символом. Конечный нуль-символ, первоначально завершающий строку **str1**, перезаписывается первым символом строки ***str2***. Строка ***str2*** в результате этой операции конкатенации не меняется. Если строки перекрываются, поведение функции *strncat()* не определено.
Функция *strncat()* возвращает значение указателя ***str1***.
При выполнении операций с массивами символов контроль нарушения их границ не выполняется, поэтому программист должен сам позаботиться о достаточном размере массива str1, позволяющем вместить как его исходное содержимое, так и содержимое присоединяемого массива ***str2***.

Пример

Данная программа конкатенирует первую строку ко второй строке и предотвращает переполнение массива s1, в который записывается результат.

```
#include <sat500.h>
#include <string.h>

void main(void)
{
    char s1[80], s2[80];
    unsigned int len;
    ClrScr();
    strcpy(s1,"привет");
    strcpy(s2,"Всем ");
    /* вычисление подходящей длины */
    len = 79-strlen(s2);
    strncat(s2, s1, len);
    printf(s2);
    ShowScr();
    while(1)
    {
    }
}
```

См. также
strcat *strchr* *strncmp* *strncpy*

2.22.13. STRNCMP

```
#include <string.h>

int strncmp(const char *str1, const char *str2, size_t count);
```

Функция *strncmp()* сравнивает в лексикографическом порядке не более ***count*** символов из двух строк, заканчивающихся символом конца строки, и возвращает целое значение, зависящее от результата сравнения следующим образом:

Значение	***Результат сравнения***
Меньше нуля	str1 меньше str2
Нуль	str1 равен str2
Больше нуля	str1 больше str2

Если в какой-нибудь из заданных строк меньше count символов, сравнение заканчивается при обнаружении первого нулевого символа.

Пример
Следующая функция сравнивает первые восемь символов двух своих аргументов, взятых из командной строки, и выдает сообщение в случае их равенства.

```
#include <sat500.h>
#include <string.h>

const char str1[]="Это строка 1";
const char str2[]="Это строка 2";

void main(void)
{
    ClrScr();

    if(!strncmp(str1,str2, 8))
        printf("Строки одинаковые.\n");
    ShowScr();
    while(1)
    {
    }
}
```

См. также
strcmp *strncpy*

2.22.14. STRNCPY

```
#include <string.h>

char *strncpy(char *str1, const char *str2, size_t count);
```

Функция *strncpy()* копирует не более ***count*** символов из строки, адресуемой параметром ***str2***, в массив, адресуемый параметром ***str1***. Параметр ***str2*** должен указывать на строку, заканчивающуюся символом конца строки.
Если заданные массивы символов перекрываются, поведение функции *strncpy()* не определено.

Если длина строки, адресуемой параметром ***str2***, меньше значения ***count***, то в конец строка - результата ***str1*** добавляются "недостающие" нулевые символы.
Если же длина строки, адресуемой параметром ***str2***, больше значения ***count***, то строка - результат, адресуемая параметром ***str1***, не будет заканчиваться символом конца строки.
Функция *strncpy()* возвращает значение указателя ***str1***.

См. также
strcpy

2.22.15. STRPBRK

```
#include <string.h>

char *strpbrk(const char *str1, const char *str2);
```

Функция *strpbrk()* возвращает указатель на первый символ в строке, адресуемой параметром ***str1***, который совпадает с любым символом в строке, адресуемой параметром ***str2***. Символы конца строки, которыми должны оканчиваться данные строки, в расчет не берутся.
Если совпадений нет, возвращается нулевой указатель.

Пример
Данная программа выводит на экран сообщение
```
о тест
```

```
#include <sat500.h>
#include <string.h>

void main(void)
{
    char *p;
    ClrScr();
    p = strpbrk("это тест", " есою");
    printf(p);
    ShowScr();
    while(1)
    {
    }
}
```

См. также
strspn *strrchr* *strstr* *strtok*

2.22.16. STRRCHR

```
#include <string.h>

char *strrchr(const char *str, int ch) ;
```

Функция *strrchr()* возвращает указатель на последнее вхождение младшего байта параметра ***ch*** в строку, адресуемую параметром ***str***.
Если совпадение не обнаружено, возвращается нулевой указатель.

Пример

Данная программа выводит сообщение

```
пыль летит
```

```
#include <sat500.h>
#include <string.h>

void main(void)
{
    char *p;
    ClrScr();
    p = strrchr("по полю пыль летит", 'п');
    printf(p);
    ShowScr();
    while(1)
    {
    }
}
```

См. также
strpbrk strspn strstr strtok

2.22.17. STRSPN

```
#include <string.h>

size_t strspn(const char *str1, const char *str2);
```

Функция *strspn()* возвращает длину начальной подстроки строки, адресуемой параметром ***str1***, которая состоит только из символов, содержащихся в строке, адресуемой параметром ***str2***.
Другими словами, функция *strspn()* возвращает индекс первого символа в строке ***str1***, который не совпадает ни с одним из символов в строке ***str2***.

Пример
Эта программа выводит число
```
11
```

```
#include <sat500.h>
#include <string.h>

void main(void)
{
    int len;
    ClrScr();
    len = strspn("это строка из примера", "акортэ с");
    printf("%d", len);
    ShowScr();
    while(1)
    {
    }
}
```

См. также
strpbrk strrchr strstr strtok

2.22.18. STRSTR

```
#include <string.h>

char *strstr(const char *str1, const char *str2);
```

Функция *strstr()* возвращает указатель на первое вхождение подстроки, адресуемой параметром ***str2***, в строку, адресуемую параметром ***str1***.
Если совпадение не обнаружено, возвращается нулевой указатель.

Пример

Данная программа выводит сообщение
```
от невпроворот
```

```
#include <sat500.h>
#include <string.h>

void main(void)
{
    char *p;
    ClrScr();
    p = strstr("хлопот невпроворот", "от");
    printf(p);
    ShowScr();
    while(1)
    {
    }
}
```

См. также
strchr *strcspn* *strpbrk* *strspn* *strrchr* *strtok*

2.22.19. STRTOK

```
#include <string.h>

char *strtok(char *str1, const char *str2);
```

Функция *strtok()* возвращает указатель на следующую лексему в строке, адресуемой параметром ***str1***. Символы, образующие строку, адресуемую параметром ***str2***, представляют собой разделители, которые определяют лексему. При отсутствии лексемы, подлежащей возврату, возвращается нулевой указатель.

Чтобы разделить некоторую строку на лексемы, при первом вызове функции *strtok()* параметр ***str1*** должен указывать на начало этой строки. При последующих вызовах функции в качестве параметра ***str1*** нужно использовать нулевой указатель. Этим способом вся строка разбивается на лексемы.
При каждом обращении к функции *strtok()* можно использовать различные наборы разделителей.

Пример
Эта программа разбивает строку "Травка зеленеет, солнышко блестит" на лексемы, разделителями которых служат пробелы и запятые. В результате получится
```
Травка|зеленеет|солнышко|блестит
```

```
#include <sat500.h>
#include <string.h>
```

```
void main(void)
{
    char *p;
    ClrScr();
    p = strtok("Травка зеленеет, солнышко блестит", " ");
    printf(p);
    do
    {
        p = strtok('\0', ", ");
        if(p) printf("|%s", p);
    } while(p);
    ShowScr();
    while(1)
    {
    }
}
```

См. также
strchr strcspn strpbrk strrchr strspn

2.22.20. ATOI

```
#include <string.h>

int atoi(const char *s)
```

Функция *atoi* преобразует строку ***s*** в целое значение типа *int*.
Анализируя строку ***s***, *atoi* интерпретирует её содержание, как целое число, которое возвращается как *int*.
Функция сначала отбрасывает символы пробелов до тех пор, пока не будет найден символ отличный от нуля.
Затем, начиная с этого символа, функция принимает необязательный начальный знак плюс или минус.
После чего, следует последовательность цифр, которая интерпретируется в числовое значение.
Строка может содержать другие символы после считанного целого числа, эти символы игнорируются и никак не влияют на поведение этой функции.

Пример

```
#include <sat500.h>
#include <string.h>

char str[]="1234567";

void main(void)
{
    int val;
    ClrScr();
    val=atoi(str);
    printf("val=%d",val);
    ShowScr();
    while(1);
}
```

2.23. CTYPE.H

ctype.h - заголовочный файл стандартной библиотеки языка программирования C, содержащий объявления функций для классификации и преобразования отдельных символов.

Имя	***Описание***
int isalnum (int c);	Если аргумент функции является либо буквой, либо цифрой, она возвращает ненулевое значение
int isalpha (int c);	Возвращает ненулевое значение, если её аргумент является буквой, в противном случае возвращается нуль
int isblank (int c);	Возвращает true, если c - пробел или горизонтальная табуляция
int iscntrl (int c);	Возвращает true, если c - управляющий символ, такой как <Ctrl+B>
int isdigit (int c);	Возвращает ненулевое значение, если её аргумент является десятичной цифрой, в противном случае возвращается нуль
int isgraph (int c);	Возвращает true, если c - печатаемый символ, отличный от пробела
int islower (int c);	Возвращает true, если c - символ нижнего регистра
int isprint (int c);	Возвращает true, если c — печатаемый символ
int ispunct (int c);	Возвращает true, если c - знак препинания (любой печатаемый символ, отличный от пробела или алфавитно-цифрового символа)
int isspace (int c);	Возвращает true, если c — пробельный символ: пробел, новая строка,перевод страницы, возврат каретки, вертикальная табуляция, горизонтальная табуляция или, возможно, другой символ, определяемый реализацией
int isupper (int c);	Возвращает true, если c - символ верхнего регистра
int isxdigit (int c);	Возвращает true, если c — шестнадцатеричная цифра
int toupper (int c);	Переводит букву нижнего регистра в верхний регистр
int tolower (int c);	Переводит букву верхнего регистра в нижний регистр

2.23.1. ISALNUM

```
#include <ctype.h>

int isalnum(int ch);
```

Если аргумент ***ch*** функции *isalnum()* является либо буквой, либо цифрой, она возвращает ненулевое значение. Если же тестируемый символ не относится к алфавитно-цифровым, возвращается нуль.

Пример

Данная программа читает из массива символы, проверяет их и выдает сообщение о каждом алфавитно-цифровом символе.

```
#include <sat500.h>
#include <ctype.h>

char str[]="ASqw12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
    i=0;
    while(str[i])
    {
        ch = str[i];
        if(isalnum(ch))printf("Символ %c - алфавитно-цифровой\n", ch);
    }
    ShowScr();
    ...
}
```

См. также
isalpha *isdigit* *isgraph* *ispunct* *isspace*

2.23.2. ISALPHA

```
#include  <ctype.h>

int isalpha(int ch);
```

Функция *isalpha()* возвращает ненулевое значение, если ее аргумент ***ch*** является буквой, в противном случае возвращается нуль. Принадлежность символа к буквам зависит от конкретного языка. Для английского языка таковыми являются прописные и строчные буквы от A до Z.

Пример
Данная программа делает проверку каждого символа из массива ***str***, и выдает сообщение, если этот символ окажется буквой.

```
#include <sat500.h>
#include <ctype.h>

char str[]="ASqw12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
    i=0;
    while(str[i])
    {
        ch = str[i];
        if(isalpha(ch))printf("Символ %c - буква\n", ch);
    }
```

```
    ShowScr();
    ...
}
```

См. также
isalnum *iscntrl* *isdigit* *isgraph* *isprint* *ispunct* *isspace*

2.23.3. ISBLANK

```
#include <ctype.h>

int isblank(int ch);
```

Функция isblank() возвращает ненулевое значение, если ее аргумент ***ch*** является символом, для которого функция *isspace()* возвращает значение "истина".
Этот символ используется в качестве разделителя слов. Так, для английского языка пробельными символами являются пробел и символ горизонтальной табуляции.

Пример
Данная программа проверяет все символы массива str, и выдает сообщение о каждом пробельном символе.

```
#include <sat500.h>
#include <ctype.h>

char str[]="AS qw\t12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
    i=0;
    while(str[i])
    {
        ch = str[i];
        if(isblank(ch))printf("Символ %c - разделитель\n", ch);
    }
    ShowScr();
    ...
}
```

См. также
isalnum *isalpha* *iscntrl* *isdigit* *isgraph* *ispunct* *isspace*

2.23.4. ISCNTRL

```
#include <ctype.h>

int iscntrl(int ch);
```

Функция *iscntrl()* возвращает ненулевое значение, если ее аргумент ***ch*** является управляющим символом, значение которого в ASCII-средах лежит в диапазоне между нулем и 0x1F или равно 0x7F (символ DEL).
В противном случае возвращается нуль.

Пример
Данная программа проверяет все символы, прочитанные из массива str, и выдает сообщение о каждом управляющем символе.

```
#include <sat500.h>
#include <ctype.h>

char str[]="AS qw\t12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
    i=0;
    while(str[i])
    {
        ch = str[i];
        if(iscntrl(ch))printf("Символ %c - управляющий\n", ch);
    }
    ShowScr();
    ...
}
```

См. также
isalnum *isalpha* *isdigit* *isgraph* *isprint* *ispunct* *isspace*

2.23.5. ISDIGIT

```
#include <ctype.h>

int isdigit(int ch);
```

Функция *isdigit()* возвращает ненулевое значение, если ее аргумент ***ch*** является цифрой, т.е. попадает в диапазон 0-9. В противном случае возвращается нуль.

Пример
Данная программа проверяет каждый символ, прочитанный из массива ***str***, и выдает сообщение, если этот символ окажется цифрой.

```
#include <sat500.h>
#include <ctype.h>

char str[]="AS qw\t12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
    i=0;
```

```
    while(str[i])
    {
        ch = str[i];
        if(isdigit(ch))printf("Символ %c - цифра\n", ch);
    }
    ShowScr();
    ...
}
```

См. также
isalnum isalpha iscntrl isgraph isprint ispunct isspace

2.23.6. ISGRAPH

```
#include <ctype.h>

int isgraph(int ch);
```

Функция *isgraph()* возвращает ненулевое значение, если ее аргумент ***ch*** является любым печатаемым символом, но не пробелом. В противном случае возвращается нуль. Для ASCII-сред значения печатаемых символов лежат в диапазоне от 0x21 до 0x7E.
Пример
Данная программа проверяет все символы, прочитанные из массива ***str***, и выдает сообщение о каждом печатаемом символе.

```
#include <sat500.h>
#include <ctype.h>

char str[]="AS qw\t12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
    i=0;
    while(str[i])
    {
        ch = str[i];
        if(isgraph(ch))printf("Символ %c - печатаемый\n", ch);
    }
    ShowScr();
    ...
}
```

См. также
isalnum isalpha iscntrl isdigit isprint ispunct isspace

2.23.7. ISLOWER

```
#include <ctype.h>

int islower(int ch);
```

Функция *islower()* возвращает ненулевое значение, если аргумент ***ch*** является строчной буквой. В противном случае возвращается нуль.

Пример
Данная программа проверяет все символы, прочитанные из массива ***str***, и выдает сообщение о каждой строчной букве.

```
#include <sat500.h>
#include <ctype.h>

char str[]="AS qw\t12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
    i=0;
    while(str[i])
    {
        ch = str[i];
        if(islower(ch))printf("%c - строчная буква\n", ch);
    }
    ShowScr();
    ...
}
```

См. также
isupper

2.23.8. ISPRINT

```
#include <ctype.h>

int isprint(int ch);
```

Функция *isprint()* возвращает ненулевое значение, если аргумент ***ch*** является печатаемым символом, включая пробел. В противном случае возвращается нуль.
В ASCII-средах значения печатаемых символов лежат в диапазоне от 0x20 до 0x7E.

Пример
Данная программа проверяет все символы, прочитанные из массива ***str***, и выдает сообщение о каждом печатаемом символе.

```
#include <sat500.h>
#include <ctype.h>

char str[]="AS qw\t12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
    i=0;
    while(str[i])
```

```
    {
        ch = str[i];
        if(isprint(ch))printf("%c - печатаемый символ\n", ch);
    }
    ShowScr();
    ...
}
```

См. также
isalnum isalpha iscntrl isdigit isgraph ispunct isspace

2.23.9. ISPUNCT

```
#include <ctype.h>

int ispunct(int ch);
```

Функция *ispunct()* возвращает ненулевое значение, если аргумент ***ch*** является знаком пунктуации. В противном случае возвращается нуль.
Под знаками пунктуации подразумеваются все печатаемые символы за исключением пробела, которые не относятся к алфавитно-цифровым.

Пример
Данная программа проверяет все символы, прочитанные из массива ***str***, и выдает сообщение о каждом знаке пунктуации.

```
#include <sat500.h>
#include <ctype.h>

char str[]="AS qw\t12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
    i=0;
    while(str[i])
    {
        ch = str[i];
        if(ispunct(ch))printf("%c - знак пунктуации\n", ch);
    }
    ShowScr();
    ...
}
```

См. также
isalnum isalpha iscntrl isdigit isgraph isspace

2.23.10. ISSPACE

```
#include <ctype.h>

int isspace(int ch);
```

Функция *isspace()* возвращает ненулевое значение, если аргумент ***ch*** является пробельным символом. (К пробельным символам, помимо пробела, относятся символы горизонтальной и вертикальной табуляции, перевода страницы, возврата каретки и новой строки.)
В противном случае возвращается нуль.

Пример
Данная программа проверяет все символы, прочитанные из массива ***str***, и выдает сообщение о каждом пробельном символе.

```
#include <sat500.h>
#include <ctype.h>

char str[]="AS qw\t12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
    i=0;
    while(str[i])
    {
        ch = str[i];
        if(isspace(ch))printf("%c - пробельный символ\n", ch);
    }
    ShowScr();
    ...
}
```

См. также
isalnum *isalpha* *isblank* *iscntrl* *isdigit* *isgraph* *ispunct*

2.23.11. ISUPPER

```
#include <ctype.h>

int isupper(int ch);
```

Функция *isupper()* возвращает ненулевое значение, если аргумент ***ch*** является прописной буквой.
В противном случае возвращается нуль.

Пример
Данная программа проверяет все символы, прочитанные из массива ***str***, и выдает сообщение о каждой прописной букве.

```
#include <sat500.h>
#include <ctype.h>

char str[]="AS qw\t12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
```

```
    i=0;
    while(str[i])
    {
        ch = str[i];
        if(isupper(ch))printf("%c - прописная буква\n", ch);
    }
    ShowScr();
    ...
}
```

См. также
islower

2.23.12. ISXDIGIT

```
#include <ctype.h>

int isxdigit(int ch);
```

Функция *isxdigit()* возвращает ненулевое значение, если аргумент ***ch*** является шестнадцатеричной цифрой.
В противном случае возвращается нуль. Шестнадцатеричная цифра должна попадать в один из следующих диапазонов: A-F, a-f или 0-9.

Пример
Данная программа проверяет все символы, прочитанные из массива ***str***, и выдает сообщение о каждой шестнадцатеричной цифре.

```
#include <sat500.h>
#include <ctype.h>

char str[]="ABCS def\t12!&\xFF\r\n";

void main(void)
{
    char ch;
    int i;
    ClrScr();
    i=0;
    while(str[i])
    {
        ch = str[i];
        if(isxdigit(ch))printf("%c - шестнадцатеричная цифра\n", ch);
    }
    ShowScr();
    ...
}
```

См. также
isalnum *isalpha* *iscntrl* *isdigit* *isgraph* *ispunct* *isspace*

2.23.13. TOUPPER

```
#include <ctype.h>
```

```
int toupper(int ch);
```

Функция *toupper()* возвращает прописной эквивалент параметра ***ch***, если ***ch*** - буква, в противном случае ***ch*** возвращается без изменений.

Пример
Этот фрагмент программы выводит букву
```
A
```

```
putchar(toupper('a'));
```

См. также
tolower

2.23.14. TOLOWER

```
#include <ctype.h>

int tolower(int ch) ;
```

Функция *tolower()* возвращает строчный эквивалент параметра ***ch***, если он является буквой, в противном случае возвращается ***ch*** без изменений.

Пример
Этот фрагмент программы выводит букву
```
a
```

```
putchar(tolower('A'));
```

См. также
toupper

2.24. MATH.H

math.h - заголовочный файл библиотеки языка Си, содержащий математические функции.
ПРИМЕЧАНИЕ: Функции данной библиотеки отличаются от функций стандартной библиотеки языка С.

Доступны следующие функции:

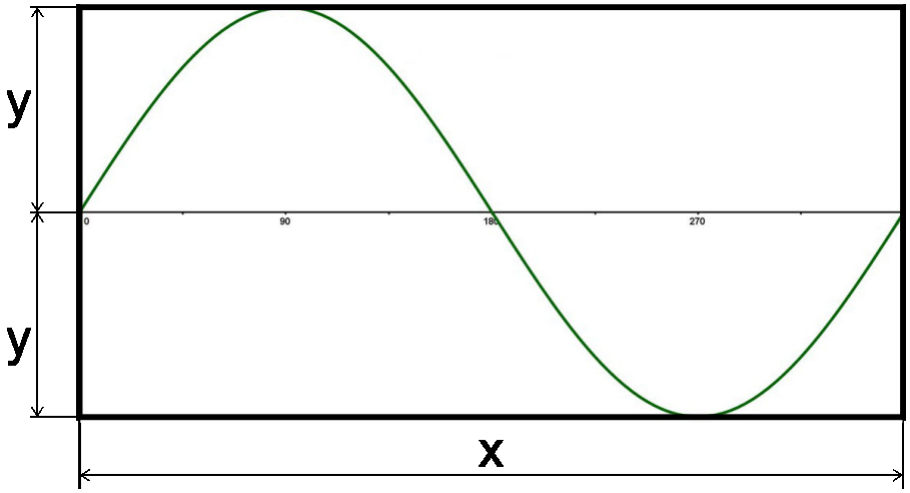
Имя	***Описание***
int sin (int pos,unsigned int x,unsigned int y)	Вычисление синуса
int cos(int pos,unsigned int x,unsigned int y)	Вычисление косинуса
int tan (int pos,unsigned int x,unsigned int y)	Вычисление тангенса
int asin (int x,unsigned int divx,unsigned int muly)	Вычисление обратного синуса
int acos (int x,unsigned int divx,unsigned int muly)	Вычисление обратного косинуса
int atan (int x,unsigned int divx,unsigned int muly)	Вычисление обратного тангенса
int sinh (int x,unsigned int divx,unsigned int muly)	Вычисление гиперболического синуса
int cosh (int x,unsigned int divx,unsigned int muly)	Вычисление гиперболического косинуса

int tanh (int x,unsigned int divx,unsigned int muly)	Вычисление гиперболического тангенса
int pow (int x,unsigned int divx,int y,unsigned int divy,unsigned int mul)	Вычисление числа, возведённого в степень
unsigned int log (unsigned int x,unsigned int divx,unsigned int muly)	Вычисление натурального логарифма
unsigned int log10 (unsigned int x,unsigned int divx,unsigned int muly)	Вычисление десятичного логарифма
unsigned int sqrt (unsigned int x,unsigned int divx,unsigned int muly)	Вычисление квадратного корня
int exp (int x,unsigned int divx,unsigned int muly)	Вычисление значение экспоненты

2.24.1. SIN

```
#include <math.h>

int sin(int pos,unsigned int x,unsigned int y);
```

Функция *sin()* позволяет вычислить синус аргумента *pos*.
int pos - значение аргумента вычисления функции *sin()*.
unsigned int x – значение масштаба аргумента. Задаваемое значение x соответствует 360 градусам или 2*PI (см.рисунок ниже).
unsigned int y – масштабный множитель результата.



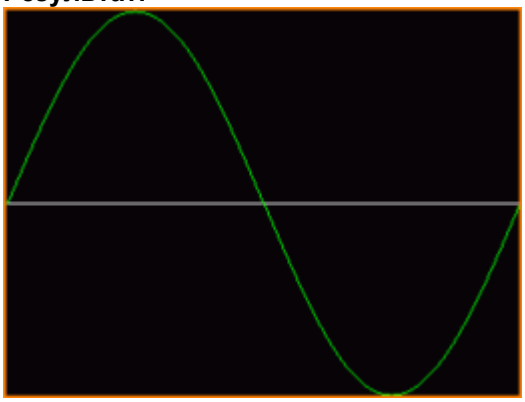
Функция *sin()* возвращает значение в пределах: *-y ...+y*.

Пример
Данная программа рисует на дисплее бегущий "синус":

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int f=0;
    while(1)
    {
        ClrScr();
        PenWidth=1;
        PenColor=COLOR_WHITE;
```

```
        GotoXY(0,120);
        LineTo(319,120);
        PenColor=COLOR_GREEN;
        //Sinus
        for(int i=0;i<320;i++)
        {
            int pos=i+f;
            if(pos>=320)pos-=320;
            int v=sin(pos,320,120);
            if(!i)GotoXY(0,120-v);
            else LineTo(i,120-v);
        }
        f+=3;
        ShowScr();
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100)
        {
        }
    }
}
```

Результат:



См. также
cos tan

2.24.2. COS

```
#include <math.h>

int cos(int pos,unsigned int x,unsigned int y);
```

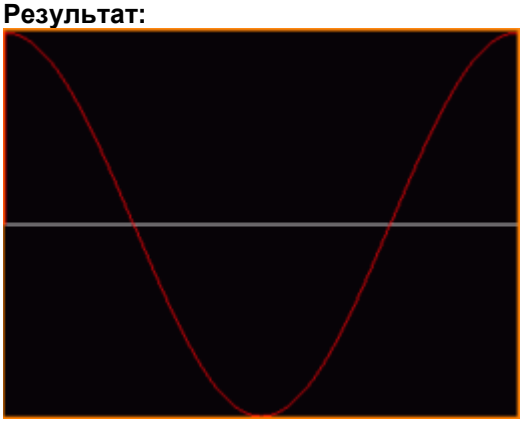
Функция *cos()* позволяет вычислить косинус аргумента *pos*.
int pos - значение аргумента вычисления функции *cos()*.
unsigned int x – значение масштаба аргумента. Задаваемое значение *x* соответствует 360 градусам или 2*PI.
unsigned int y – масштабный множитель результата.
Функция *cos()* возвращает значение в пределах: *-y ...+y*.

Пример
Данная программа рисует на дисплее функцию косинуса:

```
#include <sat500.h>
#include <math.h>
```

```
void main(void)
{
    ClrScr();
    PenWidth=1;
    PenColor=COLOR_WHITE;
    GotoXY(0,120);
    LineTo(319,120);
    //Cosinus
    PenColor=COLOR_RED;
    GotoXY(0,120);
    for(int i=0;i<320;i++)
    {
        int v=cos(i,320,120);
        if(!i)GotoXY(0,120-v);
        else  LineTo(i,120-v);
    }
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
sin *tan*

2.24.3. TAN

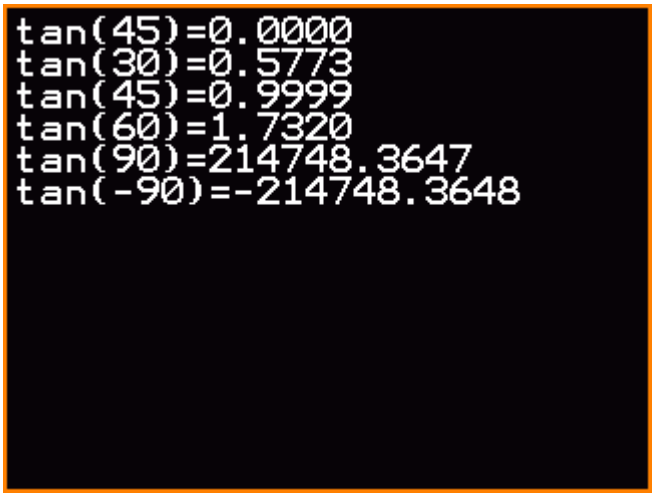
```
#include <math.h>

int tan(int pos,unsigned int x,unsigned int y);
```

Функция tan*()* позволяет вычислить тангенс аргумента *pos*.
int pos - значение аргумента вычисления функции *tan()*.
unsigned int x – значение масштаба аргумента. Задаваемое значение x соответствует 360 градусам или 2*PI.
unsigned int y – масштабный множитель результата.

Пример
Данная программа вычисляет тангенс углов 0, 30, 45, 60, 90, -90 градусов:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    v=tan(0,360,10000);
    printf("tan(45)=%d.%04d\n",v/10000,abs(v%10000));
    v=tan(30,360,10000);
    printf("tan(30)=%d.%04d\n",v/10000,abs(v%10000));
    v=tan(45,360,10000);
    printf("tan(45)=%d.%04d\n",v/10000,abs(v%10000));
    v=tan(60,360,10000);
    printf("tan(60)=%d.%04d\n",v/10000,abs(v%10000));
    //!!!!!!!!!!!!!!!!!!!!
    v=tan(90,360,10000);
    printf("tan(90)=%d.%04d\n",v/10000,abs(v%10000));
    //!!!!!!!!!!!!!!!!!!!!
    v=tan(-90,360,10000);
    printf("tan(-90)=%d.%04d\n",v/10000,abs(v%10000));
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
sin *cos*

2.24.4. ASIN

```
#include <math.h>

int asin(int x,unsigned int divx,unsigned int muly);
```

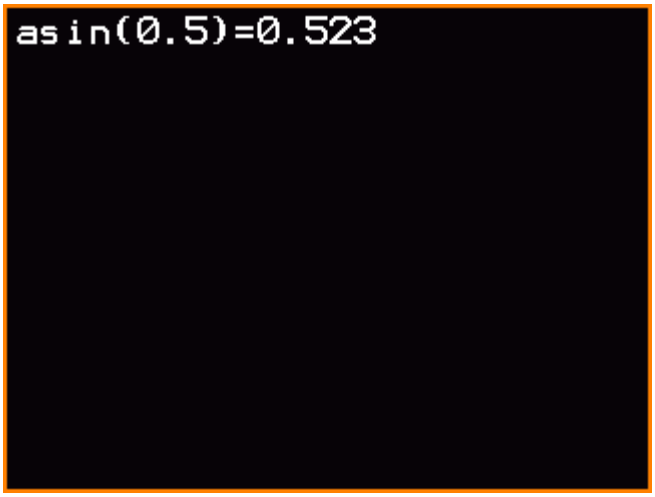
Функция *asin()* возвращает обратный синус числа в радианах.

Значения выражения `x/divx` должно находится в диапазоне [-1, 1]. Это потому, что значение синуса находится в диапазоне от 1 до -1.
int x - значение аргумента вычисления функции *asin*.
unsigned int divx - делитель *x*.
unsigned int muly – масштабный множитель результата.
Результат, полученный при вычислении *ARCSIN(x/divx)* умножается на масштабный множитель *muly* и полученное значение возвращается как результат функции в виде целого знакового числа.

Пример

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    ClrScr();
    BigFont=false;
    int v=asin(5,10,1000);
    printf("asin(0.5)=%d.%03d\n",v/1000,abs(v%1000));
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
acos *atan*

2.24.5. ACOS

```
#include <math.h>

int acos(int x,unsigned int divx,unsigned int muly);
```

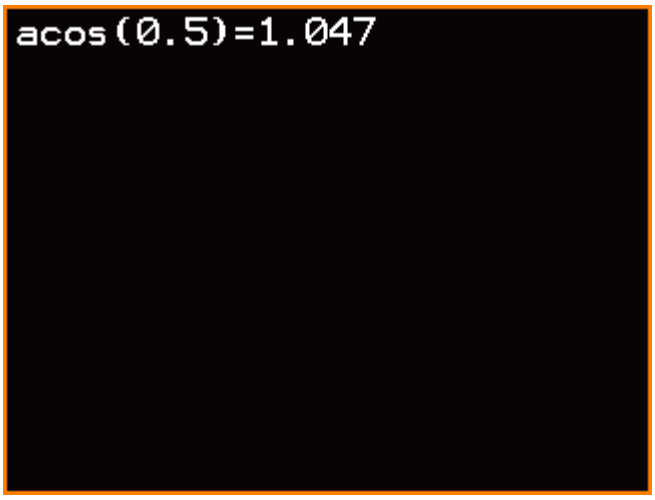
Функция *acos()* возвращает обратный косинус числа в радианах.
Значения выражения `x/divx` должно находится в диапазоне [-1, 1]. Это потому, что значение косинуса находится в диапазоне от 1 до -1.
int x - значение аргумента вычисления функции *acos*.

unsigned int divx - делитель *x*.
unsigned int muly – масштабный множитель результата.
Результат, полученный при вычислении *ARCCOS(x/divx)* умножается на масштабный множитель *muly* и полученное значение возвращается как результат функции в виде целого знакового числа.

Пример

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    ClrScr();
    BigFont=false;
    int v=acos(5,10,1000);
    printf("acos(0.5)=%d.%03d\n",v/1000,abs(v%1000));
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
asin *atan*

2.24.6. ATAN

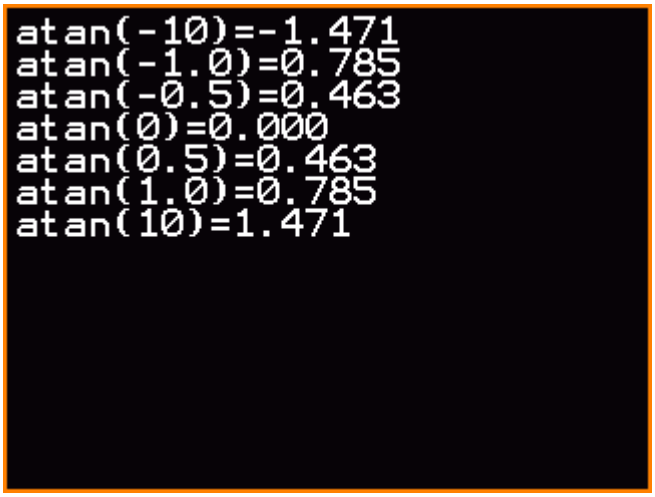
```
#include <math.h>

int atan(int x,unsigned int divx,unsigned int muly);
```

Функция *atan()* возвращает арктангенс числа (аргумента) в радианах.
int x - значение аргумента вычисления функции *atan*.
unsigned int divx - делитель *x*.
unsigned int muly – масштабный множитель результата.
Результат, полученный при вычислении *ARCTG(x/divx)* умножается на масштабный множитель *muly* и полученное значение возвращается как результат функции в виде целого знакового числа.

Пример

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    v=atan(-10,1,1000);
    printf("atan(-10)=%d.%03d\n",v/1000,abs(v%1000));
    v=atan(-1,1,1000);
    printf("atan(-1.0)=%d.%03d\n",v/1000,abs(v%1000));
    v=atan(-5,10,1000);
    printf("atan(-0.5)=%d.%03d\n",v/1000,abs(v%1000));
    v=atan(0,1,1000);
    printf("atan(0)=%d.%03d\n",v/1000,abs(v%1000));
    v=atan(5,10,1000);
    printf("atan(0.5)=%d.%03d\n",v/1000,abs(v%1000));
    v=atan(1,1,1000);
    printf("atan(1.0)=%d.%03d\n",v/1000,abs(v%1000));
    v=atan(10,1,1000);
    printf("atan(10)=%d.%03d\n",v/1000,abs(v%1000));
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
asin *acos*

2.24.7. SINH

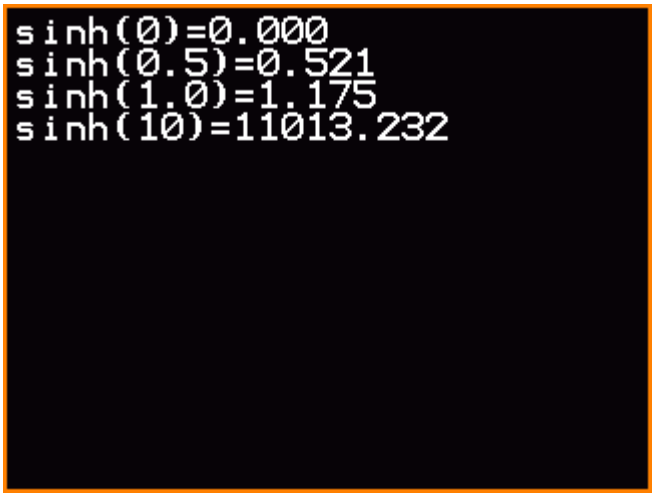
```
#include <math.h>

int sinh(int x,unsigned int divx,unsigned int muly);
```

Функция *sinh()* возвращает гиперболический синус угла в радианах.
int x - значение аргумента вычисления функции *sinh*.
unsigned int divx - делитель *x*.
unsigned int muly – масштабный множитель результата.
Результат, полученный при вычислении *SINH(x/divx)* умножается на масштабный множитель *muly* и полученное значение возвращается как результат функции в виде целого знакового числа.

Пример

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    v=sinh(0,1,1000);
    printf("sinh(0)=%d.%03d\n",v/1000,abs(v%1000));
    v=sinh(5,10,1000);
    printf("sinh(0.5)=%d.%03d\n",v/1000,abs(v%1000));
    v=sinh(1,1,1000);
    printf("sinh(1.0)=%d.%03d\n",v/1000,abs(v%1000));
    v=sinh(10,1,1000);
    printf("sinh(10)=%d.%03d\n",v/1000,abs(v%1000));
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
cosh tanh

2.24.8. COSH

```
#include <math.h>
```

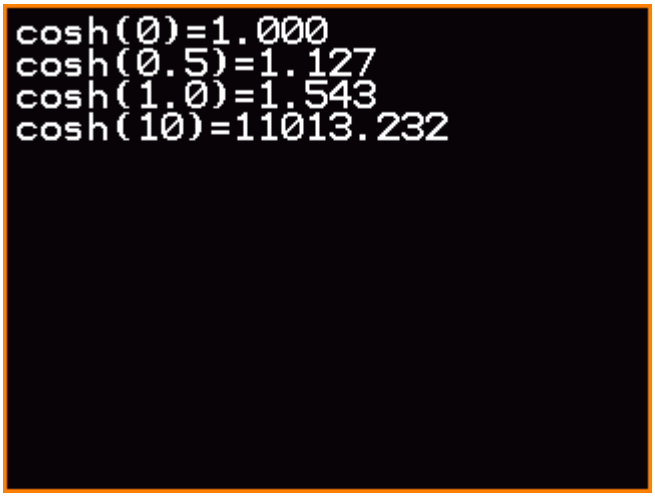
```
int cosh(int x,unsigned int divx,unsigned int muly);
```

Функция `cosh()` возвращает гиперболический косинус угла в радианах.
int x - значение аргумента вычисления функции `cosh`.
unsigned int divx - делитель *x*.
unsigned int muly – масштабный множитель результата.
Результат, полученный при вычислении *COSH(x/divx)* умножается на масштабный множитель *muly* и полученное значение возвращается как результат функции в виде целого знакового числа.

Пример

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    v=cosh(0,1,1000);
    printf("cosh(0)=%d.%03d\n",v/1000,abs(v%1000));
    v=cosh(5,10,1000);
    printf("cosh(0.5)=%d.%03d\n",v/1000,abs(v%1000));
    v=cosh(1,1,1000);
    printf("cosh(1.0)=%d.%03d\n",v/1000,abs(v%1000));
    v=cosh(10,1,1000);
    printf("cosh(10)=%d.%03d\n",v/1000,abs(v%1000));
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
sinh *tanh*

2.24.9. TANH

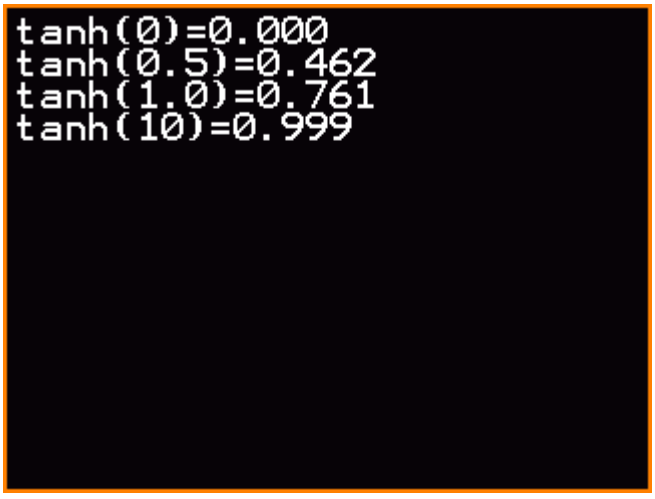
```
#include <math.h>

int tanh(int x,unsigned int divx,unsigned int muly);
```

Функция *tanh()* возвращает гиперболический тангенс угла в радианах.
int x - значение аргумента вычисления функции *tanh*.
unsigned int divx - делитель *x*.
unsigned int muly – масштабный множитель результата.
Результат, полученный при вычислении *TGH(x/divx)* умножается на масштабный множитель *muly* и полученное значение возвращается как результат функции в виде целого знакового числа.

Пример

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    v=tanh(0,1,1000);
    printf("tanh(0)=%d.%03d\n",v/1000,abs(v%1000));
    v=tanh(5,10,1000);
    printf("tanh(0.5)=%d.%03d\n",v/1000,abs(v%1000));
    v=tanh(1,1,1000);
    printf("tanh(1.0)=%d.%03d\n",v/1000,abs(v%1000));
    v=tanh(10,1,1000);
    printf("tanh(10)=%d.%03d\n",v/1000,abs(v%1000));
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
sinh *cosh*

2.24.10. POW

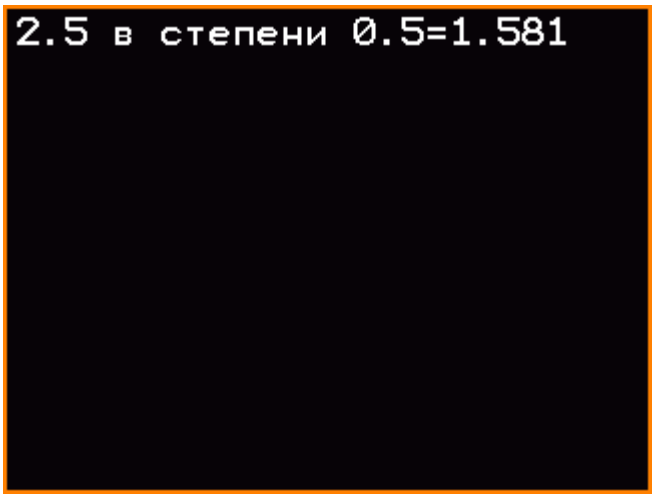
```
#include <math.h>

int pow(int x,unsigned int divx,int y,unsigned int divy,unsigned int mul);
```

Функция pow() вычисляет базовое число *x/divx*, возведенное в степень *y/divy*.
int x - значение аргумента вычисления функции *pov*.
unsigned int divx - делитель *x*.
int y - значение степени.
unsigned int divy - делитель y.
unsigned int mul – масштабный множитель результата.
Результат, полученный при возведении плавающего числа *x/divx* в степень *y/divy* умножается на масштабный множитель *mul* и полученное значение возвращается как результат функции в виде целого знакового числа.

Пример

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    ClrScr();
    BigFont=false;
    int v=pow(25,10,5,10,1000);
    printf("2.5 в степени 0.5=%d.%03d\n",v/1000,abs(v%1000));
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
sqrt

2.24.11. LOG

```
#include <math.h>

unsigned int log(unsigned int x,unsigned int divx,unsigned int muly);
```

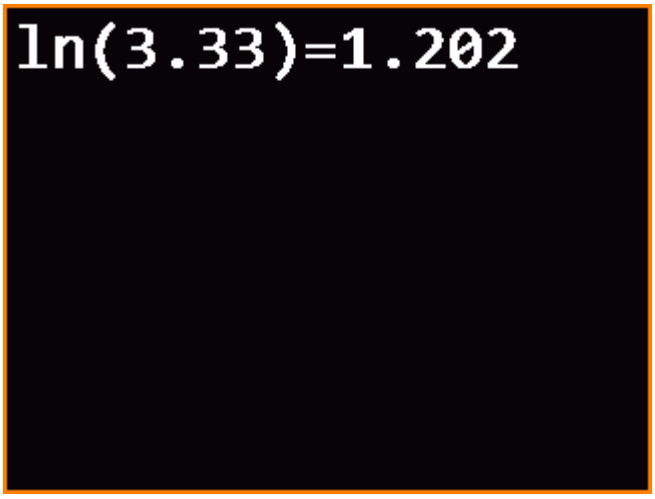
Функция вычисляет натуральный логарифм числа *x/divx*.
unsigned int x - значение аргумента вычисления функции *log()*.
unsigned int divx - делитель *x*.
unsigned int mul - масштабный множитель результата.

ПРИМЕЧАНИЕ
Не допускается в качестве аргумента v функции передавать нулевое значение.
Тем не менее, если число *v=0*, то возвращается заведомо неверное нулевое значение.

Пример
Данная программа вычисляет натуральный логарифм числа 3.33 с точностью 1/1000:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    ClrScr();
    int v=log(333,100,1000);
    printf("ln(3.33)=%d.%03d",v/1000,v%1000);
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
log10 exp

2.24.12. LOG10

```
#include <math.h>

unsigned int log10(unsigned int x,unsigned int divx,unsigned int muly);
```

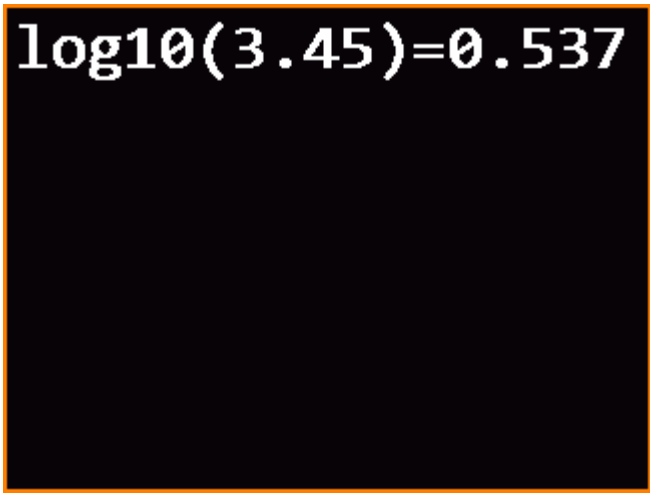
Функция вычисляет десятичный логарифм числа *x/divx*.
unsigned int v - значение аргумента вычисления функции *log()*.
unsigned int divx - делитель *x*.
unsigned int mul - масштабный множитель результата.

ПРИМЕЧАНИЕ
Не допускается в качестве аргумента v функции передавать нулевое значение.
Тем не менее если число *v=0*, то возвращается заведомо неверное нулевое значение.

Пример
Данная программа вычисляет десятичный логарифм числа 3.45 с точностью 1/1000:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    ClrScr();
    int v=log10(345,100,1000);
    printf("log10(3.45)=%d.%03d",v/1000,v%1000);
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
log exp

2.24.13. EXP

```
#include <math.h>

int exp(int x,unsigned int divx,unsigned int muly);
```

Функция exp вычисляет значение экспоненты от *x/divx* и возвращает результат умноженный на *muly*.

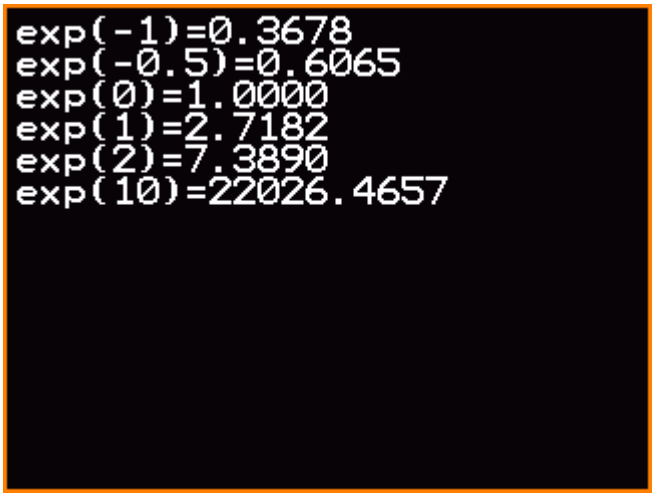
Возвращается цело число *y*, рассчитанное по формуле: *y=EXP(x/divx)*muly*

Выполнение функции в контроллере:
Сначала выполняется преобразование аргументов *x* и *divx* следующим образом:
double v=(double)x/(double)divx;
Затем выполняется собственно вычисление значение экспоненты от *v* и умножение на коэффициент *muly*:
double result=EXP(v)(double)muly;*
Возвращается целочисленное значение результата result.

Пример

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    v=exp(-1,1,10000);
    printf("exp(-1)=%d.%04d\n",v/10000,abs(v%10000));
    v=exp(-5,10,10000);
    printf("exp(-0.5)=%d.%04d\n",v/10000,abs(v%10000));
    v=exp(0,1,10000);
    printf("exp(0)=%d.%04d\n",v/10000,abs(v%10000));
    v=exp(1,1,10000);
    printf("exp(1)=%d.%04d\n",v/10000,abs(v%10000));
    v=exp(2,1,10000);
    printf("exp(2)=%d.%04d\n",v/10000,abs(v%10000));
    v=exp(10,1,10000);
    printf("exp(10)=%d.%04d\n",v/10000,abs(v%10000));
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также

log log10

2.24.14. SQRT

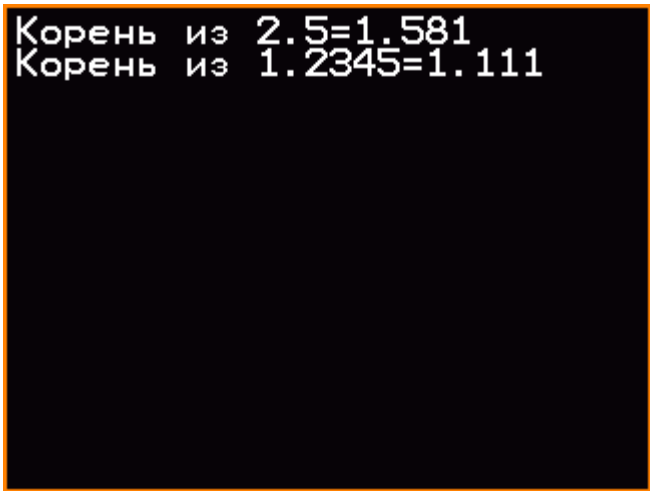
```
#include <math.h>

unsigned int sqrt(unsigned int x,unsigned int divx,unsigned int muly);
```

Функция *sqrt()* позволяет вычислить квадратный корень аргумента *x/divx*.
int x - значение аргумента вычисления функции *sqrt*.
unsigned int divx - делитель *x*.
unsigned int muly – масштабный множитель результата.
Результат, полученный при вычислении квадратного корня из *x/divx* умножается на масштабный множитель *muly* и полученное значение возвращается как результат функции в виде целого знакового числа.

Пример

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    v=sqrt(25,10,1000);
    printf("Корень из 2.5=%d.%03d\n",v/1000,abs(v%1000));
    v=sqrt(12345,10000,1000);
    printf("Корень из 1.2345=%d.%03d\n",v/1000,abs(v%1000));
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
pow

2.25. LIMITS.H

`limits.h` - заголовочный файл стандартной библиотеки общего назначения языка программирования Си, который включает определения характеристик общих типов переменных.
Содержит определение следующих констант:

Имя	***Описание***	***Значение***
CHAR_BIT	Число бит в байте	8
SCHAR_MIN	Минимальное значение для знакового char	−128
SCHAR_MAX	Максимальное значение для знакового char	+127
UCHAR_MAX	Максимальное значение для беззнакового char	+255
CHAR_MIN	Минимальное значение для char	−128
CHAR_MAX	Максимальное значение для char	+127
SHRT_MIN	Минимальное значение для short int	−32768
SHRT_MAX	Максимальное значение для short int	+32767
USHRT_MAX	Максимальное значение для беззнакового short int	+65535
INT_MIN	Минимальное значение для int	−2147483648
INT_MAX	Максимальное значение для int	+2147483647
UINT_MAX	Максимальное значение для беззнакового int	+4294967295
LONG_MIN	Минимальное значение для long int	−2147483648
LONG_MAX	Максимальное значение для long int	+2147483647
ULONG_MAX	Максимальное значение для беззнакового long int	+4294967295

3. CAT-500

В данном разделе описываются функции и переменные, обеспечивающие доступ программы на языке С-Скрипт к аппаратным возможностям контроллера CAT-500.

3.1. ВХОДЫ И ВЫХОДЫ CAT-500

Для доступа к входам и выходам контроллера CAT-500 необходимо включить заголовочный файл *<sat500.h>* следующим образом.

```
#include <sat500.h>
```

В заголовочном файле *<sat500.h>* определены специальные переменные:

Переменная	***Описание***
char DI[8]	входы DI1...DI8 значение 1 - вход замкнут значение 0 - вход разомкнут
int T[5]	входы T1...T5 значение температуры в сотых долях градуса
int AI1	вход AI1
int AI2	вход AI2
char DO[11]	выходы DO1...DO11 значение 1 - выход замкнут значение 0 - выход разомкнут
int AO1	выход AO1

int AO2	выход AO2

3.1.1. ВХОДЫ САТ-500

Для доступа к входам контроллера САТ-500 в заголовочном файле *<sat500.h>* определены специальные переменные:

Переменная	*Описание*
char DI[10]	Входы DI1...DI10 Значение 1 - вход замкнут Значение 0 - вход разомкнут
int T[5]	Входы T1...T5 Значение температуры в сотых долях градуса
int AI1	Аналоговый вход AI1 Значение 1000 соответствует напряжению на входе 10 Вольт
int AI2	Аналоговый вход AI2 Значение 1000 соответствует напряжению на входе 10 Вольт

При получении значений температуры и аналоговых входов могут возникать ошибки. Достоверность значений можно получить, прочитав переменную *ADCErrors*.

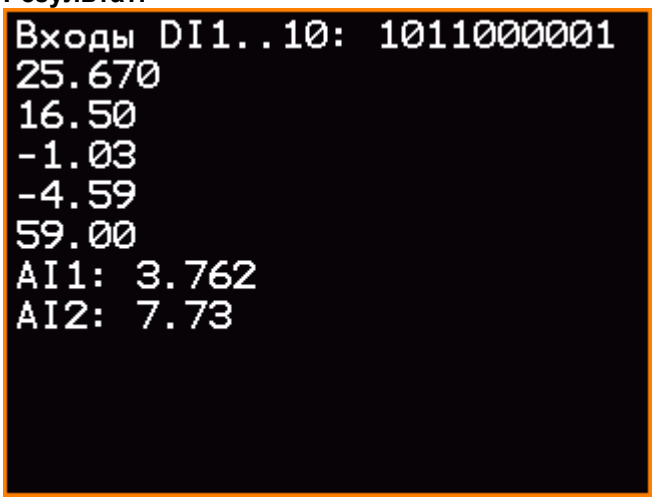
ВНИМАНИЕ! Все входные переменные доступны только для чтения. При выполнении записи в переменные никакие действия не выполняются!

Пример:

```
#include <sat500.h>

void PrintDec2(int v);
void PauseMs(unsigned int ms);

void main(void)
{
    BrushColor=COLOR_BLACK;          //Цвет кисти - чёрный
    ClrScr();                        //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;           //Цвет шрифта - белый
    BigFont=0;                       //Установить обычный шрифт (12x16, 12 пик-
селей  по горизонтали, 16 пикселей  по вертикали)
    while(1)
    {
        GotoXY(4,4);                 //Курсор в позицию x=4,  y=4
        printf("Входы DI1..10: ");    //Вывод на дисплей заголовка
        for(char i=0;i<10;i++)
        {
            if(DI[i])putchar('1');  //Вывод на дисплей состояния входа
DI1...DI10
            else  putchar('0');
        }
        //Вывод на дисплей температур T1...5
        for(char i=0;i<5;i++)
        {
            GotoXY(4,24+20*i);       //Курсор в позицию x=4,  y=24 +20*i
            //Вывод на дисплей температуры Ti
            printf("T%d: ",i+1);
```

```
            PrintDec2(T[i]);
        }
        GotoXY(4,124);              //Курсор в позицию x=4,  y=124
        //Вывод на дисплей значения входа AI1
        puts("AI1: ");
        PrintDec2(AI1);
        GotoXY(4,144);              //Курсор в позицию x=4,  y=144
        //Вывод на дисплей значения входа AI2
        puts("AI2: ");
        PrintDec2(AI2);
        ShowScr();                  //Вывести на дисплей
        //Пауза 200 мс
        PauseMs(200);
    }
}
//*********************************************************************
//Функция выводит на экран значение v, разделённое на 100 с разделителем точка
void PrintDec2(int v)
{
    if(v < 0)
    {
        v=-v;
        putchar('-');
    }
    printf("%d.%02d",v/100,v%100);
}
//*********************************************************************
//Пауза на заданное число миллисекунд
void PauseMs(unsigned int ms)
{
    unsigned int t=GetTickCount();
    while((GetTickCount()-t)<ms)
    {
    }
}
```

Результат:



См. также

BrushColor *ClrScr* *FontColor* *BigFont* *GotoXY* *printf* *putchar* *puts* *ShowScr* *GetTickCount*
Константы цветов *Цвет*

3.1.2. ВЫХОДЫ САТ-500

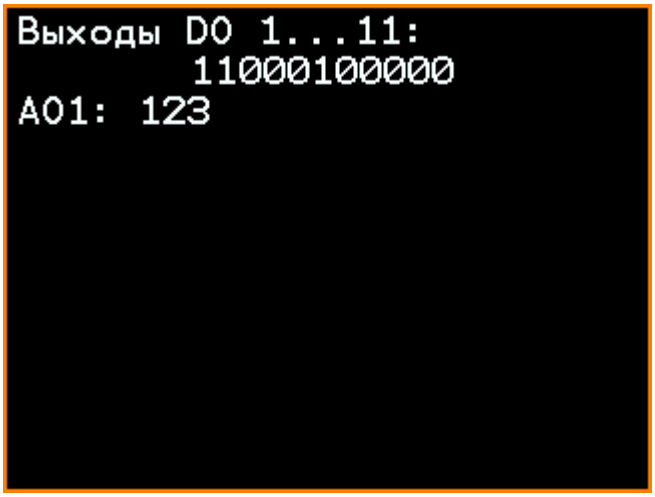
Для доступа к выходам контроллера САТ-500 в заголовочном файле *<sat500.h>* определены специальные переменные:

Переменная	*Описание*
char DO[11]	Выходы DO1...DO11 Значение 1 - выход замкнут Значение 0 - выход разомкнут
int AO1	Выход AO1 Значение 1000 соответствует напряжению на выходе 10 Вольт
int AO2	Выход AO2 Значение 1000 соответствует напряжению на выходе 10 Вольт

Пример:

```
#include <sat500.h>

//Создадим собственные имена для выходов
#define DO1    DO[0]
#define DO2    DO[1]

void main(void)
{
    BrushColor=COLOR_BLACK;          //Цвет кисти - чёрный
    ClrScr();                        //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;           //Цвет шрифта - белый
    BigFont=0;                       //Установить обычный шрифт (12x16, 12 пик-
селей  по горизонтали, 16 пикселей  по вертикали)
    AO1=123;                        //Управление аналоговым выходом AO1
    DO1=1;                           //Управление выходом DO1
    DO2=1;                           //Управление выходом DO2
    DO[5]=1;                         //Управление выходом DO6
    while(1)
    {
        GotoXY(4,4);                     //Курсор в позицию x=4,  y=4
        printf("Выходы DO 1...11: ");    //Вывод на дисплей  заголовка
        GotoXY(90,24);                   //Курсор в позицию x=4,  y=24
        for(char i=0;i<11;i++)
        {
            if(DO[i])putchar('1');       //Вывод на дисплей состояния выхода
DO1...DO11
            else     putchar('0');
        }
        GotoXY(4,44);                //Курсор в позицию x=4,  y=44
        printf("AO1: %d",AO1);       //Вывод на дисплей значения выхода AO1
        ShowScr();                   //Вывести на дисплей
    }
}
```

Результат:



См. также
BrushColor *ClrScr* *FontColor* *BigFont* *GotoXY* *printf* *putchar* *ShowScr*
Константы цветов *Цвет*

3.2. СИСТЕМНОЕ МЕНЮ

При создании собственной программы для контроллера иногда требуется задавать какие-нибудь настроечные параметры.
Контроллер "CAT-500" имеет специальное меню для ввода таких параметров.
Меню располагается следующим образом: "**Параметры программы/Настройки/***меню пользователя*".
Кроме этого можно вызвать системное меню контроллера CAT-500, используя функцию *System-Menu*.

3.2.1. ПУНКТЫ МЕНЮ

Меню "**Параметры программы/Настройки/***" контроллера CAT-500 состоит из пунктов, заданных в программе при помощи директивы *#pragma menuitem.*
Каждая такая директива создаёт переменную в программе и связывает её с новым подпунктом меню "**Параметры программы/Настройки/***".
Общий вид директивы добавления пункта меню:

```
#pragma menuitem тип имя: "Текст",min,max,value
```

, где :
тип имя - тип и имя создаваемой переменной
"Текст" - текст, отображаемый в меню
min - минимально допустимое значение переменной
max - максимально допустимое значение переменной
value - значение переменной по умолчанию (при запуске программы)

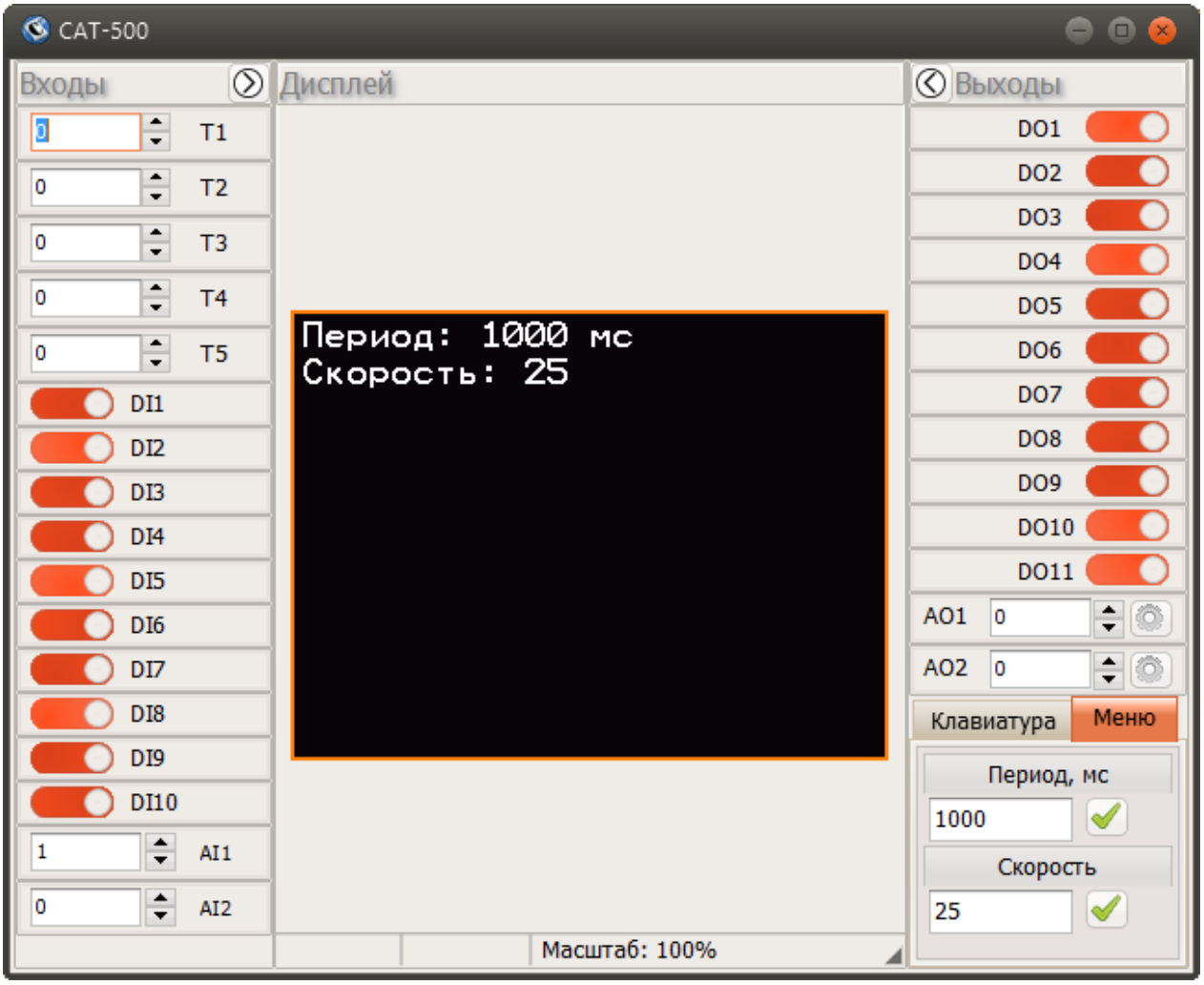
Порядок переменных в меню определяется порядком объявления переменных.
Иерархические пункты меню не поддерживаются.

В примере 1, приведённом далее в системном меню создаются две переменные типа int.

Пример 1:

```
//В этом примере создаются две переменные меню
#include <sat500.h>

#pragma menuitem int Period,"Период, мс",0,5000,1000
#pragma menuitem int Vel,"Скорость",10,50,25

void main(void)
{
    BrushColor=COLOR_BLACK;                 //Цвет кисти - чёрный
    ClrScr();                               //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;                  //Цвет шрифта - белый
    BigFont=0;                              //Установить обычный шрифт (12x16,
12 пикселей  по горизонтали, 16 пикселей  по вертикали)
    while(1)
    {
        GotoXY(4,4);                        //Курсор в позицию x=4,  y=4
        printf("Период: %d мс",Period);     //Вывод на дисплей переменной Pe-
riod
        GotoXY(4,24);                       //Курсор в позицию x=4,  y=24
        printf("Скорость: %d",Vel);         //Вывод на дисплей переменной Vel
        ShowScr();                          //Вывести на дисплей
        //Пауза 200 мс
        unsigned int t=GetTickCount();      //Текущее время от старта в мс
        while((GetTickCount()-t)<200);      //Ожидать 200 мс
    }
}
```

Результат:



В правой части окна отладчика отображаются переменные меню, где их можно изменить.
При перезапуске контроллера CAT-0500 значения переменных меню вновь будет установлено по умолчанию.
Для того чтобы переменные сохранялись их следует сохранять в энергонезависимой памяти BACK-UPRAM или EPROM.
В следующем примере две переменные меню сохраняются в энергонезависимой памяти BACK-UPRAM.

Пример 2:

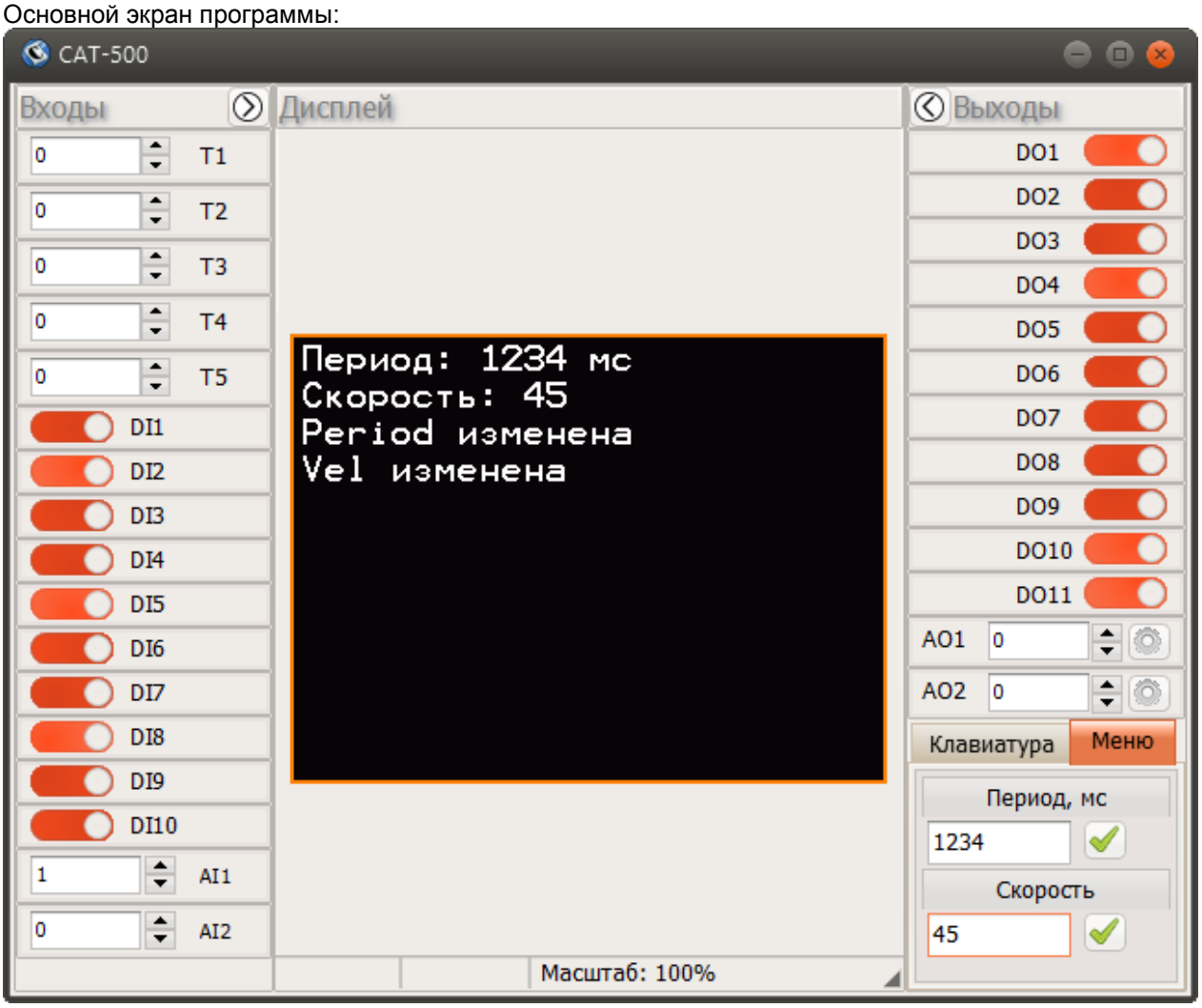
```
//-----------------------------------------------------------------------
//В этом примере создаются две переменные меню
//Эти переменные сохраняются в энергонезависимой памяти BACKUPRAM
//-----------------------------------------------------------------------
#include <sat500.h>

#pragma menuitem int Period,"Период, мс",0,5000,1000
#pragma menuitem int Vel,"Скорость",10,50,25
//-----------------------------------------------------------------------
//Чтение переменных из BACKUPRAM
void ReadFromBackupRam(void);
//Запись переменных в BACKUPRAM
void WriteToBackupRam(void);
```

```
//---------------------------------------------------------------------------
void main(void)
{
    BrushColor=COLOR_BLACK;                 //Цвет кисти - чёрный
    ClrScr();                               //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;                  //Цвет шрифта - белый
    BigFont=0;                              //Установить обычный шрифт (12x16,
12 пикселей  по горизонтали,16 пикселей  по вертикали)
    //"Старые" значения переменных
    int oldperiod,oldvel;
    oldperiod=Period;
    oldvel=Vel;
    while(1)
    {
        GotoXY(4,4);                        //Курсор в позицию x=4,  y=4
        printf("Период: %d мс",Period);     //Вывод на дисплей переменной Pe-
riod
        GotoXY(4,24);                       //Курсор в позицию x=4,  y=24
        printf("Скорость: %d",Vel);         //Вывод на дисплей переменной Vel
        //Если переменные меню изменились, то записываем их в BACKUPRAM
        if(oldperiod!=Period)
        {
            oldperiod=Period;
            //Запись переменных в BACKUPRAM
            WriteToBackupRam();
            GotoXY(4,44);                   //Курсор в позицию x=4,  y=44
            puts("Period изменена");        //Вывод сообщения на дисплей
        }
        if(oldvel!=Vel)
        {
            oldvel=Vel;
            //Запись переменных в BACKUPRAM
            WriteToBackupRam();
            GotoXY(4,64);                   //Курсор в позицию x=4,  y=64
            puts("Vel изменена");           //Вывод сообщения на дисплей
        }
        ShowScr();                          //Вывести на дисплей
        //Пауза 200 мс
        unsigned int t=GetTickCount();      //Текущее время от старта в мс
        while((GetTickCount()-t)<200);      //Ожидать 200 мс
    }
}
//---------------------------------------------------------------------------
#define VALID_ADR    0
#define PERIOD_ADR   4
#define VEL_ADR      8
#define VALID_VALUE  0xAA55FF00
//---------------------------------------------------------------------------
//Чтение переменных из BACKUPRAM
void ReadFromBackupRam(void)
{
    //Проверка, что переменные инициализированы
    unsigned int valid=BackupRamRead32(VALID_ADR);
    if(valid==VALID_VALUE)
    {
        //Переменные инициализированы
        //Читаем их из BACKUPRAM
        Period=BackupRamRead32(PERIOD_ADR);
```

```
        Vel=BackupRamRead32(VEL_ADR);
    }
    else
    {
        //Переменные не инициализированы
        //Сохраняем их в BACKUPRAM
        WriteToBackupRam();
        //И устанавливаем признак инициализации
        BackupRamWrite32(VALID_ADR,VALID_VALUE);
    }
}
//---------------------------------------------------------------------------
//Запись переменных в BACKUPRAM
void WriteToBackupRam(void)
{
    //Сохраняем в BACKUPRAM
    BackupRamWrite32(PERIOD_ADR,Period);
    BackupRamWrite32(VEL_ADR,Vel);
}
//---------------------------------------------------------------------------
```

Результат:

Основной экран программы:

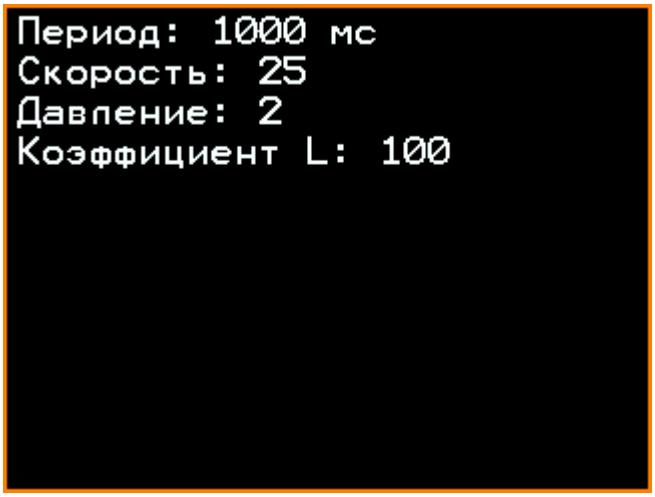


В следующем примере создаются 14 переменных меню

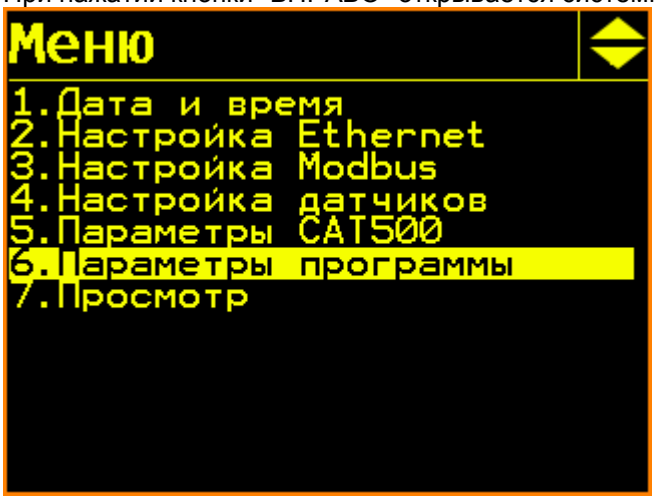
Пример 3:

```
//В этом примере создаются переменные меню
#include <sat500.h>
#pragma menuitem int Period,"1.Период, мс",0,5000,1000
#pragma menuitem char Vel,"2.Скорость",10,50,25
#pragma menuitem short int Press,"3.Давление",1,10,2
#pragma menuitem unsigned int KL,"4.Коэффициент L",1,500,100
#pragma menuitem int Var5,"5.Переменная 5",10,50,25
#pragma menuitem int Var6,"6.Переменная 6",10,50,25
#pragma menuitem int Var7,"7.Переменная 7",10,50,25
#pragma menuitem int Var8,"8.Переменная 8",10,50,25
#pragma menuitem int Var9,"9.Переменная 9",10,50,25
#pragma menuitem int Var10,"10.Переменная 10",10,50,25
#pragma menuitem int Var11,"11.Переменная 11",10,50,25
#pragma menuitem int Var12,"12.Переменная 12",10,50,25
#pragma menuitem int Var13,"13.Переменная 13",10,50,25
#pragma menuitem int Var14,"14.Переменная 14",10,50,25

void main(void)
{
    BrushColor=COLOR_BLACK;                 //Цвет кисти - чёрный
    ClrScr();                               //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;                  //Цвет шрифта - белый
    BigFont=0;                              //Установить обычный шрифт (12x16,
12 пикселей  по горизонтали, 16 пикселей  по вертикали)
    while(1)
    {
        GotoXY(4,4);                        //Курсор в позицию x=4,  y=4
        printf("Период: %d мс",Period);     //Вывод на дисплей переменной Pe-
riod
        GotoXY(4,24);                       //Курсор в позицию x=4,  y=24
        printf("Скорость: %d",Vel);         //Вывод на дисплей переменной Vel
        GotoXY(4,44);                       //Курсор в позицию x=4,  y=44
        printf("Давление: %d",Press);       //Вывод на дисплей переменной Press
        GotoXY(4,64);                       //Курсор в позицию x=4,  y=64
        printf("Коэффициент L: %d",KL);     //Вывод на дисплей переменной KL
        ShowScr();                          //Вывести на дисплей
        //Пауза 200 мс
        unsigned int t=GetTickCount();      //Текущее время от старта в мс
        while((GetTickCount()-t)<200);      //Ожидать 200 мс
    }
}
```

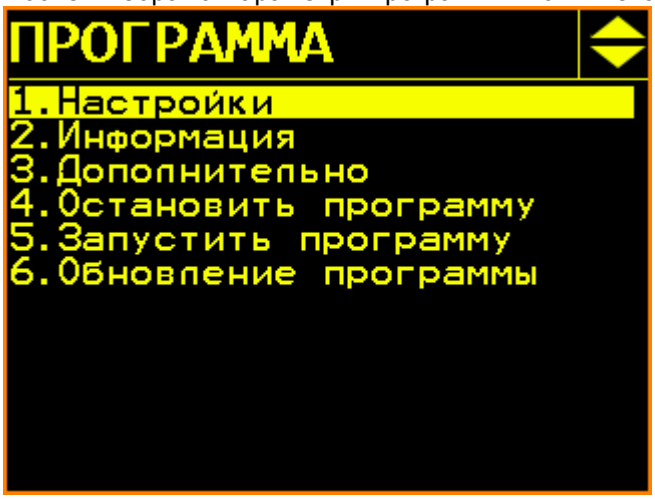
Результат:
Основной экран программы



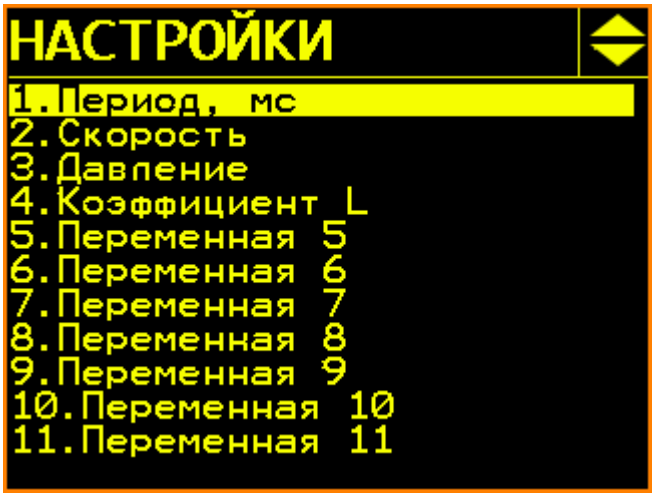
При нажатии кнопки "ВПРАВО" открывается системное меню:



После выбора "6.Параметры программы" появляется следующее меню:

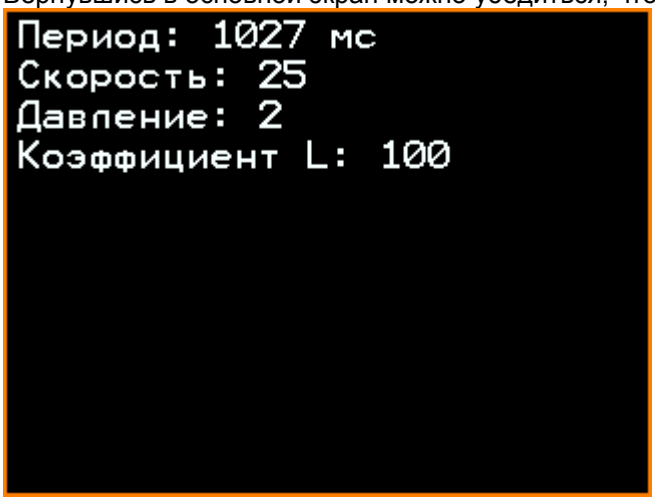


Пункт "1.Настройки" вызывает пользовательское меню программы:



Выбрав пункт меню "1.Период, мс" можно изменить значение переменной, связанной с данным пунктом меню:

Вернувшись в основной экран можно убедиться, что переменная изменилась:



См. также

BrushColor *ClrScr* *FontColor* *BigFont* *GotoXY* *printf* *ShowScr*
Константы цветов *Цвет*

3.2.2. ВЫЗОВ СИСТЕМНОГО МЕНЮ

```
#include <sat500.h>

void SystemMenu(void);
char SystemMenuVisible;
```

Функция вызывает системное меню контроллера CAT-500.
Переменная char SystemMenuVisible позволяет определить показывается ли в момент вызова системное меню контроллера CAT-500.

SystemMenuVisible	Описание
0	Системное меню CAT-500 не отображается
1	Системное меню CAT-500 отображается

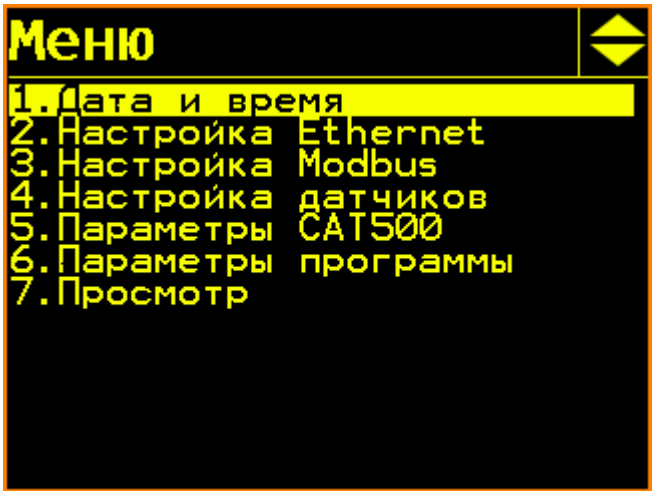
ВНИМАНИЕ! В имитаторе системное меню не отображается.

В приведённом ниже примере по нажатию любой кнопки вызывается системное меню контроллера CAT-500.
Во время когда показывается системное меню раздаётся звуковой сигнал.

Пример:

```
#include <sat500.h>

void Pause(unsigned int to)
{
    unsigned int t=GetTickCount();
    while((GetTickCount()-t) < to);
}

void main(void)
{
    BrushColor=COLOR_BLUE;      //Цвет кисти - синий
    ClrScr();                   //Очистить экран кистью - синим цветом
    ShowScr();                  //Вывести нарисованное не дисплей
    EnableKeyboard=1;           //Разрешить программе работу с клавиатурой
    while(1)                    //Бесконечный цикл
    {
        GotoXY(4,4);            //Установка позиции вывода на экран
        char key=GetKey();      //Получение нажатой кнопки
        if(key)
        {
            printf("Кнопка: %d",key); //Вывод кода нажатой кнопки
            ShowScr();                //Вывести на дисплей
            SystemMenu();             //Вызвать системное меню
        }
        if(SystemMenuVisible)
        {
            //Системное меню показывается
            Beep(100);              //Выдать звуковой сигнал 100 мс
            Pause(300);             //Пауза 300 мс
        }
    }
}
```

Результат:



3.2.3. НАЗВАНИЯ СИГНАЛОВ

```
#include <sat500.h>

void SetSigName(unsigned char index,char * name)
```

Функция позволяет задать имена сигналов, отображаемые в системных меню.
index - номер сигнала
name - строка названия сигнала

Возможные значения индекса:

Индекс	***Название сигнала***
0	Цифровой выход DO1
1	Цифровой выход DO2
2	Цифровой выход DO3
3	Цифровой выход DO4
4	Цифровой выход DO5
5	Цифровой выход DO6
6	Цифровой выход DO7
7	Цифровой выход DO8
8	Цифровой выход DO9
9	Цифровой выход DO10
10	Цифровой выход DO11
11	Цифровой вход DI1
12	Цифровой вход DI2
13	Цифровой вход DI3
14	Цифровой вход DI4
15	Цифровой вход DI5
16	Цифровой вход DI6
17	Цифровой вход DI7
18	Цифровой вход DI8
19	Цифровой вход DI9
20	Цифровой вход DI10
21	Аналоговый выход AO1
22	Аналоговый выход AO2
23	Аналоговый вход AI1
24	Аналоговый вход AI2
25	Температура T1

26	Температура T2
27	Температура T3
28	Температура T4
29	Температура T5

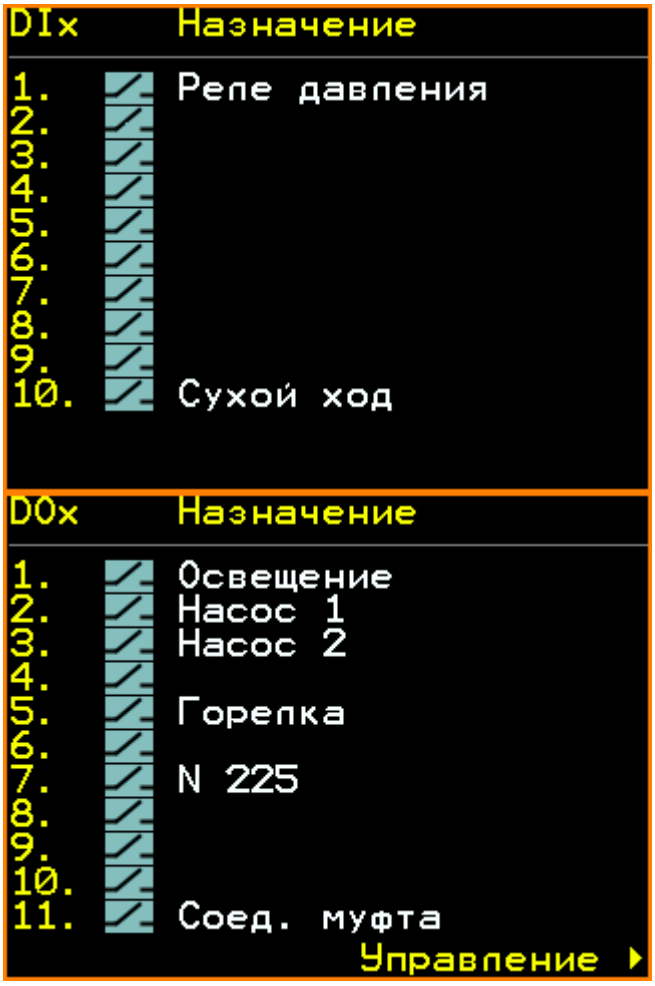
Пример

```
#include <sat500.h>
//-----------------------------------------------------------------------------
//Названия сигналов - индексы
enum
{
    //Сигналы DO1...DO11
    SIG_DO1=0,SIG_DO2=1,SIG_DO3=2,SIG_DO4=3,SIG_DO5=4,SIG_DO6=5,
    SIG_DO7=6,SIG_DO8=7,SIG_DO9=8,SIG_DO10=9,SIG_DO11=10,
    //Сигналы DI1...DI10
    SIG_DI1=11,SIG_DI2=12,SIG_DI3=13,SIG_DI4=14,SIG_DI5=15,
    SIG_DI6=16,SIG_DI7=17,SIG_DI8=18,SIG_DI9=19,SIG_DI10=20,
    //Сигналы AO1 и AO2
    SIG_AO1=21,SIG_AO2=22,
    //Сигналы AI1 и AI2
    SIG_AI1=23,SIG_AI2=24,
    //Сигналы T1...T5
    SIG_T1=25,SIG_T2=26,SIG_T3=27,SIG_T4=28,SIG_T5=29
};
//-----------------------------------------------------------------------------
int index;
char str[20];

void main(void)
{
    //Установка названий сигналов DO - строки располагаются во FLASH памяти
(const)
    SetSigName(SIG_DO1,"Освещение");
    SetSigName(SIG_DO2,"Насос 1");
    SetSigName(SIG_DO3,"Насос 2");
    SetSigName(SIG_DO5,"Горелка");
    SetSigName(SIG_DO11,"Соед. муфта");
    //Установка названий сигналов DI
    SetSigName(SIG_DI1,"Реле давления");
    SetSigName(SIG_DI10,"Сухой ход");
    //Установка названий сигналов T
    SetSigName(SIG_T1,"Темп. статора 1");
    SetSigName(SIG_T2,"Темп. статора 2");
    SetSigName(SIG_T5,"Температура НВ");
    //Установка названий сигналов AO1
    SetSigName(SIG_AO1,"Управление ПЧ1");
    SetSigName(SIG_AO2,"Управление ПЧ2");
    //Установка названий сигналов AI1
    SetSigName(SIG_AI1,"Давление 1");
    SetSigName(SIG_AI2,"Давление 2");
    ClrScr();
    BigFont=true;
    ShowScr();
    while(1)
    {
        //Динамическое имя сигнала DO7 - располагается в RAM памяти
        sprintf(str,"N %d",index++);
```

```
        //Имя динамического сигнала должно быть глобальной переменной
        SetSigName(SIG_DO7,str);
        GotoXY(4,4);
        puts(str);
        ShowScr();
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<500);
    }
}
```

Результат:
Основной экран:

Нажимая на контроллере кнопку "ВВЕРХ" можно просмотреть экраны состояния и убедиться в том, что сигналы получили новые названия.


DIx Назначение
1. Реле давления
2.
3.
4.
5.
6.
7.
8.
9.
10. Сухой ход
DOx Назначение
1. Освещение
2. Насос 1
3. Насос 2
4.
5. Горелка
6.
7. N 225
8.
9.
10.
11. Соед. муфта
Управление

3.3. ДИСПЛЕЙ

В контроллере CAT-500 используется внутренний буфер для предотвращения мерцания дисплея при перерисовке.
Это означает, что все С-функции выполняют рисование и вывод текста в копии экрана, располагающейся в оперативной памяти контроллера.
Как только экран сформирован, следует вызвать специальную функцию *ShowScr()*, которая перепишет экранный буфер из оперативной памяти в контроллер дисплея.
Такой способ вывода позволяет обеспечить высокое качество изображения - отсутствие мерцания дисплея при перерисовке.
Программист может вызывать функцию вывода *ShowScr()* в любой момент времени.
Все функции работы с дисплеем доступны после включения заголовочного файла:

```
#include <sat500.h>
```

При рисовании на дисплее используются виртуальное перо и виртуальная кисть.
Перо используется для прорисовки контура фигуры, а кисть - для заполнения нарисованного контура внутри.
Вывод информации на дисплей контроллера CAT-500 может быть организован с использованием следующих переменных и функций:

Переменная	***Описание***
short int BrushColor	Цвет кисти рисования на дисплее
short int FontColor	Цвет шрифта (букв)
short int PenColor	Цвет пера (линий) рисования на дисплее
short int PenWidth	Толщина пера в пикселях
char BigFont	Крупный шрифт

Функция	***Описание***
void ClrScr (void)	Очистка экрана и заливка цветом кисти
void GotoXY (short int x,short int y)	Установка курсора (указателя рисования и вывода текста) в положение x, y на дисплее
void putchar (char c)	Вывод символа в положение курсора и установка курсора в следующую позицию вывода символа (включая переход на новую строку)
*int printf (char * format,...)*	Форматированный вывод текста на дисплей
*void puts (char * text)*	Вывод строки на дисплей
void PrintInt (int val)	Вывод на дисплей целого числа (только значащие цифры)
void PrintIntLen (int val,unsigned char len)	Вывод на дисплей целого числа заданным количеством цифр (заполнение нулями слева)
short int RGB (unsigned char r,unsigned char g,unsigned char b)	Получение цвета по заданным значениям составляющих R, G и B
void Pixel (short int x,short int y,short int color)	Установка цвета пикселя дисплея
void LineTo (short int x,short int y)	Рисование линии из позиции курсора в позицию x,y при помощи текущей настройки пера
void Rectangle (short int left,short int top,short int right,short int bottom)	Рисование прямоугольника. Контур обводится пером, а внутренняя часть заполняется цветом кисти
void RectangleBrush (short int left,short int top,short int right,short int bottom)	Рисование прямоугольника, заполненного цветом кисти. Перо не используется
void RectanglePen (short int left,short int top,short int right,short int bottom)	Рисование контура прямоугольника при помощи пера. Внутренняя часть не заполняется. Кисть не используется
void Ellipse (short int left,short int top,short int right,short int bottom)	Рисование эллипса или круга. Контур обводится пером, а внутренняя часть заполняется цветом кисти
void EllipsePen (short int left,short int top,short int right,short int bottom)	Рисование эллипса или круга пером, внутренняя часть не заполняется. Кисть не используется
void DrawPicture (short int x,short int y,short int id,char transparent)	Рисование картинки в заданном месте дисплея. Картинка может рисоваться с прозрачным цветом
void ShowScr (void)	Завершение рисования экрана (копирование видеообласти)

3.3.1. BIGFONT

```
#include <sat500.h>

char BigFont
```

Переменная выбора шрифта для вывода текста на дисплей.
Всего поддерживается два шрифта:

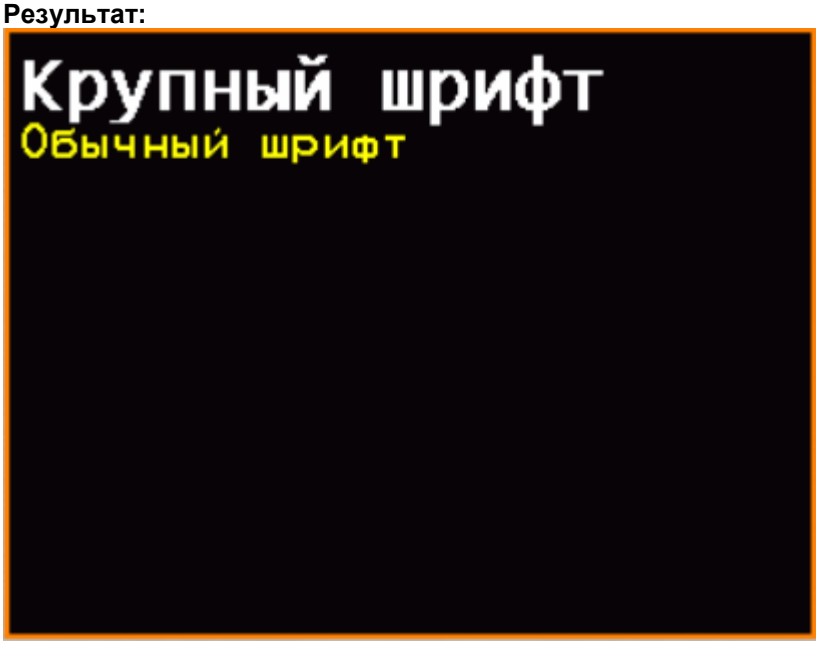
Значение	*Шрифт*
BigFont=0	Обычный шрифт 12x16 12 пикселей по горизонтали 16 пикселей по вертикали

BigFont=1	Крупный шрифт 18x32 18 пикселей по горизонтали 32 пикселя по вертикали

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=0;             //Цвет кисти - чёрный
    ClrScr();                 //Очистить экран чёрным цветом
    BigFont=1;                //Выбор шрифта - крупный шрифт
    FontColor=COLOR_WHITE;    //Цвет шрифта - белый
    GotoXY(4,4);              //Курсор в позицию x=4, y=4
    puts("Крупный шрифт");    //Вывод текста
    GotoXY(4,36);             //Курсор в позицию x=4, y=36
    FontColor=COLOR_YELLOW;   //Цвет шрифта - жёлтый
    BigFont=0;                //Выбор шрифта - обычный шрифт
    puts("Обычный шрифт");    //Вывод текста
    ShowScr();                //Вывести на дисплей
    while(1);                 //Бесконечная пауза
}
```

Результат:



См. также
BrushColor *ClrScr* *FontColor* *GotoXY* *puts* *ShowScr*

3.3.2. BRUSHCOLOR

```
#include <sat500.h>

short int BrushColor
```

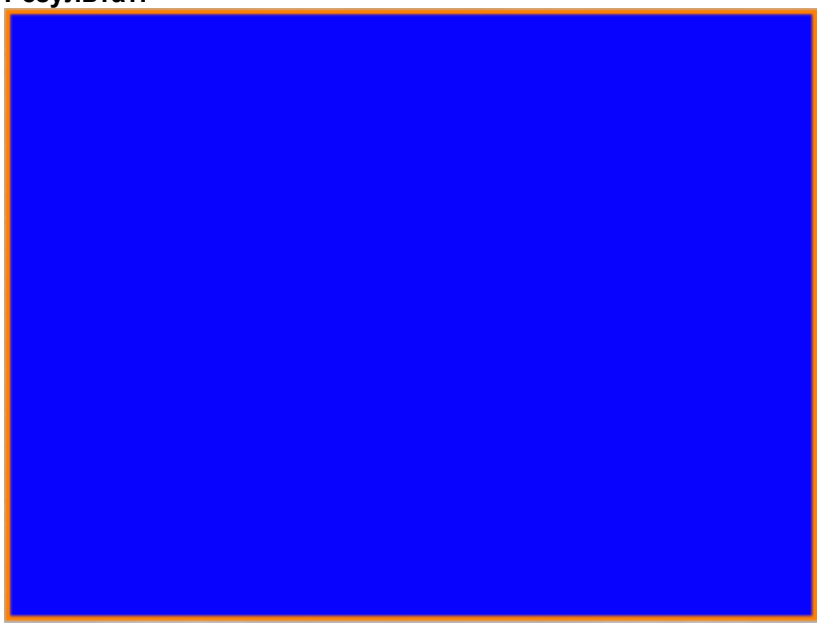
Переменная, значение которой задаёт цвет кисти рисования на дисплее.

Кисть используется для закрашивания дисплея при очистке и при заполнении фигур цветом.

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLUE;      //Цвет кисти - синий
    ClrScr();                   //Очистить экран кистью - синим цветом
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



См. также
Цвет *Константы цветов* *ClrScr* *ShowScr*

3.3.3. CLRSCR

```
#include <sat500.h>

void ClrScr (void);
```

Очистка экрана и заливка цветом кисти *BrushColor*.

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=RGB(64,128,64);     //Цвет кисти RED=64 GREEN=128 BLUE=64
```

```
    ClrScr();                    //Очистить экран цветом кисти
    ShowScr();                   //Вывести на дисплей
    while(1);                    //Бесконечная пауза
}
```

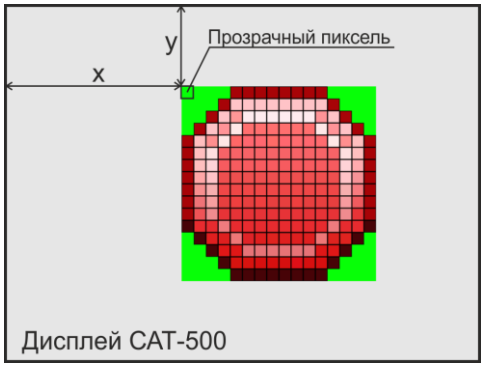
Результат:

См. также
BrushColor ShowScr RGB

3.3.4. DRAWPICTURE

```
#include <sat500.h>

void DrawPicture (short int x,short int y,short int id,char transparent);
```

Рисование картинки в заданном месте дисплея. Картинка может рисоваться с прозрачным цветом.
x - положение начала картинки по горизонтали
y - положение начала картинки по вертикали
id - уникальный идентификатор картинки
transparent - рисовать картинку с прозрачным цветом. Прозрачным считается цвет левого верхнего пикселя:



Если *transparent=1*, то все пиксели цвет которых совпадает с "прозрачным пикселем" не рисуются на дисплее.
Если *transparent=0*, то рисуются все пиксели картинки.

Перед выводом картинки на дисплей необходимо в программе указать путь на картинку, используя директиву #pragma следующего вида:

```
#pragma picture(ID) "имя файла"
```

ID – уникальный идентификатор картинки (не может быть нулевым)
Необходимо в поле *"имя файла"* задать путь к файлу картинки, а в качестве ID указать любой уникальный числовой идентификатор.
Рекомендуется задавать идентификаторы начиная с единицы.
В дальнейшем к файлу картинки можно будет обращаться по этому идентификатору.
Рекомендуется располагать файлы картинок в том же каталоге, что и исходные файлы проекта и при переносе на другое рабочее место файлы картинок следует копировать вместе с исходными файлами программы.
Любая добавленная картинка преобразуется в специальный формат для записи в контроллер САТ-500 (SATPICT).
В процессе преобразования картинка отображается на чёрном фоне и прозрачность (для PNG и ICO удаляется).
Если требуется использовать картинку с фоном другого цвета, то необходимо преобразовать картинку в файл формата SATPICT. Для этого используется специальное окно, доступное при выборе пункта меню программы: **"Файл\Подготовка картинок..."**
При выводе картинки на экран "прозрачным" цветом считается цвет верхнего левого пикселя. В случае задания режима *transparent=1* этот цвет считается прозрачным.
Перечень поддерживаемых форматов: PNG, BMP, GIF, ICO, JPG, SATPICT.

Пример:

```
#include <sat500.h>

//Определяем файл картинки и задаём ему идентификатор 1
#pragma picture(1) "Bmp\Attention 16.bmp"
//Будем использовать вместо идентификатора 1 текстовую константу PICT_ATTEN-
TION
#define PICT_ATTENTION  1

void main(void)
{
    BrushColor=RGB(0,0,0);                  //Цвет кисти - чёрный
    ClrScr();                               //Очистить экран чёрным цветом
    DrawPicture(10,10,PICT_ATTENTION,1);    //Нарисовать картинку PICT_ATTEN-
TION с прозрачным фоном
    ShowScr();                              //Вывести нарисованное на дисплей
    while(1);                               //Бесконечная пауза
}
```

Результат:

См. также
BrushColor *ClrScr* *puts* *ShowScr* *RGB*
Константы цветов *Цвет*
Директива #define

3.3.5. ELLIPSE

```
#include <sat500.h>

void Ellipse (short int left,short int top,short int right,short int bottom);
```

Рисование *эллипса* или *круга*. Контур обводится пером, а внутренняя часть заполняется цветом кисти *BrushColor*.
Цвет пера задаётся значением переменной *PenColor*.
Толщина пера в пикселях задаётся значением переменной *PenWidth.*
Если размер эллипса по вертикали равен размеру по горизонтали, то рисуется круг.

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    //Эллипс
    BrushColor=COLOR_WHITE;      //Цвет кисти - белый
    PenColor=COLOR_RED;          //Цвет пера - красный
    PenWidth=5;                  //Толщина пера - 5 пикселей
    Ellipse(24,24,204,104);      //Нарисовать эллипс круг кистью и пером
    //Круг
    BrushColor=COLOR_YELLOW;     //Цвет кисти - жёлтый
    PenColor=COLOR_BLUE;         //Цвет пера - синий
    PenWidth=3;                  //Толщина пера - 3 пикселей
    Ellipse(60,60,220,220);      //Нарисовать круг кистью и пером
    ShowScr();                   //Вывести нарисованное на дисплей
    while(1);                    //Бесконечная пауза
}
```

Результат:

См. также
BrushColor *ClrScr* *PenColor* *PenWidth* *ShowScr*
Константы цветов *Цвет* *RGB*

3.3.6. ELLIPSEPEN

```
#include <sat500.h>

void EllipsePen (short int left,short int top,short int right,short int bot-
tom);
```

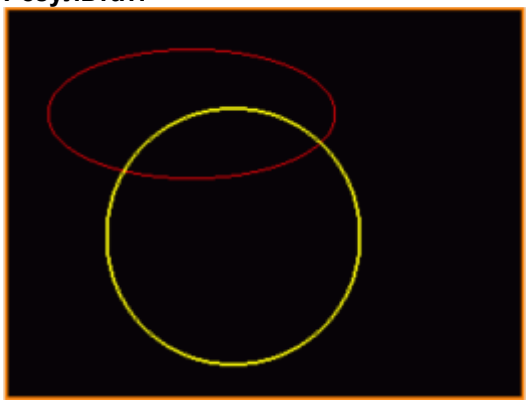
Рисование *эллипса* или *круга* текущим пером, внутренняя часть цветом не заполняется.
Цвет пера задаётся значением переменной *PenColor*.
Толщина пера в пикселях задаётся значением переменной *PenWidth.*
Если размер эллипса по вертикали равен размеру по горизонтали, то рисуется круг.

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    //Эллипс
    PenColor=COLOR_RED;          //Цвет пера - красный
```

```
    PenWidth=1;                 //Толщина пера - 1 пиксель
    EllipsePen(24,24,204,104);  //Нарисовать эллипс пером
    //Круг
    PenColor=COLOR_YELLOW;      //Цвет пера - жёлтый
    PenWidth=2;                 //Толщина пера - 2 пикселя
    EllipsePen(60,60,220,220);  //Нарисовать круг пером
    ShowScr();                  //Вывести нарисованное на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



См. также
BrushColor *ClrScr* *PenColor* *PenWidth* *ShowScr*
Константы цветов *Цвет* *RGB*

3.3.7. FONTCOLOR

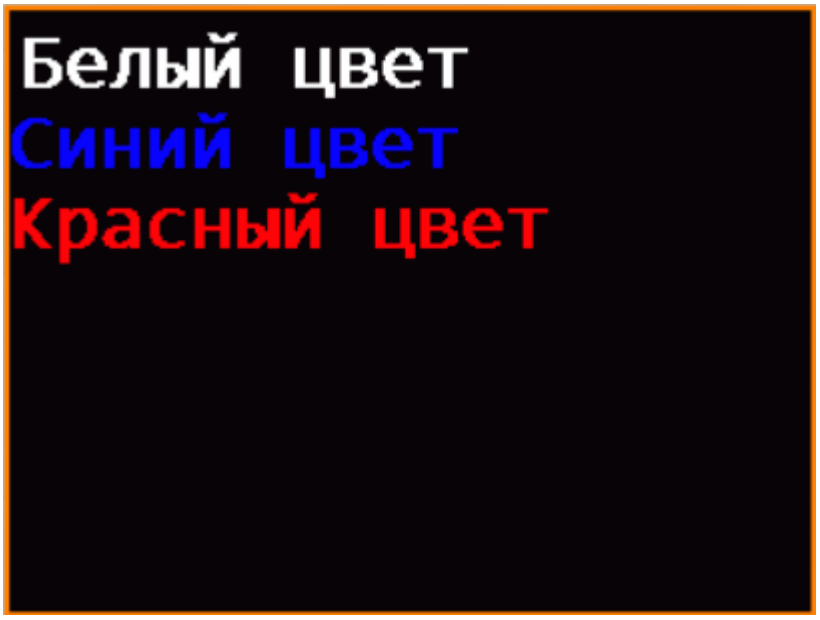
```
#include <sat500.h>

short int FontColor
```

Переменная, значение которой задаёт цвет шрифта (букв).

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;     //Цвет кисти - чёрный
    ClrScr();                   //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;      //Цвет шрифта - белый
    GotoXY(4,4);                //Курсор в позицию 4,4
    puts("Белый цвет\n");       //Вывод строки
    FontColor=COLOR_BLUE;       //Цвет шрифта - синий
    puts("Синий цвет\n");       //Вывод строки
    FontColor=COLOR_RED;        //Цвет шрифта - красный
    puts("Красный цвет\n");     //Вывод строки
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



См. также
Цвет *Константы цветов* *BrushColor* *ClrScr* *puts* *ShowScr*

3.3.8. GOTOXY

```
#include <sat500.h>

void GotoXY (short int x,short int y);
```

Установка курсора (указателя рисования и вывода текста) в положение x, y на дисплее
x - пиксельная координата по горизонтали
y - пиксельная координата по вертикали

Доступны следующие константы размера дисплея:

```
#define SCREEN_WIDTH   320u
#define SCREEN_HEIGHT  240u
```

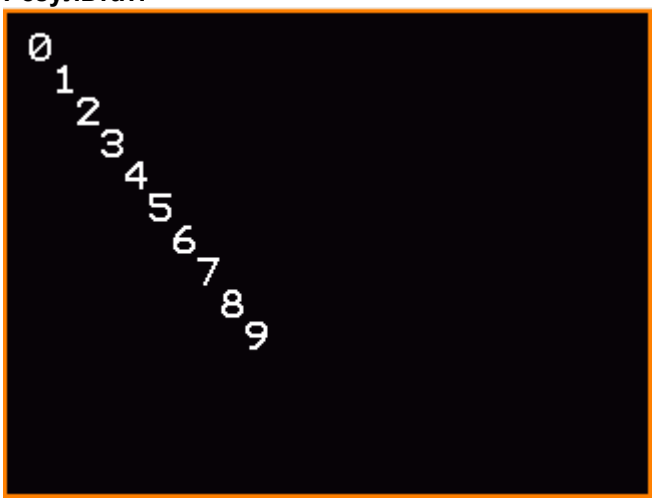
```
// CAT500 - координаты
// (0,0)                    (0,319)
// ---------------------------------- x
// |
// |
// |
// |
// | y (239,0)
```

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;         //Цвет кисти - чёрный
    ClrScr();                       //Очистить экран чёрным цветом
```

```
    FontColor=COLOR_WHITE;          //Цвет шрифта - белый
    BigFont=0;                      //Обычный шрифт 12x16 (не крупный)
    for(int i=0;i<10;i++)
    {
        GotoXY(10+i*12,10+i*16);    //Курсор в позицию x=10+i*12,  y=10+i*16
        putchar('0'+i);             //Вывести символ
    }
    ShowScr();                      //Вывести на дисплей
    while(1);                       //Бесконечная пауза
}
```

Результат:



См. также
BrushColor *ClrScr* *FontColor* *BigFont* *putchar* *ShowScr*
Константы цветов *Цвет*

3.3.9. LINETO

```
#include <sat500.h>

void LineTo (short int x,short int y);
```

Рисование *линии* из позиции курсора в позицию x,y при помощи текущей настройки пера.
Цвет пера задаётся значением переменной *PenColor*.
Толщина пера в пикселях задаётся значением переменной *PenWidth.*
Текущее положение курсора можно изменить функцией *GotoXY*.
После прорисовки линии текущему положению пера назначается координаты конца линии.

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;                 //Цвет кисти - чёрный
    ClrScr();                               //Очистить экран чёрным цветом
    PenColor=COLOR_GREEN;                   //Цвет пера - зелёный
    PenWidth=3;                             //Толщина пера - 3 пикселя
    GotoXY(4,4);                            //Курсор в позицию 4,4
    LineTo(SCREEN_WIDTH-4,SCREEN_HEIGHT-4); //Линия до позиции 316,236
```

```
    ShowScr();                              //Вывести на дисплей
    while(1);                               //Бесконечная пауза
}
```

Результат:

См. также
GotoXY BrushColor ClrScr PenColor PenWidth ShowScr
Константы цветов Цвет RGB Размеры дисплея

3.3.10. PENCOLOR

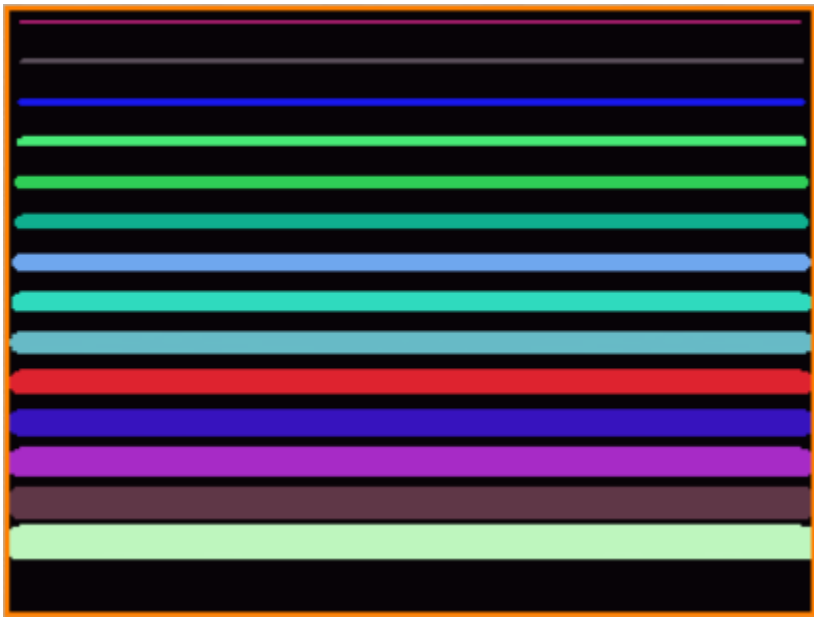
```
#include <sat500.h>

short int PenColor
```

Переменная, значение которой задаёт цвет пера.
Перо используется для рисования на дисплее линий и контура обводки фигур.

Пример:

```
#include <sat500.h>

void main(void)
{
    short int y=4;
    BrushColor=COLOR_BLACK;          //Цвет кисти - чёрный
    ClrScr();                        //Очистить экран чёрным цветом
    for(int i=1;i<15;i++)
    {
        PenColor=random(65536);      //Цвет пера - случайное число
        PenWidth=i;                  //Толщина пера - i пикселей
        GotoXY(4,y);                 //Курсор в позицию 4,y
        LineTo(SCREEN_WIDTH-4,y);    //Линия до позиции 316,236
        y+=16;                       //Увеличить координату y
    }
    ShowScr();                       //Вывести на дисплей
    while(1);                        //Бесконечная пауза
}
```

Результат:



См. также
Цвет *Константы цветов* *BrushColor* *ClrScr* *PenWidth* *GotoXY* *ShowScr*

3.3.11. PENWIDTH

```
#include <sat500.h>
```

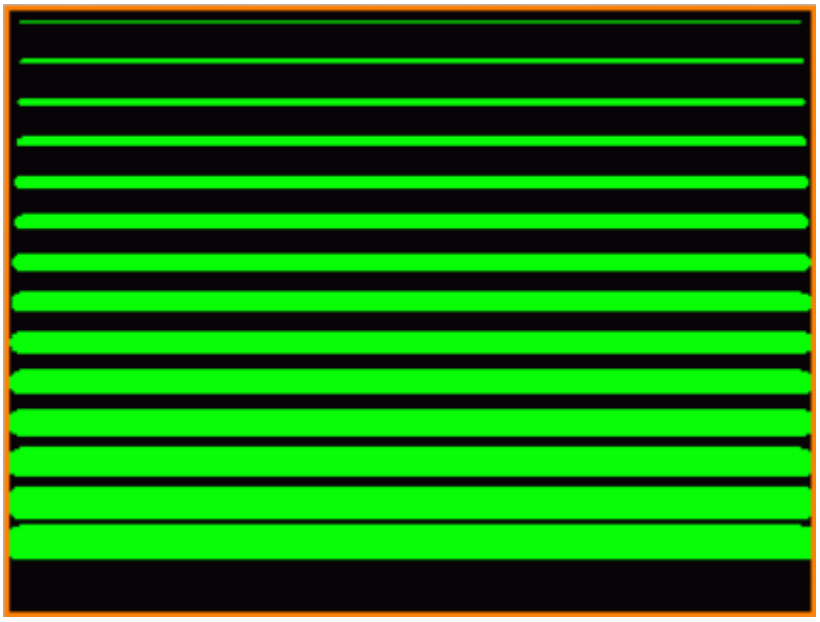
short int PenWidth

Переменная, значение которой задаёт толщину пера в пикселях.
Перо используется для рисования линий и контура обводки фигур.

Пример:

```
#include <sat500.h>

void main(void)
{
    short int y=4;
    BrushColor=COLOR_BLACK;          //Цвет кисти - чёрный
    ClrScr();                        //Очистить экран чёрным цветом
    PenColor=COLOR_GREEN;            //Цвет пера - зелёный
    for(int i=1;i<15;i++)
    {
        PenWidth=i;                  //Толщина пера - i пикселей
        GotoXY(4,y);                 //Курсор в позицию 4,y
        LineTo(SCREEN_WIDTH-4,y);    //Линия до позиции 316,y
        y+=16;                       //Увеличить координату y
    }
    ShowScr();                       //Вывести на дисплей
    while(1);                        //Бесконечная пауза
}
```

Результат:



См. также
BrushColor *ClrScr* *PenColor* *GotoXY* *LineTo* *ShowScr*
Константы цветов *Цвет* *Размеры дисплея*

3.3.12. PRINTF

```
#include <sat500.h>

int printf (char * format,...);
```

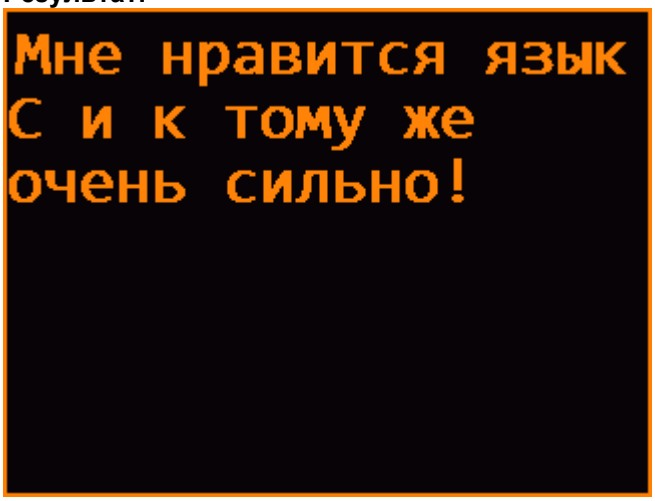
Форматированный вывод текста на дисплей. Текст выводится с использованием цвета текущей кисти *BrushColor* (задаёт цвет фона) и цвета шрифта *FontColor*.
Функция *printf()* возвращает число выведенных символов или отрицательное значение в случае ошибки.
Управляющая строка состоит из элементов двух видов. Первый из них - это символы, которые предстоит вывести на экран, второй - это спецификаторы преобразования, которые определяют способ вывода стоящих за ними аргументов. Каждый такой спецификатор начинается со знака процента, за которым следует код формата. Аргументов должно быть ровно столько, сколько и спецификаторов, причём спецификаторы преобразования и аргументы должны попарно соответствовать друг другу в направлении слева направо.

Пример:

```
#include <sat500.h>

const char str[]="и к тому же\nочень сильно!";
void main(void)
{
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    FontColor=RGB(255,128,0);    //Цвет шрифта RED=255 GREEN=128 BLUE=0
    GotoXY(4,4);                 //Курсор в позицию x=4,  y=4
    printf("Мне нравится язык %c %s",'C',str);    //Вывод строки
    ShowScr();                   //Вывести на дисплей
    while(1);                    //Бесконечная пауза
```

```
}
```

Результат:



В этом примере первому спецификатору преобразования (%c), соответствует символ 'C', а второму (%s), - строка "*и к тому же очень сильно!*".
В функции printf(), как видно из таблицы, имеется основной набор спецификаторов преобразования.

Код	*Формат*
%c	Символ
%d	Десятичное целое со знаком
%s	Строка символов
%u	Десятичное целое без знака
%x	Шестнадцатеричное без знака (буквы на нижнем регистре)
%X	Шестнадцатеричное без знака (буквы на верхнем регистре)
%%	Выводит знак %

Вывод символов

Для вывода отдельного символа используйте %c. В результате соответствующий аргумент будет выведен на экран без изменения.
Для вывода строки используйте %s.

Вывод чисел

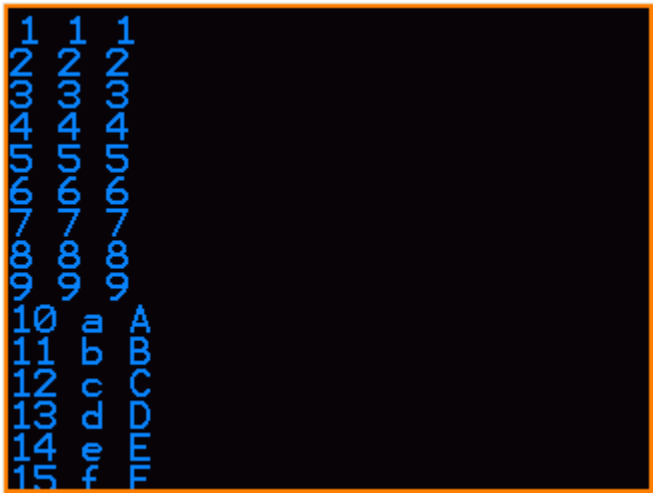
Числа в десятичном формате со знаком отображаются с помощью спецификатора преобразования %d.
Для вывода целого значения без знака используйте %u.
Целые числа без знака можно выводить в шестнадцатеричном формате, используя спецификатор преобразования %X или %x. Так как в шестнадцатеричной системе для представления чисел от 10 до 15 используются буквы от A до F, то эти буквы можно выводить в верхнем или в нижнем регистре. Как показано ниже, в первом случае используется спецификатор преобразования %X, а во втором — спецификатор преобразования %x:

Пример:

```
#include <sat500.h>

void main(void)
{
```

```
    unsigned num;
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    BigFont=0;                   //Обычный шрифт (не крупный)
    FontColor=RGB(0,128,255);    //Цвет шрифта RED=0 GREEN=128 BLUE=255
    for(num=1; num < 16; num++)
    {
        printf("%d ", num);      //Вывод в десятичном виде
        printf("%x ", num);      //Вывод шестнадцатеричном формате в нижнем ре-
гистре
        printf("%X\n", num);     //Вывод шестнадцатеричном формате в верхнем
регистре
    }
    ShowScr();                   //Вывести нарисованное не дисплей
    while(1);                    //Бесконечная пауза
}
```

Результат:



Отображение адреса

Для отображения адреса используйте спецификатор преобразования %X,%x или %d.

Модификаторы формата

Во многих спецификаторах преобразования можно указать модификаторы, которые слегка меняют их значение. Например, можно указывать минимальную ширину поля, количество десятичных разрядов и выравнивание по левому краю. Модификатор формата помещают между знаком процента и кодом формата. Об этих модификаторах сейчас и пойдет речь.

Модификаторы минимальной ширины поля

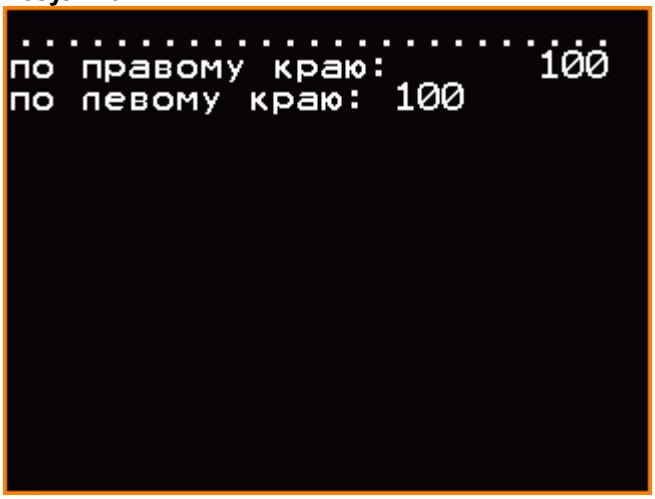
Целое число, расположенное между знаком % и кодом формата, играет роль модификатора минимальной ширины поля. Если указан модификатор минимальной ширины поля, то чтобы ширина поля вывода была не меньше указанной минимальной длины, при необходимости вывод будет дополнен пробелами. Если же выводятся строки или числа, которые длиннее указанного минимума, то они все равно будут отображаться полностью. По умолчанию для дополнения используются пробелы. А если для этого надо использовать нули, то перед модификатором ширины поля следует поместить 0. Например, *%05d* означает, что любое число, количество цифр которого меньше пяти, будет дополнено таким количеством нулей, чтобы число состояло из пяти цифр.
Модификатор минимальной ширины поля чаще всего используется при создании таблиц, в которых столбцы должны быть выровнены по вертикали.

Выравнивание вывода

По умолчанию весь вывод выравнивается по правому краю. То есть если ширина поля больше ширины выводимых данных, то эти данные располагаются по правому краю поля. Вывод по левому краю можно назначить принудительно, поместив знак минус прямо за %. Например, *%-8d* означает, что число с будет выровнено по левому краю 8-символьного поля.
В следующей программе показано, как применяется выравнивание по левому краю:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;     //Цвет кисти - чёрный
    ClrScr();                   //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;      //Цвет шрифта - белый
    BigFont=0;                  //Установить обычный шрифт (12x16, 12 пикселей
по горизонтали, 16 пикселей  по вертикали)
    printf(".......................\n");
    printf("по правому краю: %8d\n", 100);
    printf("по левому краю: %-8d\n", 100);
    ShowScr();                  //Вывести нарисованное не дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



Обработка данных других типов

Некоторые модификаторы в вызове функции *printf()* позволяют отображать целые числа типа ***short*** и ***char***. Такие модификаторы можно использовать для следующих спецификаторов типа: d, u и x.

Пример

Ниже приведён пример, который поможет разобраться с форматированным выводом *printf()*. Пример носит академический характер, т.к. такое большое число нельзя целиком вывести на дисплей CAT-500.

```
#include <sat500.h>

void main(void)
{
    ClrScr();
```

```
    char *ptr = "Hello world!";
    char *np = 0;
    int i = 5;
    unsigned int bs = sizeof(int)*8;
    int mi;
    mi = (1 << (bs-1)) + 1;
    printf("%s\n", ptr);                                    // Hello world!
    printf("printf test\n");                                // printf test
    printf("%s is null pointer\n", np);                     // (null) is null
pointer
    printf("%d = 5\n", i);                                  // 5 = 5
    printf("%d = - max int\n", mi);                         // -2147483647 = -
max int
    printf("char %c = 'a'\n", 'a');                         // char a = 'a'
    printf("hex %x = ff\n", 0xff);                          // hex ff = ff
    printf("hex %02x = 00\n", 0);                           // hex 00 = 00
    printf("signed %d = unsigned %u = hex %x\n",-3,-3,-3);  // signed -3 = un-
signed 4294967293 = hex fffffffd
    printf("%d %s(s)%", 0, "message");                      // 0 message(s)
    printf("\n");                                           //
    printf("%d %s(s) with %%\n", 0, "message");             // 0 message(s)
with %
    printf("justif: \"%-10s\"\n", "left");                  // justif: "left "
    printf("justif: \"%10s\"\n", "right");                  // justif: "
right"
    printf(" 3: %04d zero padded\n", 3);                    // 3: 0003 zero
padded
    printf(" 3: %-4d left justif.\n", 3);                   // 3: 3 left jus-
tif.
    printf(" 3: %4d right justif.\n", 3);                   // 3: 3 right jus-
tif.
    printf("-3: %04d zero padded\n", -3);                   // -3: -003 zero
padded
    printf("-3: %-4d left justif.\n", -3);                  // -3: -3 left
justif.
    printf("-3: %4d right justif.\n", -3);                  // -3: -3 right
justif.
    ShowScr();
    while(1);
}
```

См. также
BrushColor *ClrScr* *BigFont* *FontColor* *GotoXY* *ShowScr*
Константы цветов *Цвет*

3.3.13. PRINTINT

```
#include <sat500.h>

void PrintInt(int val);
```

Вывод на дисплей целого числа (только значащие цифры) в текущей позиции курсора.
Текст выводится с использованием цвета текущей кисти *BrushColor* (задаёт цвет фона) и цвета шрифта *FontColor*.

val - выводимое число

Пример:

```
#include <sat500.h>

int Year=2018;

void main(void)
{
    BrushColor=COLOR_BLUE;          //Цвет кисти - синий
    ClrScr();                       //Очистить экран синим цветом
    FontColor=COLOR_WHITE;          //Цвет шрифта - белый
    GotoXY(100,50);                 //Курсор в позицию x=100, y=50
    PrintInt(Year);                 //Вывести число - значение переменной Year
    ShowScr();                      //Вывести на дисплей
    while(1);                       //Бесконечная пауза
}
```

Результат:

См. также
BrushColor *ClrScr* *FontColor* *GotoXY* *ShowScr*
Константы цветов *Цвет* *RGB*

3.3.14. PRINTINTLEN

```
#include <sat500.h>

void PrintIntLen(int val,unsigned char len);
```

Вывод на дисплей целого числа заданным количеством цифр. Если количество выводимых цифр менее чем задано в параметре ***len***,
то выполняется заполнение нулями слева.
Текст выводится с использованием цвета текущей кисти *BrushColor* (задаёт цвет фона) и цвета шрифта *FontColor*.
val - выводимое число
len - количество значащих цифр

Пример:

```
#include <sat500.h>

int Year=2018;

void main(void)
{
    BrushColor=COLOR_BLUE;          //Цвет кисти - синий
    ClrScr();                       //Очистить экран синим цветом
    FontColor=COLOR_WHITE;          //Цвет шрифта - белый
    GotoXY(100,50);                 //Курсор в позицию x=100, y=50
    PrintIntLen(Year,8);            //Вывести число - значение переменной Year
в виде 8 цифр
    ShowScr();                      //Вывести на дисплей
    while(1);                       //Бесконечная пауза
}
```

Результат:

См. также
BrushColor *ClrScr* *FontColor* *GotoXY* *ShowScr*
Константы цветов *Цвет* *RGB*

3.3.15. PUTCHAR

```
#include <sat500.h>

void putchar (char c);
```

Вывод символа в текущее положение курсора, и установка курсора в следующую позицию вывода символа (включая переход на новую строку).
При выводе обрабатываются управляющие символы:
\n - Перевод строки и возврат каретки (увеличение координату y курсора на величину высоты текущего шрифта и установка координаты курсора x в нулевое значение)
\r - Возврат каретки (установка координаты курсора x в нулевое значение)
\t - Установка координаты курсора x в следующее значение кратное 4-м символам текущего размера

Пример:

```
#include <sat500.h>
```

```
void main(void)
{
    BrushColor=COLOR_BLACK;     //Цвет кисти - чёрный
    ClrScr();                   //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;      //Цвет шрифта - белый
    GotoXY(20,10);              //Курсор в позицию x=10,  y=10
    putchar('H');               //Вывести символ H
    putchar('i');               //Вывести символ i
    putchar('\r');              //Возврат каретки
    putchar('!');               //Вывести символ !
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:

См. также
BrushColor *ClrScr* *FontColor* *GotoXY* *ShowScr*
Константы цветов *Цвет*

3.3.16. PUTS

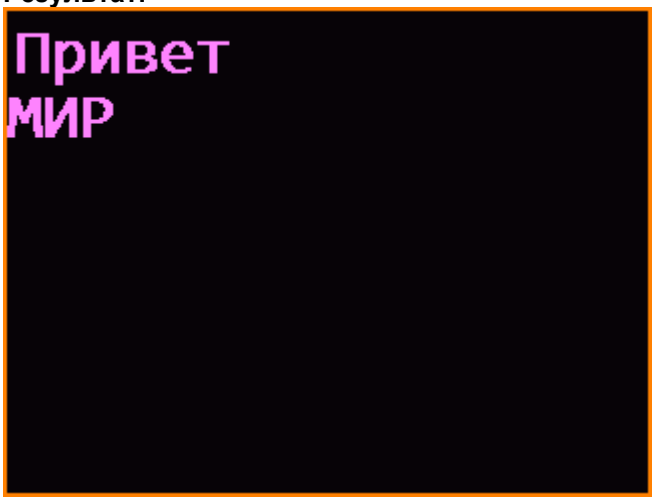
```
#include <sat500.h>

void puts(char * text);
```

Вывод строки на дисплей в текущую позицию курсора.
Текст выводится с использованием цвета текущей кисти *BrushColor* (задаёт цвет фона) и цвета шрифта *FontColor*.
При выводе обрабатываются управляющие символы:
\n - Перевод строки и возврат каретки (увеличение координату y курсора на величину высоты текущего шрифта и установка координаты курсора x в нулевое значение)
\r - Возврат каретки (установка координаты курсора x в нулевое значение)
\t - Установка координаты курсора x в следующее значение кратное 4-м символам текущего размера

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    FontColor=RGB(255,128,255); //Цвет шрифта RED=255 GREEN=128 BLUE=255
    GotoXY(4,4);                 //Курсор в позицию x=4,  y=4
    puts("Привет\n");            //Вывод строки и перевод строки
    puts("МИР");                 //Вывод строки
    ShowScr();                   //Вывести на дисплей
    while(1);                    //Бесконечная пауза
}
```

Результат:



См. также
BrushColor *ClrScr* *FontColor* *GotoXY* *ShowScr*
Константы цветов *Цвет* *RGB*

3.3.17. RGB

```
#include <sat500.h>

short int RGB (unsigned char R,unsigned char G,unsigned char B);
```

Получение цвета по заданным значениям составляющих R, G и B.
R - значение составляющей красного цвета [0...255]
G - значение составляющей зелёного цвета [0...255]
B - значение составляющей синего цвета [0...255]

Примечание:
Так как формат цвета в дисплее использует специальный формат цвета (5 бит красного цвета, 6 бит зелёного и 5 бит синего), то для получения цвета функция использует значения старших битов цветов R, G и B.

Примеры

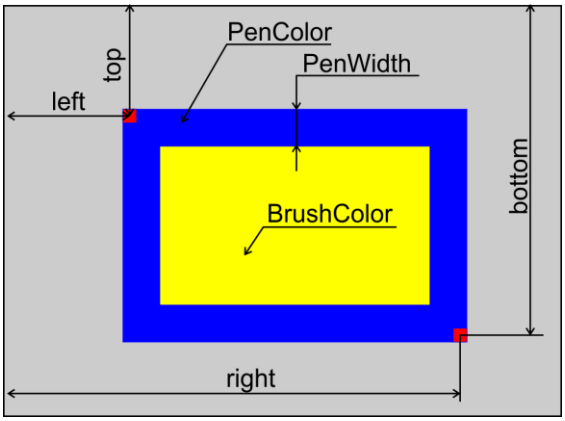
```
Черный цвет:      RGB(0,0,0)
Белый цвет:       RGB(255,255,255)
Красный цвет:     RGB(255,0,0)
Зелёный цвет:     RGB(0,255,0)
Синий цвет:       RGB(0,0,255)
```

См. также
Цвет *Константы цветов* *BrushColor* *FontColor* *PenColor*

3.3.18. RECTANGLE

```
#include <sat500.h>

void Rectangle (short int left,short int top,short int right,short int bot-
tom);
```

Рисование *прямоугольника*. Контур обводится пером, а внутренняя часть заполняется цветом кисти *BrushColor*.
Цвет пера задаётся значением переменной *PenColor*.
Толщина пера в пикселях задаётся значением переменной *PenWidth.*
Прямоугольник вписывается в заданные координаты left,top,right,bottom:



Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    BrushColor=COLOR_WHITE;      //Цвет кисти - белый
    PenColor=COLOR_RED;          //Цвет пера - красный
    PenWidth=5;                  //Толщина пера - 5 пикселей
    Rectangle(24,24,204,104);    //Нарисовать прямоугольник кистью и пером
    ShowScr();                   //Вывести нарисованное на дисплей
    while(1);                    //Бесконечная пауза
}
```

Результат:

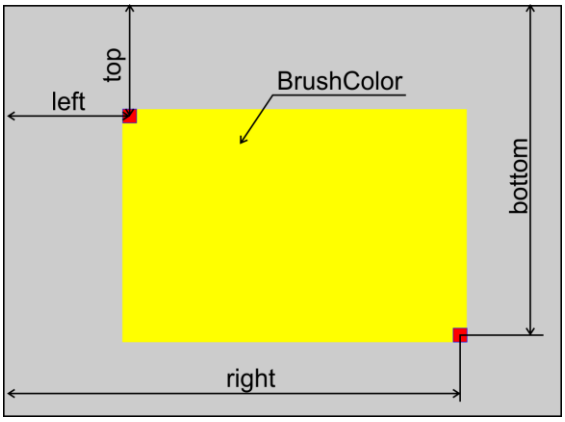
См. также

BrushColor ClrScr PenColor PenWidth ShowScr
Константы цветов Цвет RGB

3.3.19. RECTANGLEBRUSH

```
#include <sat500.h>

void RectangleBrush (short int left,short int top,short int right,short int
bottom);
```

Рисование *прямоугольника*, заполненного цветом кисти *BrushColor*.
Перо не используется.
Прямоугольник вписывается в заданные координаты left,top,right,bottom:



Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;          //Цвет кисти - чёрный
    ClrScr();                        //Очистить экран чёрным цветом
    BrushColor=COLOR_BLUE;           //Цвет кисти - синий
    RectangleBrush(24,24,204,104);   //Нарисовать прямоугольник кистью
    ShowScr();                       //Вывести нарисованное на дисплей
    while(1);                        //Бесконечная пауза
}
```

Результат:

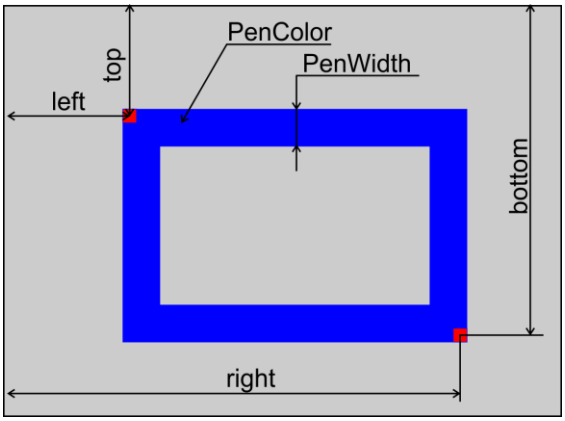
См. также
BrushColor ClrScr ShowScr
Константы цветов Цвет RGB

3.3.20. RECTANGLEPEN

```
#include <sat500.h>
```

```
void RectanglePen (short int left,short int top,short int right,short int bot-
tom);
```

Рисование контура *прямоугольника* при помощи пера.
Внутренняя часть прямоугольника не заполняется. Кисть не используется
Цвет пера задаётся значением переменной *PenColor*.
Толщина пера в пикселях задаётся значением переменной *PenWidth.*
Прямоугольник вписывается в заданные координаты left,top,right,bottom:



Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    PenColor=COLOR_GREEN;        //Цвет пера - зелёный
    PenWidth=2;                  //Толщина пера - 2 пикселя
    RectanglePen(24,24,204,104);//Нарисовать прямоугольник пером
    ShowScr();                   //Вывести нарисованное на дисплей
    while(1);                    //Бесконечная пауза
}
```

Результат:

См. также
BrushColor *ClrScr* *PenColor* *PenWidth* *ShowScr*
Константы цветов *Цвет* *RGB*

3.3.21. SETPIXEL

```
#include <sat500.h>

void SetPixel(short int x,short int y,short int color);
```

Установка цвета пикселя дисплея.
x - координата по горизонтали [0...*SCREEN_WIDTH-1*]
y - координата по вертикали [0...*SCREEN_HEIGHT-1*]
color - цвет в формате CAT-500

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;                  //Цвет кисти - чёрный
    ClrScr();                                //Очистить экран чёрным цветом
    srand(GetTickCount());                   //Инициализация случайного генера-
тора
Loop:;
    for(int i=0;i<1000;i++)                  //Вывести 1000 точек
    {
        short int x=rand()%SCREEN_WIDTH;     //Координата x - случайное число в
пределах ширины экрана
        short int y=rand()%SCREEN_HEIGHT;    //Координата y - случайное число в
пределах высоты экрана
        short int color=rand()%0x10000);     //Случайный цвет
        SetPixel(x,y,color);                 //Пиксель с координатами x,y уста-
новить в цвет color
    }
    ShowScr();                               //Вывести на дисплей
    goto Loop;                               //Бесконечное заполнение
}
```

Результат:

См. также

BrushColor ClrScr GotoXY ShowScr
srand rand Константы цветов Цвет RGB Размеры дисплея

3.3.22. SHOWSCR

```
#include <sat500.h>

void ShowScr (void);
```

Завершение рисования экрана (копирование видеообласти памяти в контроллер дисплея).
Эта функция должна вызываться в момент времени, когда нужно обновить содержимое дисплея.

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_WHITE;      //Цвет кисти - белый
    ClrScr();                    //Очистить экран цветом кисти
    FontColor=COLOR_BLACK;       //Цвет шрифта - чёрный
    puts("Привет");              //Вывод строки
    ShowScr();                   //Вывести на дисплей
    while(1);                    //Бесконечная пауза
}
```

Результат:

См. также
BrushColor *ClrScr* *FontColor* *puts*
Константы цветов *Цвет*

3.3.23. ЦВЕТ

Цвет представляется двухбайтовым числом со следующими битовыми полями.

Бит	*15*	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Цвет	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

R4...R0 - Красный цвет
G5...G0 - Зелёный цвет
B4...B0 - Синий цвет

Для формирования цвета по трём составляющим можно использовать функцию *RGB*.

См. также
Константы цветов *BrushColor* *FontColor* *PenColor* *RGB*

3.3.24. КОНСТАНТЫ ЦВЕТОВ

После включения заголовочного файла:

```
#include <sat500.h>
```

доступны следующие константы цветов:

Имя	*Цвет*	*Значение*
COLOR_RED	Красный	0xF800
COLOR_BLUE	Синий	0x001F
COLOR_GREEN	Зелёный	0x07E0
COLOR_YELLOW	Жёлтый	0xFFE0
COLOR_BLACK	Чёрный	0x0000
COLOR_WHITE	Белый	0xFFFF
COLOR_GRAY	Серый	0x8410
COLOR_LIGHTGRAY	Светло-серый	0xC618
COLOR_SILVER	Светло-серый	0xC618

COLOR_CYAN	Сине-зелёный	0x07FF
COLOR_MAGENTA	Фиолетовый	0xF81F
COLOR_ORANGE	Оранжевый	0xF300

Для формирования произвольного цвета по трём составляющим можно использовать функцию *RGB*.

См. также
Цвет *BrushColor* *FontColor* *PenColor* *RGB*

3.3.25. РАЗМЕРЫ ДИСПЛЕЯ

После включения заголовочного файла:

```
#include <sat500.h>
```

доступны следующие константы размера дисплея:

```
#define SCREEN_WIDTH   320u
#define SCREEN_HEIGHT  240u
```

См. также
GotoXY

3.4. ВРЕМЯ

Функции времени объявлены в заголовочном файле `<sat500.h>`

Функция	***Описание***
unsigned int GetTickCount (void)	Получение времени, прошедшего с момента запуска контроллера, в миллисекундах
*void GetTime (unsigned char * hour,unsigned char * min,unsigned char * sec)*	Получение текущего календарного времени
*void GetDate (unsigned char * date,unsigned char * month,unsigned short * year)*	Получение текущей календарной даты
*void GetDateTime (unsigned char * date,unsigned char * month,unsigned short * year,unsigned char * hour,unsigned char * min,unsigned char * sec)*	Получение текущего календарного времени и текущей календарной даты
unsigned char SecOfTime(unsigned int t)	Получение значения секунд из Unix-времени
unsigned char MinOfTime (unsigned int t)	Получение значения минут из Unix-времени.
unsigned char HourOfTime (unsigned int t)	Получение значения часов из Unix-времени.
unsigned char DayOfTime (unsigned int t)	Получение значения номера дня месяца из Unix-времени
unsigned char MonthOfTime (unsigned int t)	Получение значения номера месяца из Unix-времени
unsigned short YearOfTime (unsigned int t)	Получение значения номера года из Unix-времени
unsigned char WeekDayOfTime (unsigned int t)	Получение значения номера дня недели из Unix-времени

unsigned short YearDayOfTime (unsigned int t)	Получение значения номера дня года из Unix-времени
unsigned int MkTime (unsigned char hour,unsigned char min,unsigned char sec,unsigned char day,unsigned char month,unsigned short int year	Получение значения Unix-времени из задаваемого значения времени и даты
unsigned int TimeOfSunrise (unsigned int lat,unsigned int lon,unsigned char day,unsigned char month,unsigned short int year)	Получение значения времени восхода солнца в определённой географической точке в указанный день
unsigned int TimeOfSunset (unsigned int lat,unsigned int lon,unsigned char day,unsigned char month,unsigned short int year)	Получение значения времени захода солнца в определённой географической точке в указанный день

Производственный календарь

3.4.1. GETTICKCOUNT

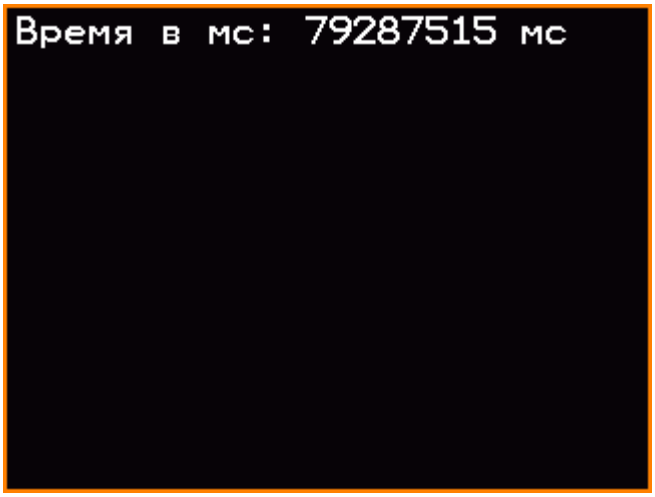
```
#include <sat500.h>

unsigned int GetTickCount(void);
```

Получение времени, прошедшего с момента запуска контроллера, в миллисекундах.

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;        //Цвет кисти - чёрный
    ClrScr();                      //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;         //Цвет шрифта - белый
    BigFont=0;                     //Установить обычный шрифт
                                   //(12x16, 12 пикселей  по горизонтали, 16
пикселей  по вертикали)
    while(1)
    {
        GotoXY(4,4);                                    //Курсор в позицию x=4,
y=4
        printf("Время в мс: %u мс",GetTickCount());     //Вывод на дисплей Get-
TickCount()
        ShowScr();                                      //Вывести на дисплей
    }
}
```

Результат:



См. также
BrushColor *ClrScr* *FontColor* *BigFont* *GotoXY* *printf* *ShowScr*
Константы цветов *Цвет*

3.4.2. GETTIME

```
#include <sat500.h>

void GetTime(unsigned char * hour,unsigned char * min,unsigned char * sec);
```

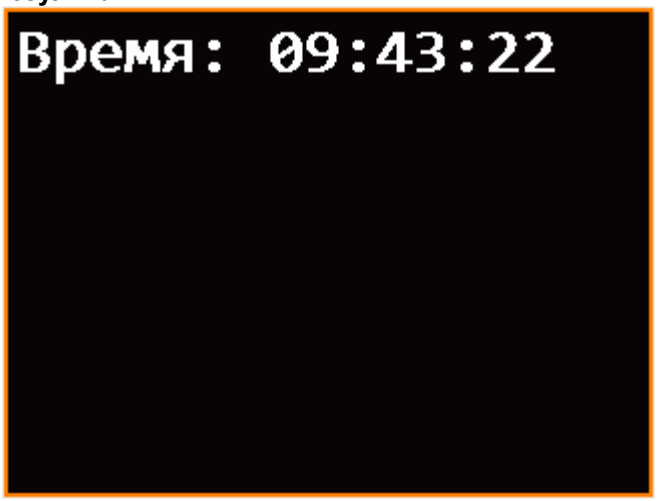
Получение текущего календарного времени.
*unsigned char * hour* - указатель на переменную, которая будет заполнена количеством часов текущего времени
*unsigned char * min* - указатель на переменную, которая будет заполнена количеством минут текущего времени
*unsigned char * sec* - указатель на переменную, которая будет заполнена количеством часов текущего времени

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;       //Цвет шрифта - белый
    while(1)
    {
        unsigned char hour,min,sec;
        GetTime(&hour,&min,&sec);                        //Получить текущее
время
        GotoXY(4,4);                                     //Курсор в позицию
x=4,  y=4
        printf("Время: %02d:%02d:%02d",hour,min,sec);    //Вывод на дисплей те-
кущего времени
        ShowScr();                                       //Вывести на дисплей
    }
```

```
}
```

Результат:



См. также
BrushColor *ClrScr* *FontColor* *GotoXY* *printf* *ShowScr*
Константы цветов *Цвет*

3.4.3. GETDATE

```
#include <sat500.h>

void GetDate(unsigned char * date,unsigned char * month,unsigned short *
year);
```

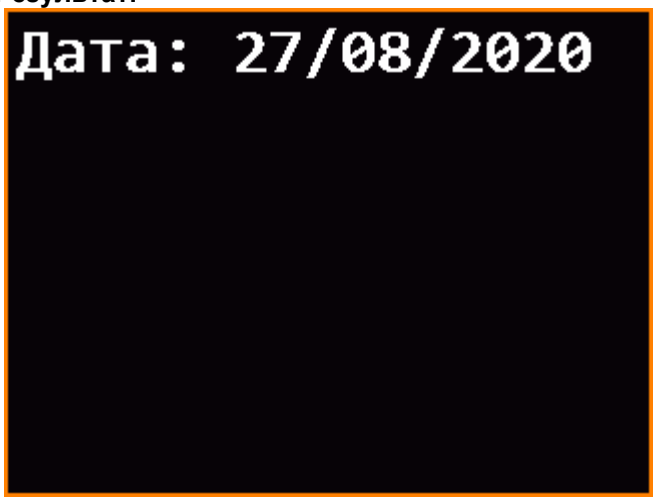
Получение текущей календарной даты.
*unsigned char * date* - указатель на переменную, которая будет заполнена текущей датой
*unsigned char * month* - указатель на переменную, которая будет заполнена номером месяца текущей даты
*unsigned short * year* - указатель на переменную, которая будет заполнена количеством текущим годом

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;       //Цвет шрифта - белый
    while(1)
    {
        unsigned char date,month;
        unsigned short year;
        GetDate(&date,&month,&year);                    //Получить текущую
дату
        GotoXY(4,4);                                    //Курсор в позицию
x=4,  y=4
```

```
        printf("Дата: %02d/%02d/%04d",date,month,year); //Вывод на дисплей
даты
        ShowScr();                                      //Вывести на дисплей
    }
}
```

Результат:



См. также
BrushColor ClrScr FontColor GotoXY printf ShowScr
Константы цветов Цвет

3.4.4. GETDATETIME

```
#include <sat500.h>

void GetDateTime(unsigned char * date,unsigned char * month,unsigned short *
year,unsigned char * hour,unsigned char * min,unsigned char * sec);
```

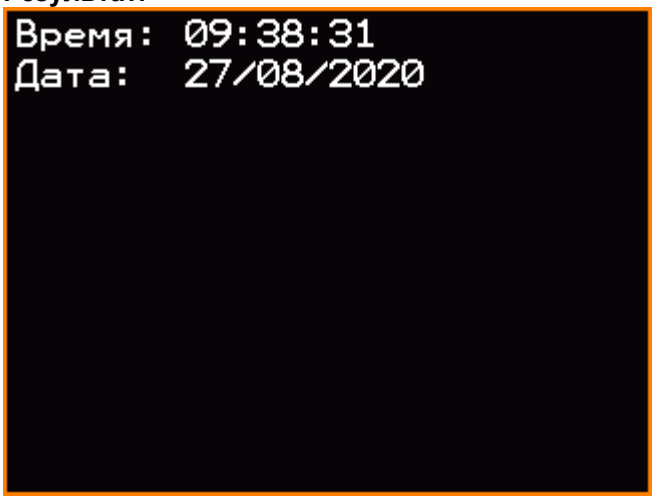
Получение текущего календарного времени и текущей календарной даты.
*unsigned char * hour* - указатель на переменную, которая будет заполнена количеством часов текущего времени
*unsigned char * min* - указатель на переменную, которая будет заполнена количеством минут текущего времени
*unsigned char * sec* - указатель на переменную, которая будет заполнена количеством часов текущего времени
*unsigned char * date* - указатель на переменную, которая будет заполнена текущей датой
*unsigned char * month* - указатель на переменную, которая будет заполнена номером месяца текущей даты
*unsigned short * year* - указатель на переменную, которая будет заполнена количеством текущим годом

Пример:

```
#include <sat500.h>
void main(void)
{
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;       //Цвет шрифта - белый
```

```
    BigFont=0;                  //Установить обычный шрифт (12x16, 12 пикселей
по горизонтали, 16 пикселей  по вертикали)
    while(1)
    {
        unsigned char hour,min,sec;
        unsigned char date,month;
        unsigned short year;
        GetDateTime(&date,&month,&year,&hour,&min,&sec);    //Получить текущую
дату и время
        GotoXY(4,4);                                        //Курсор в позицию
x=4,  y=4
        printf("Время: %02d:%02d:%02d",hour,min,sec);       //Вывод на дисплей
текущего времени
        GotoXY(4,24);                                       //Курсор в позицию
x=4,  y=24
        printf("Дата:  %02d/%02d/%04d",date,month,year);    //Вывод на дисплей
даты
        ShowScr();                                          //Вывести на дис-
плей
    }
}
```

Результат:



См. также
BrushColor *ClrScr* *FontColor* *BigFont* *GotoXY* *printf* *ShowScr*
Константы цветов *Цвет*

3.4.5. SECOFTIME

```
#include <sat500.h>

unsigned char SecOfTime(unsigned int t);
```

Получение значения секунд из Unix-времени.
`unsigned int t` – Unix-время

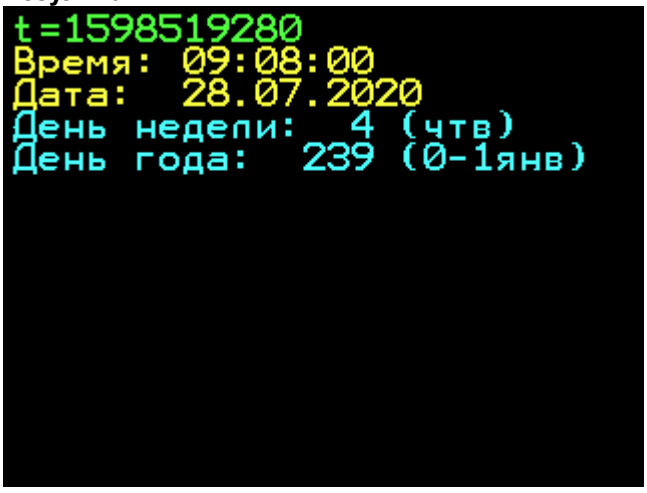
Возвращаемое значение: значение секунд во времени t.
Возможные значения: 0-59

Unix-время (англ. Unix time, также POSIX-время) — система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах. Определяется как количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года (четверг); этот момент называют «эпохой Unix».

Пример:

```
#include <sat500.h>

const char * WeekDays[]=
{"вск","пнд","втр","срд","чтв","птн","сбт"};

void main(void)
{
    ClrScr();
    BigFont=false;
    unsigned int t=MkTime(9,8,0,27,8,2020);
    FontColor=COLOR_GREEN;
    printf("t=%u",t);
    unsigned char sec=SecOfTime(t);
    unsigned char min=MinOfTime(t);
    unsigned char hour=HourOfTime(t);
    unsigned char day=DayOfTime(t);
    unsigned char month=MonthOfTime(t);
    unsigned short year=YearOfTime(t);
    unsigned char wday=WeekDayOfTime(t);
    unsigned char yday=YearDayOfTime(t);
    FontColor=COLOR_YELLOW;
    printf("\nВремя: %02d:%02d:%02d",hour,min,sec);
    printf("\nДата:  %02d.%02d.%04d",day,month,year);
    FontColor=COLOR_CYAN;
    printf("\nДень недели:  %d (%s)",wday,WeekDays[wday]);
    printf("\nДень года:  %d (0-1янв)",yday);
    ShowScr();
    while(1);
}
```

Результат:



См. также

MinOfTime HourOfTime DayOfTime MonthOfTime YearOfTime
WeekDayOfTime YearDayOfTime MkTime

3.4.6. MINOFTIME

```
#include <sat500.h>

unsigned char MinOfTime(unsigned int t);
```

Получение значения минут из Unix-времени.
`unsigned int t` – Unix-время

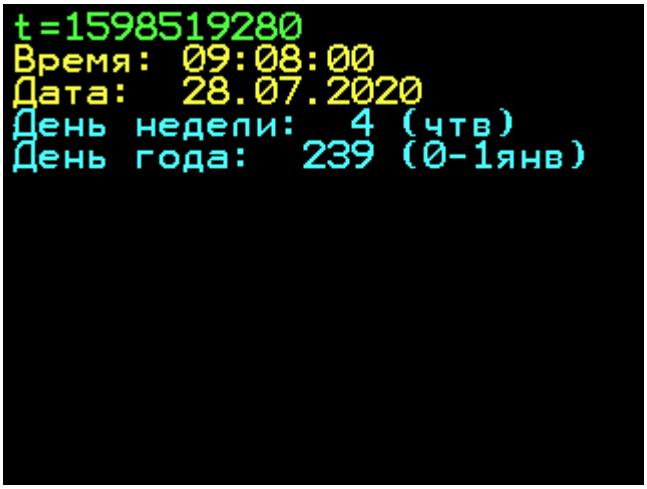
Возвращаемое значение: значение минут во времени *t*.
Возможные значения: 0-59

Unix-время (англ. Unix time, также POSIX-время) — система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах. Определяется как количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года (четверг); этот момент называют «эпохой Unix».

Пример:

```
#include <sat500.h>

const char * WeekDays[]=
{"вск","пнд","втр","срд","чтв","птн","сбт"};

void main(void)
{
    ClrScr();
    BigFont=false;
    unsigned int t=MkTime(9,8,0,27,8,2020);
    FontColor=COLOR_GREEN;
    printf("t=%u",t);
    unsigned char sec=SecOfTime(t);
    unsigned char min=MinOfTime(t);
    unsigned char hour=HourOfTime(t);
    unsigned char day=DayOfTime(t);
    unsigned char month=MonthOfTime(t);
    unsigned short year=YearOfTime(t);
    unsigned char wday=WeekDayOfTime(t);
    unsigned char yday=YearDayOfTime(t);
    FontColor=COLOR_YELLOW;
    printf("\nВремя: %02d:%02d:%02d",hour,min,sec);
    printf("\nДата:  %02d.%02d.%04d",day,month,year);
    FontColor=COLOR_CYAN;
    printf("\nДень недели:  %d (%s)",wday,WeekDays[wday]);
    printf("\nДень года:  %d (0-1янв)",yday);
    ShowScr();
    while(1);
}
```

Результат:



См. также
SecOfTime HourOfTime DayOfTime MonthOfTime YearOfTime
WeekDayOfTime YearDayOfTime MkTime

3.4.7. HOUROFTIME

```
#include <sat500.h>

unsigned char SecOfTime(unsigned int t);
```

Получение значения часов из Unix-времени.
`unsigned int t` – Unix-время

Возвращаемое значение: значение часов во времени *t*.
Возможные значения: 0-23

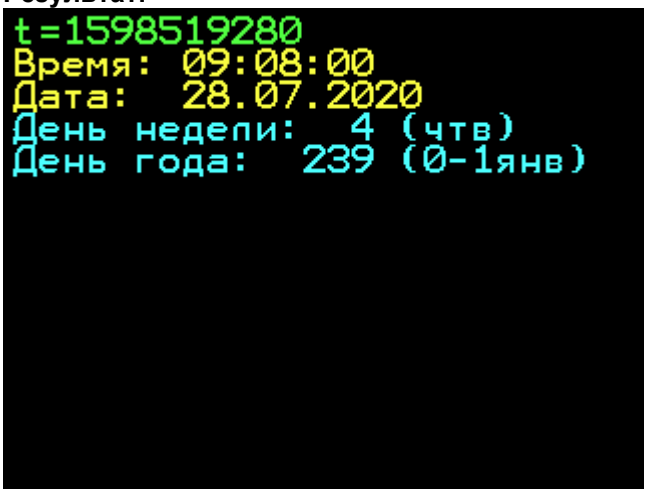
Unix-время (англ. Unix time, также POSIX-время) — система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах. Определяется как количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года (четверг); этот момент называют «эпохой Unix».

Пример:

```
#include <sat500.h>

const char * WeekDays[]=
{"вск","пнд","втр","срд","чтв","птн","сбт"};

void main(void)
{
    ClrScr();
    BigFont=false;
    unsigned int t=MkTime(9,8,0,27,8,2020);
    FontColor=COLOR_GREEN;
    printf("t=%u",t);
    unsigned char sec=SecOfTime(t);
    unsigned char min=MinOfTime(t);
    unsigned char hour=HourOfTime(t);
    unsigned char day=DayOfTime(t);
```

```
    unsigned char month=MonthOfTime(t);
    unsigned short year=YearOfTime(t);
    unsigned char wday=WeekDayOfTime(t);
    unsigned char yday=YearDayOfTime(t);
    FontColor=COLOR_YELLOW;
    printf("\nВремя: %02d:%02d:%02d",hour,min,sec);
    printf("\nДата:  %02d.%02d.%04d",day,month,year);
    FontColor=COLOR_CYAN;
    printf("\nДень недели:  %d (%s)",wday,WeekDays[wday]);
    printf("\nДень года:  %d (0-1янв)",yday);
    ShowScr();
    while(1);
}
```

Результат:



См. также
SecOfTime MinOfTime DayOfTime MonthOfTime YearOfTime
WeekDayOfTime YearDayOfTime MkTime

3.4.8. DAYOFTIME

```
#include <sat500.h>

unsigned char DayOfTime(unsigned int t);
```

Получение значения номера дня месяца из Unix-времени.
`unsigned int t` – Unix-время

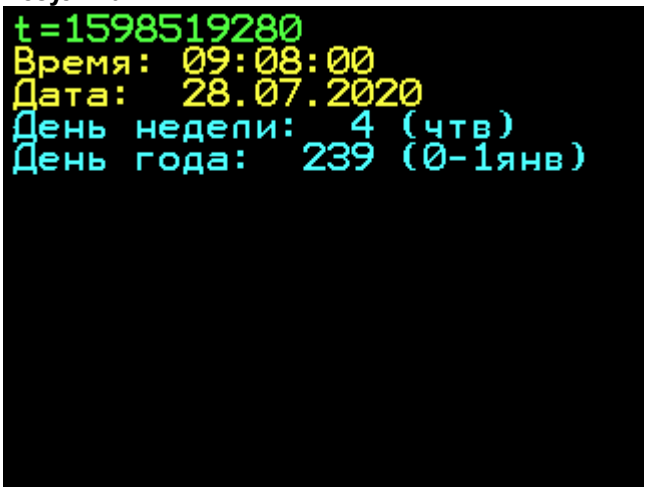
Возвращаемое значение: значение номера дня месяца во времени *t*.
Возможные значения: 1-31

Unix-время (англ. Unix time, также POSIX-время) — система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах. Определяется как количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года (четверг); этот момент называют «эпохой Unix».

Пример:

```
#include <sat500.h>

const char * WeekDays[]=
{"вск","пнд","втр","срд","чтв","птн","сбт"};

void main(void)
{
    ClrScr();
    BigFont=false;
    unsigned int t=MkTime(9,8,0,27,8,2020);
    FontColor=COLOR_GREEN;
    printf("t=%u",t);
    unsigned char sec=SecOfTime(t);
    unsigned char min=MinOfTime(t);
    unsigned char hour=HourOfTime(t);
    unsigned char day=DayOfTime(t);
    unsigned char month=MonthOfTime(t);
    unsigned short year=YearOfTime(t);
    unsigned char wday=WeekDayOfTime(t);
    unsigned char yday=YearDayOfTime(t);
    FontColor=COLOR_YELLOW;
    printf("\nВремя: %02d:%02d:%02d",hour,min,sec);
    printf("\nДата:  %02d.%02d.%04d",day,month,year);
    FontColor=COLOR_CYAN;
    printf("\nДень недели:  %d (%s)",wday,WeekDays[wday]);
    printf("\nДень года:  %d (0-1янв)",yday);
    ShowScr();
    while(1);
}
```

Результат:



См. также
SecOfTime MinOfTime HourOfTime MonthOfTime YearOfTime
WeekDayOfTime YearDayOfTime MkTime

3.4.9. MONTHOFTIME

```
#include <sat500.h>
```

```
unsigned char MonthOfTime(unsigned int t);
```

Получение значения номера месяца из Unix-времени.
`unsigned int t` – Unix-время

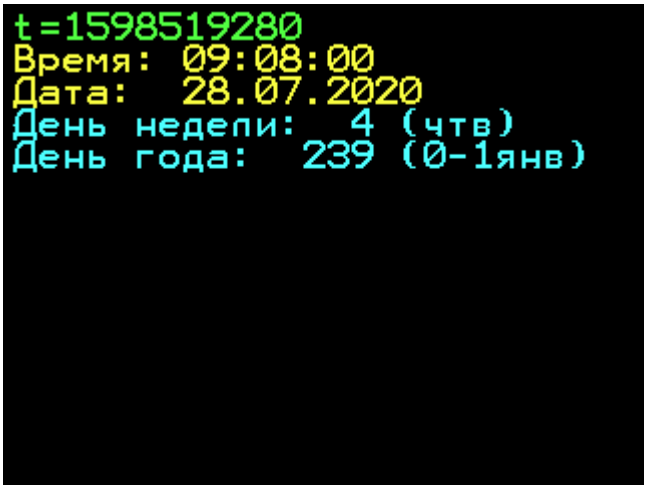
Возвращаемое значение: значение номера месяца во времени *t*.
Возможные значения: 1-12

Unix-время (англ. Unix time, также POSIX-время) — система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах. Определяется как количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года (четверг); этот момент называют «эпохой Unix».

Пример:

```
#include <sat500.h>

const char * WeekDays[]=
{"вск","пнд","втр","срд","чтв","птн","сбт"};

void main(void)
{
    ClrScr();
    BigFont=false;
    unsigned int t=MkTime(9,8,0,27,8,2020);
    FontColor=COLOR_GREEN;
    printf("t=%u",t);
    unsigned char sec=SecOfTime(t);
    unsigned char min=MinOfTime(t);
    unsigned char hour=HourOfTime(t);
    unsigned char day=DayOfTime(t);
    unsigned char month=MonthOfTime(t);
    unsigned short year=YearOfTime(t);
    unsigned char wday=WeekDayOfTime(t);
    unsigned char yday=YearDayOfTime(t);
    FontColor=COLOR_YELLOW;
    printf("\nВремя: %02d:%02d:%02d",hour,min,sec);
    printf("\nДата:  %02d.%02d.%04d",day,month,year);
    FontColor=COLOR_CYAN;
    printf("\nДень недели:  %d (%s)",wday,WeekDays[wday]);
    printf("\nДень года:  %d (0-1янв)",yday);
    ShowScr();
    while(1);
}
```

Результат:



См. также
SecOfTime MinOfTime HourOfTime DayOfTime YearOfTime
WeekDayOfTime YearDayOfTime MkTime

3.4.10. YEAROFTIME

```
#include <sat500.h>

unsigned short int YearOfTime(unsigned int t);
```

Получение значения номера года из Unix-времени.
unsigned int t – Unix-время

Возвращаемое значение: значение номера года во времени *t*.
Возможные значения: 1970...

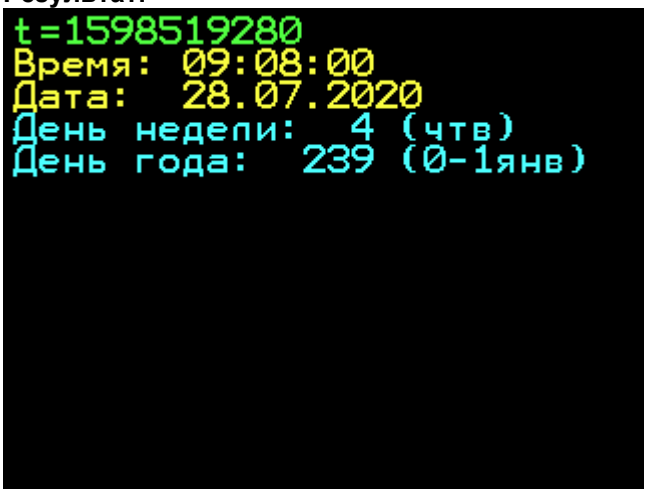
Unix-время (англ. Unix time, также POSIX-время) — система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах. Определяется как количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года (четверг); этот момент называют «эпохой Unix».

Пример:

```
#include <sat500.h>

const char * WeekDays[]=
{"вск","пнд","втр","срд","чтв","птн","сбт"};

void main(void)
{
    ClrScr();
    BigFont=false;
    unsigned int t=MkTime(9,8,0,27,8,2020);
    FontColor=COLOR_GREEN;
    printf("t=%u",t);
    unsigned char sec=SecOfTime(t);
    unsigned char min=MinOfTime(t);
    unsigned char hour=HourOfTime(t);
    unsigned char day=DayOfTime(t);
```

```
    unsigned char month=MonthOfTime(t);
    unsigned short year=YearOfTime(t);
    unsigned char wday=WeekDayOfTime(t);
    unsigned char yday=YearDayOfTime(t);
    FontColor=COLOR_YELLOW;
    printf("\nВремя: %02d:%02d:%02d",hour,min,sec);
    printf("\nДата:  %02d.%02d.%04d",day,month,year);
    FontColor=COLOR_CYAN;
    printf("\nДень недели:  %d (%s)",wday,WeekDays[wday]);
    printf("\nДень года:  %d (0-1янв)",yday);
    ShowScr();
    while(1);
}
```

Результат:



См. также
SecOfTime *MinOfTime* *HourOfTime* *DayOfTime* *MonthOfTime*
WeekDayOfTime *YearDayOfTime* *MkTime*

3.4.11. WEEKDAYOFTIME

```
#include <sat500.h>

unsigned char WeekDayOfTime(unsigned int t);
```

Получение значения номера дня недели из Unix-времени.
`unsigned int t` – Unix-время

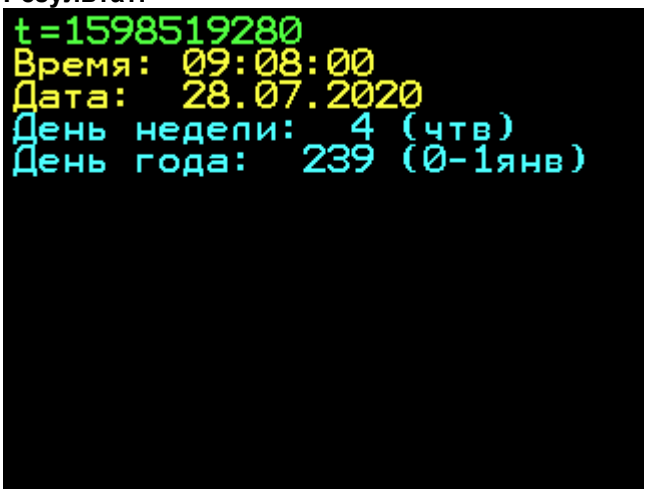
Возвращаемое значение: значение номера дня недели во времени *t*.
Возможные значения: 0-вск,1-пнд,2-втр,3-срд,4-чтв,5-птн,6-сбт

Unix-время (англ. Unix time, также POSIX-время) — система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах. Определяется как количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года (четверг); этот момент называют «эпохой Unix».

Пример:

```
#include <sat500.h>

const char * WeekDays[]=
{"вск","пнд","втр","срд","чтв","птн","сбт"};

void main(void)
{
    ClrScr();
    BigFont=false;
    unsigned int t=MkTime(9,8,0,27,8,2020);
    FontColor=COLOR_GREEN;
    printf("t=%u",t);
    unsigned char sec=SecOfTime(t);
    unsigned char min=MinOfTime(t);
    unsigned char hour=HourOfTime(t);
    unsigned char day=DayOfTime(t);
    unsigned char month=MonthOfTime(t);
    unsigned short year=YearOfTime(t);
    unsigned char wday=WeekDayOfTime(t);
    unsigned char yday=YearDayOfTime(t);
    FontColor=COLOR_YELLOW;
    printf("\nВремя: %02d:%02d:%02d",hour,min,sec);
    printf("\nДата:  %02d.%02d.%04d",day,month,year);
    FontColor=COLOR_CYAN;
    printf("\nДень недели:  %d (%s)",wday,WeekDays[wday]);
    printf("\nДень года:  %d (0-1янв)",yday);
    ShowScr();
    while(1);
}
```

Результат:



См. также
SecOfTime MinOfTime HourOfTime DayOfTime MonthOfTime YearOfTime YearDayOfTime MkTime

3.4.12. YEARDAYOFTIME

```
#include <sat500.h>
```

```
unsigned short int YearDayOfTime(unsigned int t);
```

Получение значения номера дня года из Unix-времени.
`unsigned int t` – Unix-время

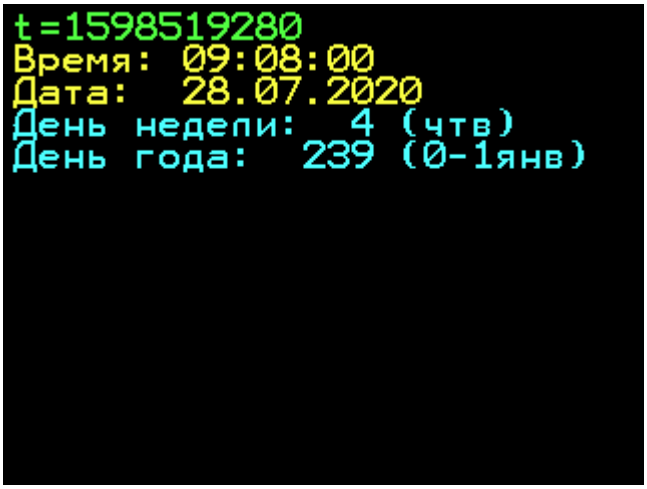
Возвращаемое значение: значение номера дня года во времени t.
Возможные значения: 0-365

Unix-время (англ. Unix time, также POSIX-время) — система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах. Определяется как количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года (четверг); этот момент называют «эпохой Unix».

Пример:

```
#include <sat500.h>

const char * WeekDays[]=
{"вск","пнд","втр","срд","чтв","птн","сбт"};

void main(void)
{
    ClrScr();
    BigFont=false;
    unsigned int t=MkTime(9,8,0,27,8,2020);
    FontColor=COLOR_GREEN;
    printf("t=%u",t);
    unsigned char sec=SecOfTime(t);
    unsigned char min=MinOfTime(t);
    unsigned char hour=HourOfTime(t);
    unsigned char day=DayOfTime(t);
    unsigned char month=MonthOfTime(t);
    unsigned short year=YearOfTime(t);
    unsigned char wday=WeekDayOfTime(t);
    unsigned char yday=YearDayOfTime(t);
    FontColor=COLOR_YELLOW;
    printf("\nВремя: %02d:%02d:%02d",hour,min,sec);
    printf("\nДата:  %02d.%02d.%04d",day,month,year);
    FontColor=COLOR_CYAN;
    printf("\nДень недели:  %d (%s)",wday,WeekDays[wday]);
    printf("\nДень года:  %d (0-1янв)",yday);
    ShowScr();
    while(1);
}
```

Результат:



См. также
SecOfTime *MinOfTime* *HourOfTime* *DayOfTime* *MonthOfTime* *YearOfTime* *WeekDayOfTime* *MkTime*

3.4.13. MKTIME

```
#include <sat500.h>

unsigned int MkTime(unsigned char hour,unsigned char min,unsigned char sec,un-
signed char day,unsigned char month,unsigned short int year);
```

Получение значения Unix-времени из задаваемого значения времени и даты.
Аргументы функции:
unsigned char hour – часы (0-23)
unsigned char min – минуты (0-59)
unsigned char sec – секунды (0-59)
unsigned char day – день месяца (1-31)
unsigned char month – номер месяца(1-12)
unsigned short int year – год (1970...)

Возвращаемое значение: значение секунд (Unix-время).

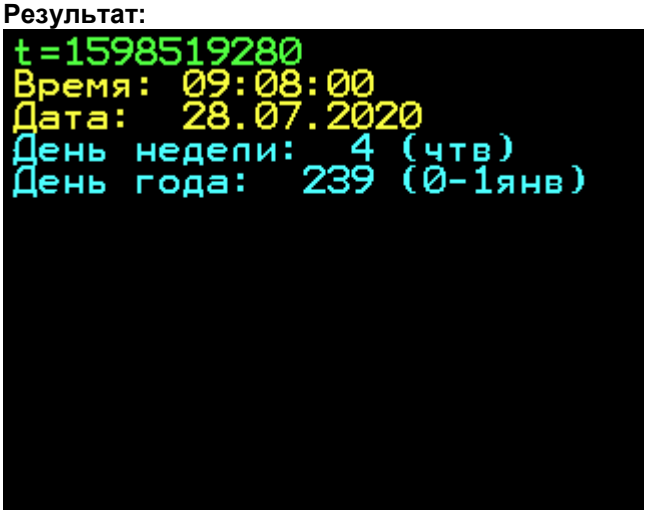
Unix-время (англ. Unix time, также POSIX-время) — система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах. Определяется как количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года (четверг); этот момент называют «эпохой Unix».

Пример:

```
#include <sat500.h>

const char * WeekDays[]=
{"вск","пнд","втр","срд","чтв","птн","сбт"};

void main(void)
{
    ClrScr();
    BigFont=false;
```

```
    unsigned int t=MkTime(9,8,0,27,8,2020);
    FontColor=COLOR_GREEN;
    printf("t=%u",t);
    unsigned char sec=SecOfTime(t);
    unsigned char min=MinOfTime(t);
    unsigned char hour=HourOfTime(t);
    unsigned char day=DayOfTime(t);
    unsigned char month=MonthOfTime(t);
    unsigned short year=YearOfTime(t);
    unsigned char wday=WeekDayOfTime(t);
    unsigned char yday=YearDayOfTime(t);
    FontColor=COLOR_YELLOW;
    printf("\nВремя: %02d:%02d:%02d",hour,min,sec);
    printf("\nДата:  %02d.%02d.%04d",day,month,year);
    FontColor=COLOR_CYAN;
    printf("\nДень недели:  %d (%s)",wday,WeekDays[wday]);
    printf("\nДень года:  %d (0-1янв)",yday);
    ShowScr();
    while(1);
}
```

Результат:



См. также
SecOfTime MinOfTime HourOfTime DayOfTime MonthOfTime YearOfTime WeekDayOfTime YearDayOfTime

3.4.14. TIMEOFSUNRISE

```
#include <sat500.h>

unsigned int TimeOfSunrise(unsigned int lat,unsigned int lon,unsigned char
day,unsigned char month,unsigned short int year);
```

Получение значения времени восхода солнца в определённой географической точке в указанный день.

Аргументы функции:
`unsigned int lat` - Широта географической точки в тысячных долях градуса
`unsigned int lon` - Долгота географической точки в тысячных долях градуса

unsigned char day - День месяца (1-31)
unsigned char month - Номер месяца(1-12)
unsigned short int year - Год (2022 ...)

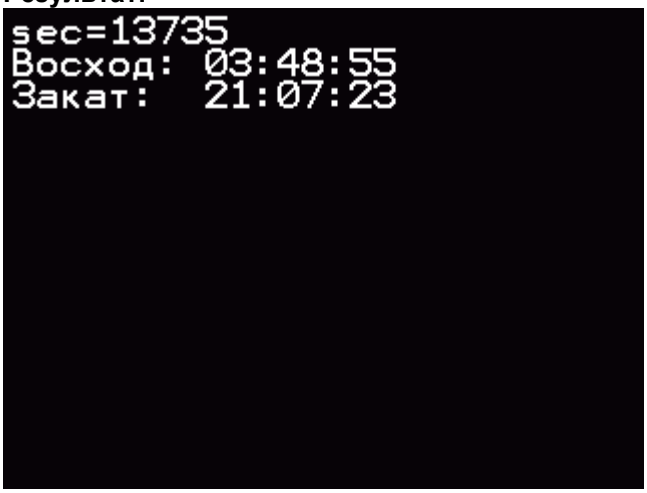
Возвращаемое значение: количество секунд от начала суток.
Если возвращается нулевое значение, то вычислить значение не удалось.
Примечание: Функция возвращает время по Гринвичу, поэтому необходимо скорректировать полученное значение на величину расхождения времени с географической точки для которой выполнен расчёт. В простейшем случае это может быть часовой пояс географической точки.

Пример:

```
//Определение времени заката и рассвета г.Москвы с координатами:
//Широта:  55,752
//Долгота: 37,615

#include <sat500.h>

typedef unsigned int  UINT;

//Коррекция часов для Москвы
#define HOUR_CORRECTION 3

void main(void)
{
    UINT utc;
    ClrScr();                //Очистить экран
    BigFont=0;               //Обычный шрифт 12x16 (не крупный)
    //6 июня 2022 года
    utc=TimeOfSunrise(55752,37615,6,6,2022);
    if(!utc)
    {
        printf("Восход: Ошибка ...\n");
    }
    else
    {
        utc+=HOUR_CORRECTION*3600;
        //utc - количество секунд от начала суток
        UINT sec=utc;
        printf("sec=%u\n",sec);
        UINT min=sec/60;
        sec%=60;
        UINT hour=min/60;
        min%=60;
        printf("Восход: %02d:%02d:%02d\n",hour,min,sec);
    }
    utc=TimeOfSunset(55752,37615,6,6,2022);
    if(!utc)
    {
        printf("Закат:  Ошибка ...\n");
    }
    else
    {
        utc+=HOUR_CORRECTION*3600;
        //Также для вывода можно использовать готовые функции ..OfTime
        printf("Закат:
%02d:%02d:%02d\n",HourOfTime(utc),MinOfTime(utc),SecOfTime(utc));
    }
    ShowScr();                  //Вывести на дисплей
```

```
    while(1);                    //Бесконечная пауза
}
```

Результат:



См. также
TimeOfSunset *SecOfTime* *MinOfTime* *HourOfTime* *printf*

3.4.15. TIMEOFSUNSET

```
#include <sat500.h>

unsigned int TimeOfSunset(unsigned int lat,unsigned int lon,unsigned char
day,unsigned char month,unsigned short int year);
```

Получение значения времени захода солнца в определённой географической точке в указанный день.

Аргументы функции:
`unsigned int lat` - Широта географической точки в тысячных долях градуса
`unsigned int lon` - Долгота географической точки в тысячных долях градуса
`unsigned char day` - День месяца (1-31)
`unsigned char month` - Номер месяца(1-12)
`unsigned short int year` - Год (2022 ...)

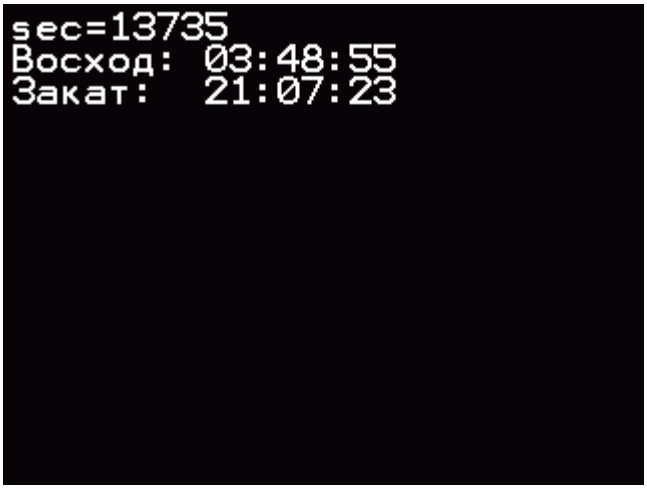
Возвращаемое значение: количество секунд от начала суток.
Если возвращается нулевое значение, то вычислить значение не удалось.
Примечание: Функция возвращает время по Гринвичу, поэтому необходимо скорректировать полученное значение на величину расхождения времени с географической точки для которой выполнен расчёт. В простейшем случае это может быть часовой пояс географической точки.

Пример:

```
//Определение времени заката и рассвета г.Москвы с координатами:
//Широта:  55,752
//Долгота: 37,615

#include <sat500.h>

typedef unsigned int  UINT;
```

```
//Коррекция часов для Москвы
#define HOUR_CORRECTION 3

void main(void)
{
    UINT utc;
    ClrScr();                //Очистить экран
    BigFont=0;               //Обычный шрифт 12x16 (не крупный)
    //6 июня 2022 года
    utc=TimeOfSunrise(55752,37615,6,6,2022);
    if(!utc)
    {
        printf("Восход: Ошибка ...\n");
    }
    else
    {
        utc+=HOUR_CORRECTION*3600;
        //utc - количество секунд от начала суток
        UINT sec=utc;
        printf("sec=%u\n",sec);
        UINT min=sec/60;
        sec%=60;
        UINT hour=min/60;
        min%=60;
        printf("Восход: %02d:%02d:%02d\n",hour,min,sec);
    }
    utc=TimeOfSunset(55752,37615,6,6,2022);
    if(!utc)
    {
        printf("Закат:  Ошибка ...\n");
    }
    else
    {
        utc+=HOUR_CORRECTION*3600;
        //Также для вывода можно использовать готовые функции ..OfTime
        printf("Закат:
%02d:%02d:%02d\n",HourOfTime(utc),MinOfTime(utc),SecOfTime(utc));
    }
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



См. также
TimeOfSunrise *SecOfTime* *MinOfTime* *HourOfTime* *printf*

3.4.16. ПРОИЗВОДСТВЕННЫЙ КАЛЕНДАРЬ

Производственный календарь - это специальный календарь, составленный с учётом рабочих дней, нерабочих праздничных дней и выходных дней на текущий год.
Производственный календарь составляется на основе постановлений правительства Российской Федерации с учётом требований трудового законодательства Российской Федерации.
В производственном календаре содержится достоверная и точная информация о количестве рабочих, выходных и праздничных дней в текущем году.
В данном разделе приведён пример реализации производственного календаря.

Несколько производственных календарей размещаются в EPROM контроллера "Saturn PLC".
Редактирование производственного календаря в EPROM контроллера выполняется с помощью специальной бесплатной программы для Windows, которую можно загрузить по следующей ссылке:
https:/www.mnppsaturn.ru/public/soft/plccalendar/plccalendarsetup.exe

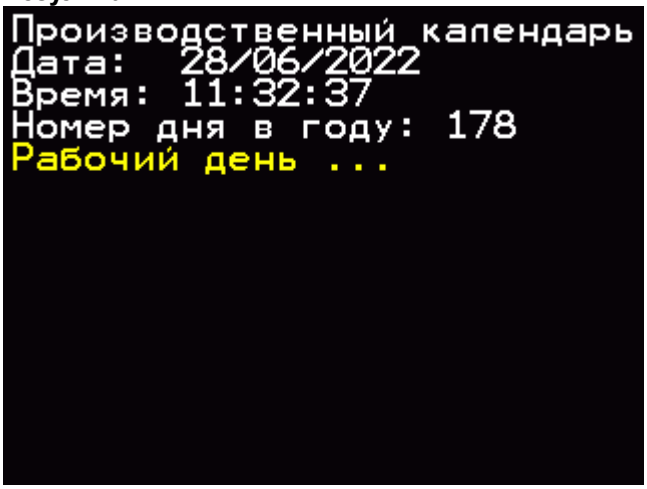
Пример:
```
#include <sat500.h>
#include "types.h"

//-----------------------------------------------------------------------------
-
/*
    В EPROM содержится несколько производственных календарей
    (записано программой PlcCalendar):
    EPROM[0]...EPROM[3] - 8 байтов - пароль (не используем)
    EPROM[4] - номер года 2022...2099, любое другое значение - конец списка
годов
    EPROM[5]...EPROM[27] - Побитово 366 дней года, 0-рабочий день, 1 - выход-
ной
    EPROM[28] - номер года 2022...2099, любое другое значение - конец списка
годов
    EPROM[29]...EPROM[51] - Побитово 366 дней года, 0-рабочий день, 1 - выход-
ной
    и т.д. до 20-ти годов
    Каждый год занимает 48 байтов (24 слова EPROM)
*/
//-----------------------------------------------------------------------------
```

```
-
//Вернуть выходной/рабочий день по данным в EPROM
char GetIsHolidayEprom(WORD year,UINT unix);
//Вернуть выходной/рабочий день по умолчанию (нет данных в EPROM)
char GetIsHolidayDefault(BYTE date,BYTE month,UINT unix);
//-----------------------------------------------------------------------------
-
//Этот пример выводит на дисплей информацию о отм является текущий день
//выходным или рабочим днём
void main(void)
{
Start:;
    BigFont=false;
    BrushColor=COLOR_BLACK;         //Цвет кисти - чёрный
    FontColor=COLOR_WHITE;          //Цвет шрифта - белый
    ClrScr();                       //Очистить экран чёрным цветом
    puts("Производственный календарь\n");
    //Получить текущую дату
    BYTE date,month;
    WORD year;
    GetDate(&date,&month,&year);
    //Вывод на дисплей даты
    printf("Дата:  %02d/%02d/%04d\n",date,month,year);
    //Время для вывода на дисплей
    BYTE hour,min,sec;
    GetTime(&hour,&min,&sec);
    //Вывод на дисплей времени
    printf("Время: %02d:%02d:%02d\n",hour,min,sec);
    //Время в формате UNIX: сегодня 12:00
    UINT unix=MkTime(12,0,0,date,month,year);
    //Вернуть выходной/рабочий день по данным в EPROM
    char holiday;
    holiday=GetIsHolidayEprom(year,unix);
    if(holiday<0)
    {
        //Нет текущего года в EPROM
        //Цвет шрифта - красный
        FontColor=COLOR_RED;
        //Вывод на дисплей сообщения
        printf("Нет календаря для %04d г\n",year);
        //Вернуть выходной/рабочий день по умолчанию (нет данных в EPROM)
        holiday=GetIsHolidayDefault(date,month,unix);
    }
    if(holiday)
    {
        FontColor=COLOR_GREEN;          //Цвет шрифта - зелёный
        puts("Выходной день...");
    }
    else
    {
        FontColor=COLOR_YELLOW;          //Цвет шрифта - желтый
        puts("Рабочий день ...");
    }
    ShowScr();
    //Пауза 1000 мс
    UINT t=GetTickCount();
    while((GetTickCount()-t)<1000)
    {
```

```
    }
    goto Start;
}
//-----------------------------------------------------------------------------
-
//Вернуть выходной/рабочий день по данным в EPROM
//Возврат:
//0-Рабочий день
//1-Выходной
//-1 - Не найден год в EPROM
char GetIsHolidayEprom(WORD year,UINT unix)
{
    WORD pos=4;
    WORD eyear;
    for(int i=0;i<20;i++)
    {
        eyear=ReadEprom(pos++);
        if(eyear<2022 || eyear>2099)break;
        if(eyear==year)
        {
            //Найден нужный год
            //Получение значения номера дня года из Unix-времени
            UINT daynum=YearDayOfTime(unix);
            printf("Номер дня в году: %03d\n",daynum);
            //Номер нужного слова
            WORD wordnum=daynum/16;
            //Номер нужного бита
            WORD bitnum=daynum%16;
            //Получить нужный бит
            WORD eprom=ReadEprom(pos+wordnum);
            if(eprom & (1<<bitnum))
            {
                //Выходной день
                return 1;
            }
            //Рабочий день
            return 0;
        }
        //Перейти к следующему году в EPROM
        pos+=23;
    }
    return -1;
}
//-----------------------------------------------------------------------------
-
const char * _WeekDays[]=
{
    "воскресенье","понедельник","вторник","среда","четверг",
    "пятница","суббота"
};
//-----------------------------------------------------------------------------
-
//Вернуть выходной/рабочий день по умолчанию (нет данных в EPROM)
//Возврат:
//0-Рабочий день
//1-Выходной
char GetIsHolidayDefault(BYTE date,BYTE month,UINT unix)
{
```

```
    //Получить день недели
    BYTE wday=WeekDayOfTime(unix);
    //Цвет шрифта - белый
    FontColor=COLOR_WHITE;
    printf("День недели: %s\n",_WeekDays[wday]);
    //Если воскресенье или суббота то это выходной день
    if(!wday || wday==6)return 1;
    //Проверка на государственные праздничные дни
    //1 Января Новый год
    //2-6 Января Новогодние каникулы
    //7 Января    Рождество Христово
    //8 Января Новогодние каникулы
    if(month==1 && date<9)return 1;
    //23 Февраля День защитника Отечества
    if(month==2 && date==23)return 1;
    //8 Марта Международный женский день
    if(month==3 && date==8)return 1;
    //1 Мая Праздник весны и труда
    if(month==5 && date==1)return 1;
    //9 Мая День Победы
    if(month==5 && date==9)return 1;
    //12 Июня День России
    if(month==6 && date==12)return 1;
    //4 Ноября День народного единства
    if(month==11 && date==4)return 1;
    return 0;
}
//-----------------------------------------------------------------------------
-
```

Результат:



Пример 2
В каталоге "C:\ProgramData\SatScript\Examples\Производственный календарь" находится пример в котором значение выходной/рабочий день выдаётся через сетевую переменную.

3.4.16.1. ПРОГРАММА "PLCCALENDAR"

Загрузка программы

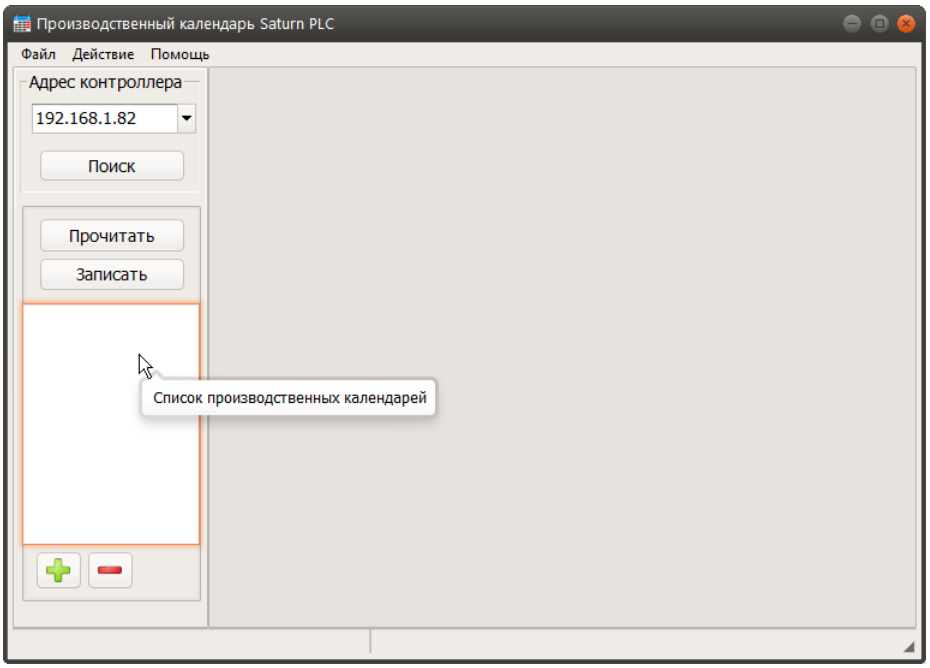
Программу "PlcCalendar" можно загрузить по следующей ссылке:

https:/www.mnppsaturn.ru/public/soft/plccalendar/plccalendarsetup.exe

Загрузите и установите программу.

Основное окно

Запустите программу. Ниже показано основное окно программы:

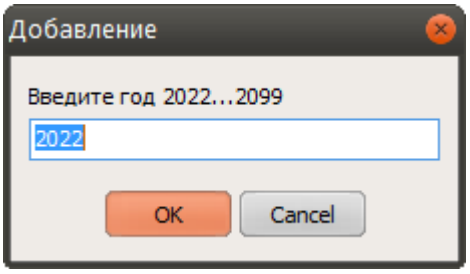


В левой части расположен список производственных календарей.
Основные управляющие элементы окна:

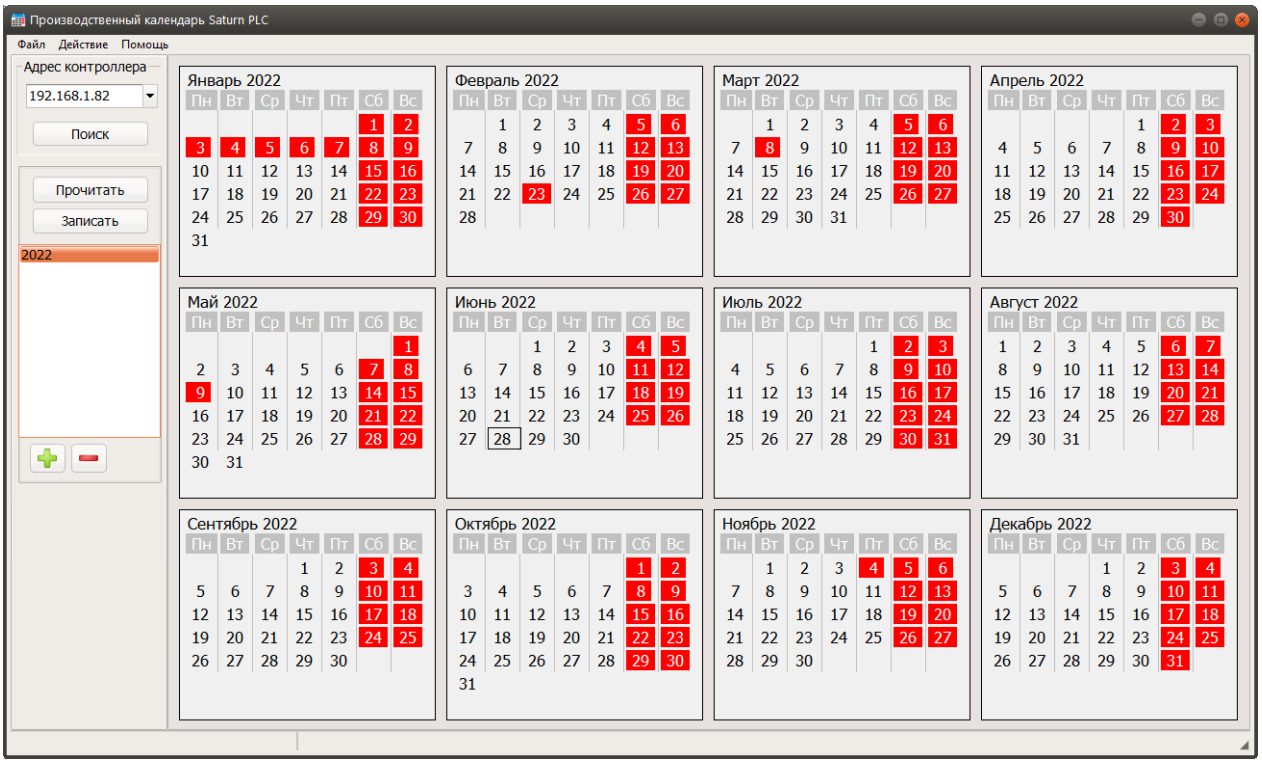
Поле "Адрес контроллера"	IP адрес контроллера "Saturn PLC"
Кнопка "Поиск"	Поиск контроллеров "Saturn PLC"
Кнопка "Прочитать"	Чтение списка производственных календарей из контроллера "Saturn PLC"
Кнопка "Записать"	Запись списка производственных календарей в контроллер "Saturn PLC"
Список календарей	Список производственных календарей для записи в контроллер "Saturn PLC"
Кнопка "Добавить год"	Добавление одного производственного календаря в список
Кнопка "Удалить год"	Удаление текущего производственного календаря из списка

Добавление календаря

Для добавления производственного календаря в список следует нажать кнопку "Добавить год" (зелёный плюс) или воспользоваться пунктом "Добавить" контекстного меню..
Далее нужно ввести номер добавляемого года:



И нажать кнопку "ОК".
В списке появится новый производственный календарь на заданный год.
В правой части окна будет показан весь производственный календарь по месяцам.
Красным цветом показаны выходные и праздничные дни:



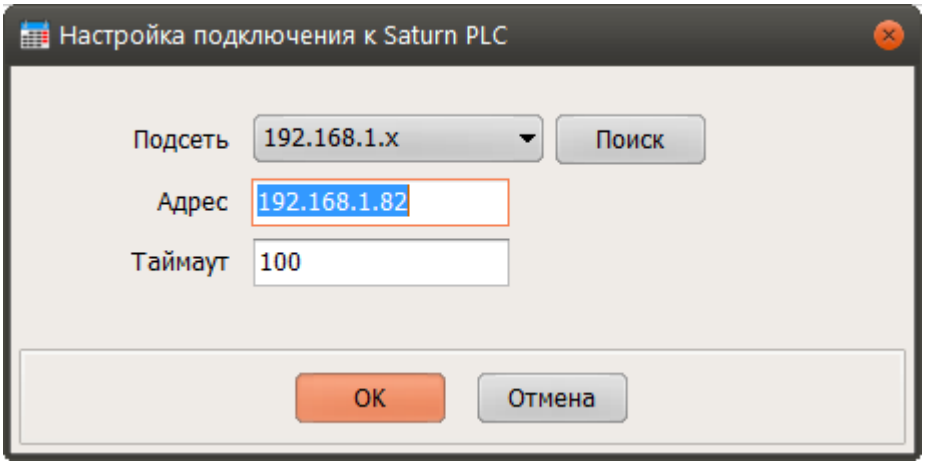
По умолчанию на календаре отмечены основные праздники РФ и выходные дни.

Изменение дней календаря

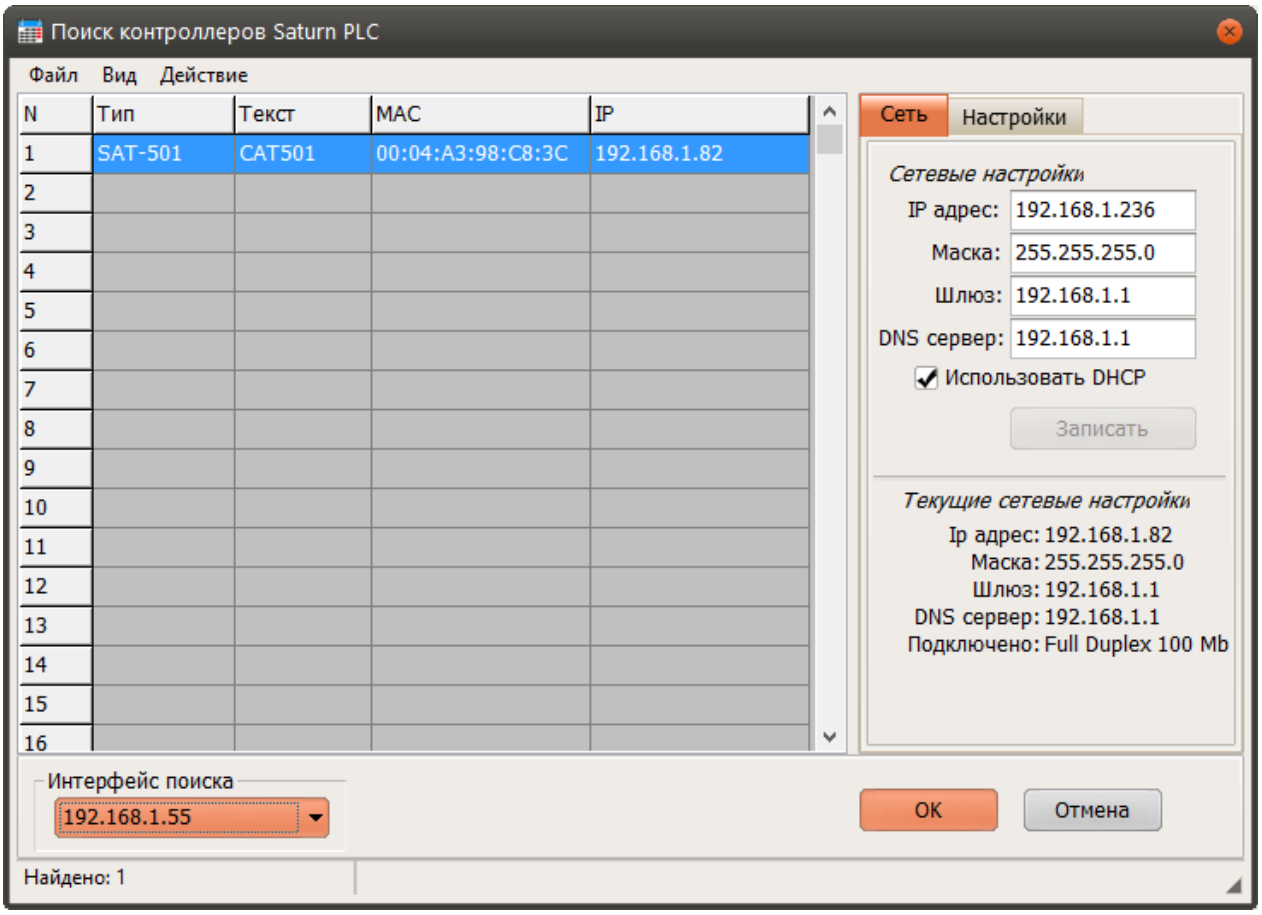
Для изменения следует щёлкнуть левой кнопкой мышки по любому дню - его статус изменится на противоположный. Выходной день станет рабочим и наоборот.
В список можно добавить до 20 производственных календарей для разных годов. Однако, следует помнить что иногда вносятся переносы выходных и праздничных дней, поэтому следует каждый год обновлять производственные календари в контроллере в соответствии с публикуемым официальным производственным календарём на текущий год.

Запись списка календарей в контроллер

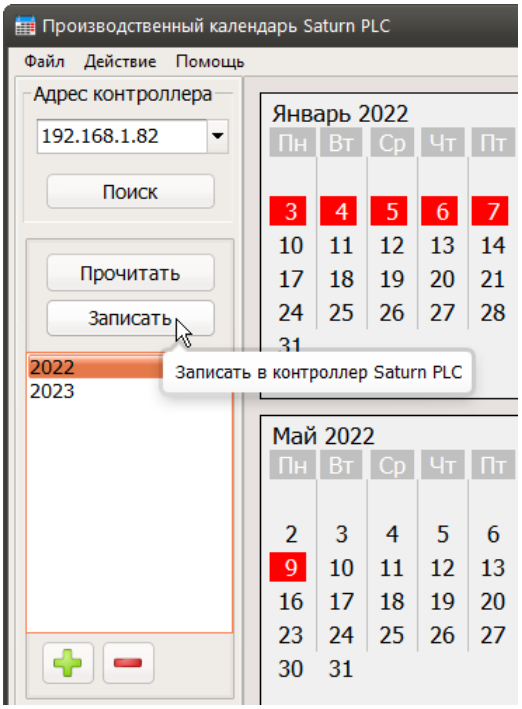
Перед записью списка календарей в контроллер следует задать правильный IP адрес контроллера "Saturn PLC" в поле ввода "Адрес контроллера".
Текущий IP адрес можно посмотреть на самом контроллере нажав кнопку "Вверх".
Если контроллер находится в той же локальной сети что и компьютер, то можно выполнить поиск контроллера.
Для этого следует нажать кнопку "Поиск".



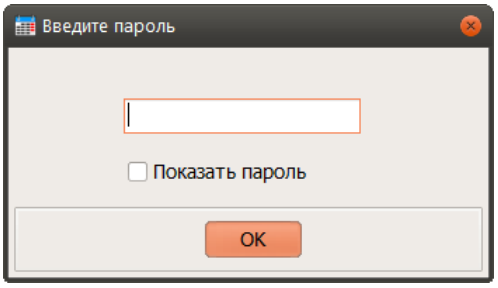
В открывшемся окне следует выбрать правильную подсеть и нажать кнопку "Поиск".
Откроется окно с найденными контроллерами:



Нужно выделить найденный контроллер и нажать кнопку "OK".
Далее ещё раз нажать кнопку "OK".
IP адрес найденного контроллера будет показан в поле "Адрес контроллера".
Для записи списка календарей в контроллер следует нажать кнопку "Записать":



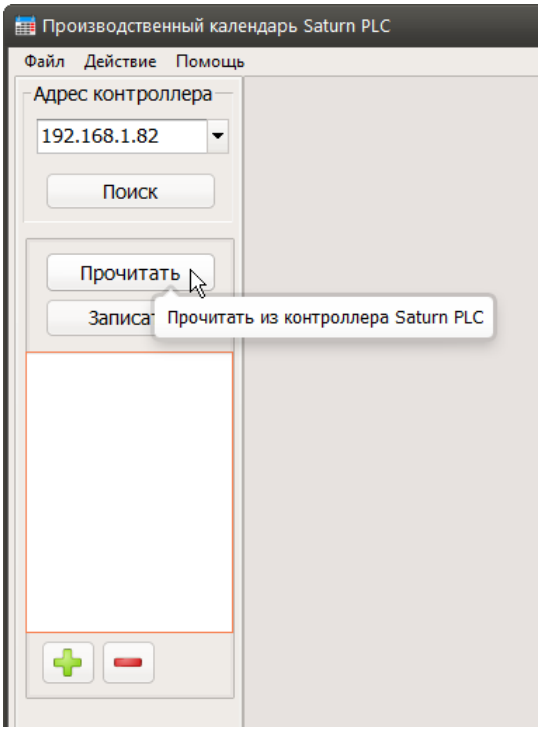
Появится окно ввода пароля:



Введите пароль, который потребуется если кто-то захочет изменить производственные календари в контроллере.
Также можно оставить поле пароля пустым и далее пароль не потребуется.
После нажатия на кнопку "OK" будет выполнена запись календаря в контроллер.
Результат отображается в статусной строке окна:

Чтение списка календарей из контроллера

Также можно прочитать список производственных календарей из контроллера. Для этого следует нажать на кнопку "Прочитать":



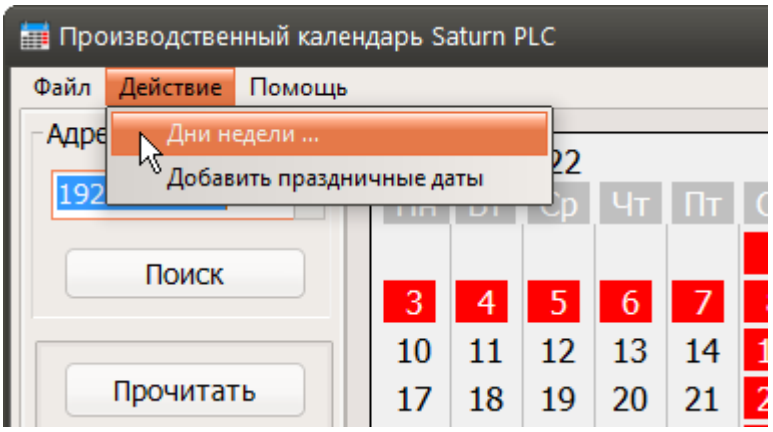
Будет выполнено чтение из контроллера.
Если ранее был установлен пароль, то появится окно с предложением ввести пароль.
Затем в списке производственных календарей будут показаны все года, записанные ранее в контроллер.
Можно удалить ненужные года, отредактировать текущий год и записать обратно в контроллер.

Удаление календаря из списка

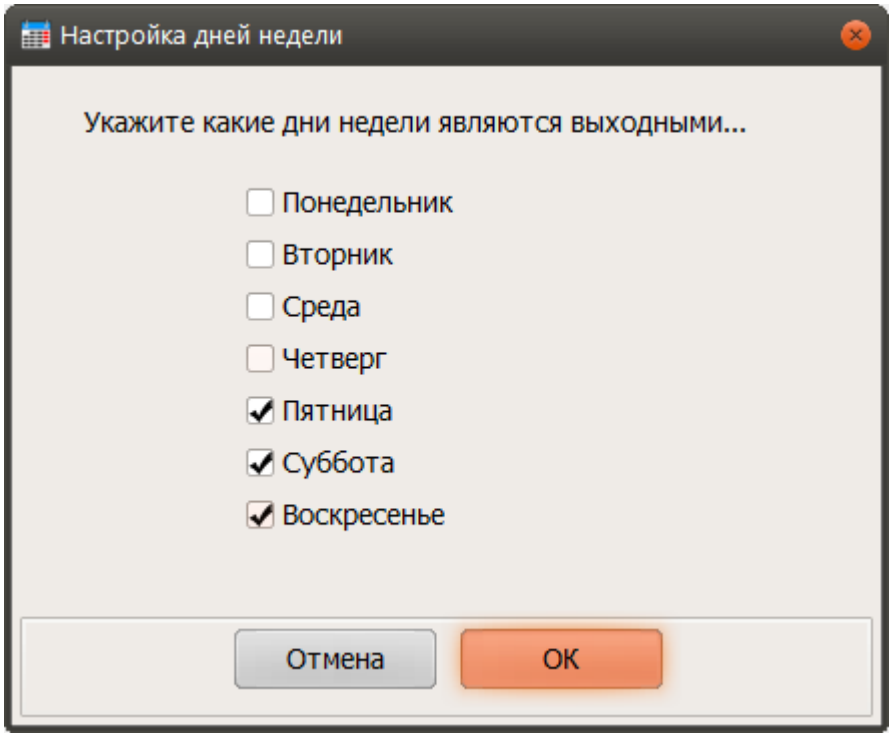
Для удаления календаря из списка следует выделить нужный календарь в списке и нажать кнопку "Удалить год" (красный минус) или воспользоваться пунктом "Удалить" контекстного меню.

Создание производственного календаря для нестандартной недели

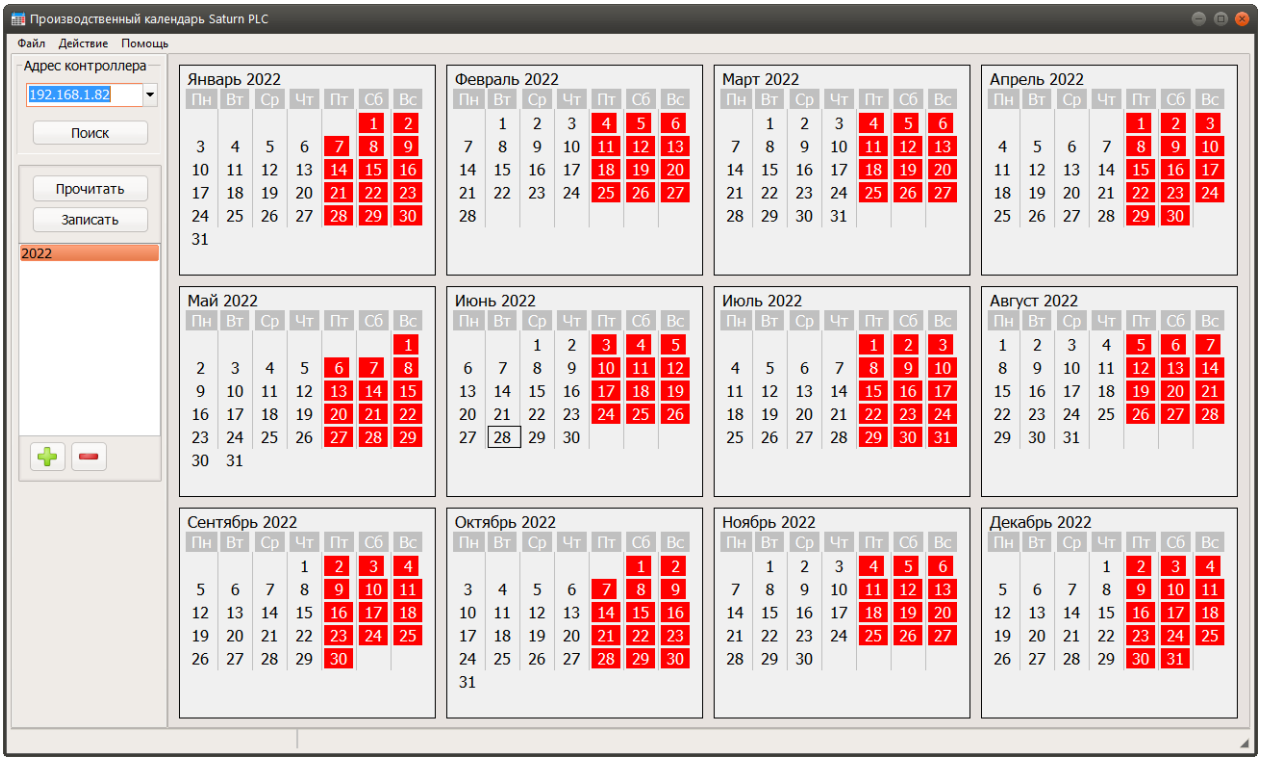
В случае когда нужно создать календарь с нестандартными рабочими днями недели следует:
1. Создать календарь на нужный год и выделить его в списке
2. В основном меню программы выбрать пункт "Действие/Дни недели"



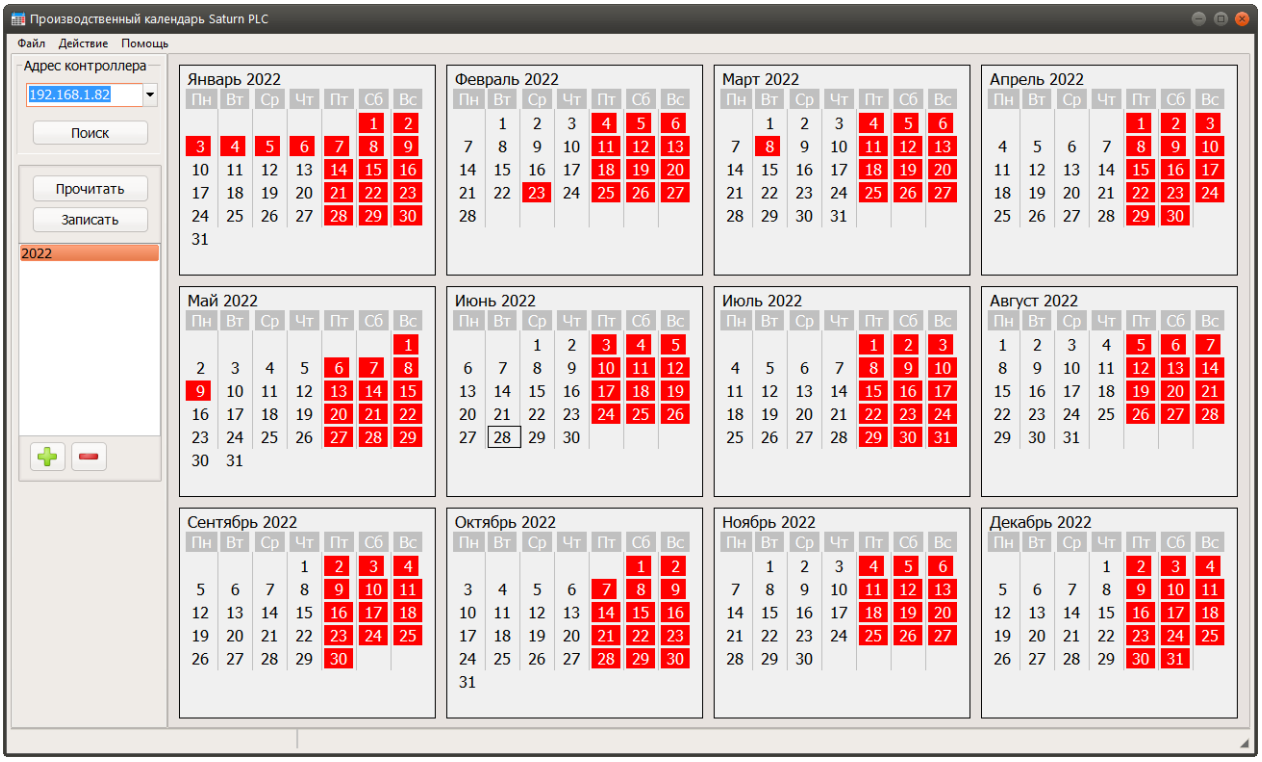
3. В открывшемся окне отметить какие дни недели являются выходными и нажать кнопку "ОК":



4. Текущий календарь будет изменён в соответствии с новыми выходными днями:



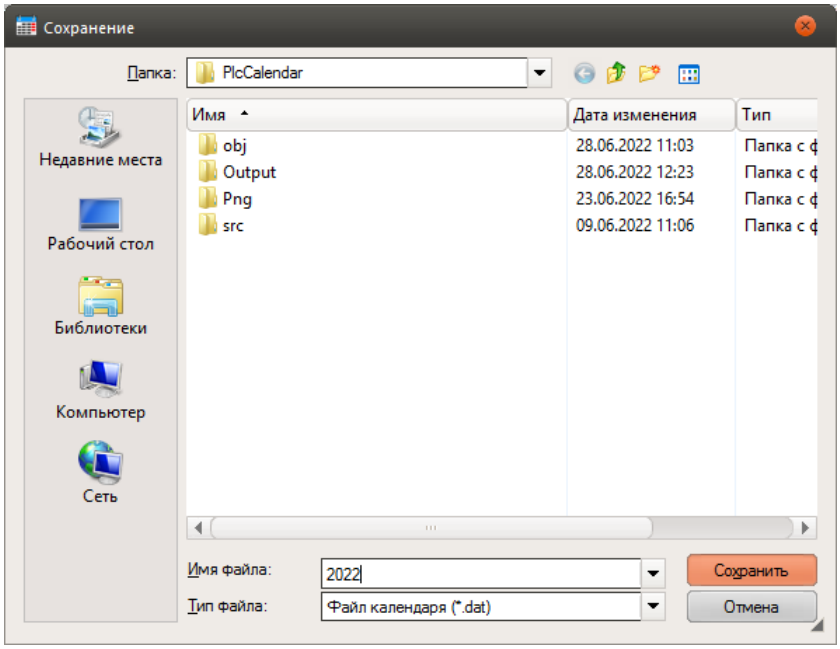
5. В основном меню программы выбрать пункт "Действие/Добавить праздничные даты". В календаре будут отмечены стандартные праздники РФ:



6. Отредактировать отдельные дни если требуется.
7. Записать новый календарь в контроллер.

Сохранение списка календарей в файл

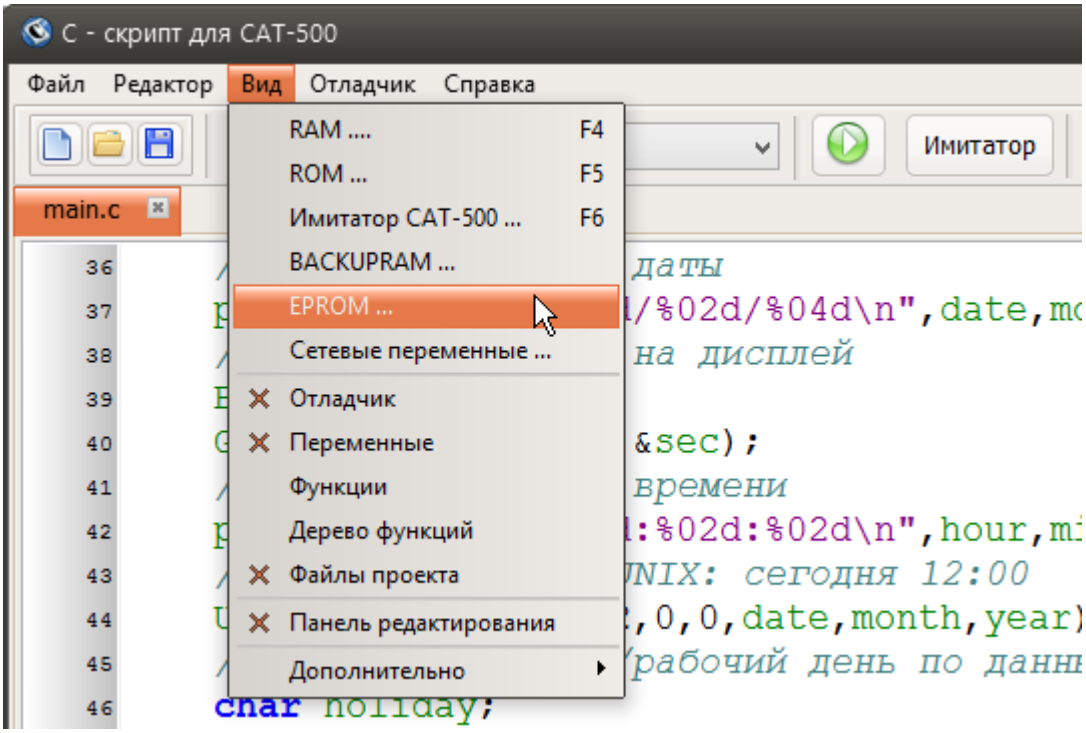
Для сохранения списка календарей в файл следует выбрать пункт основного меню "Файл/Сохранить...".
Далее следует задать имя и расположение сохраняемого файла и нажать кнопку "Сохранить".



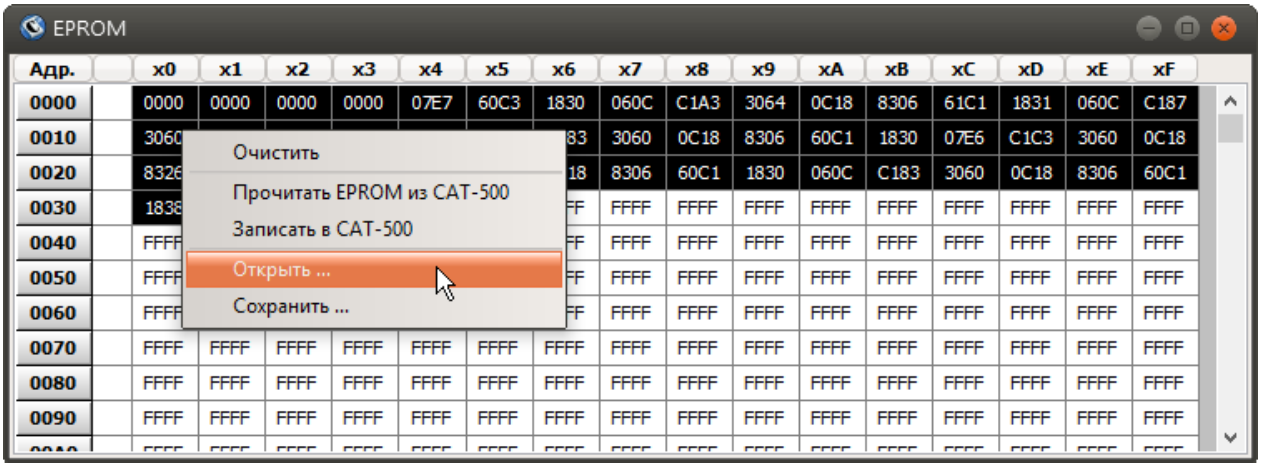
Будет создан файл с данными, предназначенными для записи в EPROM контроллера.

Загрузка списка календарей в "Имитатор" среды C-Script

Если выполняется разработка программы использующей производственный календарь в имитаторе, то можно загрузить сохранённый файл списка календарей в EPROM имитатора.
Для этого следует:
1. Выбрать пункт основного меню "Вид/EPROM ..." среды разработки:



2. В открывшемся окне "EPROM" следует вызвать контекстное меню и выбрать пункт меню "Открыть...".



Далее следует указать на файл сохранённый в программе "PlcCalendar". Этот файл будет загружен в EPROM имитатора и можно будет использовать загруженные данные через функции работы с EPROM имитатора.

3.5. ТАЙМЕРЫ

Контроллер CAT-500 может периодически вызывать функции, не имеющие параметров, и не имеющих возвращаемого результата, т.е. имеющие прототип *void func(void)*.
Всего в программе может быть определено не более 8 таких функций. Каждая такая функция должна быть описана в программе, и затем зарегистрирована как таймер при помощи функции *StartTimer*, описанной в заголовочном файле *<sat500.h>*.
После регистрации функция будет автоматически вызываться с периодом, указанным в момент вызова *StartTimer*.
Если требуется изменить период срабатывания таймера, то следует повторно вызвать функцию *StartTimer* для функции-обработчика, указав новый период.
Для прекращения работы таймера следует вызвать функцию *StopTimer*, указав функцию для которой следует прервать периодические вызовы.

Функция	***Описание***
*char StartTimer (void * func,unsigned int period)*	Установка функции - таймера
*char StopTimer (void * func)*	Останов функции - таймера

3.5.1. STARTTIMER

```
#include <sat500.h>

char StartTimer (void * func,unsigned int period);
```

Установка функции - таймера.
*void * func* - адрес функции с прототипом *void interrupt func(void).*
unsigned int period - период срабатывания таймера в миллисекундах
Для организации периодического вызова некоторой функции (таймера) следует:

1. Определить в программе функцию с прототипом *void interrupt func(void).* Функция - обработчик события таймера обязательно должно включать модификатор *interrupt.*
2. Зарегистрировать таймер при помощи функци *StartTimer*, указав функцию-обработчик и период *period* вызова в миллисекундах

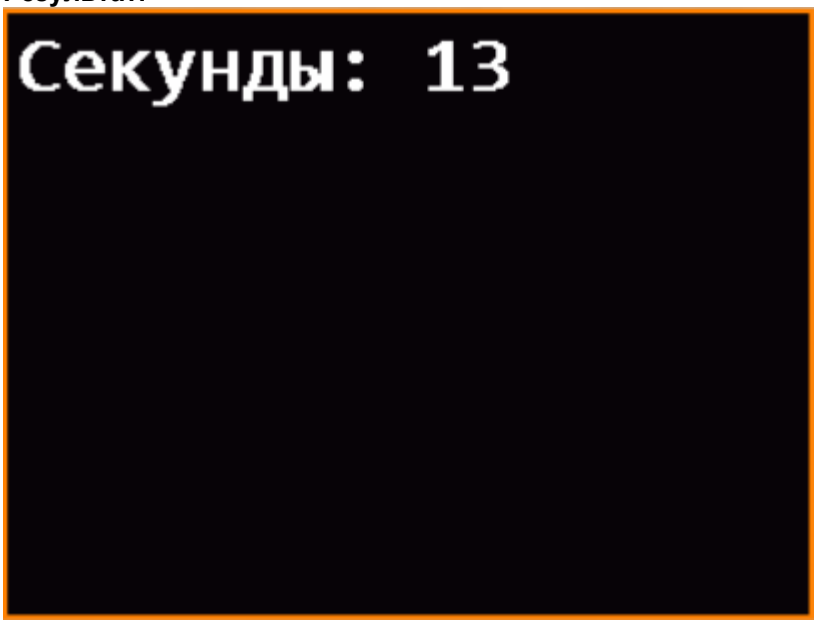
После этого функция будет автоматически вызываться с указанным периодом.
Всего можно задать до 8-ми таких функций-таймеров, каждый со своим периодом.
Повторный вызов функции *StartTimer* с указанием уже зарегистрированного обработчика позволяет задать новый период.
Если задаётся нулевое значение периода, то таймер будет удалён. Также для удаления таймера можно использовать функцию *StopTimer*.
Функция *StartTimer* возвращает количество работающих таймеров (число от 0 до 8).
0 соответствует отсутствию работающих таймеров, а 8 сообщает что все таймеры заняты.

Пример:

```
//Эта программа устанавливает таймер, который срабатывает каждую секунду
#include <sat500.h>

unsigned Counter;
//Функция - обработчик таймера
void interrupt OnTimer1(void);

void main(void)
{
    BrushColor=COLOR_BLACK;          //Цвет кисти - чёрный
    ClrScr();                        //Очистить экран чёрным цветом
```

```
    FontColor=COLOR_WHITE;          //Цвет шрифта - белый
    StartTimer(&OnTimer1,1000);     //Вызывать функцию OnTimer1 один раз в се-
кунду
    while(1)
    {
        GotoXY(4,4);                        //Курсор в позицию x=4,  y=4
        printf("Секунды: %-8d",Counter);    //Вывод на дисплей счётчика секунд
        ShowScr();                          //Вывести на дисплей
    }
}

//Функция - обработчик таймера
void interrupt OnTimer1(void)
{
    Counter++;                      //Увеличить счётчик
}
```

Результат:

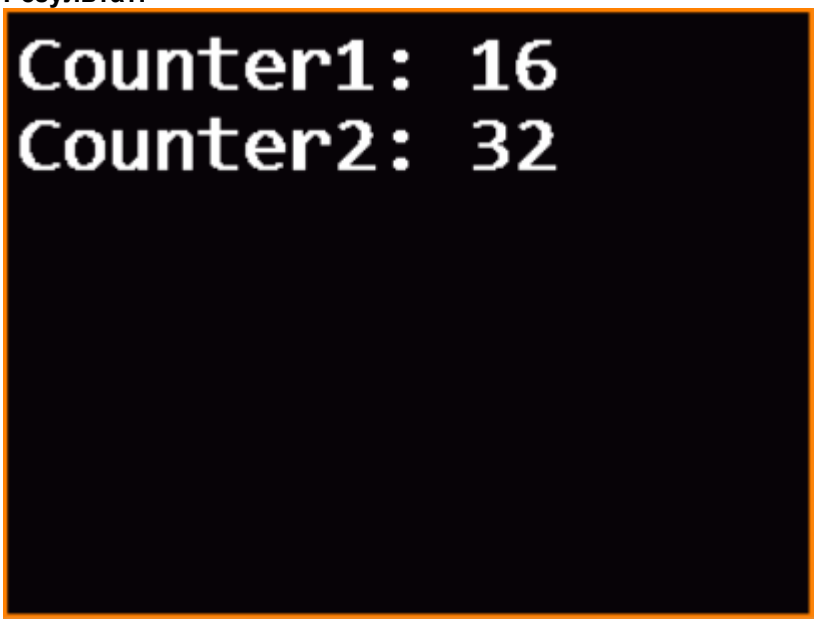


Пример с 2-мя таймерами:

```
#include <sat500.h>

unsigned Counter1;
unsigned Counter2;

//Функция - обработчик таймера 1
void OnTimer1(void);
//Функция - обработчик таймера 2
void OnTimer2(void);

void main(void)
{
    BrushColor=COLOR_BLACK;         //Цвет кисти - чёрный
    ClrScr();                       //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;          //Цвет шрифта - белый
```

```
    StartTimer(&OnTimer1,1000);     //Вызывать функцию OnTimer1 один раз в се-
кунду
    StartTimer(&OnTimer2,500);      //Вызывать функцию OnTimer2 два раза в се-
кунду
    while(1)
    {
        GotoXY(4,4);                            //Курсор в позицию x=4,  y=4
        printf("Counter1: %-8d",Counter1);      //Вывод на дисплей счётчика се-
кунд Counter1
        GotoXY(4,36);                           //Курсор в позицию x=4,  y=36
        printf("Counter2: %-8d",Counter2);      //Вывод на дисплей счётчика
Counter2
        ShowScr();                              //Вывести нарисованное не дис-
плей
    }
}

//Функция - обработчик таймера 1
void OnTimer1(void)
{
    Counter1++;                 //Увеличить счётчик 1
}

//Функция - обработчик таймера 2
void OnTimer2(void)
{
    Counter2++;                //Увеличить счётчик 2
}
```

Результат:



См. также
StopTimer *BrushColor* *ClrScr* *FontColor* *GotoXY* *printf* *ShowScr*
Константы цветов *Цвет*

3.5.2. STOPTIMER

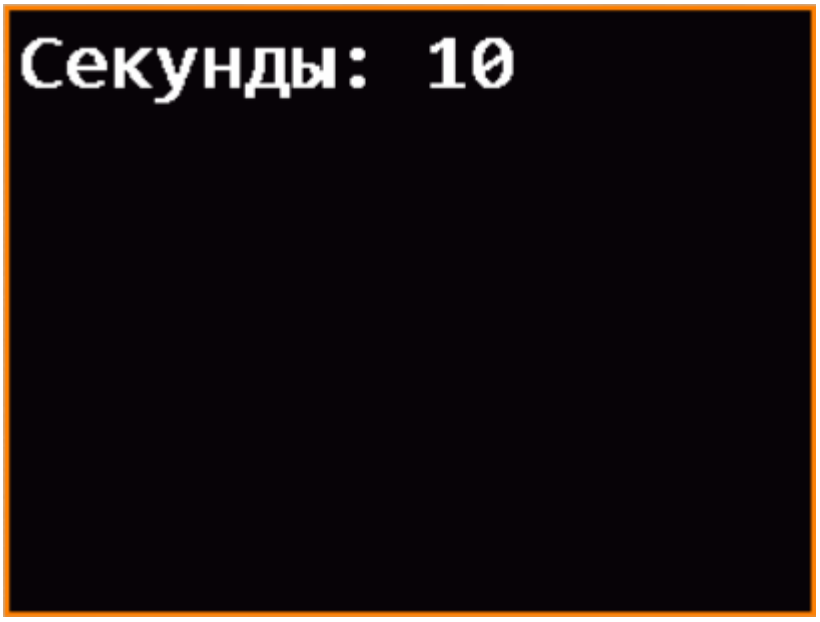
```
#include <sat500.h>

char StopTimer (void * func);
```

Останов функции - таймера.
Функция *StopTimer* используется для удаления работающего таймера, или определения количества работающих таймеров.
Для удаления таймера следует вызвать *StopTimer* с параметром - адрес функции обработчика.
Для определения количества работающих таймеров следует вызвать функцию *StopTimer* с нулевым параметром.
Функция возвращает количество работающих таймеров (число от 0 до 8).
0 соответствует отсутствию работающих таймеров, а 8 сообщает что все таймеры заняты

Пример

```
//Эта программа устанавливает таймер и через 10 секунд удаляет его
#include <sat500.h>

unsigned Counter;
//Функция - обработчик таймера
void OnTimer1(void);

void main(void)
{
    BrushColor=COLOR_BLACK;                  //Цвет кисти - чёрный
    ClrScr();                                //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;                   //Цвет шрифта - белый
    StartTimer(&OnTimer1,1000);              //Вызывать функцию OnTimer1 один
раз в секунду
    while(1)
    {
        GotoXY(4,4);                        //Курсор в позицию x=4,  y=4
        printf("Секунды: %-8d",Counter);    //Вывод на дисплей счётчика секунд
        ShowScr();                          //Вывести на дисплей
    }
}

//Функция - обработчик таймера
void OnTimer1(void)
{
    if(++Counter == 10)       //Увеличить счётчик
    {
        StopTimer(&OnTimer1);       //Остановить таймер
    }
}
```

Результат:



См. также
StartTimer *BrushColor* *ClrScr* *FontColor* *GotoXY* *printf* *ShowScr*
Константы цветов *Цвет*

3.6. КЛАВИАТУРА

Клавиатура в контроллере CAT-500 используется для вывода системного меню и экранов просмотра.
Поэтому в пользовательской программе для использования клавиатуры необходимо запретить работы клавиатуры в системе.
Только после этого программа сможет получать коды нажатых клавиш клавиатуры, но штатная работы системного меню и окон просмотра будет запрещена.
Контроллер CAT-500 не имеет клавиатурного буфера и единственным способом получения нажатия управляющий кнопки - это проверка текущего состояния нажатия кнопки.

Для работы с клавиатурой в заголовочном файле <sat500.h> определена одна переменная *EnableKeyboard* и две функции:

Переменная	***Описание***
char EnableKeyboard	Включение работы с клавиатурой

Функция	***Описание***
char GetKey (void)	Получение кода текущей нажатой кнопки клавиатуры
void PressKey(char key)	Программное нажатие кнопки клавиатуры

3.6.1. ENABLEKEYBOARD

```
#include <sat500.h>
```

```
char EnableKeyboard
```

Переменная, значение которой разрешает или запрещает работу с клавиатурой в программе.

Значение	***Описание***
EnableKeyboard=0	Программа не может работать с клавиатурой, функция GetKey возвращает значение 0. Работа системного меню (по кнопке ВПРАВО) разрешена, экраны просмотра разрешены
EnableKeyboard=1	Программа работает с клавиатурой, функция GetKey возвращает код или коды нажатых кнопок. Работа системного меню (по кнопке ВПРАВО) запрещена, экраны просмотра запрещены

Переменная доступна для чтения и для записи.

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLUE;      //Цвет кисти - синий
    ClrScr();                   //Очистить экран кистью - синим цветом
    ShowScr();                  //Вывести нарисованное не дисплей
    EnableKeyboard=1;           //Разрешить программе работу с клавиатурой
    while(1)                    //Бесконечный цикл
    {
        GotoXY(4,4);                    //Установка позиции вывода на экран
        printf("Кнопка: %d",GetKey()); //Вывод кода нажатой кнопки
        ShowScr();                      //Вывести на дисплей
    }
}
```

Результат:

См. также
GetKey *BrushColor* *ClrScr* *GotoXY* *printf* *ShowScr*
Константы цветов *Цвет*

3.6.2. GETKEY

```
#include <sat500.h>

char GetKey(void);
```

Получение кода текущей нажатой кнопки клавиатуры.

Функция возвращает следующие значения:

Значение	***Описание***
0	Нет нажатой кнопки
1	Нажата кнопка ВВЕРХ
2	Нажата кнопка ВПРАВО
4	Нажата кнопка ВНИЗ
8	Нажата кнопка ВЛЕВО

Для удобства в файле *<sat500.h> определены следующие строковые константы:*

```
#define KEY_UP           1
#define KEY_RIGHT        2
#define KEY_DOWN         4
#define KEY_LEFT         8
```

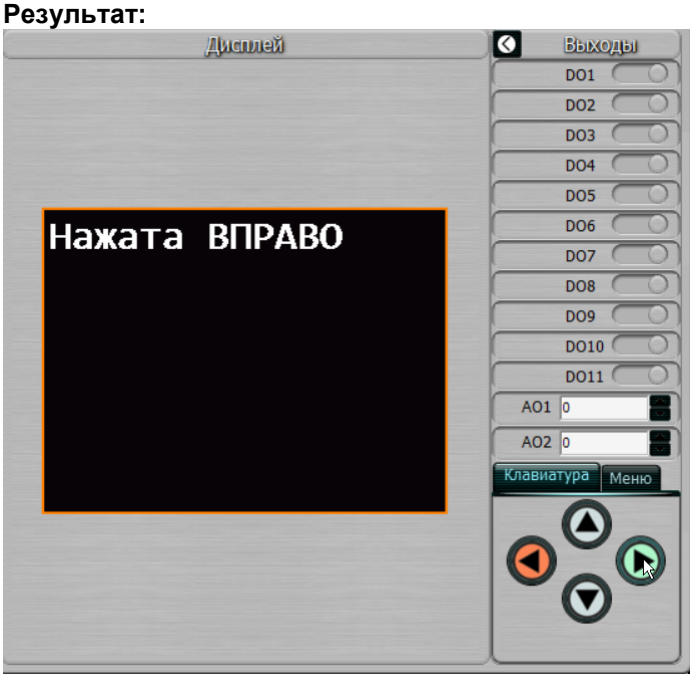
Если пользователь нажимает 2 кнопки, то возвращается сумма нажатых кнопок.

Пример:

```
#include <sat500.h>
void main(void)
{
    unsigned int t;
    BrushColor=COLOR_BLACK;     //Цвет кисти - чёрный
    ClrScr();                   //Очистить экран чёрным цветом
    EnableKeyboard=1;           //Разрешить программе работу с клавиатурой
    while(1)
    {
        GotoXY(4,4);            //Курсор в позицию x=4, y=4
        char key=GetKey();      //Код нажатой клавиши
        switch(key)
        {
            case KEY_LEFT:
            {
                puts("Нажата ВЛЕВО");
                break;
            }
            case KEY_RIGHT:
            {
                puts("Нажата ВПРАВО");
                break;
            }
            case KEY_UP:
            {
                puts("Нажата ВВЕРХ");
                break;
            }
            case KEY_DOWN:
            {
                puts("Нажата ВНИЗ");
                break;
            }
            default:
            {
```

```
                puts("Не нажата ...");
                break;
            }
        }
        ShowScr();              //Вывести на дисплей
        t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
}
```

Результат:



См. также
EnableKeyboard *BrushColor* *ClrScr* *GotoXY* *puts* *ShowScr*
Константы цветов *Цвет*

3.6.3. PRESSKEY

```
#include <sat500.h>

void PressKey(char key);
```

Программное нажатие кнопки клавиатуры.
char key - код нажатой клавиши:

Значение	*Описание*
0	Нет нажатой кнопки
1	Нажата кнопка ВВЕРХ
2	Нажата кнопка ВПРАВО
4	Нажата кнопка ВНИЗ
8	Нажата кнопка ВЛЕВО

В примере ниже программно нажимаются все кнопки клавиатуры с периодом 1 секунда.

Пример:

```
#include <sat500.h>

//Нажимаемая кнопка|
char Var=1;

void main(void)
{
    BrushColor=COLOR_BLACK;                 //Цвет кисти - чёрный
    ClrScr();                               //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;                  //Цвет шрифта - белый
    BigFont=0;                              //Установить обычный шрифт (12x16,
12 пикселей  по горизонтали, 16 пикселей  по вертикали)
    EnableKeyboard=1;
    while(1)
    {
        char key=GetKey();
        GotoXY(4,4);                        //Курсор в позицию x=4,  y=4
        switch(key)
        {
            case KEY_UP:    puts("Нажата ВВЕРХ "); break;
            case KEY_DOWN:  puts("Нажата ВНИЗ  "); break;
            case KEY_LEFT:  puts("Нажата ВЛЕВО "); break;
            case KEY_RIGHT: puts("Нажата ВПРАВО"); break;
        }
        ShowScr();                          //Вывести на дисплей
        //Пауза 1000 мс
        unsigned int t=GetTickCount();
        while(GetTickCount()-t<1000);
        //Следующая кнопка
        Var<<=1;
        if(Var>8)
        {
            Var=1;
        }
        PressKey(Var);
   }
}
```

Результат:

3.7. ЗВУКОВОЙ СИГНАЛ

Для формирования звукового сигнала в заголовочном файле *<sat500.h>* определена одна функция:

Функция	***Описание***
void Beep (unsigned short int t)	Включение/выключение звукового сигнала

3.7.1. BEEP

```
#include <sat500.h>

void Beep(unsigned short int t);
```

Включение звукового сигнала на заданное время в миллисекундах или отключение звукового сигнала при задании нулевого значения аргумента *t*.
unsigned short int t - длительность формируемого звукового сигнала:

Значение t	***Описание***
1...65535	Включить звуковой сигнал на заданное число миллисекунд (1...65535)
0	Выключить звуковой сигнал

В общем случае нет необходимости вызывать функцию *Beep* с параметром 0, т.к. звуковой сигнал автоматически выключается через заданное число миллисекунд.
Функция только запускает звук на заданное время - управление программой сразу же возвращается пользователю. В процесс воспроизведения звука выполнение программы продолжается.

Пример:

```
#include <sat500.h>

void Pause(unsigned int to)
{
    unsigned int t=GetTickCount();
    while((GetTickCount()-t) < to);
}

void main(void)
{
    BrushColor=COLOR_BLUE;      //Цвет кисти - синий
    ClrScr();                   //Очистить экран кистью - синим цветом
    ShowScr();                  //Вывести на дисплей
    //Выдать 10 коротких звуковых сигналов
    for(int i=0;i<10;i++)
    {
        Beep(100);              //Выдать звуковой сигнал 100 мс
        Pause(200);             //Пауза 200 мс
    }
    while(1)                    //Бесконечный цикл
    {
    }
}
```

Пример:

```
//Эта программа устанавливает таймер, который при каждом срабатывании включает
//звуковой сигнал на 250 мс
#include <sat500.h>

//Функция - обработчик таймера
void OnTimer1(void);

unsigned int Count;

void main(void)
{
    BrushColor=COLOR_BLACK;          //Цвет кисти - чёрный
    ClrScr();                        //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;           //Цвет шрифта - белый
    StartTimer(&OnTimer1,500);       //Вызывать функцию OnTimer1 один раз в 500
мс
    while(1);                        //Бесконечная пауза - работа по таймеру
}

//Функция - обработчик таймера
void OnTimer1(void)
{
    Beep(250);                       //Звуковой сигнал 250 мс
    GotoXY(4,4);                     //Курсор в позицию x=4,  y=4
    printf("Beep: %d",Count++);      //Вывод количества звуковых сигналов
    ShowScr();                       //Показать дисплей
}
```

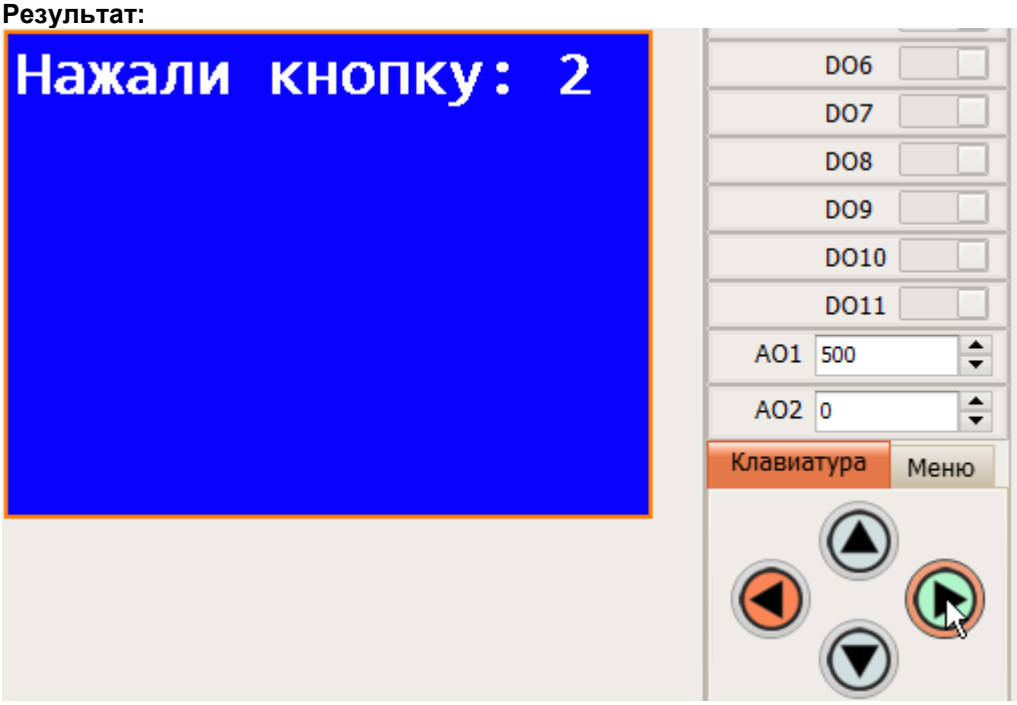
Результат:

Пример:

```
#include <sat500.h>

//Выдать звуковой сигнал на время до 60 секунд
//до нажатия пользователем кнопки на клавиатуре
```

```
void main(void)
{
    char key;
    BrushColor=_COLOR_BLUE;     //Цвет кисти - синий
    ClrScr();                   //Очистить экран кистью - синим цветом
    ShowScr();                  //Вывести нарисованное не дисплей
    //Выдать звуковой сигнал на время до 60 секунд
    Beep(60000);
    EnableKeyboard=1;           //Разрешить программе работу с клавиатурой
    while(1)
    {
        key=GetKey();           //Ввести код нажатой кнопки
        if(key)
        {
            //Кнопка нажата - остановить звук
            Beep(0);
            break;
        }
    }
    EnableKeyboard=0;           //Запретить программе работу с клавиатурой
    printf("Нажали кнопку: %d",key);
    ShowScr();                  //Вывести на дисплей
    while(1)                    //Бесконечный цикл
    {
    }
}
```

Результат:



См. также
StartTimer *BrushColor* *ClrScr* *FontColor*
Константы цветов *Цвет*

3.8. ЭНЕРГОНЕЗАВИСИМАЯ ПАМЯТЬ EPROM

Энергонезависимой память EPROM представляет собой 512 ячеек типа *unsigned short int*.
Для хранения данных в энергонезависимой памяти в заголовочном файле *<sat500.h>* определены две функции:

Функция	***Описание***
unsigned short int ReadEprom (unsigned short int adr)	Чтение данных из энергонезависимой памяти adr=0...511 - адрес в энергонезависимой память
void WriteEprom (unsigned short int adr, unsigned short int data)	Сохранение данных в энергонезависимой памяти adr=0...511 - адрес в энергонезависимой память

ВНИМАНИЕ! Используйте EPROM только для хранения настроечных данных программы.
Частое изменение ячеек EPROM может привести к повреждению FLASH памяти и выходу контроллера из строя.
Общее количество записей в EPROM за время эксплуатации контроллера не должно превышать 100 000 записей.

Имитатор сохраняет данные EPROM в подкаталоге Data с именем компилируемого файла и расширением ".eprom". Например, если компилируемый файл называется "main.c", то программа создаст соответствующий ему файл с данными:
"C:\ProgramData\SatScript\Data\main.eprom"

3.8.1. READEPROM

```
#include <sat500.h>

unsigned short int ReadEprom (unsigned short int adr);
```

Чтение данных из энергонезависимой памяти.
unsigned short int adr - адрес в энергонезависимой памяти. Допустимые значения 0...511
Функция возвращает двойное слово, хранящееся в энергонезависимой памяти по указанному адресу.

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;              //Цвет кисти - чёрный
    ClrScr();                            //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;               //Цвет шрифта - белый
    BigFont=0;                           //Установить обычный шрифт (12x16, 12
пикселей  по горизонтали, 16 пикселей  по вертикали)
    GotoXY(4,4);                         //Курсор в позицию x=4,  y=4
    printf("EPROM[0]=%X",ReadEprom(0)); //Вывод на дисплей содержимого EPROM
по адресу 0
    WriteEprom(0,0x1234);                //Запись в EPROM по адресу 0
    GotoXY(4,24);                        //Курсор в позицию x=4,  y=24
    printf("EPROM[0]=%X",ReadEprom(0)); //Вывод на дисплей содержимого EPROM
по адресу 0
    ShowScr();                           //Вывести на дисплей
    while(1);                            //Бесконечная пауза
```

```
}
```

Результат:

См. также
BrushColor ClrScr FontColor BigFont GotoXY printf ShowScr
WriteEprom

3.8.2. WRITEEPROM

```
#include <sat500.h>

void WriteEprom (unsigned short int adr,unsigned short int data);
```

Сохранение данных в энергонезависимой памяти.
unsigned short int adr - адрес в энергонезависимой памяти. Допустимые значения 0...511
unsigned short int data - данные для записи в память
Функция записывает двойное слово в энергонезависимую память по указанному адресу.

Пример:

```
#include <sat500.h>

void main(void)
{
    BrushColor=COLOR_BLACK;                 //Цвет кисти - чёрный
    ClrScr();                               //Очистить экран чёрным цветом
    FontColor=COLOR_WHITE;                  //Цвет шрифта - белый
    BigFont=0;                              //Установить обычный шрифт (12x16,
12 пикселей  по горизонтали, 16 пикселей  по вертикали)
    GotoXY(4,4);                            //Курсор в позицию x=4,  y=4
    printf("EPROM[0]=%X",ReadEprom(0));     //Вывод на дисплей содержимого
EPROM по адресу 0
    WriteEprom(0,0x1234);                   //Запись в EPROM по адресу 0
    GotoXY(4,24);                           //Курсор в позицию x=4,  y=24
    printf("EPROM[0]=%X",ReadEprom(0));     //Вывод на дисплей содержимого
EPROM по адресу 0
    ShowScr();                              //Вывести на дисплей
    while(1);                               //Бесконечная пауза
}
```

Результат:

См. также
BrushColor ClrScr FontColor BigFont GotoXY printf ShowScr
ReadEprom

3.9. ЭНЕРГОНЕЗАВИСИМАЯ ПАМЯТЬ BACKUPRAM

Энергонезависимая память BACKUPRAM представляет собой 4096 байтов статической памяти. Эта память сохраняется даже при отключении контроллера CAT-500 от сети электропитания за

счёт внутренней резервной батареи, которая также питает часы реального времени. Сброс информации, хранящейся в энергонезависимой памяти RAM возможен только при извлечении внутренней батарейки или выходе батареи из строя.
Для хранения данных в энергонезависимой памяти BACKUPRAM в заголовочном файле *<sat500.h>* определены следующие функции:

Функция	***Описание***
unsigned char BackupRamRead8 (unsigned short int adr)	Чтение байта данных из энергонезависимой памяти BACKUPRAM. adr=0...4095 - адрес в энергонезависимой памяти откуда будет выполнено чтение
void BackupRamWrite8 (unsigned short int adr, unsigned char val)	Сохранение байта данных в энергонезависимой памяти BACKUPRAM adr=0...4095 - адрес в энергонезависимой памяти, куда будет выполнена запись val - записываемое значение
unsigned short int BackupRamRead16 (unsigned short int adr)	Чтение слова данных из энергонезависимой памяти BACKUPRAM. adr=0...4094 - адрес в энергонезависимой памяти откуда будет выполнено чтение
void BackupRamWrite16 (unsigned short int adr, unsigned short int val)	Сохранение слова данных в энергонезависимой памяти BACKUPRAM adr=0...4094 - адрес в энергонезависимой памяти, куда будет выполнена запись val - записываемое значение
unsigned int BackupRamRead32 (unsigned short int adr)	Чтение двойного слова данных из энергонезави-симой памяти BACKUPRAM. adr=0...4092 - адрес в энергонезависимой памяти откуда будет выполнено чтение
void BackupRamWrite32 (unsigned short int adr, unsigned int val)	Сохранение двойного слова данных в энергонезависимой памяти BACKUPRAM adr=0...4092 - адрес в энергонезависимой памяти, куда будет выполнена запись val - записываемое значение
*void BackupRamRead (unsigned short int adr,unsigned char * data, unsigned short int len)*	Чтение массива данных из энергонезависимой памяти BACKUPRAM. adr=0...4095 - адрес в энергонезависимой памяти откуда будет выполнено чтение data - адрес массива в основной памяти куда будут прочитаны данные len - количество байтов данных, которые будут прочитаны из энергонезависимой памяти
*void BackupRamWrite (unsigned short int adr, unsigned char * data, unsigned short int len)*	Запись массива данных из энергонезависимую память BACKUPRAM. adr=0...4095 - адрес в энергонезависимой памяти куда будет выполнена запись data - адрес массива в основной памяти откуда будут браться данные для записи len - количество байтов данных, которые будут записаны в энергонезависимую память

Имитатор сохраняет данные BACKUPRAM в подкаталоге Data с именем компилируемого файла и расширением ".backupram". Например, если компилируемый файл называется "main.c", то программа создаст соответствующий ему файл с данными:
"C:\ProgramData\SatScript\Data\main.backupram"

3.9.1. BACKUPRAMREAD8

```
#include <sat500.h>

unsigned char BackupRamRead8(unsigned short int adr);
```

Чтение байта данных из энергонезависимой памяти BACKUPRAM.
unsigned short int adr - адрес в энергонезависимой памяти BACKUPRAM. Допустимые значения 0...4095
Функция возвращает байт, хранящийся в энергонезависимой памяти по указанному адресу.

В примере ниже каждый запуск программы увеличивает первые 8 байтов энергонезависимой памяти BACKUPRAM.
Пример:

```
#include <sat500.h>

void main(void)
{
    ClrScr();
    printf("BackupRamRead8");
    BigFont=false;
    GotoXY(0,40);
    for(int i=0;i<8;i++)
    {
        unsigned char val=BackupRamRead8(i);
        printf("[%d]=%d\n",i,val);
        BackupRamWrite8(i,++val);
    }
    ShowScr();
    while(1);
}
```

Результат:

3.9.2. BACKUPRAMWRITE8

```
#include <sat500.h>

void BackupRamWrite8(unsigned short int adr,unsigned char val);
```

Запись байта данных в энергонезависимую память BACKUPRAM.
unsigned short int adr - адрес в энергонезависимой памяти BACKUPRAM. Допустимые значения 0...4095
unsigned char val - записываемый байт
Функция записывает байт *val* в энергонезависимую память по указанному адресу *adr*.

В примере ниже каждый запуск программы увеличивает первые 8 байтов энергонезависимой памяти BACKUPRAM.
Пример:

```
#include <sat500.h>

void main(void)
```

```
{
    ClrScr();
    printf("BackupRamWrite8");
    BigFont=false;
    GotoXY(0,40);
    for(int i=0;i<8;i++)
    {
        unsigned char val=BackupRamRead8(i);
        printf("[%d]=%d\n",i,val);
        BackupRamWrite8(i,++val);
    }
    ShowScr();
    while(1);
}
```

Результат:

3.9.3. BACKUPRAMREAD16

```
#include <sat500.h>

unsigned short int BackupRamRead16(unsigned short int adr);
```

Чтение слова данных из энергонезависимой памяти BACKUPRAM.
unsigned short int adr - адрес в энергонезависимой памяти BACKUPRAM. Допустимые значения 0...4094
Функция возвращает слово, хранящееся в энергонезависимой памяти по указанному адресу.

В примере ниже каждый запуск программы увеличивает первые 8 слов энергонезависимой памяти BACKUPRAM.
Пример:

```
#include <sat500.h>

void main(void)
{
    ClrScr();
    printf("BackupRamRead16");
    BigFont=false;
    GotoXY(0,40);
    for(int i=0;i<16;i+=2)
    {
        unsigned short int val=BackupRamRead16(i);
        printf("[%d]=%04X\n",i,val);
        BackupRamWrite16(i,++val);
    }
    ShowScr();
    while(1);
}
```

Результат:

3.9.4. BACKUPRAMWRITE16

```
#include <sat500.h>

void BackupRamWrite16(unsigned short int adr,unsigned short int val);
```

Запись слова данных в энергонезависимую память BACKUPRAM.
unsigned short int adr - адрес в энергонезависимой памяти BACKUPRAM. Допустимые значения 0...4094
unsigned short int val - записываемое слово
Функция записывает слово *val* в энергонезависимую память по указанному адресу *adr*.

В примере ниже каждый запуск программы увеличивает первые 8 слов энергонезависимой памяти BACKUPRAM.
Пример:

```
#include <sat500.h>

void main(void)
{
    ClrScr();
    printf("BackupRamWrite16");
    BigFont=false;
    GotoXY(0,40);
    for(int i=0;i<16;i+=2)
    {
        unsigned short int val=BackupRamRead16(i);
        printf("[%d]=%04X\n",i,val);
        BackupRamWrite16(i,++val);
    }
    ShowScr();
    while(1);
}
```

Результат:

3.9.5. BACKUPRAMREAD32

```
#include <sat500.h>

unsigned int BackupRamRead32(unsigned short int adr);
```

Чтение двойного слова данных из энергонезависимой памяти BACKUPRAM.
unsigned short int adr - адрес в энергонезависимой памяти BACKUPRAM. Допустимые значения 0...4092
Функция возвращает двойное слово, хранящееся в энергонезависимой памяти по указанному адресу.

В примере ниже каждый запуск программы увеличивает первые 8 двойных слов энергонезависимой памяти BACKUPRAM.
Пример:

```
#include <sat500.h>

void main(void)
{
    ClrScr();
```

```
    printf("BackupRamRead32");
    BigFont=false;
    GotoXY(0,40);
    for(int i=0;i<32;i+=4)
    {
        unsigned int val=BackupRamRead32(i);
        printf("[%d]=%08X\n",i,val);
        BackupRamWrite32(i,++val);
    }
    ShowScr();
    while(1);
}
```

Результат:

Дополнительный пример имеется в разделе *Пункты меню*.

3.9.6. BACKUPRAMWRITE32

```
#include <sat500.h>

void BackupRamWrite32(unsigned short int adr,unsigned int val);
```

Запись двойного слова данных в энергонезависимую память BACKUPRAM.
unsigned short int adr - адрес в энергонезависимой памяти BACKUPRAM. Допустимые значения 0...4092
unsigned int val - записываемое двойного слово
Функция записывает двойного слово *val* в энергонезависимую память по указанному адресу *adr*.

В примере ниже каждый запуск программы увеличивает первые 8 двойных слов энергонезависимой памяти BACKUPRAM.

Пример:

```
#include <sat500.h>

void main(void)
{
    ClrScr();
    printf("BackupRamRead32");
    BigFont=false;
    GotoXY(0,40);
    for(int i=0;i<32;i+=4)
    {
        unsigned int val=BackupRamRead32(i);
        printf("[%d]=%08X\n",i,val);
        BackupRamWrite32(i,++val);
    }
    ShowScr();
    while(1);
}
```

Результат:

3.9.7. BACKUPRAMREAD

```
#include <sat500.h>

void BackupRamRead(unsigned short int adr,unsigned char * data, unsigned short int len);
```

Чтение массива данных из энергонезависимой памяти BACKUPRAM.
unsigned short int adr - адрес в энергонезависимой памяти BACKUPRAM. Допустимые значения 0...4095
unsigned short int data - адрес массива в основной памяти куда будут прочитаны данные
unsigned short int len - количество байтов данных, которые будут прочитаны из энергонезависимой памяти
Функция читает len байтов данных из энергонезависимой памяти по указанному адресу *adr* и помещает результат в основную память по адресу *data*.

В примере ниже в первые 8 байтов энергонезависимой памяти BACKUPRAM записываются нарастающие числа и затем они читаются.
Пример:

```
#include <sat500.h>

char Data[]={0,1,2,3,4,5,6,7};
char DataRead[8];
void main(void)
{
    ClrScr();
    printf("BackupRamRead");
    BigFont=false;
    GotoXY(0,40);
    BackupRamWrite(0,Data,8);
    BackupRamRead(0,DataRead,8);
    for(int i=0;i<8;i++)
    {
        printf("[%d]=%d\n",i,DataRead[i]);
    }
    ShowScr();
    while(1);
}
```

Результат:

3.9.8. BACKUPRAMWRITE

```
#include <sat500.h>

void BackupRamWrite(unsigned short int adr,unsigned char * data, unsigned short int len);
```

Чтение массива данных из энергонезависимой памяти BACKUPRAM.
unsigned short int adr - адрес в энергонезависимой памяти BACKUPRAM. Допустимые значения 0...4095
*unsigned char * data* - адрес массива в основной памяти откуда будут взяты данные для записи
unsigned short int len - количество байтов данных, которые будут записаны в энергонезависимую память

Функция записывает *len* байтов данных в энергонезависимую память по указанному адресу *adr* из основной памяти по адресу *data*.

В примере ниже в первые 8 байтов энергонезависимой памяти BACKUPRAM записываются нарастающие числа и затем они читаются.
Пример:

```
#include <sat500.h>

char Data[]={0,1,2,3,4,5,6,7};
char DataRead[8];
void main(void)
{
    ClrScr();
    printf("BackupRamRead");
    BigFont=false;
    GotoXY(0,40);
    BackupRamWrite(0,Data,8);
    BackupRamRead(0,DataRead,8);
    for(int i=0;i<8;i++)
    {
        printf("[%d]=%d\n",i,DataRead[i]);
    }
    ShowScr();
    while(1);
}
```

Результат:

3.10. ДОПОЛНИТЕЛЬНАЯ ПАМЯТЬ EXTRAM

Контроллер CAT-500 имеет дополнительную память EXTRAM к которой можно обращаться через несколько специальных функций.
Размер доступной дополнительной внешней памяти можно определить при помощи функции GetExtRamSize().
В момент запуска программы на языке С-скрипт содержимое дополнительной памяти обнуляется.

Функция	***Описание***
unsigned int GetExtRamSize (void)	Получение размера доступной внешней памяти
unsigned char ReadExtRamByte (unsigned int extadr)	Чтение байта из внешней памяти по адресу
unsigned short int ReadExtRamWord (unsigned int extadr)	Чтение слова из внешней памяти по адресу
unsigned int ReadExtRamUint (unsigned int extadr)	Чтение двойного слова из внешней памяти по адресу
void WriteExtRamByte (unsigned int extadr,unsigned char byte)	Запись байта во внешнюю память
void WriteExtRamWord (unsigned int extadr,unsigned short int word)	Запись слова во внешнюю память
void WriteExtRamUint (unsigned int extadr,unsigned int uint)	Запись двойного слова во внешнюю память
*void ReadExtRam (unsigned int extadr,unsigned char * ram,unsigned int cnt)*	Чтение массива байтов из внешней памяти по адресу

*void WriteExtRam (unsigned int extadr,unsigned char * ram,unsigned int cnt)*	Запись массива байтов во внешнюю память по адресу

3.10.1. GETEXTRAMSIZE

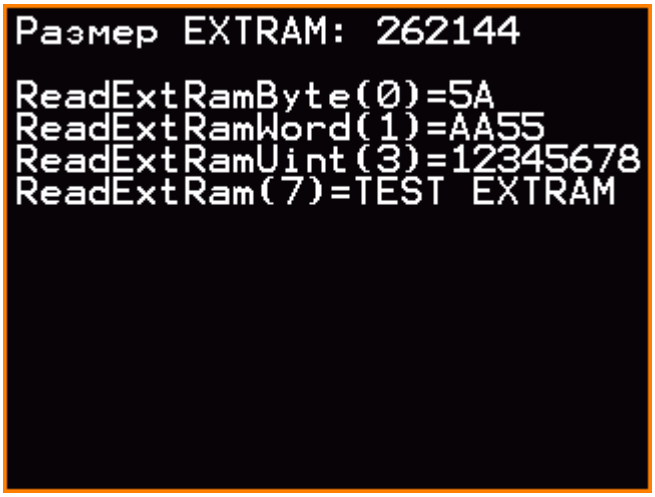
```
#include <sat500.h>

unsigned int GetExtRamSize(void);
```

Получение размера доступной внешней памяти EXTRAM.
Функция возвращает количество байт, которые можно использовать для хранения собственных данных в памяти EXTRAM.
Во всех функциях этого раздела можно использовать адреса в памяти EXTRAM в диапазоне 0...(`GetExtRamSize()-1`)

Пример:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    printf("Размер EXTRAM: %u\n\n",GetExtRamSize());
    //Запись в EXTRAM
    WriteExtRamByte(0,0x5a);
    WriteExtRamWord(1,0xAA55);
    WriteExtRamUint(3,0x12345678);
    WriteExtRam(7,"TEST EXTRAM\0",12);
    //Чтение из EXTRAM
    unsigned char byte=ReadExtRamByte(0);
    printf("ReadExtRamByte(0)=%02X\n",byte);
    unsigned short word=ReadExtRamWord(1);
    printf("ReadExtRamWord(1)=%04X\n",word);
    unsigned int uint=ReadExtRamUint(3);
    printf("ReadExtRamUint(3)=%08X\n",uint);
    char str[12];
    ReadExtRam(7,str,12);
    printf("ReadExtRam(7)=%s\n",str);
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
ReadExtRamByte *ReadExtRamWord* *ReadExtRamUint*
WriteExtRamByte *WriteExtRamWord* *WriteExtRamUint*
ReadExtRam *WriteExtRam*

3.10.2. READEXTRAMBYTE

```
#include <sat500.h>

unsigned char ReadExtRamByte(unsigned int extadr);
```

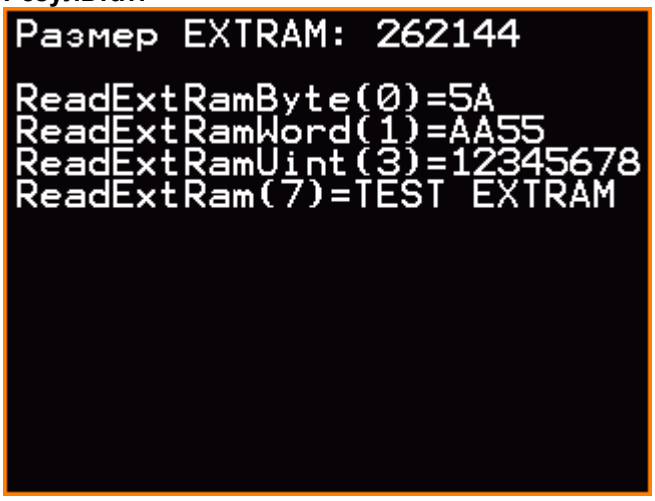
Чтение байта из внешней памяти EXTRAM по адресу *extadr*.
unsigned int extadr - адрес в дополнительной памяти EXTRAM.
Значение адреса *extadr* может находится в диапазоне 0...(*GetExtRamSize()-1*)
Функция возвращает значение байта по указанному адресу.
Если задан неправильный адрес, то возвращается нулевое значение.

Пример:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    printf("Размер EXTRAM: %u\n\n",GetExtRamSize());
    //Запись в EXTRAM
    WriteExtRamByte(0,0x5a);
    WriteExtRamWord(1,0xAA55);
    WriteExtRamUint(3,0x12345678);
    WriteExtRam(7,"TEST EXTRAM\0",12);
    //Чтение из EXTRAM
    unsigned char byte=ReadExtRamByte(0);
    printf("ReadExtRamByte(0)=%02X\n",byte);
    unsigned short word=ReadExtRamWord(1);
    printf("ReadExtRamWord(1)=%04X\n",word);
    unsigned int uint=ReadExtRamUint(3);
    printf("ReadExtRamUint(3)=%08X\n",uint);
```

```
    char str[12];
    ReadExtRam(7,str,12);
    printf("ReadExtRam(7)=%s\n",str);
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
GetExtRamSize *ReadExtRamWord* *ReadExtRamUint*
WriteExtRamByte *WriteExtRamWord* *WriteExtRamUint*
ReadExtRam *WriteExtRam*

3.10.3. READEXTRAMWORD

```
#include <sat500.h>

unsigned short int ReadExtRamWord(unsigned int extadr);
```

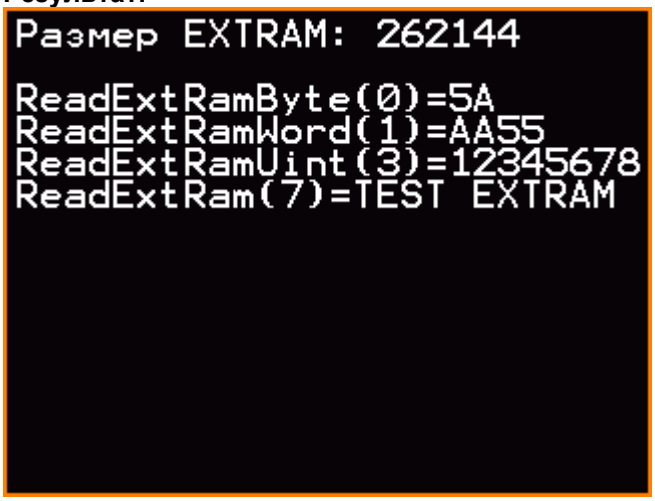
Чтение слова (два байта) из внешней памяти EXTRAM по адресу *extadr*.
unsigned int extadr - адрес в дополнительной памяти EXTRAM.
Значение адреса *extadr* может находится в диапазоне *0...(GetExtRamSize()-2)*
Функция возвращает значение двухбайтового слова по указанному адресу.
Если задан неправильный адрес, то возвращается нулевое значение.

Пример:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    printf("Размер EXTRAM: %u\n\n",GetExtRamSize());
    //Запись в EXTRAM
    WriteExtRamByte(0,0x5a);
```

```
    WriteExtRamWord(1,0xAA55);
    WriteExtRamUint(3,0x12345678);
    WriteExtRam(7,"TEST EXTRAM\0",12);
    //Чтение из EXTRAM
    unsigned char byte=ReadExtRamByte(0);
    printf("ReadExtRamByte(0)=%02X\n",byte);
    unsigned short word=ReadExtRamWord(1);
    printf("ReadExtRamWord(1)=%04X\n",word);
    unsigned int uint=ReadExtRamUint(3);
    printf("ReadExtRamUint(3)=%08X\n",uint);
    char str[12];
    ReadExtRam(7,str,12);
    printf("ReadExtRam(7)=%s\n",str);
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также

GetExtRamSize *ReadExtRamByte* *ReadExtRamUint* *WriteExtRamByte* *WriteExtRamWord* *WriteExtRamUint* *ReadExtRam* *WriteExtRam*

3.10.4. READEXTRAMUINT

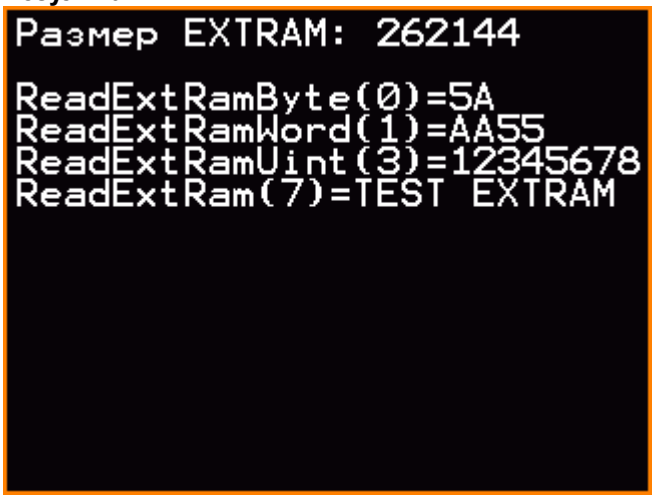
```
#include <sat500.h>

unsigned int ReadExtRamUint(unsigned int extadr);
```

Чтение двойного слова (четыре байта) из внешней памяти EXTRAM по адресу *extadr*.
unsigned int extadr - адрес в дополнительной памяти EXTRAM.
Значение адреса *extadr* может находится в диапазоне *0...(GetExtRamSize()-4)*
Функция возвращает четырёхбайтовое значение по указанному адресу.
Если задан неправильный адрес, то возвращается нулевое значение.

Пример:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    printf("Размер EXTRAM: %u\n\n",GetExtRamSize());
    //Запись в EXTRAM
    WriteExtRamByte(0,0x5a);
    WriteExtRamWord(1,0xAA55);
    WriteExtRamUint(3,0x12345678);
    WriteExtRam(7,"TEST EXTRAM\0",12);
    //Чтение из EXTRAM
    unsigned char byte=ReadExtRamByte(0);
    printf("ReadExtRamByte(0)=%02X\n",byte);
    unsigned short word=ReadExtRamWord(1);
    printf("ReadExtRamWord(1)=%04X\n",word);
    unsigned int uint=ReadExtRamUint(3);
    printf("ReadExtRamUint(3)=%08X\n",uint);
    char str[12];
    ReadExtRam(7,str,12);
    printf("ReadExtRam(7)=%s\n",str);
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также

GetExtRamSize ReadExtRamByte ReadExtRaWord
WriteExtRamByte WriteExtRamWord WriteExtRamUint
ReadExtRam WriteExtRam

3.10.5. WRITEEXTRAMBYTE

```
#include <sat500.h>
```

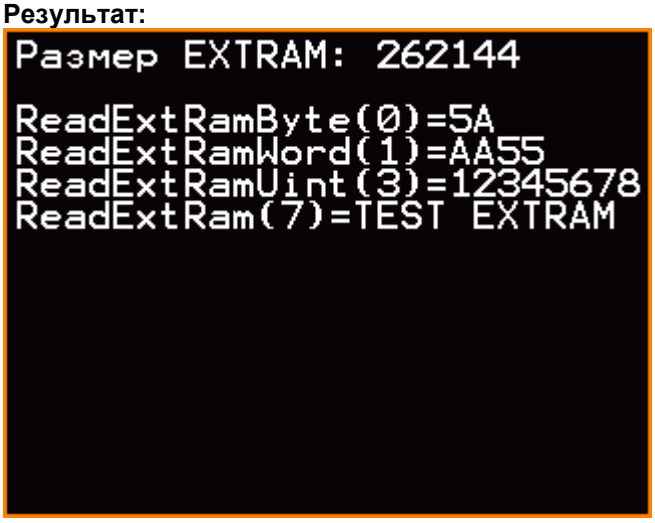
```
void WriteExtRamByte(unsigned int extadr,unsigned char byte);
```

Запись байта во внешнюю память EXTRAM по адресу `extadr`.
unsigned int extadr - адрес в дополнительной памяти EXTRAM.
Значение адреса `extadr` может находится в диапазоне `0...(GetExtRamSize()-1)`
unsigned char byte - Значение байта, записываемого в дополнительную память.
Если задан неправильный адрес, то операция записи не выполняется.

Пример:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    printf("Размер EXTRAM: %u\n\n",GetExtRamSize());
    //Запись в EXTRAM
    WriteExtRamByte(0,0x5a);
    WriteExtRamWord(1,0xAA55);
    WriteExtRamUint(3,0x12345678);
    WriteExtRam(7,"TEST EXTRAM\0",12);
    //Чтение из EXTRAM
    unsigned char byte=ReadExtRamByte(0);
    printf("ReadExtRamByte(0)=%02X\n",byte);
    unsigned short word=ReadExtRamWord(1);
    printf("ReadExtRamWord(1)=%04X\n",word);
    unsigned int uint=ReadExtRamUint(3);
    printf("ReadExtRamUint(3)=%08X\n",uint);
    char str[12];
    ReadExtRam(7,str,12);
    printf("ReadExtRam(7)=%s\n",str);
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
GetExtRamSize *ReadExtRamByte* *ReadExtRaWord* *ReadExtRamUint*
WriteExtRamWord *WriteExtRamUint*
ReadExtRam *WriteExtRam*

3.10.6. WRITEEXTRAMWORD

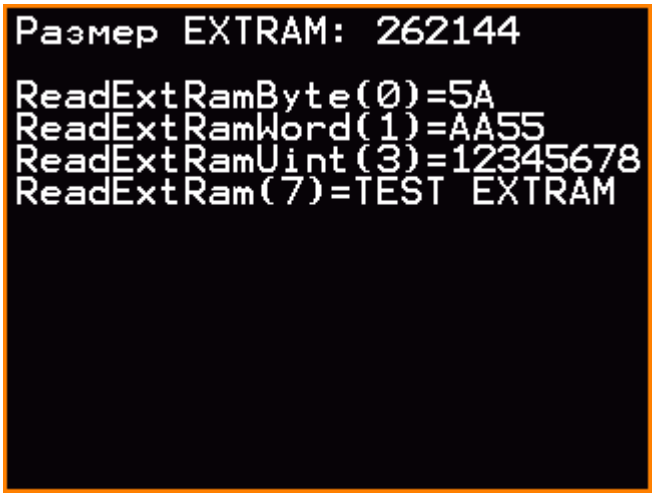
```
#include <sat500.h>

void WriteExtRamWord(unsigned int extadr,unsigned short int word);
```

Запись слова (два байта) во внешнюю память EXTRAM по адресу `extadr`.
unsigned int extadr - адрес в дополнительной памяти EXTRAM.
Значение адреса `extadr` может находится в диапазоне `0...(GetExtRamSize()-2)`
unsigned short int word - Значение слова, записываемого в дополнительную память.
Если задан неправильный адрес, то операция записи не выполняется.

Пример:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    printf("Размер EXTRAM: %u\n\n",GetExtRamSize());
    //Запись в EXTRAM
    WriteExtRamByte(0,0x5a);
    WriteExtRamWord(1,0xAA55);
    WriteExtRamUint(3,0x12345678);
    WriteExtRam(7,"TEST EXTRAM\0",12);
    //Чтение из EXTRAM
    unsigned char byte=ReadExtRamByte(0);
    printf("ReadExtRamByte(0)=%02X\n",byte);
    unsigned short word=ReadExtRamWord(1);
    printf("ReadExtRamWord(1)=%04X\n",word);
    unsigned int uint=ReadExtRamUint(3);
    printf("ReadExtRamUint(3)=%08X\n",uint);
    char str[12];
    ReadExtRam(7,str,12);
    printf("ReadExtRam(7)=%s\n",str);
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
GetExtRamSize *ReadExtRamByte* *ReadExtRaWord* *ReadExtRamUint*
WriteExtRamByte *WriteExtRamUint*
ReadExtRam *WriteExtRam*

3.10.7. WRITEEXTRAMUINT

```
#include <sat500.h>

void WriteExtRamUint(unsigned int extadr,unsigned int uint);
```

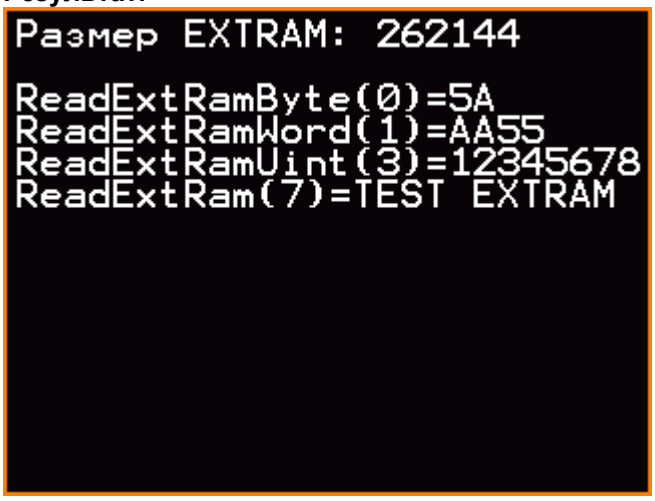
Запись двойного слова (четыре байта) во внешнюю память EXTRAM по адресу `extadr`.
unsigned int extadr - адрес в дополнительной памяти EXTRAM.
Значение адреса `extadr` может находится в диапазоне `0...(GetExtRamSize()-4)`
unsigned int uint - Значение двойного слова, записываемого в дополнительную память.
Если задан неправильный адрес, то операция записи не выполняется.

Пример:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    printf("Размер EXTRAM: %u\n\n",GetExtRamSize());
    //Запись в EXTRAM
    WriteExtRamByte(0,0x5a);
    WriteExtRamWord(1,0xAA55);
    WriteExtRamUint(3,0x12345678);
    WriteExtRam(7,"TEST EXTRAM\0",12);
    //Чтение из EXTRAM
    unsigned char byte=ReadExtRamByte(0);
    printf("ReadExtRamByte(0)=%02X\n",byte);
    unsigned short word=ReadExtRamWord(1);
    printf("ReadExtRamWord(1)=%04X\n",word);
    unsigned int uint=ReadExtRamUint(3);
    printf("ReadExtRamUint(3)=%08X\n",uint);
```

```
    char str[12];
    ReadExtRam(7,str,12);
    printf("ReadExtRam(7)=%s\n",str);
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также
GetExtRamSize *ReadExtRamByte* *ReadExtRaWord* *ReadExtRamUint*
WriteExtRamByte *WriteExtRamWord*
ReadExtRam *WriteExtRam*

3.10.8. READEXTRAM

```
#include <sat500.h>

void ReadExtRam(unsigned int extadr,unsigned char * ram,unsigned int cnt);
```

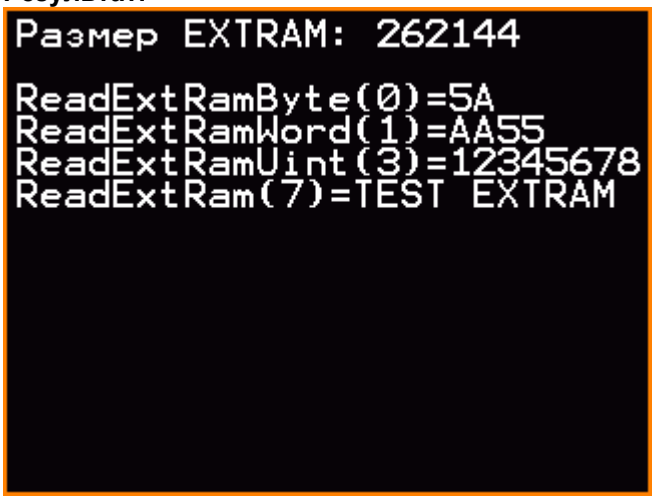
Чтение заданного количества байтов из внешней памяти EXTRAM по адресу `extadr` в оперативную память RAM по адресу `ram`.
unsigned int extadr - адрес в дополнительной памяти EXTRAM.
Значение адреса `extadr` может находится в диапазоне 0...(`GetExtRamSize()-cnt`)
*unsigned char * ram* - адрес в RAM памяти куда следует прочитать массив байтов.
unsigned int cnt - количество байтов которые нужно прочитать.
Если задан неправильный адрес или неправильное количество байтов, то массив *ram* заполняется нулевыми значениями.

Пример:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
```

```
    printf("Размер EXTRAM: %u\n\n",GetExtRamSize());
    //Запись в EXTRAM
    WriteExtRamByte(0,0x5a);
    WriteExtRamWord(1,0xAA55);
    WriteExtRamUint(3,0x12345678);
    WriteExtRam(7,"TEST EXTRAM\0",12);
    //Чтение из EXTRAM
    unsigned char byte=ReadExtRamByte(0);
    printf("ReadExtRamByte(0)=%02X\n",byte);
    unsigned short word=ReadExtRamWord(1);
    printf("ReadExtRamWord(1)=%04X\n",word);
    unsigned int uint=ReadExtRamUint(3);
    printf("ReadExtRamUint(3)=%08X\n",uint);
    char str[12];
    ReadExtRam(7,str,12);
    printf("ReadExtRam(7)=%s\n",str);
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также

GetExtRamSize *ReadExtRamByte* *ReadExtRaWord* *ReadExtRamUint* *WriteExtRamByte* *WriteExtRamWord* *WriteExtRamUint* *WriteExtRam*

3.10.9. WRITEEXTRAM

```
#include <sat500.h>

void WriteExtRamByte(unsigned int extadr,unsigned char * ram,unsigned int
cnt);
```

Запись заданного количества байтов во внешнюю память EXTRAM по адресу `extadr` из оперативной памяти RAM по адресу `ram`.

unsigned int extadr - адрес в дополнительной памяти EXTRAM.

Значение адреса `extadr` может находится в диапазоне 0...(`GetExtRamSize()-cnt`)

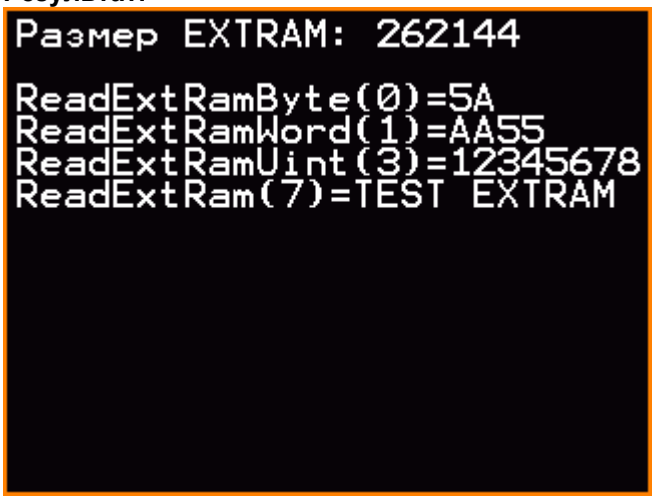
*unsigned char * ram* - адрес в RAM памяти откуда следует взять массив байтов.

unsigned int cnt - количество байтов которые нужно записать.
Если задан неправильный адрес или неправильное количество байтов, то операции записи не выполняется.

Пример:

```
#include <sat500.h>
#include <math.h>

void main(void)
{
    int v;
    ClrScr();
    BigFont=false;
    printf("Размер EXTRAM: %u\n\n",GetExtRamSize());
    //Запись в EXTRAM
    WriteExtRamByte(0,0x5a);
    WriteExtRamWord(1,0xAA55);
    WriteExtRamUint(3,0x12345678);
    WriteExtRam(7,"TEST EXTRAM\0",12);
    //Чтение из EXTRAM
    unsigned char byte=ReadExtRamByte(0);
    printf("ReadExtRamByte(0)=%02X\n",byte);
    unsigned short word=ReadExtRamWord(1);
    printf("ReadExtRamWord(1)=%04X\n",word);
    unsigned int uint=ReadExtRamUint(3);
    printf("ReadExtRamUint(3)=%08X\n",uint);
    char str[12];
    ReadExtRam(7,str,12);
    printf("ReadExtRam(7)=%s\n",str);
    ShowScr();
    while(1)
    {
    }
}
```

Результат:



См. также

GetExtRamSize *ReadExtRamByte* *ReadExtRaWord* *ReadExtRamUint*
WriteExtRamByte *WriteExtRamWord* *WriteExtRamUint*
ReadExtRam

3.11. СЛУЧАЙНОЕ ЧИСЛО

Для получения случайных чисел в заголовочном файле <*sat500.h*> определены две функции:

Функция	***Описание***
int rand(void)	Функция возвращает псевдослучайное целое в интервале от 0 до 0x7FFFFFFF
void randomize(unsigned int seed)	Инициализирует генератор псевдослучайных чисел значением *seed*

3.11.1. RAND

```
#include <sat500.h>

int rand(void);
```

Функция *rand()* возвращает псевдослучайное целое в интервале от 0 до 0x7FFFFFFF.
Использует аддитивный генератор (Mitchell & Moore) вида:
Xn = (X[n-24] + X[n-55]) MOD 2^31

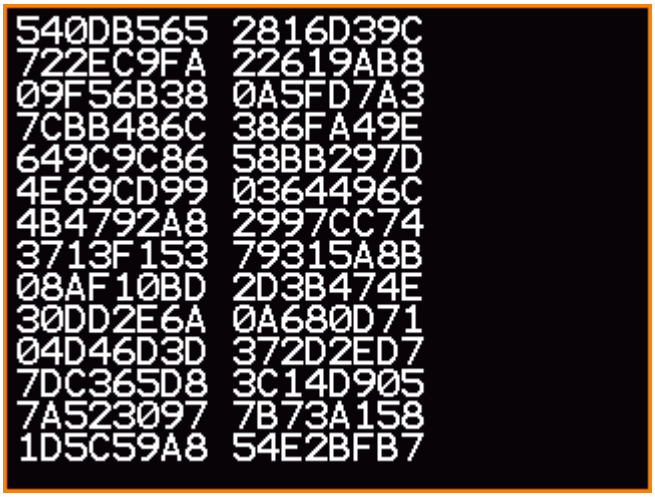
Для того чтобы ограничить диапазон значений генерируемых псевдослучайных чисел можно использовать остаток от деления.
Например следующий вызов генерит случайное число в диапазоне 0...99:

```
    int v=rand() % 100;
```

Пример:

```
#include <sat500.h>
#pragma version 1.0

void main(void)
{
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    BigFont=0;                   //Выбор шрифта - обычный шрифт
    srand(GetTickCount());       //Инициализация псевдослучайного генератора
    for(int i=0;i<15;i++)        //Вывести 15 строк чисел
    {
        for(int k=0;k<2;k++)     //Вывести 2 числа в строке
        {
            int val=rand();      //Псевдослучайное число в диапазоне
0...0x7fffffff
            printf("%08X ",val);//Вывод на дисплей
        }
        putchar('\n');           //Перевод строки
    }
    ShowScr();                   //Вывести на дисплей
    while(1);                    //Бесконечная пауза
}
```

Результат:



См. также
srand ClrScr GetTickCount printf putchar ShowScr
BrushColor BigFont

3.11.2. SRAND

```
#include <sat500.h>

void srand (unsigned int seed);
```

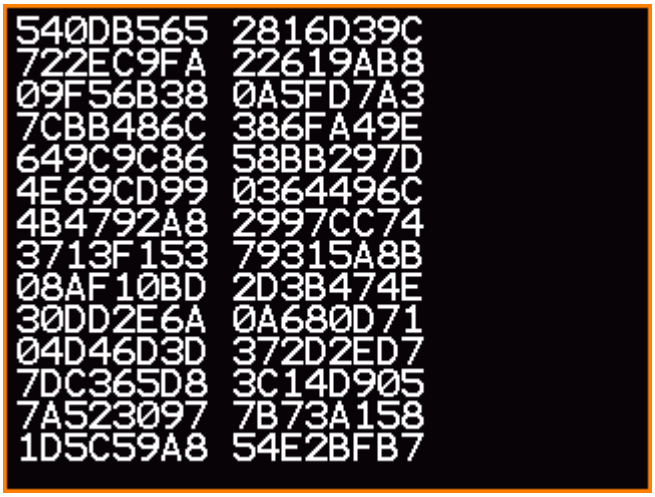
Функция *srand(seed)* инициализирует генератор псевдослучайных чисел значением seed.
Аргумент *seed* используется как начальное значение для новой последовательности псевдослучайных чисел,
возвращаемых функцией *rand()*.
Если далее srand вызываеть с помощью того же самого начального значения *seed*, то последовательность псевдослучайных чисел повторяется.
Если *rand()* вызывается до того, как были сделаны какие-либо вызовы *srand()*, то генерится псевдослучайная последовать с начальным значением *seed* по умолчанию (равно 1).

Пример:

```
#include <sat500.h>
#pragma version 1.0

void main(void)
{
    BrushColor=COLOR_BLACK;      //Цвет кисти - чёрный
    ClrScr();                    //Очистить экран чёрным цветом
    BigFont=0;                   //Выбор шрифта - обычный шрифт
    srand(GetTickCount());       //Инициализация псевдослучайного генератора
    for(int i=0;i<15;i++)        //Вывести 15 строк чисел
    {
        for(int k=0;k<2;k++)     //Вывести 2 числа в строке
        {
            int val=rand();      //Псевдослучайное число
            printf("%08X ",val);//Вывод на дисплей
        }
        putchar('\n');           //Перевод строки
    }
```

```
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



См. также
srand ClrScr GetTickCount printf putchar ShowScr
BrushColor BigFont

3.12. TCP КЛИЕНТ

Заголовочный файл <*tcp.h*> содержит функции работы с TCP/IP протоколом, обеспечивая режим клиента TCP соединения.
Поддерживается работа только одного клиента в каждый момент времени.
Для начала подключения используется функция *TcpOpen*.
Для проверки наличия соединения следует использовать функцию *TcpError* или функцию *TcpConnected*
Только после того как соединение с сервером установлено можно выполнять запись *TcpWrite* и чтение *TcpReadLength* + *TcpRead* из соединения.
Установленное соединение имеет входной буфер, в который помещаются принятые данные. Количество данных в буфере можно получить при помощи функции *TcpReadLength*, а прочитать данные с помощью функции *TcpRead*.
Если входной поток данных таков, что в приёмном буфере нет места, то увеличивается специальный счётчик потерянных байтов, который можно прочитать и обнулить при помощи переменной *TcpLost*.

Доступны следующие переменные:

Переменная	***Описание***
unsigned short int TcpLost	Переменная, которая содержит количество байтов, утерянных из-за переполнения приёмного буфера TCP соединения
char TcpState	Переменная, которая содержит состояние TCP клиента

Доступны следующие функции:

Функция	***Описание***

*char TcpOpen (char * ip,unsigned short int port,unsigned int timeout)*	Выполнить подключение к TCP серверу по адресу ip:port за время не более timeout мс. *ip* - IP или DNS адрес в текстовом виде. *port* - номер порта. *timeout* - таймаут операции подключения в миллисекундах. Возвращаемое значение: 1 - успешно запущен процесс подключения 0 - не удалось запустить процесс подключения
void TcpClose (void)	Закрыть открытое ранее TCP подключение к серверу
char TcpConnected(void)	Проверить, что имеется TCP подключение к серверу. Возвращаемое значение: 1 - подключение есть 0 - подключение отсутствует
*short int TcpWrite (char * data,short int len)*	Записать в открытое TCP подключение *data* - указательна записываемые данные l*en* - количество записываемых данных Возвращаемое значение: количество записанных данных или 0 если TCP подключение не открыто
short int TcpReadLength (void)	Получить количество принятых байтов открытого TCP подключения к серверу Возвращаемое значение: количество байтов в буфере или 0 если данных нет или TCP подключение не открыто
*short int TcpRead (char * data,short int size)*	Прочитать не более *size* байтов, полученных от сервера открытого TCP подключения Возвращаемое значение: количество байтов прочитанных в *data* или 0 если данных нет, или TCP подключение не открыто
void TcpClear(void)	Очистить приёмный буфер TCP подключения

3.12.1. TCPOPEN

```
#include <tcp.h>

char TcpOpen(char * ip,unsigned short int port,unsigned int timeout,unsigned
short bufsize);
```

Выполнить подключение к TCP серверу по адресу *ip:port* за время не более *timeout* мс.

*char * ip* - IP или DNS адрес в текстовом виде.
unsigned short int port - номер сетевого порта.
tunsigned int imeout - таймаут операции подключения в миллисекундах.
unsigned short bufsize - размер буфера в системной памяти контроллера, который будет автоматически в момент вызова функции *TcpOpen* и удалён в момент вызова *TcpClose*. Задаётся в диапазоне 256...8192. Следует задавать минимальный размер буфера, обеспечивающий корректную работу конкретного приложения.

Возвращаемое значение:
1 - успешно запущен процесс подключения
0 - не удалось запустить процесс подключения (ошибка в аргументах функции или не выделена системная память)

Успешное выполнение функции ещё не обозначает, что уже установлено соединение с сервером.

Для проверки наличия соединения следует использовать переменную *TcpState* или функцию *TcpConnected*

Только после того как соединение с сервером установлено можно выполнять запись *TcpWrite* и чтение *TcpReadLength* + *TcpRead* из соединения.
Установленное соединение имеет входной буфер (размером *bufsize*), в который помещаются принятые данные. Количество данных в буфере можно получить при помощи функции *TcpReadLength*, а прочитать данные из буфера с помощью функции *TcpRead*.
Если входной поток данных таков, что в приёмном буфере нет места, то увеличивается специальный счётчик потерянных байтов, который можно прочитать, и обнулить при помощи переменной *TcpLost*.

В примере ниже выполняется TCP подключение к второму контроллеру CAT-500 (192.168.1.8) к порту 502, на котором работает TCP MODBUS сервер.
В результате на дисплее отображается статус- доступен ли второй CAT-500.

Пример:

```
#include <sat500.h>
#include <tcp.h>

void main(void)
{
    ClrScr();                  //Очистить экран
   BigFont=0;                 //Обычный шрифт 12x16 (не крупный)
   //Открыть TCP
   puts("Открывание TCP ...\n");
   //Порт - 502, таймаут 1 сек
   if(TcpOpen("192.168.1.8",502,1000))
   {
      puts("Выполняется подключение...\n");
      ShowScr();                  //Вывести не дисплей
      while(1)
      {
         char result;
         do
         {
            result=TcpState;
         }
         while(result<0);
         GotoXY(0,40);
         if(result==0)
         {
             puts("Подключение OK\n");
         }
         else
         {
             puts("Сервер недоступен\n");
             printf("result=%d\n",result);
         }
         ShowScr();   //Вывести на дисплей
         break;
      }
      //Закрыть подключение
      TcpClose();
      puts("Подключение закрыто\n");
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                 //Вывести на дисплей
    while(1);                  //Бесконечная пауза
```

```
}
```

Результат:

Посмотреть ещё пример.

3.12.2. TCPCLOSE

```
#include <tcp.h>

void TcpClose (void);
```

Закрыть открытое ранее TCP подключение к серверу.

В примере ниже выполняется TCP подключение к второму контроллеру CAT-500 (192.168.1.8) к порту 502, на котором работает TCP MODBUS сервер.
В результате на дисплее отображается статус - доступен ли второй CAT-500.

Пример:

```
#include <sat500.h>
#include <tcp.h>

void main(void)
{
   ClrScr();                  //Очистить экран
   BigFont=0;                 //Обычный шрифт 12x16 (не крупный)
   //Открыть TCP
   puts("Открывание TCP ...\n");
   //Порт - 502, таймаут 1 сек
   if(TcpOpen("192.168.1.8",502,1000))
   {
      puts("Выполняется подключение...\n");
      ShowScr();                  //Вывести не дисплей
      while(1)
      {
         char result;
         do
         {
            result=TcpState;
         }
         while(result<0);
         GotoXY(0,40);
         if(result==0)
         {
             puts("Подключение OK\n");
         }
         else
         {
             puts("Сервер недоступен\n");
             printf("result=%d\n",result);
         }
         ShowScr();   //Вывести на дисплей
         break;
      }
```

```
      //Закрыть подключение
      TcpClose();
      puts("Подключение закрыто\n");
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:

Посмотреть ещё пример.

3.12.3. TCPCONNECTED

```
#include <tcp.h>

char TcpConnected (void);
```

Проверить, что имеется TCP подключение к серверу.

Возвращаемое значение:
1 - подключение есть
0 - подключение отсутствует

Перейти к *примеру*.

3.12.4. TCPSTATE

```
#include <tcp.h>

char TcpState;
```

Проверить состояние подключения к серверу.

Возвращаемое значение:
Возможные возвращаемые значения показаны в таблице ниже.

Значение	*Описание*
-1	Операция подключения к серверу ещё выполняется
0	Ошибок нет - есть подключение к серверу
1	Произошла ошибка - не задан адрес сервера
2	Не удалось получить IP адрес сервера
3	Произошла ошибка при выполнении DNS запроса
4	Произошла ошибка при выполнении подключения
5	Сервер разорвал подключение
6	Не удалось подключиться к серверу
7	Не удалось начать подключение к серверу
8	Произошла ошибка записи в TCP соединение

Переменная доступна только для чтения.

Для удобства доступа к переменной во включаемом файле *<tcp.h>* определены следующие константы:

```
enum
{
      tcpBUSY=-1,              //Операция подключения к серверу ещё выполняется
      tcpOK=0,                 //Ошибок нет - есть подключение к серверу
      tcpNOADR=1,              //Произошла ошибка - не задан адрес сервера
      tcpDNS=2,                //Не удалось получить IP адрес сервера
      tcpBADDNS=3,             //Произошла ошибка при выполнении DNS запроса
      tcpNOCONNECTION=4,       //Произошла ошибка при выполнении подключения
      tcpSERVER=5,             //Сервер разорвал подключение
      tcpNOSERVER=6,           //Не удалось подключиться к серверу
      tcpSTART=7,              //Не удалось начать подключение к серверу
      tcpWRITE=8               //Произошла ошибка записи в TCP соединение
};
```

Также вместо переменной *TcpState* можно использовать функцию *char TcpError(void)*, которая возвращает то же значение.

Перейти к *примеру*.

3.12.5. TCPWRITE

```
#include <tcp.h>

unsigned short int TcpWrite(char * data,unsigned short int len);
```

Записать в открытое TCP подключение.
*char * data* - указательна записываемые данные.
short int len - количество записываемых данных.

Возвращаемое значение: количество записанных данных или 0 если TCP подключение не открыто.

Перейти к *примеру*.

3.12.6. TCPREADLENGTH

```
#include <tcp.h>

unsigned short int TcpReadLength(void);
```

Получить количество принятых байтов открытого TCP подключения к серверу.

Возвращаемое значение: количество байтов в буфере или 0 если данных нет или TCP подключение не открыто.

Перейти к *примеру*.

3.12.7. TCPREAD

```
#include <tcp.h>

unsigned short int TcpRead(char * data,unsigned short int size);
```

Прочитать не более *size* байтов, полученных от сервера открытого TCP подключения.
*char * data* - указатель на массив куда будут прочитаны данные открытого TCP подключения.
short int size - размер массива data

Возвращаемое значение: количество байтов прочитанных в *data* или 0 если данных нет, или TCP подключение не открыто.

Перейти к *примеру*.

3.12.8. TCPCLEAR

```
#include <tcp.h>

void TcpClear(void);
```

Очистить приёмный буфер TCP подключения.

Перейти к *примеру*.

3.12.9. TCPLOST

```
#include <tcp.h>

unsigned short int TcpLost;
```

Переменная, которая содержит количество байтов, утерянных из-за переполнения приёмного буфера TCP соединения.

Перейти к *примеру*.

3.12.10. ПРИМЕР

В этом примере TCP клиент выполняет подключение к серверу погоды https://openweathermap.org, выполняет получение погоды для указанного города и в случае успеха выводит погоду на дисплей. Через 10 секунд получается погода для следующего из 7 городов в списке.

Пример:

```
#include <sat500.h>
#include <string.h>
#include <tcp.h>
//-----------------------------------------------------------------------------
//Для полученимя погоды используется API, доступный здесь:
//https://openweathermap.org/api
//-----------------------------------------------------------------------------
typedef char bool;
//-----------------------------------------------------------------------------
//Функции
//Показать погоду из полученного Json
void ShowCity(void);
//Поиск переменной в Json
bool FindJsonVar(char * name,char * val);
//Пауза миллисекунд
void Pause(unsigned int to);
```

```
//-----------------------------------------------------------------------------
//Шаблон HTTP запроса
//%s - название города из массива Cities[]
//9e5daae430bd4a45557ac47f51c90103 - ключ API
//Используйте свой ключ, получаемый после регистрации на openweathermap.org
const char AskTemplate[]=
"GET /data/2.5/weather?q=%s&appid=9e5daae430bd4a45557ac47f51c90103 HTTP/1.1\n\
Host: api.openweathermap.org\n\
Connection: keep-alive\n\n";
//-----------------------------------------------------------------------------
//Получаемый от сервера json
char Json[1024];
//Города:
const char * Cities[]=
{
    "Moscow",
    "Saint Petersburg",
    "Tambov",
    "Murmansk",
    "Yekaterinburg",
    "Vladivostok",
    "Sochi"
};
//Количество городов
#define Q_CITIES  7
//Текущий гоод
int CurCity;
//-----------------------------------------------------------------------------
void main(void)
{
Start:;
    //Очистить экран
    ClrScr();
    //Обычный шрифт 12x16 (не крупный)
    BigFont=0;
    //Открыть TCP
    puts("Открывание TCP ...\n");
    //Порт - 502, таймаут 5 сек
    if(TcpOpen("api.openweathermap.org",80,5000,2048))
    {
        puts("Выполняется подключение...\n");
        //Вывести не дисплей
        ShowScr();
        while(1)
        {
            char result;
            //Ожидаем подключения
            do
            {
                 result=TcpState;
            }
            while(result<0);
            GotoXY(0,40);
            if(result==0)
            {
                //Успешно подключились
                puts("Подключение OK\n");
                //Вывести на дисплей
```

```
                ShowScr();
                //Подготовить HTTP запрос
                sprintf(Json,AskTemplate,Cities[CurCity]);
                //Отправить HTTP запрос на сервер
                TcpWrite(Json,strlen(Json));
                //Ожидать ответа от сервера
                for(int i=0;i<100;i++)
                {
                    Pause(50);
                    //Сколько получено в ответ
                    //Ответ может прийти в нескольких TCP пакетах
                    int len=TcpReadLength();
                    if(len)
                    {
                        printf("\nПолучено %d",len);
                        //Читаем ответ
                        if(len<sizeof(Json))
                        {
                            //Прочитать ответ в Json
                            TcpRead(Json,sizeof(Json));
                            //Показать ответ Json (погода в текущем городе)
                            ShowCity();
                        }
                        else
                        {
                            puts("\nМного данных...");
                        }
                        //Вывести на дисплей
                        ShowScr();
                        break;
                    }
                }
            }
            else
            {
                 //Ошибка подключения
                 puts("Сервер недоступен\n");
                 printf("result=%d\n",result);
            }
            //Вывести на дисплей
            ShowScr();
            break;
        }
        //Закрыть подключение
        TcpClose();
        //Подключение закрыто
    }
    else printf("\nОшибка открытия...");
    //Вывести на дисплей
    ShowScr();
    //Пауза между запросами 10 сек
    for(int i=0;i<10;i++)
    {
        Pause(1000);
        //Показывать время в нижней части дисплея
        unsigned char hour,min,sec;
        GetTime(&hour,&min,&sec);
        FontColor=COLOR_LIGHTGRAY;
```

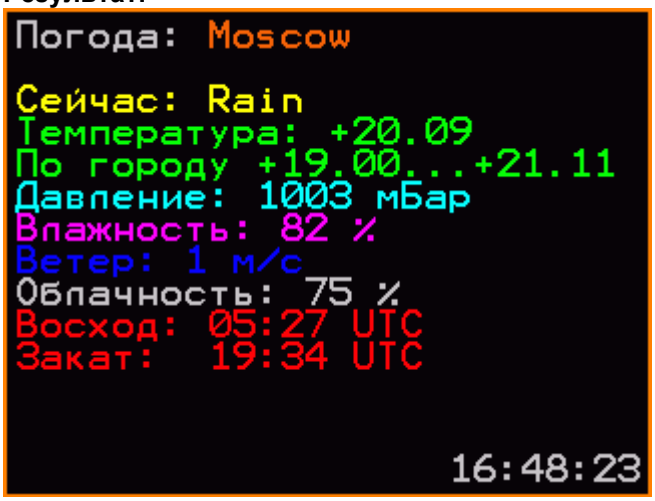
```
        GotoXY(220,220);
        printf("%02d:%02d:%02d",hour,min,sec);
        ShowScr();
    }
    //Следующий город
    if(++CurCity>=Q_CITIES)CurCity=0;
    goto Start;
}
//-----------------------------------------------------------------------------
//Поиск переменной в Json
bool FindJsonVar(char * name,char * val)
{
    val[0]=0;
    char * s=strstr(Json,name);
    if(!s)return false;
    //Поиск ':'
    s+=strlen(name);
    while(*s)
    {
        if(*s==':')break;
        s++;
    }
    if(*s!=':')return false;
    //Найдено ':'
    s++;
    //Если далее ", то пропускаем
    if(*s=='\"')
    {
        s++;
    }
    int pos=0;
    while(*s)
    {
        char v=*s++;
        //Пропускаем кавычки
        if(v=='\"')continue;
        if(v==',' || v=='}')return true;
        //Копировать символ в выходную строку val
        val[pos++]=v;
        val[pos]=0;
        if(pos==31)break;
    }
    return false;
}
//-----------------------------------------------------------------------------
//Получить температуру в сотых долях градуса
//val - строка температуры в градусах кельвина, например: 293.23
int Temperature(char * val)
{
    int t;
    t=atoi(val);
    while(*val)
    {
        if(*val++=='.')
        {
            t*=100;
            t+=atoi(val);
            break;
```

```
        }
    }
    //Перевод в градусы Цельсия
    t-=27315;
    return t;
}
//-----------------------------------------------------------------------------
//Вывод на дисплей температуры в сотых долях градуса
void PrintTemperature(int t)
{
    if(t>0)putchar('+');
    else
    {
        t=-t;
        putchar('-');
    }
    printf("%d.%02d",t/100,t%100);
}
//-----------------------------------------------------------------------------
//Пример ответа сервера:
/*
{"coord":{"lon":37.62,"lat":55.75},"weather":[{"id":500,"main":"Rain",
"description":"light rain","icon":"10d"}],"base":"stations",
"main":{"temp":293.23,"feels_like":291.77,"temp_min":292.15,"temp_max":294.26,
"pressure":1005,"humidity":60},"visibil-
ity":10000,"wind":{"speed":3,"deg":140},
"rain":{"1h":0.37},"clouds":{"all":60},"dt":1598536957,"sys":{"type":1,"id":90
29,
"country":"RU","sunrise":1598495119,"sunset":1598546225},"timezone":10800,
"id":524901,"name":"Moscow","cod":200}
*/
void ShowCity(void)
{
    ClrScr();
    FontColor=COLOR_LIGHTGRAY;
    puts("Погода: ");
    FontColor=COLOR_ORANGE;
    puts(Cities[CurCity]);
    char val[32];
    //Поиск переменных в Json
    //"main":"Rain"
    if(FindJsonVar("main\"",val))
    {
        FontColor=COLOR_YELLOW;
        puts("\n\nСейчас: ");
        puts(val);
    }
    //"temp":293.23
    if(FindJsonVar("temp\"",val))
    {
        FontColor=COLOR_GREEN;
        int t=Temperature(val);
        puts("\nТемпература: ");
        //Вывод на дисплей температуры в сотых долях градуса
        PrintTemperature(t);
    }
    //"temp_min":292.15
    if(FindJsonVar("temp_min\"",val))
```

```
{
    FontColor=COLOR_GREEN;
    int t=Temperature(val);
    puts("\nПо городу ");
    //Вывод на дисплей температуры в сотых долях градуса
    PrintTemperature(t);
}
//"temp_max":294.26
if(FindJsonVar("temp_max\"",val))
{
    FontColor=COLOR_GREEN;
    int t=Temperature(val);
    puts("...");
    //Вывод на дисплей температуры в сотых долях градуса
    PrintTemperature(t);
}
//"pressure":1005
if(FindJsonVar("pressure\"",val))
{
    FontColor=COLOR_CYAN;
    int p=atoi(val);
    printf("\nДавление: %d мБар",p);
}
//"humidity":60
if(FindJsonVar("humidity\"",val))
{
    FontColor=COLOR_MAGENTA;
    int h=atoi(val);
    printf("\nВлажность: %d %%",h);
}
//"speed":3,"deg":140
if(FindJsonVar("speed\"",val))
{
    FontColor=COLOR_BLUE;
    int v=atoi(val);
    printf("\nВетер: %d м/с",v);
}
//"all":60
if(FindJsonVar("all\"",val))
{
    FontColor=COLOR_LIGHTGRAY;
    int v=atoi(val);
    printf("\nОблачность: %d %%",v);
}
//"sunrise":1598495119
if(FindJsonVar("sunrise\"",val))
{
    FontColor=COLOR_RED;
    unsigned int ut=atoi(val);
    unsigned char hour=HourOfTime(ut);
    unsigned char min=MinOfTime(ut);
    printf("\nВосход: %02d:%02d UTC",hour,min);
}
//"sunset":1598546225
if(FindJsonVar("sunset\"",val))
{
    FontColor=COLOR_RED;
    unsigned int ut=atoi(val);
```

```
        unsigned char hour=HourOfTime(ut);
        unsigned char min=MinOfTime(ut);
        printf("\nЗакат:  %02d:%02d UTC",hour,min);
    }
}
//-----------------------------------------------------------------------------
//Пауза миллисекунд
void Pause(unsigned int to)
{
    unsigned int t=GetTickCount();
    while((GetTickCount()-t)<to);
}
//-----------------------------------------------------------------------------
```

Результат:

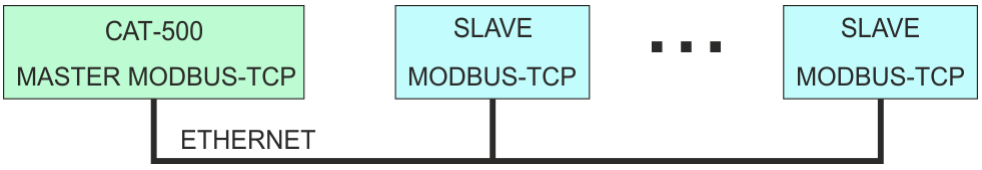


3.13. MODBUS TCP

Контроллер CAT - 500 поддерживает протокол MODBUS TCP. Контроллер может работать как мастер-устройство и как Slave устройство протокола, в том числе может быть и тем и другим одновременно.

3.13.1. MODBUS TCP MASTER

В этом режиме контроллер CAT - 500 работает как мастер-устройство протокола MODBUS TCP.



CAT - 500 устанавливает исходящее TCP соединение используя функцию *TcpOpen* и затем можно читать/записывать объекты MODBUS TCP (регистры, дискретные входы, флаги coils).
Перед использованием функций необходимо включить заголовочный файл *<tcp.h>*.

```
#include <tcp.h>
```

Доступны следующие переменные в режиме MODBUS TCP Мастер:

Переменная	***Описание***
unsigned char ModbusTcpSlave	Переменная, значение которой задаёт опрашиваемый адрес в режиме MODBUS TCP Мастер
unsigned int ModbusTcpTimeout	Переменная, значение которой задаёт таймаут в миллисекундах на выполнение операции чтения/записи в режиме MODBUS TCP Мастер

Доступны следующие функции в режиме MODBUS TCP Мастер:

Функция	***Описание***
*char ModbusTcpOpen (char * adr,unsigned short int port)*	Начать подключение к TCP MODBUS серверу. *char * adr* - строка, которая содержит доменное имя сервера или IP вдрес в текстовом представлении *unsigned short int port* - номер сетевого порта на котором работает TCP MODBUS сервер Таймаут на операцию подключения к TCP MODBUS серверу задаётся в переменной *ModbusTcpTimeout*. Возвращаемое значение: 1 - Успешно началось подключение к TCP MODBUS серверу 0 - Ошибка. Не удалось начать подключение.
char ModbusTcpConnected (void)	Проверить наличие подключения к TCP MODBUS серверу. Возвращаемое значение: 1 - Есть подключение к TCP MODBUS серверу 0 - Нет подключения
void ModbusTcpClose (void)	Завершить сеанс работы с TCP MODBUS сервером. Функция обязательно должна вызываться для завершения текущего сеанса работы с сервером, инициированного вызовом *ModbusTcpOpen*
char ModbusTcpResult (void)	Получить результат выполнения последней заданной операции протокола MODBUS TCP
*char ModbusTcpReadCoils (unsigned short int adr,unsigned short int cnt,unsigned char * coils)*	Чтение флагов (Coils) (Функция 1) Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*. *unsigned short int adr* - адрес первого читаемого флага (начиная с 0) *unsigned short int cnt* - количество читаемых регистров *unsigned char * coils* - указатель на массив куда будут записаны принятые флаги (один флаг - один байт) Возвращаемое значение: 1 - Заданное количество флагов успешно прочитано 0 - Ошибка
*char ModbusTcpReadDiscreteInputs (unsigned short int adr,unsigned short int cnt,unsigned char * inp)*	Чтение дискретных входов (Discrete Inputs) (Функция 2). Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*. *unsigned short int adr* - адрес первого читаемого дискретного входа (начиная с 0) *unsigned short int cnt* - количество читаемых дискретных входов *unsigned char * inp* - указатель на массив куда будут записаны принятые входы (один байт - один вход) Возвращаемое значение:

	1 - Заданное количество дискретных входов успешно прочитано 0 - Ошибка
*char ModbusTcpReadHoldingRegisters (unsigned short int adr,unsigned short int cnt,unsigned short int * reg)*	Прочитать регистры хранения Slave устройства (Функция 3) в соответствии с протоколом TCP MODBUS.. Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*. *unsigned short int adr* - адрес первого читаемого регистра (начиная с 0) *unsigned short int cnt* - количество читаемых регистров *unsigned short int * reg* - указатель на массив куда будут записаны принятые регистры хранения Возвращаемое значение: 1 - Заданное количество регистров успешно прочитано 0 - Ошибка
*char ModbusTcpReadInputRegisters (unsigned short int adr,unsigned short int cnt,unsigned short int * reg)*	Прочитать входные регистры Slave устройства интерфейса MODBUS RTU (Функция 4) в соответствии с протоколом TCP MODBUS. Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*. *unsigned short int adr* - адрес первого читаемого регистра (начиная с 0) *unsigned short int cnt* - количество читаемых регистров *unsigned short int * reg* - указатель на массив куда будут записаны принятые входные регистры Возвращаемое значение: 1 - Заданное количество регистров успешно прочитано 0 - Ошибка
char ModbusTcpWriteSingleCoil (unsigned short int adr,unsigned short int value)	Запись одного флага (Coil) (Функция 5)в соответствии с протоколом TCP MODBUS. Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*. *unsigned short int adr* - адрес записываемого флага (начиная с 0) *unsigned short int value* - записываемое значение Возвращаемое значение: 1 - Успешно записано 0 - Ошибка
char ModbusTcpWriteHoldingRegister (unsigned short int adr,unsigned short int value)	Запись значения одного регистра хранения Slave устройства (Функция 6) в соответствии с протоколом TCP MODBUS. Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*. *unsigned short int adr* - адрес записываемого регистра хранения (начиная с 0) *unsigned short int value* - записываемое значение Возвращаемое значение: 1 - Успешно записано 0 - Ошибка

*char ModbusTcpWriteRegisters (unsigned short int adr,unsigned short int cnt,unsigned short int * reg)*	Запись нескольких регистров хранения Slave устройства (Функция 16) в соответствии с протоколом TCP MODBUS. Адрес читаемого устройства задаётся в переменной ModbusTcpSlave. Таймаут выполняемой операции задаётся в переменной ModbusTcpTimeout. unsigned short int adr - начальный адрес записываемых регистров (начиная с 0) unsigned short int cnt - количество записываемых регистров unsigned short int * reg - указатель на массив записываемых регистров Возвращаемое значение: 1 - Успешно записано 0 - Ошибка
*char ModbusWriteCoils (unsigned short int adr,unsigned short int cnt,unsigned char * coils)*	Запись нескольких флагов (Coils) (Функция 15) в соответствии с протоколом TCP MODBUS. Адрес читаемого устройства задаётся в переменной ModbusTcpSlave. Таймаут выполняемой операции задаётся в переменной ModbusTcpTimeout. unsigned short int adr - адрес записываемого флага (начиная с 0) unsigned short int cnt - количество записываемых флагов unsigned short int coils - записываемые значения флагов (один флаг - один байт) Возвращаемое значение: 1 - Успешно записано 0 - Ошибка

3.13.1.1. MODBUSTCPSLAVE

```
#include <tcp.h>

unsigned char ModbusTcpSlave;
```

Переменная, значение которой задаёт опрашиваемый адрес в режиме MODBUS TCP Мастер.

3.13.1.2. MODBUSTCPTIMEOUT

```
#include <tcp.h>

unsigned int ModbusTcpTimeout;
```

Переменная, значение которой задаёт таймаут в миллисекундах на выполнение операции чтения/записи в режиме MODBUS TCP Мастер и на подключение к серверу.

3.13.1.3. MODBUSTCPOPEN

```
#include <tcp.h>

char ModbusTcpOpen(char * adr,unsigned short int port);
```

Начать подключение к TCP MODBUS серверу.

*char * adr* - строка, которая содержит доменное имя сервера или IP адрес в текстовом представлении

unsigned short int port - номер сетевого порта на котором работает TCP MODBUS сервер

Таймаут на операцию подключения к TCP MODBUS серверу задаётся в переменной *ModbusTcp-Timeout*.

Возвращаемое значение:
1 - Успешно началось подключение к TCP MODBUS серверу
0 - Ошибка. Не удалось начать подключение.

Успешный возврат ещё не обозначает, что уже установлено соединение с сервером.

Для проверки наличия соединения следует использовать функцию *ModbusTcpResult* или функцию *ModbusTcpConnected*
Только после того как соединение с сервером установлено можно выполнять команды чтения/записи регистров и других MODBUS объектов.

В примере далее читается второй контроллер CAT-500, работающий в режиме регулятора отопления.

Пример:

```
#include <sat500.h>
#include <serial.h>
#include <tcp.h>

void main(void)
{
    ClrScr();                  //Очистить экран
    BigFont=0;                 //Обычный шрифт 12x16 (не крупный)
    //Начать подключение к TCP MODBUS серверу
    //На подключение не более 2 секунд
    ModbusTcpTimeout=2000;
    if(ModbusTcpOpen("192.168.1.234",502))
    {
      printf("Подключение...\n");
      ShowScr();                //Вывести на дисплей
      //Ждём подключения
      char result;
      do
      {
          result=ModbusTcpResult();
      }
      while(result<0);
      if(result==0)
      {
           //Успешно подключились
           printf("Успешно...\n");
           ShowScr();                //Вывести на дисплей
           //Регистры, куда будет помещён ответ
           short regs[8];
           if(ModbusTcpReadHoldingRegisters(0,8,regs))
           {
              do
              {
                  result=ModbusResult();
              }
              while(result<0);
              if(result==0)
```

```
                {
                    //Просмотр полученных регистров
                    for(int i=0;i<8;i++)
                    printf("Reg[%d]=%x    \n",i,regs[i]&0xFFFF);
                }
            }
            ShowScr();   //Вывести на дисплей
            //Завершить сеанс работы с сервером
            ModbusTcpClose();
        }
        else
        {
            //Не удалось подключиться к серверу
            printf("Ошибка=%d  ",result);
        }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

3.13.1.4. MODBUSTCPCONNECTED

```
#include <tcp.h>

char ModbusTcpConnected(void);
```

Проверить наличие подключения к TCP MODBUS серверу.

Возвращаемое значение:
1 - Есть подключение к TCP MODBUS серверу
0 - Нет подключения

Функцию *ModbusTcpConnected* необходимо вызывать для проверки наличия подключения к серверу после вызова функции начала подключения к серверу *ModbusTcpOpen*

3.13.1.5. MODBUSTCPCLOSE

```
#include <tcp.h>

void ModbusTcpClose(void);
```

Завершить сеанс работы с TCP MODBUS сервером.
Функция обязательно должна вызываться для завершения текущего сеанса работы с сервером, инициированного вызовом *ModbusTcpOpen*

3.13.1.6. MODBUSTCPRESULT

```
#include <tcp.h>

char ModbusTcpResult(void);
```

Получить результат выполнения последней заданной операции протокола MODBUS TCP

Возможные возвращаемые значения показаны в таблице ниже.

Значение	*Описание*
-1	Операция ещё выполняется

0	Последняя операция выполнена успешно
*1**	Неправильный номер функции для заданного slave устройства
*2**	Неправильный адрес для заданного slave устройства
*3**	Неправильные данные для заданного slave устройства
*4**	Произошла ошибка в slave устройстве при выполнении заданной функции
*5**	Подтверждение (Acknowledge) - для выполнения заданной операции требуется больше времени
*6**	Slave устройство занято и сейчас не может выполнить запрошенную функцию (BUSY)
*7**	Отказ выполнения функции slave устройством (NAK)
*8**	Ошибка чётности в памяти slave
*10**	Проблема шлюза (Gateway paths not available)
*11**	Сообщение шлюза - устройство не отвечает (The targeted device failed to respond)
12	Общая ошибка
13	Не удалось послать
14	Неправильный байт адреса slave в ответе
15	Нет ответа
16	Ошибка CRC в ответе (только MODBUS RTU)
17	Мало байтов в ответе
18	Неправильный байт функции в ответе
19	Неправильный режим последовательного порта
20	Неправильный Id в ответе
21	Ошибка в TCP ответе
22	Не задан адрес сервера
23	Не удалось получить IP адрес сервера
24	Ошибка при выполнении DNS запроса
25	Ошибка при выполнении подключения
26	Сервер разорвал подключение
27	Не удалось подключиться к серверу
28	Не удалось начать подключение к серверу
29	Ошибка записи в TCP соединение
30	Нет ETHERNET подключения
31	TCP соединение разорвано

* - стандартный код исключительной ошибки протокола MODBUS RTU (EXEPTION)

В примере далее читается второй контроллер CAT-500, работающий в режиме регулятора отопления и на дисплей выводятся прочитанные регистры.

Пример:

```
#include <sat500.h>
#include <tcp.h>

void main(void)
{
    ClrScr();                  //Очистить экран
    BigFont=0;                 //Обычный шрифт 12x16 (не крупный)
    //Начать подключение к TCP MODBUS серверу
    //На подключение не более 2 секунд
    ModbusTcpTimeout=2000;
    if(ModbusTcpOpen("192.168.1.8",502))
    {
      printf("Подключение...\n");
      ShowScr();                //Вывести на дисплей
      //Ждём подключения
```

```
      char result;
      do
      {
          result=ModbusTcpResult();
      }
      while(result<0);
      if(result==0)
      {
           //Успешно подключились
           printf("Успешно...\n");
           ShowScr();              //Вывести на дисплей
           //Регистры, куда будет помещён ответ
           short regs[8];
           ModbusTcpSlave=1;
           if(ModbusTcpReadHoldingRegisters(0,8,regs))
           {
              do
              {
                  result=ModbusTcpResult();
              }
              while(result<0);
              if(result==0)
              {
                  //Просмотр полученных регистров
                  for(int i=0;i<8;i++)
                  printf("Reg[%d]=%x    \n",i,regs[i]&0xFFFF);
              }
              else
              {
                printf("result=%d  ",result);
              }
           }
           else printf("Ошибка запуска чтения...\n");
           ShowScr();   //Вывести на дисплей
           //Завершить сеанс работы с сервером
           ModbusTcpClose();
      }
      else
      {
          //Не удалось подключиться к серверу
          printf("Ошибка=%d  ",result);
      }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:

3.13.1.7. MODBUSTCPREADCOILS

```
#include <tcp.h>

char ModbusTcpReadCoils(unsigned short int adr,unsigned short int cnt,unsigned
char * coils);
```

Чтение флагов (Coils) (Функция 1) в соответствии с протоколом TCP MODBUS.
Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*.

unsigned short int adr - адрес первого читаемого флага (начиная с 0)
unsigned short int cnt - количество читаемых регистров
*unsigned char * coils* - указатель на массив куда будут записаны принятые флаги

ВНИМАНИЕ: Каждый байт массива *coils* является значением одного флага *coil*.

Возвращаемое значение:
1 - Заданное количество флагов успешно прочитано
0 - Ошибка

Наиболее часто в качестве coils используют дискретные выходные сигналы.

В примере далее читается второй контроллер CAT-500, работающий в режиме регулятора отопления и на дисплей выводятся прочитанные флаги (состояние выходов) DO1..DO11.

Пример:

```
#include <sat500.h>
#include <tcp.h>

void main(void)
{
    ClrScr();                //Очистить экран
    BigFont=0;               //Обычный шрифт 12x16 (не крупный)
    printf("Пример ModbusTcpReadCoils\n");
    //Начать подключение к TCP MODBUS серверу
    //На подключение не более 2 секунд
    ModbusTcpTimeout=2000;
    if(ModbusTcpOpen("192.168.1.8",502))
    {
      printf("Подключение...\n");
      ShowScr();              //Вывести на дисплей
      //Ждём подключения
      char result;
      do
      {
          result=ModbusTcpResult();
      }
      while(result<0);
      if(result==0)
      {
           //Успешно подключились
           printf("Успешно...\n");
           ShowScr();              //Вывести на дисплей
           ModbusTcpSlave=1;
           //В CAT-500 в режиме регулятор доступны флаги 256-266
           //Флаги, куда будет помещён ответ
           unsigned char coils[11];
           if(ModbusTcpReadCoils(256,11,coils))
           {
              do
              {
                  result=ModbusTcpResult();
              }
```

```
                while(result<0);
                if(result==0)
                {
                    //Просмотр полученных флагов
                    for(int i=0;i<11;i++)
                    printf("DO%d=coil[%d]=%x\n",i+1,i,coils[i]);
                }
                else
                {
                    printf("result=%d\n",result);
                }
            }
            else
            {
                printf("Ошибка ModbusTcpReadCoils\n");
            }
            ShowScr();   //Вывести на дисплей
            //Завершить сеанс работы с сервером
            ModbusTcpClose();
        }
        else
        {
            //Не удалось подключиться к серверу
            printf("Ошибка=%d  ",result);
        }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:

Если выполнить запрос флагов с адреса 0:

```
...
            if(ModbusTcpReadCoils(0,11,coils))
...
```

То результат будет:

Это соответствует исключительной ошибке MODBUS "Неправильный адрес для заданного slave устройства" (см. перечень ошибок в разделе *ModbusTcpResult*).

3.13.1.8. MODBUSTCPREADDISCRETEINPUTS

```
#include <tcp.h>

char ModbusTcpReadDiscreteInputs(unsigned short int adr,unsigned short int
cnt,unsigned char * inp);
```

Чтение дискретных входов (Discrete Inputs) (Функция 2) в соответствии с протоколом TCP MODBUS.
Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*.

unsigned short int adr - адрес первого читаемого дискретного входа (начиная с 0)
unsigned short int cnt - количество читаемых дискретных входов
*unsigned char * inp* - указатель на массив куда будут записаны принятые входы

ВНИМАНИЕ: Каждый байт массива *inp* является значением одного дискретного входа.

Возвращаемое значение:
1 - Заданное количество дискретных входов успешно прочитано
0 - Ошибка

В примере далее читается второй контроллер CAT-500, работающий в режиме регулятора отопления и на дисплей выводятся прочитанные дискретные входы (состояние цифровых входов) DI1..DI10.

Пример:

```
#include <sat500.h>
#include <tcp.h>

void main(void)
{
    ClrScr();                //Очистить экран
    BigFont=0;               //Обычный шрифт 12x16 (не крупный)
    printf("Чтение DiscreteInputs\n");
    //Начать подключение к TCP MODBUS серверу
    //На подключение не более 2 секунд
    ModbusTcpTimeout=2000;
    if(ModbusTcpOpen("192.168.1.8",502))
    {
      printf("Подключение...\n");
      ShowScr();              //Вывести на дисплей
      //Ждём подключения
      char result;
      do
      {
          result=ModbusTcpResult();
      }
      while(result<0);
      if(result==0)
      {
           //Успешно подключились
           printf("Успешно...\n");
           ShowScr();             //Вывести на дисплей
           ModbusTcpSlave=1;
           //В CAT-500 в режиме регулятор доступны дискретные входы 256-318
           //Байты (входы), куда будет помещён ответ
           unsigned char inp[10];
           if(ModbusTcpReadDiscreteInputs(256,10,inp))
           {
              do
              {
                  result=ModbusTcpResult();
              }
              while(result<0);
              if(result==0)
              {
                  //Просмотр полученных флагов
                  for(int i=0;i<10;i++)
```

```
                printf("DI%d=inp[%d]=%x\n",i+1,i,inp[i]);
            }
            else
            {
                printf("result=%d\n",result);
            }
        }
        else
        {
            printf("Ошибка ModbusTcpReadDiscreteInputs\n");
        }
        ShowScr();   //Вывести на дисплей
        //Завершить сеанс работы с сервером
        ModbusTcpClose();
    }
    else
    {
        //Не удалось подключиться к серверу
        printf("Ошибка=%d  ",result);
    }
}
else printf("Ошибка открытия...\n");
ShowScr();                  //Вывести на дисплей
while(1);                   //Бесконечная пауза
}
```

Результат:

Если выполнить запрос дискретных входов с адреса 0:

```
...
        if(ModbusTcpReadDiscreteInputs(0,11,coils))
...
```

То результат будет:

3.13.1.9. MODBUSTCPREADHOLDINGREGISTERS

```
#include <serial.h>

char ModbusTcpReadHoldingRegisters(unsigned short int adr,unsigned short int
cnt,unsigned short int * reg);
```

Прочитать регистры хранения Slave устройства (Функция 3) в соответствии с протоколом TCP MODBUS.
Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*.

unsigned short int adr - адрес первого читаемого регистра (начиная с 0)
unsigned short int cnt - количество читаемых регистров
*unsigned short int * reg* - указатель на массив куда будут записаны принятые регистры хранения

Возвращаемое значение:
1 - Заданное количество регистров успешно прочитано
0 - Ошибка

Пример:

```
#include <sat500.h>
#include <tcp.h>

void main(void)
{
    ClrScr();                 //Очистить экран
    BigFont=0;                //Обычный шрифт 12x16 (не крупный)
    //Начать подключение к TCP MODBUS серверу
    //На подключение не более 2 секунд
    ModbusTcpTimeout=2000;
    if(ModbusTcpOpen("192.168.1.33",502))
    {
      printf("Подключение...\n");
      ShowScr();               //Вывести на дисплей
      //Ждём подключения
      char result;
      do
      {
          result=ModbusTcpResult();
      }
      while(result<0);
      if(result==0)
      {
           //Успешно подключились
           printf("Успешно...\n");
           ShowScr();               //Вывести на дисплей
           //Регистры, куда будет помещён ответ
           short regs[8];
           ModbusTcpSlave=1;
           if(ModbusTcpReadHoldingRegisters(0,8,regs))
           {
              do
              {
                  result=ModbusTcpResult();
              }
              while(result<0);
              if(result==0)
              {
                  //Просмотр полученных регистров
                  for(int i=0;i<8;i++)
                  printf("Reg[%d]=%x    \n",i,regs[i]&0xFFFF);
              }
           }
           ShowScr();   //Вывести на дисплей
           //Завершить сеанс работы с сервером
           ModbusTcpClose();
      }
      else
      {
          //Не удалось подключиться к серверу
          printf("Ошибка=%d  ",result);
      }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
```

```
}
```

Результат:

3.13.1.10. MODBUSTCPREADINPUTREGISTERS

```
#include <tcp.h>

char ModbusTcpReadInputRegisters(unsigned short int adr,unsigned short int
cnt,unsigned short int * reg);
```

Прочитать входные регистры Slave устройства интерфейса MODBUS RTU (Функция 4) в соответствии с протоколом TCP MODBUS.
Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*.

unsigned short int adr - адрес первого читаемого регистра (начиная с 0)
unsigned short int cnt - количество читаемых регистров
*unsigned short int * reg* - указатель на массив куда будут записаны принятые входные регистры

Возвращаемое значение:
1 - Заданное количество регистров успешно прочитано
0 - Ошибка

Пример:

```
#include <sat500.h>
#include <tcp.h>

void main(void)
{
    ClrScr();                //Очистить экран
    BigFont=0;               //Обычный шрифт 12x16 (не крупный)
    //Начать подключение к TCP MODBUS серверу
    //На подключение не более 2 секунд
    ModbusTcpTimeout=2000;
    if(ModbusTcpOpen("192.168.1.33",502))
    {
      printf("Подключение...\n");
      ShowScr();              //Вывести на дисплей
      //Ждём подключения
      char result;
      do
      {
          result=ModbusTcpResult();
      }
      while(result<0);
      if(result==0)
      {
           //Успешно подключились
           printf("Успешно...\n");
           ShowScr();              //Вывести на дисплей
           //Регистры, куда будет помещён ответ
           short regs[8];
           ModbusTcpSlave=1;
           if(ModbusTcpReadInputRegisters(0,8,regs))
           {
```

```
                do
                {
                    result=ModbusTcpResult();
                }
                while(result<0);
                if(result==0)
                {
                    //Просмотр полученных регистров
                    for(int i=0;i<8;i++)
                    printf("Reg[%d]=%x    \n",i,regs[i]&0xFFFF);
                }
            }
            ShowScr();   //Вывести на дисплей
            //Завершить сеанс работы с сервером
            ModbusTcpClose();
        }
        else
        {
            //Не удалось подключиться к серверу
            printf("Ошибка=%d  ",result);
        }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:

3.13.1.11. MODBUSTCPWRITESINGLECOIL

```
#include <tcp.h>

char ModbusTcpWriteSingleCoil(unsigned short int adr,unsigned short int
value);
```

Запись одного флага (Coil) (Функция 5) в соответствии с протоколом TCP MODBUS.
Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*.

unsigned short int adr - адрес записываемого флага (начиная с 0)
unsigned short int value - записываемое значение

Возвращаемое значение:
1 - Успешно записано
0 - Ошибка

Пример:

```
#include <sat500.h>
#include <tcp.h>

void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Начать подключение к TCP MODBUS серверу
```

```
    //На подключение не более 2 секунд
    ModbusTcpTimeout=2000;
    if(ModbusTcpOpen("192.168.1.33",502))
    {
      printf("Подключение...\n");
      ShowScr();              //Вывести на дисплей
      //Ждём подключения
      char result;
      do
      {
          result=ModbusTcpResult();
      }
      while(result<0);
      if(result==0)
      {
           //Успешно подключились
           printf("Успешно...\n");
           ShowScr();              //Вывести на дисплей
           ModbusTcpSlave=1;
           if(ModbusTcpWriteSingleCoil(0,1))
           {
              do
              {
                  result=ModbusTcpResult();
              }
              while(result<0);
              if(result==0)
              {
                  //Успешно записано
                  printf("Успешно записано\n");
              }
           }
           ShowScr();   //Вывести на дисплей
           //Завершить сеанс работы с сервером
           ModbusTcpClose();
      }
      else
      {
          //Не удалось подключиться к серверу
          printf("Ошибка=%d  ",result);
      }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                 //Вывести на дисплей
    while(1);                  //Бесконечная пауза
}
```

Результат:

3.13.1.12. MODBUSTCPWRITEHOLDINGREGISTER

```
#include <tcp.h>

char ModbusTcpWriteHoldingRegister(unsigned short int adr,unsigned short int
value);
```

Запись значения одного регистра хранения Slave устройства (Функция 6) в соответствии с протоколом TCP MODBUS.
Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*.

unsigned short int adr - адрес записываемого регистра хранения (начиная с 0)
unsigned short int value - записываемое значение

Возвращаемое значение:
1 - Успешно записано
0 - Ошибка

Пример:

```
#include <sat500.h>
#include <tcp.h>

void main(void)
{
    ClrScr();                //Очистить экран
    BigFont=0;               //Обычный шрифт 12x16 (не крупный)
    //Начать подключение к TCP MODBUS серверу
    //На подключение не более 2 секунд
    ModbusTcpTimeout=2000;
    if(ModbusTcpOpen("192.168.1.33",502))
    {
      printf("Подключение...\n");
      ShowScr();              //Вывести на дисплей
      //Ждём подключения
      char result;
      do
      {
          result=ModbusTcpResult();
      }
      while(result<0);
      if(result==0)
      {
           //Успешно подключились
           printf("Успешно...\n");
           ShowScr();              //Вывести на дисплей
           ModbusTcpSlave=1;
           if(ModbusTcpWriteHoldingRegister(1,1234))
           {
              do
              {
                  result=ModbusTcpResult();
              }
              while(result<0);
              if(result==0)
              {
                  //Успешно записано
                  printf("Успешно записан\nHolding Register\n");
              }
           }
           ShowScr();   //Вывести на дисплей
           //Завершить сеанс работы с сервером
           ModbusTcpClose();
      }
```

```
      else
      {
          //Не удалось подключиться к серверу
          printf("Ошибка=%d  ",result);
      }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:

3.13.1.13. MODBUSTCPWRITEREGISTERS

```
#include <tcp.h>

char ModbusTcpWriteRegisters(unsigned short int adr,unsigned short int cnt,un-
signed short int * reg);
```

Запись нескольких регистров хранения Slave устройства (Функция 16) в соответствии с протоколом TCP MODBUS.
Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*.

unsigned short int adr - начальный адрес записываемых регистров (начиная с 0)
unsigned short int cnt - количество записываемых регистров
*unsigned short int * reg* - указатель на массив записываемых регистров

Возвращаемое значение:
1 - Успешно записано
0 - Ошибка

Пример:

```
#include <sat500.h>
#include <tcp.h>

void main(void)
{
    ClrScr();                 //Очистить экран
    BigFont=0;                //Обычный шрифт 12x16 (не крупный)
    //Начать подключение к TCP MODBUS серверу
    //На подключение не более 2 секунд
    ModbusTcpTimeout=2000;
    if(ModbusTcpOpen("192.168.1.33",502))
    {
      printf("Подключение...\n");
      ShowScr();              //Вывести на дисплей
      //Ждём подключения
      char result;
      do
      {
          result=ModbusTcpResult();
      }
      while(result<0);
      if(result==0)
```

```
        {
            //Успешно подключились
            printf("Успешно...\n");
            ShowScr();              //Вывести на дисплей
            ModbusTcpSlave=1;
            //Записываемые регистры
            unsigned short int reg[3];
            reg[0]=111;
            reg[1]=2222;
            reg[2]=33333;
            if(ModbusTcpWriteRegisters(1,3,reg))
            {
               do
               {
                   result=ModbusTcpResult();
               }
               while(result<0);
               if(result==0)
               {
                   //Успешно записано
                   printf("Успешно записаны\nHolding Registers\n");
               }
            }
            ShowScr();   //Вывести на дисплей
            //Завершить сеанс работы с сервером
            ModbusTcpClose();
        }
        else
        {
            //Не удалось подключиться к серверу
            printf("Ошибка=%d  ",result);
        }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:

3.13.1.14. MODBUSTCPWRITECOILS

```
#include <tcp.h>

char ModbusTcpWriteCoils(unsigned short int adr,unsigned short int cnt,un-
signed char * coils);
```

Запись нескольких флагов (Coils) (Функция 15) в соответствии с протоколом TCP MODBUS.
Адрес читаемого устройства задаётся в переменной *ModbusTcpSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTcpTimeout*.

unsigned short int adr - адрес записываемого флага (начиная с 0)
unsigned short int cnt - количество записываемых флагов
unsigned short int coils - записываемые значения флагов

ВНИМАНИЕ: Каждый байт массива *coils* является значением одного флага *coil*.

Возвращаемое значение:
1 - Успешно записано
0 - Ошибка

Пример:

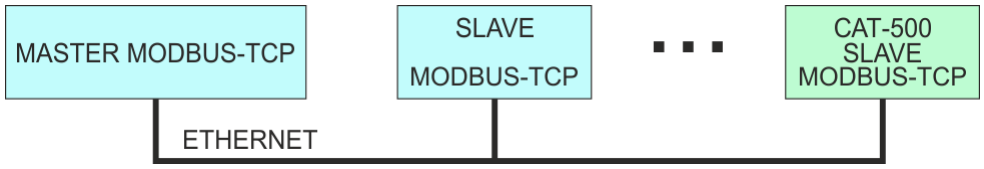
```
#include <sat500.h>
#include <tcp.h>

void main(void)
{
    ClrScr();                //Очистить экран
    BigFont=0;               //Обычный шрифт 12x16 (не крупный)
    //Начать подключение к TCP MODBUS серверу
    //На подключение не более 2 секунд
    ModbusTcpTimeout=2000;
    if(ModbusTcpOpen("192.168.1.33",502))
    {
      printf("Подключение...\n");
      ShowScr();              //Вывести на дисплей
      //Ждём подключения
      char result;
      do
      {
          result=ModbusTcpResult();
      }
      while(result<0);
      if(result==0)
      {
           //Успешно подключились
           printf("Успешно...\n");
           ShowScr();              //Вывести на дисплей
           ModbusTcpSlave=1;
           //Записываемые регистры
           unsigned char coils[4];
           coils[0]=1;
           coils[1]=1;
           coils[2]=0;
           coils[3]=0;
           //Записать 4 coils с адреса 3
           if(ModbusTcpWriteCoils(3,4,coils))
           {
              do
              {
                  result=ModbusTcpResult();
              }
              while(result<0);
              if(result==0)
              {
                  //Успешно записано
                  printf("Успешно записаны\ncoils\n");
              }
           }
           ShowScr();   //Вывести на дисплей
           //Завершить сеанс работы с сервером
           ModbusTcpClose();
      }
      else
```

```
        {
            //Не удалось подключиться к серверу
            printf("Ошибка=%d  ",result);
        }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:

3.13.2. MODBUS TCP SLAVE

В этом режиме контроллер CAT - 500 работает как slave-устройство протокола MODBUS TCP.



CAT - 500 принимает входящие TCP соединения на порту 502 и отвечает на команды работы с регистрами хранения и входными регистрами.
Перед использованием функций необходимо включить заголовочный файл *<tcp.h>*.

Доступны следующие переменные в режиме SLAVE устройство протокола MODBUS TCP:

Переменная	***Описание***
unsigned int ModbusTcpCount	Переменная, которая содержит счетчик корректных запросов TCP MODBUS Slave, на которые был выдан ответ

Доступны следующие функции в режиме SLAVE устройство протокола MODBUS TCP:

Функция	***Описание***
*unsigned char StartModbusTcpSlave (unsigned char slave,unsigned short int * hld,unsigned short int hldcnt,unsigned short int * inp,unsigned short int inpcnt)*	Функция *StartModbusTcpSlave* разрешает режим работы TCP MODBUS Slave контроллера CAT-500. В качестве параметров её передаются количество и адреса регистров хранения, и регистров, количество и адреса входных регистров. Адрес Slave устройства также задается в параметре *slave*. *unsigned char slave* - Slave адрес контроллера CAT-500 в режиме работы TCP MODBUS Slave. Если задано нулевое значение то контроллер отвесает на любое значение slave в запросе *unsigned short int * hld* - указатель на первый регистр хранения, который будет иметь MODBUS адрес 0 *unsigned short int hldcnt* - количество регистров хранения, расположенных в памяти начиная с адреса *hld* *unsigned short int * inp* - - указатель на первый входной регистр, который будет иметь MODBUS адрес 0 *unsigned short int inpcnt* - количество входных регистров, расположенных начиная с адреса *inp* Возвращаемое значение: 1 - Успешно включён TCP MODBUS SLAVE режим 0 - Ошибка

void StopModbusTcpSlave (void)	Функция *StopModbusTcpSlave* завершает режима работы TCP MODBUS Slave контроллера CAT-500

Вне зависимости от того был ли включен режим MODBUS TCP Slave функцией *StartModbusTcpSlave* контроллер CAT-500 всегда является устройством MODBUS TCP Slave и отвечает на обращение к следующим регистрам хранения (функция 3):

Адрес регистра хранения	***Регистров***	***Тип***	***Описание***
0x8000	1	WORD	Побитовое состояние дискретные выходов (0/1 Выкл/Вкл): DO1 ÷ DO11 Бит Выход 0 DO1 1 DO2 2 DO3 3 DO4 4 DO5 5 DO6 6 DO7 7 DO8 8 DO9 9 DO10 10 DO11
0x8001	1*	WORD	Побитовое состояние дискретные входов (0/1 Выкл/Вкл): DI1 ÷ DI10 Бит Вход 0 DI1 1 DI2 2 DI3 3 DI4 4 DI5 5 DI6 6 DI7 7 DI8 8 DI9 9 DI10
0x8002	2*	UINT	Значение измерительного канала T1, °C
0x8004	2*	UINT	Значение измерительного канала T2, °C
0x8006	2*	UINT	Значение измерительного канала T3, °C
0x8008	2*	UINT	Значение измерительного канала T4, °C
0x800A	2*	UINT	Значение измерительного канала T5, °C
0x800C	2*	UINT	Значение измерительного канала AI1
0x800E	2*	UINT	Значение измерительного канала AI2
0x8010	2*	FLOAT	Значение температуры T1 в формате float IEEE 754 (используйте *GetModbusFloat*)
0x8012	2*	FLOAT	Значение температуры T2 в формате float IEEE 754
0x8014	2*	FLOAT	Значение температуры T3 в формате float IEEE 754
0x8016	2*	FLOAT	Значение температуры T4 в формате float IEEE 754
0x8018	2*	FLOAT	Значение температуры T5 в формате float IEEE 754
0x801A	2*	FLOAT	Значение температуры AI1 в формате float IEEE 754
0x801C	2*	FLOAT	Значение температурыAI2 в формате float IEEE 754
0x801E	1*	WORD	Ошибки АЦП
0x801F	2	FLOAT	Значение выхода AO1
0x8021	2	FLOAT	Значение выхода AO2

0x8023	1	DATE	Текущая дата: День (1 байт) и месяц (1 байт)
0x8024	1		Текущая дата: Год
0x8025	1	TIME	Текущее время: Секунды (1 байт) и минуты (1 байт)
0x8026	1		Текущее время: Часы
0x8027	2*	FLOAT	Значение измерительного канала напряжения батареи
0x840C	24	STRING	Строка "Комментарий"

* - Только чтение

3.13.2.1. STARTMODBUSTCPSLAVE

```
#include <tcp.h>

unsigned char StartModbusTcpSlave(unsigned char slave,unsigned short int *
hld,unsigned short int hldcnt,unsigned short int * inp,unsigned short int
inpcnt);
```

Функция *StartModbusTcpSlave* разрешает режим работы TCP MODBUS Slave контроллера CAT-500. В качестве параметров её передаются количество и адреса регистров хранения, и регистров, количество и адреса входных регистров.
Адрес Slave устройства также задается в параметре *slave*.

unsigned char slave - Slave адрес контроллера CAT-500 в режиме работы TCP MODBUS Slave. Если задано нулевое значение то контроллер отвесает на любое значение slave в запросе
*unsigned short int * hld* - указатель на первый регистр хранения, который будет иметь MODBUS адрес 0
unsigned short int hldcnt - количество регистров хранения, расположенных в памяти начиная с адреса *hld*
*unsigned short int * inp* - - указатель на первый входной регистр, который будет иметь MODBUS адрес 0
unsigned short int inpcnt - количество входных регистров, расположенных начиная с адреса *inp*

Возвращаемое значение:
1 - Успешно включён TCP MODBUS SLAVE режим
0 - Ошибка

Пример:

```
#include <sat500.h>
#include <tcp.h>

short int reg[32];

void main(void)
{
    ClrScr();
    printf("TCP MODBUS SRV");
    ShowScr();
    StartModbusTcpSlave(1,reg,32,reg,32);
    while(1)
    {
        GotoXY(4,40);
        printf("%5d",reg[0]);
        GotoXY(4,68);
```

```
        printf("%5d",reg[1]);
        GotoXY(4,96);
        printf("%5d",ModbusTcpCount);
        ShowScr();
        unsigned t=GetTickCount();
        while((GetTickCount()-t)<200);
    }
}
```

Результат:

3.13.2.2. STOPMODBUSTCPSLAVE

```
#include <tcp.h>

void StopModbusTcpSlave(void);
```

Функция *StopModbusTcpSlave* завершает режима работы TCP MODBUS Slave контроллера CAT-500.

Пример:

```
#include <sat500.h>
#include <tcp.h>

short int reg[32];

void main(void)
{
    ClrScr();
    printf("TCP MODBUS SRV");
    ShowScr();
    StartModbusTcpSlave(1,reg,32,reg,32);
    while(1)
    {
        GotoXY(4,40);
        printf("%5d",reg[0]);
        GotoXY(4,68);
        printf("%5d",reg[1]);
        GotoXY(4,96);
        printf("%5d",ModbusTcpCount);
        ShowScr();
        if(ModbusTcpCount>=100)break;
        unsigned t=GetTickCount();
        while((GetTickCount()-t)<200);
    }
    StopModbusTcpSlave();
    printf("\nОстановлен...");
    ShowScr();
    while(1)
    {
    }
}
```

Результат:

3.13.2.3. MODBUSTCPCOUNT

```
#include <tcp.h>

unsigned int ModbusTcpCount;
```

Переменная, которая содержит счетчик корректных запросов TCP MODBUS Slave, на которые был выдан ответ.
Для очистки переменной можно записать в неё значение:

```
ModbusTcpCount=0;
```

3.14. ИНТЕРФЕЙС RS-485

Заголовочные файлы *<serial.h>* и *<tcp.h>* содержит функции работы с последовательным интерфейсом RS-485.
Имеются несколько групп функций:
1. Функции работы с последовательным интерфейсом без привязки к какому-либо протоколу.
2. Функции работы в режиме MODBUS RTU Мастер
3. Функции работы в режиме MODBUS RTU Slave
4. Функции работы в режиме MODBUS TCP Мастер
5. Функции работы в режиме MODBUS TCP Slave

3.14.1. БАЗОВЫЕ ФУНКЦИИ И ПЕРЕМЕННЫЕ

Базовые функции предназначены для работы с последовательным портом без какого-либо протокола.
Перед использованием функций необходимо включить заголовочный файл.

```
#include <serial.h>
```

Доступны следующие функции:

Функция	***Описание***
char SerialOpen(int baudrate,char parity,char stopbits)	Открыть последовательный порт RS-485 *int baudrate* - Скорость. Возможные значения: 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 56000, 57600, 115200 *char parity* - Чётность. Возможные значения: 0 - NONE,1 - ODD,2 - EVEN,3 - MARK,4 - SPACE *char stopbits* - Количество стоповых битов. Возможные значения: 1 - один стоп-бит, 2 - два Возвращаемое значение: 1 - Успешно открыт 0 - Ошибка
void SerialClose(void)	Закрыть последовательный порт RS-485
*char SerialWrite(char * data,short int len)*	Записать данные в последовательный порт *char * data* - массив данных для выдачи в последовательный порт *short int len* - количество записываемых данных Возвращаемое значение: 1 - Успешно записано 0 - Ошибка

short int SerialReadLength(void)	Получить количество байтов, доступных для чтения из приёмного буфера последовательного порта Возвращаемое значение: Доступное количество байтов
*short int SerialRead(char * data,short int size)*	Прочитать данные из приёмного буфера последовательного порта *char * data* - массив данных для приёма из последовательного порта *short int len* - количество читаемых данных Возвращаемое значение: количество прочитанных данных
char SerialIsOpen(void)	Проверить открыт ли последовательный порт RS-485 Возвращаемое значение: 1 - Открыт 0 - Закрыт

3.14.1.1. SERIALOPEN

```
#include <serial.h>

char SerialOpen(int baudrate,char parity,char stopbits);
```

Открыть последовательный порт RS-485.

Параметр	*Описание*
int baudrate	Скорость обмена со Slave устройством Возможные значения: 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 56000, 57600, 115200
char parity	Чётность посылаемых и принимаемых символов Возможные значения: 0 - NONE,1 - ODD,2 - EVEN
char stopbits	Количество стоповых битов Возможные значения: 1 - один стоп-бит, 2 - два

Возвращаемое значение:
1 - Успешно открыт
0 - Ошибка

Примечание:
Функцию *SerialOpen* необходимо вызывать перед началом работы последовательного порта в любом из трёх основных режимов.

Пример:

```
#include <serial.h>
#include <sat500.h>

void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    //Скорость 19200, чётность - нет, стопбитов - 1
    if(SerialOpen(19200,0,1))
    {
        printf("RS-485 открыт...");
    }
```

```
    else
    {
        printf("Ошибка открытия...");
    }
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:

3.14.1.2. SERIALCLOSE

```
#include <serial.h>

void SerialClose(void);
```

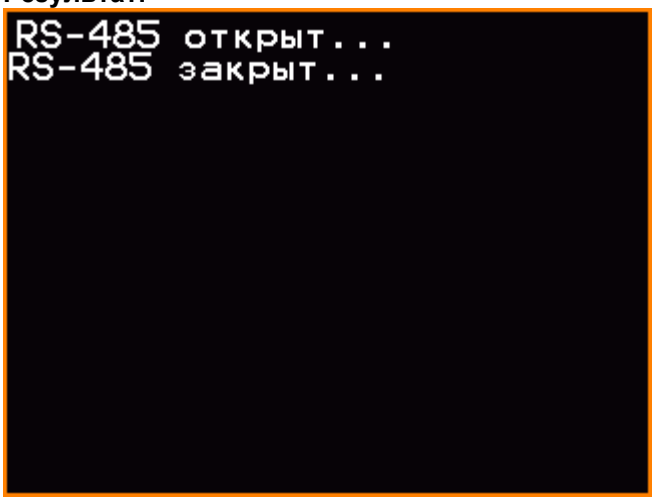
Закрыть последовательный порт RS-485.

Пример:

```
#include <sat500.h>
#include <serial.h>

void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    //Скорость 19200, чётность - нет, стопбитов - 1
    if(SerialOpen(19200,0,1))
    {
        printf("RS-485 открыт...\n");
        SerialClose();
        printf("RS-485 закрыт...\n");
    }
    else
    {
        printf("Ошибка открытия...");
    }
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
```

```
}
```

Результат:



3.14.1.3. SERIALWRITE

```
#include <serial.h>

char SerialWrite(char * data,short int len);
```

Записать данные в последовательный порт RS-485.

*char * data* - массив данных для выдачи в последовательный порт
short int len - количество записываемых данных

Возвращаемое значение:
1 - Успешно записано
0 - Ошибка

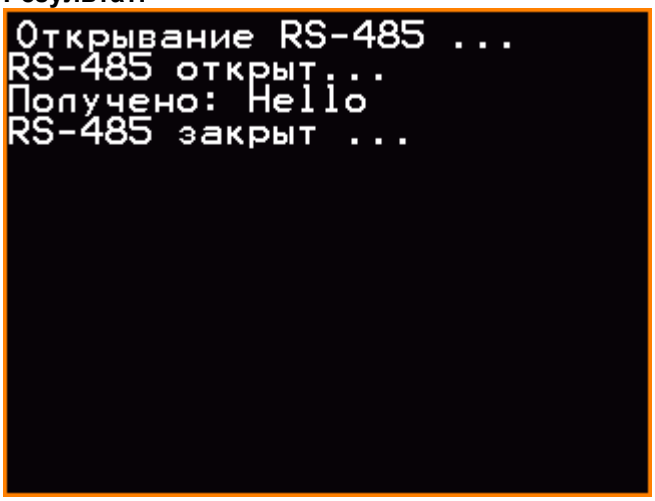
Приведённом ниже примере к порту RS-485 подключено устройство "петля" - посланные байты посылаются обратно.

Пример:

```
#include <serial.h>
#include <sat500.h>
void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    //Скорость 19200, чётность - нет, стопбитов - 1
    printf("Открывание RS-485 ...\n");
    SerialOpen(19200,0,1);
    if(SerialIsOpen())
    {
        printf("RS-485 открыт...\n");
        //Записать данные в последовательный порт
        SerialWrite("Hello\0",6);
        //Прочитать данные из приёмного буфера последовательного порта
```

```
        char data[6];
        if(SerialRead(data,6))
        {
            printf("Получено: %s\n",data);
        }
        else
        {
            printf("Нет ответа ...\n");
        }
        //Закрыть RS-485
        SerialClose();
        printf("RS-485 закрыт ...\n");
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



3.14.1.4. SERIALREADLENGTH

```
#include <serial.h>

short int SerialReadLength(void);
```

Получить количество байтов, доступных для чтения из приёмного буфера последовательного порта RS-485.
Возвращаемое значение: Доступное количество байтов.

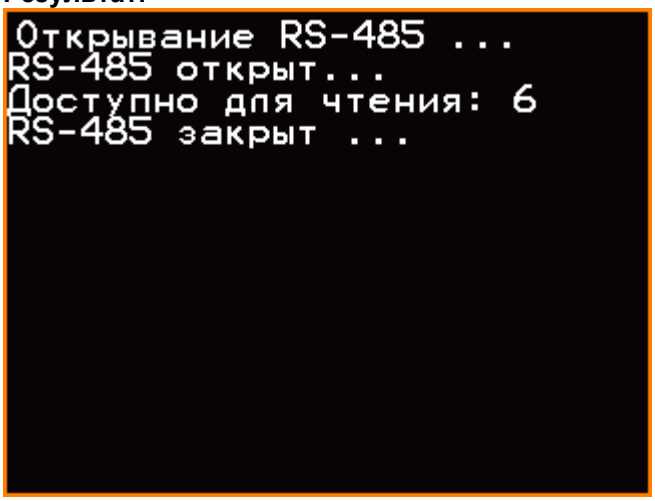
Приведённом ниже примере к порту RS-485 подключено устройство "петля" - посланные байты посылаются обратно.

Пример:

```
#include <serial.h>
#include <sat500.h>

void main(void)
{
    ClrScr();                   //Очистить экран
```

```
    BigFont=0;                //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    //Скорость 19200, чётность - нет, стопбитов - 1
    printf("Открывание RS-485 ...\n");
    SerialOpen(19200,0,1);
    if(SerialIsOpen())
    {
        printf("RS-485 открыт...\n");
        //Записать данные в последовательный порт
        SerialWrite("Hello\0",6);
        //Пауза 50 мс
        unsigned int tc=GetTickCount();
        while((GetTickCount()-tc)<50);
        //Получить количество байтов, доступных для чтения из приёмного буфера
последовательного порта
        short int len=SerialReadLength();
        printf("Доступно для чтения: %d\n",len);
        //Закрыть RS-485
        SerialClose();
        printf("RS-485 закрыт ...\n");
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                //Вывести на дисплей
    while(1);                 //Бесконечная пауза
}
```

Результат:



3.14.1.5. SERIALREAD

```
#include <serial.h>

short int SerialRead(char * data,short int size);
```

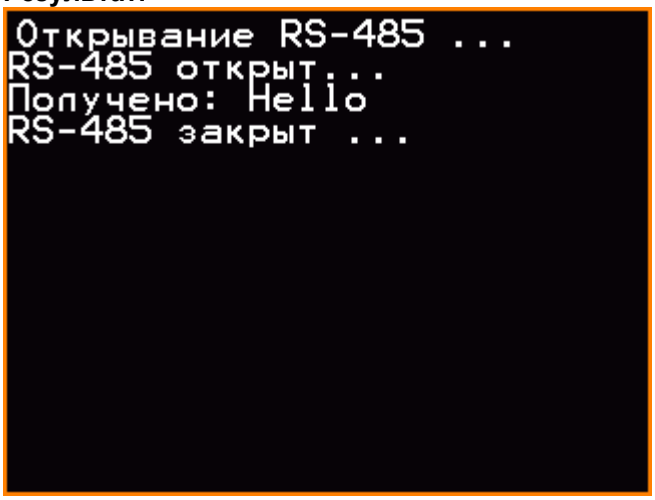
Прочитать данные из приёмного буфера последовательного порта RS-485.
*char * data* - массив данных для приёма из последовательного порта.
short int len - количество читаемых данных.
Возвращаемое значение: количество действительно прочитанных данных.

Приведённом ниже примере к порту RS-485 подключено устройство "петля" - посланные байты посылаются обратно.

Пример:

```
#include <serial.h>
#include <sat500.h>
void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    //Скорость 19200, чётность - нет, стопбитов - 1
    printf("Открывание RS-485 ...\n");
    SerialOpen(19200,0,1);
    if(SerialIsOpen())
    {
        printf("RS-485 открыт...\n");
        //Записать данные в последовательный порт
        SerialWrite("Hello\0",6);
        //Пауза 50 мс
        unsigned int tc=GetTickCount();
        while((GetTickCount()-tc)<50);
        //Прочитать данные из приёмного буфера последовательного порта
        char data[6];
        if(SerialRead(data,6))
        {
            printf("Получено: %s\n",data);
        }
        else
        {
            printf("Нет ответа ...\n");
        }
        //Закрыть RS-485
        SerialClose();
        printf("RS-485 закрыт ...\n");
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



3.14.1.6. SERIALISOPEN

```
#include <serial.h>

char SerialIsOpen(void);
```

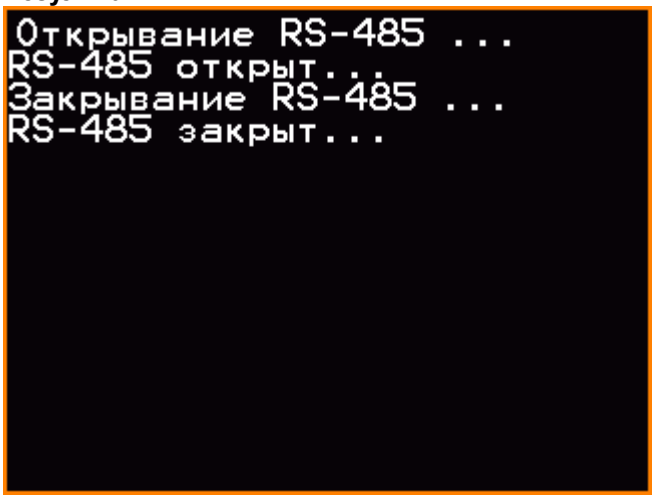
Проверить открыт ли последовательный порт RS-485.

Возвращаемое значение:
1 - Открыт
0 - Закрыт

Пример:

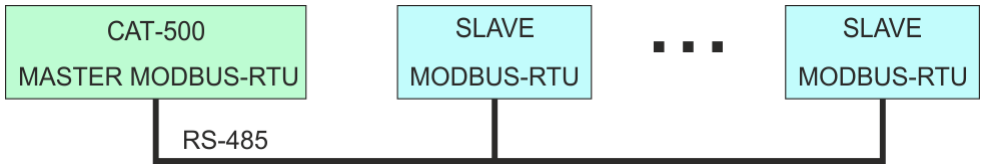
```
#include <serial.h>
#include <sat500.h>

void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    //Скорость 19200, чётность - нет, стопбитов - 1
    printf("Открывание RS-485 ...\n");
    SerialOpen(19200,0,1);
    if(SerialIsOpen())printf("RS-485 открыт...\n");
    else printf("Ошибка открытия...\n");
    //Закрыть RS-485
    printf("Закрывание RS-485 ...\n");
    SerialClose();
    if(SerialIsOpen())printf("RS-485 открыт...");
    else printf("RS-485 закрыт...");
    ShowScr();                   //Вывести на дисплей
    while(1);                    //Бесконечная пауза
}
```

Результат:



3.14.2. MODBUS RTU MASTER

В этом режиме контроллер CAT - 500 работает как мастер-устройство протокола MODBUS RTU.



Перед использованием функций необходимо включить заголовочный файл.

```
#include <serial.h>
```

Все операции с последовательным портом RS-485 выполняются асинхронными функциями - вызываемые функции возвращают результат сразу.
Операции чтения/записи протокола MODBUS RTU выполняются следующим образом:
1. Однократно вызывается функция открытия последовательного порта *SerialOpen* с нужными параметрами.
2. Выполняется вызов функции, определяющий требуемое действие, например чтение или запись регистра.
3. Периодически выполняется вызов функции ModbusResult, которая проверяет завершение выполнения заданного действия или окончание таймаута операции.

MODBUS RTU специфицирует 4 типа данных:

Discrete Inputs - однобитовый тип, доступен только для чтения.
Coils - однобитовый тип, доступен для чтения и записи.
Input Registers - 16-битовый знаковый или беззнаковый тип, доступен только для чтения.
Holding Registers - 16-битовый знаковый или беззнаковый тип, доступен для чтения и записи.

Доступны следующие переменные в режиме мастер-устройство протокола MODBUS RTU:

Переменная	***Описание***
unsigned char ModbusSlave	Опрашиваемый адрес в режиме MODBUS RTU Master Адрес CAT-500 в режиме MODBUS RTU Slave
unsigned int ModbusTimeout	Таймаут в миллисекундах выполняемой операции последовательного порта в режиме MODBUS RTU Master

Доступны следующие функции в режиме мастер-устройство протокола MODBUS RTU:

Функция	***Описание***
char ModbusResult (void)	Получить результат выполнения последней заданной операции протокола MODBUS RTU
*char ModbusReadCoils(unsigned short int adr,unsigned short int cnt,unsigned char * coils)*	Чтение флагов (Coils) (Функция 1) Адрес читаемого устройства задаётся в переменной *ModbusSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*. *unsigned short int adr* - адрес первого читаемого регистра (начиная с 0) *unsigned short int cnt* - количество читаемых регистров *unsigned char * coils* - указатель на массив куда будут записаны принятые флаги (один байт - один флаг) Возвращаемое значение: 1 - Заданное количество флагов успешно прочитано 0 - Ошибка

*char ModbusReadDiscreteInputs(unsigned short int adr,unsigned short int cnt,unsigned char * inp)*	Чтение дискретных входов (Discrete Inputs) (Функция 2) Адрес читаемого устройства задаётся в переменной *ModbusSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*. *unsigned short int adr* - адрес первого читаемого дискретного входа (начиная с 0) *unsigned short int cnt* - количество читаемых дискретных входов *unsigned char * inp* - указатель на массив куда будут записаны принятые входы (один байт - один вход) Возвращаемое значение: 1 - Заданное количество дискретных входов успешно прочитано 0 - Ошибка
*char ModbusReadHoldingRegisters (unsigned short int adr,unsigned short int cnt,unsigned short int * reg)*	Прочитать регистры хранения Slave устройства интерфейса MODBUS RTU (Функция 3) Адрес читаемого устройства задаётся в переменной *ModbusSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*. *unsigned short int adr* - адрес первого читаемого регистра (начиная с 0) *unsigned short int cnt* - количество читаемых регистров *unsigned short int * reg* - указатель на массив куда будут записаны принятые регистры хранения Возвращаемое значение: 1 - Заданное количество регистров успешно прочитано 0 - Ошибка
*char ModbusReadInputRegisters (unsigned short int adr,unsigned short int cnt,unsigned short int * reg)*	Прочитать входные регистры Slave устройства интерфейса MODBUS RTU (Функция 4) Адрес читаемого устройства задаётся в переменной *ModbusSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*. *unsigned short int adr* - адрес первого читаемого регистра (начиная с 0) *unsigned short int cnt* - количество читаемых регистров *unsigned short int * reg* - указатель на массив куда будут записаны принятые регистры хранения Возвращаемое значение: 1 - Заданное количество регистров успешно прочитано 0 - Ошибка
*char ModbusReadStatus(unsigned char * status)*	Чтение статуса (Функция 7) Адрес читаемого устройства задаётся в переменной *ModbusSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*. *unsigned char * status* - указатель на получаемое значение статуса Возвращаемое значение: 1 - Успешно прочитано 0 - Ошибка
char ModbusWriteSingleCoil(unsigned short int adr,unsigned short int value)	Запись одного флага (Coil) (Функция 5) Адрес читаемого устройства задаётся в переменной *ModbusSlave*.

	Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*. *unsigned short int adr* - адрес записываемого регистра (начиная с 0) *unsigned short int value* - записываемое значение Возвращаемое значение: 1 - Успешно записано 0 - Ошибка
char ModbusWriteHoldingRegister(unsigned short int adr,unsigned short int value)	Запись значения одного регистра хранения Slave устройства (Функция 6) Адрес читаемого устройства задаётся в переменной *ModbusSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*. *unsigned short int adr* - адрес записываемого регистра (начиная с 0) *unsigned short int value* - записываемое значение Возвращаемое значение: 1 - Успешно записано 0 - Ошибка
*char ModbusWriteRegisters (unsigned short int adr,unsigned short int cnt,unsigned short int * reg)*	Запись нескольких регистров хранения Slave устройства (Функция 16) Адрес читаемого устройства задаётся в переменной *ModbusSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*. *unsigned short int adr* - начальный адрес записываемых регистров (начиная с 0) *unsigned short int cnt* - количество записываемых регистров *unsigned short int * reg* - указатель на массив записываемых регистров Возвращаемое значение: 1 - Успешно записано 0 - Ошибка
*char ModbusWriteCoils (unsigned short int adr,unsigned short int cnt,unsigned char * coils)*	Запись нескольких флагов (Coils) (Функция 15). Адрес читаемого устройства задаётся в переменной *ModbusSlave*. Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*. *unsigned short int adr* - адрес записываемого флага (начиная с 0) *unsigned short int cnt* - количество записываемых флагов *unsigned short int coils* - записываемые значения флагов (один байт - один флаг) Возвращаемое значение: 1 - Успешно прочитано 0 - Ошибка
*unsigned short int ModbusCrc16(unsigned char * data,unsigned short int cnt)*	Расчёт контрольной суммы MODBUS RTU *unsigned char * data* - указатель на данные *unsigned short int cnt* - количество данных Возвращаемое значение: Двухбайтовая контрольная сумма

3.14.2.1. MODBUSSLAVE

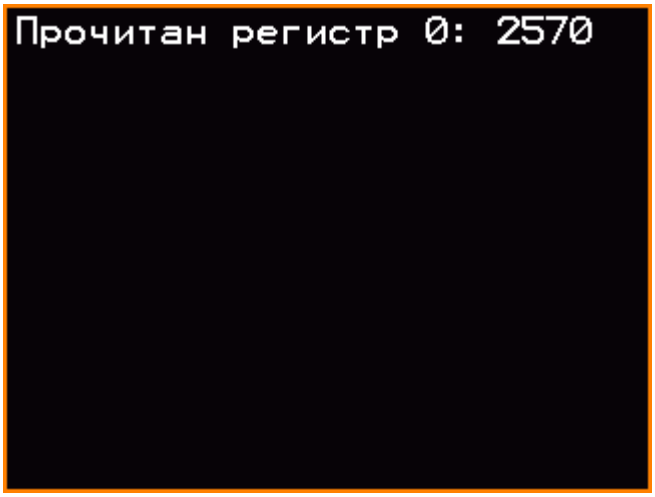
```
#include <serial.h>

unsigned char ModbusSlave
```

Переменная, значение которой задаёт опрашиваемый адрес в режиме MODBUS RTU Мастер. Также переменная используется как SLAVE адрес контроллера CAT-500 в режиме MODBUS RTU Slave.

Пример:

```
#include <sat500.h>
#include <serial.h>

typedef unsigned short int WORD;
WORD reg;

void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    SerialOpen(19200,0,1);      //Скорость 19200, чётность - нет, стоп-битов - 1
   if(SerialIsOpen())
   {
        //Успешно открыт
        //Адрес читаемого MODBUS устройства
        ModbusSlave=10;
        //Таймаут в миллисекундах
        ModbusTimeout=500;
        //Прочитать регистр хранения по адресу 0 в переменную reg
        if(ModbusReadRegisters(0,1,&reg))
        {
            //Выполняется чтение
            //Ожидать завершения
            char result;    //Результат чтения
            do
            {
                result=ModbusResult();
            }
            while(result<0);   //Пока не завершена операция чтения или не за-
кончен таймаут
            GotoXY(0,40);
            if(result==0)
            {
                  //Успешное чтение регистра
                  printf("Прочитан регистр 0: %d",reg);
            }
            else
            {
                 printf("Ошибка чтения ...");
            }
         }
    }
    else printf("Ошибка открытия...");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



3.14.2.2. MODBUSTIMEOUT

```
#include <serial.h>

unsigned int ModbusTimeout
```

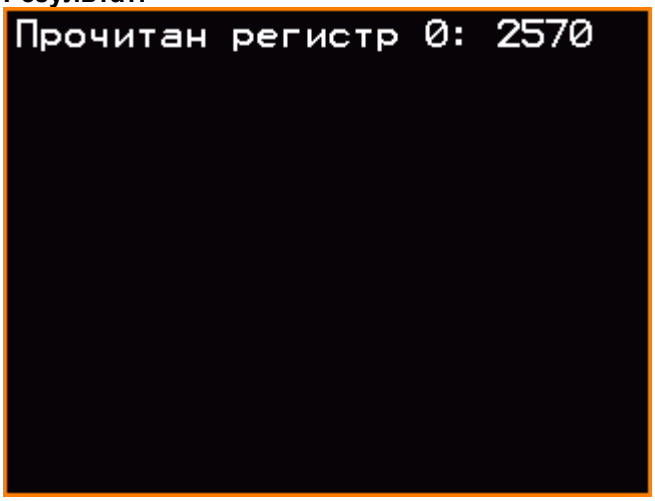
Переменная, значение которой задаёт таймаут в миллисекундах на выполнение операции чтения/записи в режиме MODBUS RTU Мастер.

Пример:

```
#include <sat500.h>
#include <serial.h>

typedef unsigned short int WORD;
WORD reg;

void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    SerialOpen(19200,0,1);      //Скорость 19200, чётность - нет, стоп-битов - 1
    if(SerialIsOpen())
    {
        //Успешно открыт
        //Адрес читаемого MODBUS устройства
        ModbusSlave=10;
        //Таймаут в миллисекундах
        ModbusTimeout=500;
        //Прочитать регистр хранения по адресу 0 в переменную reg
        if(ModbusReadRegisters(0,1,&reg))
        {
            //Выполняется чтение
            //Ожидать завершения
            char result;    //Результат чтения
            do
            {
                result=ModbusResult();
            }
```

```
            while(result<0);  //Пока не завершена операция чтения или не за-
кончен таймаут
            GotoXY(0,40);
            if(result==0)
            {
                  //Успешное чтение регистра
                  printf("Прочитан регистр 0: %d",reg);
            }
            else
            {
                 printf("Ошибка чтения ...");
            }
         }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



3.14.2.3. MODBUSRESULT

```
#include <serial.h>

char ModbusResult(void);
```

Получить результат выполнения последней заданной операции протокола MODBUS RTU
Возможные возвращаемое значение показаны в таблице ниже.

Значение	*Описание*
-1	Операция ещё выполняется
0	Последняя операция выполнена успешно
*1**	Неправильный номер функции для заданного slave устройства
*2**	Неправильный адрес для заданного slave устройства
*3**	Неправильные данные для заданного slave устройства
*4**	Произошла ошибка в slave устройстве при выполнении заданной функции
*5**	Подтверждение (Acknowledge) - для выполнения заданной операции требуется больше времени
*6**	Slave устройство занято и сейчас не может выполнить запрошенную функцию (BUSY)

*7**	Отказ выполнения функции slave устройством (NAK)
*8**	Ошибка чётности в памяти slave
*10**	Проблема шлюза (Gateway paths not available)
*11**	Сообщение шлюза - устройство не отвечает (The targeted device failed to respond)
12	Общая ошибка
13	Не удалось послать
14	Неправильный байт адреса slave в ответе
15	Нет ответа
16	Ошибка CRC в ответе
17	Мало байтов в ответе
18	Неправильный байт функции в ответе
19	Неправильный режим последовательного порта

* - стандартный код исключительной ошибки протокола MODBUS RTU (EXEPTION)

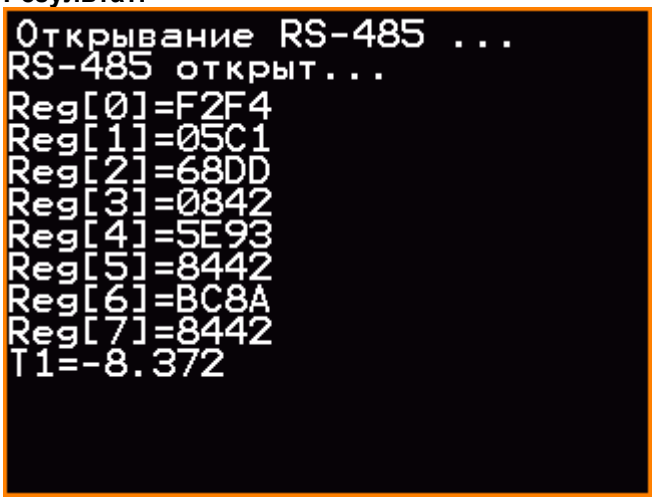
В примере далее читается второй контроллер CAT-500, работающий в режиме регулятора отопления.

Пример:

```
#include <serial.h>
#include <sat500.h>

typedef unsigned int  UINT;

void main(void)
{
    ClrScr();                  //Очистить экран
   BigFont=0;                 //Обычный шрифт 12x16 (не крупный)
   //Открыть RS-485
   //Скорость 115200, чётность - нет, стопбитов - 1
   printf("Открывание RS-485 ...\n");
   SerialOpen(115200,0,1);
   if(SerialIsOpen())
   {
      printf("RS-485 открыт...\n");
      ShowScr();                  //Вывести не дисплей
      GotoXY(4,40);
      ModbusSlave=48;
      ModbusTimeout=1000;
      while(1)
      {
        short regs[32];
        if(ModbusReadHoldingRegisters(16,8,regs))
        {
            char result;
            do
            {
                result=ModbusResult();
            }
            while(result<0);
            GotoXY(0,40);
            if(result==0)
            {
                for(int i=0;i<8;i++)
                printf("Reg[%d]=%04X    \n",i,regs[i]&0xFFFF);
                //В регистрах 16-17 температура в формате float  ieee 754
```

```
                //Температура в 1000-ных долях градуса
                int t=GetModbusFloat(regs,1000);
                printf("T1=%d.%03d    \n",t/1000,abs(t%1000));
            }
            else
            {
                printf("result=%d    ",result);
            }
            ShowScr();   //Вывести на дисплей
            UINT t=GetTickCount();
            while((GetTickCount()-t)<300);
        }
      }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                //Вывести на дисплей
   while(1);                  //Бесконечная пауза
}
```

Результат:



3.14.2.4. MODBUSREADCOILS

```
#include <serial.h>

char ModbusReadCoils(unsigned short int adr,unsigned short int cnt,unsigned
char * coils);
```

Чтение флагов (Coils) (Функция 1) в соответствии с протоколом MODBUS RTU.
Адрес читаемого устройства задаётся в переменной *ModbusSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*.

unsigned short int adr - адрес первого читаемого флага (начиная с 0)
unsigned short int cnt - количество читаемых регистров
*unsigned char * coils* - указатель на массив куда будут записаны принятые флаги

ВНИМАНИЕ: Каждый байт массива *coils* является значением одного флага *coil*.

В примере далее читается второй контроллер CAT-500, работающий в режиме регулятора отопления.
Начиная с адреса 256 расположены 11 флагов - состояние выходов контроллера.

Возвращаемое значение:
1 - Заданное количество флагов успешно прочитано
0 - Ошибка

Наиболее часто в качестве coils используют дискретные выходные сигналы.

Пример:

```
#include <serial.h>
#include <sat500.h>

typedef unsigned int  UINT;

void main(void)
{
    ClrScr();                  //Очистить экран
   BigFont=0;                 //Обычный шрифт 12x16 (не крупный)
   puts("Пример ModbusReadCoils\n");
   //Открыть RS-485
   //Скорость 115200, чётность - нет, стопбитов - 1
   puts("Открывание RS-485 ...\n");
   SerialOpen(115200,0,1);
   if(SerialIsOpen())
   {
      printf("RS-485 открыт...\n");
      ShowScr();                  //Вывести не дисплей
      ModbusSlave=48;
      ModbusTimeout=1000;
      while(1)
      {
        unsigned char coils[11];
        if(ModbusReadCoils(256,11,coils))
        {
            char result;
            do
            {
                result=ModbusResult();
            }
            while(result<0);
            GotoXY(0,40);
            if(result==0)
            {
                for(int i=0;i<11;i++)
                printf("DO%d coils[%d]=%d\n",i+1,i,coils[i]);
            }
            else
            {
                printf("result=%d    ",result);
            }
            ShowScr();   //Вывести на дисплей
            UINT t=GetTickCount();
            while((GetTickCount()-t)<300);
        }
        else
           puts("Ошибка ModbusReadCoils\n");
      }
    }
```

```
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
   while(1);                    //Бесконечная пауза
}
```

Результат:

```
Пример ModbusReadCoils
Открывание RS-485 ...
RS-485 открыт...
D01 coils[0]=0
D02 coils[1]=0
D03 coils[2]=0
D04 coils[3]=0
D05 coils[4]=0
D06 coils[5]=0
D07 coils[6]=0
D08 coils[7]=1
D09 coils[8]=0
D010 coils[9]=0
D011 coils[10]=1
```

Если в приведённом выше примере установить неправильный Slave адрес контроллера CAT-500:

```
...
      ModbusSlave=1;
...
```

То получим следующий результат:

```
Пример ModbusReadCoils
Открывание RS-485 ...
RS-485 открыт...
result=15
```

Ошибка номер 15 обозначает "**Нет ответа**" в соответствии с таблицей кодов ошибок, приведённой в разделе *ModbusResult*

3.14.2.5. MODBUSREADDISCRETEINPUTS

```
#include <serial.h>

char ModbusReadDiscreteInputs(unsigned short int adr,unsigned short int
cnt,unsigned char * inp);
```

Чтение дискретных входов (Discrete Inputs) (Функция 2) в соответствии с протоколом MODBUS RTU.

Адрес читаемого устройства задаётся в переменной *ModbusSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*.

unsigned short int adr - адрес первого читаемого дискретного входа (начиная с 0)
unsigned short int cnt - количество читаемых дискретных входов
*unsigned char * inp* - указатель на массив куда будут записаны принятые входы

ВНИМАНИЕ: Каждый байт массива *inp* является значением одного дискретного входа.

Возвращаемое значение:
1 - Заданное количество дискретных входов успешно прочитано
0 - Ошибка

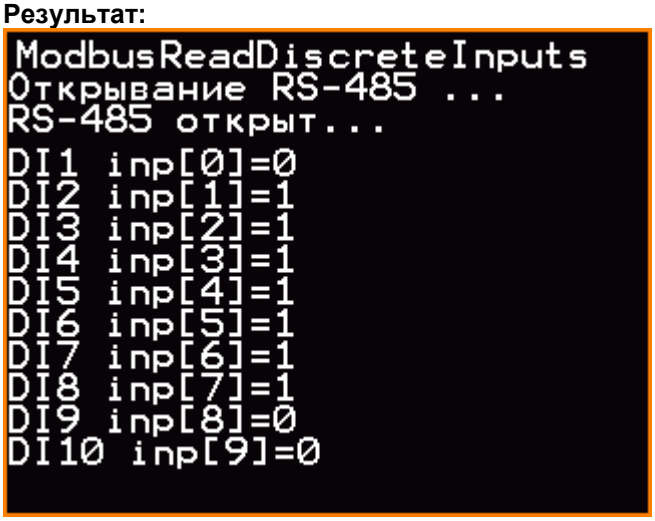
В примере далее читается второй контроллер САТ-500, работающий в режиме регулятора отопления.
Начиная с адреса 256 расположены 10 дискретных входов - состояние входов D1...D10 контроллера.

Пример:

```
#include <serial.h>
#include <sat500.h>

typedef unsigned int  UINT;

void main(void)
{
    ClrScr();                  //Очистить экран
   BigFont=0;                 //Обычный шрифт 12x16 (не крупный)
   puts("ModbusReadDiscreteInputs\n");
   //Открыть RS-485
   //Скорость 115200, чётность - нет, стопбитов - 1
   puts("Открывание RS-485 ...\n");
   SerialOpen(115200,0,1);
   if(SerialIsOpen())
   {
      printf("RS-485 открыт...\n");
      ShowScr();                  //Вывести не дисплей
      ModbusSlave=48;
      ModbusTimeout=1000;
      while(1)
      {
        unsigned char inp[10];
        if(ModbusReadDiscreteInputs(256,10,inp))
        {
            char result;
            do
            {
                result=ModbusResult();
            }
            while(result<0);
            GotoXY(0,58);
            if(result==0)
            {
                for(int i=0;i<10;i++)
                printf("DI%d inp[%d]=%d\n",i+1,i,inp[i]);
            }
            else
```

```
            {
                printf("result=%d    ",result);
            }
            ShowScr();   //Вывести на дисплей
            UINT t=GetTickCount();
            while((GetTickCount()-t)<300);
        }
        else
           puts("Ошибка ModbusReadDiscreteInputs\n");
      }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                //Вывести на дисплей
   while(1);                  //Бесконечная пауза
}
```

Результат:



3.14.2.6. MODBUSREADHOLDINGREGISTERS

```
#include <serial.h>

char ModbusReadHoldingRegisters(unsigned short int adr,unsigned short int
cnt,unsigned short int * reg);
```

Прочитать регистры хранения Slave устройства интерфейса MODBUS RTU (Функция 3) в соответствии с протоколом MODBUS RTU.
Адрес читаемого устройства задаётся в переменной *ModbusSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*.

unsigned short int adr - адрес первого читаемого регистра (начиная с 0)
unsigned short int cnt - количество читаемых регистров
*unsigned short int * reg* - указатель на массив куда будут записаны принятые регистры хранения

Возвращаемое значение:
1 - Заданное количество регистров успешно прочитано
0 - Ошибка

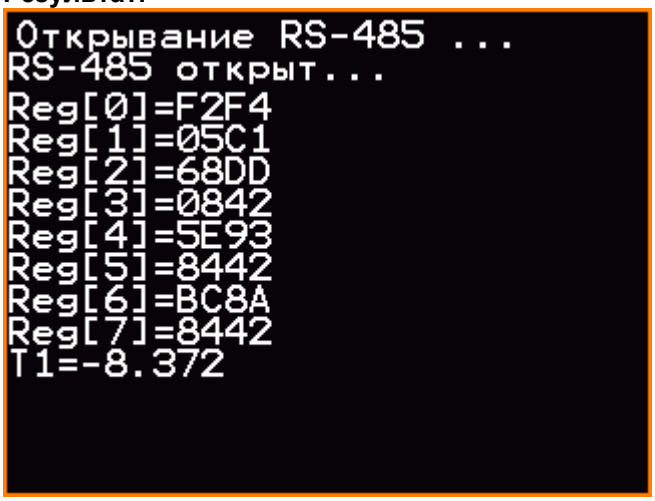
В примере далее читается второй контроллер CAT-500, работающий в режиме регулятора отопления.
Читаются 8 регистров хранения и их содержимое выводится на дисплей.

Пример:

```
#include <serial.h>
#include <sat500.h>

typedef unsigned int  UINT;

void main(void)
{
    ClrScr();                  //Очистить экран
   BigFont=0;                 //Обычный шрифт 12x16 (не крупный)
   //Открыть RS-485
   //Скорость 115200, чётность - нет, стопбитов - 1
   printf("Открывание RS-485 ...\n");
   SerialOpen(115200,0,1);
   if(SerialIsOpen())
   {
      printf("RS-485 открыт...\n");
      ShowScr();                  //Вывести не дисплей
      GotoXY(4,40);
      ModbusSlave=48;
      ModbusTimeout=1000;
      while(1)
      {
        short regs[32];
        if(ModbusReadHoldingRegisters(16,8,regs))
        {
            char result;
            do
            {
                result=ModbusResult();
            }
            while(result<0);
            GotoXY(0,40);
            if(result==0)
            {
                for(int i=0;i<8;i++)
                printf("Reg[%d]=%04X    \n",i,regs[i]&0xFFFF);
                //В регистрах 16-17 температура в формате float  ieee 754
                //Температура в 1000-ных долях градуса
                int t=GetModbusFloat(regs,1000);
                printf("T1=%d.%03d    \n",t/1000,abs(t%1000));
            }
            else
            {
                printf("result=%d    ",result);
            }
            ShowScr();   //Вывести на дисплей
            UINT t=GetTickCount();
            while((GetTickCount()-t)<300);
        }
      }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                 //Вывести на дисплей
   while(1);                   //Бесконечная пауза
```

```
}
```

Результат:



3.14.2.7. MODBUSREADINPUTREGISTERS

```
#include <serial.h>

char ModbusReadInputRegisters(unsigned short int adr,unsigned short int
cnt,unsigned short int * reg);
```

Прочитать входные регистры Slave устройства интерфейса MODBUS RTU (Функция 4) в соответствии с протоколом MODBUS RTU.
Адрес читаемого устройства задаётся в переменной *ModbusSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*.

unsigned short int adr - адрес первого читаемого регистра (начиная с 0)
unsigned short int cnt - количество читаемых регистров
*unsigned short int * reg* - указатель на массив куда будут записаны принятые входные регистры

Возвращаемое значение:
1 - Заданное количество регистров успешно прочитано
0 - Ошибка

Пример:

```
#include <sat500.h>
#include <serial.h>

typedef unsigned int  UINT;

void main(void)
{
    ClrScr();                //Очистить экран
    BigFont=0;               //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    //Скорость 19200, чётность - нет, стопбитов - 1
    printf("Открывание RS-485 ...\n");
    SerialOpen(19200,0,1);
    if(SerialIsOpen())
    {
      printf("RS-485 открыт...\n");
```

```
        ShowScr();              //Вывести на дисплей
        ModbusSlave=48;
        ModbusTimeout=1000;
        while(1)
        {
           //Регистры, куда будет помещён ответ
           short regs[8];
          if(ModbusReadInputRegisters(0,8,regs))
          {
              char result;
              do
              {
                  result=ModbusResult();
              }
              while(result<0);
              GotoXY(0,40);
              if(result==0)
              {
                  //Просмотр полученных регистров
                  for(int i=0;i<8;i++)
                  printf("Reg[%d]=%x    \n",i,regs[i]&0xFFFF);
              }
              else
              {
                  printf("result=%d    ",result);
              }
              ShowScr();   //Вывести на дисплей
              //Пауза 300 мс
              UINT t=GetTickCount();
              while((GetTickCount()-t)<300);
          }
        }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести нарисованное на дисплей
    while(1);                   //Бесконечная пауза
}
```

3.14.2.8. MODBUSREADSTATUS

```
#include <serial.h>

char ModbusReadStatus(unsigned char * status);
```

Чтение статуса (Функция 7) в соответствии с протоколом MODBUS RTU.
Адрес читаемого устройства задаётся в переменной *ModbusSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*.

*unsigned char * status* - указатель на получаемое значение статуса

Возвращаемое значение:
1 - Успешно прочитано
0 - Ошибка

3.14.2.9. MODBUSWRITESINGLECOIL

```
#include <serial.h>

char ModbusWriteSingleCoil(unsigned short int adr,unsigned short int value);
```

Запись одного флага (Coil) (Функция 5) в соответствии с протоколом MODBUS RTU.
Адрес читаемого устройства задаётся в переменной *ModbusSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*.

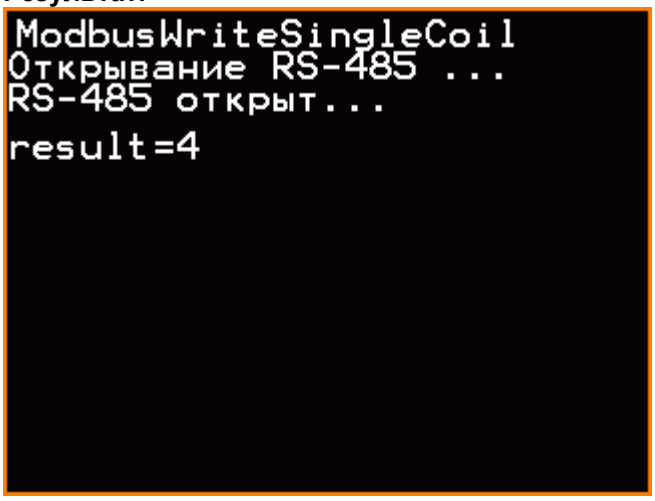
unsigned short int adr - адрес записываемого флага (начиная с 0)
unsigned short int value - записываемое значение

Возвращаемое значение:
1 - Успешно записано
0 - Ошибка

```
#include <serial.h>
#include <sat500.h>


typedef unsigned int  UINT;

void main(void)
{
    ClrScr();                  //Очистить экран
   BigFont=0;                 //Обычный шрифт 12x16 (не крупный)
   puts("ModbusWriteSingleCoil\n");
   //Открыть RS-485
   //Скорость 115200, чётность - нет, стопбитов - 1
   puts("Открывание RS-485 ...\n");
   SerialOpen(115200,0,1);
   if(SerialIsOpen())
   {
      printf("RS-485 открыт...\n");
      ShowScr();                 //Вывести не дисплей
      ModbusSlave=48;
      ModbusTimeout=1000;
      while(1)
      {
        //Записать в флаг 256 значение 0
        if(ModbusWriteSingleCoil(256,0))
        {
            char result;
            do
            {
                result=ModbusResult();
            }
            while(result<0);
            GotoXY(0,40);
            if(result==0)
            {
                puts("Успешно записано");
            }
            else
            {
                printf("result=%d    ",result);
            }
            ShowScr();   //Вывести на дисплей
            UINT t=GetTickCount();
```

```
            while((GetTickCount()-t)<300);
        }
        else
           puts("Ошибка ModbusWriteSingleCoil\n");
      }
    }
    else printf("Ошибка открытия...\n");
    ShowScr();                  //Вывести на дисплей
   while(1);                    //Бесконечная пауза
}
```

Результат:



При выполнении записи произошла исключительная ошибка номер 4 протокола MODBUS: "Произошла ошибка в slave устройстве при выполнении заданной функции" в соответствии со списком ошибок в разделе *ModbusResult*
Ошибка произошла из-за того что выход DO1, который мы записывали задействован в контуре регулирование и его изменение извне недопустимо.

3.14.2.10. MODBUSWRITEHOLDINGREGISTER

```
#include <serial.h>

char ModbusWriteHoldingRegister(unsigned short int adr,unsigned short int
value);
```

Запись значения одного регистра хранения Slave устройства (Функция 6) в соответствии с протоколом MODBUS RTU.
Адрес читаемого устройства задаётся в переменной *ModbusSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*.

unsigned short int adr - адрес записываемого регистра (начиная с 0)
unsigned short int value - записываемое значение

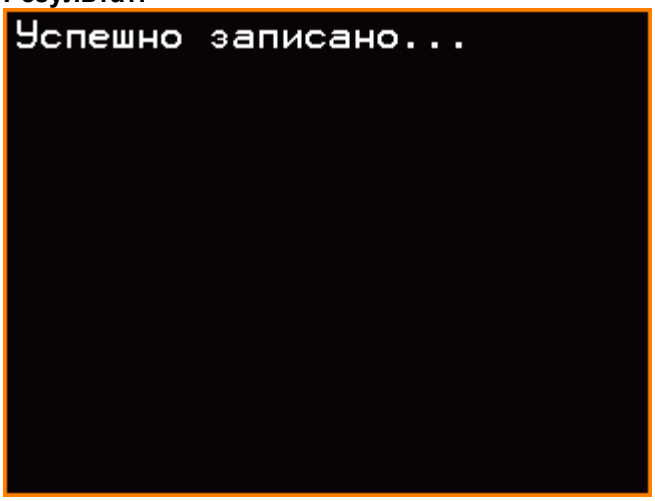
Возвращаемое значение:
1 - Успешно записано
0 - Ошибка

Пример:

```
#include <sat500.h>
#include <serial.h>
```

```
void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    SerialOpen(19200,0,1);      //Скорость 19200, чётность - нет, стопбитов - 1
    //Адрес MODBUS устройства
    ModbusSlaveAdr=10;
    //Таймаут в миллисекундах
    ModbusTimeout=500;
    //Записать регистр хранения по адресу 2048
    if(ModbusWriteHoldingRegister(2048,0))
    {
            char result;    //Результат записи
            do
            {
                result=ModbusResult();
            }
            while(result<0);   //Пока не завершена операция записи или не за-
кончен таймаут
            if(result==0)
            printf("Успешно записано...\n");
            else
            printf("Не удалось записать...\n");
    }
    else
    {
        printf("Ошибка записи...");
    }
    //Закрыть RS-485
    SerialClose();
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



3.14.2.11. MODBUSWRITEREGISTERS

```
#include <serial.h>
```

```
char ModbusWriteRegisters(unsigned short int adr,unsigned short int cnt,un-
signed short int * reg);
```

Запись нескольких регистров хранения Slave устройства (Функция 16) в соответствии с протоколом MODBUS RTU.
Адрес читаемого устройства задаётся в переменной *ModbusSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*.

unsigned short int adr - начальный адрес записываемых регистров (начиная с 0)
unsigned short int cnt - количество записываемых регистров
*unsigned short int * reg* - указатель на массив записываемых регистров

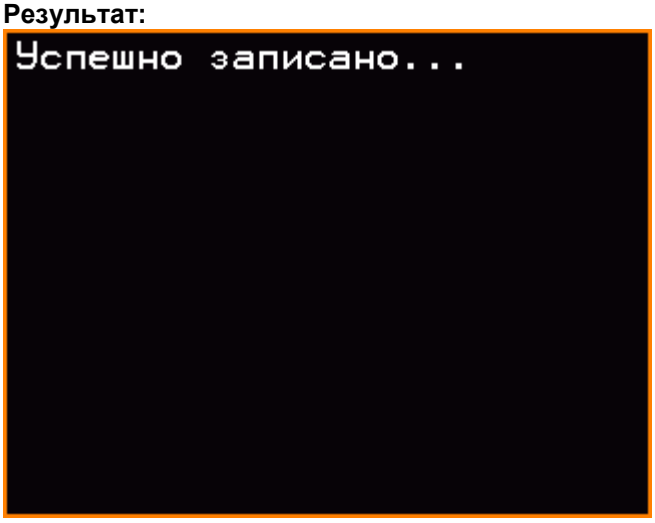
Возвращаемое значение:
1 - Успешно записано
0 - Ошибка

Пример:

```
#include <sat500.h>
#include <serial.h>

typedef unsigned short int WORD;

WORD reg[4];

void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    SerialOpen(19200,0,1);      //Скорость 19200, чётность - нет, стопбитов - 1
    //Адрес MODBUS устройства
    ModbusSlaveAdr=10;
    //Таймаут в миллисекундах
    ModbusTimeout=500;
    //Записать регистры хранения по адресам 2048-2051
    reg[0]=0;
    reg[1]=0;
    reg[2]=0;
    reg[3]=0;
    if(ModbusWriteRegisters(2048,4,reg))
    {
        char result;    //Результат записи
        do
        {
            result=ModbusResult();
        }
        while(result<0);      //Пока не завершена операция записи или не за-
кончен таймаут
        if(result==0)
        printf("Успешно записано...\n");
        else
        printf("Не удалось записать...\n");
    }
    else
    {
        printf("Ошибка записи ...");
```

```
    }
    //Закрыть RS-485
    SerialClose();
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



3.14.2.12. MODBUSWRITECOILS

```
#include <serial.h>

char ModbusWriteCoils(unsigned short int adr,unsigned short int cnt,unsigned
char * coils);
```

Запись нескольких флагов (Coils) (Функция 15) в соответствии с протоколом MODBUS RTU.
Адрес читаемого устройства задаётся в переменной *ModbusSlave*.
Таймаут выполняемой операции задаётся в переменной *ModbusTimeout*.

unsigned short int adr - адрес записываемого флага (начиная с 0)
unsigned short int cnt - количество записываемых флагов
unsigned short int coils - записываемые значения флагов

ВНИМАНИЕ: Каждый байт массива *coils* является значением одного флага coil.

Возвращаемое значение:
1 - Успешно записано
0 - Ошибка

3.14.2.13. MODBUSCRC16

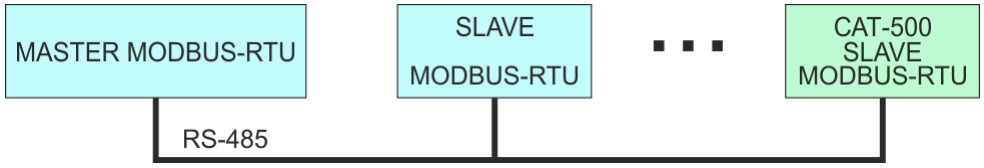
```
#include <serial.h>

unsigned short int ModbusCrc16(unsigned char * data,unsigned short int cnt);
```

Расчёт контрольной суммы MODBUS RTU.
*unsigned char * data* - указатель на данные
unsigned short int cnt - количество данных

Возвращаемое значение: Двухбайтовая контрольная сумма.

3.14.3. MODBUS RTU SLAVE

В этом режиме контроллер CAT - 500 работает как slave-устройство протокола MODBUS RTU.



Перед использованием функций необходимо включить заголовочный файл.

```
#include <serial.h>
```

Доступны следующие переменные в режиме SLAVE устройство протокола MODBUS RTU:

Переменная	***Описание***
unsigned int *ModbusCount*	Переменная, которая содержит счетчик корректных запросов MODBUS RTU Slave на которые был выдан ответ

Доступны следующие функции в режиме SLAVE устройство протокола MODBUS RTU:

Функция	***Описание***
unsigned char *StartModbusSlave* (unsigned short int * hld,unsigned short int hldcnt,unsigned short int * inp,unsigned short int inpcnt)	Функция *StartModbusSlave* разрешает режим работы MODBUS Slave контроллера CAT-500. В качестве параметров ей передаются количество и адреса регистров хранения, и регистров, количество и адреса входных регистров. Перед вызовом функции необходимо настроить последовательный порт при помощи функции *SerialOpen*. Адрес Slave устройства задается в переменной *ModbusSlave*. *unsigned short int * hld* - указатель на первый регистр хранения, который будет иметь MODBUS адрес 0 *unsigned short int hldcnt* - количество регистров хранения, расположенных в памяти начиная с адреса hld *unsigned short int * inp* - - указатель на первый входной регистр, который будет иметь MODBUS адрес 0 *unsigned short int inpcnt* - количество входных регистров, расположенных начиная с адреса inp Возвращаемое значение: 1 - Успешно включён MODBUS SLAVE режим 0 - Ошибка
void *StopModbusSlave* (void)	Функция *StopModbusSlave* завершает режима работы MODBUS Slave контроллера CAT-500

3.14.3.1. STARTMODBUSSLAVE

```
#include <serial.h>

unsigned char StartModbusSlave(unsigned short int * hld,unsigned short int
hldcnt,unsigned short int * inp,unsigned short int inpcnt);
```

Функция *StartModbusSlave* разрешает режим работы MODBUS Slave контроллера CAT-500.
В качестве параметров ей передаются количество и адреса регистров хранения, и регистров, количество и адреса входных регистров. Перед вызовом функции необходимо настроить последовательный порт при помощи функции *SerialOpen*.
Адрес Slave устройства задается в переменной *ModbusSlave*.

*unsigned short int * hld* - указатель на первый регистр хранения, который будет иметь MODBUS адрес 0
unsigned short int hldcnt - количество регистров хранения, расположенных в памяти начиная с адреса *hld*
*unsigned short int * inp* - - указатель на первый входной регистр, который будет иметь MODBUS адрес 0
unsigned short int inpcnt - количество входных регистров, расположенных начиная с адреса *inp*

Возвращаемое значение:
1 - Успешно включён MODBUS SLAVE режим
0 - Ошибка

Если регистры хранения не нужны то следует задать значение параметра *hldcnt* равным 0.
Если входные регистры не нужны то следует задать значение параметра *inpcnt* равным 0.

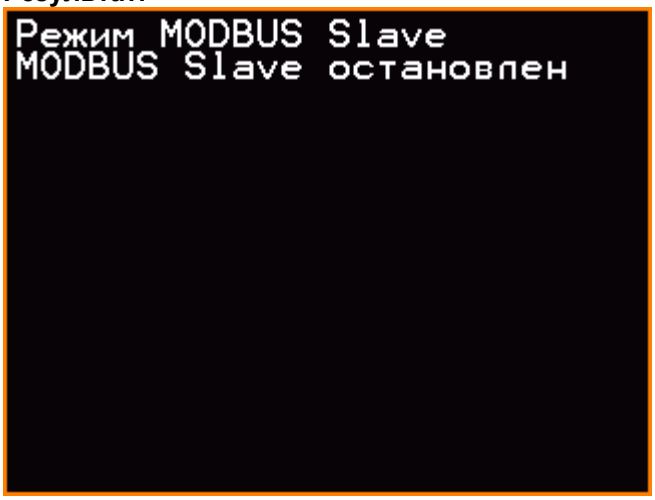
Переменные, располагаемые в памяти контроллера всегда выравниваются по байтовый границе и расположены точно так, как описано в программе. Сегментация переменных не используются.

Пример:

```
#include <sat500.h>
#include <serial.h>

short int Var1;
char MyAdr;
unsigned char MyVal;
unsigned int UiValue;
short Data[4];

void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Заполняем переменные (для проверки чтения регистров)
    Var1=12345;
    MyAdr=1;
    MyVal=2;
    UiValue=0x01020304;
    Data[0]=0xFF00;
    Data[1]=0xEE11;
    Data[2]=0xDD22;
    Data[0]=0xCC33;
    //Открыть RS-485
    SerialOpen(19200,0,1);    //Скорость 19200, чётность - нет, стопбитов - 1
    if(SerialIsOpen())
    {
        //Успешно открыт
        //Запуск MODBUS RTU Slave
        if(StartModbusSlave(&Var1,8,&Var1,0))
        {
            puts("Режим MODBUS Slave");
            ShowScr();                  //Вывести на дисплей
            EnableKeyboard=1;           //Разрешить программе работу с клавиа-
турой
            //Пока не нажата кнопка - ничего не делаем
            while(!GetKey())
            {
            }
```

```
            //Завершение режима MODBUS Slave контроллера CAT-500
            puts("\nMODBUS Slave остановлен");
            EnableKeyboard=0;
        }
    }
    else
    {
         puts("Ошибка открытия...\n");
    }
    ShowScr();                  //Вывести на дисплей
    while(1);                   //Бесконечная пауза
}
```

Результат:



В этом примере запускается режим MODBUS SLAVE и работают 8 регистров хранения с адресами 0...7.
Входные регистры не используются, т.к. задано значение 0 в количестве входных регистров.

Таким образом, переменные будут иметь следующие MODBUS адреса:

Переменная	MODBUS адрес
Var1	0
MyAdr	1 - младший байт
MyVal	1 - старший байт
UiValue	2 - младшее слово (два байта) 3 - старшее слово (два байта)
Data[0]	4
Data[1]	5
Data[2]	6
Data[3]	7

3.14.3.2. STOPMODBUSSLAVE

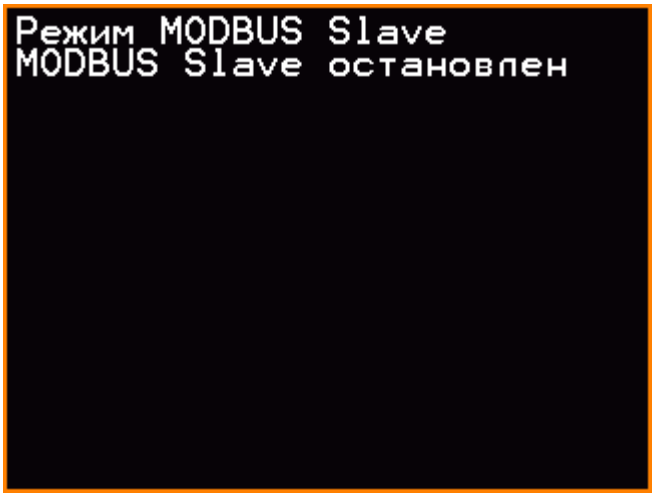
```
#include <serial.h>

void StopModbusSlave(void);
```

Функция *StopModbusSlave* завершает режима работы MODBUS Slave контроллера CAT-500.

Пример:

```
#include <sat500.h>
#include <serial.h>

short int Reg[32];
void main(void)
{
    ClrScr();                   //Очистить экран
    BigFont=0;                  //Обычный шрифт 12x16 (не крупный)
    //Открыть RS-485
    SerialOpen(19200,0,1);      //Скорость 19200, чётность - нет, стопбитов - 1
    if(SerialIsOpen())
    {
        //Успешно открыт
        //Заполнить читаемые регистры
        for(int i=0;i<32;i++)Reg[i]=i;
        //Адрес Slave MODBUS устройства (CAT-500)
        ModbusSlave=1;
        //Начать режим MODBUS Slave контроллера CAT-500
        //32 регистра хранения в переменных Reg
        //Они же являются и входными регистрами
        StartModbusSlave(Reg,32,Reg,32);
        puts("Режим MODBUS Slave");
        ShowScr();                  //Вывести на дисплей
        EnableKeyboard=1;           //Разрешить программе работу с клавиатурой
        //Пока не нажата кнопка - ничего не делаем
        while(!GetKey())
        {
        }
        //Завершение режима MODBUS Slave контроллера CAT-500
        puts("\nMODBUS Slave остановлен");
        EnableKeyboard=0;
    }
    else
    {
         printf("Ошибка открытия...");
    }
    ShowScr();                   //Вывести на дисплей
    while(1);                    //Бесконечная пауза
}
```

Результат:



3.14.3.3. MODBUSCOUNT

```
#include <serial.h>

unsigned int ModbusCount;
```

Переменная содержит счётчик корректных запросов MODBUS RTU Slave, на которые был выдан ответ.
Для очистки переменной можно записать в неё значение:

```
ModbusCount=0;
```

3.15. КЛИЕНТ LANMON СЕРВЕРА

Контроллер CAT-500 может выступать в качестве клиента сервера LanMon.
Работа с сервером заключается в подключении к серверу, передаче значений *информационных каналов*, и получении значений каналов от сервера.
С основными принципами работы с сервером LanMon можно познакомится в разделе *Сервер Lan-Mon*.
Большим преимуществом клиента сервера LanMon является то, что он устанавливает исходящее TCP подключение к серверу, и соответственно возможно подключение к глобальному IP адресу из локальной подсети.
Для работы с сервером LanMon контроллер имеет ряд функций и переменных, которые позволяют выполнить следующие действия:
1. Подключение к серверу LanMon и авторизация на сервере.
2. Передача на сервер значений каналов по текстовому имени канала.
3. Передача на сервер значений каналов по номеру канала.
4. Регистрация новых каналов на сервере.
5. Получения управляющих значений каналов от сервера.

Доступны следующие переменные для работы в качестве клиента сервера LanMon:

Переменная	***Описание***
unsigned char LmState	Переменная *LmState* возвращает состояние клиента сервера LanMon
char LmConnected	Переменная *LmConnected* возвращает состояние подключение клиента к серверу LanMon

unsigned int LmChanNum	Переменная *LmChanNum* возвращает номер зарегистрированного канала на сервере после успешной регистрации канала при помощи функции *LmChanRegister*
char LmChanActivity	Переменная *LmChanActivity* возвращает активность зарегистрированного канала на сервере после успешной регистрации канала при помощи функции *LmChanRegister*

Доступны следующие функции для работы в качестве клиента сервера LanMon:

Функция	***Описание***
*char LmOpen (char * ip,unsigned short int port,unsigned int timeout,char * login,char * password,unsigned short int ver)*	Функция включает в работу встроенного клиента сервера LanMon, начинается подключение к серверу. После вызова этой функции встроенный клиент самостоятельно поддерживает связь с сервером, выполняет все нужные действия по авторизации на сервере и переподключение в случае утери связи с сервером
void LmClose (void)	Функция завершает работу встроенного клиента сервера LanMon и производит отключение от сервера
void LmSendByNum (unsigned int chnum,int value,int div,char quality)	Функция позволяет послать значение канала на сервер. В качестве в качестве адреса используется номер канала на сервере *chnum*. Канал должен существовать на сервере и быть активным
*void LmSendByName (char * chname,int value,int div,char quality)*	Функция позволяет послать значение канала на сервер. В качестве в качестве адреса используется текстовое имя канала на сервере *chname.* . Канал должен существовать на сервере и быть активным
*void LmChanRegister (char * chname,char * unit,char * fmt,char * cmnts,char ctrl)*	Функция позволяет выполнить регистрацию канала на сервере и получить номер канала на сервере. Функция может создавать новый канал на сервере если он отсутствует
*char LmGetCtrlName (char * chname)*	Функция позволяет прочитать имя канала из сообщения полученного от сервера LanMon о новом значении канала
unsigned int LmGetCtrlValue (int mul)	Функция возвращает новое значение канала, полученное от сервера LanMon. Предварительно должна быть вызвана функция *LmGetCtrlName*

Для подключения к серверу LanMon необходимо в приложении сервера выполнить *настройку авторизации*.
В некоторых случаях может потребоваться *ручное создание каналов на сервере*.

3.15.1. СЕРВЕР LANMON

Канал сервера

Канал сервера LanMon представляет собой переменную для хранения какого-либо параметра. Существует два типа каналов:
Канал тип 1 - адресация каналов этого типа выполняются с помощью 4-х чисел, и принято записывать адрес в следующем виде: 2.45.2390.1804 . Контроллер CAT-500 не имеет встроенных функций для работы с каналами первого типа.
Канал тип 2 - адресация каналов этого типа выполняются с помощью текстовой строки длиной не более 63 символов. Например, адрес может быть таким: "temp1_58302". Контроллер CAT-500 имеет встроенные функции для работы с каналами второго типа. Имеется альтернативный способ адресации канала второго типа - вместо текстовой строки используется номер канала на сервере - четырехбайтовое беззнаковое число.

Каждый канал на сервере имеет собственный **тип**, например, канал может быть целочисленным числом (размером 1 байт, 2 байта, 4 байта, со знаком или без знака), плавающим числом (одинарной или двойной точности), текстовой строкой, массивом и т.д. . Контроллер CAT-500 умеет работать только с каналами типа плавающее число двойной точности.
Кроме собственно значения, каждой канал может иметь дополнительные свойства, которые принято называть **атрибутами**. Контроллер CAT-500 работает со следующими атрибутами:
- *Единица измерения* - текстовая строка, описывающая единицы измерения переменной значения канала.
- *Формат* - текстовая строка, показывающая способ вывода значения канала на экран (в формате С-функции printf).
- *Комментарий* - текстовая строка, описывающая переменную данного канала.
Эти атрибуты могут только передаваться при регистрации канала на сервере.
Контроллер CAT-500 может не только передавать значения каналов на сервер, но также может получать значения каналов от сервера для управления.
Получения атрибутов канала от сервера не поддерживаются.
Также каналы обладают свойством "Активность". Активные каналы принимаются сервером и пересылаются рабочим местам разрабатываемой системы.
Неактивные каналы сервером не принимаются, и не используются в системе.
Все каналы, передаваемые контроллером CAT-500, следует делать активными.

Качество канала

Каждый канал сервера имеет дополнительное значение, которое называется **качеством** канала.
Нулевое значение качество канала говорит о том что значение самого канала достоверно.
Ненулевое значение качества говорит о том, что при получении параметра, отображаемого в канале произошла ошибка.
В таблице ниже показаны возможные значения качества (Quality) канала:

Значение Quality	Стандартное наименование	Пояснение
0	OK	Все работает. Поле значение канала (Value) достоверно. При всех других значениях Quality значение канала (Value) НЕ достоверно.
1	Выключен	Сознательно не опрашивается контроллером. Например: датчик временно отключен по причине неисправности.
2	Состояние не определено	Нет данных о состоянии канала. Например: драйвер устройства не загружен или опросчик - владелец данного канала не подключен к серверу или произошел запуск программы, но информация о состоянии датчиков еще не поступила.
3	Неисправен датчик	Неисправен датчик или устройство — источник первичной информации. (аппаратный уровень 1)
4	Неисправен контроллер	Неисправен контроллер, производящий первичную обработку сигнала от датчика или адресный расширитель. (аппаратный уровень 2)
5	Значение недостоверно	Показания датчика вышли за допустимые пределы измерения.
6	Датчик не подключен	Нарушение линии связи с датчиком. Например обрыв.
7	Нет связи	Неисправность канала связи между сервером (или регистратором) и контроллером. Кроме случая нарушения линии связи контроллера с датчиком (Quality=6). Для каналов тип 1 не поддерживается.
8	Неисправен регистратор	Неисправен регистратор, производящий сбор информации от контроллеров. (аппаратный уровень 3) Для каналов тип 1 не поддерживается.

Подключение к серверу

Для подключения к серверу LanMon контроллер CAT-500 устанавливает TCP соединение.
Для подключения необходимо указать адрес сервера, номер сетевого порта сервера, логин и пароль для подключения.
Для подключения к серверу LanMon используется функция:
*char LmOpen(char * ip,unsigned short int port,unsigned int timeout,char * login,char * password,unsigned short int ver)*
После вызова данной функции включается встроенный клиент, который обеспечивает подключение к серверу LanMon.
Этот клиент умеет проверять работоспособность канала подключения, и в случае необходимости автоматически выполняет повторное подключение.
Программист может проверять наличие авторизованного подключения, используя встроенную переменную *LmConnected*.
Программист может контролировать состояние подключения, используя встроенную переменную *LmState*.
Для завершения работы встроенного клиента сервера используется функция:
void LmClose(void)

Создание каналов на сервере

Контроллер CAT-500 может передавать значения в уже существующие каналы на сервере, а также может создавать новые каналы на сервере.
Если контроллер работает с уже существующими каналами на сервере, то следует выполнить следующие условия:
1. Канал должен быть типа плавающие число двойной точности
2. Канал должен быть активен на сервере
Для создания каналов на сервере используется специальная функция регистрации канала:
*void LmChanRegister(char * chname,char * unit,char * fmt,char * cmnts,char ctrl)*
Допускается повторная регистрация уже существующих каналов на сервере.
В процессе регистрации или создания канала на сервере можно получить номер канала на сервере, используя переменную *LmChanNum*, и активность канала, используя переменную *LmChanActivity*.

Передача значения канала

Контроллер CAT-500 может передавать значение канала в любой момент времени, используя любую из двух следующих функций:
1. Запись значения канала с адресацией по номеру канала на сервере:
void LmSendByNum(unsigned int chnum,int value,int div,char quality)
2. Запись значения канала с адресацией по имени канала на сервере:
*void LmSendByName (char * chname,int value,int div,char quality)*
Обе функции выполняют посылку значения канала на сервер без какой либо фильтрации по значению.
Поэтому программист должен самостоятельно вести необходимое прореживание передачи канала по времени, или фильтрацию по значению, для того чтобы избежать слишком частой передачи каналов на сервер.
Можно рекомендовать передачу каналов на сервер только по изменению значения передаваемого параметра и не чаще чем раз в секунду.
Если сервер сообщил о том, что канал неактивен, то посылка значения на сервер не имеет смысла - значение канала на сервере обновляться не будет.

Сценарий работы без создания каналов

Наиболее простой вариант работы с сервером LanMon состоит в подключении к серверу, проверке наличия подключения, и передачи каналов на сервер. Для такой работы используются следующие функции и переменные:
1. Подключение - функция *LmOpen*.
2. Проверка наличия подключения - переменная *LmConnected*.

3. Передача значений каналов на сервер по мере необходимости, используя любую из функций - *LmSendByName* и *LmSendByNum*.
ВНИМАНИЕ! Каналы уже должны быть созданы на сервере LanMon и они должны быть активны!

Сценарий работы с созданием каналов

Более сложный вариант работы с сервером LanMon состоит в подключении к серверу, проверке наличия подключения, регистрации каналов на сервере, и передаче каналов на сервер. Для такой работы используются следующие функции и переменные:
1. Подключение - функция *LmOpen*.
2. Проверка наличия подключения - переменная *LmConnected*.
3. Регистрация каналов на сервере с помощью функции *LmChanRegister*, получении номера канала на сервере с помощью переменной *LmChanNum* и получении активности канала на сервере с помощью переменной *LmChanActivity*.
4. Передача значений активных каналов на сервер по мере необходимости, используя любую из функций - *LmSendByName* и *LmSendByNum*.
Допускается вызывать функцию регистрации *LmChanRegister* даже если каналы уже имеются на сервере.

Получение значения канала от сервера

При создании канала на сервере LanMon можно задать признак, разрешающий передачу значения канала от сервера LanMon к контроллеру CAT-500.
Каналы, разрешенные для управления, могут получать значения от сервера LanMon. Изменение этих каналов выполняется в автоматизированных рабочих местах системы LanMon.
Для получения значения канала от сервера приложение должно периодически просматривать значение переменной состояния сервера *LmState*, и как только значение покажет наличие управления от сервера (7 - Получено управление от сервера) вызывать функцию получения названия канала - *LmGetCtrlName* и функцию получения значения канала от сервера - *LmGetCtrlValue*.
Вместо проверки переменной состояния сервера *LmState* приложение может постоянно вызывать функцию получения названия канала - *LmGetCtrlName* и если функция возвращает ***true***, то получено управление от сервера, и вызовом функции уже получено имя канала, которому прислано управление. Далее следует прочитать новое значение канала вызовом функции функции получения названия канала - *LmGetCtrlValue* и выполнить действия по установки нового значения канала.

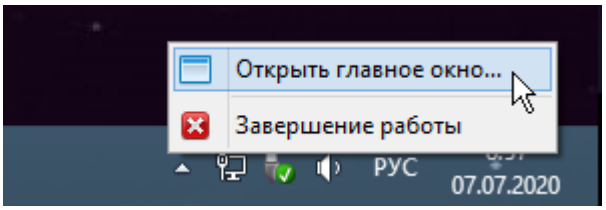
3.15.1.1. УСТАНОВКА СЕРВЕРА

Если у вас имеется приобретённая лицензия на сервер LanMon, то воспользуйтесь следующей ссылкой для загрузки сервера:
ftp://ftp.mnppsaturn.ru/public/soft/lmserver/lmserver.zip
Также вы можете загрузить бесплатную версию LanMon сервера, которая называется персональная. В этой версии у вас может быть только один опросчик (например контроллер CAT-500) и одно рабочее место АРМ LanMon.
Для загрузки персонального сервера за воспользуйтесь следующей ссылкой:
ftp://ftp.mnppsaturn.ru/public/soft/lmserver/lmserver_personal.zip
Установка сервера не имеет особенностей - следуйте инструкциям установщика во время инсталляции.
Дальнейшее описание предполагает что сервер установлен как приложение.
С кратким описанием сервера системы LanMon можно познакомиться на сайте производителя:
http://www.mnppsaturn.ru/?topic_id=3&good_id=184

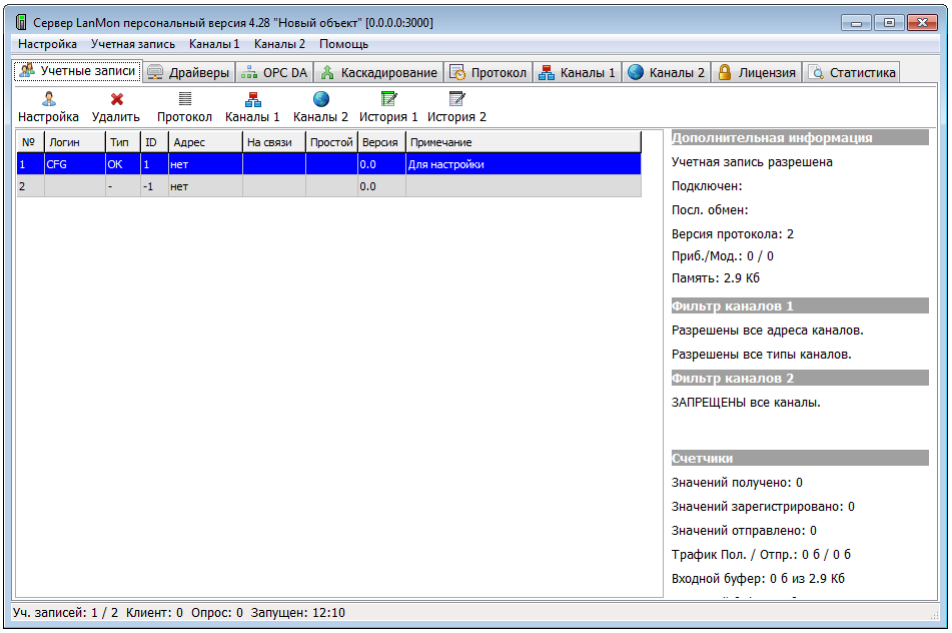
3.15.1.2. ВЫЗОВ ОКНА ПРИЛОЖЕНИЯ СЕРВЕРА

Если сервер запущен как приложение, то в статусной строке Windows отображается иконка приложения, которая имеет следующий вид:

Для открытия основного окна приложения необходимо щёлкнуть по иконке приложения правой кнопкой мышки и в открывшемся контекстном меню выбрать пункт "Открыть главное окно":

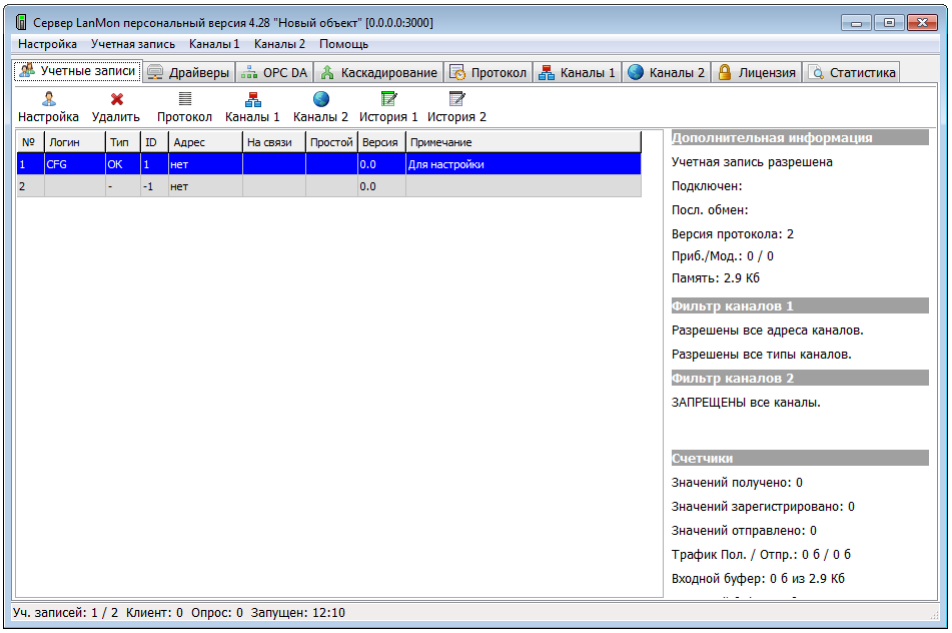


Откроется основное окно приложения который выглядит следующим образом:

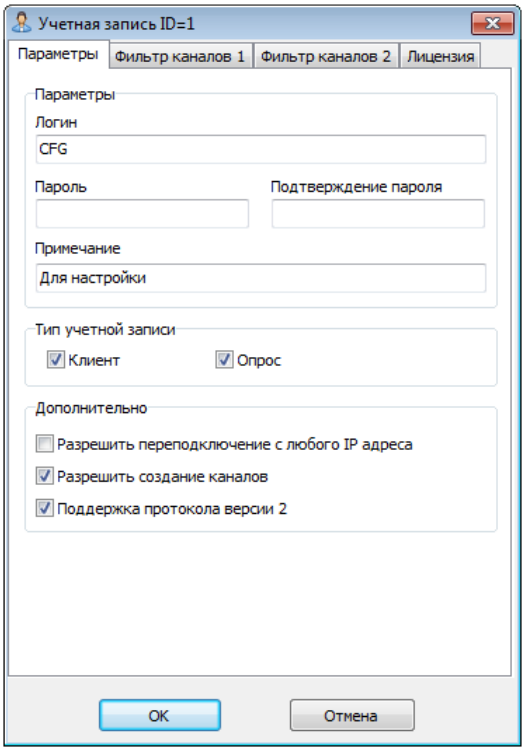


3.15.1.3. НАСТРОЙКА АВТОРИЗАЦИИ

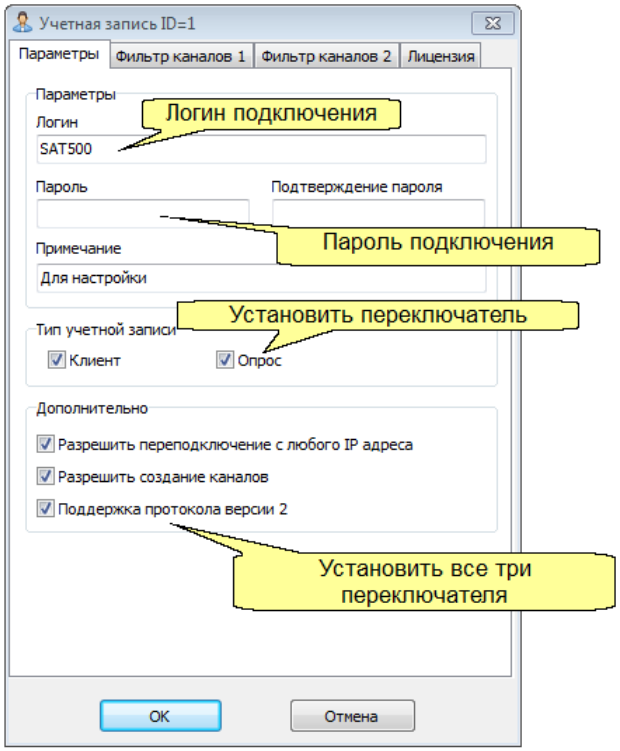
Для настройки получения CAT-500 к серверу необходимо открыть вкладку "Учетные записи" приложения сервера:



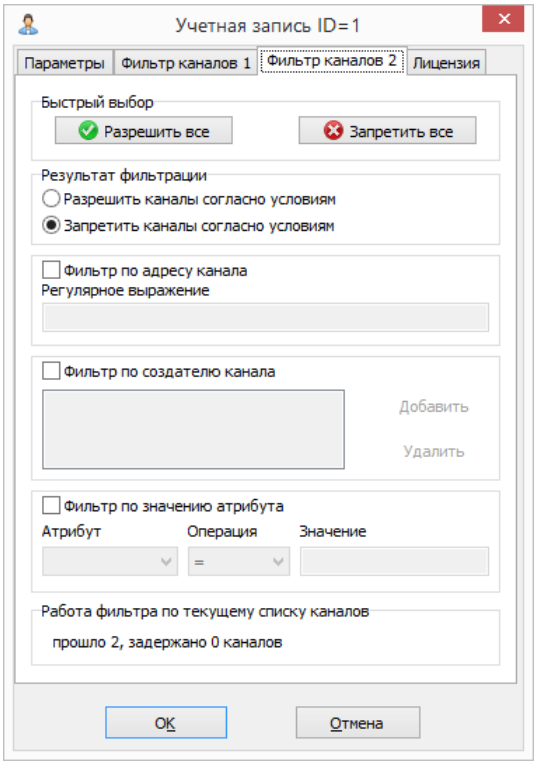
Далее необходимо выполнить двойной клик мышкой по строке учетной записи, которая будет использоваться для подключения контроллера CAT-500.
Откроется окно настройки учетной записи:



Необходимо ввести логин подключения в поле ввода "Логин". Этот плагин будет использоваться контроллером CAT-500.
Необходимо ввести пароль и подтверждение пароля в одноименных полях ввода. Этот пароль будет использоваться для подключения контроллером CAT-500.
Переключатель "Опрос" в поле "Тип учетной записи" обязательно должен быть установлен.
В поле ввода "Примечание" можно вести комментарий к учетной записи.
В поле "Дополнительно" следует установить все три переключателя.



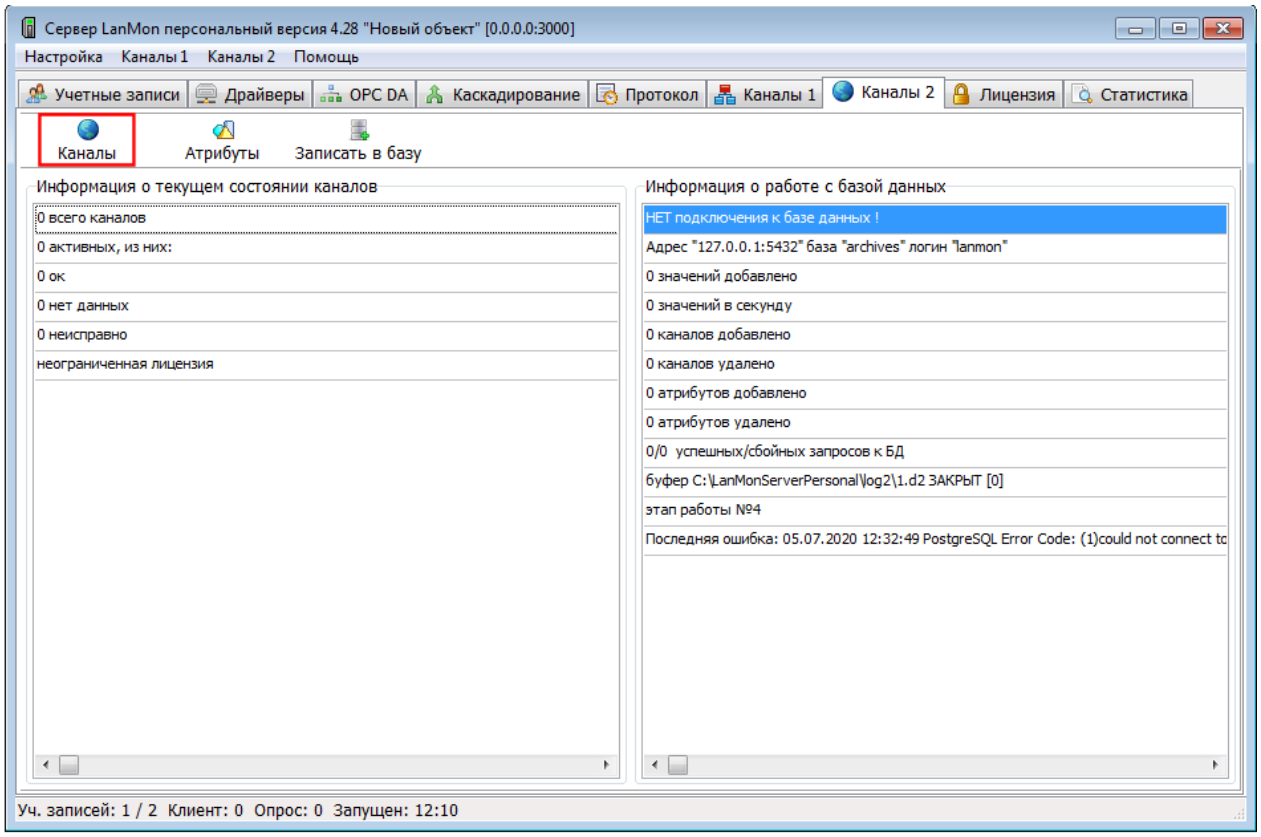
Далее следует перейти на вкладку фильтр каналов 2:



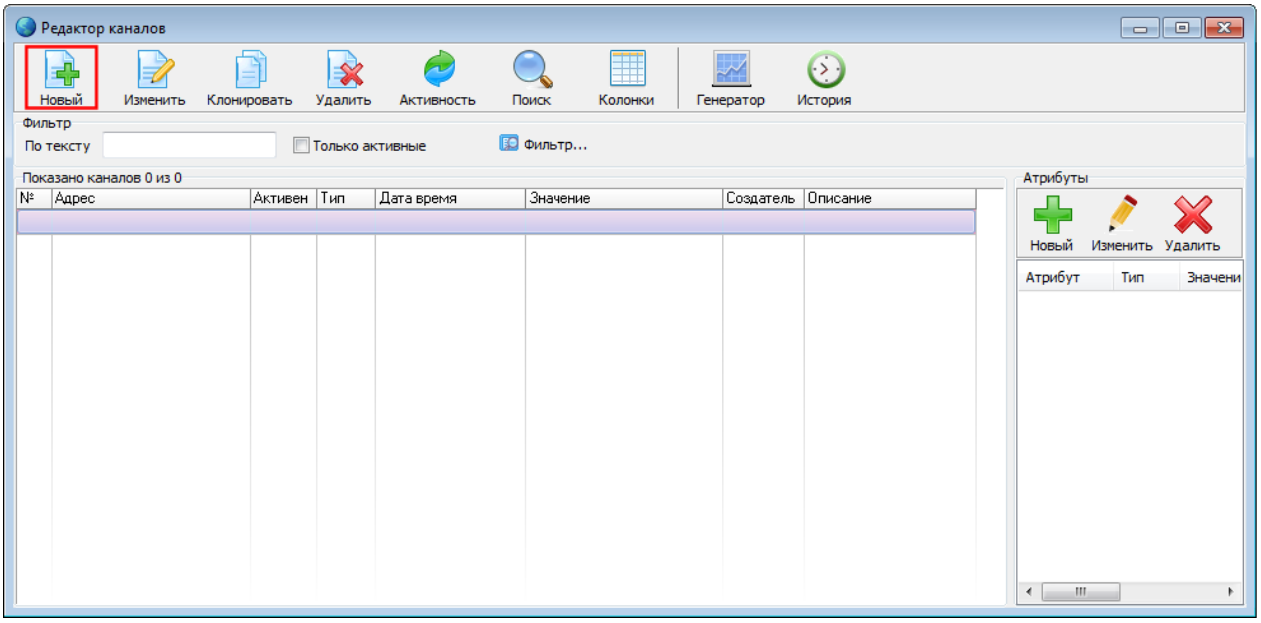
Для разрешения создания каналов следует нажать кнопку "Разрешить все".
Для завершения настройки учетной записи следует нажать кнопку "OK".

3.15.1.4. ДОБАВЛЕНИЕ КАНАЛОВ

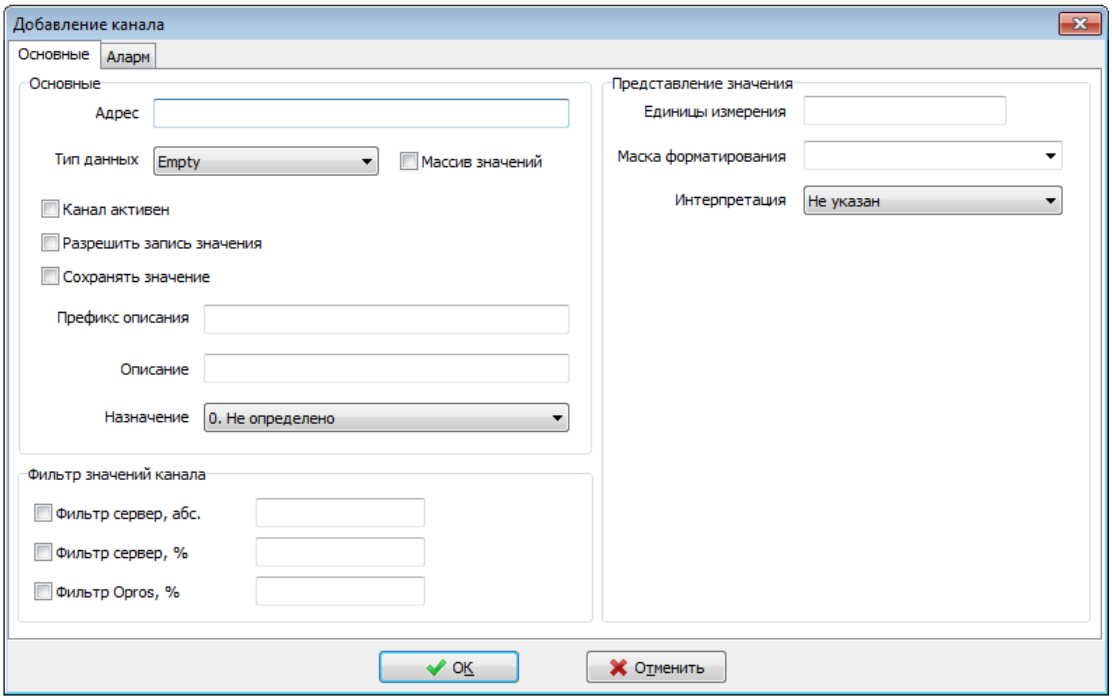
Для ручного добавления каналов в приложении сервер LanMon необходимо перейти на вкладку "Каналы 2":



И нажать на кнопку "Каналы". Откроется окно со списком каналов второго типа:



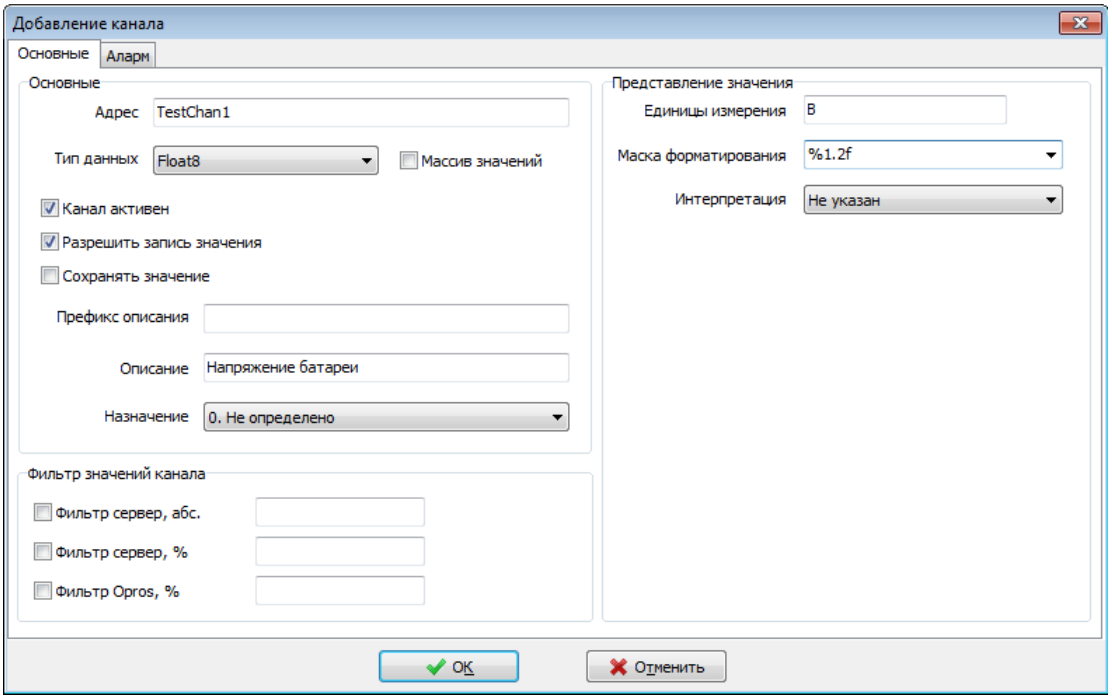
Для создания канала необходимо нажать на кнопку "Новый".
Откроется окно "Добавление канала":



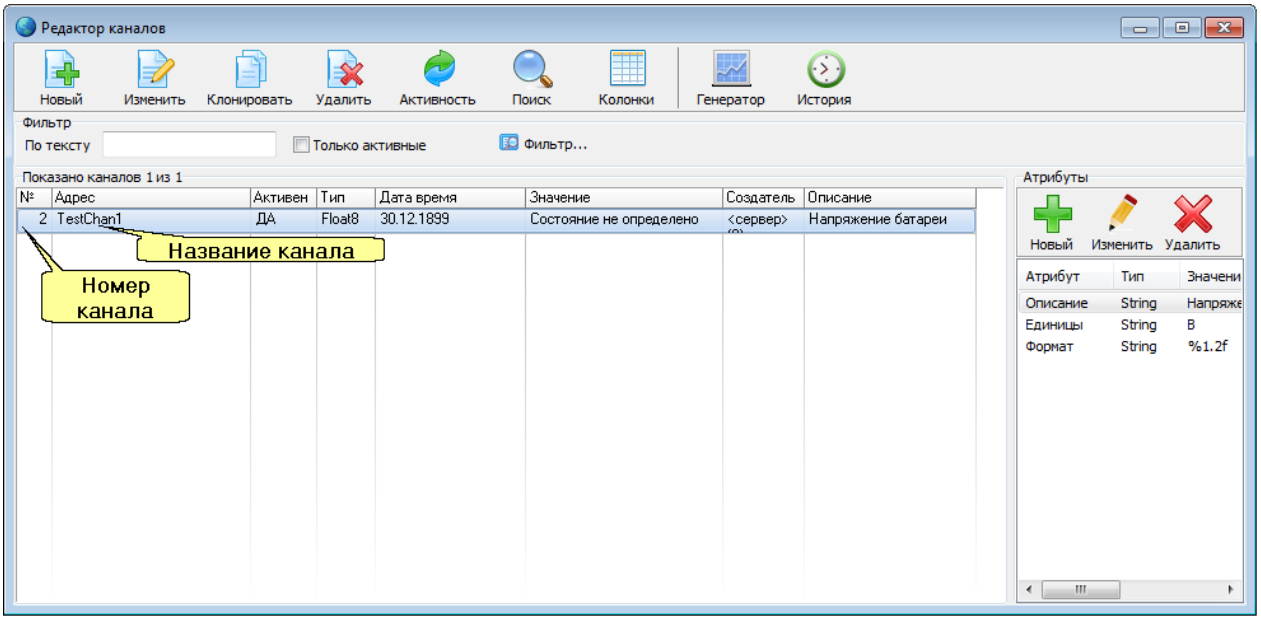
Необходимо ввести следующие данные:

Поле	Пояснение
Основные/Адрес	Текстовое имя канала длиной не более 25 символов
Основные/Тип данных	Выбрать значение Float8
Основные/Канал активен	Установить переключатель
Основные/Разрешить запись значения	Установить переключатель
Основные/Описание	Текстовое описание канала (комментарий)
Представление значения/ Единицы измерения	Текстовое описание единиц измерения параметра канала
Представление значения/ Маска форматирования	Маска форматирования при выводе значения канала на экран

На рисунке ниже показан пример добавления канала с именем "TestChan1":

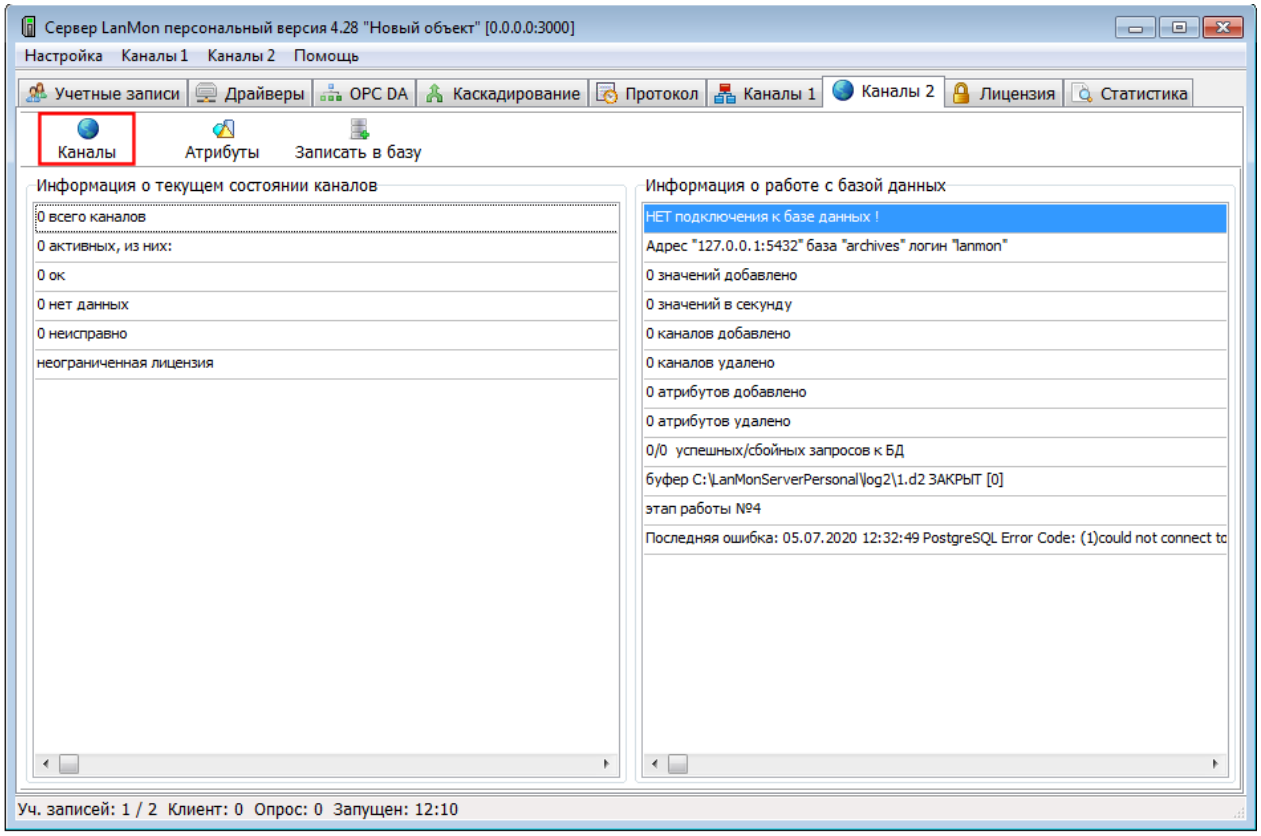


Для добавления канала необходимо нажать кнопку "ОК".
В списке каналов появился новый канал, который имеет номер 2. Контроллер CAT-500 может обращаться к этому каналу по имени "TestChan1" или по числовому номеру 2.

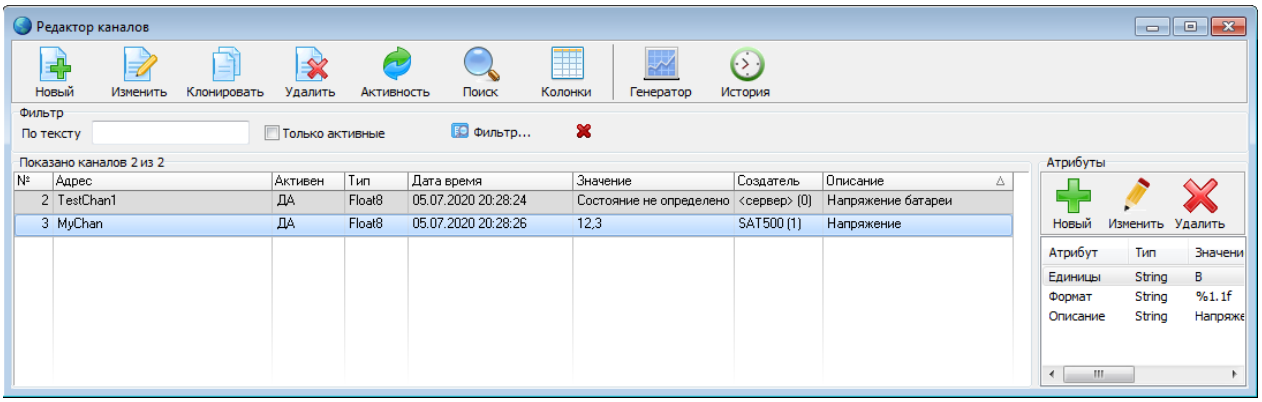


3.15.1.5. ПРОСМОТР КАНАЛОВ

Для просмотра каналов в приложении сервер LanMon необходимо перейти на вкладку "Каналы 2":

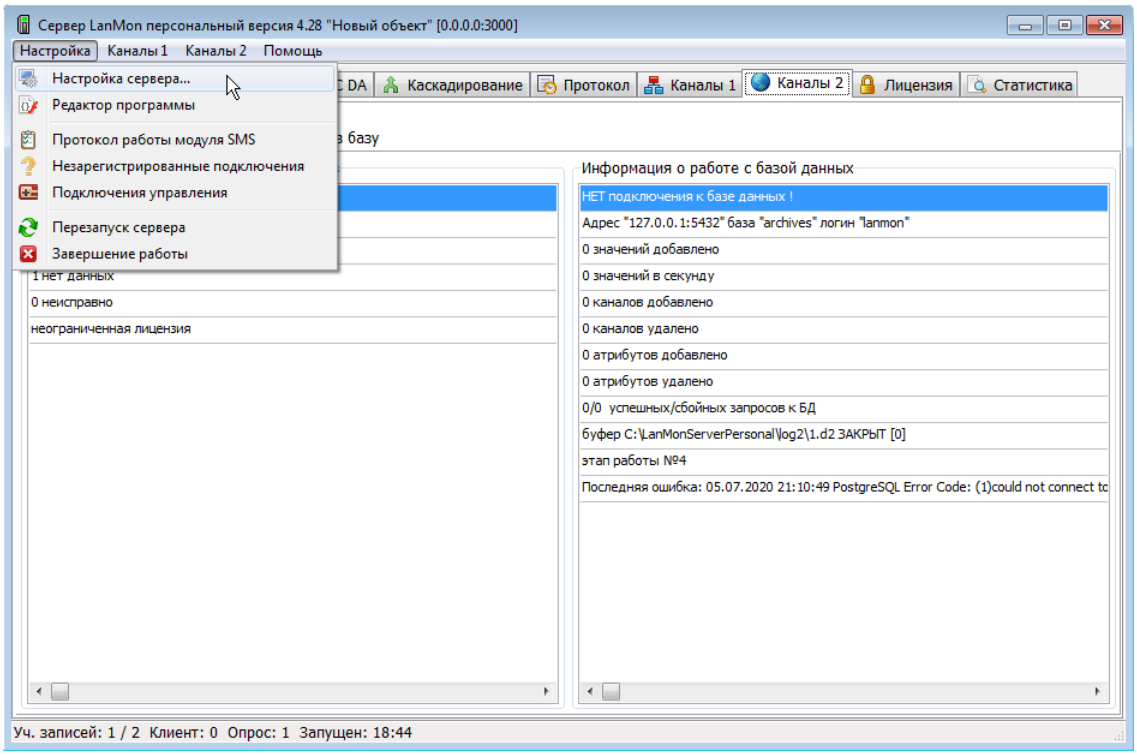


И нажать на кнопку "Каналы". Откроется окно со списком каналов типа 2:

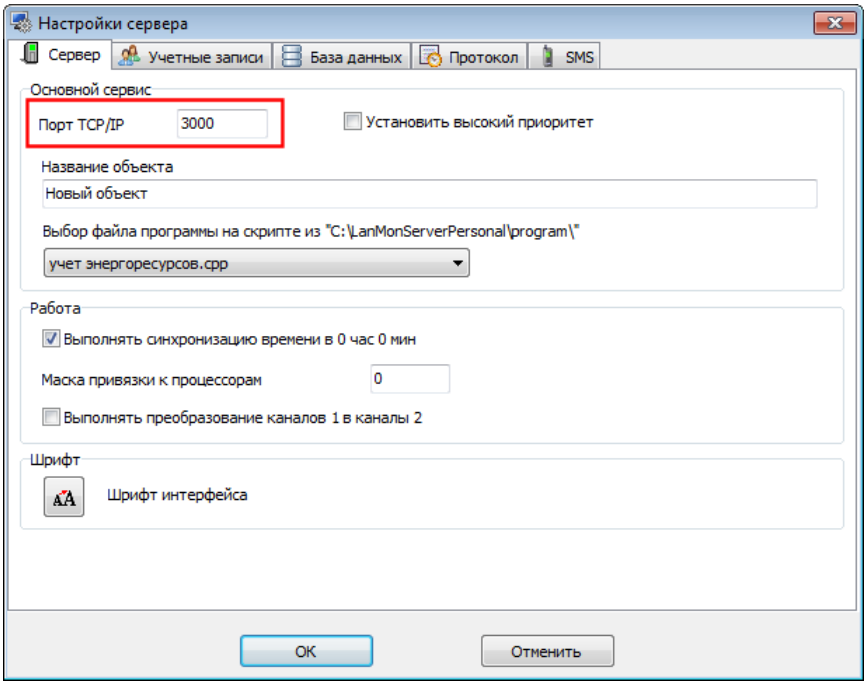


3.15.1.6. НАСТРОЙКА НОМЕРА ПОРТА СЕРВЕРА

Для настройки номера порта сервера LanMon необходимо в основном меню окна сервера выбрать пункт "Настройка/Настройка сервера":



В открывшемся окне на вкладке "Сервер" можно задать сетевой номер порта. По умолчанию значение порта 3000.



3.15.2. LMCHANACTIVITY

```
#include <lanmon.h>

char LmChanActivity;
```

Переменная *LmChanActivity* возвращает активность зарегистрированного канала на сервере после успешной регистрации канала при помощи функции *LmChanRegister*.
Нулевое значение переменной *LmChanActivity* говорит о том, что сервер не принимает значения данного канала, значение можно посылать, но сервер отбрасывает полученное значение.
Ненулевое значение активности сообщает, что сервер принимает значение канала и рассылает это значение другим клиентам сервера.
Переменная доступна только для чтения.

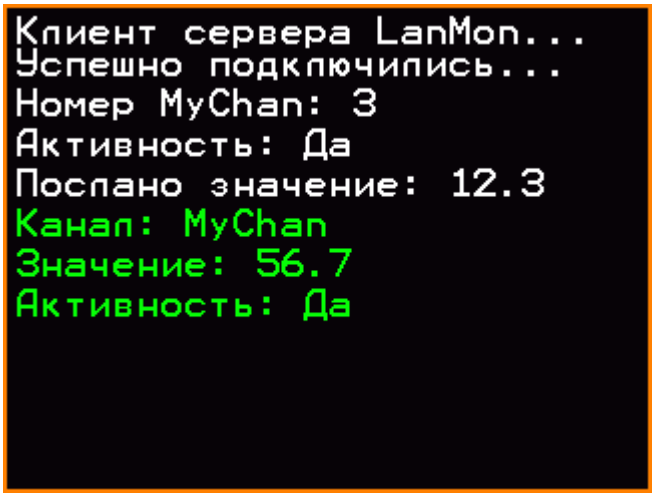
В приведенном далее примере на сервере создается новый канал с именем "MyChan" и выполняется получение активности и номера канала на сервере.
В дальнейшем значение канала можно записывать как по имени так и по номеру.
В примере выполняется запись вновь созданного канала по полученному номеру, если канал активен.
После записи значения канала на сервер программа непрерывно ожидает нового значения канала от сервера.
При получении нового значения канала на дисплей выводится имя канала и значение полученное от сервера.

Пример:

```
#include <sat500.h>
#include <lanmon.h>

typedef  unsigned int UINT;
UINT ChanNum;
char ChanActivity;

void main(void)
{
Restart:;
    ClrScr();
    BigFont=false;
    printf("Клиент сервера LanMon...");
    ShowScr();
    LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
    //Ожидаем подключения
    while(!LmConnected);
    //Подключены
    GotoXY(4,20);
    puts("Успешно подключились...");
    ShowScr();
    //Регистрация канала "MyChan" на сервере
    //Канал "MyChan" может изменяться сервером
    LmChanRegister("MyChan","B","%1.1f","Напряжение",true);
    //Ожидание регистрации 60 секунд
    UINT t=GetTickCount();
    while((GetTickCount()-t)<60000)
    {
        if(LmState==lmsREGISTERED)break;
    }
    if(LmState!=lmsREGISTERED)goto Restart;
    //Канал зарегистрирован
    //Запомнить активность канала "MyChan" на сервере
    ChanActivity=LmChanActivity;
    //Запомнить номер канала "MyChan" на сервере
    ChanNum=LmChanNum;
    GotoXY(4,40);
    printf("Номер MyChan: %u",ChanNum);
```

```
    GotoXY(4,60);
    char * s="Нет";
    if(ChanActivity)s="Да";
    printf("Активность: %s",s);
    ShowScr();
    if(ChanActivity)
    {
        //Канал активен - можно посылать значение на сервер
        //Послать значение канала "MyChan" (12.3) на сервер по номеру
        int value=123;
        LmSendByNum(ChanNum,value,10,0);
        GotoXY(4,80);
        printf("Послано значение: %d.%1d",value/10,value%10);
    }
    ShowScr();
    //Ожидаем изменения канала от сервера
    while(1)
    {
        if(LmState==lmsCTRL)
        {
            //Получено новое значение канала
            FontColor=COLOR_GREEN;
            char name[64];
            //Получить имя канала
            if(LmGetCtrlName(name))
            {
                GotoXY(4,100);
                if(name[0])
                {
                    printf("Канал: %s    ",name);
                }
                else
                {
                    printf("Номер канала: %u     ",ChanNum);
                }
                //Получить новое значение канала
                UINT value=LmGetCtrlValue(10);
                GotoXY(4,120);
                printf("Значение: %d.%1d  ",value/10,value%10);
                GotoXY(4,140);
                s="Нет";
                if(ChanActivity)s="Да";
                printf("Активность: %s",s);
                ShowScr();
            }
        }
    }
}
```

Результат:



3.15.3. LMCHANNUM

```
#include <lanmon.h>

unsigned int LmChanNum;
```

Переменная *LmChanNum* возвращает номер зарегистрированного канала на сервере после успешной регистрации канала при помощи *LmChanRegister*.
Нулевое значение переменной *LmChanNum* говорит об отсутствии номера канала (недопустимое значение номера канала).
Переменная доступна только для чтения.

В приведенном далее примере на сервере создается новый канал с именем "MyChan" и выполняется получение номера канала на сервере.
В дальнейшем значение канала можно записывать как по имени так и по номеру.
В примере выполняется запись вновь созданного канала по полученному номеру.
После записи значения канала на сервер программа непрерывно ожидает нового значения канала от сервера.
При получении нового значения канала на дисплей выводится имя канала и значение полученное от сервера.

Пример:

```
#include <sat500.h>
#include <lanmon.h>

typedef  unsigned int UINT;
//Номер канала на сервере
UINT ChanNum;
//Активность канала на сервере
char ChanActivity;

void main(void)
{
Restart:;
    ClrScr();
    BigFont=false;
    printf("Клиент сервера LanMon...");
    ShowScr();
```

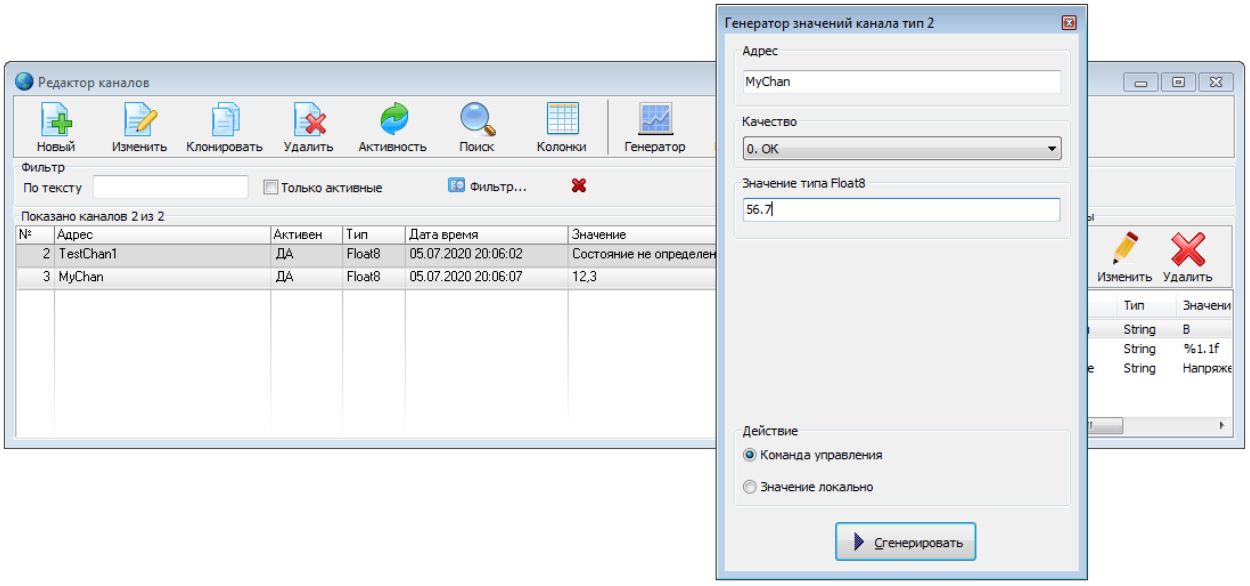
```
LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
//Ожидаем подключения
while(!LmConnected);
//Подключены
GotoXY(4,20);
puts("Успешно подключились...");
ShowScr();
//Регистрация канала "MyChan" на сервере
//Канал "MyChan" может изменяться сервером
LmChanRegister("MyChan","B","%1.1f","Напряжение",true);
//Ожидание регистрации 60 секунд
UINT t=GetTickCount();
while((GetTickCount()-t)<60000)
{
    if(LmState==lmsREGISTERED)break;
}
if(LmState!=lmsREGISTERED)goto Restart;
//Канал зарегистрирован
//Запомнить активность канала "MyChan" на сервере
ChanActivity=LmChanActivity;
//Запомнить номер канала "MyChan" на сервере
ChanNum=LmChanNum;
GotoXY(4,40);
printf("Номер MyChan: %u",ChanNum);
GotoXY(4,60);
char * s="Нет";
if(ChanActivity)s="Да";
printf("Активность: %s",s);
ShowScr();
//Послать значение канала "MyChan" (12.3) на сервер по номеру
int value=123;
LmSendByNum(ChanNum,value,10,0);
GotoXY(4,80);
printf("Послано значение: %d.%1d",value/10,value%10);
ShowScr();
//Ожидаем изменения канала от сервера
while(1)
{
    if(LmState==lmsCTRL)
    {
        FontColor=COLOR_YELLOW;
        //Получено новое значение канала
        char name[64];
        //Получить имя канала
        if(LmGetCtrlName(name))
        {
            GotoXY(4,100);
            if(name[0])
            {
                printf("Канал: %s    ",name);
            }
            else
            {
                printf("Номер канала: %u     ",ChanNum);
            }
            //Получить новое значение канала
            value=LmGetCtrlValue(10);
            GotoXY(4,120);
```

```
                    printf("Значение: %d.%1d  ",value/10,value%10);
                    GotoXY(4,140);
                    s="Нет";
                    if(ChanActivity)s="Да";
                    printf("Активность: %s",s);
                    ShowScr();
                }
            }
        }
}
```

Результат:

```
Клиент сервера LanMon...
Успешно подключились...
Номер MyChan: 3
Активность: Да
Послано значение: 12.3
Канал: MyChan
Значение: 12.9
Активность: Да
```

В окне "Редактор каналов" сервера LanMon можно послать новые значения канала используя кнопку "Генератор":



3.15.4. LMCHANREGISTER

```
#include <lanmon.h>
```

```
void LmChanRegister(char * chname,char * unit,char * fmt,char * cmnts,char
ctrl);
```

Функция *LmChanRegister* позволяет выполнить регистрацию канала на сервере и получить номер канала и активность канала на сервере.
Регистрация - это процесс создания канала на сервере.
Если канал на сервере уже существует, то будет выполнено обновление атрибутов существующего канала на сервере (*Единица измерения, Формат, Комментарий*).
Т.е. вызывать функцию регистрации канала безопасно в любом случае, независимо от того существует канал на сервере, или нет.
Аргументы функции:
*char * chname* - текстовое имя канала на сервере (непустая строка длиной не более 63 символов).
*char * unit* - текстовая строка описывающая единицы измерения переменной значения канала.
*char * fmt* - текстовая строка, показывающая способ вывода значения канала на экран (в формате С-функции printf).
*char * cmnts* - текстовая строка описывающая переменную данного канала.
char ctrl - разрешение управления канала от сервера. 0 соответствует запрещению управления,1 соответствует разрешению управление канала от сервера (сервер может присылать значение каналу).

Функция выполняется только, если имеется авторизованное подключение к серверу LanMon *(LmConnected == true)* и задано непустое имя канала.
В случае успешного волнения выполнение данной функции на сервер посылается сообщение о регистрации и состояние клиента *LmState* изменяется на значение 5 (*lmsREGISTERING* - Выполняется регистрация канала).
Как только сервер подтверждает регистрацию канала посылкой ответного сообщения, состояние клиента *LmState* изменяется на значение 6 (*lmsREGISTERED* - Канал зарегистрирован) и из переменной *LmChanNum* можно прочитать номер зарегистрированного канала сервере, а из переменной *LmChanActivity* прочитать активность канала на сервере.

В приведенном далее примере на сервере создается новый канал с именем "MyChan" и выполняется получение номера канала на сервере.
В дальнейшем значение канала можно записывать как по имени так и по номеру.
В примере выполняется запись вновь созданного канала по полученному номеру.

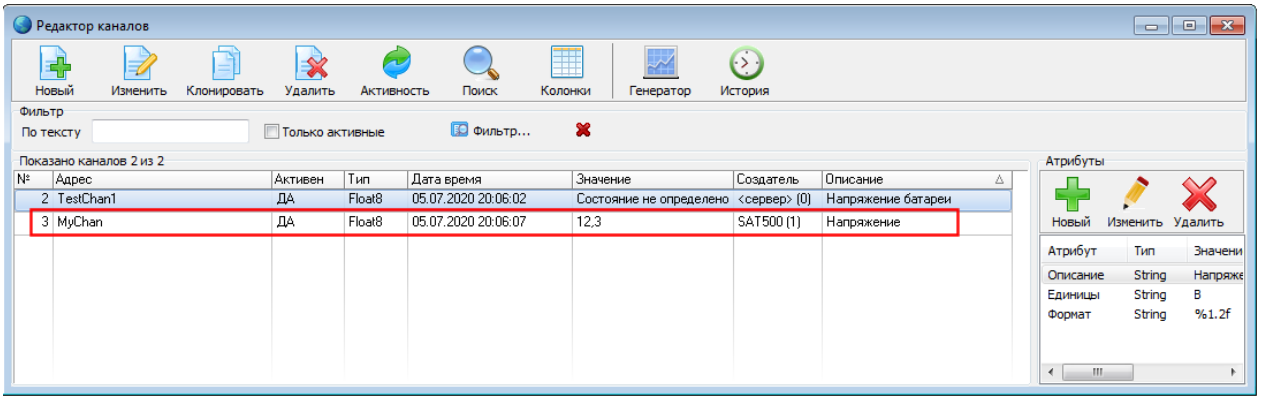
Пример:

```
#include <sat500.h>
#include <lanmon.h>

typedef  unsigned int UINT;
UINT ChanNum;

void main(void)
{
Restart:;
    ClrScr();
    BigFont=false;
    printf("Клиент сервера LanMon...");
    ShowScr();
    LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
    //Ожидаем подключения
    while(!LmConnected);
    //Подключены
    GotoXY(4,20);
    puts("Успешно подключились...");
    ShowScr();
    //Регистрация канала "MyChan" на сервере
```

```
    //Канал "MyChan" может изменяться сервером
    LmChanRegister("MyChan","В","%1.1f","Напряжение",true);
    //Ожидание регистрации 60 секунд
    UINT t=GetTickCount();
    while((GetTickCount()-t)<60000)
    {
        if(LmState==lmsREGISTERED)break;
    }
    if(LmState!=lmsREGISTERED)goto Restart;
    //Запомнить номер канала "MyChan" на сервере
    ChanNum=LmChanNum;
    GotoXY(4,40);
    printf("Номер MyChan: %u",ChanNum);
    ShowScr();
    //Послать значение канала "MyChan" (12.3) на сервер по номеру
    int value=123;
    LmSendByNum(ChanNum,value,10,0);
    GotoXY(4,60);
    printf("Послано значение: %d.%1d",value/10,value%10);
    ShowScr();
    //Ожидаем изменения канала от сервера
    while(1)
    {
        if(LmState==lmsCTRL)
        {
            char name[26];
            //Получить имя канала
            if(LmGetCtrlName(name))
            {
                //Получить новое значение канала
                value=LmGetCtrlValue(10);
                GotoXY(4,100);
                printf("Канал: %s ",name);
                GotoXY(4,120);
                printf("Значение: %d.%1d  ",value/10,value%10);
                ShowScr();
            }
        }
    }
}
```

Результат:

В окне "Редактор каналов" сервера LanMon можно увидеть новый появившийся канал "MyChan" номер 3 и полученное значения канала:



3.15.5. LMCLOSE

```
#include <lanmon.h>

void LmClose(void);
```

Завершить работу встроенного клиента сервера LanMon.
Если имеется подключение к серверу, то оно будет завершено.
После выполнения данной функции переменные *LmConnected* и *LmState* принимают нулевые значения.

В приведенном далее примере выполняется подключение к серверу LanMon, и затем в течении 20 секунд ежесекундно передаются нарастающие значение в канал с именем "TestChan1". Канал должен быть создан на сервере вручную до запуска примера.
Передаваемые значения в канал - это целые числа от 0 до 19.
Т.к. при посылки в канал используется делитель 10, поэтому канал на сервере принимает значение от 0 до 1,9.
По истечении 20 секунд работа клиента сервера LanMon завершается вызовом функции *LmClose*.

Пример:

```
#include <sat500.h>
#include <lanmon.h>

typedef  unsigned int UINT;

void main(void)
{
    ClrScr();
    BigFont=false;
    printf("Клиент сервера LanMon...");
    ShowScr();
    //Запуск клиента LanMon
    LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
    //Ожидаем подключения
    while(!LmConnected);
    //Подключены
    GotoXY(4,20);
    puts("Успешно подключились...");
    //Посылка значений каналу "TestChan1"
    for(int i=0;i<20;i++)
    {
        if(LmConnected)
```

```
        {
            //Послать канал по имени
            LmSendByName("TestChan1",i,10,0);
            GotoXY(4,100);
            printf("Значение: %d.%1d  ",i/10,i%10);
            ShowScr();
        }
        //Пауза 1 сек
        UINT t=GetTickCount();
        while((GetTickCount()-t)<1000)
        {
        }
    }
    //Отключение от сервера
    LmClose();
    GotoXY(4,140);
    puts("Отключены...");
    ShowScr();
    while(1);
}
```

Результат:

3.15.6. LMCONNECTED

```
#include <lanmon.h>

char LmConnected;
```

Переменная *LmConnected* возвращает состояние подключение клиента к серверу LanMon. Переменная может принимать следующие значения:

Значение	*Описание*
0	Нет подключения к серверу
1	Есть авторизованное подключение к серверу

Переменная доступна только для чтения.
Считается что клиент подключён к серверу если значение состояние клиента *LmState* не менее 4 (*lmsOK* - Подключены и авторизованы на сервере).

В приведенном далее примере выполняется подключение к серверу LanMon при помощи вызова функции *LmOpen* и далее на дисплей выводится состояние клиента *LmState* и состояние подключения *LmConnected*.

Пример:

```
#include <sat500.h>
#include <lanmon.h>

typedef  unsigned int UINT;

void main(void)
{
    ClrScr();
    BigFont=false;
    printf("Клиент сервера LanMon...");
```

```
    ShowScr();
    //Запуск клиента LanMon
    LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
    //Ожидаем подключения
    while(!LmConnected);
    //Подключены
    GotoXY(4,20);
    puts("Успешно подключились...");
    while(1)
    {
        //Состояние клиента LanMon
        GotoXY(4,100);
        printf("LmState=%d",LmState);
        GotoXY(4,120);
        printf("LmConnected=%d",LmConnected);
        ShowScr();
        UINT t=GetTickCount();
        while((GetTickCount()-t)<100)
        {
        }
    }
}
```

Результат:

3.15.7. LMGETCTRLNAME

```
#include <lanmon.h>

char LmGetCtrlName(char * chname);
```

Функция *LmGetCtrlName* позволяет прочитать имя канала из сообщения полученного от сервера LanMon о новом значении канала.
В качестве аргумента функции посылается указатель на массив, в который будет записано имя канала. Размер массива должен быть не менее 64 символов.
Читаемое имя имеет длину не более 63 символов. В конце строки обязательно будет записан символ 0.
*char * chname* - указатель на строку, куда будет записано название (имя) канала, которому прислано управляющее значение.

Возвращаемое значение:
0 - нет управления от сервера.
1 - получено управление от сервера и имя канала записано строку или номер канала находится в переменной *LmChanNum*.

Функция *LmGetCtrlName* возвращает единицу только в том случае, если переменная состояния клиента *LmState* приняла значение 7 (*lmsCTRL* - Получено управление).
После успешного выполнения функции и получении имени канала, можно получить новое значение канала вызвав функцию *LmGetCtrlValue*.
Если в результате вызова вернулось значение 1 (получено управление от сервера), но строка пустая, то следовательно вместо имени канала сервер прислал номер канала. Этот номер находится в переменной *LmChanNum*.
Также после вызова доступно значение активности канала на сервере в переменной *LmChanActivity*.

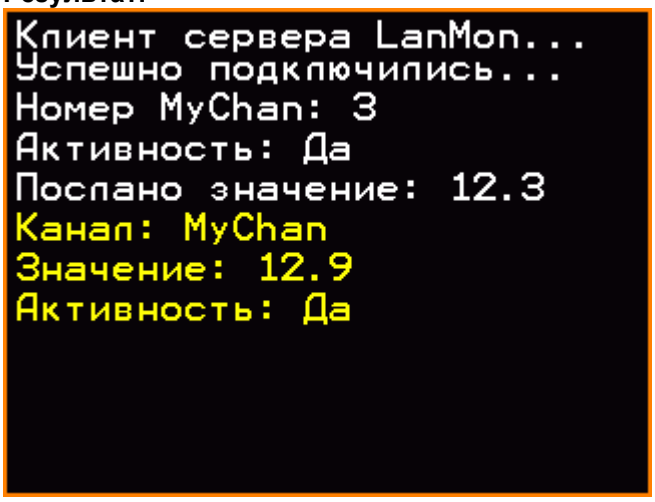
В приведенном далее примере на сервере создается новый канал с именем "MyChan" и выполняется получение номера канала на сервере.
В дальнейшем значение канала можно записывать как по имени так и по номеру.
В примере выполняется запись вновь созданного канала по полученному номеру.
После записи значения канала на сервер программа непрерывно ожидает нового значения канала от сервера.
При получении нового значения канала на дисплей выводится имя канала, активность и значение, полученное от сервера.

Пример:

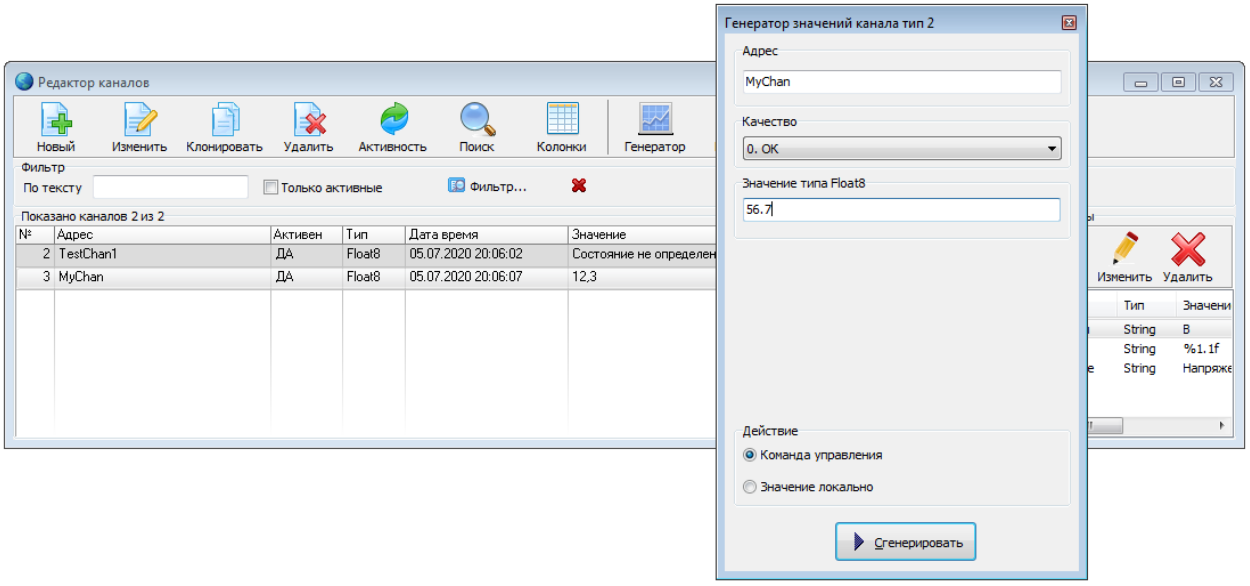
```
#include <sat500.h>
#include <lanmon.h>

typedef  unsigned int UINT;
//Номер канала на сервере
UINT ChanNum;
//Активность канала на сервере
char ChanActivity;

void main(void)
{
Restart:;
    ClrScr();
    BigFont=false;
    printf("Клиент сервера LanMon...");
    ShowScr();
    LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
    //Ожидаем подключения
    while(!LmConnected);
    //Подключены
    GotoXY(4,20);
    puts("Успешно подключились...");
    ShowScr();
    //Регистрация канала "MyChan" на сервере
    //Канал "MyChan" может изменяться сервером
    LmChanRegister("MyChan","B","%1.1f","Напряжение",true);
    //Ожидание регистрации 60 секунд
    UINT t=GetTickCount();
    while((GetTickCount()-t)<60000)
    {
        if(LmState==lmsREGISTERED)break;
    }
    if(LmState!=lmsREGISTERED)goto Restart;
    //Канал зарегистрирован
    //Запомнить активность канала "MyChan" на сервере
    ChanActivity=LmChanActivity;
    //Запомнить номер канала "MyChan" на сервере
    ChanNum=LmChanNum;
    GotoXY(4,40);
    printf("Номер MyChan: %u",ChanNum);
    GotoXY(4,60);
    char * s="Нет";
    if(ChanActivity)s="Да";
    printf("Активность: %s",s);
    ShowScr();
    //Послать значение канала "MyChan" (12.3) на сервер по номеру
    int value=123;
```

```
    LmSendByNum(ChanNum,value,10,0);
    GotoXY(4,80);
    printf("Послано значение: %d.%1d",value/10,value%10);
    ShowScr();
    //Ожидаем изменения канала от сервера
    while(1)
    {
        if(LmState==lmsCTRL)
        {
            FontColor=COLOR_YELLOW;
            //Получено новое значение канала
            char name[64];
            //Получить имя канала
            if(LmGetCtrlName(name))
            {
                GotoXY(4,100);
                if(name[0])
                {
                    printf("Канал: %s    ",name);
                }
                else
                {
                    printf("Номер канала: %u     ",ChanNum);
                }
                //Получить новое значение канала
                value=LmGetCtrlValue(10);
                GotoXY(4,120);
                printf("Значение: %d.%1d  ",value/10,value%10);
                GotoXY(4,140);
                s="Нет";
                if(ChanActivity)s="Да";
                printf("Активность: %s",s);
                ShowScr();
            }
        }
    }
}
```

Результат:



В окне "Редактор каналов" сервера LanMon можно послать новые значения канала используя кнопку "Генератор":



3.15.8. LMGETCTRLVALUE

```
#include <lanmon.h>

unsigned int LmGetCtrlValue(int mul);
```

Функция возвращает новое значение канала, полученное от сервера LanMon.
int mul - множитель, используемый для преобразования плавающего значение канала на сервере (value) в целочисленное значение канала в контроллере CAT-500.
Преобразование в целочисленное значения выполняется следующим образом: возвращаемое значение=(double)value * (double)mul.
Например, если плавающее значение канала на сервере равно 54.321 (54 целых, 321 тысячная) и задан множитель mul=1000, то функция вернёт целочисленное значения канала в контроллере CAT-500 равное 54321.
Вызов функции имеет смысл только в том случае, если получено новое значение управления от сервера и предварительно получено имя канала путем вызова функции *LmGetCtrlName*.

В приведенном далее примере на сервере создается новый канал с именем "MyChan" и выполняется получение номера канала на сервере.
В дальнейшем значение канала можно записывать как по имени так и по номеру.
В примере выполняется запись вновь созданного канала по полученному номеру.
После записи значения канала на сервер программа непрерывно ожидает нового значения канала от сервера.
При получении нового значения канала на дисплей выводится имя канала, активность и значение, полученное от сервера.

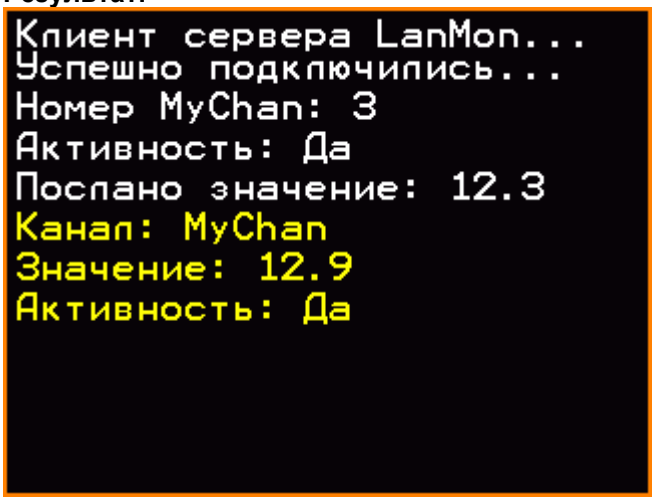
Пример:

```
#include <sat500.h>
#include <lanmon.h>

typedef  unsigned int UINT;
//Номер канала на сервере
UINT ChanNum;
//Активность канала на сервере
char ChanActivity;
```

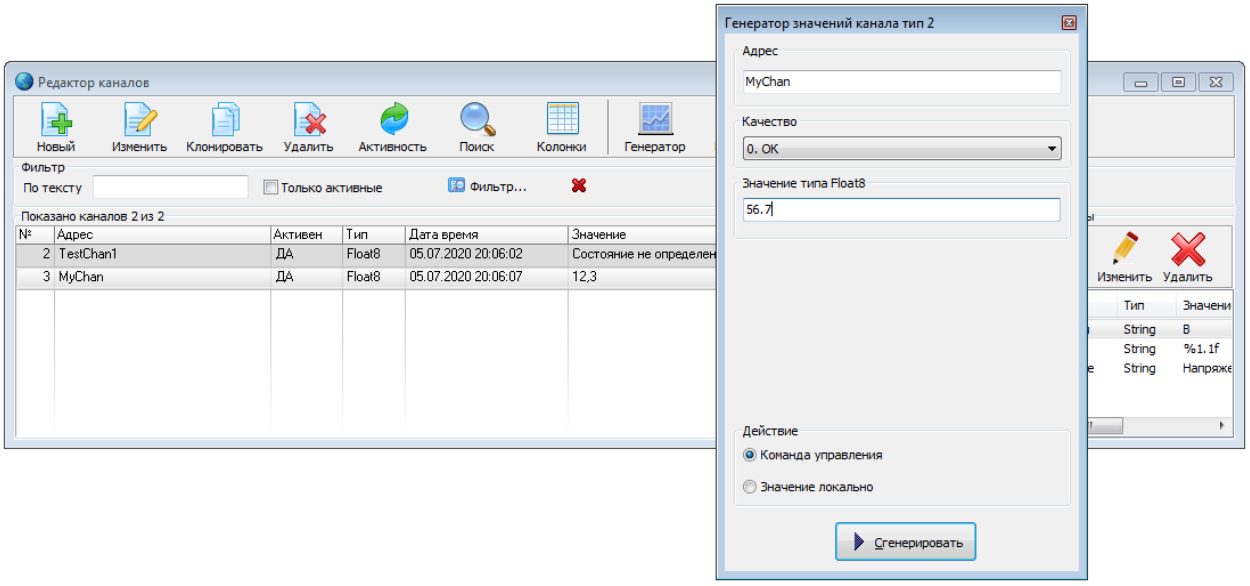
```
void main(void)
{
Restart:;
    ClrScr();
    BigFont=false;
    printf("Клиент сервера LanMon...");
    ShowScr();
    LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
    //Ожидаем подключения
    while(!LmConnected);
    //Подключены
    GotoXY(4,20);
    puts("Успешно подключились...");
    ShowScr();
    //Регистрация канала "MyChan" на сервере
    //Канал "MyChan" может изменяться сервером
    LmChanRegister("MyChan","B","%1.1f","Напряжение",true);
    //Ожидание регистрации 60 секунд
    UINT t=GetTickCount();
    while((GetTickCount()-t)<60000)
    {
        if(LmState==lmsREGISTERED)break;
    }
    if(LmState!=lmsREGISTERED)goto Restart;
    //Канал зарегистрирован
    //Запомнить активность канала "MyChan" на сервере
    ChanActivity=LmChanActivity;
    //Запомнить номер канала "MyChan" на сервере
    ChanNum=LmChanNum;
    GotoXY(4,40);
    printf("Номер MyChan: %u",ChanNum);
    GotoXY(4,60);
    char * s="Нет";
    if(ChanActivity)s="Да";
    printf("Активность: %s",s);
    ShowScr();
    //Послать значение канала "MyChan" (12.3) на сервер по номеру
    int value=123;
    LmSendByNum(ChanNum,value,10,0);
    GotoXY(4,80);
    printf("Послано значение: %d.%1d",value/10,value%10);
    ShowScr();
    //Ожидаем изменения канала от сервера
    while(1)
    {
        if(LmState==lmsCTRL)
        {
            FontColor=COLOR_YELLOW;
            //Получено новое значение канала
            char name[64];
            //Получить имя канала
            if(LmGetCtrlName(name))
            {
                GotoXY(4,100);
                if(name[0])
                {
                    printf("Канал: %s    ",name);
```

```
                    }
                    else
                    {
                        printf("Номер канала: %u      ",ChanNum);
                    }
                    //Получить новое значение канала
                    value=LmGetCtrlValue(10);
                    GotoXY(4,120);
                    printf("Значение: %d.%1d  ",value/10,value%10);
                    GotoXY(4,140);
                    s="Нет";
                    if(ChanActivity)s="Да";
                    printf("Активность: %s",s);
                    ShowScr();
                }
            }
        }
}
```

Результат:



В окне "Редактор каналов" сервера LanMon можно послать новые значения канала используя кнопку "Генератор":



3.15.9. LMOPEN

```
#include <lanmon.h>

char LmOpen (char * ip,unsigned short int port,unsigned int timeout,char *
login,char * password,unsigned short int ver);
```

Функция *LmOpen* включает в работу встроенного клиента сервера LanMon и начинает подключение к серверу.

*char * ip* - строка, которая содержит доменное имя сервера LanMon или IP адрес сервера в текстовом представлении
unsigned short int port - номер сетевого порта на котором работает удалённый сервер LanMon. В случае задания нулевого номера порта будет использован порт 3000.
unsigned int timeout - таймаут на операции обмена с LanMon сервером в миллисекундах. При подключении по сети ETHERNET рекомендуется устанавливать таймаут в значение 10000 (10 сек). При подключении по сети GSM рекомендуется устанавливать таймаут в значение не ниже 30000 (30 сек).
*char * login* - логин учётной записи, предназначенной для подключения контроллера CAT-500 на сервере.
*char * password* - пароль учётной записи, предназначенной для подключения контроллера CAT-500 на сервере.
unsigned short int ver - номер версии ПО контроллера CAT-500 отображаемый на сервере - старший байт - старший номер версии, младший байт - младший номер версии. Например значение 0x0102 соответствует номеру версии "1.2".

Возвращаемое значение:
1 - Успешно началось подключение к серверу LanMon
0 - Ошибка. Не удалось начать подключение - не задан адрес сервера.

После запуска клиента сервера LanMon можно определить что клиент успешно подключен при помощи чтения переменной *LmConnected*, и определить текущую стадию работы клиента при помощи чтения переменной *LmState*.
В случае возникновения проблем с подключением к серверу клиент самостоятельно выполняет переподключение к серверу.

Для контроля работоспособности сервера клиент каждую минуту посылает запрос к серверу, и получает ответ. При отсутствии ответа клиент считает, что соединения с сервером утеряно, и автоматически переподключается. Данный функционал не требует от программиста никаких действий. Для завершения работы встроенного клиента сервера используется функция *LmClose*.

В приведенном далее примере выполняется подключение к серверу LanMon и далее на дисплей выводится состояние клиента *LmState* и состояние подключения *LmConnected*.

Пример:

```
#include <sat500.h>
#include <lanmon.h>

typedef  unsigned int UINT;

void main(void)
{
    ClrScr();
    BigFont=false;
    printf("Клиент сервера LanMon...");
    ShowScr();
    //Запуск клиента LanMon
    LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
    //Ожидаем подключения
    while(!LmConnected);
    //Подключены
    GotoXY(4,20);
    puts("Успешно подключились...");
    while(1)
    {
        //Состояние клиента LanMon
        GotoXY(4,100);
        printf("LmState=%d",LmState);
        GotoXY(4,120);
        printf("LmConnected=%d",LmConnected);
        ShowScr();
        UINT t=GetTickCount();
        while((GetTickCount()-t)<100)
        {
        }
    }
}
```

Результат:

3.15.10. LMSENDBYNAME

```
#include <lanmon.h>

void LmSendByName(char * chname,int value,int div,char quality);
```

Функция *LmSendByName* позволяет послать значение канала на сервер. В качестве в качестве адреса используется текстовое имя канала на сервере *chname*.
Аргументы функции:
*char * chname* - текстовое имя канала на сервере (непустая строка длиной не более 63 символов).
int value - целочисленное значение канала в контроллере САТ-500.

int div - делитель целочисленного значение канала для вычисления плавающего значение канала на сервере. Передаваемое на сервер значение канала является плавающим числом, которое вычисляется следующим образом: значение на сервере=(double)value/(double)div.
Например, если целочисленные значения канала в контроллере CAT-500 value равно 54321, и задан делитель div=1000, то посылаемое на сервер значение канала равно 54.321 (54 целых, 321 тысячная).
char quality - значение качества канала на сервере. 0 соответствует достоверному значению канала, не 0 соответствует ошибке при получении значения канала. *Подробное описание качества канала*.

Функцию *LmSendByName* выполняет посылку значения канала с качеством на сервер только если имеется авторизированное подключение к серверу LanMon (переменная *LmConnected* равна true). При отсутствии подключения никакие действия не выполняются.

В приведенном далее примере выполняется подключение к серверу LanMon и затем в течении 20 секунд ежесекундно передаются нарастающие значение в канал с именем "TestChan1". Канал должен быть создан на сервере вручную до запуска примера.
Передаваемые значения в канал - это целые числа от 0 до 19.
Т.к. при посылки в канал используется делитель 10, поэтому канал на сервере принимает значение от 0 до 1,9.

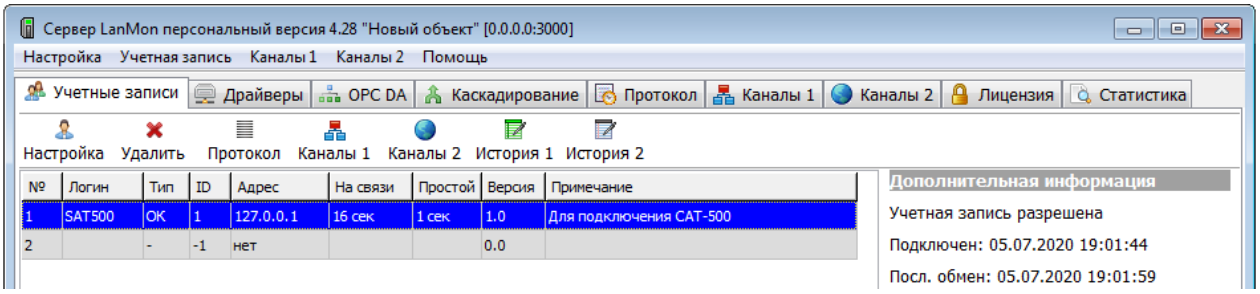
Пример:

```
#include <sat500.h>
#include <lanmon.h>

typedef  unsigned int UINT;

void main(void)
{
    ClrScr();
    BigFont=false;
    printf("Клиент LanMon...");
    ShowScr();
    //Запуск клиента LanMon
    LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
    //Ожидаем подключения
    while(!LmConnected);
    //Подключены
    GotoXY(4,20);
    puts("Успешно подключились...");
    //Посылка значений каналу "TestChan1"
    for(int i=0;i<20;i++)
    {
        if(LmConnected)
        {
            //Послать канал по имени
            LmSendByName("TestChan1",i,10,0);
            GotoXY(4,100);
            printf("Значение: %d.%1d  ",i/10,i%10);
            ShowScr();
        }
        //Пауза 1 сек
        UINT t=GetTickCount();
        while((GetTickCount()-t)<1000)
        {
        }
    }
```

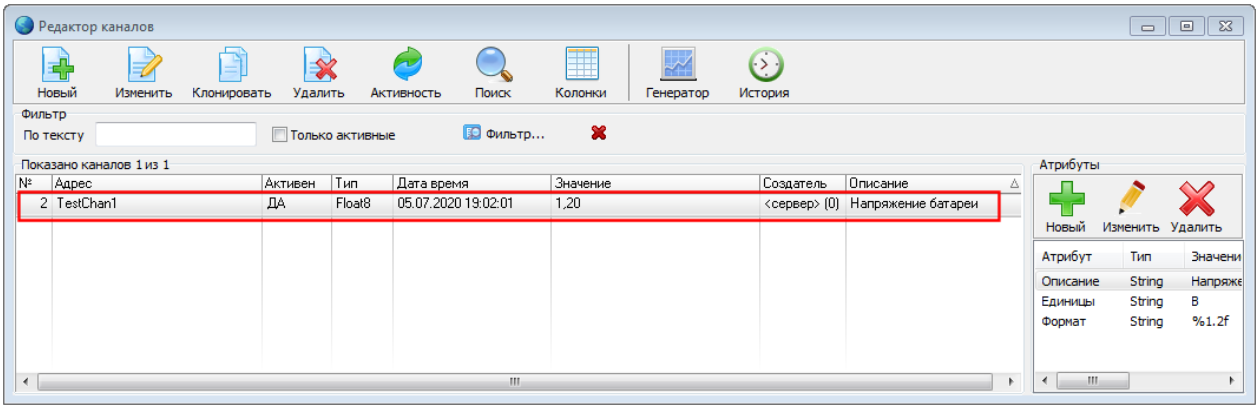
```
    //Отключение от сервера
    LmClose();
    while(1);
}
```

Результат:

На вкладке "Учетные записи" сервера LanMon можно наблюдать активность учетной записи номер 1 (контроллер CAT-500):



В окне "Редактор каналов" сервера LanMon можно наблюдать изменение значения канала:



3.15.11. LMSENDBYNUM

```
#include <lanmon.h>

void LmSendByNum(unsigned int chnum,int value,int div,char quality);
```

Функция *LmSendByNum* позволяет послать значение канала на сервер. В качестве в качестве адреса используется номер канала на сервере *chnum*.
Аргументы функции:
unsigned int chnum - номер канала на сервере (ненулевое значение)
int value - целочисленное значение канала в контроллере CAT-500.
int div - делитель целочисленного значение канала для вычисления плавающего значение канала на сервере. Передаваемое на сервер значение канала является плавающим числом, которое вычисляется следующим образом: значение на сервере=(double)value/(double)div.
Например, если целочисленные значения канала в контроллере CAT-500 value равно 54321, и задан делитель div=1000, то посылаемое на сервер значение канала равно 54.321 (54 целых, 321 тысячная).

char quality - значение качества канала на сервере. 0 соответствует достоверному значению канала, не 0 соответствует ошибке при получении значения канала. Подробное описание качества канала.

Функцию *LmSendByNum* выполняет посылку значения канала с качеством на сервер только если имеется авторизированное подключение к серверу LanMon (переменная *LmConnected* равна true). При отсутствии подключения никакие действия не выполняются.

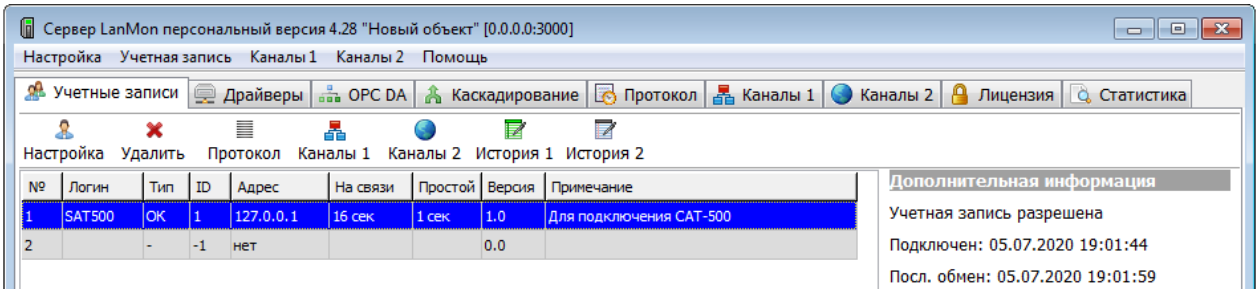
В приведенном далее примере выполняется подключение к серверу LanMon и затем в течении 20 секунд ежесекундно передаются нарастающие значение в канал с номером 2. Канал должен быть создан на сервере вручную до запуска примера.
Передаваемые значения в канал - это целые числа от 0 до 19.
Т.к. при посылки в канал используется делитель 10, поэтому канал на сервере принимает значение от 0 до 1,9.

Пример:

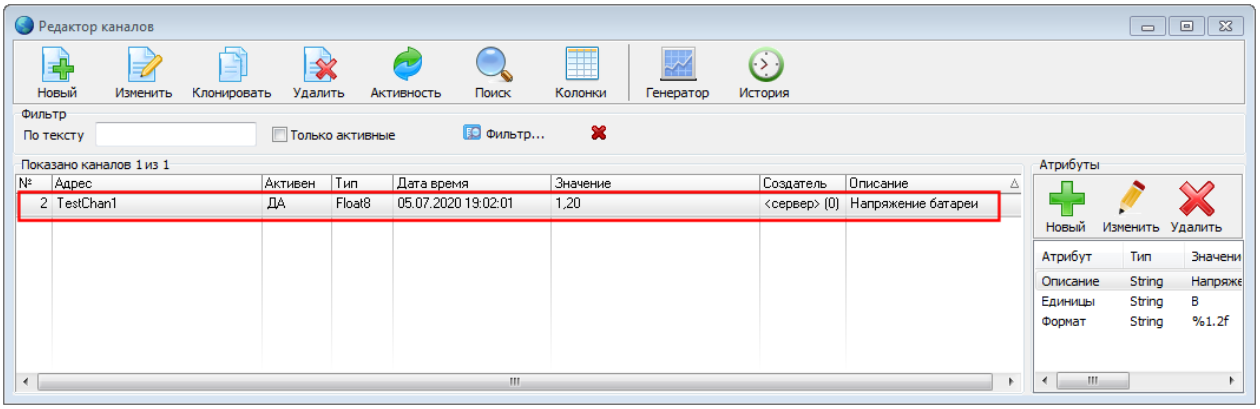
```
#include <sat500.h>
#include <lanmon.h>

typedef  unsigned int UINT;

void main(void)
{
    ClrScr();
    BigFont=false;
    printf("Клиент LanMon...");
    ShowScr();
    //Запуск клиента LanMon
    LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
    //Ожидаем подключения
    while(!LmConnected);
    //Подключены
    GotoXY(4,20);
    puts("Успешно подключились...");
    //Посылка значений каналу "TestChan1"
    for(int i=0;i<20;i++)
    {
        if(LmConnected)
        {
            //Послать канал по номеру
            LmSendByNum(2,i,10,0);
            GotoXY(4,100);
            printf("Значение: %d.%1d  ",i/10,i%10);
            ShowScr();
        }
        //Пауза 1 сек
        UINT t=GetTickCount();
        while((GetTickCount()-t)<1000)
        {
        }
    }
    //Отключение от сервера
    LmClose();
    while(1);
}
```

Результат:

На вкладке "Учетные записи" сервера LanMon можно наблюдать активность учетной записи номер 1 (контроллер CAT-500):



В окне "Редактор каналов" сервера LanMon можно наблюдать изменение значения канала:



3.15.12. LMSTATE

```
#include <lanmon.h>

unsigned char LmState;
```

Переменная *LmState* возвращает состояние клиента сервера LanMon.
Переменная может принимать следующие значения:

Значение	*Описание*
0	Клиент сервера LanMon не запущен
1	Выполняется подключение к серверу
2	Произошло отключение от сервера
3	Выполняется авторизация на сервере
4	Подключены и авторизованы на сервере
5	Выполняется регистрация канала
6	Канал зарегистрирован
7	Получено управление от сервера

Переменная доступна только для чтения.

Для удобства доступа к переменной во включаемом файле *<lanmon.h>* определены следующие константы:

```
enum
{
    lmsNONE=0,          //Клиент не запущен
    lmsCONNECTING=1,    //Подключение к серверу
    lmsDISCONNECTED=2,  //Отключены
    lmsAUTH=3,          //Авторизация на сервере
    lmsOK=4,            //Подключены и авторизованы на сервере
    lmsREGISTERING=5,   //Регистрация канала
    lmsREGISTERED=6,    //Канал зарегистрирован
    lmsCTRL=7           //Получено управление
};
```

В приведенном далее примере выполняется подключение к серверу LanMon при помощи вызова функции *LmOpen* и далее на дисплей выводится состояние клиента *LmState* и состояние подключения *LmConnected*.

Пример:

```
#include <sat500.h>
#include <lanmon.h>

typedef  unsigned int UINT;

void main(void)
{
    ClrScr();
    BigFont=false;
    printf("Клиент сервера LanMon...");
    ShowScr();
    //Запуск клиента LanMon
    LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
    //Ожидаем подключения
    while(!LmConnected);
    //Подключены
    GotoXY(4,20);
    puts("Успешно подключились...");
    while(1)
    {
        //Состояние клиента LanMon
        GotoXY(4,100);
        printf("LmState=%d",LmState);
        GotoXY(4,120);
        printf("LmConnected=%d",LmConnected);
        ShowScr();
        UINT t=GetTickCount();
        while((GetTickCount()-t)<100)
        {
        }
    }
}
```

Результат:

3.15.13. ПРИМЕР

Далее приведён пример, в котором на сервере автоматически создаются все каналы, соответствующие входам и выходам CAT-500.
В имя каждого канала кроме названия канала входит серийный номер CAT-500.
Далее контроллер CAT-500 передаёт все изменения входов и выходов на сервер, а также принимает управления для выходов контроллера (каналов AO1, AO2, DO1...DO11).
В качестве имени каждого канала используется строка, состоящая из текстового префикса названия сигнала и строки серийного номера контроллера CAT-500.

Пример

```
#include <sat500.h>
#include <limits.h>
#include <string.h>
#include <lanmon.h>
//-------------------------------------------------------------------------------
--
typedef  unsigned int UINT;
typedef  char bool;
//-------------------------------------------------------------------------------
--
enum
{
    DIS_CTRL=0,     //Канал без управления
    EN_CTRL=1       //Канал с управлением
};
//-------------------------------------------------------------------------------
--
//Префиксы имён каналов
char * Names[]=
{
    "T1",   //0
    "T2",   //1
    "T3",   //2
    "T4",   //3
    "T5",   //4
    "AI1",  //5
    "AI2",  //6
    "DI1",  //7
    "DI2",  //8
    "DI3",  //9
    "DI4",  //10
    "DI5",  //11
    "DI6",  //12
    "DI7",  //13
    "DI8",  //14
    "DI9",  //15
    "DI10", //16
    "AO1",  //17
    "AO2",  //18
    "DO1",  //19
    "DO2",  //20
    "DO3",  //21
    "DO4",  //22
    "DO5",  //23
    "DO6",  //24
    "DO7",  //25
    "DO8",  //26
    "DO9",  //27
```

```
    "DO10", //28
    "DO11"  //29
};
#define Q_CHAN   30
//-------------------------------------------------------------------------------
--
//Локальные значения каналов
int ChanVal[Q_CHAN];
//Для имени канала
char ChanName[32];
//Номера каналов с управлением на сервере (получаются при регистрации канала)
UINT ChanNumAO1;
UINT ChanNumAO2;
UINT ChanNumDO[11];
//-------------------------------------------------------------------------------
--
//Получить значение канала
int GetChanVal(int index)
{
    if(index<5)return T[index];
    if(index>=7 && index<=16)
    {
        int pos=index-7;
        return DI[pos];
    }
    if(index>=19 && index<=29)
    {
        int pos=index-19;
        return DO[pos];
    }
    switch(index)
    {
        case 5: return AI1;
        case 6: return AI2;
        case 17: return AO1;
        case 18: return AO2;
    }
    return 0;
}
//-------------------------------------------------------------------------------
--
//Получить значение делителя канала
int GetChanDiv(int index)
{
    if(index<5)return 100;
    switch(index)
    {
        case 5: return 1;
        case 6: return 1;
        case 17: return 1;
        case 18: return 1;
    }
    return 1;
}
//-------------------------------------------------------------------------------
--
//Функция возвращает строку имени канал, состоящую из префикса
//и серийного номера CAT-500
```

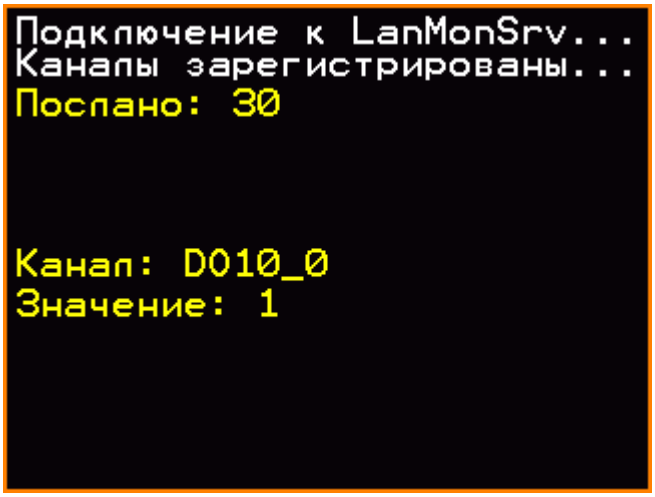
```
char * GetChanName(char * prefix)
{
    sprintf(ChanName,"%s_%u",prefix,SatSerNumber);
    //Длина имени канала не более 25 символов
    ChanName[25]=0;
    return ChanName;
}
//-------------------------------------------------------------------------------
--
//Функция регистрирует канал на сервере LanMon
bool RegChan(char * prefix,char * unit,char * fmt,char * cmnts,bool ctrl)
{
    if(!LmConnected)return false;
    LmChanRegister(GetChanName(prefix),unit,fmt,cmnts,ctrl);
    //Ожидание регистрации 60 секунд
    UINT t=GetTickCount();
    while((GetTickCount()-t)<60000)
    {
        if(LmState==lmsREGISTERED)return true;
    }
    return false;
}
//-------------------------------------------------------------------------------
--
void main(void)
{
    UINT cnt;
Restart:;
    cnt=0;
    ClrScr();
    BigFont=false;
    printf("Подключение к LanMonSrv...");
    ShowScr();
    LmOpen("127.0.0.1",3000,10000,"SAT500","",0x0100);
    //Ожидаем подключения
    while(!LmConnected);
    //Подключены
    //Регистрация каналов температуры T1..T5 без возможности управления
    if(!RegChan("T1","°C","%1.2f","Температура 1",DIS_CTRL))goto Restart;
    if(!RegChan("T2","°C","%1.2f","Температура 2",DIS_CTRL))goto Restart;
    if(!RegChan("T3","°C","%1.2f","Температура 3",DIS_CTRL))goto Restart;
    if(!RegChan("T4","°C","%1.2f","Температура 4",DIS_CTRL))goto Restart;
    if(!RegChan("T5","°C","%1.2f","Температура 5",DIS_CTRL))goto Restart;
    //Регистрация каналов AI1,AI2
    if(!RegChan("AI1","mB","%1.0f","Вход AI1",DIS_CTRL))goto Restart;
    if(!RegChan("AI2","mB","%1.0f","Вход AI2",DIS_CTRL))goto Restart;
    //Регистрация каналов DI1...DI10 без возможности управления
    for(int i=1;i<=10;i++)
    {
        char name[8];
        sprintf(name,"DI%u",i);
        if(!RegChan(name,"","%1.0f",name,DIS_CTRL))goto Restart;
    }
    //Регистрация каналов AO1,AO2 с управлением
    if(!RegChan("AO1","mB","%1.0f","Выход AO1",EN_CTRL))goto Restart;
    //Запомнить номер канала на сервере
    ChanNumAO1=LmChanNum;
    if(!RegChan("AO2","mB","%1.0f","Выход AO2",EN_CTRL))goto Restart;
```

```
    //Запомнить номер канала на сервере
    ChanNumAO2=LmChanNum;
    //Регистрация каналов DO1...DO11 с управлением
    for(int i=1;i<=11;i++)
    {
        char name[8];
        sprintf(name,"DO%u",i);
        if(!RegChan(name,"","%1.0f",name,EN_CTRL))goto Restart;
        //Запомнить номер канала на сервере
        ChanNumDO[i-1]=LmChanNum;
    }
    GotoXY(4,20);
    puts("Каналы зарегистрированы...");
    ShowScr();
    //Локальные значения каналов
    for(int i=0;i<Q_CHAN;i++)
    {
        ChanVal[i]=INT_MAX;
    }
    //Посылать значение каналов на сервер
    while(1)
    {
        for(int i=0;i<Q_CHAN;i++)
        {
            //i=0-6 Температуры и аналоговые входы
            //i=7-16 Цифровые входы DI
            //i=17-18 Аналоговые выходы
            //Получить значение канала
            int cv=GetChanVal(i);
            bool changed=false;
            if(i<5)
            {
                //Каналы температуры
                if(abs(ChanVal[i]-cv)>=50)
                {
                    //Канал изменился не менее чем на 50 (0.5 градуса для
температуры)
                    changed=true;
                }
            }
            if(i==5 || i==6)
            {
                //Аналоговые входы
                if(abs(ChanVal[i]-cv)>=100)
                {
                    //Канал изменился не менее чем на 100
                    changed=true;
                }
            }
            if(i>=7 && i<=16)
            {
               //Цифровые входы DI
                if(ChanVal[i]!=cv)
                {
                    //Вход изменился
                    changed=true;
                }
            }
```

```
        if(i==17 || i==18)
        {
            //Аналоговые выходы
            if(abs(ChanVal[i]-cv)>=100)
            {
                //Канал изменился не менее чем на 100
                changed=true;
            }
        }
        if(i>=19 && i<=29)
        {
           //Цифровые выходы DO
            if(ChanVal[i]!=cv)
            {
                //Выход изменился
                changed=true;
            }
        }
        //Значение изменилось?
        if(changed)
        {
            //Канал изменился
            ChanVal[i]=cv;
            //Послать новое значение канала
            LmSendByName(GetChanName(Names[i]),cv,GetChanDiv(i),0);
            GotoXY(4,40);
            printf("Послано: %u",cnt++);
            ShowScr();
        }
    }
    //200 мс ожидаем значения канала от сервера
    UINT t=GetTickCount();
    while((GetTickCount()-t)<200)
    {
        if(LmState==lmsDISCONNECTED)
        {
            //Произошло отключение от сервера
            goto Restart;
        }
        if(LmState==lmsCTRL)
        {
            FontColor=COLOR_YELLOW;
            //Получено новое значение канала от сервера
            char name[64];
            int val;
            LmGetCtrlName(name);
            val=LmGetCtrlValue(1);
            GotoXY(4,120);
            if(name[0])
            {
                //Сервер прислал значение канала по имени
                printf("Канал: %s",name);
                GotoXY(4,140);
                printf("Значение: %d       ",val);
                ShowScr();
                //Проверка каналов AO1 и AO2
                if(!strcmp(name,GetChanName("AO1")))
                {
```

```
                        //Получено значение канала AO1
                        AO1=val;
                    }
                    if(!strcmp(name,GetChanName("AO2")))
                    {
                        //Получено значение канала AO2
                        AO2=val;
                    }
                    //Проверка каналов DO1...DO11
                    for(int i=0;i<11;i++)
                    {
                        char prefix[8];
                        sprintf(prefix,"DO%u",i+1);
                        if(!strcmp(name,GetChanName(prefix)))
                        {
                            //Получено значение канала DO[i]
                            DO[i]=val;
                        }
                    }
                }
                else
                {
                    //Сервер прислал значение канала по номеру
                    UINT chnum=LmChanNum;
                    printf("Номер канала: %d",chnum);
                    GotoXY(4,140);
                    printf("Значение: %d       ",val);
                    ShowScr();
                    //Проверка каналов AO1 и AO2
                    if(ChanNumAO1==chnum)
                    {
                        //Получено значение канала AO1
                        AO1=val;
                    }
                    if(ChanNumAO2==chnum)
                    {
                        //Получено значение канала AO2
                        AO2=val;
                    }
                    //Проверка каналов DO1...DO11
                    for(int i=0;i<11;i++)
                    {
                        if(ChanNumDO[i]==chnum)
                        {
                            //Получено значение канала DO[i]
                            DO[i]=val;
                        }
                    }
                }
            }
        }
    }
}
```

Результат:



3.16. СЕТЕВЫЕ ПЕРЕМЕННЫЕ

Сетевые переменные служат для организации обмена данными между несколькими контроллерами CAT-500 находящимися в режиме С-скрипт или в режиме свободно программируемого контроллера FBD.
Каждый контроллер CAT-500 может являться одновременно как источником, так и приёмником сетевых переменных. Кроме того, источником сетевых переменных может являться блок БКД-ПК-RF.

Технология

Для обмена данными между контроллерами используются интерфейс «Ethernet» и широковещательные UDP-пакеты, соответственно объединяемые контроллеры CAT-500 должны быть расположены в рамках одного широковещательного домена (сегмента) сети. Для объединения контроллеров по сети «Ethernet» необходимо использование коммутатора уровня L1, L2 или L3. Для некоторых моделей управляемых коммутаторов может потребоваться изменение настроек с целью неограниченного прохождения широковещательных пакетов.
Для построения территориально распределённого широковещательного домена может быть использована технология VPN (L2TP, GRE) при условии соответствующей настройки маршрутизаторов (включение Bridge Control Protocol для маршрутизаторов Mikrotik, настройка Policy Based Routing для маршрутизаторов Cisco и т.п.). Более подробную информацию о настройке маршрутизаторов можно получить в их технической документации.
Передача сетевых переменных производится контроллером CAT-500 при изменении их значений, а также при подключении к сети нового контроллера. Используемый механизм передачи не гарантирует фиксированное время доставки переменных до всех контроллеров сети, поэтому не рекомендуется использование сетевых переменных для передачи импульсных сигналов для которых значимо время их изменения.

В С-Скрипт доступны следующие переменные:

Переменная	***Описание***
unsigned int NetVarsPacketCount	Счетчик принятых пакетов, содержащих сетевые переменные

Доступны следующие функции для работы с сетевыми переменными:

Функция	***Описание***
*char NetVarsStart (unsigned short group,unsigned short port,unsigned int * nums,int * vals,unsigned int * sns,unsigned short cnt)*	Функция задаёт настройки, а также включает в работу модуль работы с сетевыми переменными. Функция указывает в какие переменные программы необходимо помещать принятые сетевые переменные

void NetVarsStop (void)	Функция завершает работу модуля работы с сетевыми переменными
*char NetVarsSend (unsigned int * nums,int * vals,unsigned short cnt)*	Функция позволяет послать значения сетевых переменных в произвольный момент времени

3.16.1. NETVARSPACKETCOUNT

```
#include <netvars.h>

unsigned int NetVarsPacketCount;
```

Переменная *NetVarsPacketCount* возвращает количество принятых ETHERNET пакетов с момента запуска модуля работы с сетевыми переменными (вызов функции *NetVarsStart*).
Позволяет отследить сетевую активность устройств своей группы.

3.16.2. NETVARSSTART

```
#include <netvars.h>

char NetVarsStart(unsigned short group,unsigned short port,unsigned int *
nums,int * vals,unsigned int * sns,unsigned short cnt);
```

Функция *NetVarsStart* включает в работу встроенный модуль работы с сетевыми переменными.
Аргументы функции:
unsigned short group - номер группы сетевых переменных с которыми будет работать данный CAT-500. Контроллер будет видеть только переменные указанной группы, и передавать переменные только в указанную группу. Таким образом в одной подсети могут работать несколько групп сетевых переменных не мешая друг другу.
unsigned short port - номер сетевого порта, который будет использоваться для переменных указанной группы. На других контроллерах CAT-500 или других устройствах, работающих с сетевыми переменными должен быть задан такой же номер порта. Если задаётся нулевое значение порта, то будет использован порт номер 3301.
*unsigned int * nums* - указатель на массив переменных, который содержит номера принимаемых сетевых переменных. Количество номеров в массиве должно быть не меньше, чем количество переменных в аргументе *cnt*.
*int * vals* - указатель на массив переменных, который содержит значения принимаемых сетевых переменных. Количество переменных в массиве должно быть не меньше, чем количество переменных в аргументе *cnt*.
*int * sns* - указатель на массив , который содержит серийные номера принимаемых сетевых переменных. Количество переменных в массиве должно быть не меньше, чем количество переменных в аргументе *cnt*. Если серийные номера не требуются, то можно указать нулевое значение *sns*.
unsigned short cnt - количество сетевых переменных, параметры которых задано в аргументах *nums, vals* и *sns*.
Если контроллер не должен принимать сетевые переменные, то допускается указывать нулевое значение аргумента *cnt* и нулевые значения в аргументах *nums, vals* и *sns*.
Функция возвращает нулевое значение, если в контроллере CAT-500 недостаточно сетевых UDP сокетов.
В случе успешного запуска модуля работы с сетевыми переменными функция возвращает единичное значение.

Далее приведены два примера.
В первом примере программа принимает сетевые переменные с номерами 1..4 в группе 1 и передаёт сетевые переменные с номерами 5...8 в этой же группе.

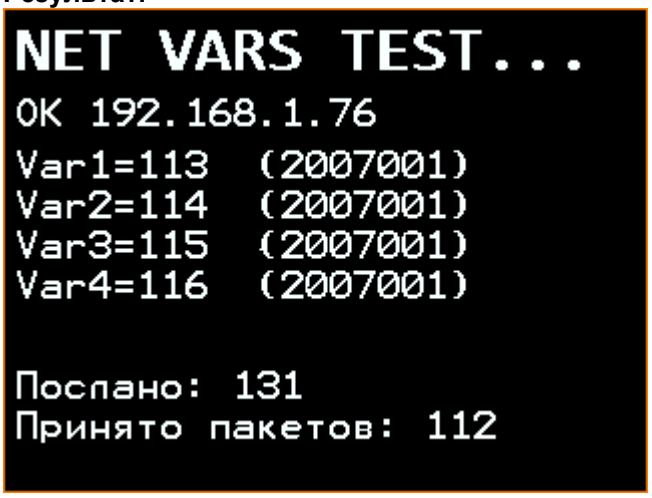
Во втором примере программа принимает сетевые переменные с номерами 5..8 в группе 1 и передаёт сетевые переменные с номерами 1...4 в этой же группе.
Таким образом два контроллера CAT-500 обмениваются 8-ю сетевыми переменными. Вместо одного из контроллеров может использоваться имитатор.
Кроме номеров и значений на дисплей выводятся серийные номера источников переменных.

Пример 1:

```
#include <sat500.h>
#include <netvars.h>

//Количество сетевых переменных
#define Q_VARS   4

//Принимаемые переменные
//Номера
unsigned int NetVarNum[Q_VARS];
//Значения
int NetVarVal[Q_VARS];
//Серийные номера источников
unsigned int NetVarSerNum[Q_VARS];

//Посылаемые переменные
unsigned int SendNetVarNum[Q_VARS];
int SendNetVarVal[Q_VARS];
unsigned int SendCnt;
void main(void)
{
    ClrScr();
    puts("NET VARS TEST...");
    BigFont=false;
    GotoXY(4,28);
    //Принимаемые переменные номера 1...4
    NetVarNum[0]=1;
    NetVarNum[1]=2;
    NetVarNum[2]=3;
    NetVarNum[3]=4;
    if(NetVarsStart(1,3301,NetVarNum,NetVarVal,NetVarSerNum,Q_VARS))
    {
        puts("\nOK");
    }
    else
    {
        puts("\nERROR");
    }
    //Показать IP адрес CAT-500
    union
    {
        unsigned int ip;
        unsigned char bip[4];
    };
    ip=SatIp;
    printf(" %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ShowScr();
    //Посылаемые переменные номера 5...8
    SendNetVarNum[0]=5; SendNetVarVal[0]=0;
    SendNetVarNum[1]=6; SendNetVarVal[1]=1;
    SendNetVarNum[2]=7; SendNetVarVal[2]=2;
```

```
    SendNetVarNum[3]=8; SendNetVarVal[3]=3;
    while(1)
    {
        for(int i=0;i<Q_VARS;i++)
        {
            GotoXY(4,70+i*20);
            printf("Var%u=%d  (%d)
",NetVarNum[i],NetVarVal[i],NetVarSerNum[i]);
        }
        ShowScr();
        //Пауза 1 сек
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<1000)
        {
        }
        //Увеличить на единицу посылаемые переменные
        for(int i=0;i<Q_VARS;i++)
        {
            SendNetVarVal[i]=SendNetVarVal[i]+1;
        }
        //Послать переменные 5...8
        NetVarsSend(SendNetVarNum,SendNetVarVal,Q_VARS);
        GotoXY(4,180);
        printf("Послано: %u   ",++SendCnt);
        GotoXY(4,200);
        printf("Принято пакетов: %u   ",NetVarsPacketCount);
        ShowScr();
    }
}
```

Результат:



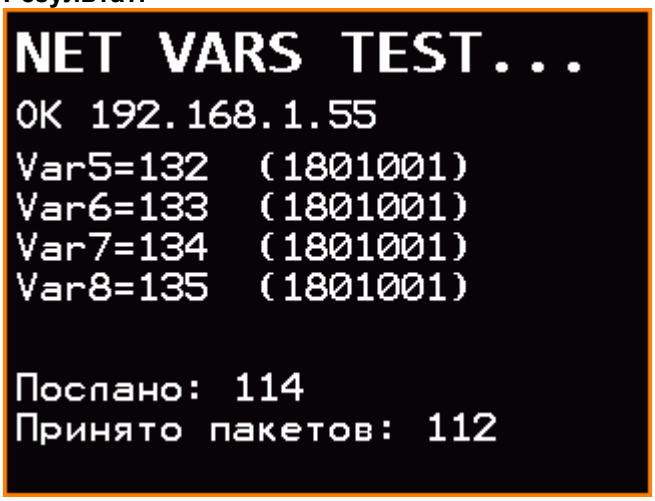
Пример 2:

```
#include <sat500.h>
#include <netvars.h>

//Количество сетевых переменных
#define Q_VARS   4

//Принимаемые переменные
//Номера
```

```
unsigned int NetVarNum[Q_VARS];
//Значения
int NetVarVal[Q_VARS];
//Серийные номера источников
unsigned int NetVarSerNum[Q_VARS];

//Посылаемые переменные
unsigned int SendNetVarNum[Q_VARS];
int SendNetVarVal[Q_VARS];
unsigned int SendCnt;

void main(void)
{
    ClrScr();
    puts("NET VARS TEST...");
    BigFont=false;
    GotoXY(4,28);
    //Принимаемые переменные номера 5...8
    NetVarNum[0]=5;
    NetVarNum[1]=6;
    NetVarNum[2]=7;
    NetVarNum[3]=8;
    if(NetVarsStart(1,3301,NetVarNum,NetVarVal,NetVarSerNum,Q_VARS))
    {
        puts("\nOK");
    }
    else
    {
        puts("\nERROR");
    }
    //Показать IP адрес CAT-500
    union
    {
        unsigned int ip;
        unsigned char bip[4];
    };
    ip=SatIp;
    printf(" %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ShowScr();
    //Посылаемые переменные номера 1...4
    SendNetVarNum[0]=1; SendNetVarVal[0]=0;
    SendNetVarNum[1]=2; SendNetVarVal[1]=1;
    SendNetVarNum[2]=3; SendNetVarVal[2]=2;
    SendNetVarNum[3]=4; SendNetVarVal[3]=3;
    while(1)
    {
        for(int i=0;i<Q_VARS;i++)
        {
            GotoXY(4,70+i*20);
            printf("Var%u=%d  (%d)  ",NetVarNum[i],NetVarVal[i],Net-
VarSerNum[i]);
        }
        ShowScr();
        //Пауза 1 сек
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<1000)
        {
        }
```

```
        //Увеличить на единицу посылаемые переменные
        for(int i=0;i<Q_VARS;i++)
        {
            SendNetVarVal[i]=SendNetVarVal[i]+1;
        }
        //Послать переменные 1...4
        NetVarsSend(SendNetVarNum,SendNetVarVal,Q_VARS);
        GotoXY(4,180);
        printf("Послано: %u    ",++SendCnt);
        GotoXY(4,200);
        printf("Принято пакетов: %u    ",NetVarsPacketCount);
        ShowScr();
    }
}
```

Результат:



Для просмотра передаваемых переменных можно воспользоваться специальным окном просмотра сетевых переменных.

3.16.3. NETVARSSTOP

```
#include <netvars.h>

void NetVarsStop(void);
```

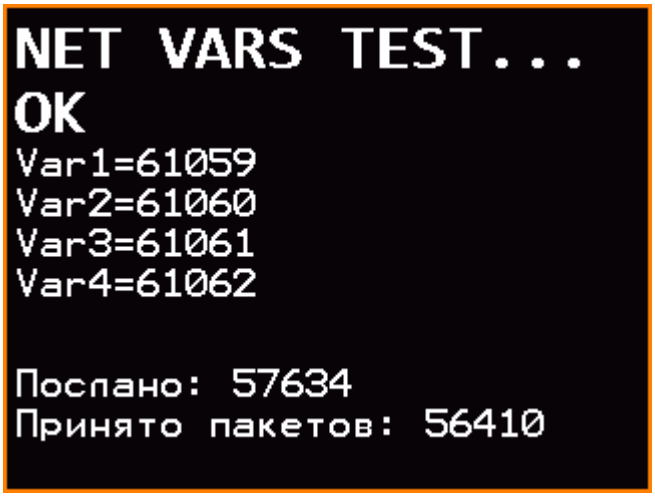
Функция *NetVarsStop* завершает работу модуля работы с сетевыми переменными и обнуляет значение количества принятых сетевых пакетов *NetVarsPacketCount*.

Пример:

```
#include <sat500.h>
#include <netvars.h>

#define Q_VARS   4

//Принимаемые переменные
unsigned int NetVarNum[Q_VARS];
int NetVarVal[Q_VARS];
```

```
//Посылаемые переменные
unsigned int SendNetVarNum[Q_VARS];
int SendNetVarVal[Q_VARS];
unsigned int SendCnt;

void main(void)
{
    ClrScr();
    puts("NET VARS TEST...");
    //Принимаемые переменные номера 1...4
    NetVarNum[0]=1;
    NetVarNum[1]=2;
    NetVarNum[2]=3;
    NetVarNum[3]=4;
    if(NetVarsStart(1,3301,NetVarNum,NetVarVal,Q_VARS))
    {
        puts("\nOK");
    }
    ShowScr();
    BigFont=0;
    //Посылаемые переменные номера 5...8
    SendNetVarNum[0]=5; SendNetVarVal[0]=0;
    SendNetVarNum[1]=6; SendNetVarVal[1]=1;
    SendNetVarNum[2]=7; SendNetVarVal[2]=2;
    SendNetVarNum[3]=8; SendNetVarVal[3]=3;
    while(1)
    {
        for(int i=0;i<Q_VARS;i++)
        {
            GotoXY(4,70+i*20);
            printf("Var%u=%d      ",NetVarNum[i],NetVarVal[i]);
        }
        ShowScr();
        //Пауза 1 сек
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<1000)
        {
        }
        //Увеличить на единицу посылаемые переменные
        for(int i=0;i<Q_VARS;i++)
        {
            SendNetVarVal[i]=SendNetVarVal[i]+1;
        }
        //Послать переменные 5...8
        NetVarsSend(SendNetVarNum,SendNetVarVal,Q_VARS);
        GotoXY(4,180);
        printf("Послано: %u   ",++SendCnt);
        GotoXY(4,200);
        printf("Принято пакетов: %u   ",NetVarsPacketCount);
        ShowScr();
    }
}
```

Результат:



3.16.4. NETVARSSEND

```
#include <netvars.h>

char NetVarsSend(unsigned int * nums,int * vals,unsigned short cnt);
```

Функция позволяет послать значения сетевых переменных в произвольный момент времени.
Аргументы функции:
*unsigned int * nums* - указатель на массив переменных, который содержит номера посылаемых сетевых переменных. Количество номеров в массиве должно быть не меньше, чем количество переменных в аргументе *cnt*.
*int * vals* - указатель на массив переменных, который содержит значения посылаемых сетевых переменных. Количество переменных в массиве должно быть не меньше, чем количество переменных в аргументе *cnt*.
unsigned short cnt - количество сетевых переменных, параметры которых задано в аргументах nums и vals.
Не допускается указывать нулевое значение аргумента cnt и NULL в аргументах nums и vals.
Функция возвращает нулевое значение если не запущена работв модуля работы сетевых переменных вызовом функции *NetVarsStart*.
В случе успешной посылки сетевых переменных функция возвращает единичное значение.
В посланном UDP пакете указывается номер группы сетевых переменных и номер сетевого порта, указанные в вызове функции *NetVarsStart*. Переданные переменные будут видеть только контроллеры, в которых заданы те же значения группы и такой же номер сетевого порта.

Пример:

```
#include <sat500.h>
#include <netvars.h>

#define Q_VARS   4

//Принимаемые переменные
unsigned int NetVarNum[Q_VARS];
int NetVarVal[Q_VARS];

//Посылаемые переменные
unsigned int SendNetVarNum[Q_VARS];
int SendNetVarVal[Q_VARS];
unsigned int SendCnt;
```

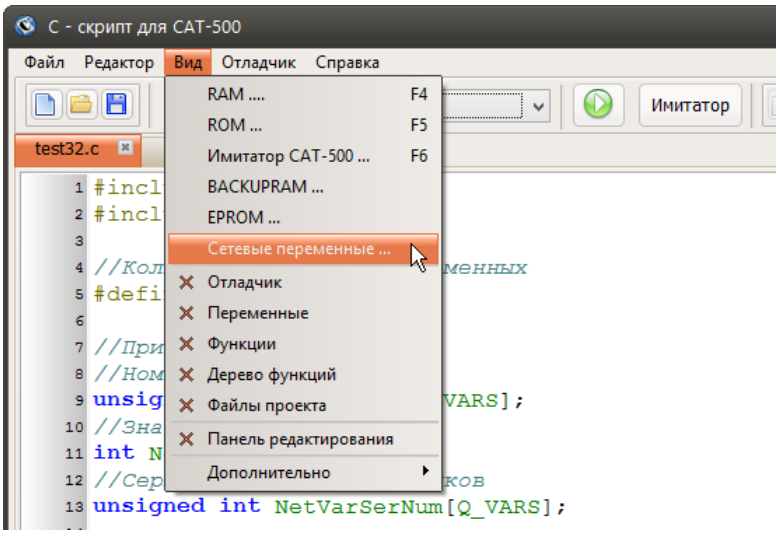
```
void main(void)
{
    ClrScr();
    puts("NET VARS TEST...");
    //Принимаемые переменные номера 1...4
    NetVarNum[0]=1;
    NetVarNum[1]=2;
    NetVarNum[2]=3;
    NetVarNum[3]=4;
    if(NetVarsStart(1,3301,NetVarNum,NetVarVal,Q_VARS))
    {
        puts("\nOK");
    }
    ShowScr();
    BigFont=0;
    //Посылаемые переменные номера 5...8
    SendNetVarNum[0]=5; SendNetVarVal[0]=0;
    SendNetVarNum[1]=6; SendNetVarVal[1]=1;
    SendNetVarNum[2]=7; SendNetVarVal[2]=2;
    SendNetVarNum[3]=8; SendNetVarVal[3]=3;
    while(1)
    {
        for(int i=0;i<Q_VARS;i++)
        {
            GotoXY(4,70+i*20);
            printf("Var%u=%d      ",NetVarNum[i],NetVarVal[i]);
        }
        ShowScr();
        //Пауза 1 сек
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<1000)
        {
        }
        //Увеличить на единицу посылаемые переменные
        for(int i=0;i<Q_VARS;i++)
        {
            SendNetVarVal[i]=SendNetVarVal[i]+1;
        }
        //Послать переменные 5...8
        NetVarsSend(SendNetVarNum,SendNetVarVal,Q_VARS);
        GotoXY(4,180);
        printf("Послано: %u   ",++SendCnt);
        GotoXY(4,200);
        printf("Принято пакетов: %u   ",NetVarsPacketCount);
        ShowScr();
    }
}
```

Результат:

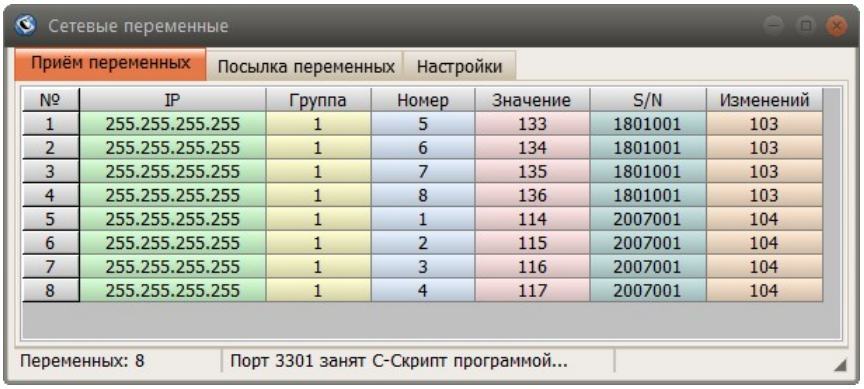
```
NET VARS TEST...
OK
Var1=61059
Var2=61060
Var3=61061
Var4=61062

Послано: 57634
Принято пакетов: 56410
```

3.16.5. ПРОСМОТР СЕТЕВЫХ ПЕРЕМЕННЫХ

Среда разработки С-Скрипт позволяет просмотреть сетевые переменные, передаваемые по сети ETHERNET.
Для открытия окна просмотра сетевых переменных необходимо выбрать в основном меню программы пункт **"Вид/Сетевые переменные ..."**:



Откроется окно в котором можно просматривать сетевые переменные:

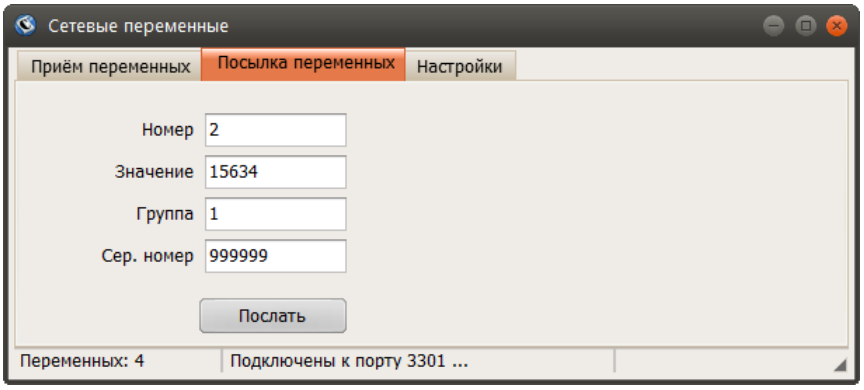


№	IP	Группа	Номер	Значение	S/N	Изменений
1	255.255.255.255	1	5	133	1801001	103
2	255.255.255.255	1	6	134	1801001	103
3	255.255.255.255	1	7	135	1801001	103
4	255.255.255.255	1	8	136	1801001	103
5	255.255.255.255	1	1	114	2007001	104
6	255.255.255.255	1	2	115	2007001	104
7	255.255.255.255	1	3	116	2007001	104
8	255.255.255.255	1	4	117	2007001	104

Переменные в окне добавляются в таблицу автоматически и обновляются по мере появления изменений переменных.

Вкладка "Посылка переменных"

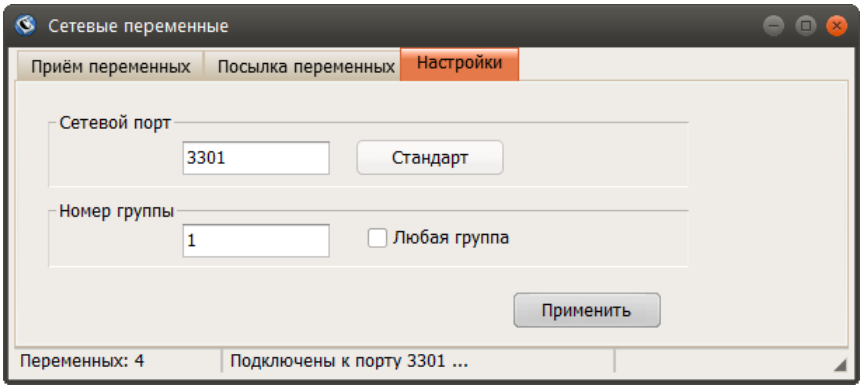
На вкладке "Посылка переменных" можно послать значение любой сетевой переменной:



Посылка переменной возможна если сетевой порт сетевых переменных не занят С-Скрипт программой или на компьютере не работает другая программа, использующая сетевые переменные и этот же порт (например, редактор FBD для CAT-500).

Вкладка "Настройки"

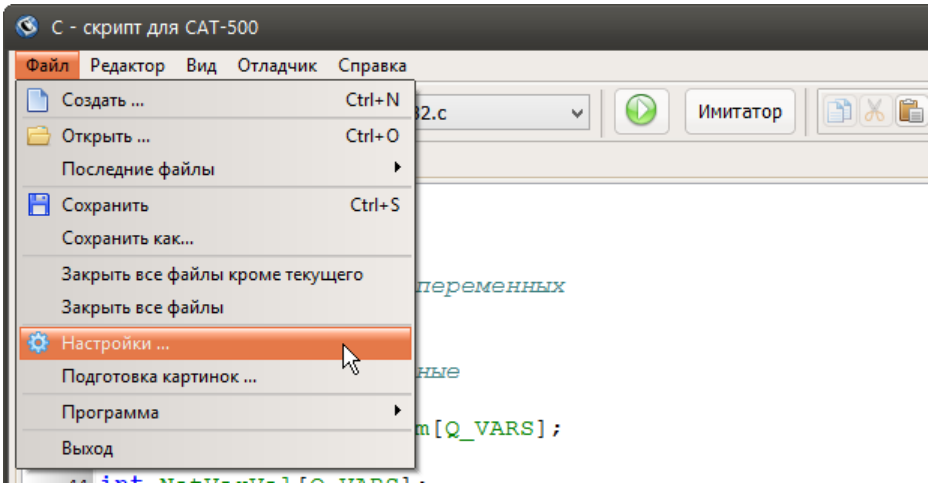
На вкладке "Настройки" можно указать номер сетевого порта, который используется для просмотра сетевых переменных и задать номер группы отображаемых сетевых переменных. Также можно задать просмотр всех групп (установить переключатель "Любая группа")



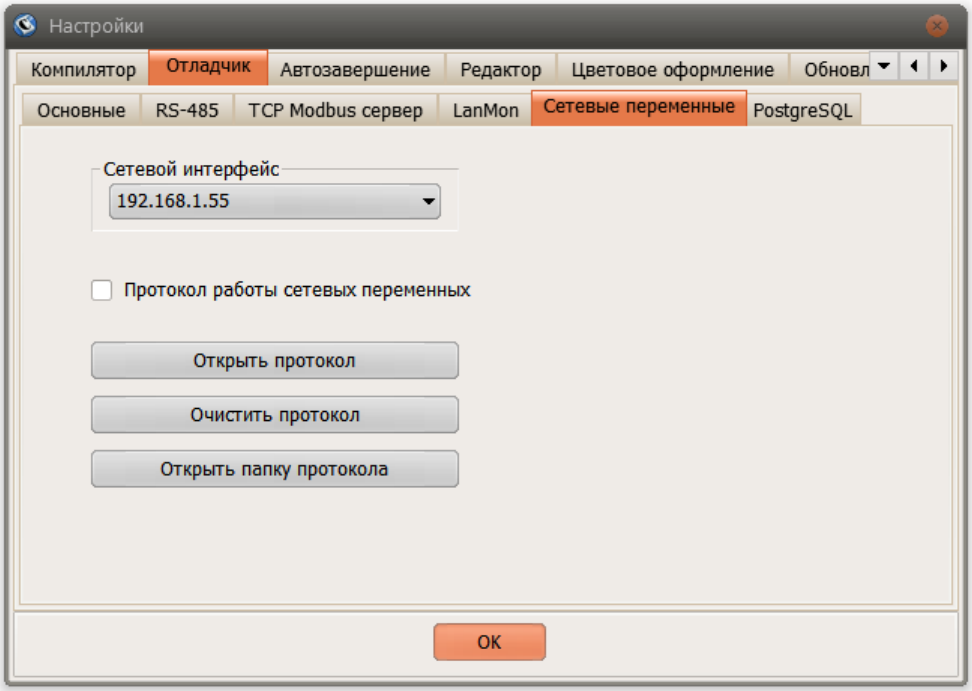
После изменения настроек следует нажать кнопку "Применить".

3.16.6. НАСТРОЙКИ СЕТЕВЫХ ПЕРЕМЕННЫХ

Для вызова настроек сетевых переменных необходимо вызвать окно настройки программы, выбрав пункт основного меню "**Файл/Настройки...**":



Далее следует перейти на вкладку "**Отладчик**" и затем на подвклаку "**Сетевые переменные**":
На подвкладке "**Сетевые переменные**" настраивается сетевой интерфейс, через которые среда разработки С-Скрипт работает с сетевыми переменными, а также можно включить и просмотреть текстовый протокол работы сетевых переменных.



Основные управляющие элементы интерфейса:

Элемент интерфейса	***Описание***
Сетевой интерфейс	Позволяет выбрать сетевой интерфейс компьютера через который среда разработки С-Скрипт работает с сетевыми переменными
Протокол работы с сетевыми переменными	Переключатель позволяет разрешить или запретить текстовый протокол работы с сетевыми переменными. Установленный переключатель разрешает ведение протокола
Открыть протокол	По нажатию на данную кнопку открывается окно, в котором можно просмотреть текстовый протокол работы с сетевыми переменными
Очистить протокол	По нажатию на данную кнопку выполняется очистка текстового протокола работы с сетевыми переменными

Открыть папку протокола	По нажатию на данную кнопку открывается папка, в которой расположен текстовый файл протокола работы сетевыми переменными

3.17. КЛИЕНТ СЕРВЕРА POSTGRE SQL

Контроллер CAT-500 может выступать в качестве клиента сервера баз данных Postgre SQL.
Работу с сервером баз данных контроллер CAT-500 выполняет через tcp подключение.
Используя описанные далее функции и переменные приложение может подключиться к серверу баз данных и выполнять SQL запросы.
Единственным ограничением в работе с сервером баз данных является небольшой объем оперативной памяти контроллера CAT-500, доступной приложению на языке С-скрипт.
Для работы с функциями доступа к базе данных необходимо включить заголовочный файл *<pgsql.h>*.

Приложению на языке С-скрипт доступны следующие переменные:

Переменная	***Описание***
char PgConnected	Переменная возвращает состояние подключения клиента к серверу Postgre SQL
unsigned char PgState	Переменная возвращает состояние клиента сервера Postgre SQL
unsigned short int PgColCount	Переменная возвращает количество столбцов(колонок) в ответе сервера Postgre SQL
unsigned short int PgRowCount	Переменная возвращает количество рядов в ответе сервера Postgre SQL
unsigned short int PgDataLen	Переменная возвращает ответа сервера Postgre SQL даже если принятый ответ оказался слишком длинным и привёл к возникновению ошибочного состояния (*PgState*=7)
unsigned int PgTimeout	Таймаут выполнения SQL запроса в миллисекундах. Устанавливается в функции PgOpen и в дальнейшем может быть изменён

Доступны следующие функции для работы с сервером баз данных Postgre SQL:

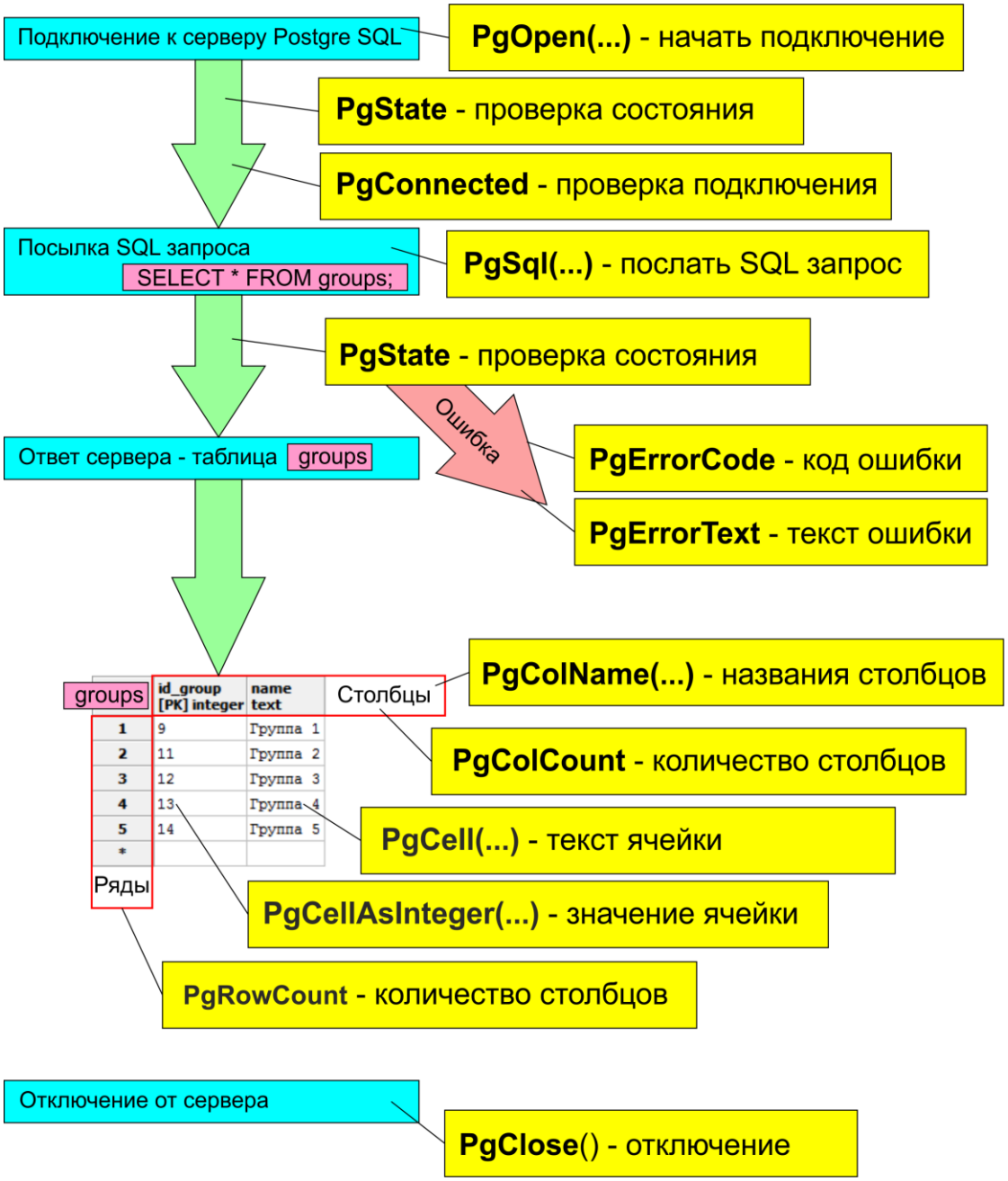
Функция	***Описание***
*char PgOpen (char * ip,unsigned short int port,unsigned int timeout,char * database,char * login,char * password,char win1251,char * buf,unsigned short int bufsize)*	Функция запускает процесс подключения к серверу баз данных Postgre SQL. Кроме основных параметров подключения в аргументах указывается буфер данных *buf*, который будет использоваться для получения ответа сервера Postgre SQL
void PgClose (void)	Функция выполняет отключение от сервера баз данных Postgre SQL
*char PgSql (char * sql)*	Функция запускает выполнение SQL запроса. Возвращаемое значение 1 подтверждает, что SQL запрос послан серверу и происходит выполнение запроса на сервере. Для получения результата следует воспользоваться проверкой переменной *PgState*. Как только в переменной появится ответ об успешном выполнении запроса, то полученный ответ сервера (таблица ответа) будет находится в буфере *buf* размером *bufsize*, заданном в функции *PgOpen*
*char * PgColName (unsigned short int col)*	Функция позволяет получить текстовое название колонки таблицы ответа, полученного от сервера на вызов функции *PgSql*. Функция возвращает 0, если нет ответа от сервера или заданный номер колоки col превышает количество колонок в таблице ответа.

*char PgCell (unsigned short int col,unsigned short int row,char * cell,unsigned short int cellsize)*	Функция позволяет прочитать текстовое значения ячейки с номером колонки *col* и номером ряда *row* из таблицы ответа сервера на SQL запрос, выполненный при помощи функции PgSql. Если при выполнении SQL запроса на сервере произошла ошибка, то возвращается значение 0. Если задан несуществующий номер колонки или ряда, то функция также возвращает нулевое значение. Если функция выполнена успешно, то массив символов *cell* размером *cellsize* содержит С-строку с текстом указанной ячейки.
*char PgCellAsInteger (unsigned short int col,unsigned short int row,int * value)*	Функция позволяет прочитать числовое значения ячейки с номером колонки *col* и номером PgCell ряда *row*. Возвращаемое возвращаемое значение совпадает с значением функции . В случае успеха целочисленное значение из указанной ячейки находится в переменной *value*
*char PgErrorCode (char * code)*	Получение пятибайтового кода ошибки выполнения SQL запроса на сервере в буфер code. В конце полученной строки будет завершающий ноль. Длина буфера строки должна быть не менее 6 байтов
*char PgErrorText (char * text,unsigned short int textsize)*	Получение текста ошибки выполнения SQL запроса на сервере. В конце полученной строки будет завершающий ноль. Если размер буфера недостаточный для получения всей строки, то в строку будет записана начальная часть текста ошибки

Алгоритм работы с сервером Postgre SQL

1. Начать подключение к серверу баз данных Postgre SQL используя функцию *PgOpen*.
2. Проверять состояние подключения, используя переменную *PgState*, значение *pgsOK* соответствует успешному подключению к серверу, значение *pgsDISCONNECTED* соответствует ошибке подключения к серверу. Если ошибка произошла на сервере, то можно получить код ошибки используя функцию *PgErrorCode* и текст ошибки используя функцию *PgErrorText*.
3. В случае успешного подключения начать выполнение SQL запроса функцией *PgSql*.
4. Проверять состояние выполнения запроса, используя переменную *PgState*. Пока переменная имеет значение *pgsSQL* запрос ешё выполняется. Как только появляется другое значение - выполнение завершено. Значение pgsSQL_OK соответствует успешно выполненому запросу - получен ответ от сервера с таблицей данных ответа, другие значения соответствуют ошибке выполнения SQL запроса: *pgsDISCONNECTED* - произошло тключение от сервера, *pgsSQL_BUFSIZE* - недостаточный размер буфера ответа, *pgsSQL_ERROR* - при выполнении SQL запроса на сервере произошла (код и текст ошибки можно прочитать функциями *PgErrorCode* и *PgErrorText*), *pgsSQL_TIMEOUT* - за время таймаута не получен ответ сервера, *pgsTCP_ERROR* - произошла ошибка в работе TCP сокета.
5. Если SQL запрос успешно выполнен, то буфер *buf*, заданный в функции *PgOpen* содержит таблицу ответа - результат запроса. Таблица имеет размерность: *PgColCount* - количество столбцов (колонок), *PgRowCount* - количество рядов. Для получения названий столбцов следует воспользоваться функцией *PgColName*. Для получения текста определённой ячейки таблицы следует воспользоваться функцией *PgCell*. Если ячейка имеет числовое значение, то можно использовать функцию *PgCellAsInteger*. Таблица ответа может иметь 0 строк и 0 столбцов если SQL запрос не предполагает наличие ответа.
6. При необходимости можно повторно посылать SQL запросы, проверив наличе подключения используя переменную PgConnected (пункт 3).
7. После завершения выполнения нужных запросов отключиться от сервера можно используя функцию PgClose.

Алгоритм работы с сервером Postgre SQL показан на рисунке ниже:



Рекомендации по работе с сервером PostgreSQL

1. Избегайте SQL запросов, возвращающих большое количество данных.
2. Выбирайте размер буфера для таблицы ответа достаточного размера.
3. Старайтесь больше использовать функции на сервере. Например, вместо того, чтобы использовать вызов INSERT для вставки данных в таблицу перенесите функционал вставки в собственную функцию на сервере и вызывайте эту функцию для добавления данных. Это позволить абстрагироваться от конкретных таблиц базы данных, что даст большую гибкость разработчику базы данных. Для чтения нужных данных также можно пользоваться вызовом собственной функции, хранимой на сервере.

3.17.1. PGCELL

```
#include <pgsql.h>

char PgCell(unsigned short int col,unsigned short int row,char * cell,unsigned
short int cellsize);
```

Функция *PgCell* позволяет прочитать текстовое значения ячейки с номером колонки *col* и номером ряда *row* из таблицы ответа сервера на SQL запрос, выполненный при помощи функции *PgSql*.
Аргументы функции:
unsigned short int col - номер колонки (0...65535) в таблице ответа.
unsigned short int row - номер ряда (0...65535) в таблице ответа.
*char * cell* - указатель на буфер, куда будет записан текст значения указанной ячейки ответа.
unsigned short int cellsize - размер буфера *cell*.

Возвращаемое значение:
1 - Успешно получено значение ячейки *col, row*
0 - Ошибка получения значение ячейки

Если при выполнении SQL запроса на сервере произошла ошибка, то возвращается значение 0.
Если задан несуществующий номер колонки или ряда, то функция также возвращает нулевое значение.
Если функция выполнена успешно, то массив символов *cell* размером *cellsize* содержит С-строку с названием колонки номер *col* и возвращаемое значение равно 1.
Чтение ячейки таблицы ответа может быть успешным только в случае, если переменная состояния *PgState* равна значению 6 (**Сервер успешно выполнил SQL запрос**).

В приведённом ниже примере выполняется подключение к существующей базе **"lift4"**. Затем выполняется чтение из таблицы **"public.groups"** и на дисплей выводится таблица ответа.

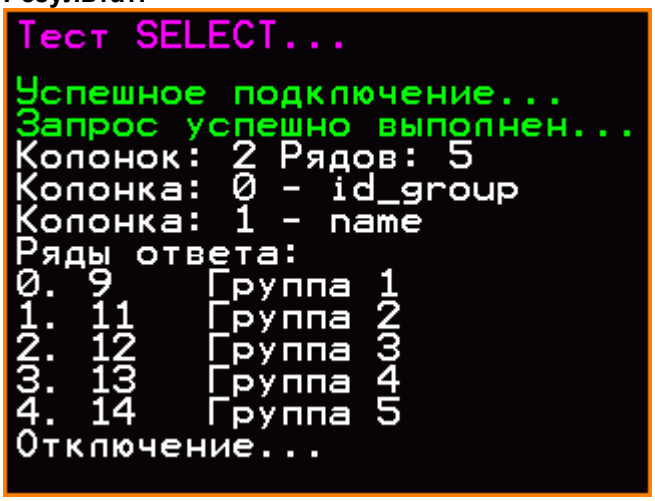
Пример:

```
#include <sat500.h>
#include <pgsql.h>

//Настройки подключения к Postgre SQL
#define PG_IP       "192.168.1.55"
#define PG_PORT     5432
#define PG_TIMEOUT  5000
#define PG_DATABASE "lift4"
#define PG_LOGIN    "postgres"
#define PG_PASSWORD "postgres"
#define PG_WIN1251  true
//Буфер для работы с Postgre SQL
char PgBuf[1024];
//SQL запрос
#define PG_SQL "SELECT * from groups LIMIT 6;"

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест SELECT...");
    if(!PgOpen(PG_IP,PG_PORT,PG_TIMEOUT,PG_DATABASE,
    PG_LOGIN,PG_PASSWORD,PG_WIN1251,PgBuf,sizeof(PgBuf)))
    {
```

```
        //Не удалось
        puts("\nОшибка PgOpen...");
        goto Exit;
    }
    //Ожидаем подключения
    FontColor=COLOR_WHITE;
    while(1)
    {
        if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    GotoXY(4,32);
    if(PgState==pgsOK)
    {
        //Подключены к базе данных
        FontColor=COLOR_GREEN;
        puts("Успешное подключение...");
        //Выполняем SQL запрос
        if(!PgSql(PG_SQL))
        {
            FontColor=COLOR_RED;
            puts("Не удалось выполнить запрос...");
            goto Exit;
        }
        //Ожидаем завершения выполнения запроса
        do
        {
            unsigned int t=GetTickCount();
            while((GetTickCount()-t)<100);
        }
        while(PgState==pgsSQL);
        if(PgState==pgsSQL_OK)
        {
            puts("\nЗапрос успешно выполнен...");
            FontColor=COLOR_WHITE;
            int colcount=PgColCount;
            int rowcount=PgRowCount;
            printf("\nКолонок: %u Рядов: %u",colcount,rowcount);
            for(int col=0;col<colcount;col++)
            {
                char * name=PgColName(col);
                if(name)
                {
                    printf("\nКолонка: %u - %s",col,name);
                }
            }
            puts("\nРяды ответа:");
            for(int row=0;row<rowcount;row++)
            {
                printf("\n%u. ",row);
                for(int col=0;col<colcount;col++)
                {
                    char cell[32];
                    if(PgCell(col,row,cell,sizeof(cell)))
                    {
                        printf("%-4s ",cell);
                    }
```

```
                }
            }
        }
        else
        {
            FontColor=COLOR_RED;
            puts("\nОшибка выполнения запроса\n");
            char text[128];
            FontColor=COLOR_YELLOW;
            if(PgErrorCode(text))
            {
                puts("\nКод ошибки: ");
                puts(text);
            }
            if(PgErrorText(text,sizeof(text)))
            {
                puts("\nОшибка: ");
                puts(text);
            }
        }
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("Ошибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
    while(1);
}
```

Результат:



3.17.2. PGCELLASINTEGER

```
#include <pgsql.h>

char PgCellAsInteger(unsigned short int col,unsigned short int row,int *
value);
```

Функция *PgCellAsInteger* позволяет прочитать числовое значения ячейки с номером колонки *col* и номером ряда *row* из таблицы ответа сервера на SQL запрос, выполненный при помощи функции *PgSql*.
Аргументы функции:
unsigned short int col - номер колонки (0...32767) в таблице ответа.
unsigned short int row - номер ряда (0...32767) в таблице ответа.
*int * value* - указатель на переменную которая будет содержать целочисленное значение из ячейке таблицы ответа.

Возвращаемое значение:
1 - Успешно получено числовое значение ячейки *col, row*
0 - Ошибка получения значение ячейки

В случае успеха целочисленное значение из указанной ячейки находится в переменной *value* и функция возвращает значение 1.
Если при выполнении SQL запроса произошла ошибка, то возвращается значение 0.
Если задан несуществующий номер колонки *col* или несуществующий номер ряда *row*, то функция также возвращает нулевое значение.
Вызов функции *PgCellAsInteger* может быть успешным только в случае, если переменная состояния *PgState* равна значению 6 (**Сервер успешно выполнил SQL запрос**).
Если указанная ячейка *col, row* присутствует в ответе, но содержит нечисловые символы, то функция возвращает 1, но значение *value* равно 0.

В приведённом ниже примере выполняется подключение к существующей базе **"lift4"**. Затем выполняется добавление строки в таблицу **"public.groups"** и на дисплей выводится ответа - значение поля **"id_group"** новой записи.

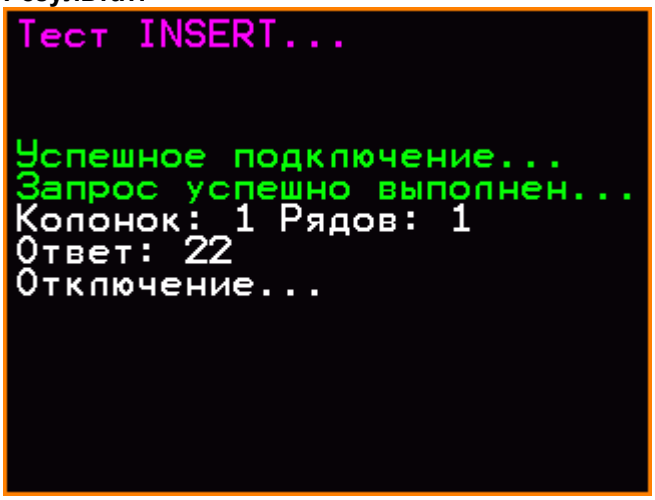
Пример:

```
#include <sat500.h>
#include <pgsql.h>

//Настройки подключения к Postgre SQL
#define PG_IP       "192.168.1.55"
#define PG_PORT     5432
#define PG_TIMEOUT  2000
#define PG_DATABASE "lift4"
#define PG_LOGIN    "postgres"
#define PG_PASSWORD "postgres"
#define PG_WIN1251  true
//Буфер для работы с Postgre SQL
char PgBuf[1024];
//SQL запрос
#define PG_SQL "INSERT INTO groups(name) VALUES('Группа 6') RETURNING
id_group;"

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест INSERT...");
    if(!PgOpen(PG_IP,PG_PORT,PG_TIMEOUT,PG_DATABASE,
```

```
PG_LOGIN,PG_PASSWORD,PG_WIN1251,PgBuf,sizeof(PgBuf)))
{
    //Не удалось
    puts("\nОшибка PgOpen...");
    goto Exit;
}
//Ожидаем подключения
while(1)
{
    if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
    unsigned int t=GetTickCount();
    while((GetTickCount()-t)<100);
}
GotoXY(4,64);
if(PgState==pgsOK)
{
    //Подключены к базе данных
    FontColor=COLOR_GREEN;
    puts("Успешное подключение...");
    //Выполняем SQL запрос
    if(!PgSql(PG_SQL))
    {
        FontColor=COLOR_RED;
        puts("Не удалось выполнить запрос...");
        goto Exit;
    }
    //Ожидаем завершения выполнения запроса
    do
    {
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    while(PgState==pgsSQL);
    if(PgState==pgsSQL_OK)
    {
        puts("\nЗапрос успешно выполнен...");
        FontColor=COLOR_WHITE;
        int colcount=PgColCount;
        int rowcount=PgRowCount;
        printf("\nКолонок: %u Рядов: %u",colcount,rowcount);
        int value;
        if(PgCellAsInteger(0,0,&value))
        {
           printf("\nОтвет: %d",value);
        }
    }
    else
    {
        FontColor=COLOR_RED;
        puts("\nОшибка выполнения запроса\n");
        char text[128];
        FontColor=COLOR_YELLOW;
        if(PgErrorCode(text))
        {
            puts("\nКод ошибки: ");
            puts(text);
        }
        if(PgErrorText(text,sizeof(text)))
```

```
            {
                puts("\nОшибка: ");
                puts(text);
            }
        }
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("Ошибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
    while(1);
}
```

Результат:



3.17.3. PGCLOSE

```
#include <pgsql.h>

void PgClose(void);
```

Функция выполняет безусловное отключение от сервера баз данных Postgre SQL.
Вызов функции безопасен и в случае если подключение к серверу отсутствует.

В приведённом ниже примере выполняется подключение к существующей базе **"lift4"** и сразу же выполняется отключение.

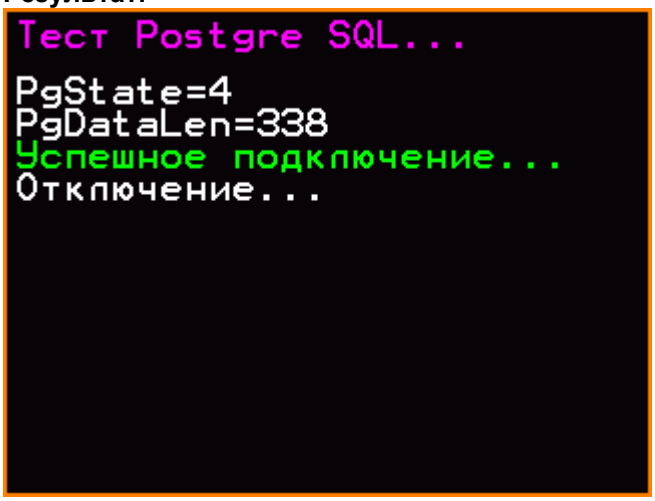
Пример:

```
#include <sat500.h>
#include <pgsql.h>
```

```
//Настройки подключения к Postgre SQL
#define PG_IP       "192.168.1.55"
#define PG_PORT     5432
#define PG_TIMEOUT  2000
#define PG_DATABASE "lift4"
#define PG_LOGIN    "postgres"
#define PG_PASSWORD "postgres"
#define PG_WIN1251  true
//Буфер для работы с Postgre SQL
char PgBuf[1024];

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест Postgre SQL...");
    if(!PgOpen(PG_IP,PG_PORT,PG_TIMEOUT,PG_DATABASE,
    PG_LOGIN,PG_PASSWORD,PG_WIN1251,PgBuf,sizeof(PgBuf)))
    {
        //Не удалось
        puts("\nОшибка PgOpen...");
        goto Exit;
    }
    //Ожидаем подключения
    FontColor=COLOR_WHITE;
    while(1)
    {
        GotoXY(4,32);
        printf("PgState=%u\nPgDataLen=%u",PgState,PgDataLen);
        ShowScr();
        if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    GotoXY(4,64);
    if(PgState==pgsOK)
    {
        //Подключены к базе данных
        FontColor=COLOR_GREEN;
        puts("Успешное подключение...");
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("Ошибка подключения...");
    }
    ShowScr();
    //....
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    ShowScr();
    PgClose();
Exit:;
    ShowScr();
```

```
    while(1);
}
```

Результат:



3.17.4. PGCOLCOUNT

```
#include <pgsql.h>

unsigned short int PgColCount;
```

Переменная *PgColCount* возвращает количество колонок из таблицы ответа сервера на SQL запрос, успешно выполненный при помощи функции *PgSql* .
Если при выполнении SQL запроса произошла ошибка, то возвращается значение 0.
Переменная доступна только для чтения.

В приведённом ниже примере выполняется подключение к существующей базе **"lift4"**. Затем выполняется чтение из таблицы **"public.groups"** и на дисплей выводится количество колонок и рядов таблицы ответа.

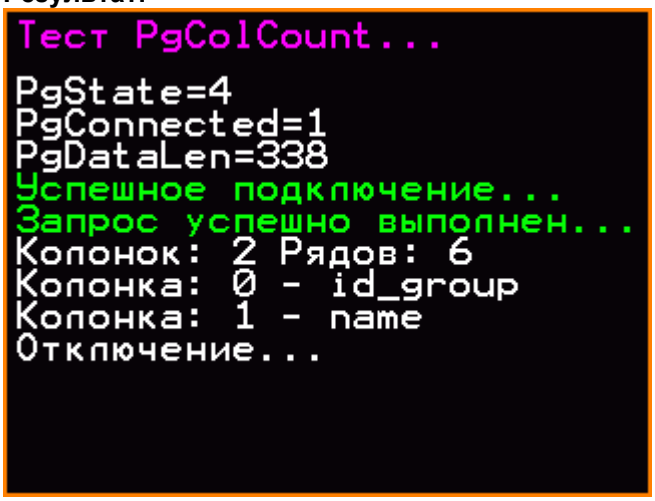
Пример:

```
#include <sat500.h>
#include <pgsql.h>

//Буфер для работы с Postgre SQL
char PgBuf[1024];

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест PgColCount...");
    if(!PgOpen("192.168.1.55",5432,2000,"lift4","postgres","postgres",
         true,PgBuf,sizeof(PgBuf)))
    {
        //Не удалось
        puts("\nОшибка PgOpen...");
        goto Exit;
    }
    //Ожидаем подключения
```

```
    FontColor=COLOR_WHITE;
    while(1)
    {
        GotoXY(4,32);
        printf("PgState=%u\nPgConnected=%u\nPgDataLen=%u",PgState,PgCon-
nected,PgDataLen);
        ShowScr();
        if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    GotoXY(4,64);
    if(PgConnected)
    {
        FontColor=COLOR_GREEN;
        puts("\nУспешное подключение...");
        //Выполняем SQL запрос
        if(!PgSql("SELECT * from groups LIMIT 6;"))
        {
            FontColor=COLOR_RED;
            puts("Не удалось выполнить запрос...");
            goto Exit;
        }
        //Ожидаем завершения выполнения запроса
        while(PgState==pgsSQL)
        {
            unsigned int t=GetTickCount();
            while((GetTickCount()-t)<100);
        }
        if(PgState==pgsSQL_OK)
        {
            puts("\nЗапрос успешно выполнен...");
            FontColor=COLOR_WHITE;
            int colcount=PgColCount;
            int rowcount=PgRowCount;
            printf("\nКолонок: %u Рядов: %u",colcount,rowcount);
            for(int col=0;col<colcount;col++)
            {
                char * name=PgColName(col);
                if(name)
                {
                    printf("\nКолонка: %u - %s",col,name);
                }
            }
        }
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("\nОшибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
```

```
    while(1);
}
```

Результат:



3.17.5. PGCOLNAME

```
#include <pgsql.h>

char * PgColName(unsigned short int col);
```

Функция позволяет получить текстовое название колонки таблицы ответа, полученного от сервера на вызов функции *PgSql*.
Аргументы функции:
unsigned short int col - номер колонки для которой запрашивается имя.

Возвращаемое значение - указатель на С-строку названия колонки. Сама строка располагается в полученном ответе сервера, находящемся в буфере *buf*, заданном в вызове *PgOpen*.

Функция возвращает 0, если нет ответа от сервера, или заданный номер колонки *col* превышает количество колонок в таблице ответа.
Функция может вернуть ненулевое значение только в том случае, если переменная состояния *PgState* равна значению 6 (**Сервер успешно выполнил SQL запрос**).

В приведённом ниже примере выполняется подключение к существующей базе **"lift4"**. Затем выполняется чтение из таблицы **"public.groups"** и на дисплей выводится таблица ответа.

Пример:

```
#include <sat500.h>
#include <pgsql.h>

//Настройки подключения к Postgre SQL
#define PG_IP       "192.168.1.55"
#define PG_PORT     5432
#define PG_TIMEOUT  5000
#define PG_DATABASE "lift4"
#define PG_LOGIN    "postgres"
#define PG_PASSWORD "postgres"
#define PG_WIN1251  true
//Буфер для работы с Postgre SQL
```

```
char PgBuf[1024];
//SQL запрос
#define PG_SQL "SELECT * from groups LIMIT 6;"

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест SELECT...");
    if(!PgOpen(PG_IP,PG_PORT,PG_TIMEOUT,PG_DATABASE,
    PG_LOGIN,PG_PASSWORD,PG_WIN1251,PgBuf,sizeof(PgBuf)))
    {
       //Не удалось
       puts("\nОшибка PgOpen...");
       goto Exit;
    }
    //Ожидаем подключения
    FontColor=COLOR_WHITE;
    while(1)
    {
        if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    GotoXY(4,32);
    if(PgState==pgsOK)
    {
        //Подключены к базе данных
        FontColor=COLOR_GREEN;
        puts("Успешное подключение...");
        //Выполняем SQL запрос
        if(!PgSql(PG_SQL))
        {
            FontColor=COLOR_RED;
            puts("Не удалось выполнить запрос...");
            goto Exit;
        }
        //Ожидаем завершения выполнения запроса
        do
        {
            unsigned int t=GetTickCount();
            while((GetTickCount()-t)<100);
        }
        while(PgState==pgsSQL);
        if(PgState==pgsSQL_OK)
        {
            puts("\nЗапрос успешно выполнен...");
            FontColor=COLOR_WHITE;
            int colcount=PgColCount;
            int rowcount=PgRowCount;
            printf("\nКолонок: %u Рядов: %u",colcount,rowcount);
            for(int col=0;col<colcount;col++)
            {
                char * name=PgColName(col);
                if(name)
                {
                    printf("\nКолонка: %u - %s",col,name);
```

```
                }
            }
            puts("\nРяды ответа:");
            for(int row=0;row<rowcount;row++)
            {
                printf("\n%u. ",row);
                for(int col=0;col<colcount;col++)
                {
                    char cell[32];
                    if(PgCell(col,row,cell,sizeof(cell)))
                    {
                        printf("%-4s ",cell);
                    }
                }
            }
        }
        else
        {
            FontColor=COLOR_RED;
            puts("\nОшибка выполнения запроса\n");
            char text[128];
            FontColor=COLOR_YELLOW;
            if(PgErrorCode(text))
            {
                puts("\nКод ошибки: ");
                puts(text);
            }
            if(PgErrorText(text,sizeof(text)))
            {
                puts("\nОшибка: ");
                puts(text);
            }
        }
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("Ошибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
    while(1);
}
```

Результат:

```
Тест SELECT...

Успешное подключение...
Запрос успешно выполнен...
Колонок: 2 Рядов: 5
Колонка: 0 - id_group
Колонка: 1 - name
Ряды ответа:
0. 9    Группа 1
1. 11   Группа 2
2. 12   Группа 3
3. 13   Группа 4
4. 14   Группа 5
Отключение...
```

3.17.6. PGCONNECTED

```
#include <pgsql.h>

char PgConnected;
```

Переменная *PgConnected* возвращает состояние подключение клиента к серверу Postgre SQL. Переменная может принимать следующие значения:

Значение	*Описание*
0	Нет подключения к серверу
1	Есть подключение к серверу

Переменная доступна только для чтения.

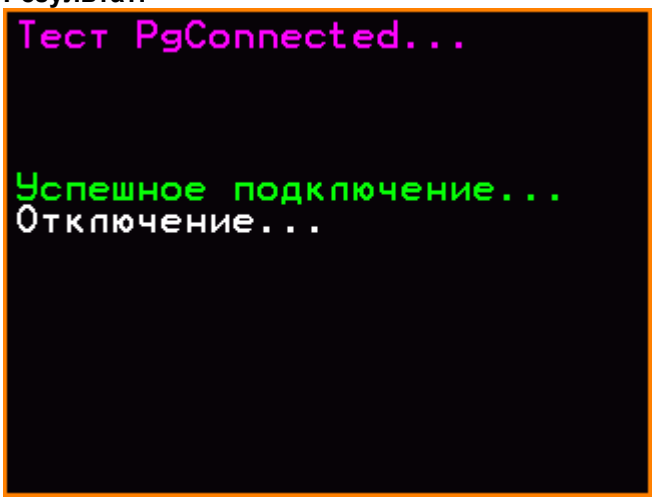
В приведённом далее примере выполняется подключение к серверу Postgre SQL при помощи вызова функции *PgOpen*.

Пример:

```
#include <sat500.h>
#include <pgsql.h>
//Буфер для работы с Postgre SQL
char PgBuf[1024];

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест PgConnected...");
    if(!PgOpen("192.168.1.55",5432,2000,"lift4","postgres","postgres",
         true,PgBuf,sizeof(PgBuf)))
    {
        //Не удалось
        puts("\nОшибка PgOpen...");
        goto Exit;
    }
    //Ожидаем подключения
    while(1)
```

```
    {
        if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    GotoXY(4,64);
    if(PgConnected)
    {
        FontColor=COLOR_GREEN;
        puts("\nУспешное подключение...");
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("\nОшибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
    while(1);
}
```

Результат:



3.17.7. PGDATALEN

```
#include <pgsql.h>

unsigned short int PgDataLen;
```

Переменная *PgDataLen* возвращает действительную длину ответа сервера Postgre SQL. Значением переменной можно воспользоваться при возникновении ошибки "**Сервер выполнил запрос, но переданный буфер слишком мал для приёма ответа сервера**" (*PgState* = 7). Следует посмотреть значение длины ответа и увеличить размер буфера *buf* в функции *PgOpen* или соответствующим образом изменить SQL запрос.
Переменная доступна только для чтения.

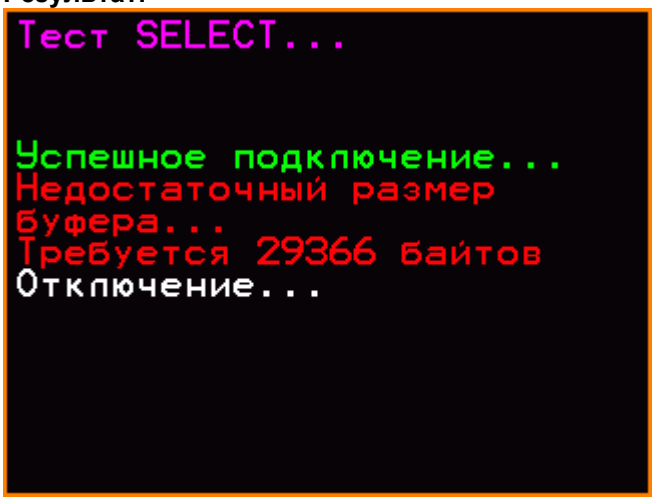
В приведённом ниже примере выполняется подключение к существующей базе **"lift4"**. Затем выполняется чтение таблицы **"public.alarms"**. Т.к. добавляемое значение *name* уже существует, то при добавлении на сервере возникает ошибка и на дисплей выводится код и текст ошибки сервера.

Пример:

```
#include <sat500.h>
#include <pgsql.h>

//Настройки подключения к Postgre SQL
#define PG_IP       "192.168.1.55"
#define PG_PORT     5432
#define PG_TIMEOUT  2000
#define PG_DATABASE "lift4"
#define PG_LOGIN    "postgres"
#define PG_PASSWORD "postgres"
#define PG_WIN1251  true
//Буфер для работы с Postgre SQL
char PgBuf[1024];
//SQL запрос
#define PG_SQL "SELECT * from alarms;"

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест SELECT...");
    if(!PgOpen(PG_IP,PG_PORT,PG_TIMEOUT,PG_DATABASE,
    PG_LOGIN,PG_PASSWORD,PG_WIN1251,PgBuf,sizeof(PgBuf)))
    {
        //Не удалось
        puts("\nОшибка PgOpen...");
        goto Exit;
    }
    //Ожидаем подключения
    while(1)
    {
        if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
    }
    GotoXY(4,64);
    if(PgState==pgsOK)
    {
        //Подключены к базе данных
        FontColor=COLOR_GREEN;
        puts("Успешное подключение...");
        //Выполняем SQL запрос
        if(!PgSql(PG_SQL))
        {
            FontColor=COLOR_RED;
            puts("Не удалось выполнить запрос...");
            goto Exit;
        }
        //Ожидаем завершения выполнения запроса
        while(PgState==pgsSQL)
        {
        }
        if(PgState==pgsSQL_OK)
```

```
        {
            puts("\nЗапрос успешно выполнен...");
            printf("\nPgDataLen=%d",PgDataLen);
        }
        if(PgState==pgsSQL_BUFSIZE)
        {
            FontColor=COLOR_RED;
            puts("\nНедостаточный размер\nбуфера...");
            printf("\nТребуется %u байтов",PgDataLen);
        }
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("Ошибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
    while(1);
}
```

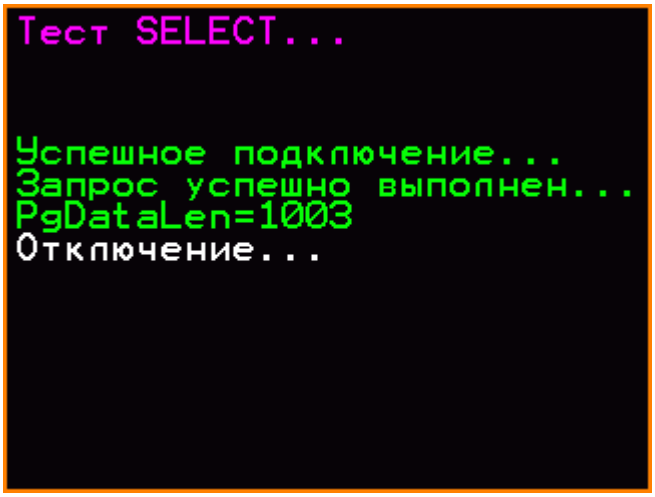
Результат:



В случае изменения SQL запроса:

```
//SQL запрос
#define PG_SQL "SELECT * from alarms limit 9;"
```

результат может быть успешным:



3.17.8. PGERRORCODE

```
include <pgsql.h>

char PgErrorCode (char * code);
```

Функция *PgErrorCode* заполняет посылаемый в аргументах буфер *code* пятибайтовой строкой с кодом ошибки выполнение SQL запроса сервером.
Аргументы функции:
*char * code* - указатель на буфер, который будет заполнен строкой кода ошибки. Размер буфера должен быть не менее 6 байтов - 5 символов кода ошибки и ноль в конце строки.

Возвращаемое значение:
1 - Успешно получена пятибайтовая С-строка кода ошибки
0 - Ошибка получения строки ошибки

ВНИМАНИЕ! Размер буфера должен быть не менее 6 символов.
Функция возвращает код ошибки только в том случае, если переменная состоянии *PgState* имеет значение 8 (**При выполнении SQL запроса на сервере произошла ошибка**). В этом случае буфер строки заполнен кодом ошибки и функция возвращает значение 1.
Во всех других случаях функция возвращает значение 0 и буфер заполнен пустой строкой.

Таблица кодов ошибок сервера Postgre SQL

Код ошибки	Имя условия
	Класс 00 — Успешное завершение
00000	successful_completion
	Класс 01 — Предупреждение
01000	warning
0100C	dynamic_result_sets_returned
01008	implicit_zero_bit_padding
01003	null_value_eliminated_in_set_function
01007	privilege_not_granted
01006	privilege_not_revoked
01004	string_data_right_truncation
01P01	deprecated_feature

	Класс 02 — Нет данных (это также класс предупреждений согласно стандарту SQL)
02000	no_data
02001	no_additional_dynamic_result_sets_returned
	Класс 03 — SQL-оператор ещё не завершён
03000	sql_statement_not_yet_complete
	Класс 08 — Исключение, связанное с подключением
08000	connection_exception
08003	connection_does_not_exist
08006	connection_failure
08001	sqlclient_unable_to_establish_sqlconnection
08004	sqlserver_rejected_establishment_of_sqlconnection
08007	transaction_resolution_unknown
08P01	protocol_violation
	Класс 09 — Исключение с действием триггера
09000	triggered_action_exception
	Класс 0A — Неподдерживаемая функциональность
0A000	feature_not_supported
	Класс 0B — Неверное начало транзакции
0B000	invalid_transaction_initiation
	Класс 0F — Исключение с указателем на данные
0F000	locator_exception
0F001	invalid_locator_specification
	Класс 0L — Неверный праводатель
0L000	invalid_grantor
0LP01	invalid_grant_operation
	Класс 0P — Неверное указание роли
0P000	invalid_role_specification
	Класс 0Z — Исключение диагностики
0Z000	diagnostics_exception
0Z002	stacked_diagnostics_accessed_without_active_handler
	Класс 20 — Case не найден
20000	case_not_found
	Класс 21 — Нарушение количества
21000	cardinality_violation
	Класс 22 — Исключение в данных
22000	data_exception
2202E	array_subscript_error
22021	character_not_in_repertoire
22008	datetime_field_overflow
22012	division_by_zero
22005	error_in_assignment
2200B	escape_character_conflict
22022	indicator_overflow
22015	interval_field_overflow
2201E	invalid_argument_for_logarithm
22014	invalid_argument_for_ntile_function
22016	invalid_argument_for_nth_value_function
2201F	invalid_argument_for_power_function
2201G	invalid_argument_for_width_bucket_function
22018	invalid_character_value_for_cast
22007	invalid_datetime_format
22019	invalid_escape_character
2200D	invalid_escape_octet

22025	invalid_escape_sequence
22P06	nonstandard_use_of_escape_character
22010	invalid_indicator_parameter_value
22023	invalid_parameter_value
22013	invalid_preceding_or_following_size
2201B	invalid_regular_expression
2201W	invalid_row_count_in_limit_clause
2201X	invalid_row_count_in_result_offset_clause
2202H	invalid_tablesample_argument
2202G	invalid_tablesample_repeat
22009	invalid_time_zone_displacement_value
2200C	invalid_use_of_escape_character
2200G	most_specific_type_mismatch
22004	null_value_not_allowed
22002	null_value_no_indicator_parameter
22003	numeric_value_out_of_range
2200H	sequence_generator_limit_exceeded
22026	string_data_length_mismatch
22001	string_data_right_truncation
22011	substring_error
22027	trim_error
22024	unterminated_c_string
2200F	zero_length_character_string
22P01	floating_point_exception
22P02	invalid_text_representation
22P03	invalid_binary_representation
22P04	bad_copy_file_format
22P05	untranslatable_character
2200L	not_an_xml_document
2200M	invalid_xml_document
2200N	invalid_xml_content
2200S	invalid_xml_comment
2200T	invalid_xml_processing_instruction
22030	duplicate_json_object_key_value
22032	invalid_json_text
22033	invalid_sql_json_subscript
22034	more_than_one_sql_json_item
22035	no_sql_json_item
22036	non_numeric_sql_json_item
22037	non_unique_keys_in_a_json_object
22038	singleton_sql_json_item_required
22039	sql_json_array_not_found
2203A	sql_json_member_not_found
2203B	sql_json_number_not_found
2203C	sql_json_object_not_found
2203D	too_many_json_array_elements
2203E	too_many_json_object_members
2203F	sql_json_scalar_required
	Класс 23 — Нарушение ограничения целостности
23000	integrity_constraint_violation
23001	restrict_violation
23502	not_null_violation
23503	foreign_key_violation
23505	unique_violation

23514	check_violation
23P01	exclusion_violation
	Класс 24 — Неверное состояние курсора
24000	invalid_cursor_state
	Класс 25 — Неверное состояние транзакции
25000	invalid_transaction_state
25001	active_sql_transaction
25002	branch_transaction_already_active
25008	held_cursor_requires_same_isolation_level
25003	inappropriate_access_mode_for_branch_transaction
25004	inappropriate_isolation_level_for_branch_transaction
25005	no_active_sql_transaction_for_branch_transaction
25006	read_only_sql_transaction
25007	schema_and_data_statement_mixing_not_supported
25P01	no_active_sql_transaction
25P02	in_failed_sql_transaction
25P03	idle_in_transaction_session_timeout
	Класс 26 — Неверное имя SQL-оператора
26000	invalid_sql_statement_name
	Класс 27 — Нарушение при изменении данных в триггере
27000	triggered_data_change_violation
	Класс 28 — Неверное указание авторизации
28000	invalid_authorization_specification
28P01	invalid_password
	Класс 2B — Зависимые описания привилегий всё ещё существуют
2B000	dependent_privilege_descriptors_still_exist
2BP01	dependent_objects_still_exist
	Класс 2D — Неверное завершение транзакции
2D000	invalid_transaction_termination
	Класс 2F — Исключение в подпрограмме SQL
2F000	sql_routine_exception
2F005	function_executed_no_return_statement
2F002	modifying_sql_data_not_permitted
2F003	prohibited_sql_statement_attempted
2F004	reading_sql_data_not_permitted
	Класс 34 — Неверное имя курсора
34000	invalid_cursor_name
	Класс 38 — Исключение во внешней подпрограмме
38000	external_routine_exception
38001	containing_sql_not_permitted
38002	modifying_sql_data_not_permitted
38003	prohibited_sql_statement_attempted
38004	reading_sql_data_not_permitted
	Класс 39 — Исключение при вызове внешней подпрограммы
39000	external_routine_invocation_exception
39001	invalid_sqlstate_returned
39004	null_value_not_allowed
39P01	trigger_protocol_violated
39P02	srf_protocol_violated
39P03	event_trigger_protocol_violated
	Класс 3B — Исключение точки сохранения
3B000	savepoint_exception
3B001	invalid_savepoint_specification
	Класс 3D — Неверное имя каталога

3D000	invalid_catalog_name
	Класс 3F — Неверное имя схемы
3F000	invalid_schema_name
	Класс 40 — Откат транзакции
40000	transaction_rollback
40002	transaction_integrity_constraint_violation
40001	serialization_failure
40003	statement_completion_unknown
40P01	deadlock_detected
	Класс 42 — Ошибка синтаксиса или нарушение правила доступа
42000	syntax_error_or_access_rule_violation
42601	syntax_error
42501	insufficient_privilege
42846	cannot_coerce
42803	grouping_error
42P20	windowing_error
42P19	invalid_recursion
42830	invalid_foreign_key
42602	invalid_name
42622	name_too_long
42939	reserved_name
42804	datatype_mismatch
42P18	indeterminate_datatype
42P21	collation_mismatch
42P22	indeterminate_collation
42809	wrong_object_type
428C9	generated_always
42703	undefined_column
42883	undefined_function
42P01	undefined_table
42P02	undefined_parameter
42704	undefined_object
42701	duplicate_column
42P03	duplicate_cursor
42P04	duplicate_database
42723	duplicate_function
42P05	duplicate_prepared_statement
42P06	duplicate_schema
42P07	duplicate_table
42712	duplicate_alias
42710	duplicate_object
42702	ambiguous_column
42725	ambiguous_function
42P08	ambiguous_parameter
42P09	ambiguous_alias
42P10	invalid_column_reference
42611	invalid_column_definition
42P11	invalid_cursor_definition
42P12	invalid_database_definition
42P13	invalid_function_definition
42P14	invalid_prepared_statement_definition
42P15	invalid_schema_definition
42P16	invalid_table_definition
42P17	invalid_object_definition

	Класс 44 — Нарушение WITH CHECK OPTION
44000	with_check_option_violation
	Класс 53 — Нехватка ресурсов
53000	insufficient_resources
53100	disk_full
53200	out_of_memory
53300	too_many_connections
53400	configuration_limit_exceeded
	Класс 54 — Превышение ограничения программы
54000	program_limit_exceeded
54001	statement_too_complex
54011	too_many_columns
54023	too_many_arguments
	Класс 55 — Объект не в требуемом состоянии
55000	object_not_in_prerequisite_state
55006	object_in_use
55P02	cant_change_runtime_param
55P03	lock_not_available
55P04	unsafe_new_enum_value_usage
	Класс 57 — Вмешательство оператора
57000	operator_intervention
57014	query_canceled
57P01	admin_shutdown
57P02	crash_shutdown
57P03	cannot_connect_now
57P04	database_dropped
	Класс 58 — Ошибка системы (ошибка, внешняя по отношению к PostgreSQL)
58000	system_error
58030	io_error
58P01	undefined_file
58P02	duplicate_file
	Класс 72 — Ошибка снимка
72000	snapshot_too_old
	Класс F0 — Ошибка файла конфигурации
F0000	config_file_error
F0001	lock_file_exists
	Класс HV — Ошибка обёртки сторонних данных (SQL/MED)
HV000	fdw_error
HV005	fdw_column_name_not_found
HV002	fdw_dynamic_parameter_value_needed
HV010	fdw_function_sequence_error
HV021	fdw_inconsistent_descriptor_information
HV024	fdw_invalid_attribute_value
HV007	fdw_invalid_column_name
HV008	fdw_invalid_column_number
HV004	fdw_invalid_data_type
HV006	fdw_invalid_data_type_descriptors
HV091	fdw_invalid_descriptor_field_identifier
HV00B	fdw_invalid_handle
HV00C	fdw_invalid_option_index
HV00D	fdw_invalid_option_name
HV090	fdw_invalid_string_length_or_buffer_length
HV00A	fdw_invalid_string_format
HV009	fdw_invalid_use_of_null_pointer

HV014	fdw_too_many_handles
HV001	fdw_out_of_memory
HV00P	fdw_no_schemas
HV00J	fdw_option_name_not_found
HV00K	fdw_reply_handle
HV00Q	fdw_schema_not_found
HV00R	fdw_table_not_found
HV00L	fdw_unable_to_create_execution
HV00M	fdw_unable_to_create_reply
HV00N	fdw_unable_to_establish_connection
	Класс P0 — Ошибка PL/pgSQL
P0000	plpgsql_error
P0001	raise_exception
P0002	no_data_found
P0003	too_many_rows
P0004	assert_failure
	Класс XX — Внутренняя ошибка
XX000	internal_error
XX001	data_corrupted
XX002	index_corrupted

В случае отсутствия ошибки в приведённой таблице следует обратиться к документации по серверу Postgre SQL, доступной на следующей странице:
https://postgrespro.ru/docs/postgresql

В приведённом ниже примере выполняется подключение к существующей базе **"lift4"**. Затем выполняется добавление строки в таблицу **"public.groups"**. Т.к. добавляемое значение name уже существует, то при добавлении на сервере возникает ошибка и на дисплей выводится код и текст ошибки сервера.

Пример:

```
#include <sat500.h>
#include <pgsql.h>

//Настройки подключения к Postgre SQL
#define PG_IP       "192.168.1.55"
#define PG_PORT     5432
#define PG_TIMEOUT  2000
#define PG_DATABASE "lift4"
#define PG_LOGIN    "postgres"
#define PG_PASSWORD "postgres"
#define PG_WIN1251  true
//Буфер для работы с Postgre SQL
char PgBuf[1024];
//SQL запрос
#define PG_SQL "INSERT INTO groups(name) VALUES('Группа 6') RETURNING
id_group;"

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест INSERT...");
    if(!PgOpen(PG_IP,PG_PORT,PG_TIMEOUT,PG_DATABASE,
```

```
PG_LOGIN,PG_PASSWORD,PG_WIN1251,PgBuf,sizeof(PgBuf)))
{
    //Не удалось
    puts("\nОшибка PgOpen...");
    goto Exit;
}
//Ожидаем подключения
while(1)
{
    if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
    unsigned int t=GetTickCount();
    while((GetTickCount()-t)<100);
}
GotoXY(4,64);
if(PgState==pgsOK)
{
    //Подключены к базе данных
    FontColor=COLOR_GREEN;
    puts("Успешное подключение...");
    //Выполняем SQL запрос
    if(!PgSql(PG_SQL))
    {
        FontColor=COLOR_RED;
        puts("Не удалось выполнить запрос...");
        goto Exit;
    }
    //Ожидаем завершения выполнения запроса
    do
    {
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    while(PgState==pgsSQL);
    if(PgState==pgsSQL_OK)
    {
        puts("\nЗапрос успешно выполнен...");
        FontColor=COLOR_WHITE;
        int colcount=PgColCount;
        int rowcount=PgRowCount;
        printf("\nКолонок: %u Рядов: %u",colcount,rowcount);
        int value;
        if(PgCellAsInteger(0,0,&value))
        {
           printf("\nОтвет: %d",value);
        }
    }
    else
    {
        FontColor=COLOR_RED;
        puts("\nОшибка выполнения запроса\n");
        char text[128];
        FontColor=COLOR_YELLOW;
        if(PgErrorCode(text))
        {
            puts("\nКод ошибки: ");
            puts(text);
        }
        if(PgErrorText(text,sizeof(text)))
```

```
            {
                puts("\nОшибка: ");
                puts(text);
            }
        }
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("Ошибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
    while(1);
}
```

Результат:

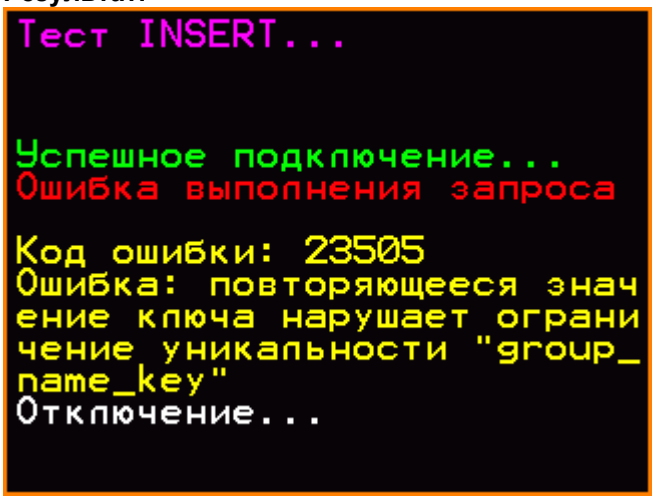


Таблица "groups" определена следующим образом:

```
CREATE TABLE public.groups
(
  id_group integer NOT NULL DEFAULT nextval('group_id_group_seq'::regclass),
  name text,
  CONSTRAINT group_pkey PRIMARY KEY (id_group),
  CONSTRAINT group_name_key UNIQUE (name)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.groups
  OWNER TO postgres;

CREATE SEQUENCE public.group_id_group_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 23
  CACHE 1;
ALTER TABLE public.group_id_group_seq
  OWNER TO postgres;
```

3.17.9. PGERRORTEXT

```
include <pgsql.h>

char PgErrorText(char * text,unsigned short int textsize);
```

Функция *PgErrorText* позволяет получить текст ошибки выполнения SQL запроса на сервере.
Для этого в аргументах функции передаётся буфер *text* для заполнения текстом ошибки. Размер буфера *textsize* рекомендуется задавать не менее 128 байтов.
Аргументы функции:
*char * text* - указатель на буфер, который будет заполнен строкой ошибки из ответа сервера
unsigned short int textsize - размер буфера в байтах

Возвращаемое значение:
1 - Успешно получена строка ошибки
0 - Ошибка получения строки ошибки

Функция возвращает строку ошибки только в том случае, если переменная состоянии *PgState* имеет значение 8 (**При выполнении SQL запроса на сервере произошла ошибка**). В этом случае буфер строки заполнен текстом ошибки и функция возвращает значение 1.
Во всех других случаях функция возвращает значение 0 и буфер заполнен пустой строкой.
Если размер буфера недостаточный для получения всей строки, то в строку будет записана начальная часть текста ошибки.

В приведённом ниже примере выполняется подключение к существующей базе **"lift4"**. Затем выполняется добавление строки в таблицу **"public.groups"**. Т.к. добавляемое значение name уже существует, то при добавлении на сервере возникает ошибка и на дисплей выводится код и текст ошибки сервера.

Пример:

```
#include <sat500.h>
#include <pgsql.h>

//Настройки подключения к Postgre SQL
#define PG_IP       "192.168.1.55"
#define PG_PORT     5432
#define PG_TIMEOUT  2000
#define PG_DATABASE "lift4"
#define PG_LOGIN    "postgres"
#define PG_PASSWORD "postgres"
#define PG_WIN1251  true
//Буфер для работы с Postgre SQL
char PgBuf[1024];
//SQL запрос
#define PG_SQL "INSERT INTO groups(name) VALUES('Группа 6') RETURNING
id_group;"

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест INSERT...");
    if(!PgOpen(PG_IP,PG_PORT,PG_TIMEOUT,PG_DATABASE,
```

```
PG_LOGIN,PG_PASSWORD,PG_WIN1251,PgBuf,sizeof(PgBuf)))
{
    //Не удалось
    puts("\nОшибка PgOpen...");
    goto Exit;
}
//Ожидаем подключения
while(1)
{
    if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
    unsigned int t=GetTickCount();
    while((GetTickCount()-t)<100);
}
GotoXY(4,64);
if(PgState==pgsOK)
{
    //Подключены к базе данных
    FontColor=COLOR_GREEN;
    puts("Успешное подключение...");
    //Выполняем SQL запрос
    if(!PgSql(PG_SQL))
    {
        FontColor=COLOR_RED;
        puts("Не удалось выполнить запрос...");
        goto Exit;
    }
    //Ожидаем завершения выполнения запроса
    do
    {
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    while(PgState==pgsSQL);
    if(PgState==pgsSQL_OK)
    {
        puts("\nЗапрос успешно выполнен...");
        FontColor=COLOR_WHITE;
        int colcount=PgColCount;
        int rowcount=PgRowCount;
        printf("\nКолонок: %u Рядов: %u",colcount,rowcount);
        int value;
        if(PgCellAsInteger(0,0,&value))
        {
           printf("\nОтвет: %d",value);
        }
    }
    else
    {
        FontColor=COLOR_RED;
        puts("\nОшибка выполнения запроса\n");
        char text[128];
        FontColor=COLOR_YELLOW;
        if(PgErrorCode(text))
        {
            puts("\nКод ошибки: ");
            puts(text);
        }
        if(PgErrorText(text,sizeof(text)))
```

```
            {
                puts("\nОшибка: ");
                puts(text);
            }
        }
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("Ошибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
    while(1);
}
```

Результат:

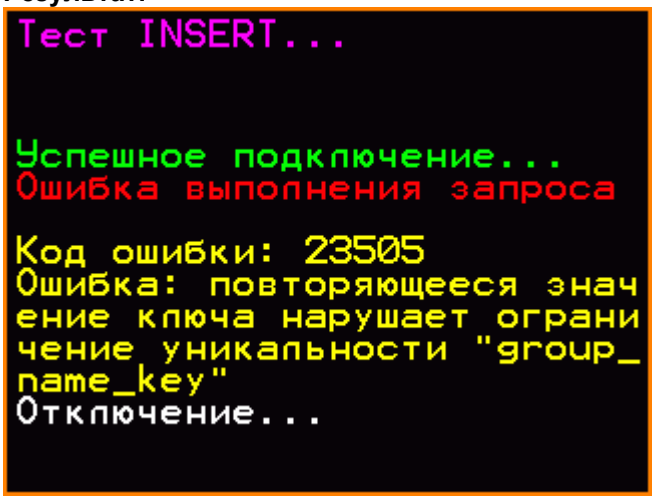


Таблица "groups" определена следующим образом:

```
CREATE TABLE public.groups
(
  id_group integer NOT NULL DEFAULT nextval('group_id_group_seq'::regclass),
  name text,
  CONSTRAINT group_pkey PRIMARY KEY (id_group),
  CONSTRAINT group_name_key UNIQUE (name)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.groups
  OWNER TO postgres;

CREATE SEQUENCE public.group_id_group_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 23
  CACHE 1;
ALTER TABLE public.group_id_group_seq
  OWNER TO postgres;
```

3.17.10. PGOPEN

```
#include <pgsql.h>

char PgOpen(char * ip,unsigned short int port,unsigned int timeout,char * database,char * login,char * password,char win1251,char * buf,unsigned short int bufsize);
```

Функция *PgOpen* запускает процесс подключения к серверу баз данных Postgre SQL.
Аргументы функции:
*char * ip* - строка, которая содержит доменное имя сервера Postgre SQL или IP адрес сервера в текстовом представлении
unsigned short int port - номер сетевого порта на котором работает удалённый сервер Postgre SQL. В случае задания нулевого номера порта будет использован порт 5432 (порт сервера Postgre SQL по умолчанию).
unsigned int timeout - таймаут на операцию подключения к серверу и выполнения SQL запроса сервером в миллисекундах.
*char * database* - название базы данных к которой необходимо выполнить подключение на сервере.
*char * login* - логин учётной записи, предназначенной для подключения контроллера CAT-500 на сервере Postgre SQL.
*char * password* - пароль учётной записи, предназначенной для подключения контроллера CAT-500 на сервере Postgre SQL.
char *win1251* - задаёт необходимость перекодировки текстовых строк сервера в кодировку windows 1251. Перекодировку выполняет сервер. Если в качестве параметра задана единица, то сервер будет выполнять перекодировку. Если задан параметр 0, то перекодировку выполняться не будет и приложение будет получать текстовые строки с кодировкой используемой на сервере (например UTF8).
*char * buf* - указатель на буфер куда будет записан ответ сервера Postgre SQL. Буфер всегда хранит последний ответ сервера Postgre SQL.
unsigned short int bufsize - размер буфера ответа сервера Postgre SQL. Минимальнов возможное значение - 256 байтов. Рекомендуется не менее 1024 байтов.

Возвращаемое значение:
1 - Успешно началось подключение к серверу Postgre SQL
0 - Ошибка. Не удалось начать подключение - не задан адрес сервера, отсутствуют свободные TCP сокеты у контроллера CAT-500 или задан слишком маленький размер буфера *buf*.

После того как функция *PgOpen* успешно выполнена подключение к серверу ещё не установлено. В течении установленного значения таймаута клиент сервера будет пытаться подключиться и авторизироваться на сервере Postgre SQL.
Для определения результата подключения необходимо проверять переменную *PgState*.
Переменная примет одно из следующих значений:

PgState	PgState (константа)	Описание ошибки
2	pgsDISCONNECTED	Не удалось подключиться к серверу. Если имеется ответ сервера - ненулевое значение *PgDataLen* то можно получить код ошибки сервера, используя функцию *PgErrorCode* и текст ошибки, используя функцию *PgErrorText*. Если контроллеру CAT-500 не удалось установить TCP подключение к серверу PostgreSQL, то функция *PgErrorCode* возвращает неуспешное значение, а функция *PgErrorText* возвращает успешное значение и текстовую строку "*Server not found*"
4	pgsOK	Подключены и авторизованы на сервере.

		В буфере buf содержится ответа сервера длиной *PgDataLen*. Из этого ответа можно получить параметры, переданные сервером Postgre SQL. Используйте документацию Postgre SQL, доступную на официальном сайте: https://www.postgresql.org/docs/ Пример 2 ниже показывает возможность получения параметров, переданных сервером в момент подключения

ВНИМАНИЕ! Для авторизации на сервере Postgre SQL используется метод с использованием алгоритма MD5. Метод авторизации в сервере задаётся в файле конфигурации "pg_hba.conf":

```
# TYPE   DATABASE   USER   ADDRESS        METHOD
# IPv4 local connections:
host     all        all    192.168.1.0/24 md5
```

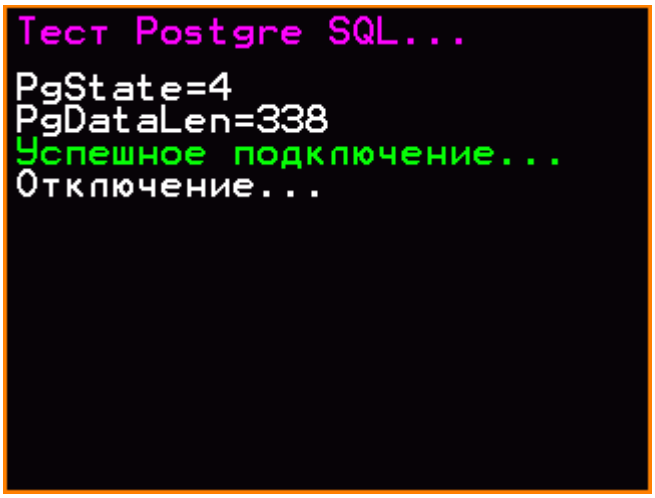
В приведённом ниже примере выполняется подключение к существующей базе **"lift4"**.

Пример 1:

```
#include <sat500.h>
#include <pgsql.h>

//Настройки подключения к Postgre SQL
#define PG_IP       "192.168.1.55"
#define PG_PORT     5432
#define PG_TIMEOUT  2000
#define PG_DATABASE "lift4"
#define PG_LOGIN    "postgres"
#define PG_PASSWORD "postgres"
#define PG_WIN1251  true
//Буфер для работы с Postgre SQL
char PgBuf[1024];

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест Postgre SQL...");
    if(!PgOpen(PG_IP,PG_PORT,PG_TIMEOUT,PG_DATABASE,
    PG_LOGIN,PG_PASSWORD,PG_WIN1251,PgBuf,sizeof(PgBuf)))
    {
        //Не удалось
        puts("\nОшибка PgOpen...");
        goto Exit;
    }
    //Ожидаем подключения
    FontColor=COLOR_WHITE;
    while(1)
    {
        GotoXY(4,32);
        printf("PgState=%u\nPgDataLen=%u",PgState,PgDataLen);
        ShowScr();
        if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    GotoXY(4,64);
    if(PgState==pgsOK)
```

```
    {
        //Подключены к базе данных
        FontColor=COLOR_GREEN;
        puts("Успешное подключение...");
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("Ошибка подключения...");
    }
    //Проверить ошибку
    FontColor=COLOR_YELLOW;
    char code[6];
    if(PgErrorCode(code))
    {
        printf("\nКод ошибки: %s",code);
    }
    char text[128];
    if(PgErrorText(text,sizeof(text)))
    {
        puts("\nОшибка: ");
        puts(text);
    }
    ShowScr();
    //....
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    ShowScr();
    PgClose();
Exit:;
    ShowScr();
    while(1);
}
```

Результат:
Если все настройки заданный правильно то будет получен следующий результат:



Если задано неправильное имя базы данных, то подключение будет успешным, но можно прочитать ошибку:

```
Тест Postgre SQL...

PgState=4
PgDataLen=94
Успешное подключение...
Код ошибки: 3D000
Ошибка: база данных "lift"
 не существует
Отключение...
```

В случае задания неправильного логина или пароля подключение будет неуспешным, и можно просмотреть ошибки:

```
Тест Postgre SQL...

PgState=2
PgDataLen=112
Ошибка подключения...
Код ошибки: 28P01
Ошибка: пользователь "Иван
ов" не прошёл проверку под
линности (по паролю)
Отключение...
```

В случае задания несуществующего адреса сервера в аргументах функции результат будет следующим:

```
Тест Postgre SQL...

PgState=2
PgDataLen=0
Ошибка подключения...
Ошибка: Server not found
Отключение...
```

В приведённом ниже примере выполняется подключение к существующей базе **"lift4"** и в случае успешного подключения на дисплей выводятся параметры полученные от сервера в момент подключения.

Пример 2:

```
#include <sat500.h>
#include <string.h>
#include <pgsql.h>

typedef unsigned char      BYTE;
typedef short int          WORD;
typedef unsigned int       UINT;

#define PG_TYPE_ParameterStatus 'S'

//Настройки подключения к Postgre SQL
#define PG_IP       "192.168.1.55"
#define PG_PORT     5432
#define PG_TIMEOUT  2000
#define PG_DATABASE "lift4"
#define PG_LOGIN    "postgres"
#define PG_PASSWORD "postgres"
#define PG_WIN1251  true
//Буфер для работы с Postgre SQL
char PgBuf[1024];

//4-х байтовая переменная с сетевым порядком байтов
UINT htonl(BYTE * data)
{
    union
    {
        UINT v;
        BYTE vbytes[4];
    };
    vbytes[0]=data[3];
    vbytes[1]=data[2];
    vbytes[2]=data[1];
    vbytes[3]=data[0];
    return v;
}
//Просмотр параметров Postgre SQL
void ShowPgParams(void)
{
    WORD datalen=PgDataLen;
    BYTE * data=PgBuf;
    while(datalen>0)
    {
        BYTE pg_type=*data++;
        datalen--;
        switch(pg_type)
        {
            case PG_TYPE_ParameterStatus:   //'S'
            //(B) в сообщении передаётся состояние параметра времени выполне-
ния
            {
                //*data - Длина содержимого сообщения в байтах, включая само
поле длины
```

```
                char * par=data+4;
                //Далее 2 с-строки
                //par-название параметра
                char * val=par+strlen(par)+1;
                char str[256];
                str[0]=0;
                if((strlen(par)+strlen(val)+2)<256)
                {
                    sprintf(str,"\n%s=%s",par,val);
                }
                puts(str);
                break;
            }
        }
        //Длина содержимого сообщения в байтах, включая само поле длины
        //Считываем длину
        UINT len=htonl(data);
        data+=len;
        datalen-=len;
    }
}
void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест Postgre SQL...");
    if(!PgOpen(PG_IP,PG_PORT,PG_TIMEOUT,PG_DATABASE,
    PG_LOGIN,PG_PASSWORD,PG_WIN1251,PgBuf,sizeof(PgBuf)))
    {
        //Не удалось
        puts("\nОшибка PgOpen...");
        goto Exit;
    }
    //Ожидаем подключения
    FontColor=COLOR_WHITE;
    while(1)
    {
        GotoXY(4,32);
        printf("PgState=%u\nPgDataLen=%u",PgState,PgDataLen);
        ShowScr();
        if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    GotoXY(4,64);
    if(PgState==pgsOK)
    {
        //Успешное подключение.
        //Подключены к базе данных
        FontColor=COLOR_GREEN;
        //Просмотр параметров Postgre SQL
        ClrScr();
        ShowPgParams();
    }
    else
    {
        //Произошла ошибка при подключении
```

```
        FontColor=COLOR_RED;
        puts("Ошибка подключения...");
        //Проверить ошибку
        FontColor=COLOR_YELLOW;
        char code[6];
        if(PgErrorCode(code))
        {
            printf("\nКод ошибки: %s",code);
        }
        char text[128];
        if(PgErrorText(text,sizeof(text)))
        {
            puts("\nОшибка: ");
            puts(text);
        }
    }
    //Отключение
    ShowScr();
    PgClose();
Exit:;
    ShowScr();
    while(1);
}
```

Результат:

```
application_name=SAT500
client_encoding=WIN1251
DateStyle=ISO, DMY
integer_datetimes=on
IntervalStyle=postgres
is_superuser=on
server_encoding=UTF8
server_version=9.5.0
session_authorization=post
gres
standard_conforming_string
s=on
TimeZone=Europe/Moscow
```

3.17.11. PGROWCOUNT

```
#include <pgsql.h>

unsigned short int PgRowCount;
```

Переменная *PgRowCount* возвращает количество оядов из таблицы ответа сервера на SQL запрос, успешно выполненный при помощи функции *PgSql*.
Если при выполнении SQL запроса произошла ошибка, то возвращается значение 0.
Переменная доступна только для чтения.

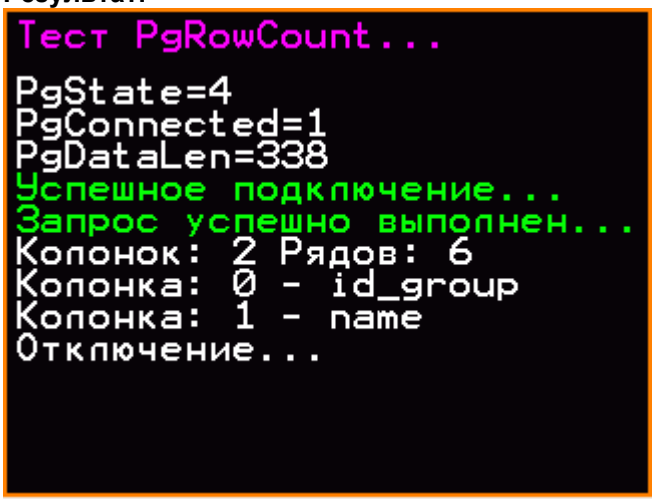
В приведённом ниже примере выполняется подключение к существующей базе **"lift4"**. Затем выполняется чтение из таблицы **"public.groups"** и на дисплей выводится количество колонок и рядов таблицы ответа.

Пример:

```
#include <sat500.h>
#include <pgsql.h>

//Буфер для работы с Postgre SQL
char PgBuf[1024];

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест PgRowCount...");
    if(!PgOpen("192.168.1.55",5432,2000,"lift4","postgres","postgres",
         true,PgBuf,sizeof(PgBuf)))
    {
        //Не удалось
        puts("\nОшибка PgOpen...");
        goto Exit;
    }
    //Ожидаем подключения
    FontColor=COLOR_WHITE;
    while(1)
    {
        GotoXY(4,32);
        printf("PgState=%u\nPgConnected=%u\nPgDataLen=%u",PgState,PgCon-
nected,PgDataLen);
        ShowScr();
        if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    GotoXY(4,64);
    if(PgConnected)
    {
        FontColor=COLOR_GREEN;
        puts("\nУспешное подключение...");
        //Выполняем SQL запрос
        if(!PgSql("SELECT * from groups LIMIT 6;"))
        {
            FontColor=COLOR_RED;
            puts("Не удалось выполнить запрос...");
            goto Exit;
        }
        //Ожидаем завершения выполнения запроса
        while(PgState==pgsSQL)
        {
            unsigned int t=GetTickCount();
            while((GetTickCount()-t)<100);
        }
        if(PgState==pgsSQL_OK)
        {
            puts("\nЗапрос успешно выполнен...");
            FontColor=COLOR_WHITE;
            int colcount=PgColCount;
            int rowcount=PgRowCount;
```

```
            printf("\nКолонок: %u Рядов: %u",colcount,rowcount);
            for(int col=0;col<colcount;col++)
            {
                char * name=PgColName(col);
                if(name)
                {
                    printf("\nКолонка: %u - %s",col,name);
                }
            }
        }
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("\nОшибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
    while(1);
}
```

Результат:



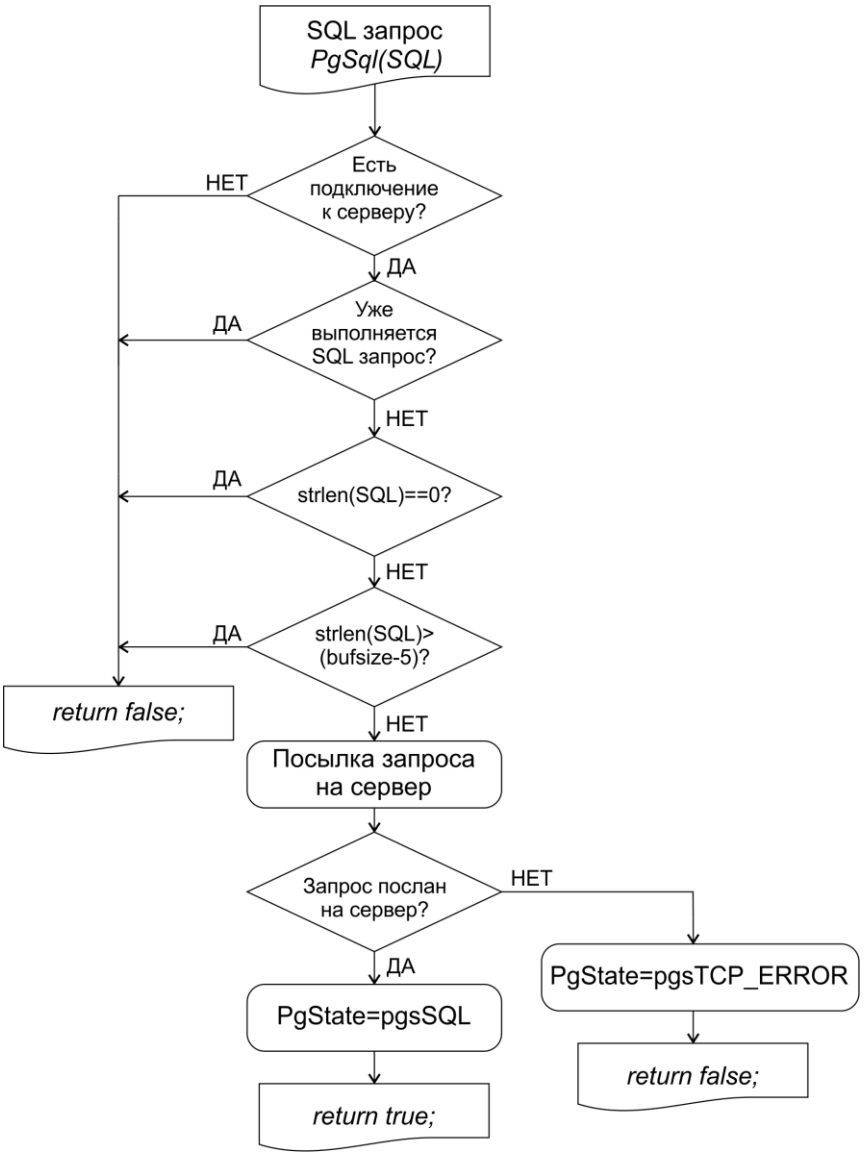
3.17.12. PGSQL

```
#include <pgsql.h>

char PgSql(char * sql);
```

Функция *PgSql* запускает выполнение SQL запроса на сервере.
Аргументы функции:
*char * sql* - С-строка, содержащая текст SQL запроса

Возвращаемое значение:
1 - Успешно началось выполнение SQL на сервере Postgre SQL
0 - Ошибка.

Возвращаемое значение 1 подтверждает, что SQL запрос послан серверу и происходит выполнение запроса на сервере.
В случае, если при вызове функции возвращается нулевое значение, то это обозначает что не удалось послать запрос на сервер Postgre SQL.
Причины ошибки вызова функции могут быть:
- Отсутствует подключение к серверу Postgre SQL
- Уже выполняется SQL запрос (повторный вызов *PgSql* без ожидания результата)
- Пустая строка SQL запроса в аргументе
- Длина строки SQL запроса превышает ограничение - (размер буфера ответа -5). Буфер ответа задан в вызове функции
Блок-схема вызова функции *PgSql* показана на рисунке:

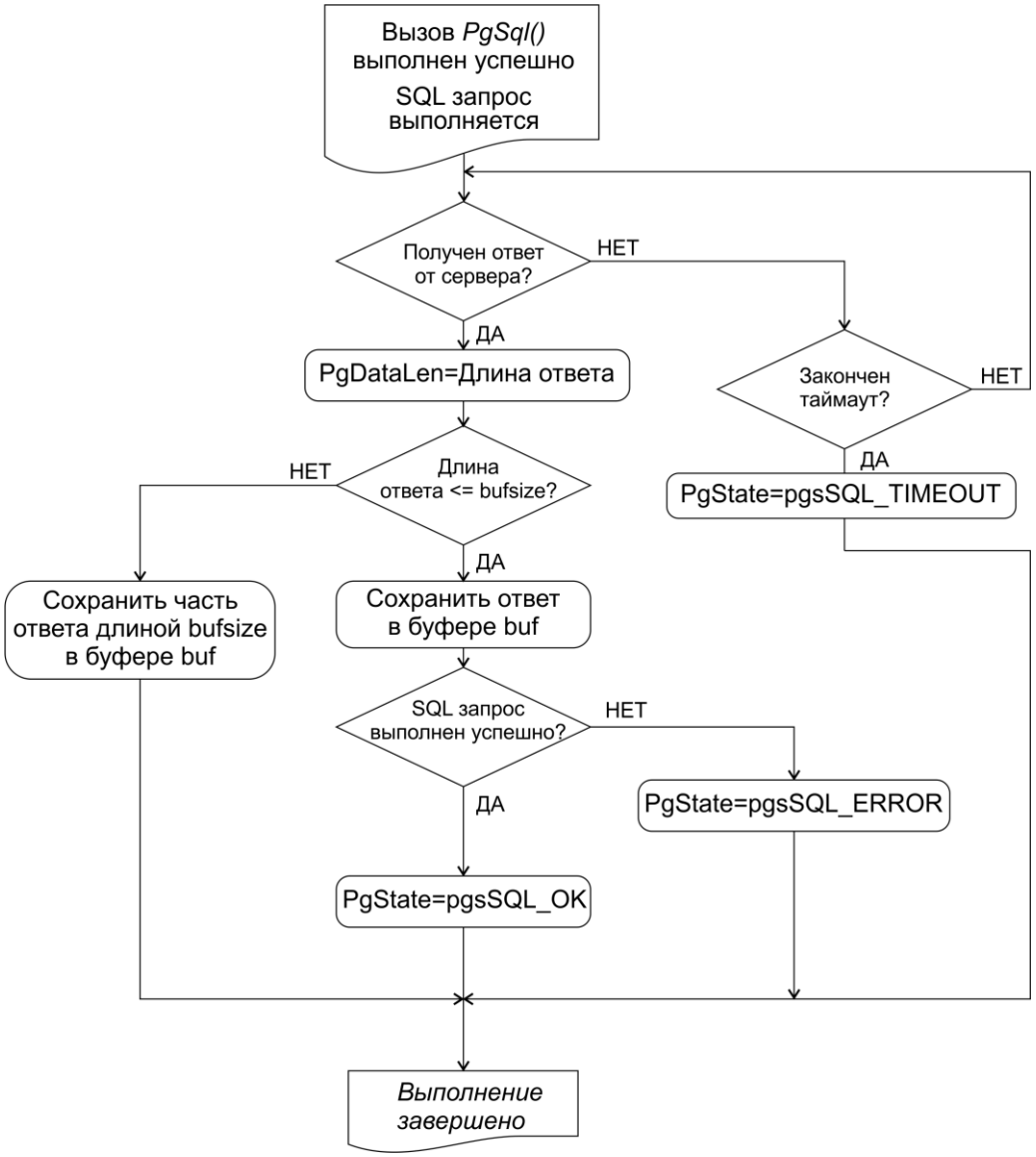


Для получения результата выполнения SQL запроса на сервере следует воспользоваться проверкой переменной *PgState*.
Как только в переменной появится ответ об успешном выполнении запроса (*PgState* = pgsSQL - Сервер успешно выполнил SQL запрос), то полученный ответ сервера (таблица ответа) будет находится в буфере *buf* (см.*PgOpen*) и можно просмотреть ответ, используя переменные *PgColCount* , *PgRowCount* и функции *PgColName*, *PgCell*, *PgCellAsInteger*.

SQL запрос на сервере должен выполниться за время не превышающее значение таймаута, задаваемого переменной *PgTimeout*. Начальное значение таймаута задаётся в функции *PgOpen*.
Также в могут возникнуть следующие ошибки при выполнении SQL запроса:

PgState	PgState (константа)	Описание ошибки
2	pgsDISCONNECTED	Произошло отключение от сервера
7	pgsSQL_BUFSIZ	Недостаточный размер буфера ответа. В буфер ответа помещены только первые bufsize (см.*PgOpen*) байтов ответа. Можно проверить начальную часть ответа используя документацию Postgre SQL, доступную на официальном сайте: https://www.postgresql.org/docs/
8	pgsSQL_ERROR	При выполнении SQL запроса на сервере произошла ошибка Можно получить код ошибки сервера, используя функцию *PgErrorCode* и текст ошибки, используя функцию *PgErrorText*
9	pgsSQL_TIMEOUT	За время таймаута не получен ответ сервера. Возможно серверу требуется больше времени на выполнение этого SQL запроса. Перед вызовом функции PgSql можно задать таймаут выполнения запроса используя переменную *PgTimeout*
10	pgsTCP_ERROR	Ошибка в работе TCP сокета

Ниже приведена блок-схема выполнения SQL запроса:



В приведённом ниже примере выполняется подключение к существующей базе **"lift4"**.

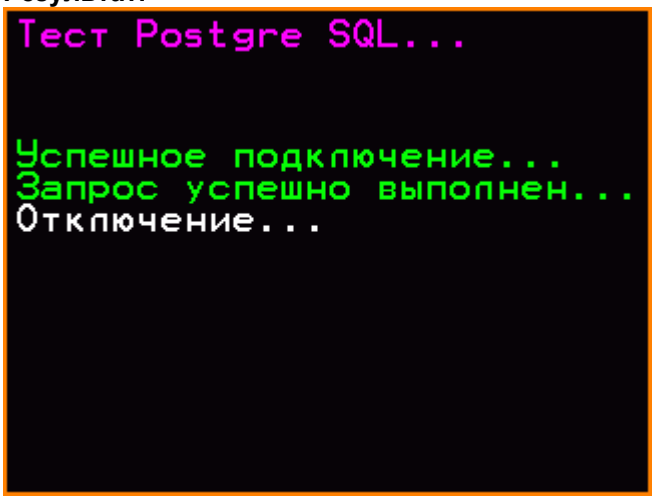
Пример:

```
#include <sat500.h>
#include <pgsql.h>

//Настройки подключения к Postgre SQL
#define PG_IP       "192.168.1.55"
#define PG_PORT     5432
#define PG_TIMEOUT  2000
#define PG_DATABASE "lift4"
#define PG_LOGIN    "postgres"
#define PG_PASSWORD "postgres"
#define PG_WIN1251  true
//Буфер для работы с Postgre SQL
char PgBuf[1024];
//SQL запрос
```

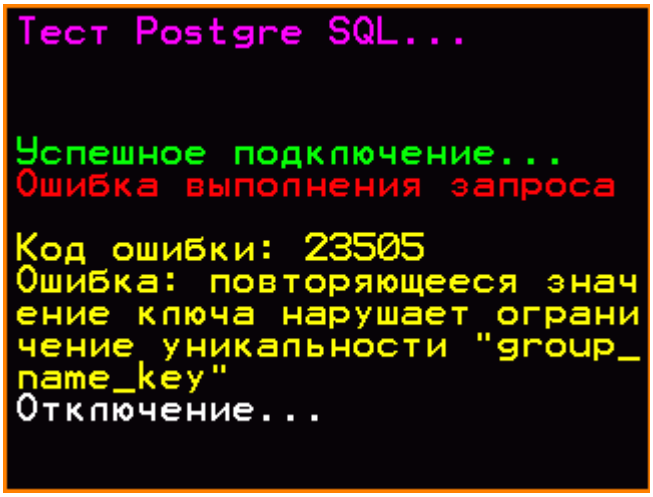
```
#define PG_SQL "INSERT INTO groups(name) VALUES('Группа 3');"

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест Postgre SQL...");
    if(!PgOpen(PG_IP,PG_PORT,PG_TIMEOUT,PG_DATABASE,
    PG_LOGIN,PG_PASSWORD,PG_WIN1251,PgBuf,sizeof(PgBuf)))
    {
        //Не удалось
        puts("\nОшибка PgOpen...");
        goto Exit;
    }
    //Ожидаем подключения
    while(1)
    {
        if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    GotoXY(4,64);
    if(PgState==pgsOK)
    {
        //Подключены к базе данных
        FontColor=COLOR_GREEN;
        puts("Успешное подключение...");
        //Выполняем SQL запрос
        if(!PgSql(PG_SQL))
        {
            FontColor=COLOR_RED;
            puts("Не удалось выполнить запрос...");
            goto Exit;
        }
        //Ожидаем завершения выполнения запроса
        do
        {
            unsigned int t=GetTickCount();
            while((GetTickCount()-t)<100);
        }
        while(PgState==pgsSQL);
        if(PgState==pgsSQL_OK)
        {
            puts("\nЗапрос успешно выполнен...");
        }
        else
        {
            FontColor=COLOR_RED;
            puts("\nОшибка выполнения запроса\n");
            char text[128];
            FontColor=COLOR_YELLOW;
            if(PgErrorCode(text))
            {
                puts("\nКод ошибки: ");
                puts(text);
            }
            if(PgErrorText(text,sizeof(text)))
```

```
            {
                puts("\nОшибка: ");
                puts(text);
            }
        }
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("Ошибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
    while(1);
}
```

Результат:



В случае возникновения ошибки результат будет следующим:



3.17.13. PGSTATE

```
#include <pgsql.h>

unsigned char PgState;
```

Переменная *PgState* возвращает состояние клиента сервера Postgre SQL.
Переменная может принимать следующие значения:

Значение	*Описание*
0	Клиент сервера Postgre SQL не запущен
1	Выполняется подключение к серверу
2	Произошло отключение от сервера
3	Выполняется авторизация на сервере
4	Подключены и авторизованы на сервере
5	Выполняется SQL запрос
6	Сервер успешно выполнил SQL запрос
7	Сервер выполнил запрос, но переданный буфер слишком мал для приёма ответа сервера - ошибка
8	При выполнении SQL запроса на сервере произошла ошибка, которую можно получить функциями *PgErrorCode* и *PgErrorText*
9	За время таймаута не получен ответ сервера
10	Ошибка в работе TCP сокета

Переменная доступна только для чтения.

Для удобства доступа к переменной во включаемом файле *<pgsql.h>* определены следующие константы:

```
enum
{
    pgsNONE=0,          //Клиент сервера Postgre SQL не запущен
    pgsCONNECTING=1,    //Подключение к серверу
    pgsDISCONNECTED=2,  //Отключены от сервера
    pgsAUTH=3,          //Авторизация на сервере
    pgsOK=4,            //Подключены и авторизованы на сервере
    pgsSQL=5,           //Выполняется SQL запрос
    pgsSQL_OK=6,        //Сервер успешно выполнил SQL запрос
    pgsSQL_BUFSIZE=7,   //Недостаточный размер буфера ответа
    pgsSQL_ERROR=8,     //При выполнении SQL запроса на сервере произошла
ошибка
    pgsSQL_TIMEOUT=9,   //За время таймаута не получен ответ сервера
    pgsTCP_ERROR=10     //Ошибка в работе TCP сокета
};
```

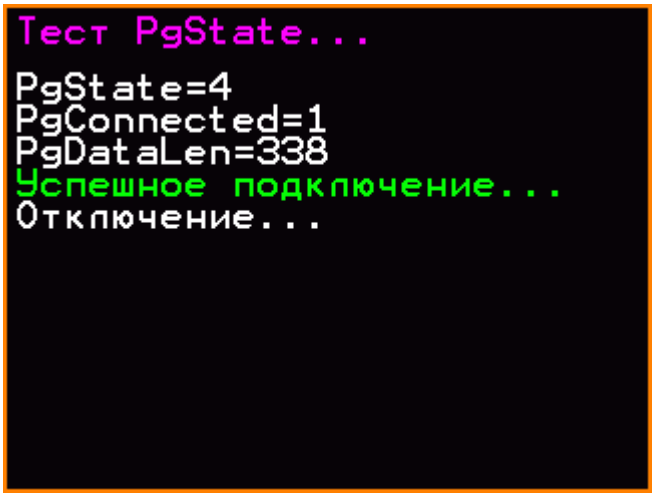
В приведенном далее примере выполняется подключение к серверу Postgre SQL при помощи вызова функции *PgOpen*, и далее на дисплей выводится состояние клиента Pg*State* и состояние подключения *PgConnected*.

Пример:

```
#include <sat500.h>
#include <pgsql.h>
```

```
//Буфер для работы с Postgre SQL
char PgBuf[1024];

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_MAGENTA;
    puts("Тест PgState...");
    if(!PgOpen("192.168.1.55",5432,2000,"lift4","postgres","postgres",
         true,PgBuf,sizeof(PgBuf)))
    {
        //Не удалось
        puts("\nОшибка PgOpen...");
        goto Exit;
    }
    //Ожидаем подключения
    FontColor=COLOR_WHITE;
    while(1)
    {
        GotoXY(4,32);
        printf("PgState=%u\nPgConnected=%u\nPgDataLen=%u",PgState,PgCon-
nected,PgDataLen);
        ShowScr();
        if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<100);
    }
    GotoXY(4,64);
    if(PgConnected)
    {
        FontColor=COLOR_GREEN;
        puts("\nУспешное подключение...");
    }
    else
    {
        //Произошла ошибка при подключении
        FontColor=COLOR_RED;
        puts("\nОшибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
    while(1);
}
```

Результат:



3.17.14. PGTIMEOUT

```
#include <pgsql.h>
```

unsigned int PgTimeout;

Переменная позволяет получить и изменить таймаут выполнения SQL запроса в миллисекундах. Начальное значение таймаута устанавливается в функции *PgOpen* и в дальнейшем может быть изменено.
Таймаут - время в течение которого запрос может выполняться. По достижении указанного значения времени в случае отсутствия ответа сервера Postgre SQL состояние *PgState* будет установлено в значение 9 (**pgsSQL_TIMEOUT - За время таймаута не получен ответ сервера**) и выполнение запроса будет остановлено. (см. Блок-схему выполнения SQL запроса)

В приведённом ниже примере выполняется подключение к существующей базе **"lift4"**.
Далее устанавливается новое значение таймаута 1 мс. Это значение слишком мало для выполнения запроса в контроллере CAT-500. Поэтому SQL запрос выполнить не удаётся.
ВНИМАНИЕ! В имитаторе запрос может выполняться, если 1 мс достаточно для выполнения запроса.

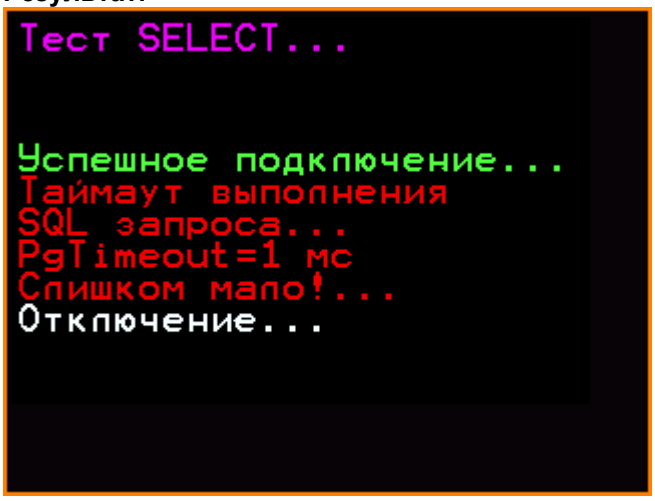
Пример:

```
#include <sat500.h>
#include <pgsql.h>

//Настройки подключения к Postgre SQL
#define PG_IP       "192.168.1.55"
#define PG_PORT     5432
#define PG_TIMEOUT  2000
#define PG_DATABASE "lift4"
#define PG_LOGIN    "postgres"
#define PG_PASSWORD "postgres"
#define PG_WIN1251  true
//Буфер для работы с Postgre SQL
char PgBuf[1024];
//SQL запрос
#define PG_SQL "SELECT * from alarms;"

void main(void)
{
```

```
ClrScr();
BigFont=false;
FontColor=COLOR_MAGENTA;
puts("Тест SELECT...");
if(!PgOpen(PG_IP,PG_PORT,PG_TIMEOUT,PG_DATABASE,
PG_LOGIN,PG_PASSWORD,PG_WIN1251,PgBuf,sizeof(PgBuf)))
{
    //Не удалось
    puts("\nОшибка PgOpen...");
    goto Exit;
}
//Ожидаем подключения
while(1)
{
    if(PgState==pgsOK || PgState==pgsDISCONNECTED)break;
}
GotoXY(4,64);
if(PgState==pgsOK)
{
    //Подключены к базе данных
    FontColor=COLOR_GREEN;
    puts("Успешное подключение...");
    //Установка слишком маленького таймаута 1 мс
    PgTimeout=1;
    //Выполняем SQL запрос
    if(!PgSql(PG_SQL))
    {
        FontColor=COLOR_RED;
        puts("Не удалось выполнить запрос...");
        goto Exit;
    }
    //Ожидаем завершения выполнения запроса
    while(PgState==pgsSQL)
    {
    }
    if(PgState==pgsSQL_OK)
    {
        puts("\nЗапрос успешно выполнен...");
        printf("\nPgDataLen=%d",PgDataLen);
    }
    if(PgState==pgsSQL_BUFSIZE)
    {
        FontColor=COLOR_RED;
        puts("\nНедостаточный размер\nбуфера...");
        printf("\nТребуется %u байтов",PgDataLen);
    }
    if(PgState==pgsSQL_TIMEOUT)
    {
        FontColor=COLOR_RED;
        puts("\nТаймаут выполнения\nSQL запроса...");
        printf("\nPgTimeout=%u мс",PgTimeout);
        puts("\nСлишком мало!...");
    }
}
else
{
    //Произошла ошибка при подключении
    FontColor=COLOR_RED;
```

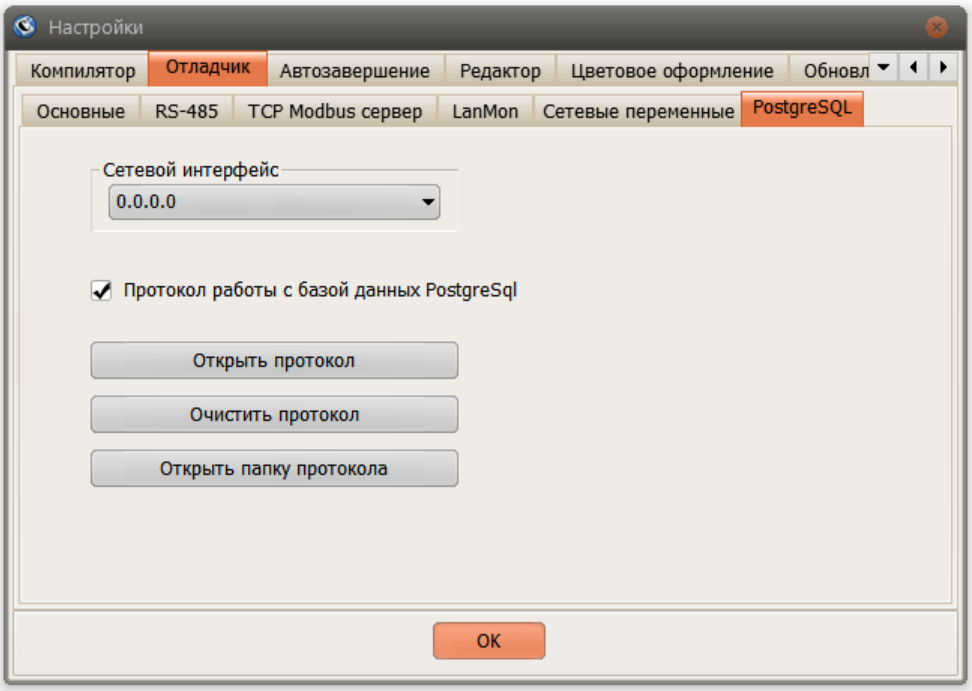
```
        puts("Ошибка подключения...");
    }
Exit:;
    //Отключение
    FontColor=COLOR_WHITE;
    puts("\nОтключение...");
    PgClose();
    ShowScr();
    while(1);
}
```

Результат:



3.17.15. ОТЛАДКА РАБОТЫ С POSTGRE SQL

В случае если в среде разработки С-скрипт выбрана работа с имитатором, то доступен отладочный протокол работы с сервером, который содержит полную информацию об обмене с сервером Post-gre SQL.
Для включения протокола необходимо:
1. Открыть окно *настроек программы*
2. Перейти на подвкладку "Отладчик"
3. Перейти на подвкладку "PostgreSQL":



4. Установить переключатель "Протокол работы с базой данных PostgreSql"
5. Нажать кнопку "ОК"

Для просмотра протокола работы с базой данных PostgreSql необходимо:
1. Открыть окно *настроек программы*
2. Перейти на подвкладку "Отладчик"
3. Перейти на подвкладку "PostgreSQL"
4. Нажать кнопку "Открыть протокол". Откроется окно с текстом протокола:

```
Протокол работы с БД PostgreSQL  [C:\ProgramData\SatScript\Postgresql.log]
12:59:02.0268    Подключение к 192.168.1.55:5432
12:59:02.0366    Авторизация ...
12:59:02.0366 <-
00,00,00,57,00,03,00,00,75,73,65,72,00,70,6F,73,74,67,72,65,73,00,64,61,74,61,62,61,73,6
5,00,6C,69,66,74,34,00,61,70,70,6C,69,63,61,74,69,6F,6E,5F,6E,61,6D,65,00,63,73,63,72,69
,70,74,00,63,6C,69,65,6E,74,5F,65,6E,63,6F,64,69,6E,67,00,57,49,4E,31,32,35,31,00,00
12:59:02.0377    Получено байтов: 13
12:59:02.0378 -> 52,00,00,00,0C,00,00,00,05,3D,A3,A5,A9
12:59:02.0378    TYPE: R Authentication MD5 request
12:59:02.0378          salt:3D A3 A5 A9
12:59:02.0378    PG <<< AUTH Послано: 41
12:59:02.0378 <-
70,00,00,00,28,6D,64,35,38,64,30,64,61,39,30,38,62,32,32,65,36,32,37,39,61,62,62,34,37,6
6,39,38,36,35,39,36,32,36,63,37,00
12:59:02.0380    PG >>> Получено: 338
12:59:02.0380    PG >>> Конец ответа: Z
12:59:02.0380    PG >>> Всего получено: 338
12:59:02.0380 ->
52,00,00,00,08,00,00,00,00,53,00,00,00,1D,61,70,70,6C,69,63,61,74,69,6F,6E,5F,6E,61,6D,6
5,00,63,73,63,72,69,70,74,00,53,00,00,00,1C,63,6C,69,65,6E,74,5F,65,6E,63,6F,64,69,6E,67
,00,57,49,4E,31,32,35,31,00,53,00,00,00,17,44,61,74,65,53,74,79,6C,65,00,49,53,4F,2C,20,
44,4D,59,00,53,00,00,00,19,69,6E,74,65,67,65,72,5F,64,61,74,65,74,69,6D,65,73,00,6F,6E,0
0,53,00,00,00,1B,49,6E,74,65,72,76,61,6C,53,74,79,6C,65,00,70,6F,73,74,67,72,65,73,00,53
,00,00,00,14,69,73,5F,73,75,70,65,72,75,73,65,72,00,6F,6E,00,53,00,00,00,19,73,65,72,76,
65,72,5F,65,6E,63,6F,64,69,6E,67,00,55,54,46,38,00,53,00,00,00,19,73,65,72,76,65,72,5F,7
6,65,72,73,69,6F,6E,00,39,2E,35,2E,30,00,53,00,00,00,23,73,65,73,73,69,6F,6E,5F,61,75,74
,68,6F,72,69,7A,61,74,69,6F,6E,00,70,6F,73,74,67,72,65,73,00,53,00,00,00,23,73,74,61,6E,
64,61,72,64,5F,63,6F,6E,66,6F,72,6D,69,6E,67,5F,73,74,72,69,6E,67,73,00,6F,6E,00,53,00,0
0,00,1B,54,69,6D,65,5A,6F,6E,65,00,45,75,72,6F,70,65,2F,4D,6F,73,63,6F,77,00,4B,00,00,00
,0C,00,00,4F,3C,5B,03,F2,49,5A,00,00,00,05,49
12:59:02.0380    TYPE: R Authentication OK
12:59:02.0380    TYPE: S ParameterStatus: application_name=cscript
12:59:02.0380    TYPE: S ParameterStatus: client_encoding=WIN1251
12:59:02.0380    TYPE: S ParameterStatus: DateStyle=ISO, DMY
12:59:02.0380    TYPE: S ParameterStatus: integer_datetimes=on
12:59:02.0380    TYPE: S ParameterStatus: IntervalStyle=postgres
12:59:02.0381    TYPE: S ParameterStatus: is_superuser=on
12:59:02.0381    TYPE: S ParameterStatus: server_encoding=UTF8
12:59:02.0381    TYPE: S ParameterStatus: server_version=9.5.0
12:59:02.0381    TYPE: S ParameterStatus: session_authorization=postgres
```

Для отладки работы с контроллером CAT-500 используйте любой сетевой сниффер, например https://www.wireshark.org/download.html.

3.18. HTTP СЕРВЕР

Scada системы могут получать данные из контроллера CAT-500 используя протокол HTTP.
Для этого составе контроллера имеется постоянно работающий HTTP сервер.
Для получения данных HTTP клиент (Scada система) должен выполнить TCP подключение к контроллеру CAT-500 и послать HTTP запрос, который в минимальном варианте выглядит следующим образом:
GET /data HTTP/1.1
Передаваемые далее в запросе заголовки и тело запроса могут отсутствовать.
В ответ контроллер CAT-500 посылает сообщение в формате протокола HTTP, состоящее из заголовка и тела сообщения.
Приложение на языке С-скрипт может формировать тело ответа на запрос в виде С-строки, и указывать эту строку серверу контроллера используя функцию *StartHttpData*.
В этой функции также можно указать какие данные держит С-строка и HTTP сервер автоматически перед телом ответного сообщения отправит нужный заголовок.
Также имеется режим, где С-скрипт программа формирует не только тело ответа, но и заголовок, что позволяет передавать данные в произвольном формате в соответствии с протоколом HTTP.
Для использования приведённых далее переменных и функций необходимо включить в исходный текст программы заголовочный файл:

```
#include <http.h>
```

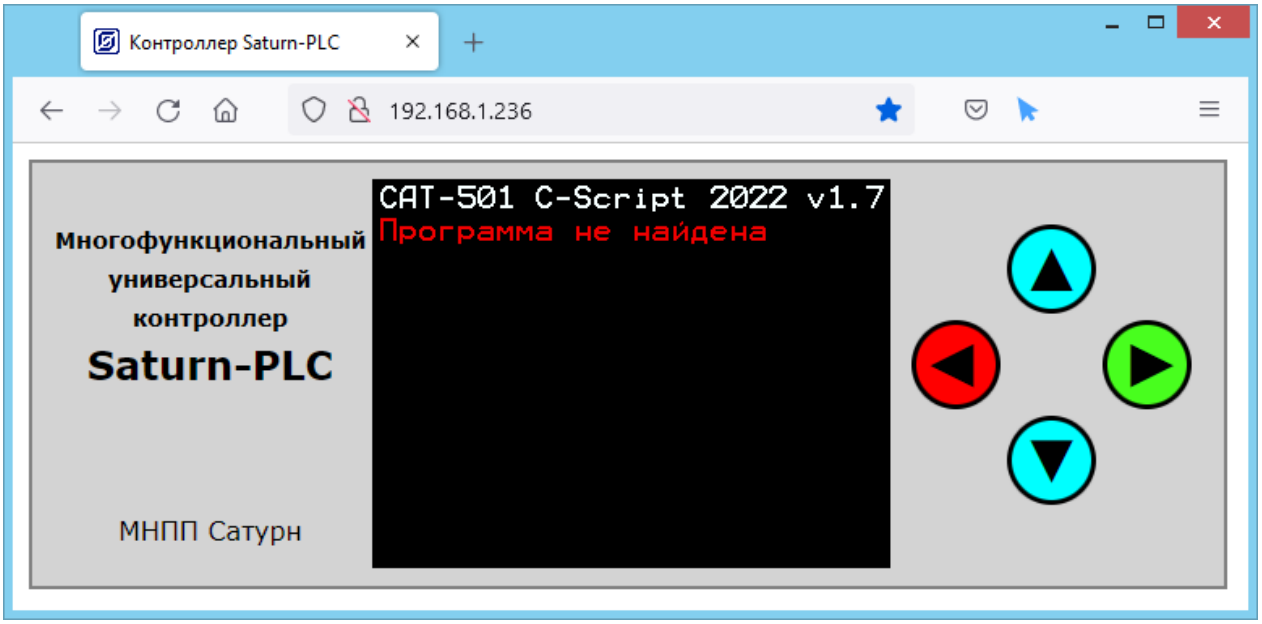
Доступны следующие функции и переменные, связанные с протоколом HTTP:

Переменная	Описание
unsigned int HttpDataCount	Переменная возвращает количество запросов, сделанных к HTTP серверу. Переменная доступна для чтения и для записи

Функция	Описание
*void StartHttpData(char * data,char type)*	Функция задает указатель на С-строку *data*, используемую в качестве тела ответа на HTTP запрос Scada системы. В аргументе *type* функции указывается какие данные содержит С-строка, и, соответственно, какой заголовок будет послан перед телом ответа
void StopHttpData(void)	Функция убирает С-строку, выдаваемую в качестве тела ответа на HTTPзапрос. После вызова этой функции в ответ на запрос будет возвращаться *стандартный JSON текст*, содержащий состояния входов и выходов контроллера CAT-500

Web - интерфейс

При обращении к контроллеру по адресу отличному от **/data** контроллер возвращает изображение, показываемое на дисплее, а также изображение кнопочной панели.
Нажимая на эти виртуальные кнопки можно дистанционно управлять контроллером так, как будто Вы находитесь рядом и нажимаете кнопки. На виртуальном дисплее отображается картинка с дисплея контроллера. Изображение автоматически обновляется раз в две секунды.



3.18.1. ПРОТОКОЛ HTTP

HTTP (HyperText Transfer Protocol — протокол передачи гипертекста) — символьно-ориентированный клиент-серверный протокол прикладного уровня без сохранения состояния, используемый сервисом World Wide Web.
Протокол HTTP позволяет указать способ представления (кодирования) одного и того же ресурса по различным параметрам: mime-типу, языку и т. д. Благодаря этой возможности клиент и веб-сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

Структура протокола

Структура протокола определяет, что каждое HTTP-сообщение состоит из трёх блоков, которые передаются в следующем порядке:
1. Стартовая строка (англ. Starting line) — определяет тип сообщения
2. Заголовки (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения
3. Тело сообщения (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Стартовая строка HTTP

Стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа, заголовки и тело сообщения могут отсутствовать.
Стартовые строки различаются для запроса и ответа. Строка запроса выглядит так:
Метод URI HTTP/Версия протокола
Пример запроса:
GET /data HTTP/1.1
Стартовая строка ответа сервера имеет следующий формат:
HTTP/Версия КодСостояния [Пояснение]
Например, на предыдущий запрос клиентом данной страницы сервер ответил строкой:
HTTP/1.1 200 Ok

Заголовки HTTP

Заголовок HTTP (HTTP Header) — это строка в HTTP-сообщении, содержащая разделённую двоеточием пару вида «параметр-значение». Формат заголовка соответствует общему формату заголовков текстовых сетевых сообщений ARPA (RFC 822). Как правило, браузер и веб-сервер включают в сообщения более чем по одному заголовку. Заголовки должны отправляться раньше тела сообщения и отделяться от него хотя бы одной пустой строкой (CRLF).
Название параметра должно состоять минимум из одного печатного символа (ASCII-коды от 33 до 126). После названия сразу должен следовать символ двоеточия. Значение может содержать любые символы ASCII, кроме перевода строки (CR, код 10) и возврата каретки (LF, код 13).
Пробельные символы в начале и конце значения обрезаются. Последовательность нескольких пробельных символов внутри значения может восприниматься как один пробел. Регистр символов в названии и значении не имеет значения (если иное не предусмотрено форматом поля).
Контроллер CAT-500 игнорирует заголовки и тело HTTP запроса.

Тело сообщения

Тело HTTP сообщения (message-body), если оно присутствует, используется для передачи сущности, связанной с запросом или ответом. Тело сообщения (message-body) отличается от тела сущности (entity-body) только в том случае, когда при передаче применяется кодирование, указанное в заголовке Transfer-Encoding. В остальных случаях тело сообщения идентично телу сущности.
Заголовок Transfer-Encoding должен отправляться для указания любого кодирования передачи, примененного приложением в целях гарантирования безопасной и правильной передачи сообщения. Transfer-Encoding - это свойство сообщения, а не сущности, и оно может быть добавлено или удалено любым приложением в цепочке запросов/ответов.
Присутствие тела сообщения в запросе отмечается добавлением к заголовкам запроса поля заголовка Content-Length или Transfer-Encoding. Тело сообщения (message-body) может быть добавлено в запрос только когда метод запроса допускает тело объекта (entity-body).
Все ответы содержат тело сообщения, возможно нулевой длины, кроме ответов на запрос методом HEAD и ответов с кодами статуса 1xx (Информационные), 204 (Нет содержимого, No Content), и 304 (Не модифицирован, Not Modified).

3.18.2. ДАННЫЕ ОТВЕТА ПО УМОЛЧАНИЮ

HTTP сервер контроллера CAT-500 начинает работу сразу после включения, и на запрос "*GET /data HTTP/1.1"* выдает в ответ в формате HTTP, где в теле сообщения передаётся JSON текст, содержащий информацию о состоянии входов и выходов контроллера.

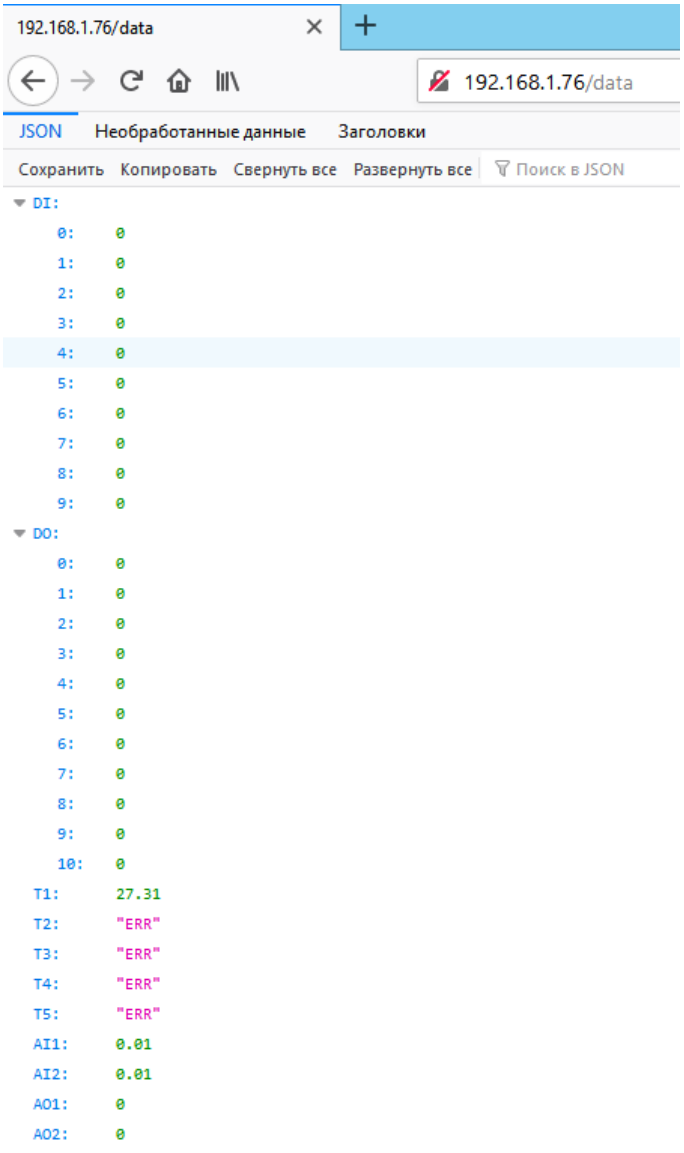
Пример выдаваемого JSON текста:

{"DI":[0,0,0,0,0,0,0,0,0,0],"DO":[0,0,0,0,0,1,0,0,0,1,0],"T1":20.20,"T2":21.63,"T3":20.00,"T4":20.00, "T5":20.00,"AI1":0.00,"AI2":0.00,"AO1":0.23,"AO2":0.00}

Здесь:
DI - массив состояния входов контроллера (первым в массиве сигнал DI1)
DO - массив состояние выходов контроллера (первым в массиве сигнал DO1)
T1...T5 - показания 5 температурных датчиков (температура в градусах цельсия)
AI1 - состояние аналогового входа AI1 (напряжение в вольтах или ток в миллиамперах)
AI2 - состояние аналогового входа AI2 (напряжение в вольтах или ток в миллиамперах)
AO1 - состояние аналогового выхода AO1 (напряжение в вольтах)
AO2 - состояние аналогового выхода AO2 (напряжение в вольтах)

После вызова функции *StartHttpData* данные ответа на HTTP запрос заменяются указанной C-строкой. В зависимости от указанного в аргументе типа посылается необходимый заголовок.
После вызова функции *StopHttpData* в ответ на HTTP запрос опять посылаются данные по умолчанию - JSON текст.

На рисунке ниже показан запрос, сделанный браузером, и ответ контроллера CAT-500:



3.18.3. ИСПОЛЬЗОВАНИЕ БРАУЗЕРА ДЛЯ ПРОВЕРКИ

Вместо Scada системы можно послать HTTP запрос любым браузером и увидеть ответ контроллера.
Например, если IP адрес контроллера CAT-500 - 192.168.1.76, то для выполнения запроса необходимо в адресной строке браузера набрать следующий запрос:

192.168.1.76/data

или

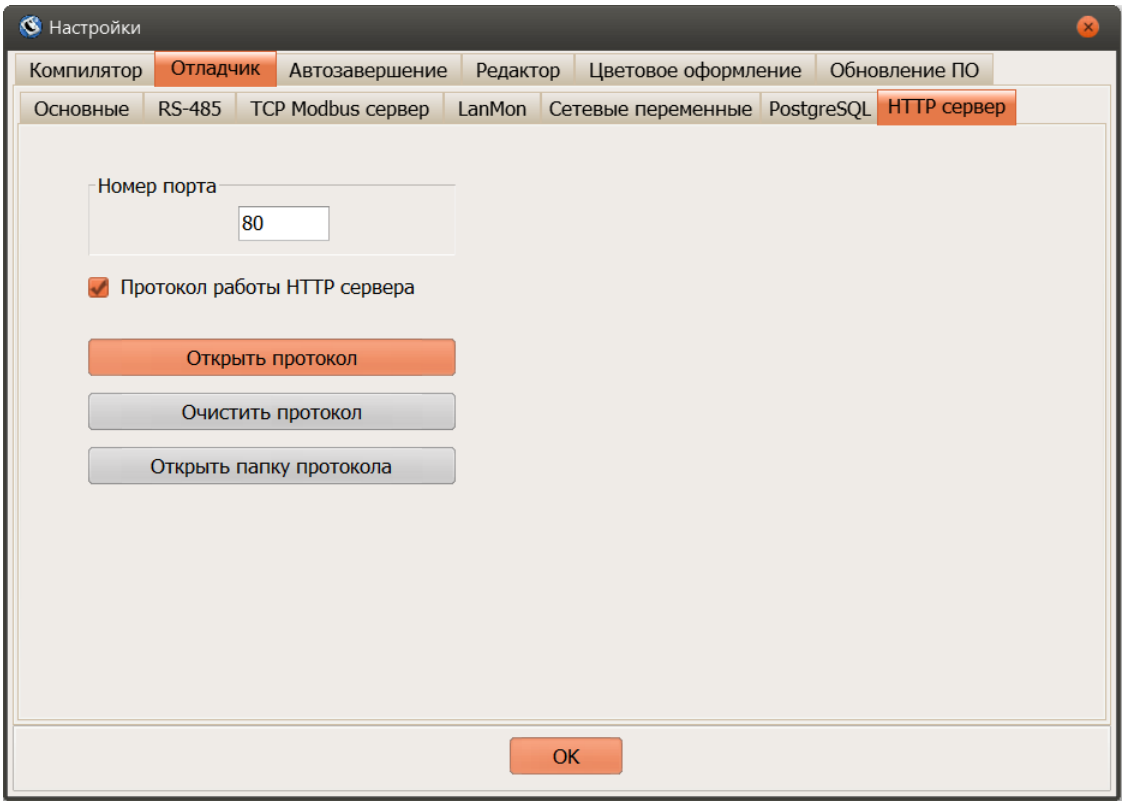
http://192.168.1.76/data

Внимание! CAT-500 не поддерживает работу протокола https, поэтому запрос "https://192.168.1.76/data" является ошибочным - ответ не будет получен.

В случае, когда С-скрипт программа отлаживается в имитаторе, то в браузере можно использовать IP адрес компьютера, или адрес 127.0.0.1, или имя "localhost":

http://127.0.0.1/data

Также в настройках программы На вкладке "Отладчик/HTTP сервер" можно задать номер порта на котором будет работать HTTP сервер имитатора:



Если стандартный порт 80 на компьютере занят другим веб сервером, то можно использовать любой другой свободный порт. В этом случае в запросе браузера необходимо указывать номер порта следующим образом (для порта 1080):

http://127.0.0.1:1080/data

3.18.4. HTTPDATACOUNT

```
#include <http.h>

unsigned int HttpDataCount;
```

Переменная *HttpDataCount* возвращает количество принятых HTTP запросов, на которые был выдан ответ.
Позволяет отследить сетевую активность Scada систем.

В примере ниже в качестве тела HTTP ответа используется С-строка *Json*.
Заполнение Json выполняется один раз в секунду в функции-обработчике события таймера *OnTimer1*
Для заполнения используется буферная строка *Tmp*.
После заполнения строки *Tmp* ответ копируется в массив *Json* при помощи функции *strcpy*.

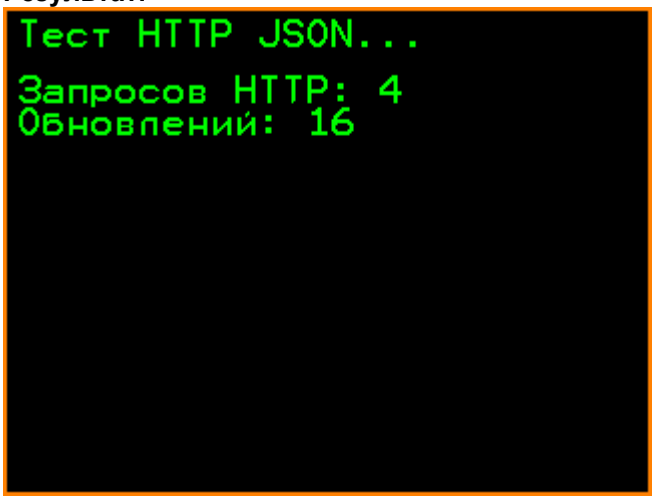
В Json строке передаётся строка IP адреса и время.

Пример:

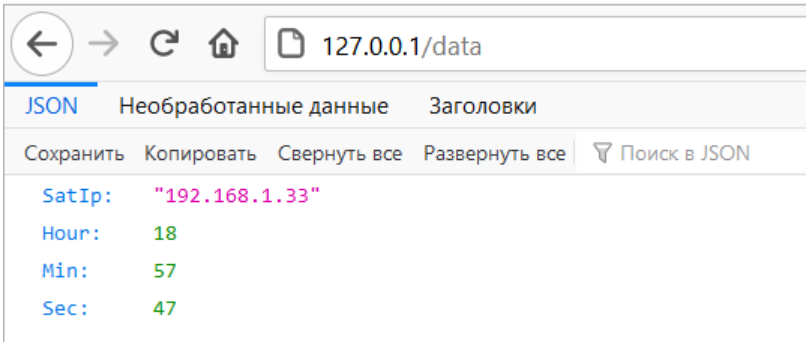
```
#include <sat500.h>
#include <string.h>
#include <http.h>
//-----------------------------------------------------------------
typedef char bool;
//-----------------------------------------------------------------
//Тело HTTP ответа
char Json[1024];
char Tmp[1024];
unsigned int Counter;
//-----------------------------------------------------------------
//Функция - обработчик таймера
void interrupt OnTimer1(void);
//Добавить в Tmp целочисленный параметр JSON
void AddJsonParam(char * name,int value,bool comma);
//Добавить в Tmp строковый параметр JSON
void AddJsonStrParam(char * name,char * value,bool comma);
//-----------------------------------------------------------------

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_GREEN;
    puts("Тест HTTP JSON...");
    //Вызывать функцию OnTimer1 один раз в секунду
    //Каждую секунду заполняется Json
    StartTimer(&OnTimer1,1000);
    //Json - массив тела HTTP ответа
    StartHttpData(Json,httpdJSON);
    //Обновление экрана
    ShowScr();
    while(1)
    {
        //Пауза 1 секунда
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<1000){};
        //Вывод на дисплей количества HTTP запросов
        GotoXY(4,32);
        printf("Запросов HTTP: %u",HttpDataCount);
        printf("\nОбновлений: %u",Counter);
        //Обновление экрана
        ShowScr();
    }
}
//-----------------------------------------------------------------
//Функция - обработчик таймера
void interrupt OnTimer1(void)
{
    Counter++;
    //Заполняем массив Tmp JSON сообщением
    strcpy(Tmp,"{");
    //Добавляем текстовую переменную - серийный номер CAT-500
    union
```

```
    {
        unsigned int ip;
        unsigned char bip[4];
    };
    ip=SatIp;
    char s[32];
    sprintf(s,"%d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    AddJsonStrParam("SatIp",s,false);
    //Получаем текущее время
    unsigned char hour,min,sec;
    GetTime(&hour,&min,&sec);
    //Добавляем время в JSON
    AddJsonParam("Hour",hour,true);
    AddJsonParam("Min",min,true);
    AddJsonParam("Sec",sec,true);
    //Завершение json
    strcat(Tmp,"}");
    //Копирование буферной строки Tmp в Json
    strcpy(Json,Tmp);
}
//-----------------------------------------------------------------
//Добавить в Tmp целочисленный параметр JSON
void AddJsonParam(char * name,int value,bool comma)
{
    if(comma)strcat(Tmp,",");
    sprintf(Tmp+strlen(Tmp),"\"%s\":%d",name,value);
}
//-----------------------------------------------------------------
//Добавить в Tmp строковый параметр JSON
void AddJsonStrParam(char * name,char * value,bool comma)
{
    if(comma)strcat(Tmp,",");
    sprintf(Tmp+strlen(Tmp),"\"%s\":\"%s\"",name,value);
}
//-----------------------------------------------------------------
```

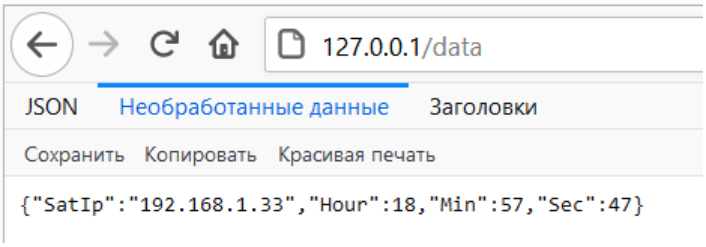
Результат:



Отображение запрашиваемого JSON в браузере:



Вкладка "Необработанные данные" в браузере:



Запрос, посланный браузером:
GET /data HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/*;q=0.8*
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

Ответ контроллера CAT-500:
HTTP/1.0 200 OK
Connection: close
Content-Length: 52
Content-type: application/json

{"SatIp":"192.168.1.33","Hour":18,"Min":57,"Sec":46}

3.18.5. STARTHTTPDATA

```
#include <http.h>

void StartHttpData(char * data,char type);
```

Функция задает С-строку, используемую в качестве тела ответа на HTTP запрос Scada системы.
*char * data* - указатель на С-строку, являющуюся телом HTTP ответа
char type - тип HTTP ответа - указывает какие данные содержат содержит С-строка и соответственно какой заголовок будет послан перед телом ответа (С-строкой)

При помощи аргумента *type* можно подать следующие виды HTTP ответа:

type	HTTP ответ
0	С-строка содержит произвольные данные, включающие в себя заголовок ответа и тело ответа
1	С-строка содержит JSON текст, используемый в качестве тела ответа. HTTP сервер автоматический посылает необходимой заголовок ответа
2	С-строка содержит текст в формате HTML, используемый в качестве тела ответа. HTTP сервер автоматический посылает необходимой заголовок ответа
3	С-строка содержит обычный текст, используемый в качестве тела ответа. HTTP сервер автоматический посылает необходимой заголовок ответа

Для удобства заголовочный файл <*http.h*> содержит следующие константы, описывающие типы:

```
enum
{
    httpdUSER=0,
    httpdJSON=1,
    httpdHTML=2,
    httpdTEXT=3
};
```

Далее приведены подробные описания с примерами для каждого из типов:
JSON
HTML
TEXT
Пользовательский

3.18.6. STOPHTTPDATA

```
#include <http.h>

void StopHttpData(void);
```

Функция убирает С-строку, выдаваемую в качестве тела ответа на HTTPзапрос. После вызова этой функции в ответ на запрос будет возвращаться стандартный JSON текст, содержащий состояния входов и выходов контроллера CAT-500

3.18.7. JSON

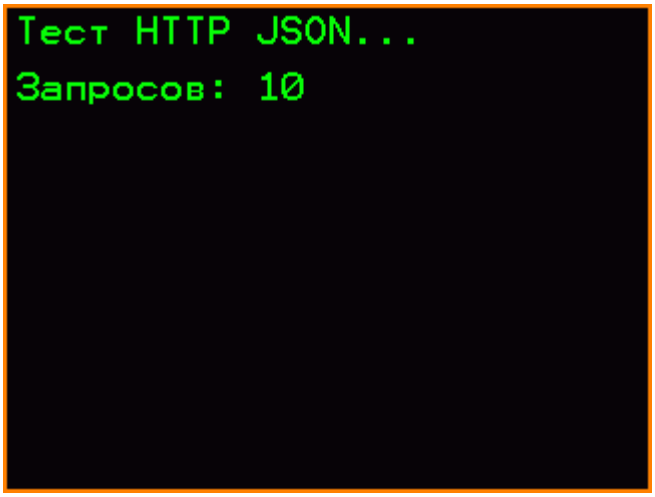
Для того чтобы в ответ на HTTP запрос сервер CAT-500 выдавал нужные JSON данные в вызове функции *StartHttpData* следует указать С-строку содержащую тело ответа - корректный JSON текст. В аргументе *type* необходимо указать значение 1 (*httpdJSON*) и по мере необходимости изменять С-строку при изменении посылаемых JSON данных. Особенностью работы с этой строкой является то, что строка должна заполняться за один оператор языка С-скрипт. Если операторов несколько, то следует заполнить буферную строку и затем выполнить копирование из буферной строки в С-строку тела HTTP ответа.

В примере ниже в качестве тела HTTP ответа используется С-строка *Json*. Для заполнения используется буферная строка *Tmp*.
После заполнения строки *Tmp* ответ копируется в массив *Json* при помощи функции *strcpy*.

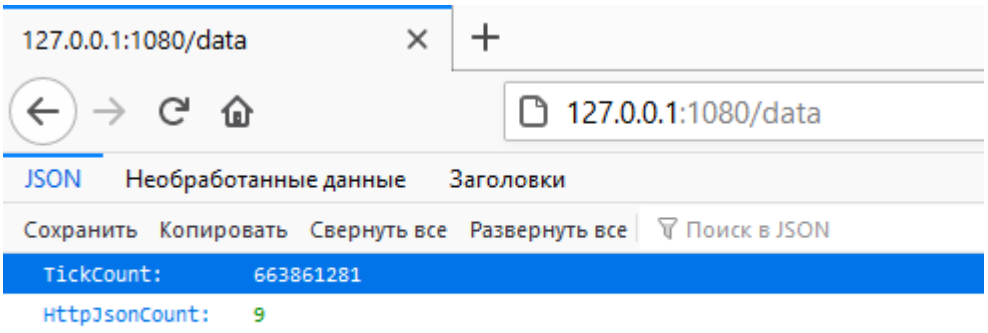
Пример:

```
#include <sat500.h>
#include <string.h>
#include <http.h>

char Json[1024];
char Tmp[1024];

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_GREEN;
    puts("Тест HTTP JSON...");
    StartHttpData(Json,httpdJSON);
    ShowScr();
    while(1)
    {
        //Формирование JSON текста во временном буфере Tmp
        //Начало JSON
        strcpy(Tmp,"{");
        //Первый параметр
        sprintf(Tmp+strlen(Tmp),"\"TickCount\":%u",GetTickCount());
        //Второй параметр
        sprintf(Tmp+strlen(Tmp),",\"HttpJsonCount\":%u",HttpDataCount);
        //Завершение JSON
        strcat(Tmp,"}");
        //Копирование буферной строки в Json
        strcpy(Json,Tmp);
        //В данном случае можно было обойтись без временного буфера:

//sprintf(Json,"{\"{TickCount\":%u,\"HttpJsonCount\":%u}",GetTickCount(),
        //HttpDataCount);
        //Пауза 1 секунда
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<1000){};
        //Вывод на дисплей количества HTTP запросов
        GotoXY(4,32);
        printf("Запросов: %u",HttpDataCount);
        //Обновление экрана
        ShowScr();
    }
}
```

Результат:



Отображение запрашиваемого JSON в браузере:



Запрос, посланный браузером:
GET /data HTTP/1.1
Host: 127.0.0.1:1080
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/*;q=0.8*
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

Ответ контроллера CAT-500:
08:35:16.0830 HTTP/1.0 200 OK
Connection: close
Content-Length: 41
Content-type: application/json

{"TickCount":663861281,"HttpJsonCount":9}

3.18.8. HTML

Для того чтобы в ответ на HTTP запрос сервер CAT-500 выдавал нужную HTML страницу в вызове функции *StartHttpData* следует указать С-строку содержащую тело ответа - корректную HTML страницу. В аргументе *type* необходимо указать значение 2 (*httpdHTML*) и по мере необходимости изменять С-строку при изменении посылаемой HTML страницы. Особенностью работы с этой строкой является то, что строка должна заполняться за один оператор языка С-скрипт. Если операторов несколько, то следует заполнить буферную строку и затем выполнить копирование из буферной строки в С-строку тела HTTP ответа.

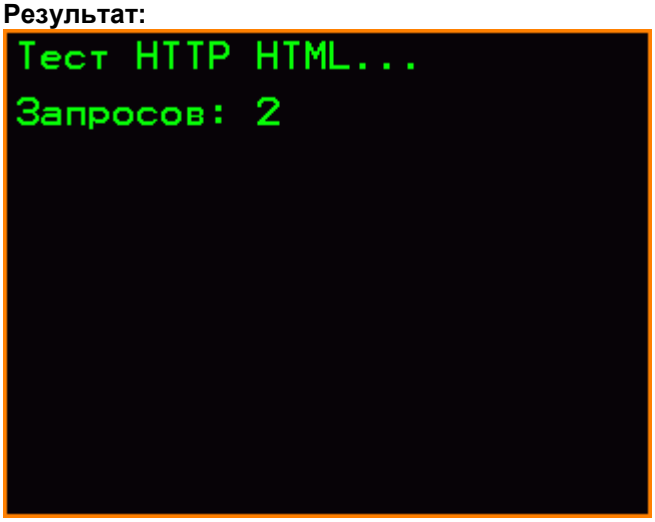
В примере ниже в качестве тела HTTP ответа используется С-строка *Html*. Для заполнения используется буферная строка *Tmp*.
Шаблон HTML страницы находится в const строке *HtmlTemplate*.

Пример:

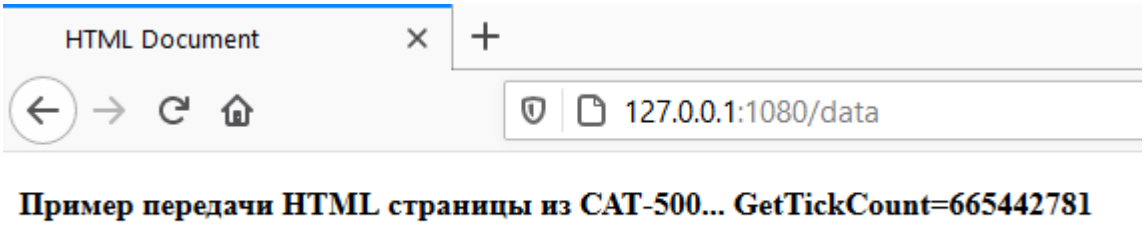
```
#include <sat500.h>
#include <string.h>
#include <http.h>

char Html[512];
char Tmp[512];

const char HtmlTemplate[]=
"<!DOCTYPE html>\
<html>\
<head>\
<meta charset=\"windows-1251\" />\
<title>HTML Document</title>\
</head>\
<body>\
<p>\
<b>\
%s\
</b>\
</p>\
</body>\
</html>";

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_GREEN;
    puts("Тест HTTP HTML...");
    Html[0]=0;
    StartHttpData(Html,httpdHTML);
    ShowScr();
    while(1)
    {
        sprintf(Tmp,"Пример передачи HTML страницы из CAT-500... Get-
TickCount=%u",
        GetTickCount());
        sprintf(Html,HtmlTemplate,Tmp);
        unsigned int t=GetTickCount();
        //Пауза 1 секунда
        while((GetTickCount()-t)<1000){};
        //Вывод на дисплей количества HTTP запросов
        GotoXY(4,32);
        printf("Запросов: %u",HttpDataCount);
```

```
        ShowScr();
    }
}
```

Результат:



Отображение запрашиваемого HTML в браузере:



Запрос, посланный браузером:
GET /data HTTP/1.1
Host: 127.0.0.1:1080
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/*;q=0.8*
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

Ответ контроллера CAT-500:
HTTP/1.0 200 OK
Connection: close
Content-Length: 193
Content-type: text/html

<!DOCTYPE html><html><head><meta charset="windows-1251" /><title>HTML Document</title></head><body> Пример передачи HTML страницы из CAT-500... GetTickCount=665442781 </body></html>

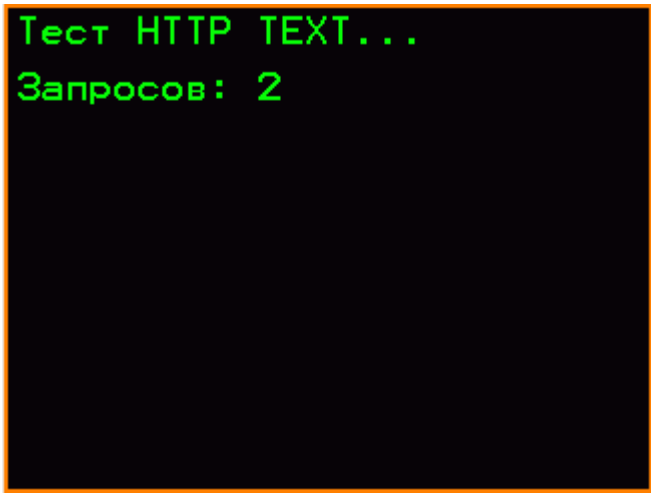
3.18.9. TEXT

Для того чтобы в ответ на HTTP запрос сервер CAT-500 выдавал нужный текст в вызове функции *StartHttpData* следует указать С-строку содержащую тело ответа - корректный JSON текст. В аргументе *type* необходимо указать значение 3 (*httpdTEXT*) и по мере необходимости изменять С-строку при изменении посылаемых текстовых данных. Особенностью работы с этой строкой является то, что строка должна заполняться за один оператор языка С-скрипт. Если операторов несколько, то следует заполнить буферную строку и затем выполнить копирование из буферной строки в С-строку тела HTTP ответа.

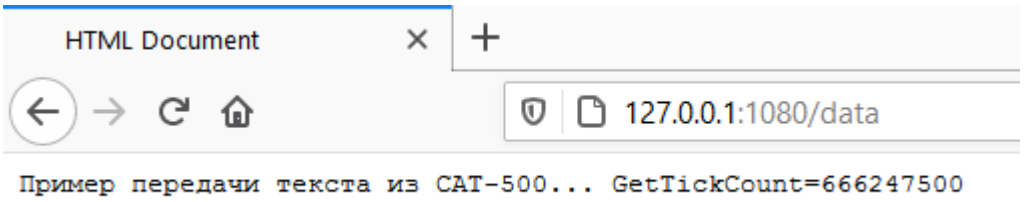
В примере ниже в качестве тела HTTP ответа используется С-строка *Text*. Для заполнения используется буферная строка *Tmp*.
После заполнения строки *Tmp* ответ копируется в массив *Text* при помощи функции *strcpy*.
Пример:

```
#include <sat500.h>
#include <string.h>
#include <http.h>

char Text[512];
char Tmp[512];

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_GREEN;
    puts("Тест HTTP TEXT...");
    Text[0]=0;
    StartHttpData(Text,httpdTEXT);
    ShowScr();
    while(1)
    {
        //Формирование текста во временном буфере Tmp
        strcpy(Tmp,"Пример передачи текста из CAT-500... ");
        sprintf(Tmp+strlen(Tmp),"GetTickCount=%u",GetTickCount());
        //Копирование буферной строки в Text
        strcpy(Text,Tmp);
        //Пауза 1 секунда
        unsigned int t=GetTickCount();
        while((GetTickCount()-t)<1000){};
        //Вывод на дисплей количества HTTP запросов
        GotoXY(4,32);
        printf("Запросов: %u",HttpDataCount);
        //Обновление экрана
        ShowScr();
    }
}
```

Результат:



Отображение запрашиваемого текста в браузере:



Запрос, посланный браузером:
GET /data HTTP/1.1
Host: 127.0.0.1:1080
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/*;q=0.8*
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

Ответ контроллера CAT-500:

```
HTTP/1.0 200 OK
Connection: close
Content-Length: 59
Content-type: text/plain;charset=cp1251


Пример передачи текста из CAT-500... GetTickCount=666247500
```

3.18.10. ПОЛЬЗОВАТЕЛЬСКИЙ

Если в вызове функции *StartHttpData* в аргументе *type* указано значение 0 (*httpdUSER*), то С-строка, задаваемая в аргументе *data* содержит не только тело, но и заголовок ответа, т.е. содержит текст полного ответа на HTTP запрос. Таким образом возможна передача любых данных клиенту.
По мере необходимости следует изменять С-строку при изменении посылаемых данных. Особенностью работы с этой строкой является то, что строка должна заполняться за один оператор языка

С-скрипт. Если операторов несколько, то следует заполнить буферную строку и затем выполнить копирование из буферной строки в С-строку тела HTTP ответа.

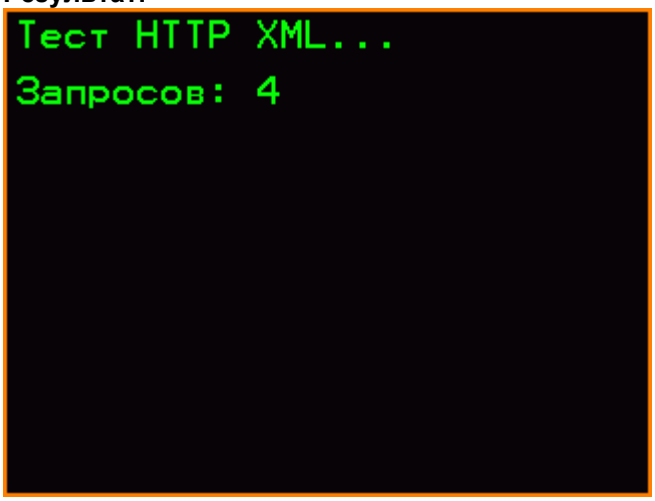
В примере далее выполняется передача XML данных, используя пользовательский режим.
В качестве HTTP ответа на запрос используется С-строка *Xml*. Для заполнения используются временные буферные строки *Tmp* и *XmlBody*.

Пример:

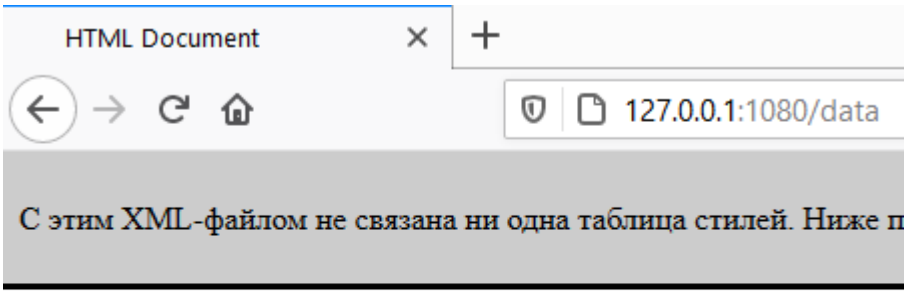
```
#include <sat500.h>
#include <string.h>
#include <http.h>

//Шаблон заголовка отыета на запрос
const char Header_XML[]="HTTP/1.0 200 OK\nConnection: close\nContent-Length:
%d\nContent-type: text/xml;charset=windows-1251\n\n";

char Xml[512];
char XmlBody[512];
char Tmp[512];

void main(void)
{
    ClrScr();
    BigFont=false;
    FontColor=COLOR_GREEN;
    puts("Тест HTTP XML...");
    Xml[0]=0;
    StartHttpData(Xml,httpdUSER);
    ShowScr();
    while(1)
    {
        //Формируем XML
        //Т.к. это несколько операций, то выполняем заполнение через промежу-
точный буфер Tmp
        //Заполняем тело ответа (после заголовка)
        strcpy(XmlBody,"<?xml version=\"1.0\" encoding=\"windows-
1251\"?><data><title>Пример передачи xml из CAT-500</title>");
        int len=strlen(XmlBody);
        unsigned int t=GetTickCount();
        sprintf(XmlBody+len,"<GetTickCount>%u</GetTickCount></data>",t);
        //Длина тела xml
        len=strlen(XmlBody);
        //Заполняем заголовок HTTP ответа
        sprintf(Tmp,Header_XML,len);
        len=strlen(Tmp);
        //Добавляем тело ответа
        strcat(Tmp+len,XmlBody);
        //Копирование полной строки ответа
        strcpy(Xml,Tmp);
        //Пауза 1 секунда
        t=GetTickCount();
        while((GetTickCount()-t)<1000);
        //Вывод количество сделанных запросов от клиентов
        GotoXY(4,32);
        printf("Запросов: %u",HttpDataCount);
        ShowScr();
    }
```

}

Результат:



Отображение запрашиваемого XML в браузере:




```
−<data>
    <title>Пример передачи xml из CAT-500</title>
    <GetTickCount>667525906</GetTickCount>
  </data>
```

Запрос, посланный браузером:
GET /data HTTP/1.1
Host: 127.0.0.1:1080
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/*;q=0.8*
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

Ответ контроллера CAT-500:
HTTP/1.0 200 OK
Connection: close
Content-Length: 139
Content-type: text/xml;charset=windows-1251

<?xml version="1.0" encoding="windows-1251"?><data><title>Пример передачи xml из CAT-500</title><GetTickCount>2854093</GetTickCount></data>

3.19. ДОПОЛНИТЕЛЬНЫЕ ФУНКЦИИ

Перед использованием функций необходимо включить заголовочный файл.

```
#include <sat500.h>
```

Доступны следующие функции:

Функция	***Описание***
*int GetModbusFloat (unsigned short int * reg , unsigned int mul)*	Получение целого значения из числа в формате float. Эта функция позволяет преобразовать значение в формате float IEEE 754 в целочисленное значение, где одна единица целого числа соответствует 1/mul исходного float числа. Число float умножается на множитель mul и возвращается целая часть результата *unsigned short int * reg* - указатель на два слова в формате MODBUS (в каждом слове по младшему адресу находится старший байт), где хранится четырёхбайтовое float значение в формате IEEE 754. *unsigned int * mul* - Множитель, определяющий точность преобразования float числа в целое.
*int GetFloat (unsigned char * ptr,unsigned int mul)*	Получение целого значения из числа в формате float. Эта функция позволяет преобразовать значение в формате float IEEE 754 в целочисленное значение, где одна единица целого числа соответствует 1/*mul* исходного float числа. Число в формате float занимает 4 байта, на начало которых указывает *ptr*. Число float умножается на множитель *mul* и возвращается целая часть результата *unsigned chart * ptr* - указатель на четыре байта, которые следует рассматривать как плавающее число одинарной точности в формате IEEE 754. По младшему адресу расположен младший байт числа float. *unsigned int * mul* - Множитель, определяющий точность преобразования float числа в целое.
*int GetDouble (unsigned char * ptr,unsigned int mul)*	Получение целого значения из числа в формате double. Эта функция позволяет преобразовать значение в формате double IEEE 754 в целочисленное значение, где одна единица целого числа соответствует 1/*mul* исходного double числа. Число в формате double занимает 8 байт, на начало которых указывает *ptr*. Число double умножается на множитель *mul* и возвращается целая часть результата. *unsigned chart * ptr* - указатель на восемь байт, которые следует рассматривать как плавающее число двойной точности в формате IEEE 754. По младшему адресу расположен младший байт числа float. *unsigned int * mul* - Множитель, определяющий точность преобразования float числа в целое.
*void CreateFloat (unsigned char * ptr,int value,int div)*	Получение числа в формате float IEEE 754 из целого значения и делителя.

	Формат float не используется в языке C-Script, но для передачи данных в формате float в сторонние системы можно использовать данную функцию. Эта функция позволяет преобразовать целое число в значение в формате float IEEE 754. Входное значение *value* преобразуется в формат float IEEE 754, затем делится на число *div* и затем 4 байта результата помещаются в массив *ptr*. *unsigned chart * ptr* - указатель на четыре байта, которые следует рассматривать как плавающее число одинарной точности в формате IEEE 754. По младшему адресу расположен младший байт числа float. *int * value* - Значение, которое будет использовано для создания float числа. *unsigned int * div* - Делитель для значения *value* при преобразовании целого во float число.
*void CreateDouble (unsigned char * ptr,int value,int div)*	Получение числа в формате double IEEE 754 из целого значения и делителя Формат double не используется в языке C-Script, но для передачи данных в формате double в сторонние системы можно использовать данную функцию. Эта функция позволяет преобразовать целое число в значение в формате double IEEE 754. Входное значение *value* преобразуется в формат double IEEE 754, затем делится на число *div* и затем 8 байтов результата помещаются в массив *ptr*. *unsigned chart * ptr* - указатель на восемь байтов, которые следует рассматривать как плавающее число двойной точности в формате IEEE 754. По младшему адресу расположен младший байт числа double. *int * value* - Значение, которое будет использовано для создания double числа. *unsigned int * div* - Делитель для значения *value* при преобразовании целого в double число.
int abs (int v)	Получение абсолютного значения числа

3.19.1. GETMODBUSFLOAT

```
#include <sat500.h>

int GetModbusFloat(unsigned short int * reg,unsigned int mul);
```

Получение целого значения из числа в формате float.
Эта функция позволяет преобразовать значение в формате float IEEE 754 в целочисленное значение, где одна единица целого числа соответствует 1/*mul* исходного float числа.
Число в формате float занимает 4 байта (два регистра MODBUS).
Число float умножается на множитель *mul* и возвращается целая часть результата

*unsigned short int * reg* - указатель на два слова в формате MODBUS (в каждом слове по младшему адресу находится старший байт), где хранится четырёхбайтовое float значение в формате IEEE 754.
*unsigned int * mul* - Множитель, определяющий точность преобразования float числа в целое.

Примечание:
Т.к. функция предназначена для получения значения из регистров MODBUS, то в некоторых случаях потребуется "развернуть" слова *reg* - поменять местами старший и младший байт.

3.19.2. GETFLOAT

```
#include <sat500.h>

int GetFloat(unsigned char * ptr,unsigned int mul);
```

Получение целого значения из числа в формате float.
Эта функция позволяет преобразовать значение в формате float IEEE 754 в целочисленное значение, где одна единица целого числа соответствует 1/*mul* исходного float числа.
Число в формате float занимает 4 байта, на начало которых указывает *ptr*.
Число float умножается на множитель *mul* и возвращается целая часть результата

*unsigned chart * ptr* - указатель на четыре байта, которые следует рассматривать как плавающее число одинарной точности в формате IEEE 754. По младшему адресу расположен младший байт числа float.
*unsigned int * mul* - Множитель, определяющий точность преобразования float числа в целое.

В примере ниже float число 1234.567, представленное в виде байтового массива val преобразуется в целое число в тысячных долях.

Пример:

```
#include <sat500.h>

void main(void)
{
    unsigned char val[4];
    ClrScr();
    puts("Тест GetFloat");
    GotoXY(0,40);
    val[0]=0x25;
    val[1]=0x52;
    val[2]=0x9A;
    val[3]=0x44;
    int value=GetFloat(val,1000);
    printf("value=%d",value);
    ShowScr();
    while(1);
}
```

Результат:

3.19.3. GETDOUBLE

```
#include <sat500.h>

int GetDouble(unsigned char * ptr,unsigned int mul);
```

Получение целого значения из числа в формате double.
Эта функция позволяет преобразовать значение в формате double IEEE 754 в целочисленное значение, где одна единица целого числа соответствует 1/*mul* исходного double числа.
Число в формате double занимает 8 байт, на начало которых указывает *ptr*.
Число double умножается на множитель *mul* и возвращается целая часть результата.

*unsigned chart * ptr* - указатель на восемь байт, которые следует рассматривать как плавающее число двойной точности в формате IEEE 754. По младшему адресу расположен младший байт числа float.
*unsigned int * mul* - Множитель, определяющий точность преобразования float числа в целое.

В примере ниже double число 123456.789, представленное в виде байтового массива val преобразуется в целое число в тысячных долях.

Пример:

```
#include <sat500.h>
void main(void)
{
    unsigned char val[8];
    ClrScr();
    puts("Тест GetDouble");
    GotoXY(0,40);
    val[0]=0xC9;
    val[1]=0x76;
    val[2]=0xBE;
    val[3]=0x9F;
    val[4]=0x0C;
    val[5]=0x24;
    val[6]=0xFE;
    val[7]=0x40;
    int value=GetDouble(val,1000);
    printf("value=%d",value);
    ShowScr();
    while(1);
}
```

Результат:

3.19.4. CREATEFLOAT

```
#include <sat500.h>

void CreateFloat(unsigned char * ptr,int value,int div);
```

Получение числа в формате float IEEE 754 из целого значения и делителя.
Формат float не используется в языке C-Script, но для передачи данных в формате float в сторонние системы можно использовать данную функцию.
Эта функция позволяет преобразовать целое число в значение в формате float IEEE 754.
Входное значение *value* преобразуется в формат float IEEE 754, затем делится на число *div* и затем 4 байта результата помещаются в массив *ptr*.

*unsigned chart * ptr* - указатель на четыре байта, которые следует рассматривать как плавающее число одинарной точности в формате IEEE 754. По младшему адресу расположен младший байт числа float.
*int * value* - Значение, которое будет использовано для создания float числа.
*unsigned int * div* - Делитель для значения *value* при преобразовании целого во float число.

В примере ниже целое число 1234567 преобразуется в плавающее число одинарной точности 1234.567, представленное в виде байтового массива.

Пример:

```
#include <sat500.h>

void main(void)
{
    unsigned char s[4];
    ClrScr();
    puts("Тест CreateFloat");
    CreateFloat(s,1234567,1000);
    GotoXY(0,40);
    for(int i=0;i<4;i++)
    {
        printf("s[%d]=%02X\n",i,s[i]);
    }
    ShowScr();
    while(1);
}
```

Результат:

3.19.5. CREATEDOUBLE

```
#include <sat500.h>

void CreateDouble(unsigned char * ptr,int value,int div);
```

Получение числа в формате double IEEE 754 из целого значения и делителя
Формат double не используется в языке C-Script, но для передачи данных в формате double в сторонние системы можно использовать данную функцию.
Эта функция позволяет преобразовать целое число в значение в формате double IEEE 754.
Входное значение *value* преобразуется в формат double IEEE 754, затем делится на число *div* и затем 8 байтов результата помещаются в массив *ptr*.

*unsigned chart * ptr* - указатель на восемь байтов, которые следует рассматривать как плавающее число двойной точности в формате IEEE 754. По младшему адресу расположен младший байт числа double.
*int * value* - Значение, которое будет использовано для создания double числа.
*unsigned int * div* - Делитель для значения *value* при преобразовании целого в double число.

В примере ниже целое число 123456789 преобразуется в плавающее число двойной точности 123456.789, представленное в виде байтового массива.

Пример:

```
#include <sat500.h>

void main(void)
{
    unsigned char s[8];
    ClrScr();
    BigFont=0;
    puts("Тест CreateDouble");
    CreateDouble(s,123456789,1000);
    GotoXY(0,40);
    for(int i=0;i<8;i++)
```

```
    {
        printf("s[%d]=%02X\n",i,s[i]);
    }
    ShowScr();
    while(1);
}
```

Результат:

3.19.6. ABS

```
#include <sat500.h>

int abs(int v);
```

Получение абсолютного значения числа.

3.20. ETHERNET

Перед использованием переменных работы с интерфейсом ETHERNET необходимо включить заголовочный файл.

```
#include <sat500.h>
```

Доступны следующие переменные:

Переменная	***Описание***
char EthernetRestart	Переменная выполняет разрешение/запрещение перезапуска интерфейса ETHERNET в случае отсутствия подключения к сети ETHERNET (линка)
char EthernetLink	Переменная *EthernetLink* позволяет определить текущее состояние интерфейса ETHERNET

3.20.1. ETHERNETRESTART

```
#include <sat500.h>

char EthernetRestart
```

Переменная выполняет разрешение/запрещение перезапуска интерфейса ETHERNET в случае отсутствия подключения к сети ETHERNET (линка).

EthernetRestart	**Описание**
0	Перезапуск интерфейса ETHERNET запрещён (по умолчанию)
1	Перезапуск интерфейса ETHERNET разрешён

В случае наличия разрешения (*EthernetRestart=1*) и отсутствия линка периодически будет выполняться инициализация интерфейса ETHERNET, которая выполняется несколько секунд, в течение которых все функции контроллера CAT-500 приостанавливаются.

Если программе не требуется работа с IP сетью, то рекомендуется установить переменную в значение 0.
По умолчанию при старте программы переменная равна 0 - перезапуск интерфейса ETHERNET запрещён.

ПРИМЕЧАНИЕ: Значение переменной в имитаторе контроллера CAT-500 не имеет значения.

3.20.2. ETHERNETLINK

```
#include <sat500.h>

char EthernetLink
```

Переменная *EthernetLink* позволяет определить текущее состояние интерфейса ETHERNET:

EthernetLink	**Описание**
0	Нет подключения к сети ETHERNET (нет линка)
1	Есть подключение 10 Mbit полудуплекс
2	Есть подключение 10 Mbit дуплекс
3	Есть подключение 100 Mbit полудуплекс
4	Есть подключение 100 Mbit дуплекс
-1	Ошибка (неисправна PHY микросхема)

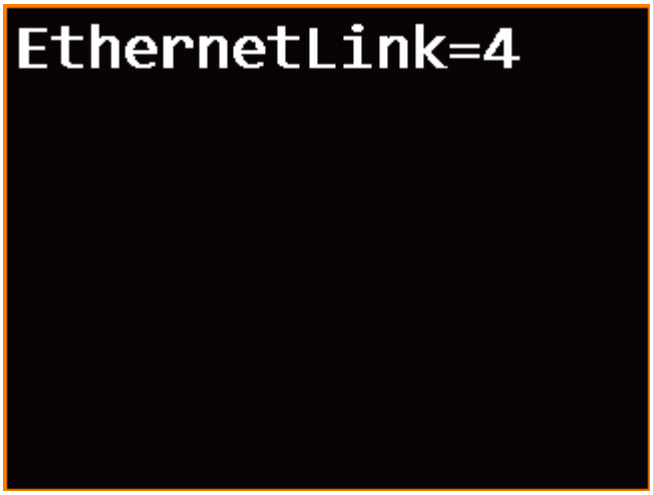
Переменная *EthernetLink* доступна только для чтения.

ПРИМЕЧАНИЕ: В имитаторе переменная всегда возвращает значение 4.

Пример:

```
#include <sat500.h>
void main(void)
{
    while(1)
    {
        ClrScr();
        printf("EthernetLink=%d",EthernetLink);
        ShowScr();
    }
}
```

Результат:



3.21. ПАРАМЕТРЫ CAT-500

В этом разделе описаны настроечные параметры контроллера CAT-500, доступные для чтения в С-Скрипт.

Переменная	***Описание***
unsigned int SatSerNumber	Серийный номер контроллера CAT-500

3.21.1. SATSERNUMBER

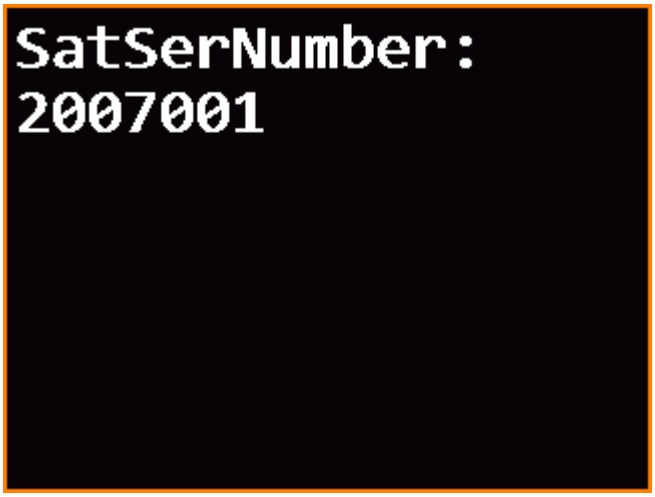
```
#include <sat500.h>

unsigned int SatSerNumber;
```

Переменная позволяет получить серийный номер контроллера CAT-500.
Переменная доступна только для чтения.

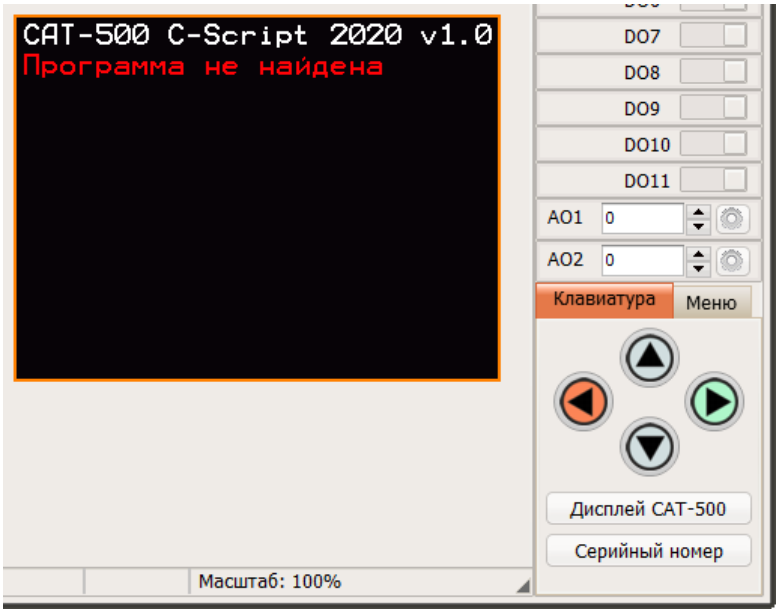
Пример:

```
#include <sat500.h>

void main(void)
{
    ClrScr();
    printf("SatSerNumber:\n%u",SatSerNumber);
    ShowScr();
    while(1);
}
```

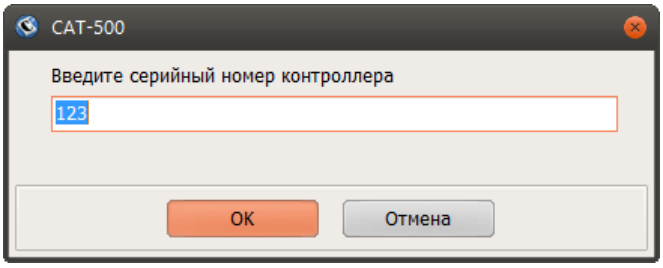
Результат:



Значение переменной в имитаторе контроллера CAT-500 задаётся после нажатия на кнопку "**Серийный номер**", расположенную в правой нижней части окна имитатора:



После нажатия на кнопку можно ввести новый серийный номер контроллера:



Для просмотра серийного номера на контроллере CAT-500 необходимо войти в меню контроллера, нажав на кнопку "ВПРАВО":

Далее необходимо кнопками "ВВЕРХ" или "ВНИЗ" выбрать пункт меню "5. Параметры CAT-500" и нажать кнопку "ВПРАВО":

В открывшемся меню следует выбрать пункт "Об устройстве". На открывшемся экране имеется серийный номер контроллера в строке "Сер. номер:"

Расшифровка значения серийного номера соответствует таблице:

Год выпуска партии (2 цифры)	Месяц выпуска партии (2 цифры)	Порядковый номер в партии (3 цифры)
18	01	001

3.21.2. SATIP

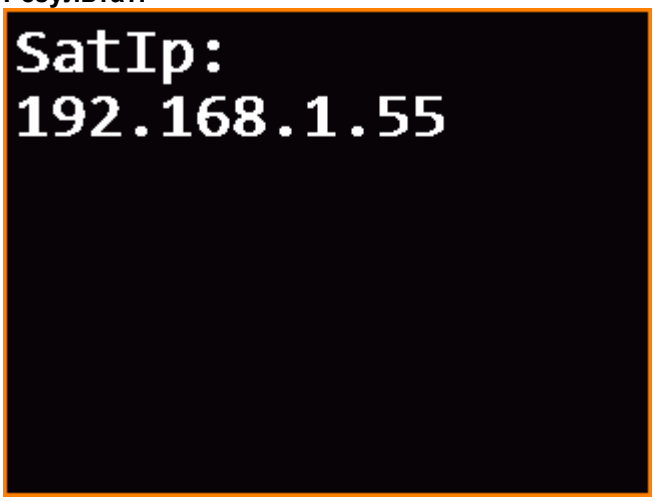
```
#include <sat500.h>

unsigned int SatIp;
```

Переменная позволяет получить текущий IP адрес контроллера CAT-500.
Переменная доступна только для чтения.

Пример:

```
#include <sat500.h>

void main(void)
{
    union
    {
        unsigned int ip;
        unsigned char bip[4];
    };
    ClrScr();
    ip=SatIp;
    printf("SatIp:\n%d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ShowScr();
    while(1);
}
```

Результат:



3.21.3. SATMASK

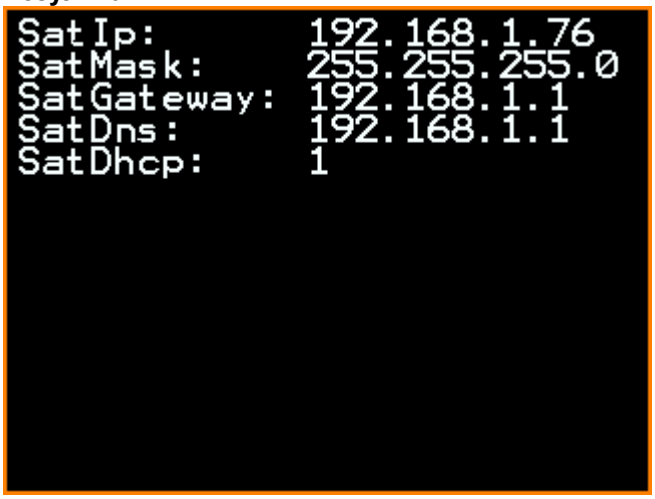
```
#include <sat500.h>

unsigned int SatMask;
```

Переменная позволяет получить текущую маску подсети контроллера CAT-500.
Переменная доступна только для чтения.

Пример:

```
#include <sat500.h>

void main(void)
{
    union
    {
        unsigned int ip;
        unsigned char bip[4];
    };
    ClrScr();
    BigFont=false;
    ip=SatIp;
    printf("SatIp:      %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatMask;
    printf("\nSatMask:    %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatGateway;
    printf("\nSatGateway: %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatDns;
    printf("\nSatDns:     %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    printf("\nSatDhcp:    %d",SatDhcp);
    ShowScr();
    while(1);
}
```

Результат:



3.21.4. SATGATEWAY

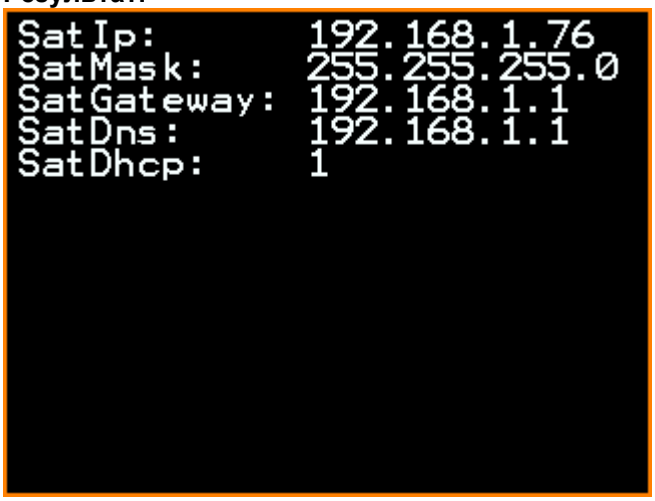
```
#include <sat500.h>

unsigned int SatGateway;
```

Переменная позволяет получить текущий адрес шлюза контроллера CAT-500.
Переменная доступна только для чтения.

Пример:

```
#include <sat500.h>

void main(void)
{
    union
    {
        unsigned int ip;
        unsigned char bip[4];
    };
    ClrScr();
    BigFont=false;
    ip=SatIp;
    printf("SatIp:      %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatMask;
    printf("\nSatMask:    %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatGateway;
    printf("\nSatGateway: %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatDns;
    printf("\nSatDns:     %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    printf("\nSatDhcp:    %d",SatDhcp);
    ShowScr();
    while(1);
}
```

Результат:



3.21.5. SATDNS

```
#include <sat500.h>

unsigned int SatDns;
```

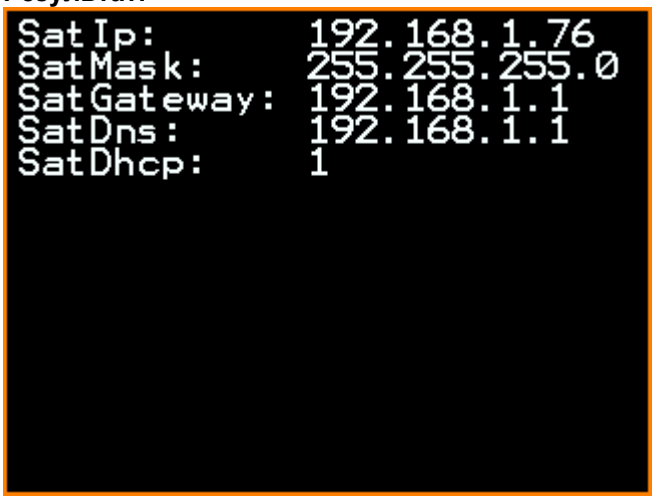
Переменная позволяет получить текущий адрес DNS сервера, используемого контроллером CAT-500.
Переменная доступна только для чтения.

Примечание. Переменную *SatDns* можно использовать для определения момента получения предложения DHCP сервера когда контроллер в режиме "Получать IP адрес автоматически". Когда переменная принимает ненулевое значение - IP адрес получен от DHCP сервера.

Пример:

```
#include <sat500.h>

void main(void)
{
    union
    {
        unsigned int ip;
        unsigned char bip[4];
    };
    ClrScr();
    BigFont=false;
    ip=SatIp;
    printf("SatIp:      %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatMask;
    printf("\nSatMask:    %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatGateway;
    printf("\nSatGateway: %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatDns;
    printf("\nSatDns:     %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    printf("\nSatDhcp:    %d",SatDhcp);
    ShowScr();
    while(1);
}
```

Результат:



3.21.6. SATDHCP

```
#include <sat500.h>

unsigned int SatDhcp;
```

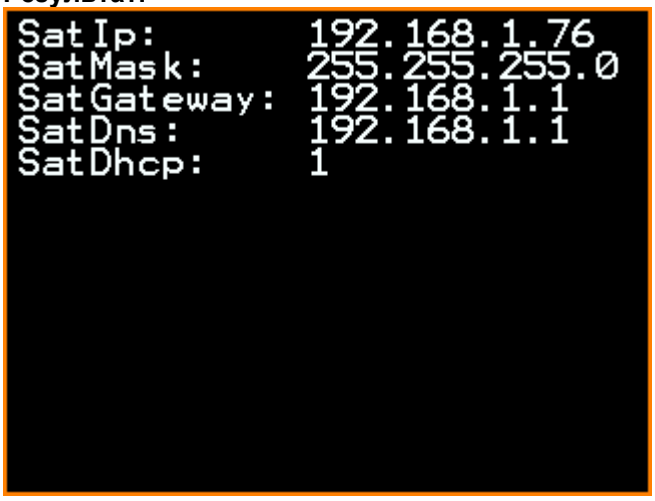
Переменная позволяет определить использует ли контроллера CAT-500 автоматическое получение IP адреса.
Переменная доступна только для чтения.

SatDhcp	Описание
0	Используется статический IP адрес
1	IP адрес получается автоматически от DHCP сервера

Пример:

```
#include <sat500.h>

void main(void)
{
    union
    {
        unsigned int ip;
        unsigned char bip[4];
    };
    ClrScr();
    BigFont=false;
    ip=SatIp;
    printf("SatIp:      %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatMask;
    printf("\nSatMask:    %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatGateway;
    printf("\nSatGateway: %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    ip=SatDns;
    printf("\nSatDns:     %d.%d.%d.%d",bip[0],bip[1],bip[2],bip[3]);
    printf("\nSatDhcp:    %d",SatDhcp);
    ShowScr();
    while(1);
}
```

Результат:



3.22. ОШИБКИ

В этом разделе описаны ошибки контроллера САТ-500, доступные для чтения в С-Скрипт.

Переменная	*Описание*

unsigned short int *ADCErrors*	Ошибки при получении значений датчиков температуры и каналов АЦП

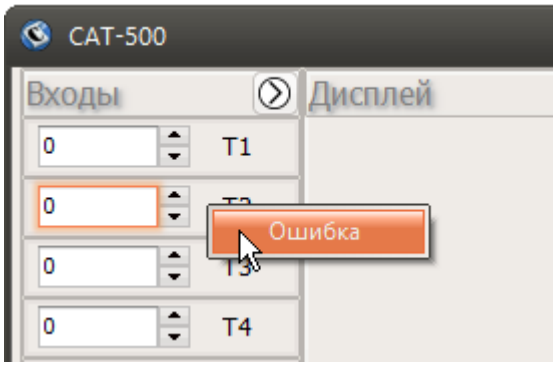
3.22.1. ADCERRORS

```
#include <sat500.h>

unsigned short int ADCErrors
```

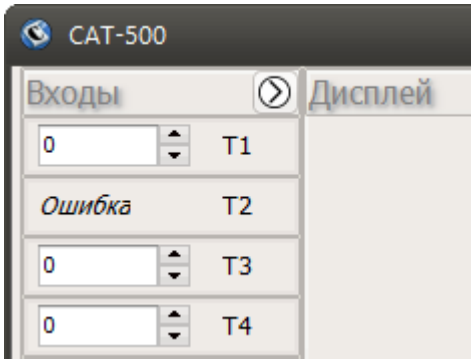
Переменная позволяет определить имеются ли ошибки при получении значений датчиков температуры и каналов АЦП.
Переменная доступна только для чтения.
Значения ошибок передаются побитово в соответствии со следующей таблицей:

Номер бита	**Описание**
0	Датчик температуры T[0]: 0 - нет ошибки (значение в норме), 1 - ошибка (значение недостоверно)
1	Датчик температуры T[1]: 0 - нет ошибки (значение в норме), 1 - ошибка (значение недостоверно)
2	Датчик температуры T[2]: 0 - нет ошибки (значение в норме), 1 - ошибка (значение недостоверно)
3	Датчик температуры T[3]: 0 - нет ошибки (значение в норме), 1 - ошибка (значение недостоверно)
4	Датчик температуры T[4]: 0 - нет ошибки (значение в норме), 1 - ошибка (значение недостоверно)
5	Аналоговый вход AI1: 0 - нет ошибки (значение в норме), 1 - ошибка (значение недостоверно)
6	Аналоговый вход AI2: 0 - нет ошибки (значение в норме), 1 - ошибка (значение недостоверно)

Значение переменной в имитаторе контроллера CAT-500 задаётся через контекстное меню, доступное для каждого датчика температуры и аналогового входа. На рисунке ниже вызвано контекстное меню температурного датчика T2:



В случае выбора пункта меню "Ошибка" датчик или аналоговый вход переходят в состояние "Ошибка":



3.23. ОБНОВЛЕНИЕ ПРОГРАММЫ ЧЕРЕЗ ИНТЕРНЕТ

Контроллер CAT-500 может автоматически загрузить новую версию скрипт-программы с HTTP сервера (web сервера).
Для включения данной функции следует выполнить следующие действия:
1. В исходном тексте программе на языке C-Script необходимо указать адрес сервера обновления, используя следующую директиву pragma:

```
#pragma inf("$up") "web-сервер/путь к каталогу с файлами обновления/"
```

,здесь
web-сервер - адрес web сервера, где в подкаталоге лежат файлы обновления
путь к каталогу с файлами обновления - подкаталог web-сервера с файлами обновления

Пример задания папки с файлами обновления программы:

```
#pragma inf("$up") "www.mnppsaturn.ru/update/demo/"
```

2. Включить в исходный текст программы директиву с номером версии разрабатываемой программы

```
#pragma version verh.verl
```

verh - число, задающее старший номер версии (0-65535)
verl - число, задающее младший номер версии (0-65535)

Пример задания номера версии программы:

```
#pragma version 1.0
```

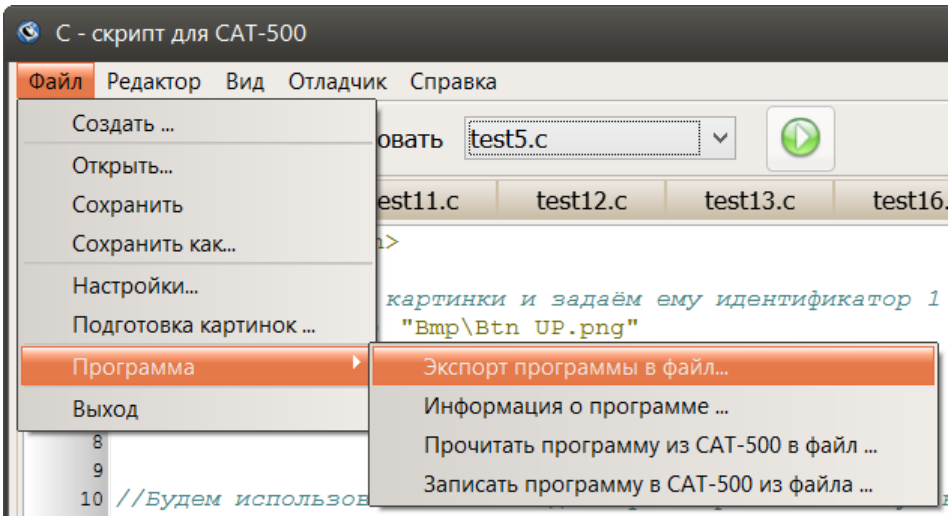
3. Откомпилировать программу и записать её в контроллер CAT-500

4. Внести в программу необходимые изменения, увеличить номер версии и откомпилировать программу

5. Создать 2 файла обновления программы:
- Файл программы с именем совпадающим с именем исходный кода. Если компилируется исходный код с именем **"test5.c"**, то следует создать файл программы **"test5.prog"**
- Файл версии программы. Имя файла должно совпадать с именем исходный кода. Если компилируется исходный код с именем **"test5.c"**, то следует создать файл версии с именем **"test5.ver"**
Файл версии содержит следующую строку: "Version: 1.2"

1.2 - номер версии, заданный в директиве `#pragma version`

Для создания обоих файлов после компиляции следует использовать пункт основного меню **"Файл/Программа/Экспорт программы в файл"**



После выбора пункта меню необходимо задать локальный каталог для файлов.
Будут созданы оба необходимых для обновления файла.

6. Поместить оба созданных файла (программу и файл версии) на указанный web-сервер в указанную папку.

7. Для выполнения обновления программы в контроллере САТ-500 следует выбрать пункт меню: "Программа/6.Обновить программу"

Обновление возможно только в том случае когда контроллер САТ-500 имеет доступ по сети к указанному серверу обновления.

ВНИМАНИЕ! Обновление можно разместить на любом web-сервере. Сервер www.mnppsaturn.ru указанный выше приведён только для примера.

Для проверки работы системы обновления программы можно загрузить следующую программу в контроллер САТ-500 и выполнить обновление программы.
На web-сервере *www.mnppsaturn.ru/update/demo/* размещены: файл программы "test5.prog" и файл "test5.ver"

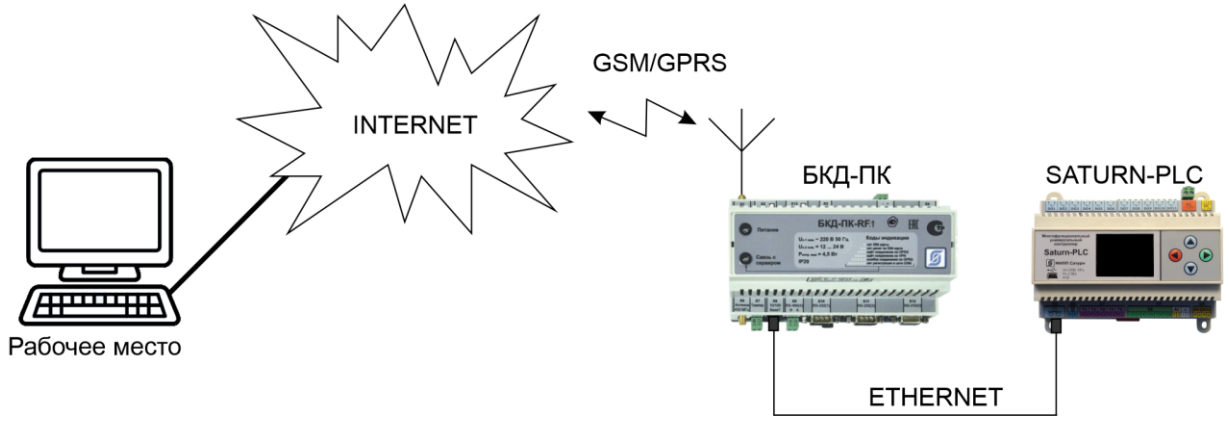
Исходный текст программы "test5.c":

```
#include <sat500.h>

#pragma version 1.0
#pragma inf("$or") "МНПП САТУРН 2020"
#pragma inf("$es") "info@mnppsaturn.ru"
#pragma source on
#pragma inf("$up") "www.mnppsaturn.ru/update/demo/"

//Определяем файл картинки и задаём ему идентификатор 1
#pragma picture(1) "Bmp\Script Badge.png"
//Определяем файл картинки и задаём ему идентификатор 2
#pragma picture(2) "Bmp\script_128x128.png"

//Будем использовать вместо идентификаторов текстовые константы
```

```
#define PICT_1   1
#define PICT_2   2

unsigned int t;

void main(void)
{
   int i=0;
   ClrScr();
   while(1)
   {
        if(i++ & 1)
        {
            DrawPicture(0,0,PICT_1,1);
        }
        else
        {
            DrawPicture(0,0,PICT_2,1);
        }
        ShowScr();
        t=GetTickCount();
        while((GetTickCount()-t)<1000);
    }
}
```

3.24. БКД-ПК И САТ-500

В некоторых случаях контроллер САТ-500 (Saturn-PLC) работает в паре с блоком БКД-ПК как показано на рисунке:

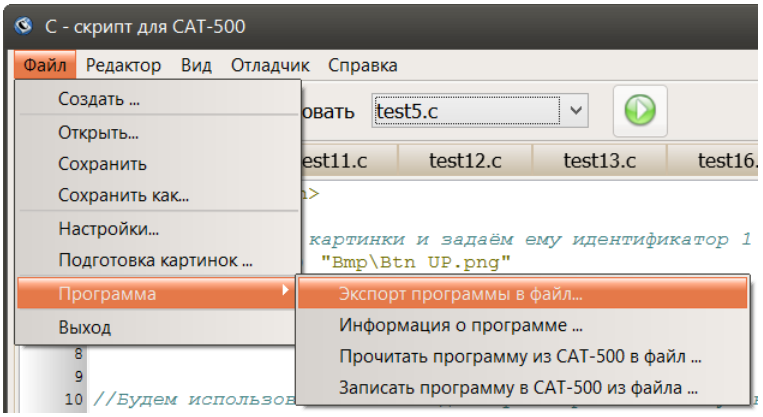


В такой схеме подключения контроллер "Saturn-PLC" не имеет прямого доступа к сети Интернет и, соответственно, недоступен с рабочего места.
Тем не менее, используя специальную утилиту "**rasat**" можно дистанционно обновить программу на языке "С-Скрипт" в контроллере "Saturn-PLC".

Последовательность действий для обновления "С-Скрипт" программы контроллера "Saturn-PLC":

1. Откомпилировать нужную "С-Скрипт" программу в среде разработки.

2. Сохранить откомпилированную программы, выбрав пункт основного меню программы "**Файл/Программа/Экспорт программы в файл...**"



Например, имя программы *"my.prog"*.

2. Загрузить программу "**rasat**" для блока БКД-ПК (Linux ARM) по следующей ссылке:

www.mnppsaturn.ru/ftp/public/soft/rasat/linux-arm/rasat

3. Выполнить консольное подключение к блоку БКД-ПК и записать в любую доступную папку на блоке БКД-ПК два полученных файла:
- утилита "**rasat**"
- С-скрипт программа "my.prog"

Далее все действия выполняются консольным доступом к БКД-ПК.

4. После копирования утилиты "**rasat**" на БКД-ПК, ей нужно дать права на выполнения. Делается это командой:

chmod a+x [/.../]rasat

[/.../] - это путь к файлу "**rasat**", если он в текущей папке, то путь можно не указывать.

5. Краткую справку по консольной утилите "**rasat**" можно получить выполнив утилиту без параметров. В результате будет получена следующая подсказка:

```
Remote Access for SATURN-PLC controller
Usage: rasat.exe [OPTIONS] [FILE_NAME]

Select operation:
  -t, --type=TYPE          type of operation, where TYPE is:
            'find'     - broadcast searching
            'fbd'      - upload fbd program from FILE_NAME
            'clone'    - download fbd program to FILE_NAME
            'cs'       - upload c-script program from FILE_NAME (only via eth)
            'csclone'  - download c-script program to FILE_NAME (only via eth)
            'firmware' - upload firmware from FILE_NAME

Broadcast searching options:
  --baddr=IP               broadcast IP address (default 255.255.255.255)

MODBUS options:
  -a, --addr=NUM           slave MODBUS address (default 48)

For UDP/Modbus/TCP connection:
  -i, --ip=<IP>            IP-address
  -p, --port=NUM           use Modbus/TCP port NUM (default 502)
```

For MODBUS/RTU connection:
-s, --serial=COMn serial port
-b, --baudrate=NNN baud rate: 9600,19200,38400,57600,115200(default)
--parity=PAR parity check, where PAR is: none(default),odd or even

Miscellaneous:
-v, --verbose verbose output
-f, --force force overwrite output file
-V, --version display version information and exit
-h, --help display this help text and exit
--license display license text and exit

Examples:
rasat -t find # broadcast network searching
rasat -t find --baddr=192.168.1.255 # broadcast network searching
rasat -t fbd -i 192.168.1.100 test.fbdbin # upload fbd program via ethernet
rasat -t fbd -s com1 test.fbdbin # upload fbd program via RS-485
rasat -t clone -i 192.168.1.100 test.fbdbin # clone fbd program via ethernet to file
rasat -t cs -i 192.168.1.100 test.prog # upload c-script program via ethernet
rasat -t csclone -i 192.168.1.100 test.prog # clone c-script program via ethernet to file

6. Выполнить поиск контроллеров "Saturn-PLC" в локальной ETHERNET сети (соединение БКД-ПК и Saturn-PLC), запустив утилиту "**rasat"** следующим образом:

rasat -t find

или с указанием адреса подсети:

rasat -t find --baddr=192.168.1.255

Пример результата поиска:

```
>rasat  -t find --baddr=192.168.1.255
#  |Model   |MAC               |IP              |IP-Mask         |GW              |Version
---+--------+------------------+----------------+----------------+----------------+-------
1  |SAT-501 |00-04-A3-98-C8-3C|192.168.1.236   |255.255.255.0   |192.168.1.1     |1.7
2  |SAT-501 |00-04-A3-71-1A-B0|192.168.1.96    |255.255.255.0   |192.168.1.1     |1.7

completed
```

Здесь найдены два контроллера Saturn-PLC (SAT-501) с адресами 192.168.1.98 и 192.168.1.236

6. Выполнить запись С-скрипт программы из БКД-ПК в нужный "Saturn-PLC" следующим образом:

rasat -t cs -i 192.168.1.236 my.prog

,где 192.168.1.236 - IP адрес контроллера Saturn-PLC
my.prog - имя файла С-скрипт программы

Пример успешной записи С-скрипт программы *my.prog*:

```
>  rasat  -t cs -i 192.168.1.236 my.prog
loading file 'my.prog' -> Success
writing program flash memory
[0000000000000000000000000000000000000000] 100% -> Success
finishing writing -> Success
running script -> Success

completed
```

Примечание: Доступна версия программы "**rasat**" для ОС Windows. Её можно загрузить по следующей ссылке:

www.mnppsaturn.ru/ftp/public/soft/rasat/win32/rasat.exe

Используя данную версию можно обновить С-скрипт программу через локальную сеть без блока БКД-ПК.

Также можно выполнить чтение программы из контроллера Saturn-PLC выполнив следующую консольную команду:

rasat -t csclone -i 192.168.1.236 mycopy.prog

,где 192.168.1.236 - IP адрес контроллера Saturn-PLC
mycopy.prog - имя файла С-скрипт программы, прочитанной из контроллера Saturn-PLC

3.25. РЕГИСТРЫ

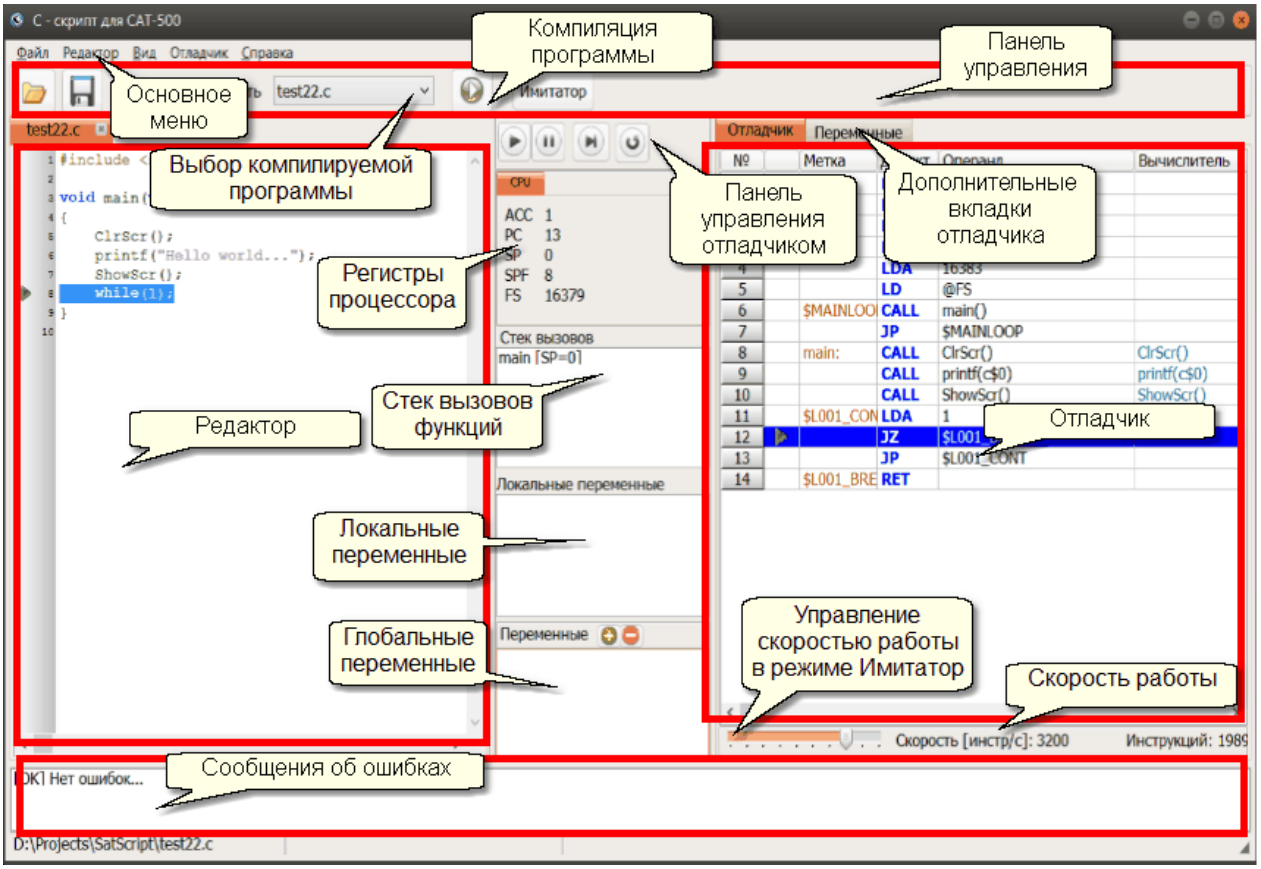
Основные регистры виртуального процессора доступны для чтения.
Это следующие регистры:

Имя	***Описание***
_ACC	Аккумулятор
_PC	Программный счётчик
_SP	Указатель С-стека
_SPF	Указатель на свободную областьС-стека
_VARG	Адрес расположения переменных функции переменной длины

Регистры не могут быть использованы для записи.
Чтение регистров используется только в отладочных целях разработчиками С-скрипта.

4. СРЕДА РАЗРАБОТКИ

После запуска приложения показывается основное окно:



Основное окно состоит из следующих элементов:

Элемент окна	*Описание*
Основное меню	Основное меню программы
Панель управления	Панель управления компиляцией
Выбор компилируемой программы	В этом списке выбирается компилируемый файл - любой из файлов, открытых в редакторе
Кнопка "Компиляция программы"	Позволяет выполнить компиляцию указанной программы. Результат отображается в поле ""Сообщения об ошибках
Редактор	Содержит набор файлов, где каждый файл представляет собой одну вкладку
Сообщения об ошибках	В поле выводятся сообщение об ошибках или сообщение об отсутствии ошибок
Отладчик	Отладчик позволяет выполнить проверку работы написанной программы без загрузки в контроллер CAT-500
Панель управления отладчиком	Набор кнопок, позволяющих запустить программу на выполнение, выполнить пошаговую отладку программы
Дополнительные вкладки отладчика	Набор дополнительных элементов отладки
Регистры процессора	Регистры виртуального процессора
Стек вызовов функций	Показывает последовательность вызова функций. Обновляется при выполнении останова работы программы
Локальные переменные	*Локальные переменные исполняемой функции*. Обновляется при выполнении останова работы программы
Глобальные переменные	*Глобальные переменные программы*. Обновляется при выполнении останова работы программы
Управление скоростью работы в режиме Имитатор	*Регулятор позволяет задать производительность виртуального процессора*

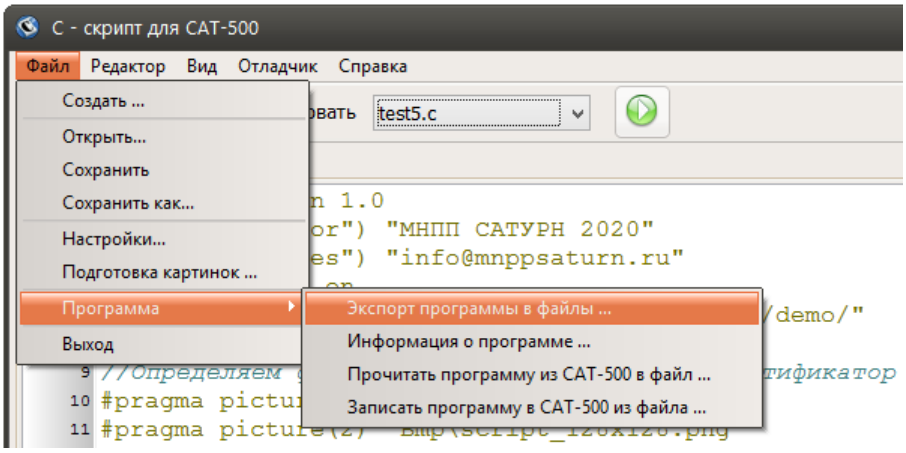
Скорость работы	*Панель индикации производительности на которой показывается количество выполненных инструкций в секунду и общее количество инструкций*

4.1. ИСХОДНЫЙ ТЕКСТ И ИСПОЛНЯЕМАЯ ПРОГРАММА

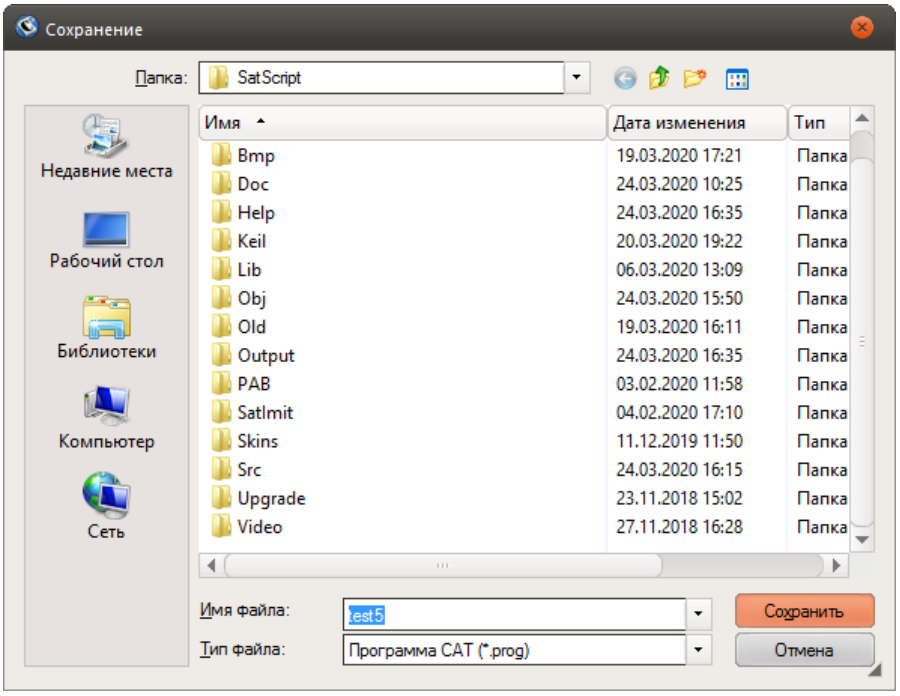
Исходный текст (также исходный код) — текст компьютерной программы на языке программирования C-Script, который может быть прочтён человеком.
Именно этот текст и разрабатывает программист. Контроллер CAT-500 не понимает исходный код, поэтому при помощи компилятора, встроенного в среду разработки исходный текст превращается в исполняемую программу или просто программу.
Исходный код компилируется в программу до записи в контроллер CAT-500.
Исхдный текст, как правило, состоит из одного или нескольких файлов с расширениями ***.c** и ***.h** .
Компилятор превращает исходные коды в файл программы с расширением ***.prog** .
Это файл ***.prog** включает в себя всю необходимую информацию для работы контроллера CAT-500. Например заданные в программе картинки также включаются компилятором в файл программы.
После компиляции файл программы записывается в контроллер и запускается на выполнение.
Программу, записанную в контроллер CAT-500 можно прочитать, и затем загрузить в другой контроллер CAT-500.
Также контроллер CAT-500 умеет обновлять программу с web-сервера.

4.1.1. ЭКСПОРТ ПРОГРАММЫ

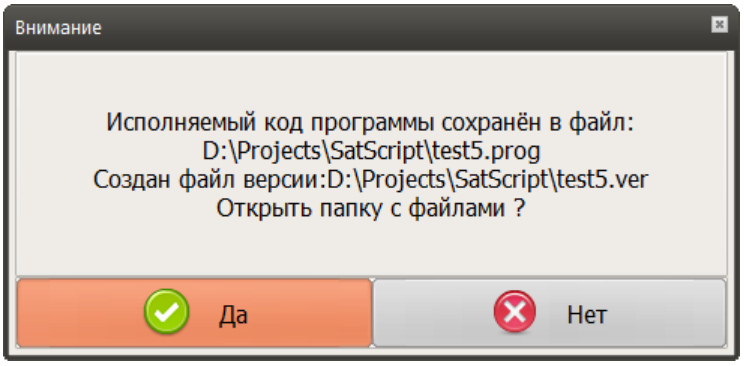
После того как программа откомпилирована можно сохранить исполняемую программу в отдельный файл, который можно записать в контроллер CAT-500.
Для записи программы в отдельный файл необходимо выбрать пункт основного меню **"Файл/Программа/Экспорт программы в файлы ..."**:



Далее откроется диалог выбора пути и имени файла программы:



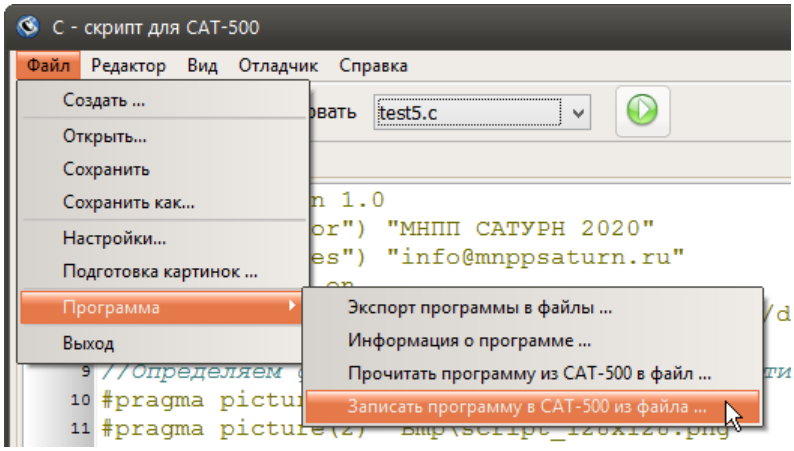
По умолчанию имя файла программы совпадает с именем файла исходного текста программы. Следует выбрать путь для сохранения, ввести имя файла программы и нажать кнопку "*Сохранить*".
Программа будет сохранена в указанном месте с указанным именем.
Кроме файла программы будет создан дополнительный файл номера версии, который используется для создания сервера обновления программы (см. раздел Обновление программы компилятора).
После сохранения будет предложено открыть папку, где были созданы файл программы и файл номера версии.



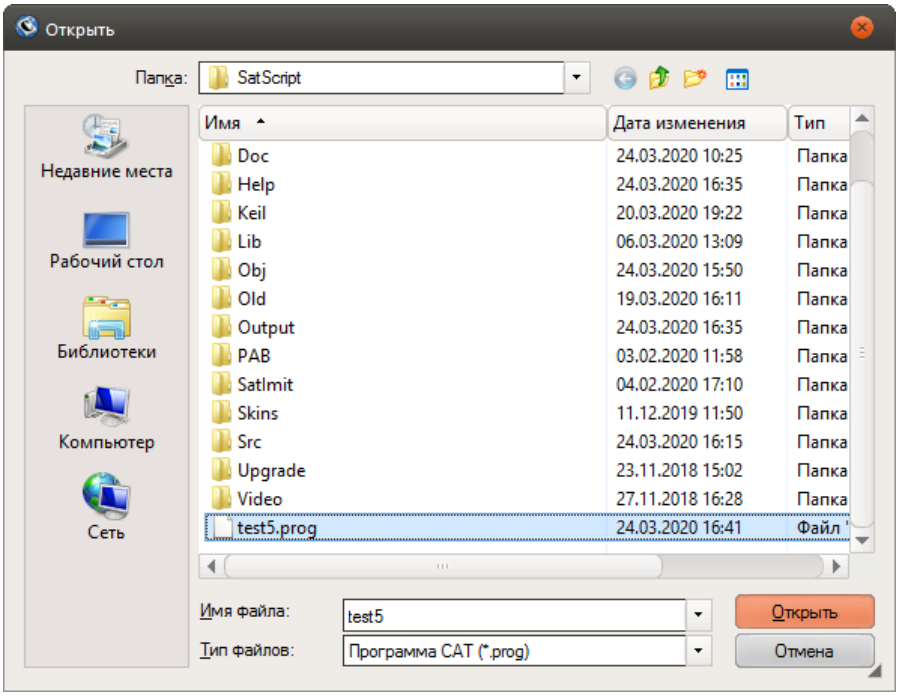
Созданный файл программы "*.prog" можно записать в контроллер CAT-500 даже не имея исходного кода программы. Это действие описано в разделе "Запись программы в CAT-500 из файла".

4.1.2. ЗАПИСЬ ПРОГРАММЫ В CAT-500 ИЗ ФАЙЛА

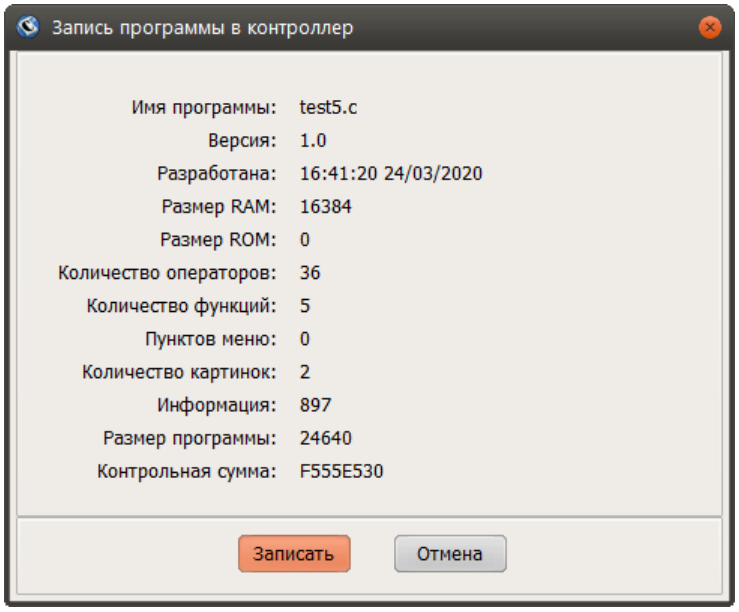
Для записи программы в контроллер CAT-500 из файла следует выполнить следующие действия:
Выбрать пункт основного меню **"Файл/Программа/Записать программу в CAT-500 из файла ..."**:



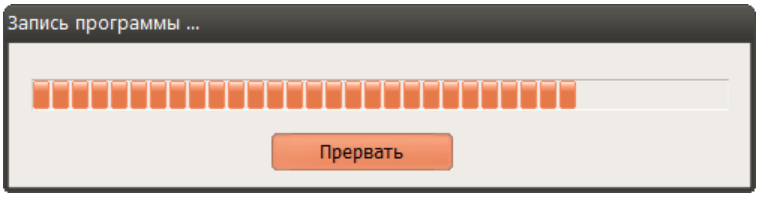
Далее откроется диалог выбора файла программы.



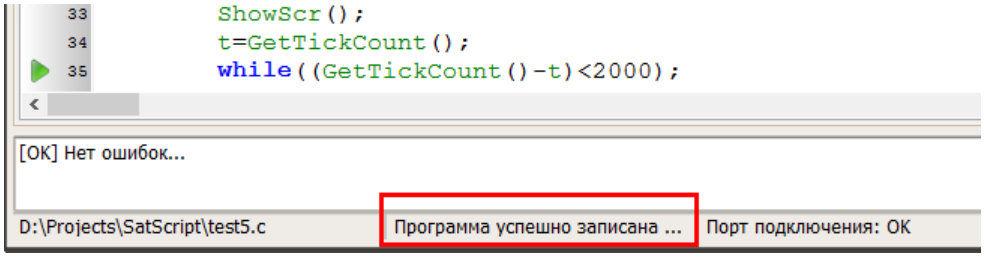
Следует выбрать нужную программу (файл с расширением *.prog) и нажать кнопку "*Открыть*". Откроется дополнительное окно, содержащее основную информацию о программе:



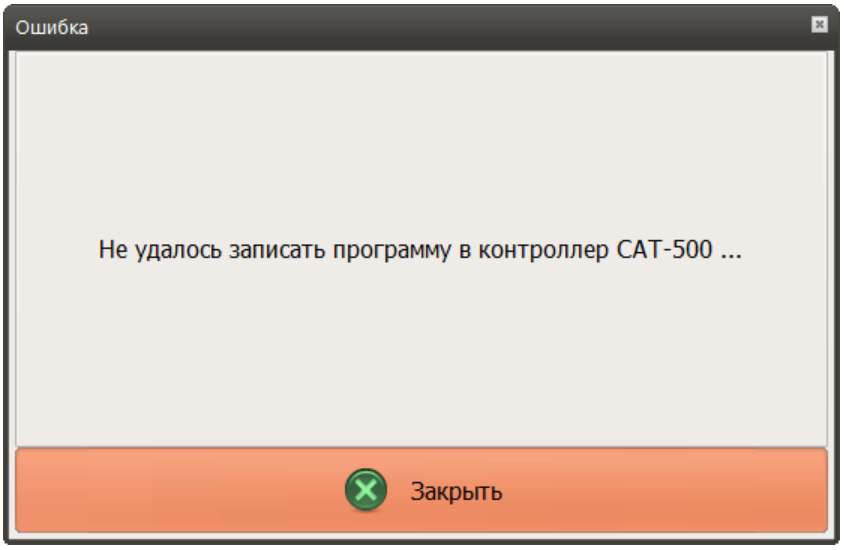
Если программа выбрана правильно, то следует нажать кнопку "*Записать*" - будет выполнена запись файла программы в контроллер CAT-500:



Информация об успешной записи программы показывается только в статусной строке, находящейся в нижней строке основного окна программы:

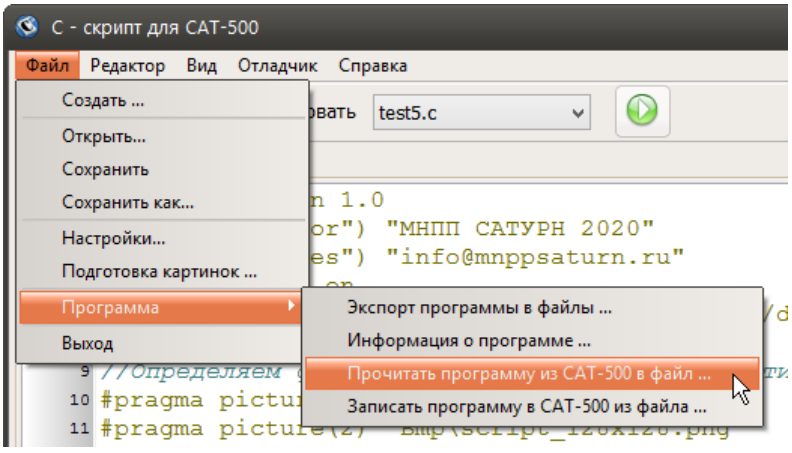


Если программу записать не удалось, то появится специальное окно с сообщением об ошибке:

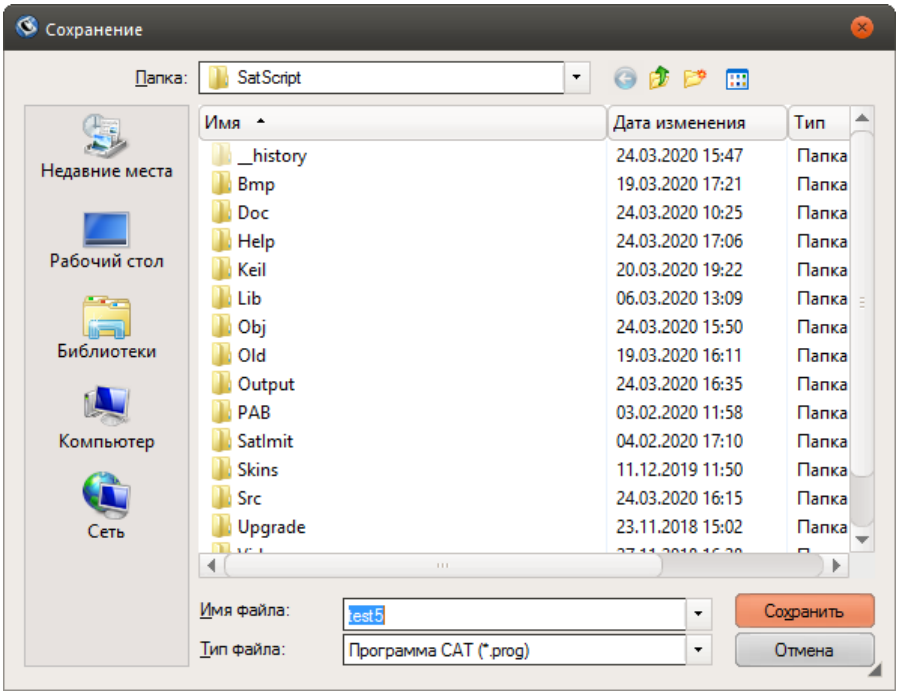


4.1.3. ЧТЕНИЕ ПРОГРАММЫ ИЗ CAT-500 В ФАЙЛ

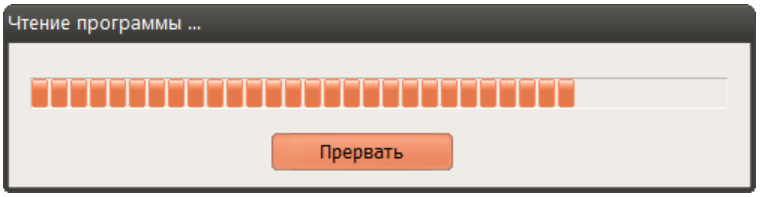
Программа, работающая в контроллере CAT-500, может быть прочитана и сохранена в файл, например, чтобы записать программу в другой контроллер.
Для чтения программы из контроллера CAT-500 в файла следует выполнить следующие действия:
Выбрать пункт основного меню **"Файл/Программа/Прочитать программу из CAT-500 в файл ..."**:



Далее откроется диалог выбора пути и имени файла программы:

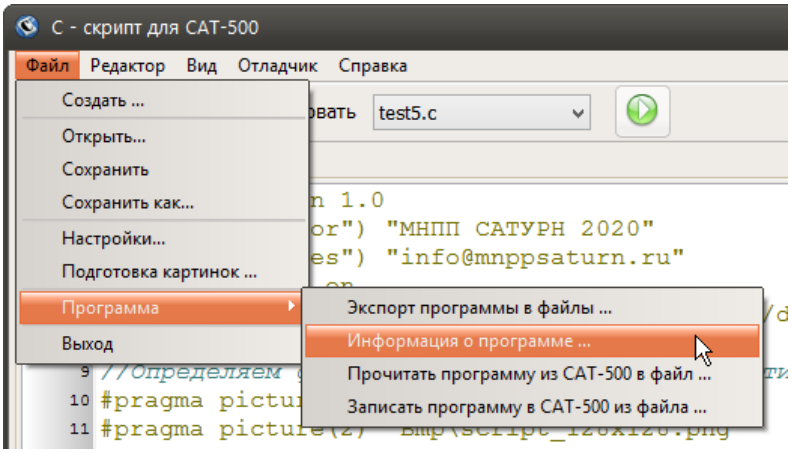


Следует выбрать путь для сохранения, ввести имя файла программы и нажать кнопку "*Сохранить*".
Программа будет сохранена в указанном месте с указанным именем.

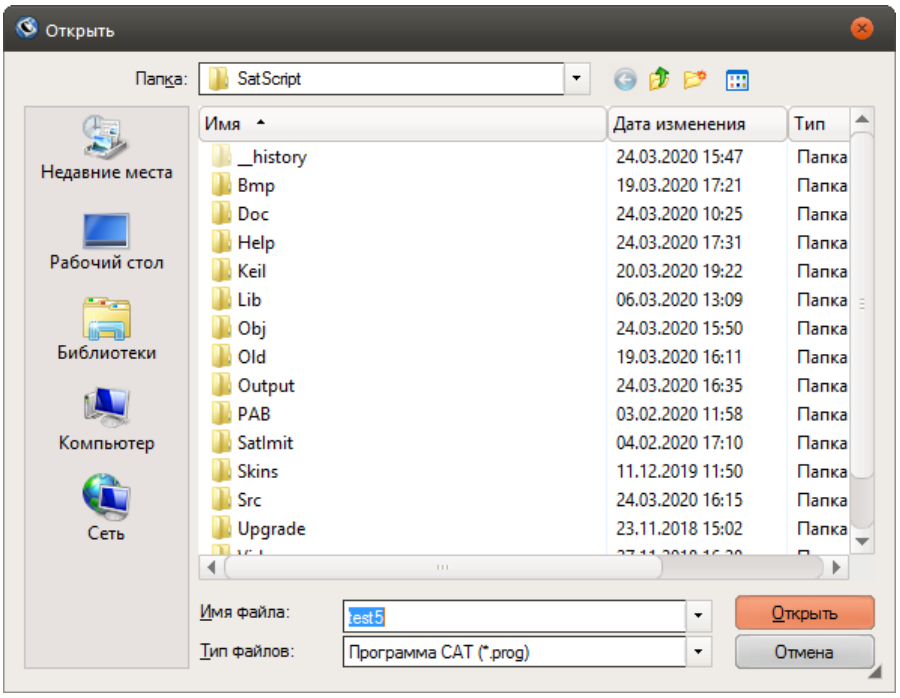


4.1.4. ИНФОРМАЦИЯ О ФАЙЛЕ ПРОГРАММЫ

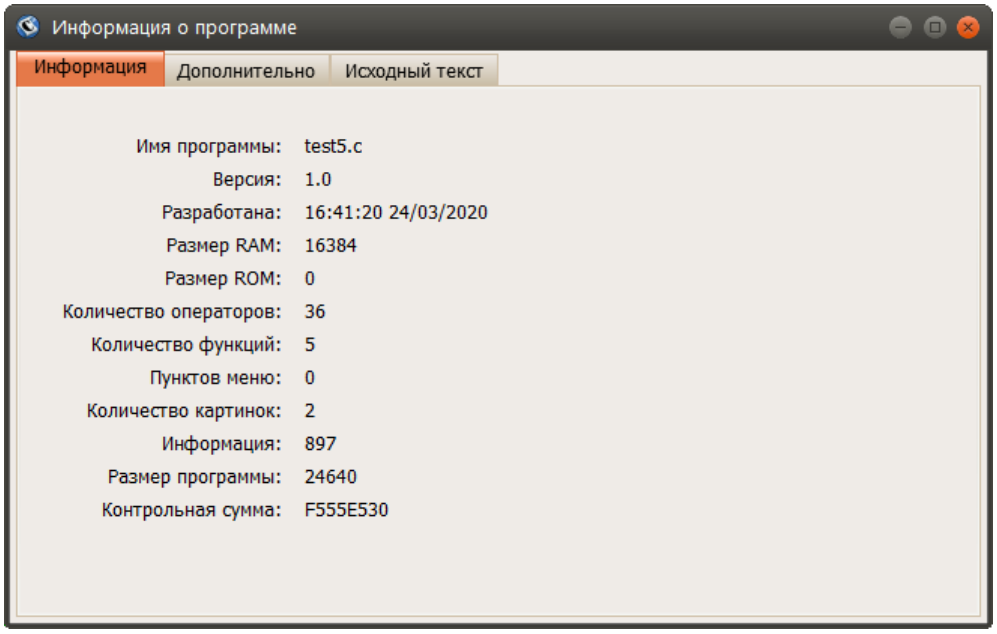
Можно получить различные сведения о программе, находящейся в файле "*.prog".
Для этого следует выполнить следующие действия:
Выбрать пункт основного меню **"Файл/Программа/Информация о программе ..."**:



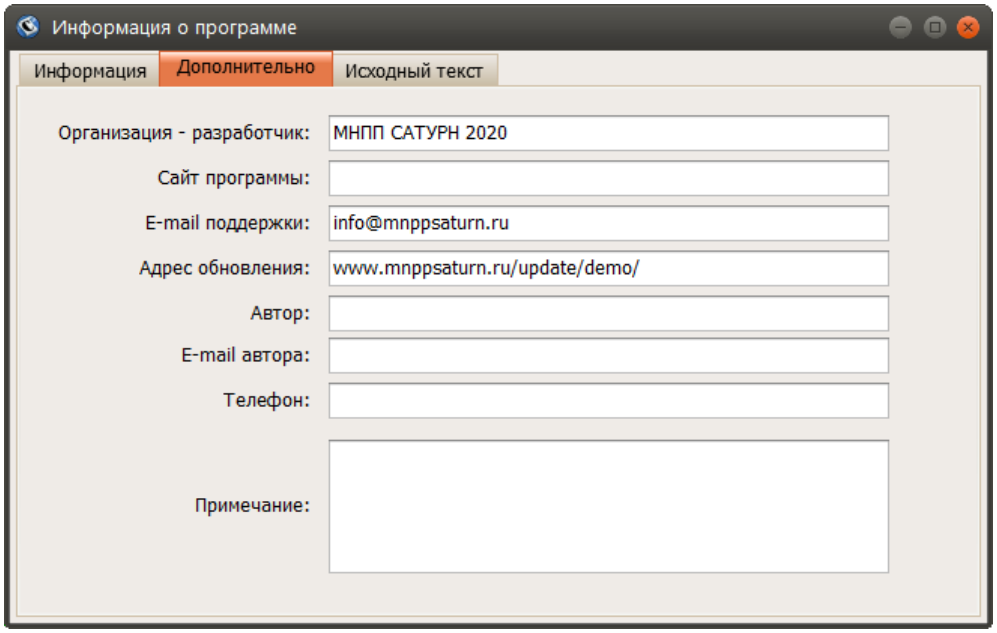
Далее откроется диалог выбора пути и имени файла программы:



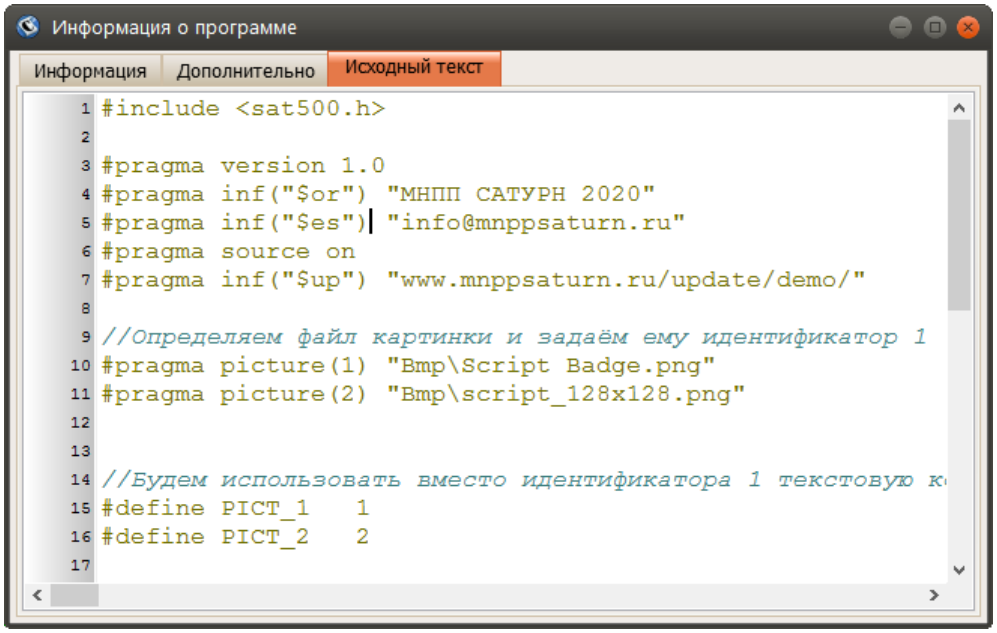
Следует выбрать нужную программу (файл с расширением *.prog) и нажать кнопку "*Открыть*". Откроется дополнительное окно, содержащее несколько вкладок с информацией о программе. На первой вкладке содержится основная информация о программе:



На второй вкладке содержится дополнительная информация - она доступна только если разработчик программы поместил эту информацию в исходном тексте (см. Раздел [Включение в программу дополнительной информации]):



Третья вкладка может содержать исходный текст только в случае если разработчик поместил исходный текст в исполняемую программу исходный текст (см. Раздел [Включение исходного текста программы в файл программы САТ-500](#)).



Этот исходный текст можно выделить, скопировать и в дальнейшем изменить программу, откомпилировать и использовать уже новую версию программы.

Внимание! Исходный текст будет доступен только если разработчик поместил его в исполняемую программу.

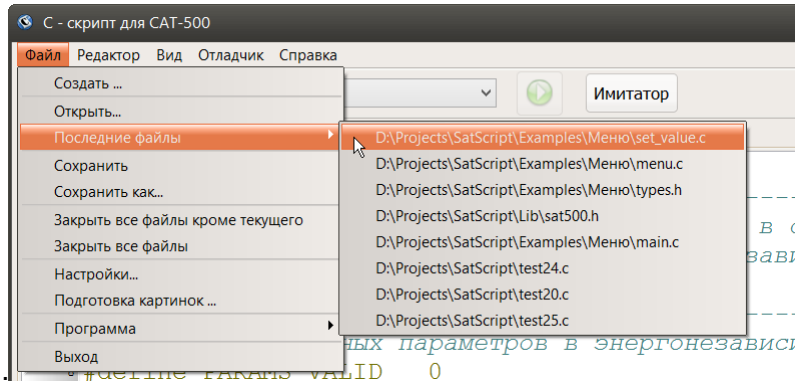
4.2. ОСНОВНОЕ МЕНЮ

Основное меню программы позволяет выполнять множество различных действий.

В основном меню пункты действий сгруппированы по назначению в несколько подменю.

Подменю "Файл"

Подменю "**Файл**"содержит действия предназначенные работы с файлами:

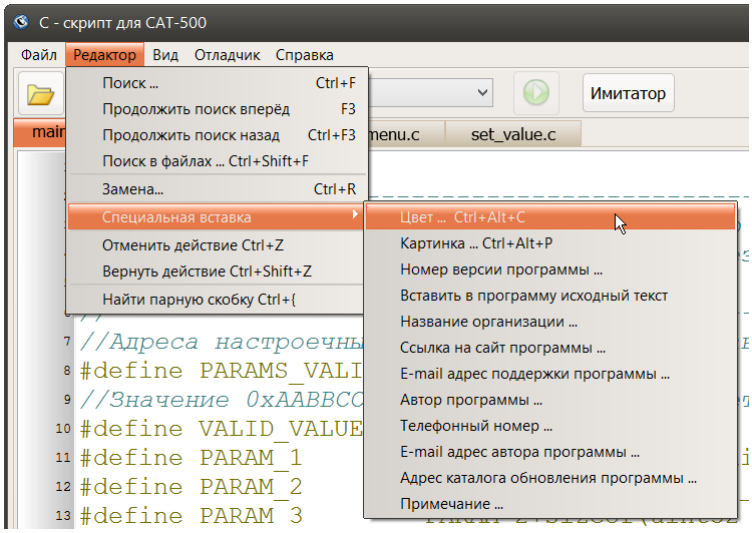


Содержит следующие пункты:

Пункт меню	***Действие***
Создать	Позволяет создать новый файл. Новый созданный файл содержит минимальную программ,у готовую к компиляции
Открыть	Позволяет открыть существующий файл или группу файлов
Последние файлы	Позволяет повторно открыть один из 10 последних открывавшихся ранее файлов
Сохранить	Позволяет сохранить файл измененный файл
Сохранить как	Позволяет сохранить открытый ранее файл с новым именем
Закрыть все файлы кроме текущего	Позволяет закрыть все открытые в редакторе файлы кроме текущего редактируемого
Закрыть все файлы	Позволяет закрыть все открытые в редакторе файлы
Настройки	Открывает окно основных настроек программы
Подготовка в картинок	Открывает дополнительные окно в котором можно подготовить картинку загрузки в контроллер САТ-500
Программа	Позволяет выполнить различные действия с откомпилированные программой или программой загруженной в контроллер САТ-500: *Экспорт программы* *Запись программы в САТ-500 из файла* *Чтение программы из САТ-500 в файл* Информация о файле программы
Выход	Позволяет закрыть программу

Подменю "Редактор"

Подменю содержит действия связанные с редактированием открытых файлов:

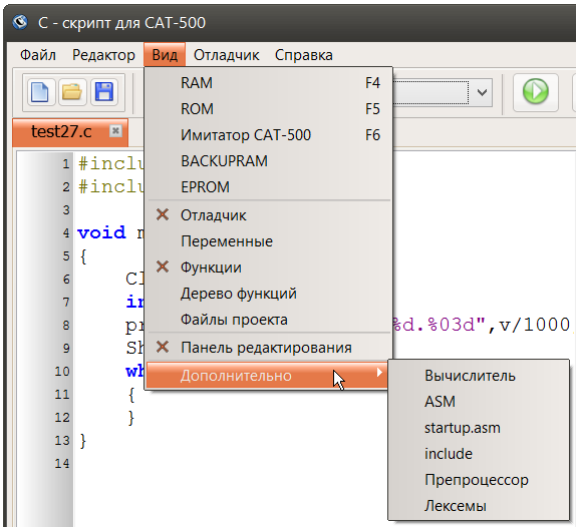


Содержит следующие пункты:

Пункт меню	***Действие***
Поиск	Выполняет поиск в текущем редактируемом файле
Продолжить поиск вперед	При выборе данного пункта меню или по нажатию клавиши "***F3***" продолжается поиск в направлении к концу файла текста, заданного в подменю *Поиск*
Продолжить поиск назад	При выборе данного пункта меню или по нажатию сочетания клавиши "***Ctrl***" и "***F3***" продолжается поиск в направлении к началу файла текста, заданного в подменю *Поиск*
Поиск в файлах	Открывается дополнительное окно в котором можно выполнить поиск во всех открытых в редакторе файлах
Замена	Позволяет выполнить замену текста в текущем редактируемом файле
Специальная вставка	Позволяет править в исходный текст программы дополнительные опции *Директива #pragma*
Отменить действие	Позволяет отменить последнее действие в редакторе
Вернуть действие	Позволяет вернуть последнее отмененное действие в редакторе
Найти парную скобку	Позволяет нет найти парную скобку в редакторе. Для этого необходимо предварительно поставить курсор на любую скобку

Подменю "Вид"

Подменю содержит действия позволяющие настроить вид отладчика и открыть дополнительные окна:



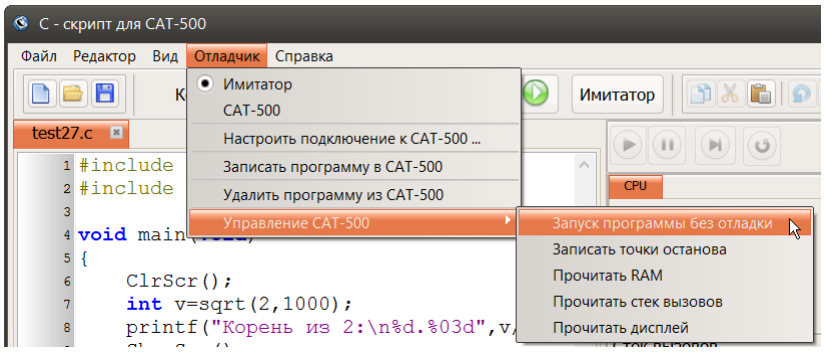
Содержит следующие пункты:

Пункт меню	***Действие***
RAM	Открывает специальное дополнительное *окно RAM*, показывающее содержимое памяти контроллера (RAM)
ROM	Открывает специальное дополнительное *окно ROM*, показывающее содержимое неизменяемой памяти контроллера (ROM)
Имитатор CAT-500	Открывает специальное окно программы *Имитатор CAT-500* для отладки программы без оборудования
BACKUP RAM	Открывает специальное *окно BACKUPRAM*, показывающее содержимое энергонезависимой памяти контроллера BACKUPRAM
EPROM	Открывает специальное *окно EPROM*, показывающее содержимое энергонезависимой памяти контроллера EPROM
Отладчик	Это основная отладочная вкладка программы на котором на которой показывается пополняемая инструкции языка псевдоассемблер
Переменные	На панели отладчика открывает специальную *вкладку "Переменные"*, на которой можно посмотреть состояние глобальных переменных программы
Функции	На панели отладчика открывает специальную *вкладку "Функции"*, на которой можно посмотреть все используемые в программе функции и их параметры
Дерево функций	Позволяет открыть специальную *вкладку "Дерево функций"*, на которой отображается дерево взаимных вызовов функций
Файлы проекта	Позволяет открыть специальную *вкладку "Файлы проекта"*, на которой отображается список всех файлов, входящих в проект
Вычислитель	Позволяет открыть *вкладку "Вычислитель"* на которойможно набрать любое арифметическое или логическое выражение языка С-скрипт с использованием глобальных переменных и числовых констант и получить результат

ASM	Позволяет открыть вкладку, на которой отображается откомпилированная программа на языке псевдоассемблер
startup.asm	Позволяет открыть вкладку, на которой отображается стартап код программы
include	Позволяет открыть вкладку, на которой показывается отладочная информация - это результат компиляции на этапе включения
Препроцессор	Позволяет открыть вкладку, на которой показывается отладочная информация - это результат компиляции на этапе после обработки файла препроцессором
Лексемы	Позволяет открыть вкладку, на которой показывается отладочная информация - исходный текст программы, разбиты на лексемы. Вкладка используется только разработчиками программы

Подменю "Отладчик"

Подменю содержит действия, связанные с отладкой разрабатываемой программы как на имитаторе, так и в контроллере CAT-500:

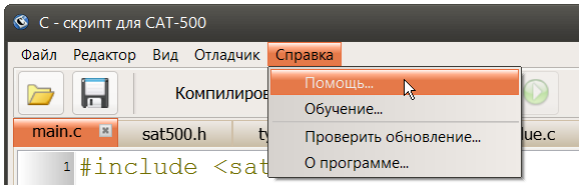


Содержит следующие пункты:

Пункт меню	***Действие***
Имитатор	Позволяет выбрать режим отладки программы без контроллера CAT-500 в имитаторе программы *Отладка в имитаторе*
CAT-500	Выбирает режим отладки программы в контроллере CAT-500 *Отладка в CAT-500*
Настроить подключение к CAT-500	Позволяет открыть дополнительное окно, в котором можно настроить IP адрес контроллера
Записать программу в CAT-500	Позволяет запись программу в контроллер
Удалить программу из CAT-500	Позволяет удалить программу из контроллера
Управление CAT-500	Содержит подменю дполнительных действий с контроллером

Подменю "Справка"

Подменю содержит дополнительные действия

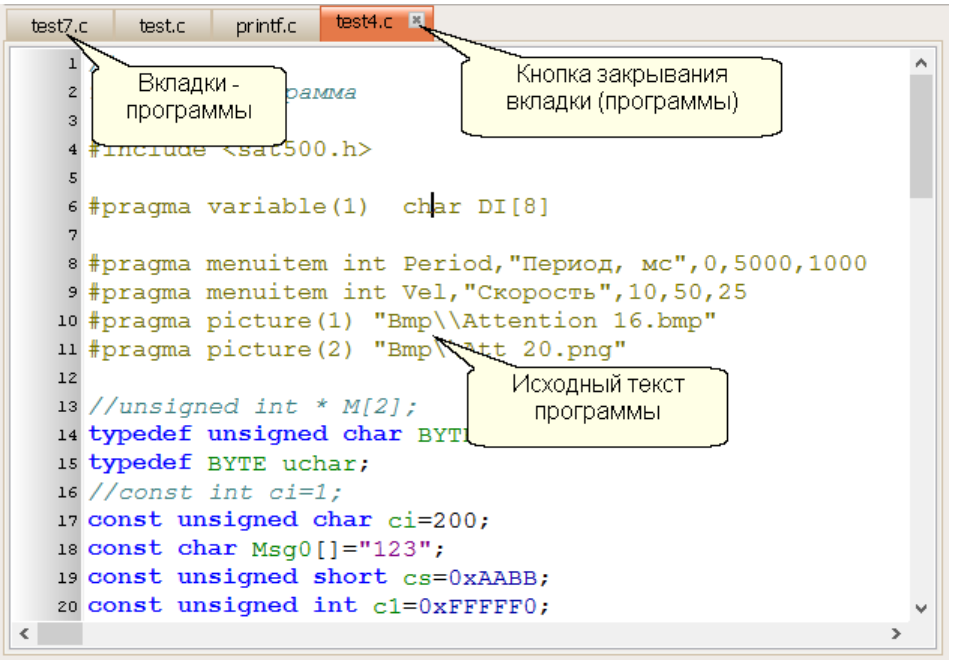


Содержит следующие пункты:

Пункт меню	*Действие*
Помощь	Позволяет вызвать окно справочной системы программы
Обучение	Позволяет просмотреть видеоуроки по использованию программы
Проверить обновления	Позволяет выполнить автоматическое обновление программы при наличии подключения к сети интернет *Обновление программы компилятора*
О программе	Открывает дополнительные окно с информацией о версии программного обеспечения

4.3. РЕДАКТОР

Редактор представляет собой набор вкладок, где один редактируемый файл отображается на одной вкладке.

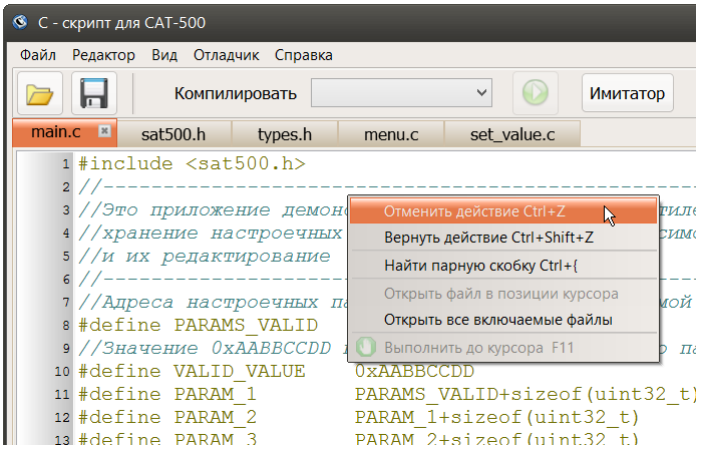


Набор функций редактирования соответствует стандартному редактору.

Предложение кода
Вставка/удаление комментариев
Автозавершение текста
Поиск фигурной скобки
Поиск в текущем окне редактирования
Быстрый поиск

Поиск в файлах
Замена текста
Отменить/вернуть последнее действие
"Горячие" клавиши

Контекстное меню редактора



Содержит следующие пункты:

Пункт меню	***Действие***
Отменить действие	Позволяет отменить последнее действие в редакторе
Вернуть действие	Позволяет вернуть последнее отмененное действие в редакторе
Найти парную скобку	Позволяет нет найти парную скобку в редакторе. Для этого необходимо предварительно поставить курсор на любую скобку
Открыть файл в позиции курсора	Позволяет открыть файл заданный в исходном тексте в директиве *#include*
Открыть все включаемые файлы	Позволяет открыть в редакторе все файлы включаемые при помощи директивы *#include* в текущем редактируемом файле
Выполнить до курсора	Позволяет запустить выполнение программы в имитаторе или в контроллере CAT-500 до от текущей позиции до положение курсора в редакторе

4.3.1. ПРЕДЛОЖЕНИЕ КОДА

Одна из самых полезных функций редактора вызывается сочетанием клавиш "*Ctrl+ПРОБЕЛ".*
Введите один или несколько символов начала имени функции, переменной и т.д. и нажмите сочетание клавиш *"Ctrl+ПРОБЕЛ"*.
Появится окно в котором будут отображены варианты предложения завершения уже набранного кода:

```
14          GotoXY(4,40);
15          printf("%5d",reg[0]);
16          GotoXY(4,68);
17          printf("%5d",reg[1]);
18          ShowScr();
19          Mo
20      }
21  }
22
```

Завершение кода

ModbusCount

unsigned int **ModbusCount**
unsigned short int **ModbusCrc16(unsigned char * da**
char **ModbusReadCoils(unsigned short int adr,unsi**
char **ModbusReadDiscreteInputs(unsigned short i**
char **ModbusReadHoldingRegisters(unsigned shor**
char **ModbusReadInputRegisters(unsigned short i**
char **ModbusReadStatus(unsigned char * status)**
char **ModbusResult(void)**
unsigned char **ModbusSlave**
void **ModbusTcpClose(void)**
char **ModbusTcpConnected(void)**
unsigned int **ModbusTcpCount**

[OK] Нет ошибок...

Можно продолжить ввод - количество предложений будет уменьшаться. Как только увидите нужный текст в окне предложения передвиньте указатель текущего текста клавишами "*ВВЕРХ*" и "*ВНИЗ*", и нажмите клавишу "*ENTER*" - выбранный текст будет вставлен в редактор:

```
12      while(1)
13      {
14          GotoXY(4,40);
15          printf("%5d",reg[0]);
16          GotoXY(4,68);
17          printf("%5d",reg[1]);
18          ShowScr();
19          ModbusCount
20      }
21  }
22
```

Также можно вызывать окно предложения кода не вводя предварительно текст - нажать сочетание клавиш *"Ctrl+ПРОБЕЛ"* и далее вводить текст.

4.3.2. ВСТАВКА/УДАЛЕНИЕ КОММЕНТАРИЕВ

Для вставки или удаления комментариев удобно использовать сочетание клавиш "*Ctrl+/".* Необходимо предварительно выделить нужные строки:

```
14          GotoXY(4,40);
15          printf("%5d",reg[0]);
16          GotoXY(4,68);
17          printf("%5d",reg[1]);
18          ShowScr();
19      }
20 }
```

и нажать сочетание клавиш "*Ctrl+/*".
В начале выделенных строк появятся символы комментария:

```
14          GotoXY(4,40);
15          printf("%5d",reg[0]);
16          GotoXY(4,68);
17 //         printf("%5d",reg[1]);
18 //      ShowScr();
19      }
```

Повторное нажатие сочетания клавиш "*Ctrl+/*" уберёт символы комментария.

4.3.3. АВТОЗАВЕРШЕНИЕ ТЕКСТА

Удобным средством быстрого набор текста является функция "Автозавершение".
Для того, чтобы воспользоваться функции автозавершения. необходимо набрать две буквы ключевого слова языка С и затем нажать сочетание клавиш "*Ctrl + J*".
Редактор добавит весь необходимый текст вместо введенных двух символов.
Например, если набрать сочетание **sw** и затем "*Ctrl + J*", то получим следующее текст :

sw CTRL+J

```
switch()
{
   case :
   {
      break;
   }
   default:
   {
      break;
   }
}
```

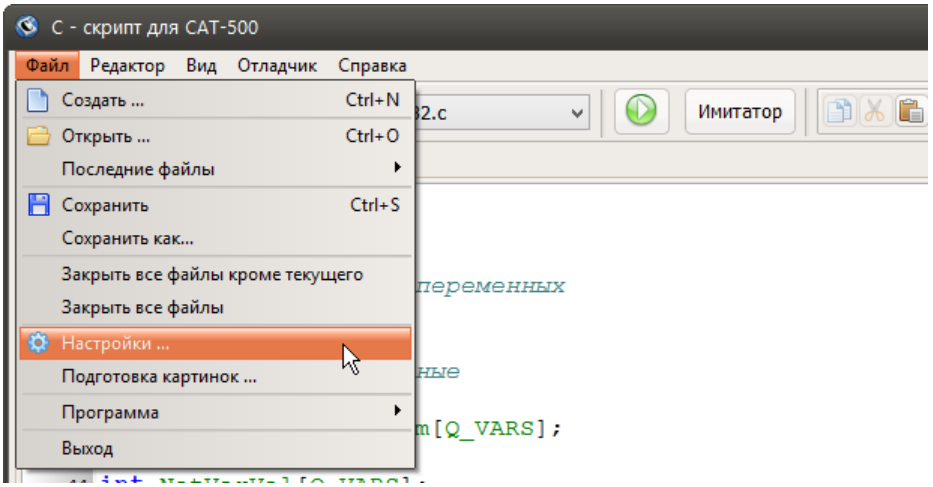
Ниже приведена таблица стандартных автозавершений:

Текст	***Автозавершение***
fo	*for(|;;)* *{* *}*
wh	*while(|)* *{*

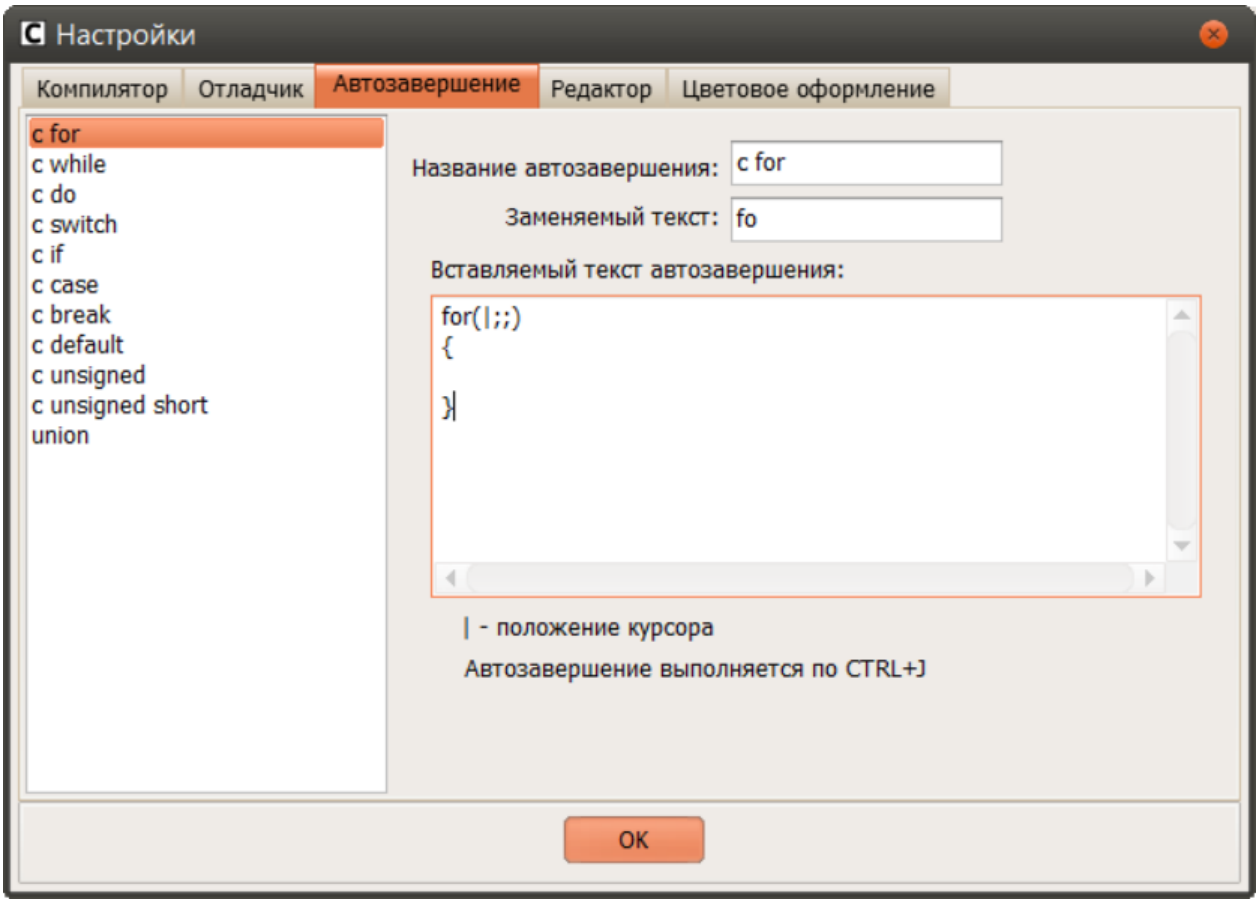
	}
do	*do* *{* *}* *while(|);*
sw	*switch(|)* *{* *case :* *{* *break;* *}* *default:* *{* *break;* *}* *}*
if	*if(|)* *{* *}* *else* *{* *}*
ca	*case |:* *{* *break;* *}*
br	*break;*
de	*default:* *{* | *break;* *}*
un	*unsigned |*
us	*unsigned short |*

Добавление текстов автозавершения

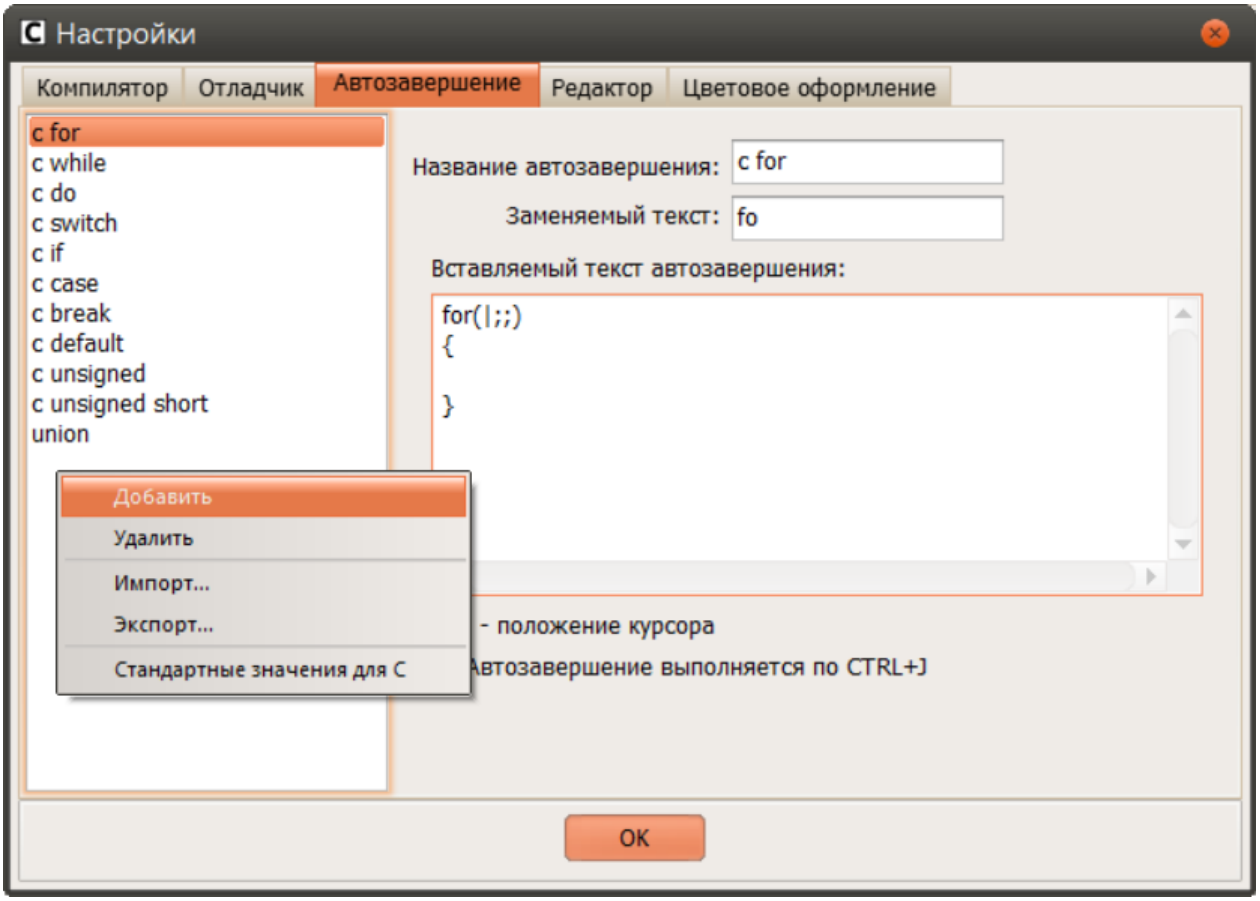
Тексты автозавершения можно дополнить своими собственными. Для этого необходимо выбрать пункт "*Файл/Настройки*" основного меню программы.



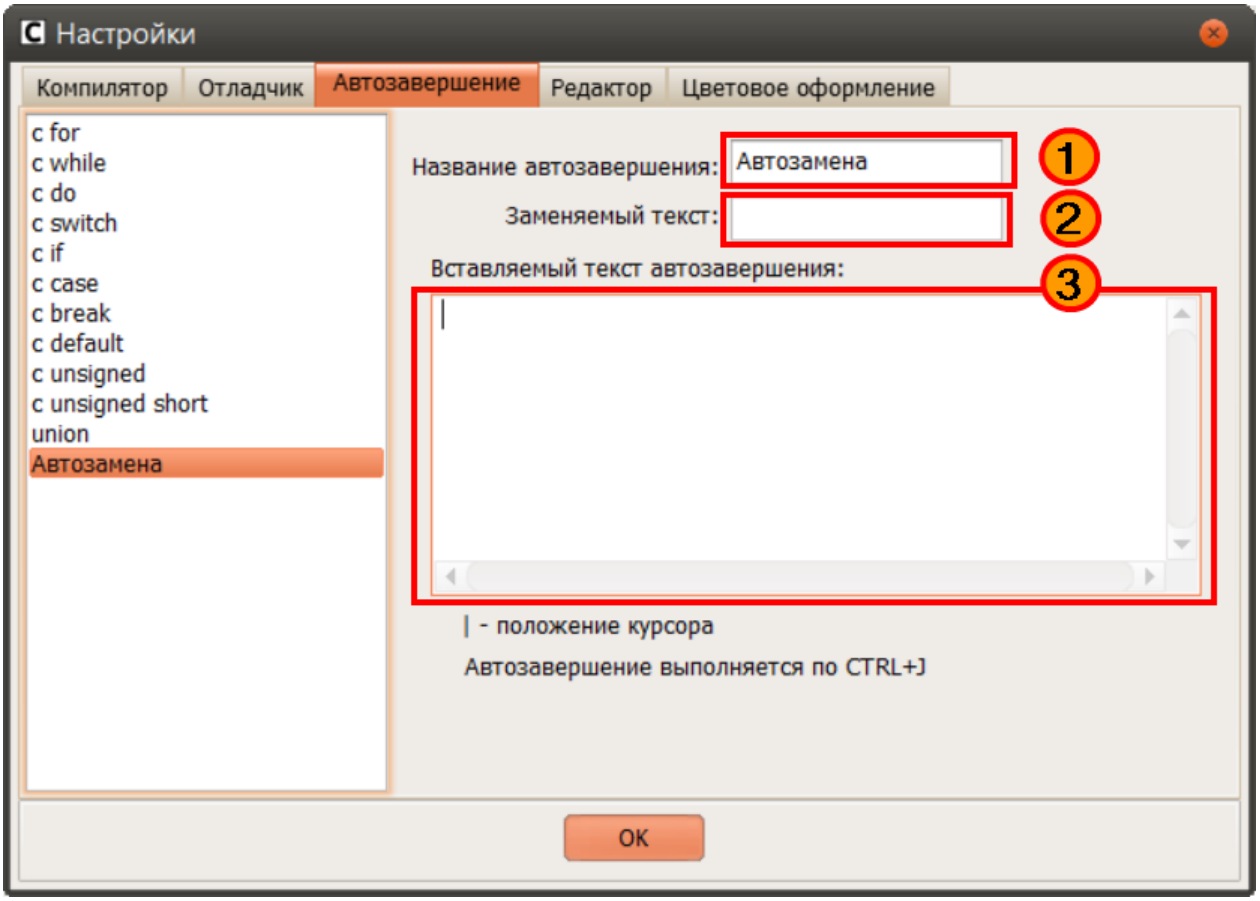
Далее необходимо перейти на вкладку "*Автозавершение*":



На вкладке показаны все элементы автозавершения, которой можно изменить.



Для добавления нового элемента автозавершения необходимо вызвать контекстное меню списка элементов правой кнопкой мышки и выбрать в меню пункт "*Добавить*".



Появится новый элемент списка автозавершения и он будет выбран в качестве текущего.
Для этого элемента необходимо задать:

1. Название автозавершения
2. Заменяемый текст
3. Вставляемый текст автозавершения

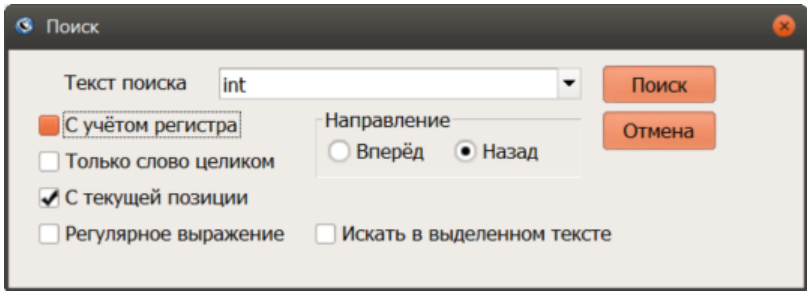
По нажатию кнопки OK, можно закрыть окно настройки. Все изменения автозавершения будут сохранены.

4.3.4. ПОИСК ФИГУРНОЙ СКОБКИ

Удобным способом поиска утерянной фигурной скобки или лишней скобки является использование сочетаний клавиш "*Ctrl+{*" и "*Ctrl+}*".
Необходимо поставить курсор на любую фигурную скобки и нажать сочетания клавиш "*Ctrl+{*" или "*Ctrl+}*".
Курсор будет перемещён на парную фигурную скобку.

4.3.5. ПОИСК В ТЕКУЩЕМ ОКНЕ РЕДАКТИРОВАНИЯ

Для выполнения поиска можно воспользоваться сочетанием клавиш "*Ctrl+F*".
Можно предварительно установить курсор на слово, которое будет искаться и нажать сочетание "*Ctrl+F*".
Откроется окно поиска, показанное ниже:



В поле ввода "*Текст поиска*" можно ввести другой текст, который будет искаться.
Также можно дополнительно задать:

С учётом регистра	Поиск будет выполнен с учётом регистра "Текста поиска"
Только слово целиком	При поиске будет искаться слово целиком
С текущей позиции	Поиск будет выполнен, начиная с позиции в которой находится курсор
Регулярное выражение	В поле "Текст поиска" задано регулярное выражение, используемое для поиска
Направление	Задаёт направление поиска: *Вперёд* - далее после позиции курсора в сторону конца текста *Назад* - в сторону начала текста
Искать в выделенном тексте	Поиск будет выполнен только в выделенном фрагменте текста

Для начала поиска следует нажать кнопку "*Поиск*" в окне или клавишу "*ENTER*" на клавиатуре.
Для отмены поиска следует нажать кнопку "*Отмена*" в окне или клавишу "*ESC*" на клавиатуре.

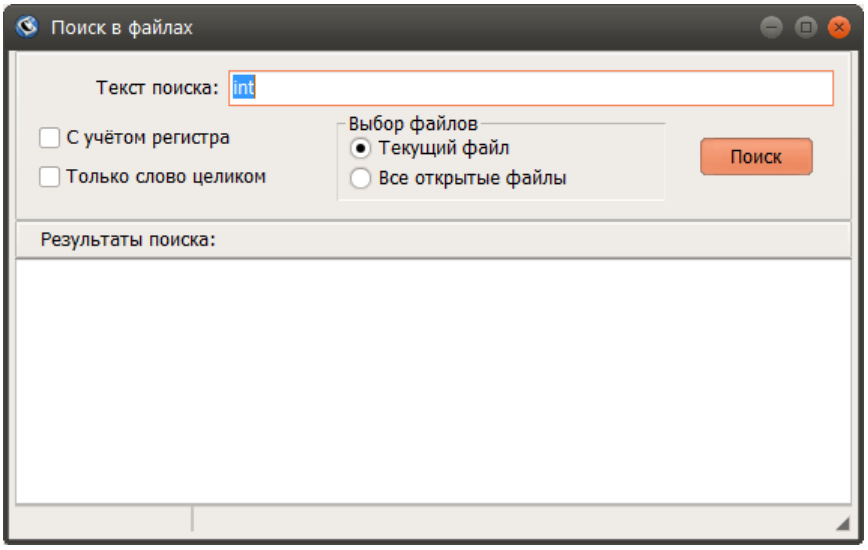
ВНИМАНИЕ!
После выполнения поиска повторный поиск (продолжение в направлении к концу текста) выполняется клавишей "*F3*" на клавиатуре.
Для поиска в обратном направлении (к началу текста) используется сочетание клавиш "*Ctrl+F3*".

4.3.6. БЫСТРЫЙ ПОИСК

Для быстрого поиска нужно предварительно установить курсор на слово, которое будет искаться и нажать клавишу "*F12*" на клавиатуре.
Будет выполнен поиск в направлении конца файла слова, на которое указывает курсор.
Для поиска в обратном направлении к началу файла следует удерживать нажатой клавишу "*Ctrl*" и затем нажать "*F12*".
Данный тип поиска является регистрозависимым.

4.3.7. ПОИСК В ФАЙЛАХ

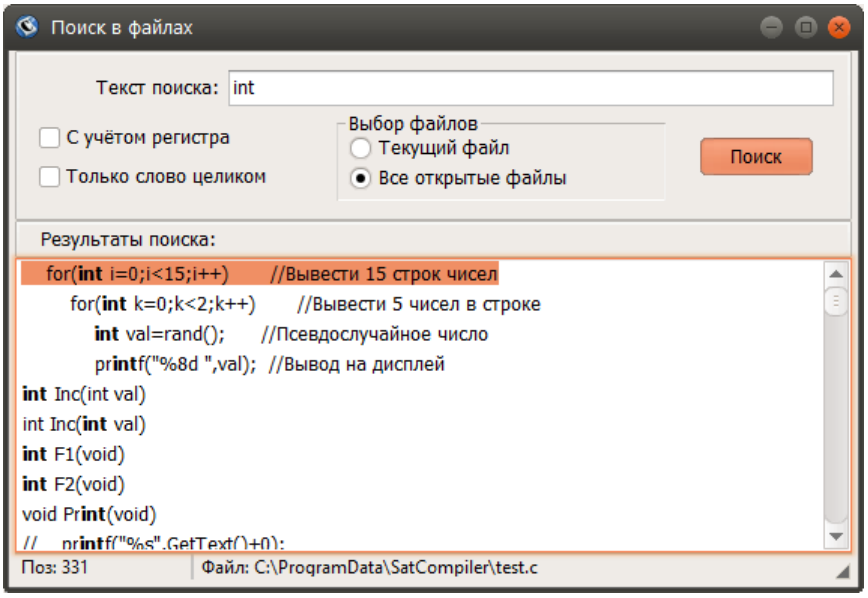
Для поиска в одном или нескольких файлах можно использовать сочетания сочетания клавиш "*Ctrl+Shift+F*".
Можно предварительно установить курсор на слово, которое будет искаться и нажать сочетание "*Ctrl+Shift+F*".
Откроется окно поиска, показанное ниже:



В поле ввода "*Текст поиска*" можно ввести другой текст, который будет искаться.
Также можно дополнительно задать:

С учётом регистра	Поиск будет выполнен с учётом регистра "Текста поиска"
Только слово целиком	При поиске будет искаться слово целиком
Выбор файлов	Поиск будет выполнен: *Текущий файл* - только в одном файле, который редактируется *Все открытые файлы* - во всех открытых файлах

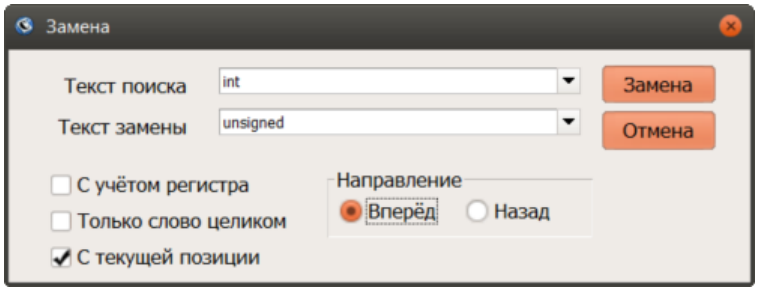
Для начала поиска следует нажать кнопку "*Поиск*" в окне или клавишу "*ENTER*" на клавиатуре.
Для отмены поиска следует нажать кнопку "*Отмена*" в окне или клавишу "*ESC*" на клавиатуре.
Будет выполнен поиск и окно примет следующий вид:



В нижней части окна показаны строки найденные в открытых файлах.
Для перехода к редактору следует выполнить двойной клик по нужной строке - будет показан искомый фрагмент в редакторе.

4.3.8. ЗАМЕНА ТЕКСТА

Для выполнения замены текста можно воспользоваться сочетанием клавиш "*Ctrl+R*".
Можно предварительно установить курсор на слово, которое будет заменяться и нажать сочетание "*Ctrl+R*".
Откроется окно замены, показанное ниже:

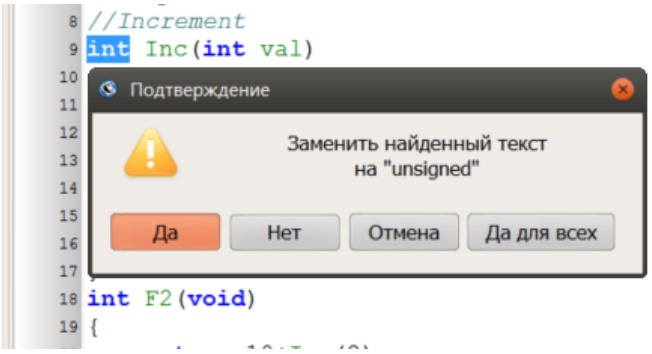


В поле ввода "*Текст поиска*" можно ввести текст, который будет заменяться, а в поле "Текст замены" - текст который будет вставляться.
Также можно дополнительно задать:

С учётом регистра	Замена будет выполнена с учётом регистра текста в поле "Текста поиска"
Только слово целиком	При замене будет искаться слово целиком
С текущей позиции	Поиск для замены будет выполнен, начиная с позиции в которой находится курсор
Направление	Задаёт направление поиска: *Вперёд* - далее после позиции курсора в сторону конца текста *Назад* - в сторону начала текста

Для начала поиска и замены следует нажать кнопку "*Поиск*" в окне или клавишу "*ENTER*" на клавиатуре.
Для отмены поиска и замены следует нажать кнопку "*Отмена*" в окне или клавишу "*ESC*" на клавиатуре.

После того как заданный текст найден появится окно с запросом подтверждения замены:



Необходимо нажать кнопку "Да" для выполнения замены или кнопку "Нет" для отказа от замены - поиск будет продолжен.
Кнопка "Отмена" прекращает дальнейший поиск и замену.
Кнопка "Да для всех" выполняет замены без подтверждения до конца текста в редакторе.

4.3.9. ОТМЕНИТЬ/ВЕРНУТЬ ПОСЛЕДНЕЕ ДЕЙСТВИЕ

Отменить последнее действие можно нажатием сочетаний клавиш "*Ctrl+Z*".
Вернуть отменённое действие можно нажатием сочетаний клавиш "*Ctrl+Shift+Z*".

Редактор запоминает все действия пользователя до сохранения изи загрузки файла.

4.3.10. "ГОРЯЧИЕ" КЛАВИШИ

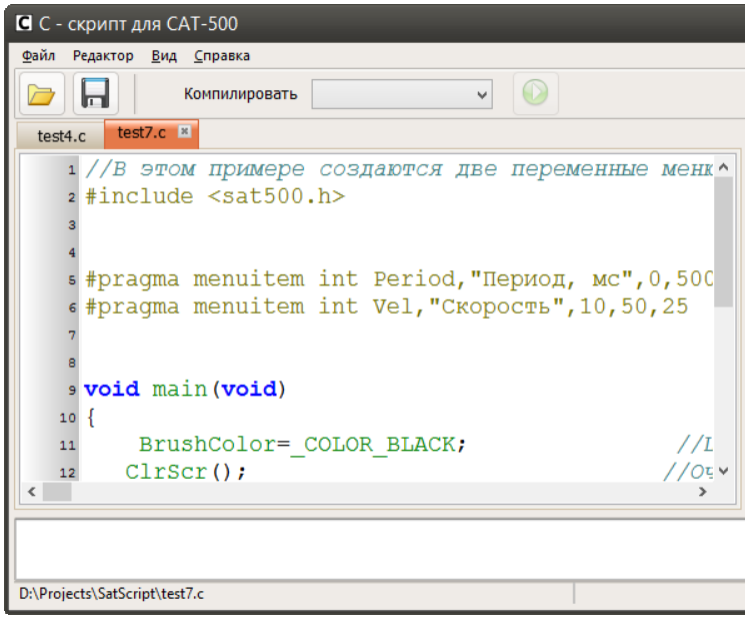
Некоторые дополнительные функции доступны через следующие "горячие" клавиши:

Клавиша или сочетание клавиш	***Функция***
Ctrl+0 *Ctrl+1* *...* *Ctrl+9*	Перейти к маркеру номер 0 Перейти к маркеру номер 1 ... Перейти к маркеру номер 9
Ctrl+Shift+0 *Ctrl+Shift+1* *...* *Ctrl+Shift+9*	Установить маркер номер 0 Установить маркер номер 1 ... Установить маркер номер 9
Up	Курсор вверх
Shift+Up	Выделение вверх
Ctrl+Up	Скроллинг вверх без изменения положения курсора
Down	Курсор вниз
Shift+Down	Выделение вниз
Ctrl+Down	Скроллинг вниз без изменения положения курсора
Left	Курсор влево
Shift+Left	Выделение влево
Ctrl+Left	Перемещение влево по словам
Ctrl+Shift+Left	Выделение слова влево
Right	Курсор вправо
Shift+Right	Выделение вправо
Ctrl+Right	Перемещение вправо по словам
Ctrl+Shift+Left	Выделение слова вправо
PgDn	Курсор вниз на страницу
Ctrl+PgDn	Вниз страницы
Shift+PgDn	Выделение страниц вниз
Ctrl+Shift+PgDn	Выделение вниз до конца
PgUp	Курсор вверх на страницу
Ctrl+PgUp	Вверх страницы
Shift+PgUp	Выделение страниц вверх
Ctrl+Shift+PgUp	Выделение вверх до конца
Home	Курсор в начало строки
Shift+Home	Выделение до начала строки
Ctrl+Home	В начало текста
Ctrl+Shift+Home	Выделить всё вверх до начала текста
End	Курсор в конец строки
Shift+End	Выделение до конца строки
Ctrl+End	В конец текста
Ctrl+Shift+End	Выделить всё вниз до конца текста
Ins	Переключение режима вставка/замена
Ctrl+Ins	Копировать
Shift+Ins	Вставить
Del	Удалить
Shift+Del	Вырезать
Backspace	Удалить последний символ
Ctrl+Backspace	Удалить последнее слово

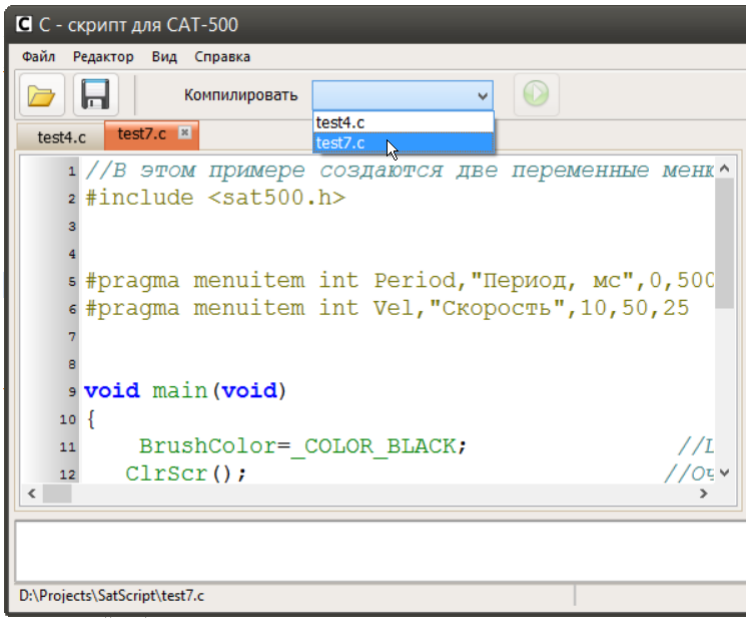
Alt+Backspace	Отменить последнее действие
Shift+Alt+Backspace	Вернуть последнее действие
Enter	Перевод строки
Tab	Символ табуляции или если есть выделение, то перемещение выделенного блока вправо
Shift+Tab	Обратная табуляция или если есть выделение, то перемещение выделенного блока вправо
Ctrl+A	Выбрать всё
Ctrl+C	Копировать
Ctrl+V	Вставить
Ctrl+X	Вырезать
Ctrl+Shift+I	Перемещение выделенного блока вправо
Ctrl+Shift+U	Перемещение выделенного блока влево
Ctrl+M	Перевод строки
Ctrl+N	Вставить строку
Ctrl+T	Удалить слово
Ctrl+Y	Удалить строку
Ctrl+Shift+Y	Удалить всё до конца строки
Ctrl+Shift+Z	Вернуть последнее действие
Ctrl+Z	Отменить последнее действие
Ctrl+Shift+L	Включить режим выделения строк
Ctrl+Shift+C	Включить режим выделения колонок
Ctrl+Shift+N	Включить обычный режим выделения символов
Ctrl+Space	Предложение кода
Alt+Right	Предложение кода
Ctrl+J	Автозавершение текста
Ctrl+/	Вставка/Удаление комментариев
Ctrl+{	Переход на парную фигурную скобку
Ctrl+}	Переход на парную фигурную скобку
F12	Поиск вниз текста на котором стоит курсор
Ctrl+F12	Поиск вверх текста на котором стоит курсор
Ctrl+F	Начать поиск текста
F3	Продолжить поиск текста вперёд
Ctrl+F3	Продолжить поиск текста назад
Ctr+R	Замена текста
Ctrl+Shift+F	Поиск в файлах
F11	Выполнять программу до текущей позиции курсора
Ctrl+'+'	Увеличить размер шрифта
Ctrl+'-'	Уменьшить размер шрифта
Ctrl+'O'	Форматирование выделенного фрагмента текста

4.4. КОМПИЛЯЦИЯ

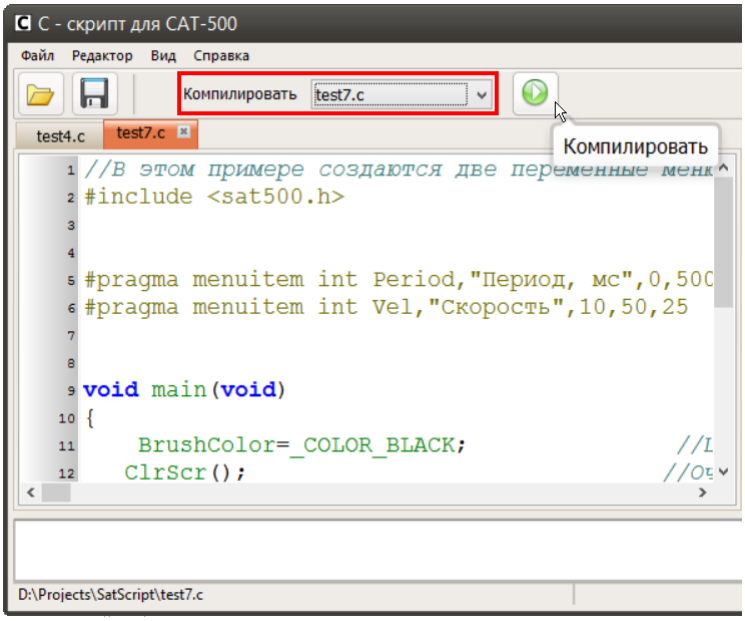
Для компиляции необходимо выполнить следующие действия:
1. Загрузить программу на языке "С-Скрипт" в редактор и внести необходимые изменения в текст программы:



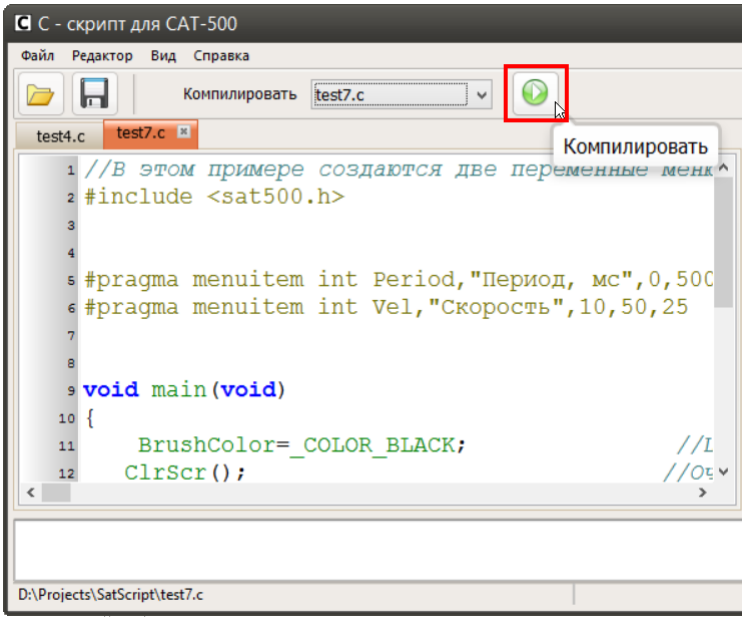
2. В поле выбора компилируемой программы задать какой из загруженных файлов должен быть откомпилирован:



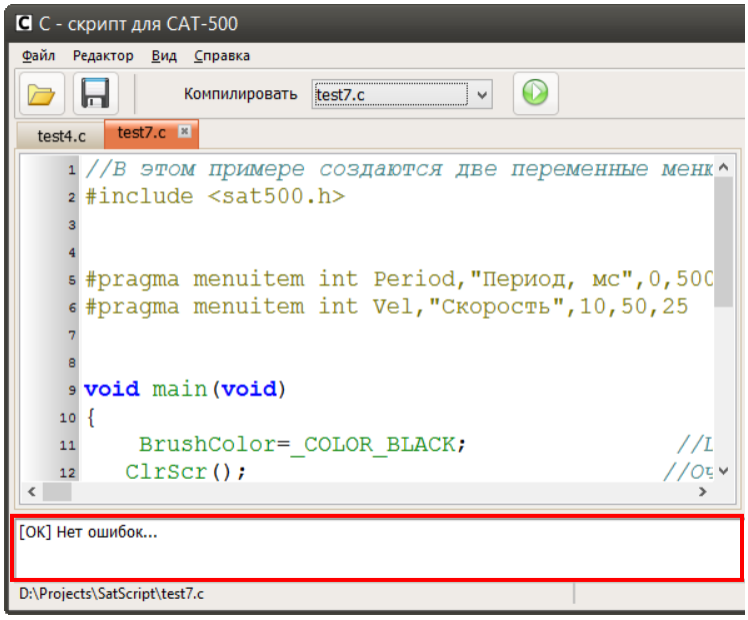
3. Имя выбранного файла появится в поле выбора:



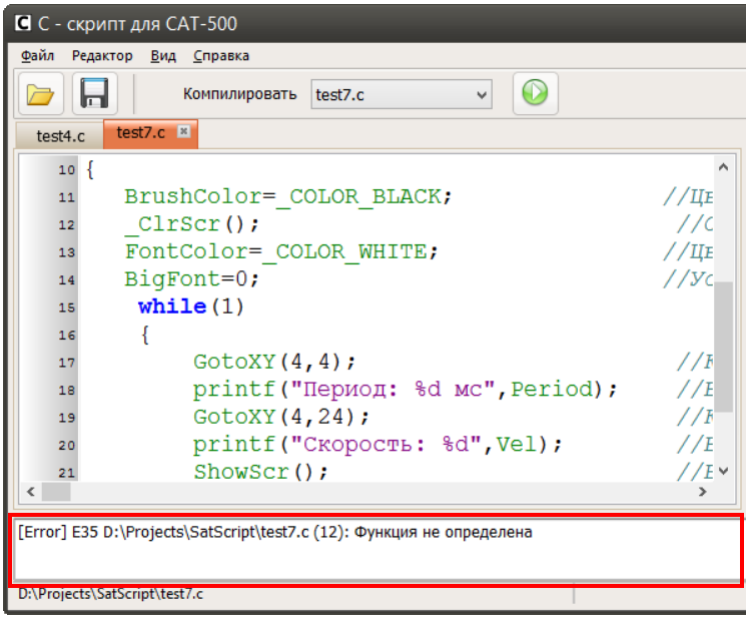
4. Нажать кнопку "*Компилировать*" или клавишу "*F2*" на клавиатуре компьютера



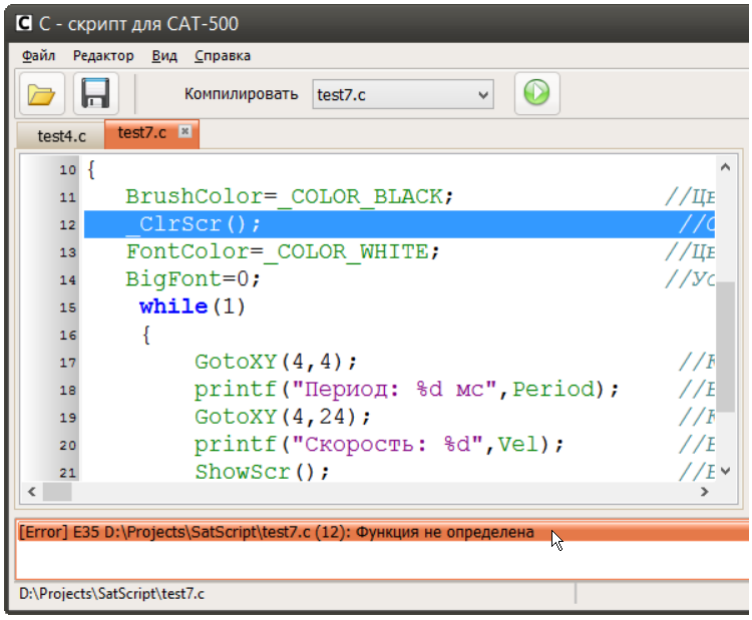
5. В нижней части окна появится сообщение об успешной компиляции "**[OK] Нет ошибок**":



Или сообщение об ошибке:

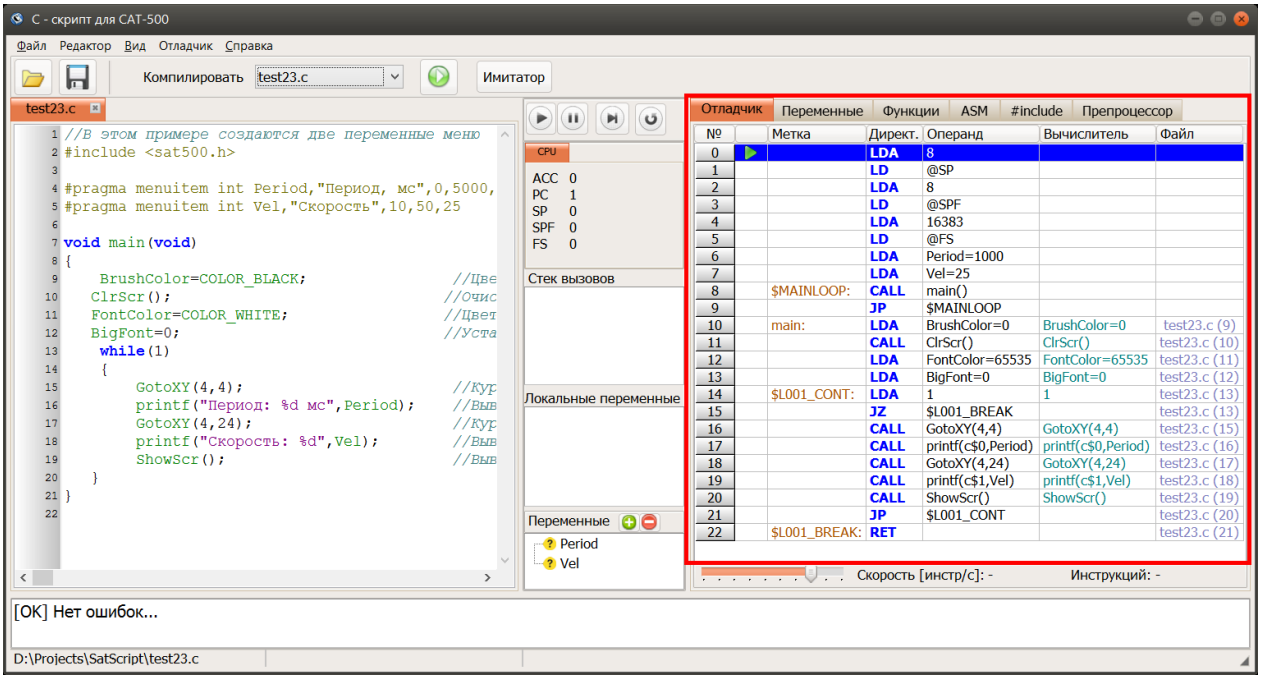


Для показа ошибки в исходном тексте необходимо выполнить двойной клик мышкой полю сообщения об ошибки. В окне редактора будет подсвечена строка, приводящая к ошибке:



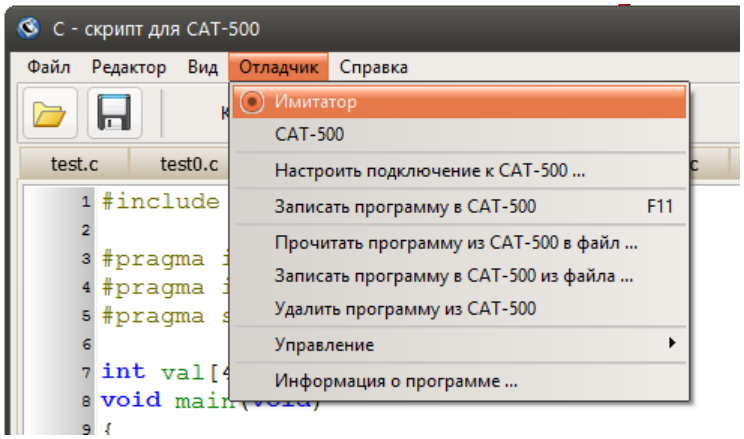
В данном случае ошибку вызывает обращение к несуществующей функции **_ClrScr**() (лишний символ подчёркивания).

6. В случае успешной компиляции (Нет ошибок) в правой части окна на вкладке "Отладчик" появится программа на языке "Псевдоассемблер", являющейся откомпилированной версией выбранной программы. Можно переходить к отладке приложения:

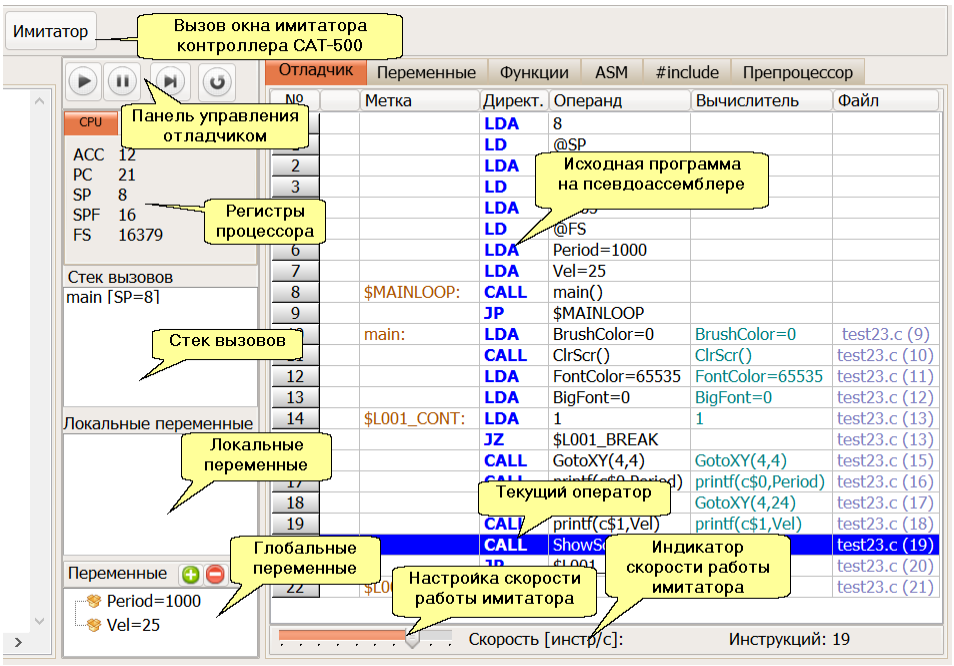


4.5. ОТЛАДКА В ИМИТАТОРЕ

Для разрешения отладки в имитаторе необходимо в основном меню программы "*Отладчик*" выбрать опцию "*Имитатор*":



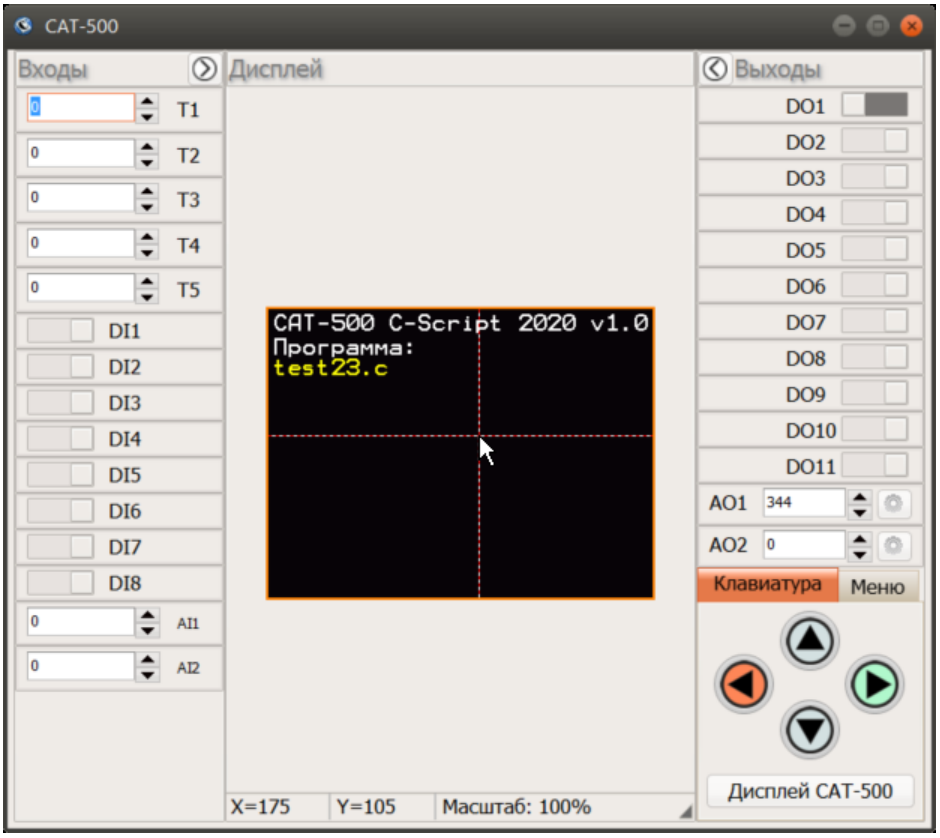
Отладка выполняется в правой части окна на вкладке "*Отладчик*".



Кнопки панели управления отладчиком:

Кнопка	Название	Описание
	"Запуск программы (F9)"	Запускает откомпилированную программу на выполнение
	"Пауза (F9)"	Приостанавливает выполнение запущенной программы
	"Шаг (F7)"	Делает один шаг выполнения программы (Выполняет один оператор псевдоассемблера)
	"Сброс (F8)"	Выполняет сброс контроллера

Последовательность действий по отладке программы:
1. Открыть окно имитатора контроллера CAT-500 нажав на кнопку "*Имитатор*":



2. Нажать кнопку "*Запуск программы (F9)*" для выполнения откомпилированной программы.
3. Проверить работу в окне имитатора, изменяя состояние входных сигналов, настроечные параметры в меню и нажимая виртуальные кнопки контроллера.
4. Если поведение контроллера не соответствует задуманному, то можно использовать пошаговое выполнение программы и точки останова.

Приостановка программы

Если программа выполняется, то можно приостановить ее работу нажав на кнопку *"Пауза (F9)"* на панели управления отладчиком или на клавишу "*F9*" на клавиатуре компьютера
Далее можно выполнить программу пошагово, используют кнопку *"Шаг (F7)"* или клавишу *"F7"* на клавиатуре компьютера? или можно повторно запустить программу.
Выполнение программы будет продолжена с места, где программа была приостановлена.
Программа также может останавливаться автоматически если, на исполняемом операторе установлена точка останова.

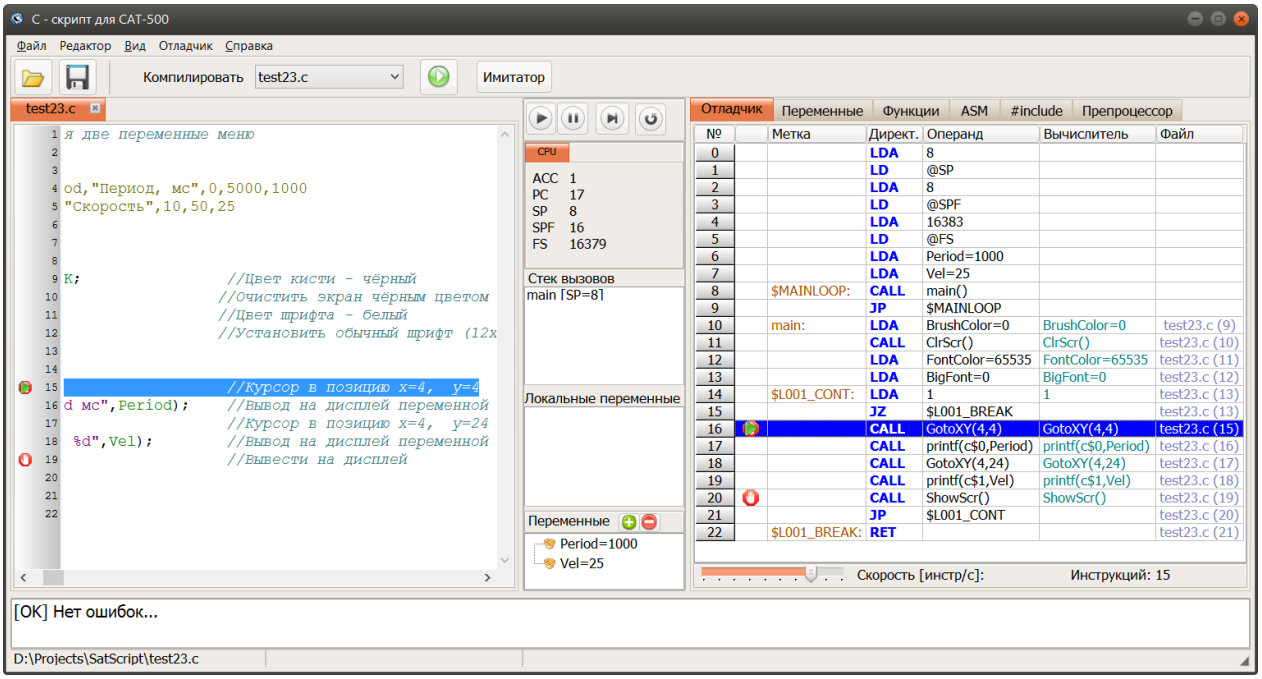
Точки останова

В любой момент времени можно поставить точку останова программы. Для этого необходимо выполнить двойной клик мышкой по любой строке на вкладке отладчик (по любому оператору псевдоассемблера). Во второй колонке появится специальный символ красного цвета, говорящей об установленной точке останова. Такой же символ появится и на вкладке исходного текста, в месте соответствующему оператору псевдоассемблера. Устанавливать точку останова также можно двойным кликом по левой информационной полосе исходного текста.

На рисунке выше двойной клик левой строчкой мышки по строке номер 20 на вкладке отладчик выполнил установку точки останова, которая так же показана на информационной полосе вкладке исходного текста (строка номер 19).
Можно установить несколько точек останова. В контроллер САТ-500 можно загрузить не более 16 точек останова.

На рисунке выше двойной клик левой строчкой мышки по информационной полосе исходного текста по строке номер 15 выполнил установку точки останова, которая так же автоматически появилась на вкладке отладчик (строка номер 16).
После запуска программы на выполнение кнопкой "*Запуск программы (F9)*" выполнение будет продолжаться до любой точки останова. Затем выполнение будет остановлено и в элементах окна от-

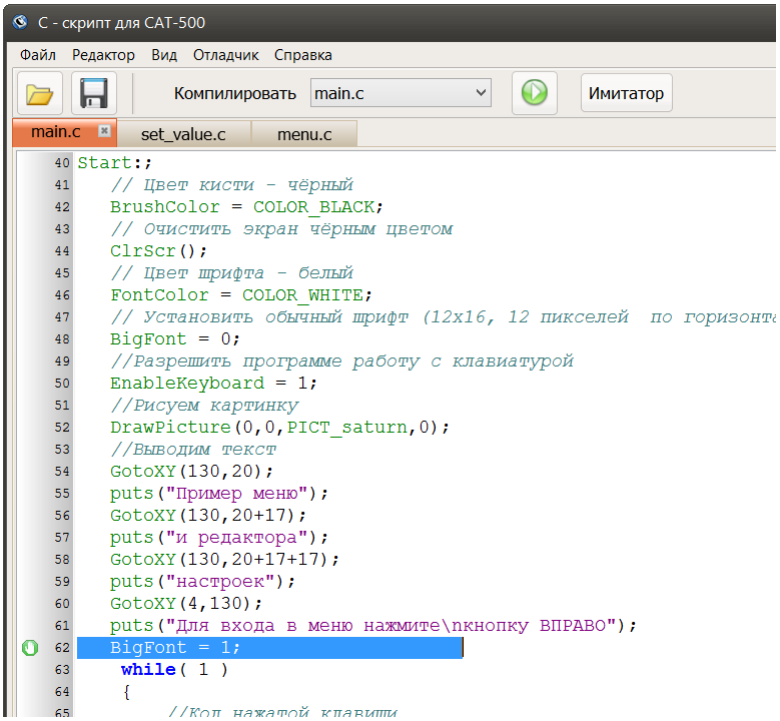
ладки будет показано состояние стека, локальные и глобальные переменные. Можно пронализировать текущее состояние программы и выполнить пошаговое выполнение, убеждаясь в правильности или неправильности работы написанного программного кода.



На рисунке выше выполнение программы остановилось по достижении точки останова в строке 17. Отладчик показывает текущую строку (ещё не выполнена) зелёным треугольником как на вкладке "*Отладчик*", так и на информационной полосе исходного текста программы.

Выполнить до курсора

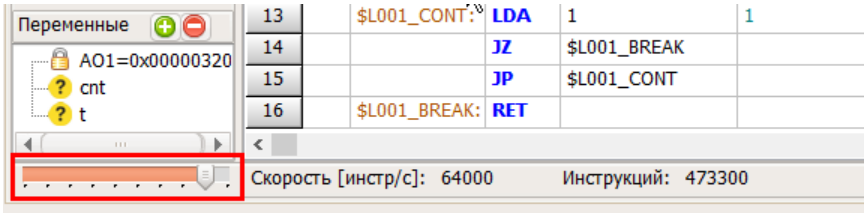
Кроме точек останова имеется ещё одно специальное действие быстрого запуска и останова программы, которое называется "*Выполнить до до курсора*".
Для использования действия *"Выполнить до до курсора"* следует установить курсор текущей позиции редактирования в любое место исходного текста программы и нажать кнопку "***F11***" на клавиатуре компьютера или выбрать специальный одноимённый пункт контекстного меню. Произойдет автоматический запуск программы на выполнение.
По достижении указанной строки исходного текста будет выполнен автоматический останов программы.
Строка, заданная для останова действием "*Выполнить до курсора*" показывается специальным зеленым маркером правой части окна редактирования:



Так же клавиша "***F11***" работает и в правой части окна на панели "Отладчик".

Скорость выполнения программы

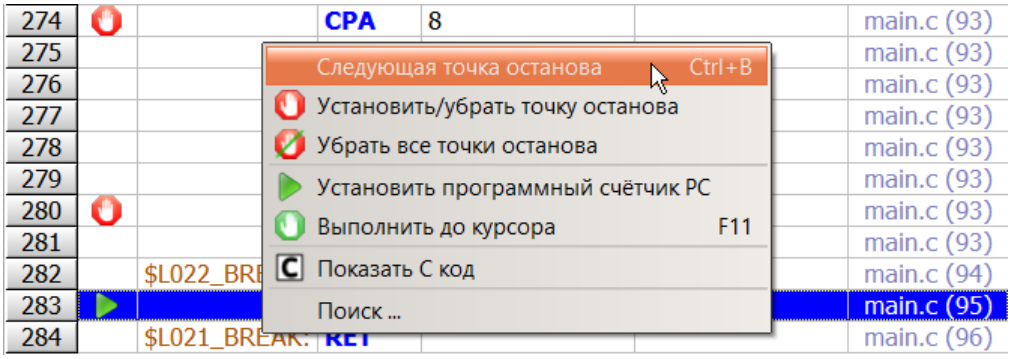
Скорость выполнения инструкций псевдоассемблера определяется специальным регулятором в нижней части вкладки "*Отладка*":



При установке большого числа инструкций в секунду может наблюдаться "притормаживание" работы интерфейса отладчика.
Рядом с регулятором скорости в статусной расположены два информационных текстовых поля:
Скорость [инстр/с] - Количество выполняемых инструкций в секунду.
Инструкций - Общее количество выполненных инструкций с момента запуска программы.

Контекстное меню отладчика

На вкладке "*Отладчик*" можно вызвать контекстное меню щелчком правой кнопкой мышки по любой строке:

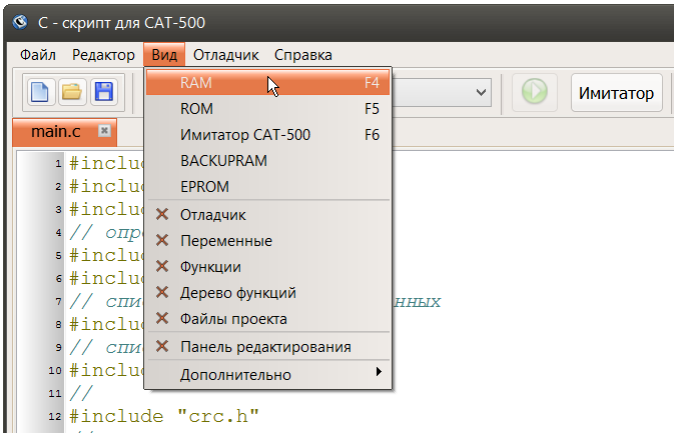


В появившемся меню доступны следующие опции:

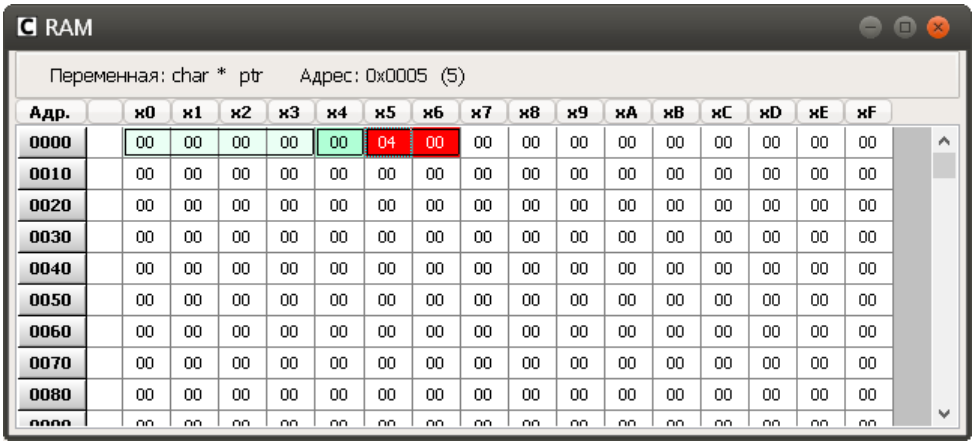
Пункт меню	***Описание***
Следующая точка останова	Перейти к следующей точке останова
Установить/Убрать точку останова	Устанавливает/Убирает точку останова на текущем операнде псевдоассемблера
Убрать все точки останова	Убираются все точки останова
Установить программный счётчик PC	Программный счётчик будет установлен на текущую инструкцию (операнд) псевдоассемблера. **ВНИМАНИЕ! При использовании этого пункта меню следует понимать, что работа С стека программы может нарушится и программа будет работать неправильно!**
Выполнить до курсора	Запустить выполнения программы до текущей выбранной строки
Показать С код	Выделяет на вкладе исходного текста программы строку, соответствующую текущему оператору псевдоассемблера
Поиск	Поиск в отладчике

4.5.1. ОКНО RAM

Отладчик содержит специальное окно, показывающее содержимое памяти контроллера (RAM). Для его показа необходимо выбрать пункт основного меню "***Вид***/**RAM**" или нажать кнопку "***F4***" на клавиатуре.



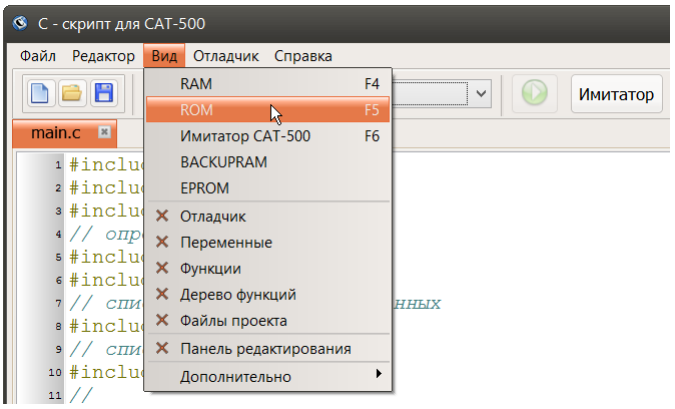
Откроется окно в котором показано текущее содержимое RAM контроллера:



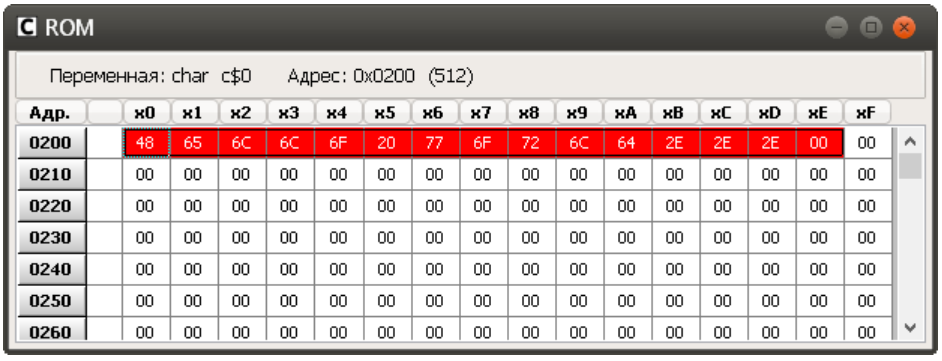
Щёлкнув по любой ячейки памяти левой кнопкой мышки можно получить дополнительную информацию по этой ячейке памяти в информационной панели в верхней части окна. Например в ячейках памяти с адресами 5 и 6 находится указатель char * ptr, который имеет значение 0x0004.

4.5.2. ОКНО ROM

Отладчик содержит специальное окно, показывающее содержимое неизменяемой памяти контроллера (ROM).
Для его показа необходимо выбрать пункт основного меню "***Вид***/**ROM**" или нажать кнопку "***F5***" на клавиатуре



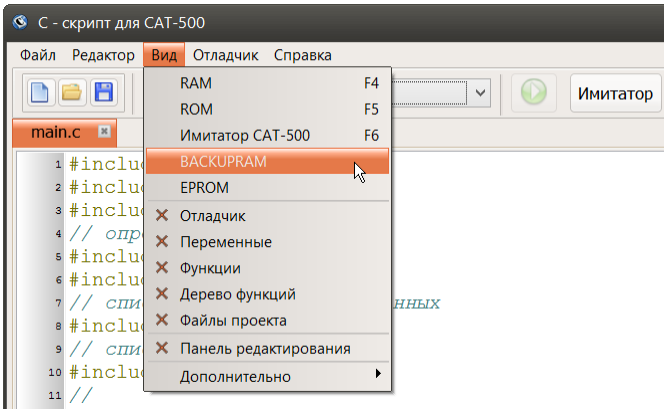
Откроется окно в котором показано текущее содержимое ROM контроллера:



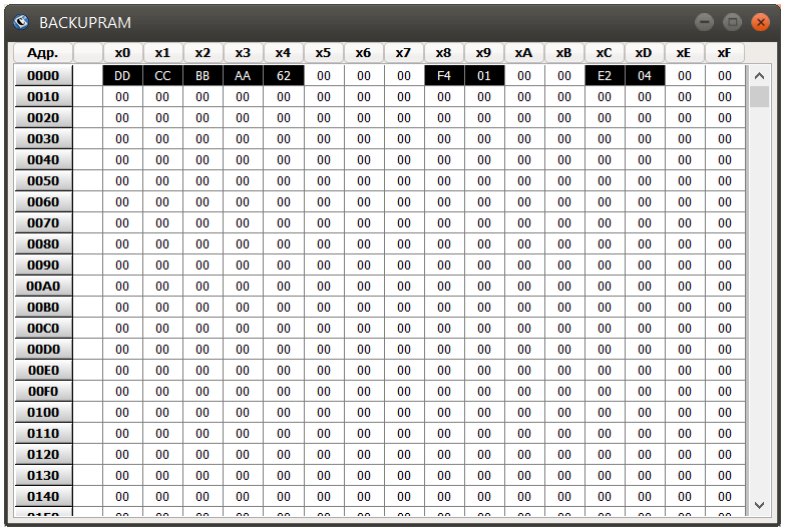
Щёлкнув по любой ячейки памяти левой кнопкой мышки можно получить дополнительную информацию по этой ячейке памяти в информационной панели в верхней части окна. Например в ячейках памяти с адресами 0..14 находится указатель *const char * c$0* на строку.

4.5.3. ОКНО BACKUPRAM

Отладчик содержит специальное окно, показывающее содержимое энергонезависимой памяти контроллера BACKUPRAM.
Для его показа необходимо выбрать пункт основного меню "***Вид***/**BACKUPRAM**".

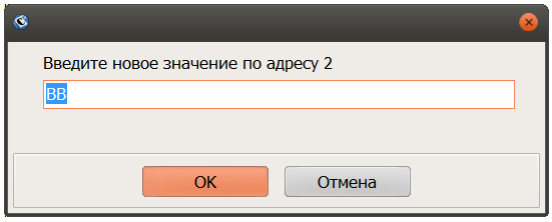


Откроется окно в котором показано текущее содержимое BACKUPRAM контроллера:

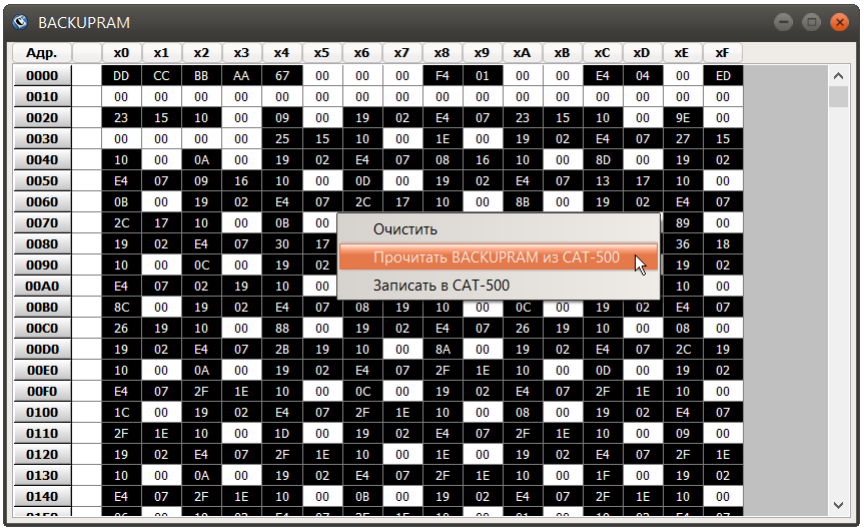


Адр.	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0000	DD	CC	BB	AA	62	00	00	00	F4	01	00	00	E2	04	00	00
0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

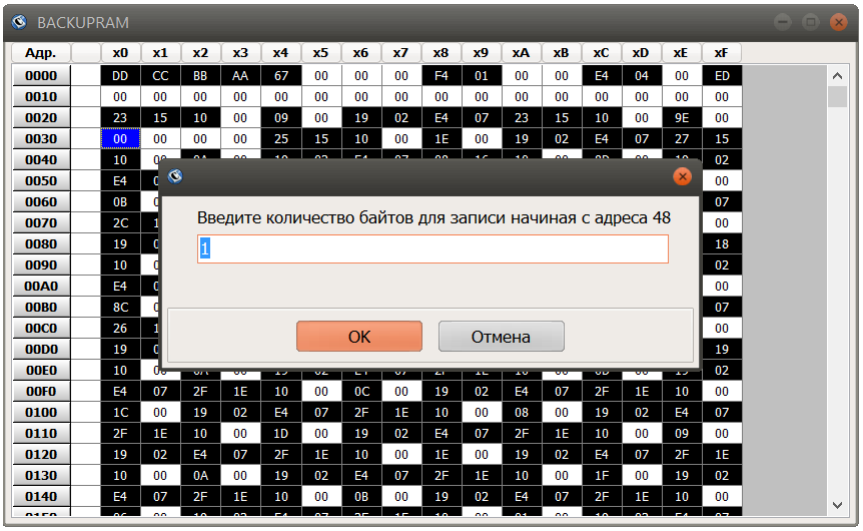
Выполнив двойной клик левой кнопкой мышкой по любой ячейки памяти можно изменить содержимое ячейки памяти BACKUPRAM.



В контекстном меню окна доступны некоторые дополнительные функции.
Здесь можно очистить всю память.
Также можно прочитать память BACKUPRAM из контроллера CAT-500, выбрав пункт меню "*Прочитать BACKUPRAM из CAT-500*":

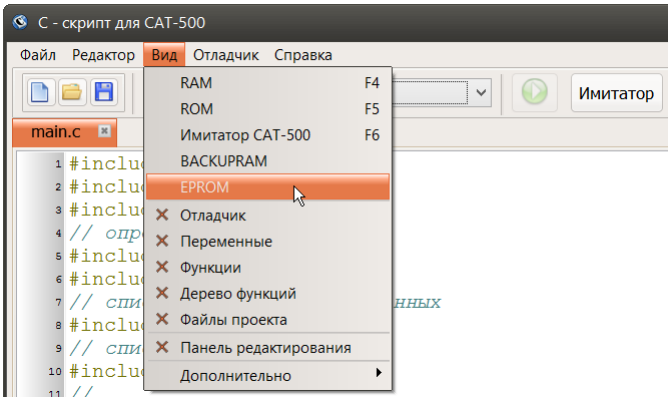


Также можно записать произвольный фрагмент памяти BACKUPRAM из окна в контроллер CAT-500.
Для этого следует установить текущий указатель адреса на начальную ячейку памяти откуда нужно выполнить запись и выбрать в контекстном меню пункт "Записать в CAT-500".
Появится дополнительное окно в котором необходимо указать количество ячеек которые будут записаны в контроллер CAT-500:



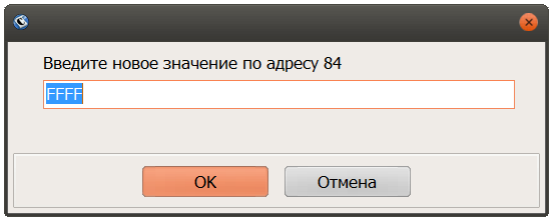
4.5.4. ОКНО EPROM

Отладчик содержит специальное окно, показывающее содержимое энергонезависимой памяти контроллера EPROM.
Для его показа необходимо выбрать пункт основного меню "***Вид***/**EPROM**".

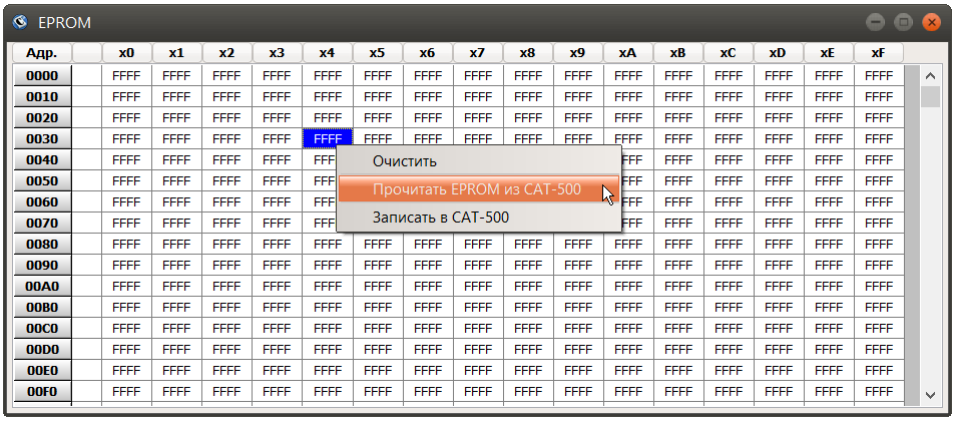


Откроется окно в котором показано текущее содержимое EPROM контроллера:

Выполнив двойной клик левой кнопкой мышкой по любой ячейки памяти можно изменить содержимое ячейки памяти EPROM.



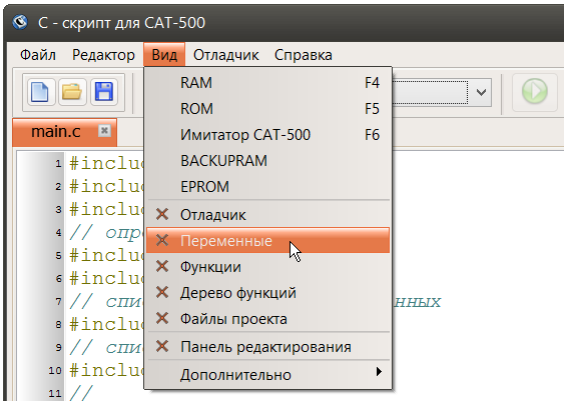
В контекстном меню окна доступны некоторые дополнительные функции.
Здесь можно очистить всю память EPROM.
Также можно прочитать память EPROM из контроллера CAT-500, выбрав пункт меню "*Прочитать EPROM из CAT-500*":



Также можно записать произвольный фрагмент памяти EPROM из окна в контроллер CAT-500.
Для этого следует установить текущий указатель адреса на начальную ячейку памяти откуда нужно выполнить запись и выбрать в контекстном меню пункт "Записать в CAT-500".
Появится дополнительное окно в котором необходимо указать количество ячеек которые будут записаны в контроллер CAT-500:

4.5.5. ВКЛАДКА "ПЕРЕМЕННЫЕ"

После компиляции программы можно посмотреть все глобальные переменные на специальной вкладке "Переменные"
Если вкладка не отображается, то включить её показ можно в основном меню программы "**Вид/Переменные**":



На вкладке отображаются переменные разных групп:
- глобальные переменные программы
- const переменные программы (расположенные в ROM)
- переменные, определённые с помощью директивы #pragma variable (аппаратные переменные)
- все переменные

Набор отображаемых групп определяется полем выбора в нижней части вкладки:

Отладчик | Переменные | Функции | Дерево функций | Файлы проекта

№	Тип	Имя	Размер	Адрес	Инициализация	Примечание
1.	char	DI	[10]	-		ID=1..10
2.	int	T	[5]	-		ID=11..15
3.	int	AI1		-		ID=16
4.	int	AI2		-		ID=17
5.	char	DO	[11]	-		ID=18..28
6.	int	AO1		-		ID=29
7.	int	AO2		-		ID=30
8.	short int	BrushColor		-		ID=31
9.	short int	FontColor		-		ID=32
10.	short int	PenColor		-		ID=33
11.	short int	PenWidth		-		ID=34
12.	char	BigFont		-		ID=35
13.	char	EnableKeyboard		-		ID=40
14.	char	SystemMenuVisible		-		ID=57
15.	unsigned char	ModbusSlave		-		ID=50
16.	unsigned int	ModbusTimeout		-		ID=51
17.	unsigned int	ModbusCount		-		ID=55
18.	unsigned short int	Value_x		0		
19.	unsigned short int	Value_y		2		
20.	unsigned int	Value_div		4		
21.	char *	Value_strName		8		
22.	char *	Value_strFormat		10		
23.	unsigned int	UINT32_min		12		
24.	unsigned int	UINT32_max		16		
25.	unsigned int	UINT32_value		20		
26.	unsigned int	UINT32_delta		24		
27.	int	INT32_min		28		
28.	int	INT32_max		32		
29.	int	INT32_value		36		
30.	int	INT32_delta		40		

Все | RAM: 459 | const: 2644 | #pragma: 17

RAM
const
#pragma
Все

На следующем рисунке показаны все переменные программы:

C - скрипт для CAT-500

Файл Редактор Вид Отладчик Справка

Компилировать test23.c | Имитатор

test23.c

```
1  оздаются две переменные меню
2  >
3
4  nt Period,"Период, мс",0,5000,1000
5  nt Vel,"Скорость",10,50,25
6
7
8
9  OR_BLACK;                    //Цвет кисти - чёрный
10                             //Очистить экран чёрным
11 _WHITE;                     //Цвет шрифта - белый
12                             //Установить обычный шри
13
14
15 );                          //Курсор в позицию x=4,
16 риод: %d мс",Period);       //Вывод на дисплей пере
17 4);                         //Курсор в позицию x=4,
18 орость: %d",Vel);           //Вывод на дисплей пере
19                             //Вывести на дисплей
20
21
22
```

CPU

ACC	25
PC	11
SP	8
SPF	16
FS	16379

Стек вызовов: main [SP=8]

Локальные переменные

Переменные: Period=1000, Vel=25

Отладчик | Переменные | Функции

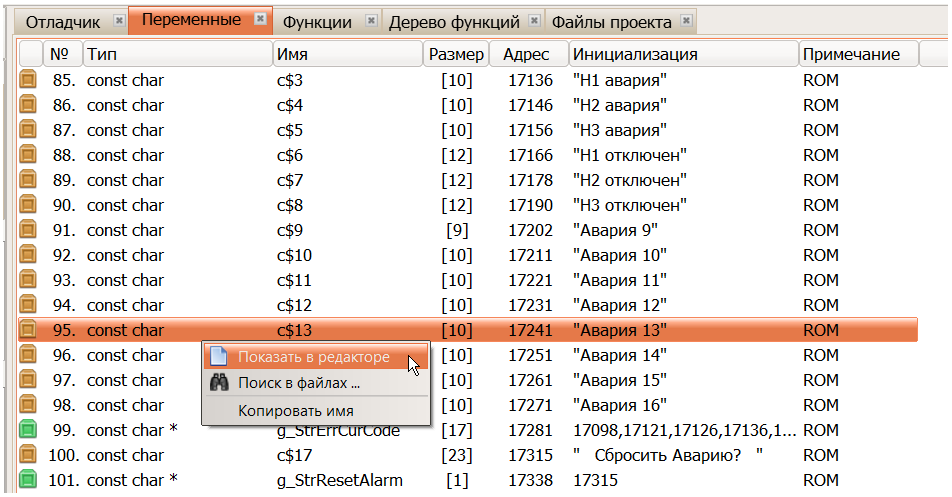
№	Тип	Имя	Раз...	Ад...	Инициализа...	Примечание
1.	char	DI	[8]	-	0,0,0,0,0,0,0,0	ID=1..8
2.	int	T	[5]	-	0,0,0,0,0	ID=9..13
3.	int	AI1		-		ID=14
4.	int	AI2		-		ID=15
5.	char	DO	[11]	-	0,0,0,0,0,0,0,0,0...	ID=16..26
6.	int	AO1		-		ID=27
7.	int	AO2		-		ID=28
8.	short int	BrushC...		-		ID=29
9.	short int	FontColor		-		ID=30
10.	short int	PenColor		-		ID=31
11.	short int	PenWidth		-		ID=32
12.	char	BigFont		-		ID=33
13.	char	EnableK...		-		ID=40
14.	int	Period		0	1000	
15.	int	Vel		4	25	
16.	const c...	c$0	[14]	16384	".....: %d .."	ROM
17.	const c...	c$1	[13]	16398	".....: %d"	ROM

Все | RAM: 8 | const: 27 | #pragma: 13

[OK] Нет ошибок...

D:\Projects\SatScript\test23.c

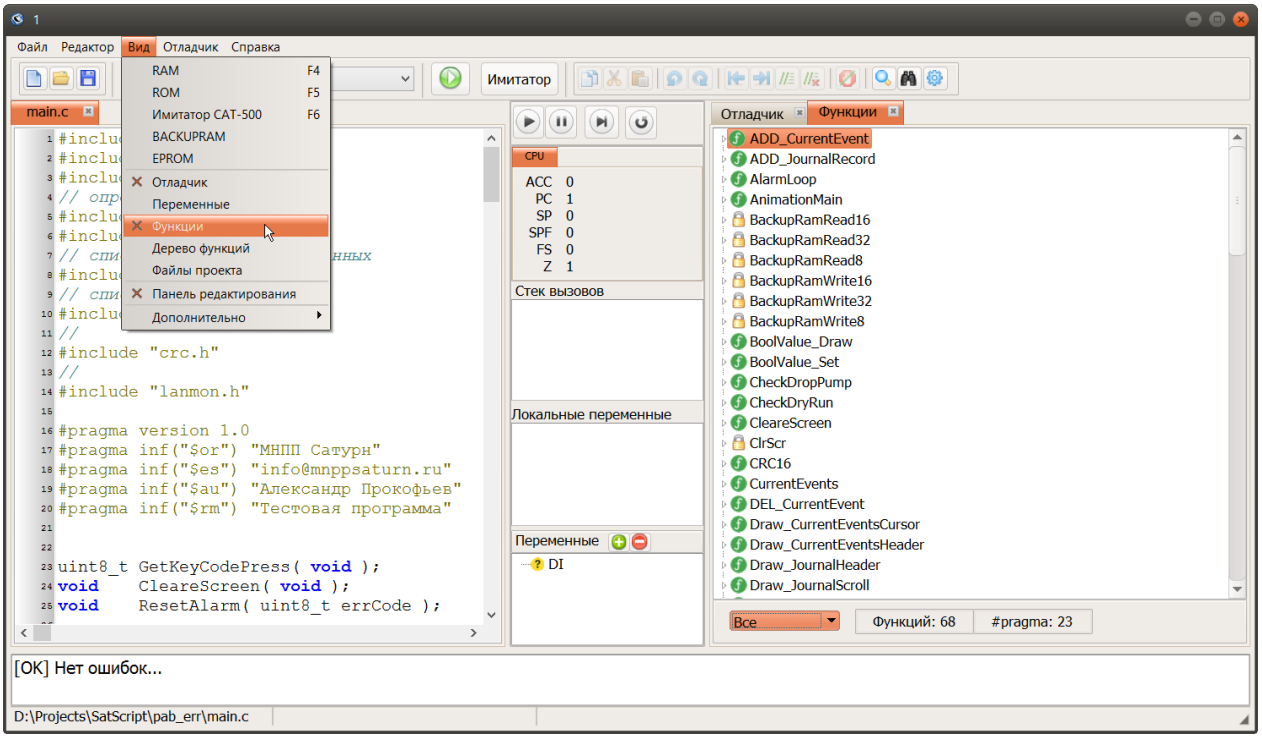
Используя контекстное меню вкладки можно выполнить некоторые действия:



Показать в редакторе	в редакторе будет показано определение выделенной переменной
Поиск в файлах	откроется окно поиска в файлах текущей выделенной переменной
Копировать имя	имя текущей выделенной переменной будет скопировано в буфер обмена

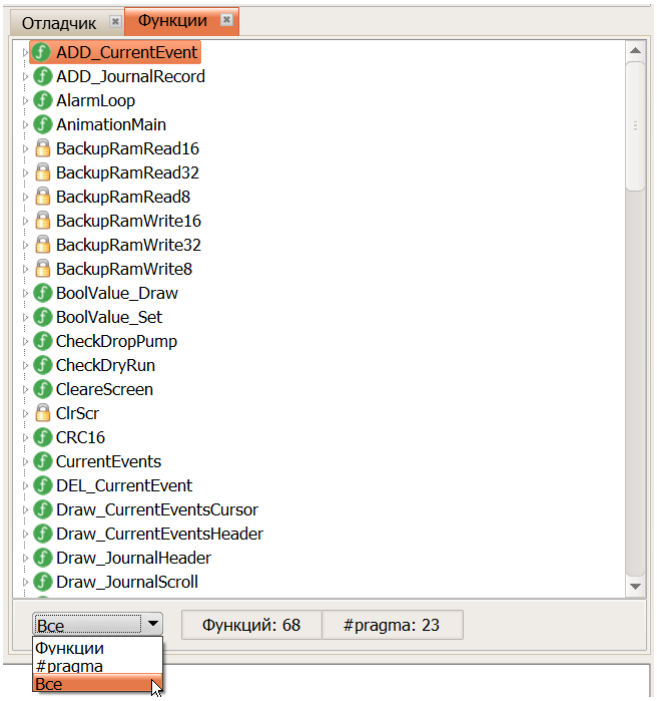
4.5.6. ВКЛАДКА "ФУНКЦИИ"

После компиляции программы можно посмотреть все функции программы на специальной вкладке "Функции"
Если вкладка не отображается, то включить её показ можно в основном меню программы "**Вид/Функции**":

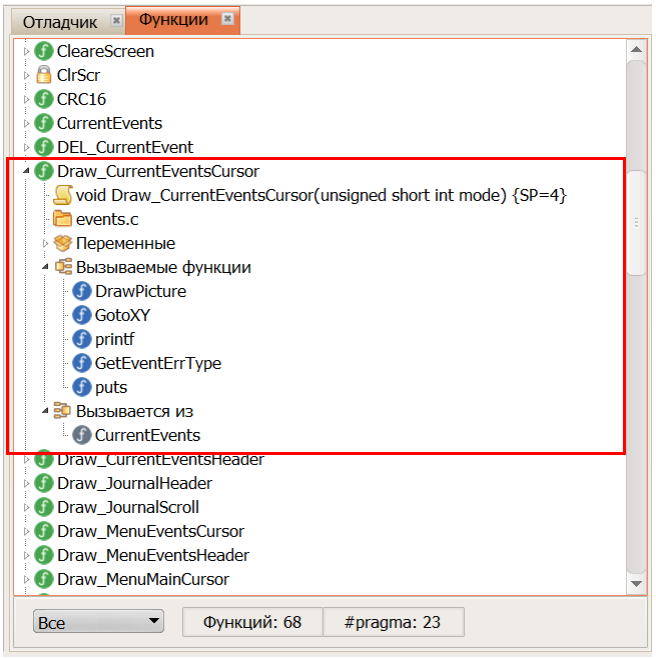


На вкладке в виде дерева в алфавитном порядке отображаются функции разных групп:
- функции , определённые программистом
- функции, определённые с помощью директивы #pragma function (аппаратные функции)
- все функции

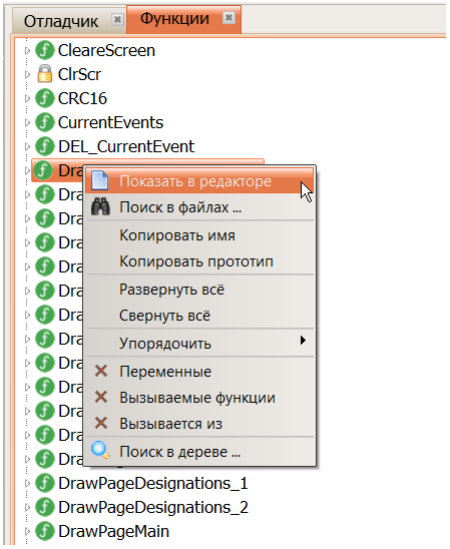
Набор отображаемых групп определяется полем выбора в нижней части вкладки:



Каждое имя функции является корневым элементом дерева.
Имя функции можно раскрыть. В этом случае в открытой функций (корневом элементе дерева) отображаются дочерние ветки:
- прототип функции
- имя файла где функция определена
- локальные переменные функции
- вызываемые функции
- функции, откуда вызывается данная функция

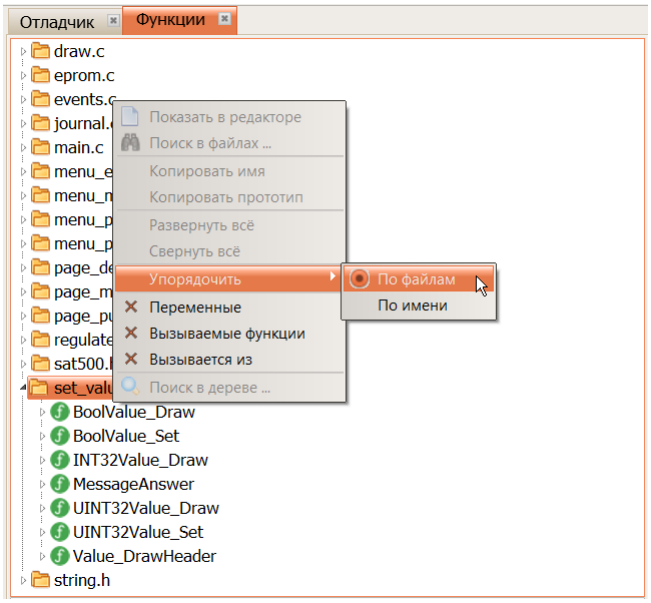


Используя контекстное меню вкладки можно выполнить дополнительные действия с текущей выделенной функцией:



Показать в редакторе	в редакторе будет показано определение выделенной функции
Поиск в файлах	откроется окно поиска в файлах текущей выделенной функции
Копировать имя	имя текущей выделенной функции будет скопировано в буфер обмена
Развернуть всё	все элементы дерева будут развернуты
Свернуть всё	все элементы дерева будет будут свернуты
Упорядочить	упорядочить функции по файлам, в которых они определены
Переменные	включить отображение в дереве переменных функций
Вызываемые функции	включить отображение в дереве вызываемых функций
Вызывается из	включить отображение в дереве функций, из которых вызывается данная функция

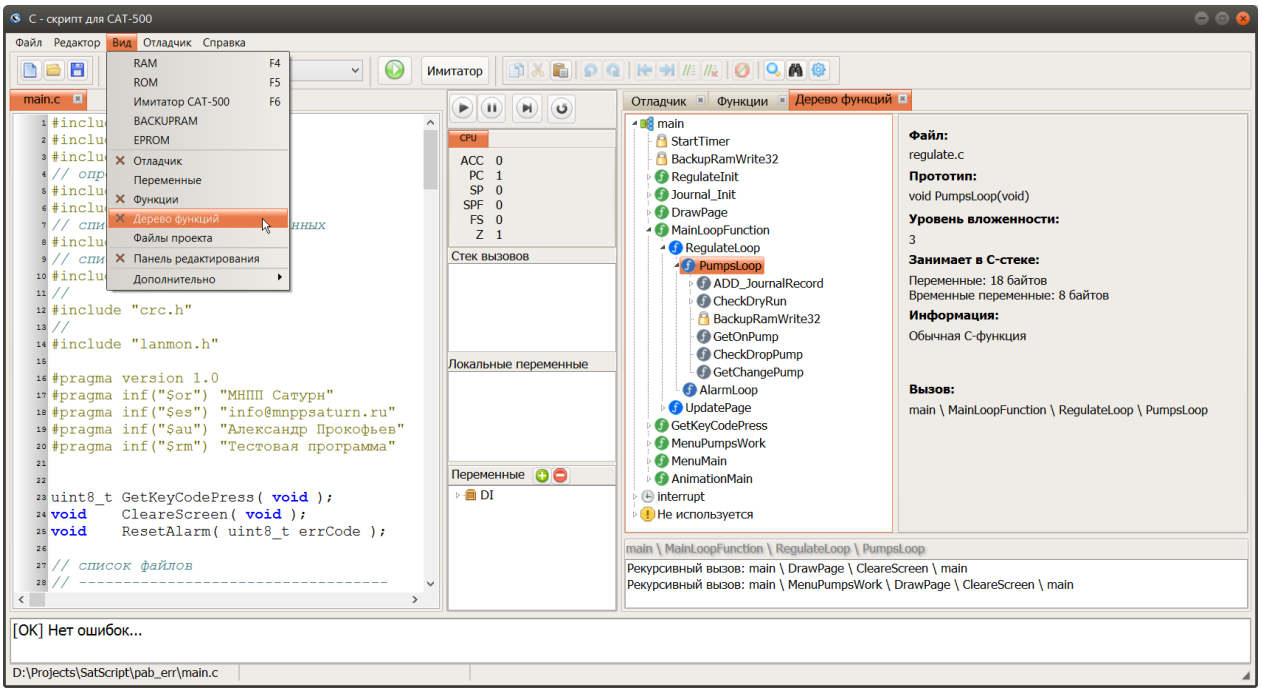
В случае выбора "Упорядочить/По файлам" вид вкладки изменяется - корневыми элементами становятся имена файлов в которых функции определяются:



В режиме "Упорядочить/По файлам" всегда отображаются все функции.

4.5.7. ВКЛАДКА "ДЕРЕВО ФУНКЦИЙ"

После компиляции программы можно посмотреть все функции программы в виде дерева вызовов, в котором отображается взаимная связь между функциями.
Эта информация отображается на специальной вкладке "Дерево функций"
Если вкладка не отображается, то включить её показ можно в основном меню программы "**Вид/Дерево функций**":

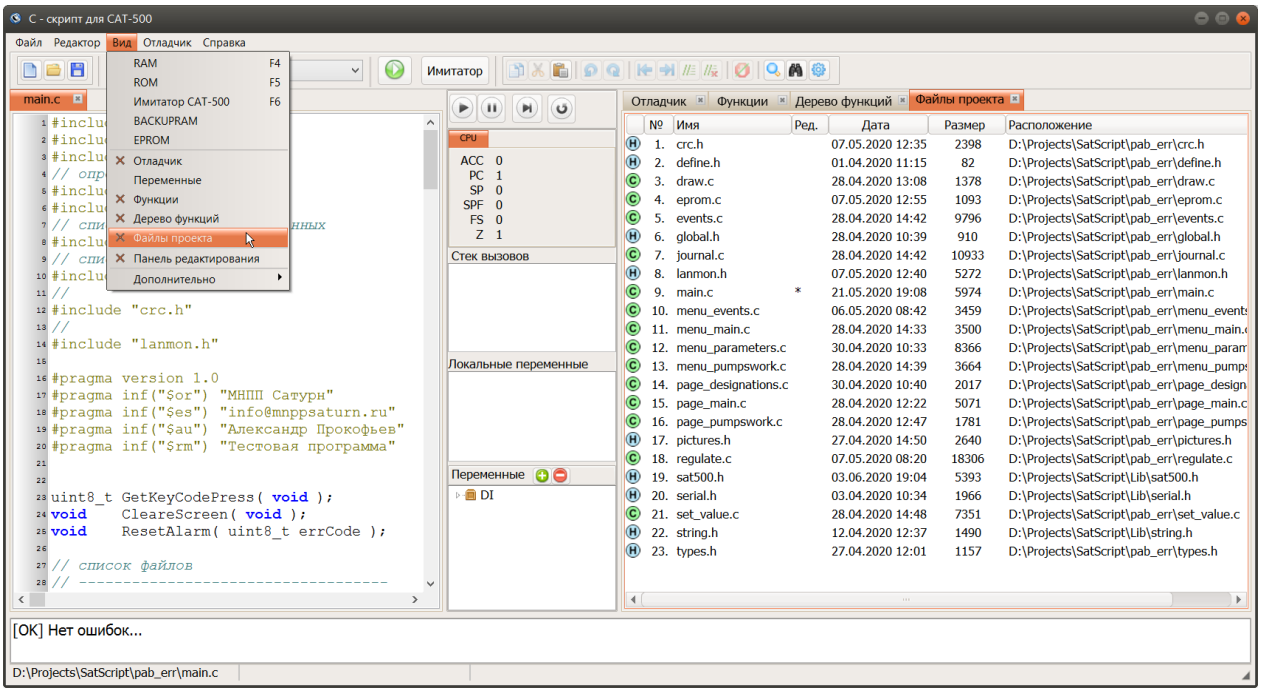


На вкладке отображаются три основных корневых элемента дерева:
main - функция main
Interrupt - функции, определённые с ключевым словом interrupt - эти функции не вызываются явно.

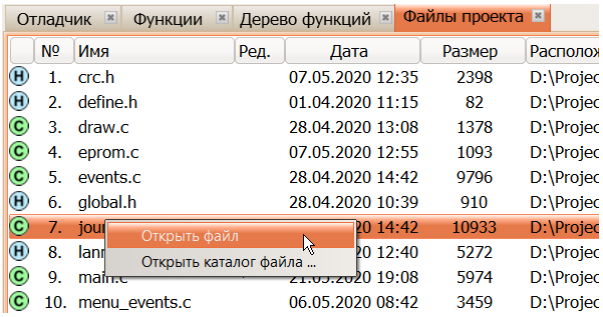
Не используется - отображаются функции, которой не вызываются в программе и соответственно могут быть удалены. Компилятор удаление функции не выполняет. Функции может удалить только программист.
Раскрыв корневой элемент *main* можно увидеть список всех функций, которые вызываются из функции *main.*
В свою очередь раскрыв любую функцию, можно увидеть список функций которые вызываются из этой функции и так далее.
Вся информация о вызовах функций представляется в виде дерева.
В правой части вкладки располагается специальное информационное окно, на котором можно просмотреть сведения о выделенной функции в дереве.
В нижней части вкладки имеется список функций, которые вызываются рекурсивно.
Рекомендуется внимательно ознакомиться с этими функциями и принять решение является ли эти вызовы функций ошибкой.

4.5.8. ВКЛАДКА "ФАЙЛЫ ПРОЕКТА"

На вкладке «Файлы проекта» отображается список всех файлов участвующих в компиляции.
Открыть вкладку можно выбрав в основном меню пункт «Вид/Файлы проекта».

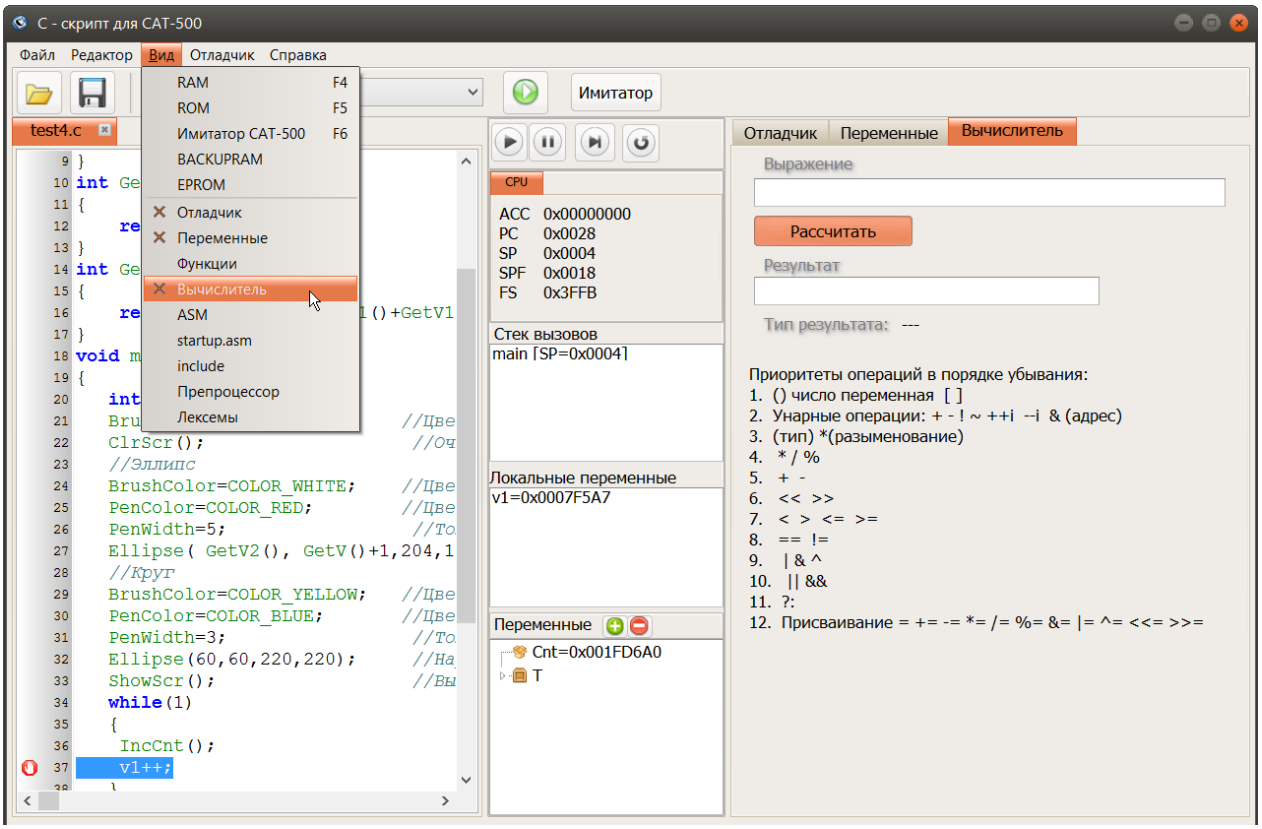


Файлы на вкладке отображаются в алфавитном порядке. Данный список формируется автоматически средой разработки.
Двойной клик мышкой по любому имени файла приводит к открытию этого файла.
Также текущий выбранный файл можно открыть из контекстного меню.
Кроме этого в контекстном меню имеется пункт "Открыть каталог файла", позволяющий открыть папку содержащую данный файл.

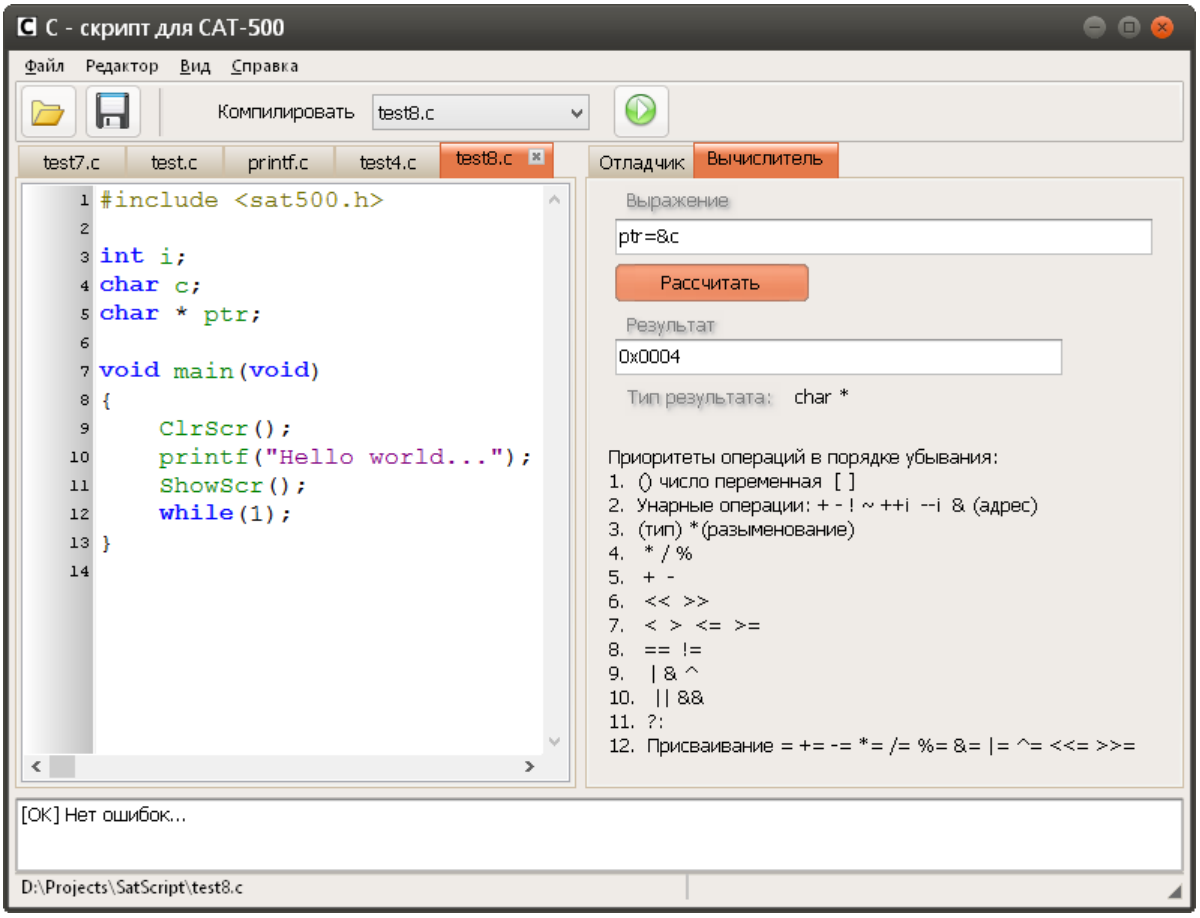


4.5.9. ВКЛАДКА "ВЫЧИСЛИТЕЛЬ"

При написании программы для CAT-500 может оказаться полезной вкладка "*Вычислитель*". Если вкладка не отображается, то включить её показ можно в основном меню программы "**Вид/Вычислитель**":



После того как программа откомпилирована отладчик "знает" все глобальные переменные. На вкладке "Вычислитель" можно набрать любое арифметическое или логическое выражение языка С-скрипт с использованием глобальных переменных и числовых констант и получить результат.



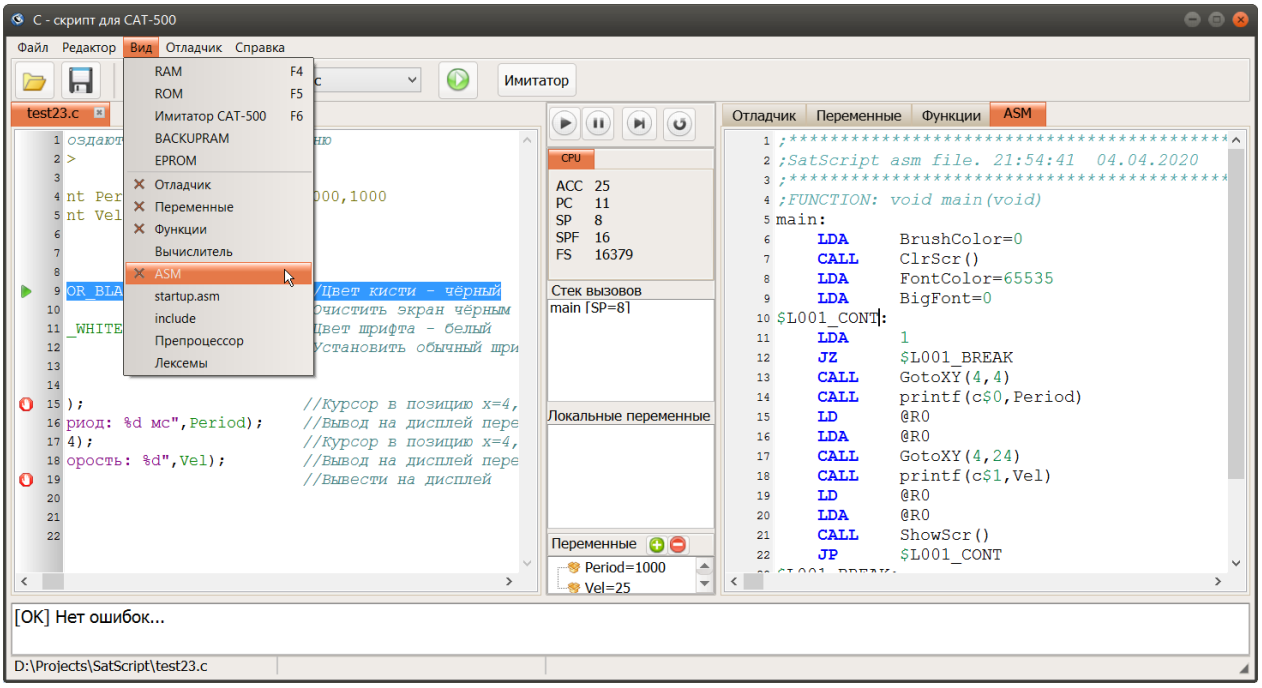
На рисунке выше откомпилирована программа "test8.c" и на вкладке "Вычислитель" задано выражение:

```
ptr=&c   //Присвоить адрес переменной с указателю ptr.
```

После нажатия кнопки "*Рассчитать*" выражение будет выполнено и в поле "*Результат*" появится значение выражения. Как известно в языке С присваивание имеет результат - именно его мы и видим - это значение "0x0004" типа char * (показано в поле "*Тип результата*")
Вычислитель может быть полезным программисту для определения правильности набираемого выражения ещё до окончательной компиляции программы.

4.5.10. ДОПОЛНИТЕЛЬНЫЕ ВКЛАДКИ

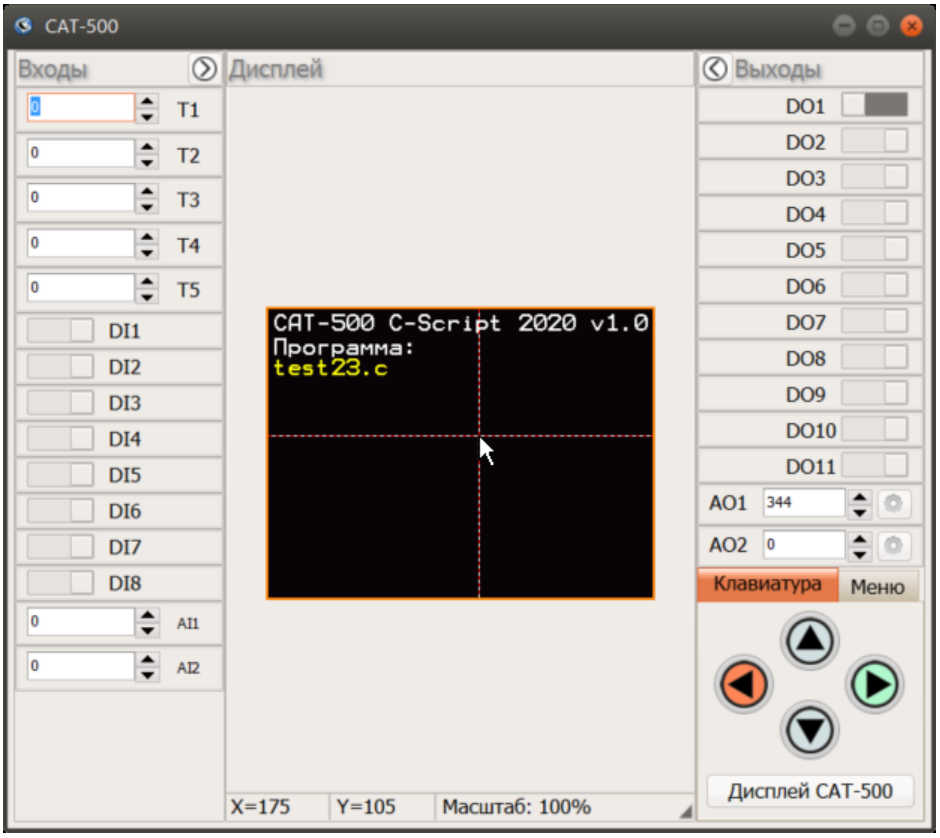
В правой части основного окна программы можно открыть несколько дополнительных вкладок, доступных через основное меню "*Вид*":



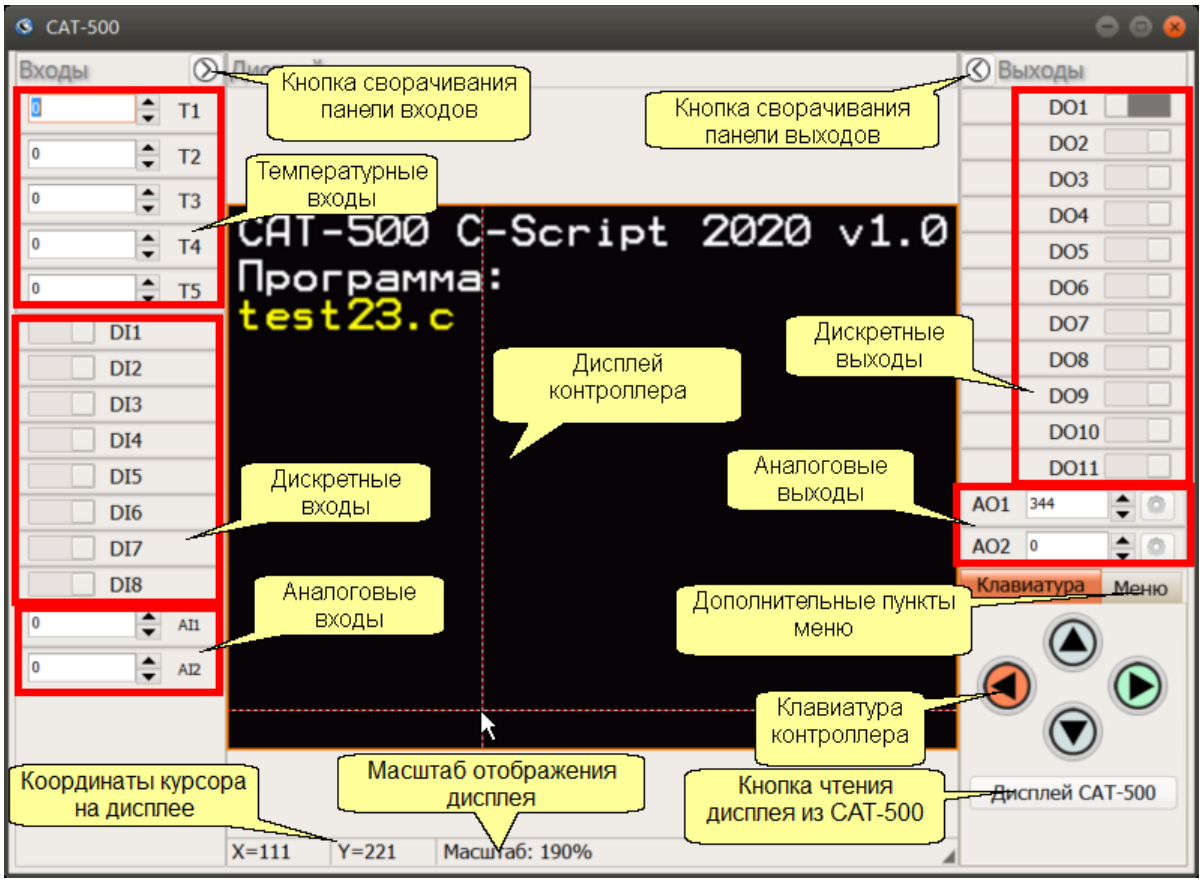
Эти вкладки предназначены для разработчиков компилятора "С-скрипт" и в окончательной версии программы будут отсутствовать.

4.6. ИМИТАТОР CAT-500

В процессе отладки приложения можно задавать состояние входов, контролировать состояние выходных сигналов и наблюдать состояние дисплея при помощи имитатора CAT-500.
Имитатора CAT-500 - это специальное окно программы для отладки программы без оборудования:



Рассмотрим более подробно фрагменты окна имитатора:



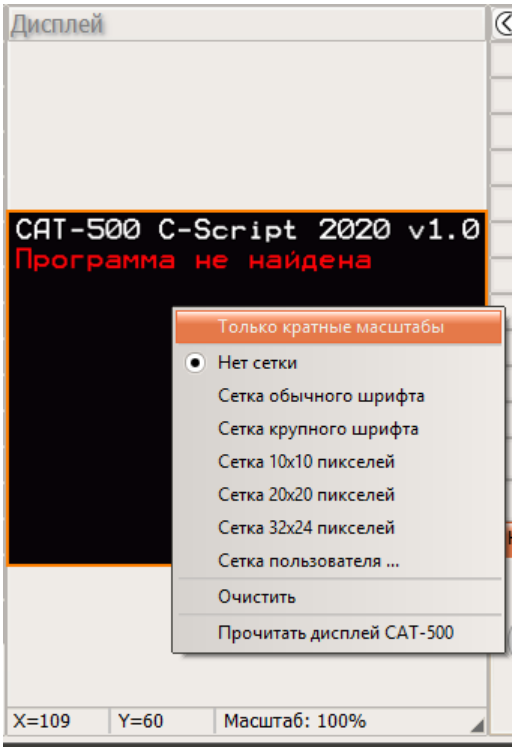
В левой части окна на панели входов расположены:

Элемент панели входов	***Описание***
Кнопка сворачивания панели входов	Убирает панель "Входы" для освобождения места на экране, если панель не требуется для отладки. В момент сворачивания на панели "Дисплей" в левой части появляется новая кнопка разворачивания панели входов
Температурные входы	Содержит пять элементов задания значений датчиков температуры T1...T5. Температура задаётся в сотых долях градуса. Значение 2000 соответствует 20 градусам Цельсия.
Дискретные входы	Содержит восемь элементов задания состояния дискретных входов DI1...DI8
Аналоговые входы	Содержит элементы задания значений аналоговых входов AI1 и AI2

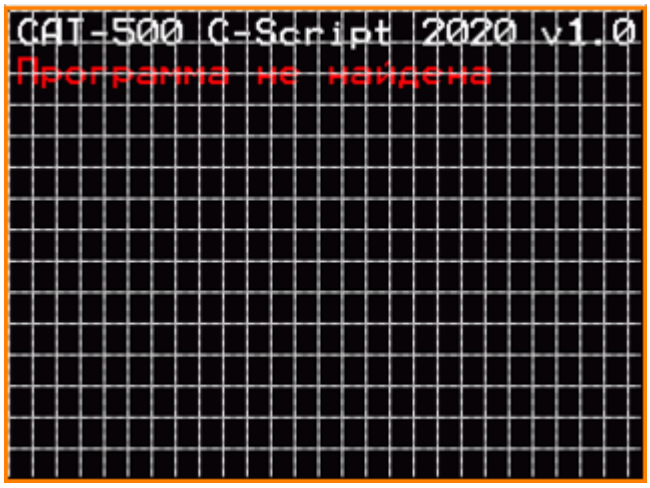
В правой части окна на панели выходов расположены:

Элемент панели выходов	***Описание***
Кнопка сворачивания панели выходов	Убирает панель "Выходы" для освобождения места на экране, если панель не требуется для отладки. В момент сворачивания на панели "Дисплей" в правой части появляется новая кнопка разворачивания панели выходов
Дискретные выходы	Содержит одиннадцать элементов просмотра состояния дискретных выходов DO1...DO11
Аналоговые выходы	Содержит элементы контроля значений аналоговых выходов AO1 и AO2. Диапазон выдаваемых значений AO1 и AO2: 0...1000
Клавиатура	Четыре виртуальные кнопки, соответствующие кнопкам контроллера CAT-500. Нажатие левой кнопкой мышки на виртуальную кнопку эквивалентно нажатию на кнопку контроллера CAT-500
Меню	Содержит элементы для работы с дополнительными пунктами меню, созданными програмно с помощью директивы #pragma menuitem
Кнопка чтения дисплея из CAT-500	Нажав на данную кнопку можно прочитать дисплей контроллера CAT-500 в дисплей имитатора

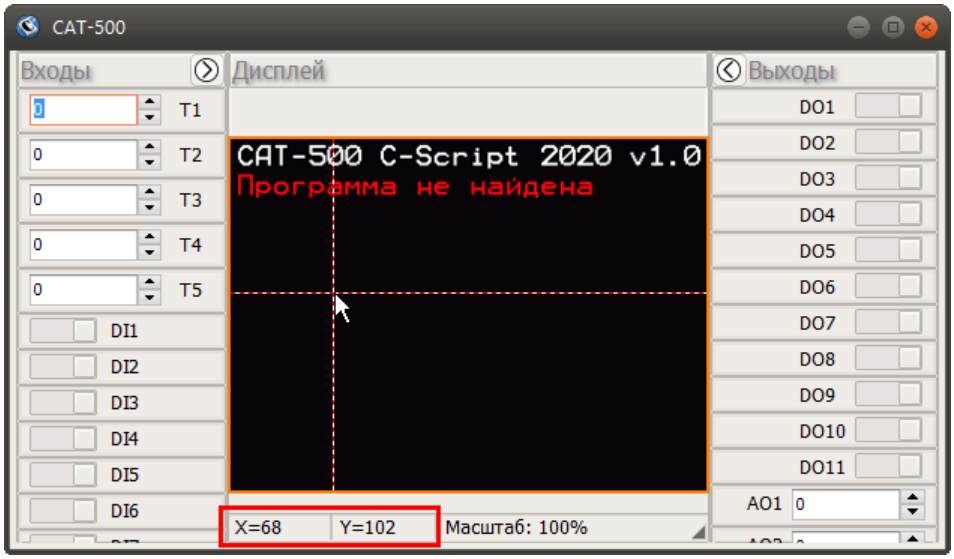
В центральной части окна расположена панель виртуального дисплея контроллера. Дисплей может быть настроен с помощью контекстного меню (меню, вызываемое правой кнопкой мышки).



В меню можно установить переключатель "Только кратные масштабы". В этом случае масштаб дисплей будет изменять размер только кратно целому числу оригинального дисплея контроллера CAT-500 (320x240).
Также можно отобразить сетку, по которой удобно определять позиции знакомест символов обоих шрифтов.
На рисунке ниже показана сетка обычного шрифта.

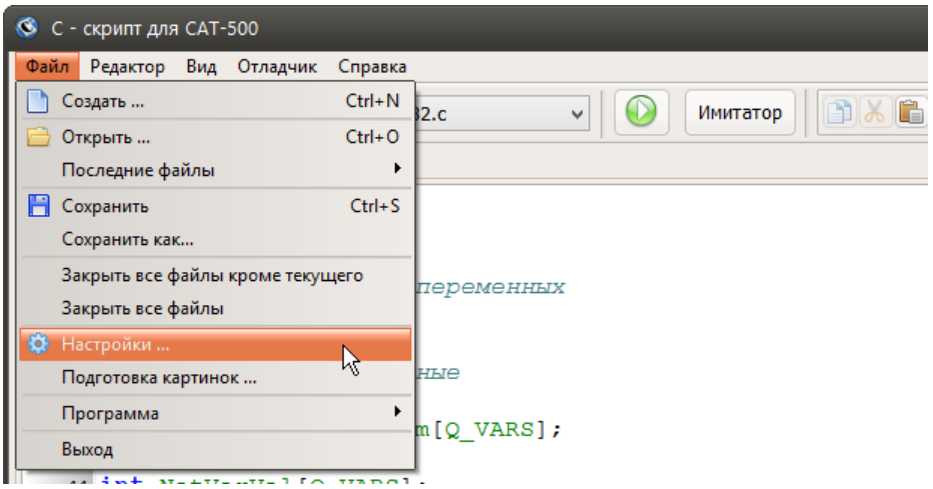


При перемещении указателя мышки по дисплею в статусной строке показываются текущие координаты курсора мышки:



4.7. НАСТРОЙКА ПРОГРАММЫ

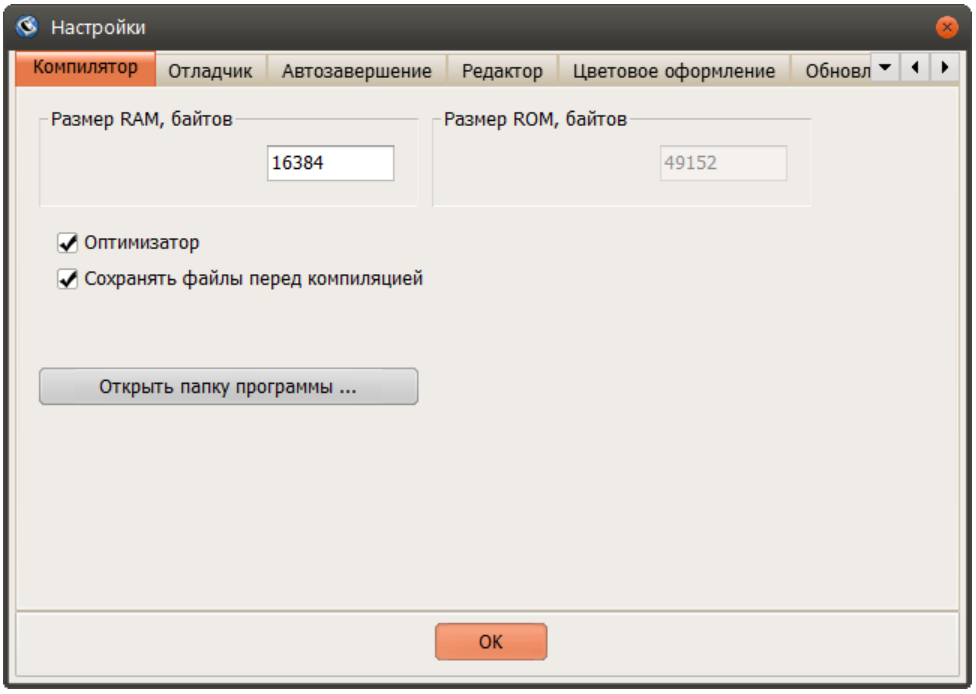
Окно настройки программы можно вызвать, выбрав пункт основного меню "**Файл/Настройки...**"



Окно настройки содержит несколько вкладок на которых можно выполнить некоторые настройки функциональности программы:

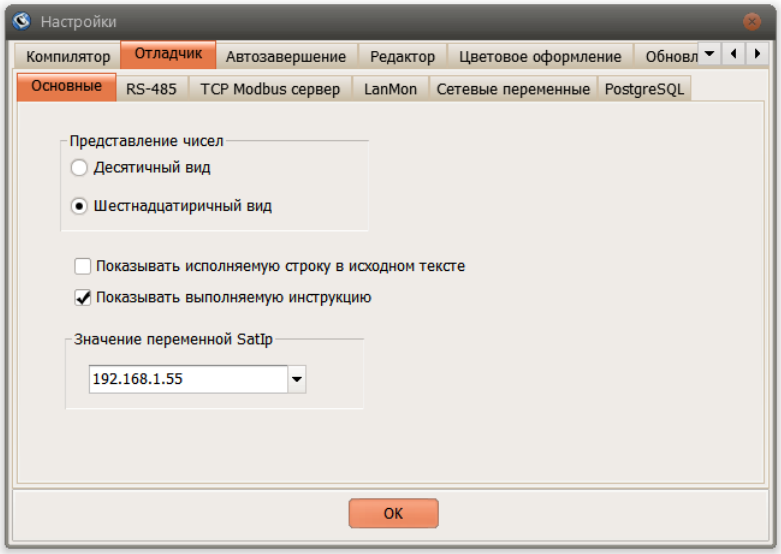
Вкладка "Компилятор"

На вкладке настраивается компилятор - можно задать необходимый размер памяти (ОЗУ), а так же включить оптимизатор и разрешить автоматическое сохранение изменённых файлов исходных текстов программы.

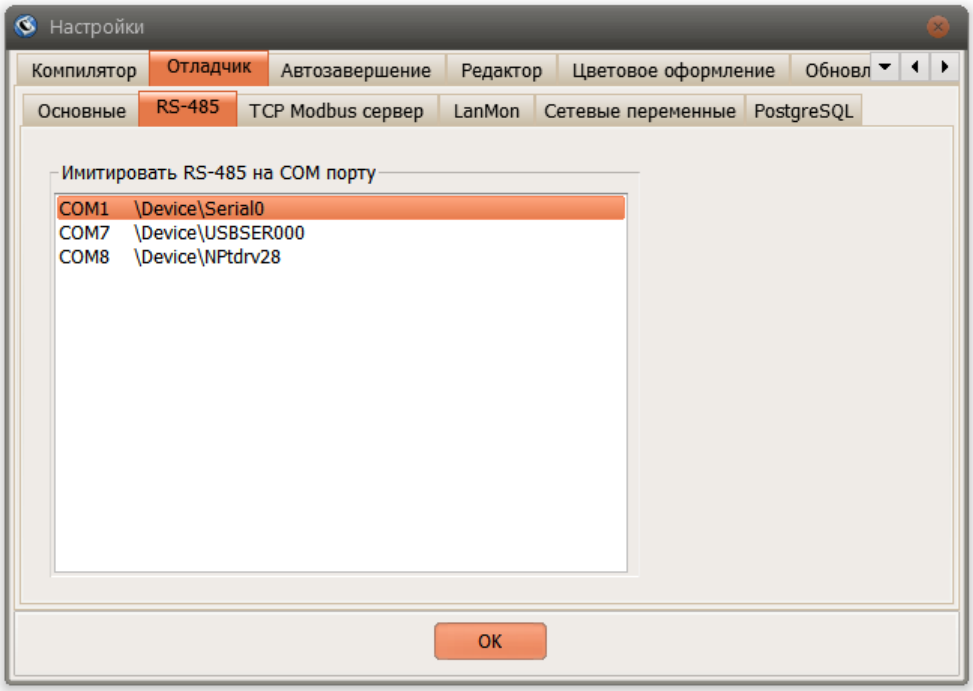


Вкладка "Отладчик"

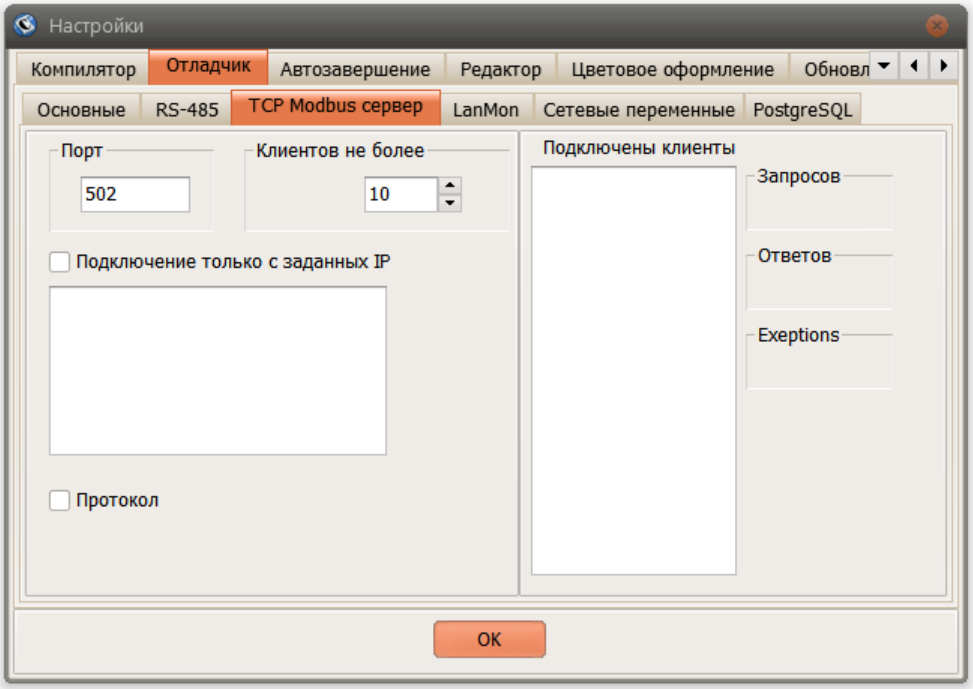
На подвкладке "Основные" настраивается вид числовых данных в отладчике. Также настраивается отображение отладки в панели редактора и панели отладчика.



В поле выбора "Значение переменной SatIp" можно выбрать IP адрес, который будет считаться IP адресом имитируемого в среде разработки контроллера CAT-500.
На подвкладке "RS-485" настраивается какой COM порт компьютера будет использоваться для имитации последовательного интерфейса контроллера CAT-500.



На подвкладке "TCP Modbus сервер" настраивается имитация TCP Modbus сервера контроллера CAT-500.

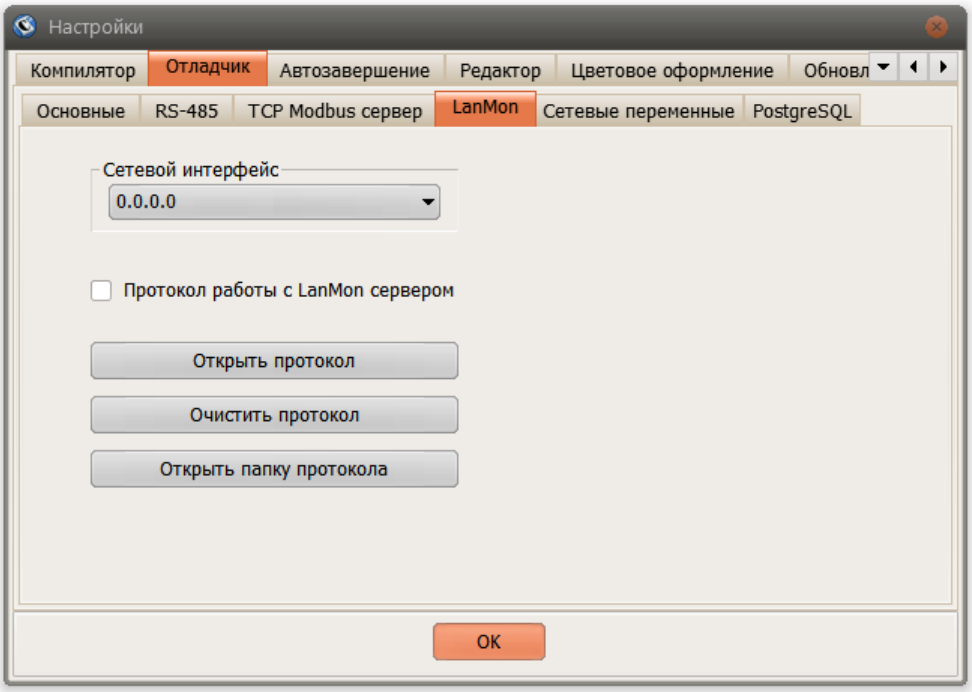


Основные управляющие элементы интерфейса:

Элемент интерфейса	***Описание***
Порт	Сетевой порт на котором работает TCP Modbus сервер
Клиентов не более	Максимальное разрешённое количество клиентов TCP Modbus сервера

Подключение только с заданных IP	Включение режима работы только с клиентами, заданными в списке адресов. Список адресов расположен ниже. Добавление/удаление адресов в список выполняется через контекстное меню.
Подключены клиенты	Перечень клиентов, подключённых в данный момент

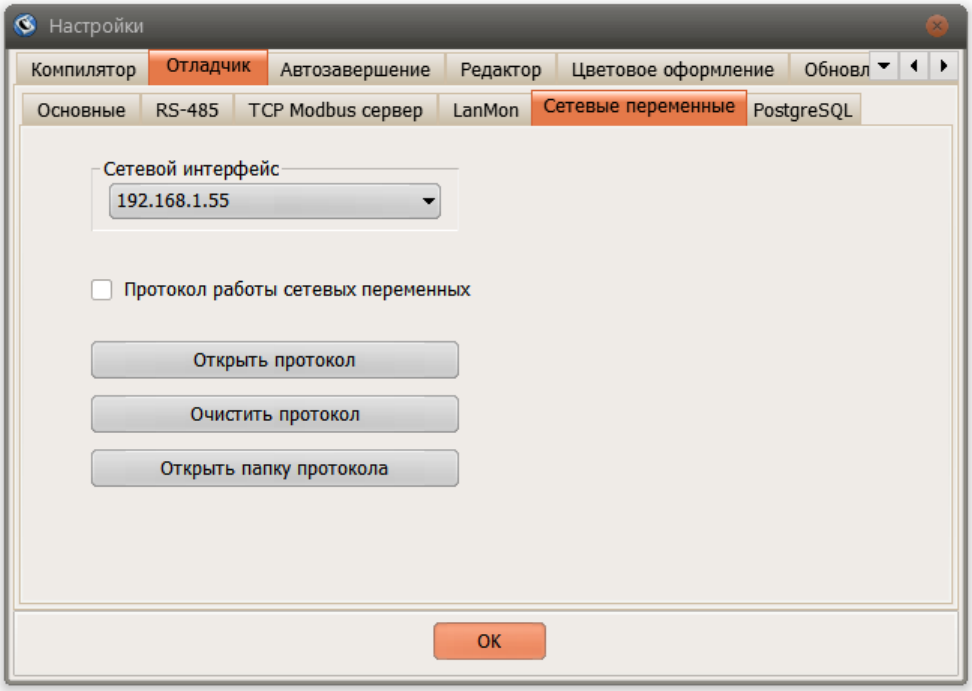
На подвкладке "LanMon" настраивается сетевой интерфейс, через которые клиент сервера LanMon, имитируемый средой разработки С-Скрипт работает с сервером, а также можно включить и просмотреть текстовый протокол работы клиента сервера LanMon.



Основные управляющие элементы интерфейса:

Элемент интерфейса	***Описание***
Сетевой интерфейс	Позволяет выбрать сетевой интерфейс компьютера через который работает клиент сервера LanMon, встроенный в среду разработки С-Скрипт
Протокол работы с LanMon сервером	Переключатель позволяет разрешить или запретить текстовый протокол работы с сервером LanMon. Установленный переключатель разрешает ведение протокола
Открыть протокол	По нажатию на данную кнопку открывается окно, в котором можно просмотреть текстовый протокол работы клиента LanMon
Очистить протокол	По нажатию на данную кнопку выполняется очистка текстового протокола работы клиента сервера LanMon
Открыть папку протокола	По нажатию на данную кнопку открывается папка, в которой расположен текстовый файл протокола работы клиента сервера LanMon

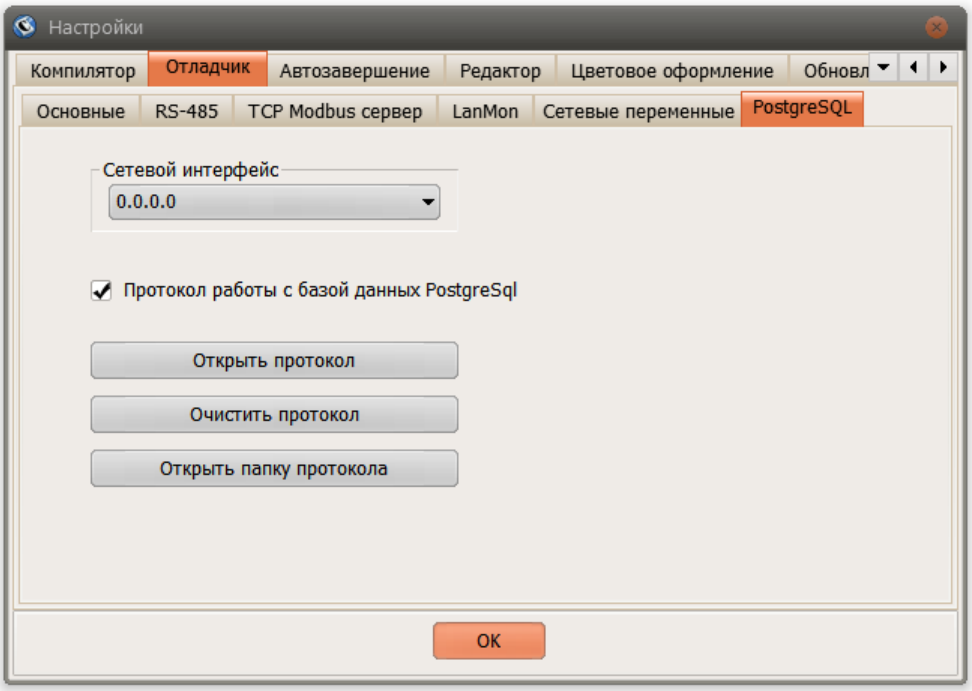
На подвкладке "Сетевые переменные" настраивается сетевой интерфейс, через которые среда разработки С-Скрипт работает с сетевыми переменными, а также можно включить и просмотреть текстовый протокол работы сетевых переменных.



Основные управляющие элементы интерфейса:

Элемент интерфейса	***Описание***
Сетевой интерфейс	Позволяет выбрать сетевой интерфейс компьютера через который среда разработки С-Скрипт работает с сетевыми переменными
Протокол работы с сетевыми переменными	Переключатель позволяет разрешить или запретить текстовый протокол работы с сетевыми переменными. Установленный переключатель разрешает ведение протокола
Открыть протокол	По нажатию на данную кнопку открывается окно, в котором можно просмотреть текстовый протокол работы с сетевыми переменными
Очистить протокол	По нажатию на данную кнопку выполняется очистка текстового протокола работы с сетевыми переменными
Открыть папку протокола	По нажатию на данную кнопку открывается папка, в которой расположен текстовый файл протокола работы с сетевыми переменными

На подвкладке "PostgreSQL" настраивается сетевой интерфейс, через которые среда разработки С-Скрипт работает с сервером баз данных Postgre SQL, а также можно включить и просмотреть текстовый протокол работы с сервером.

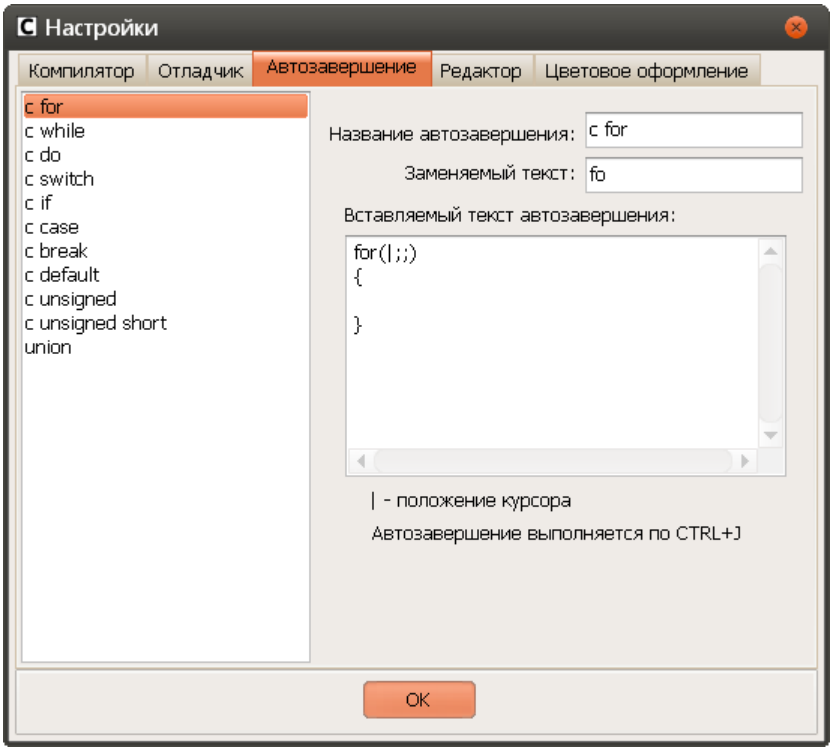


Основные управляющие элементы интерфейса вкладки:

Элемент интерфейса	***Описание***
Сетевой интерфейс	Позволяет выбрать сетевой интерфейс компьютера через который среда разработки С-Скрипт работает с сервером баз данных Post-gre SQL
Протокол работы с базой данных PostgreSql	Переключатель позволяет разрешить или запретить текстовый протокол работы с сервером баз данных Postgre SQL. Установленный переключатель разрешает ведение протокола
Открыть протокол	По нажатию на данную кнопку открывается окно, в котором можно просмотреть текстовый протокол работы с сервером баз данных Postgre SQL
Очистить протокол	По нажатию на данную кнопку выполняется очистка текстового протокола работы с сервером баз данных Postgre SQL
Открыть папку протокола	По нажатию на данную кнопку открывается папка, в которой расположен текстовый файл протокола работы с сервером баз данных Postgre SQL

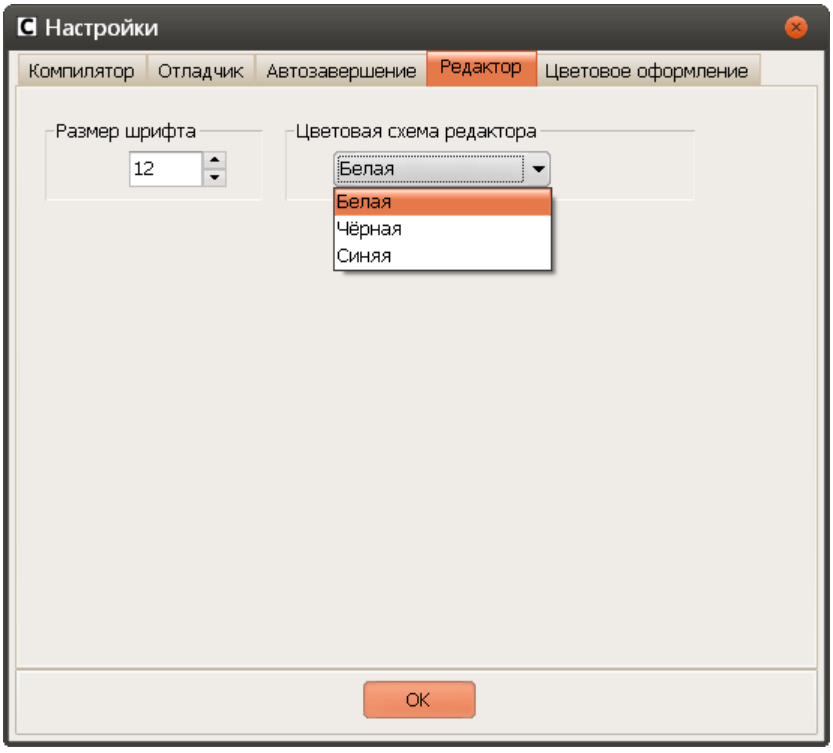
Вкладка "Автозавершение"

На вкладке настраивается функция Автозавершение текста.



Вкладка "Редактор"

На вкладке можно настроить размер шрифта редактора и цветовое оформление редактора исходного текста.

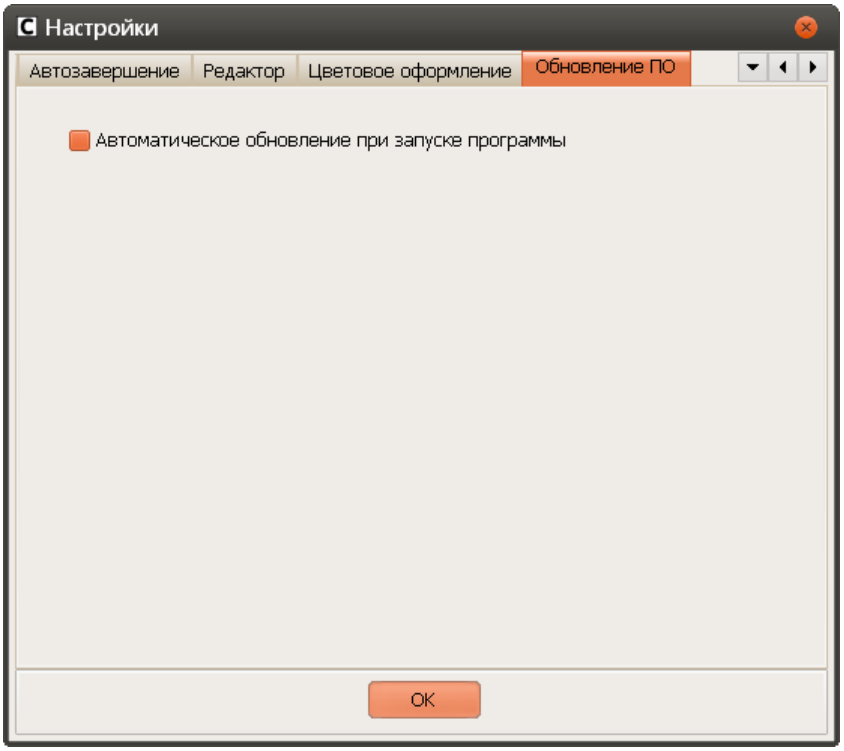


Вкладка "Цветовое оформление"

На вкладке настраивается внешний вид и цветовое оформление всех управляющих элементов программы (кроме текстового редактора).

Вкладка "Обновление ПО"

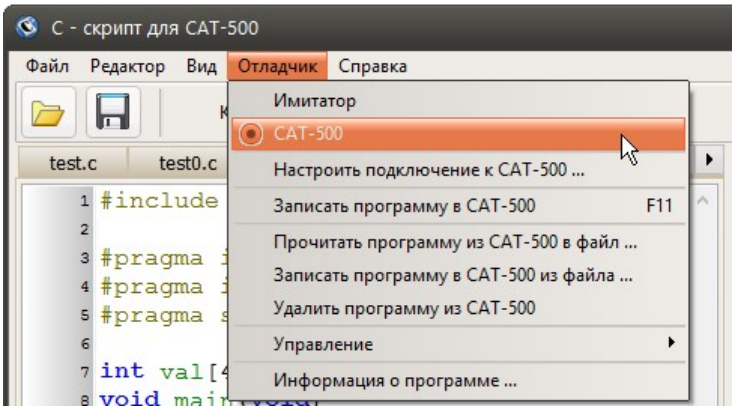
На вкладке можно включить/выключить автоматическое обновление программы.
Для обновления компьютер должен быть подключён к сети Интернет. Если переключатель "Автоматическое обновление при запуске программы" установлен, то через 10 секунд после запуска программы произойдёт автоматическая проверка обновления.



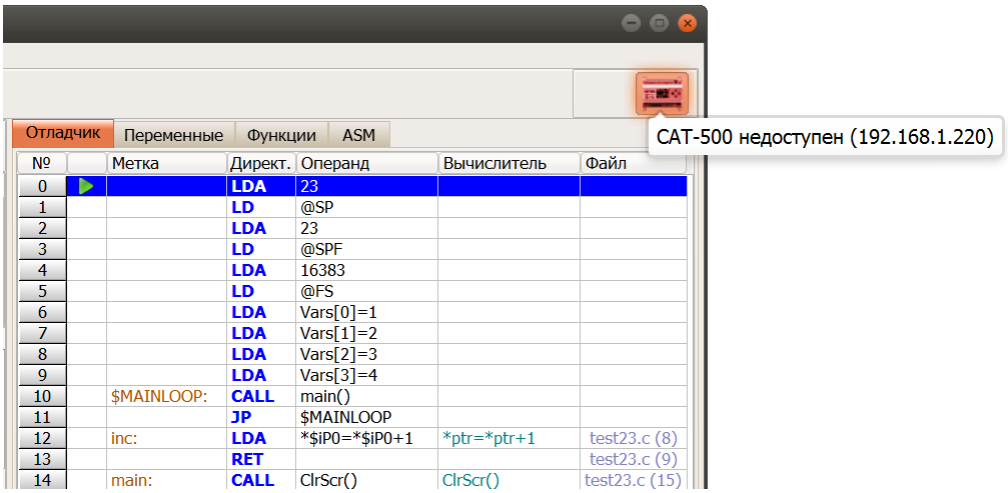
Для более подробной информации см. раздел Обновление программы.

4.8. ОТЛАДКА В CAT-500

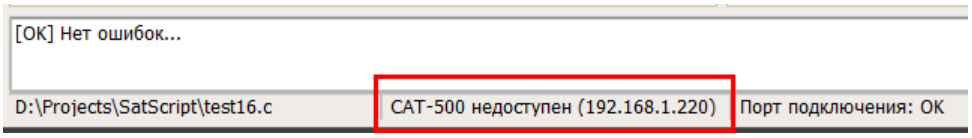
Для разрешения отладки в имитаторе необходимо в основном меню программы "Отладчик" выбрать опцию "**CAT-500**":



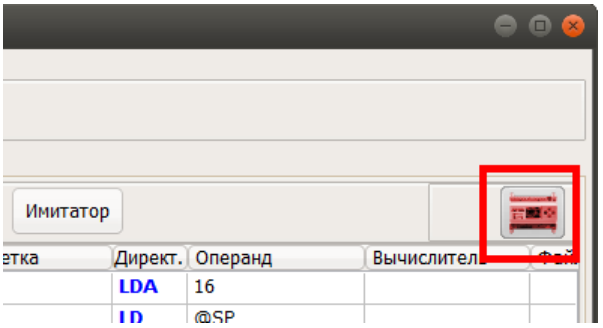
После выбора CAT-500 в правой части вкладки "Отладчик" появится кнопка с картинкой CAT-500 красного цвета.



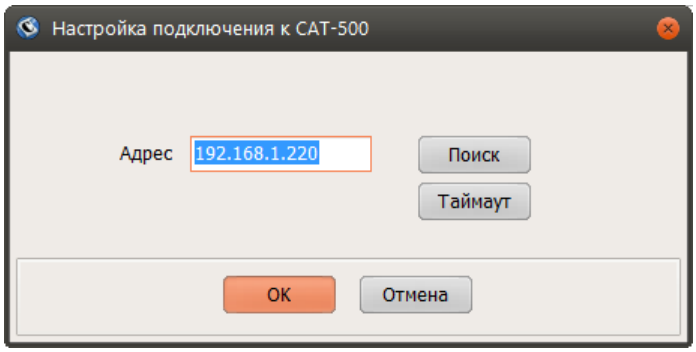
Красный цвет картинки говорит о том что контроллер CAT-500 сейчас недоступен.
Информация о подключении также отображается в статусной строке:



Для подключения к контроллеру CAT-500 следует нажать на кнопку с изображением CAT-500:

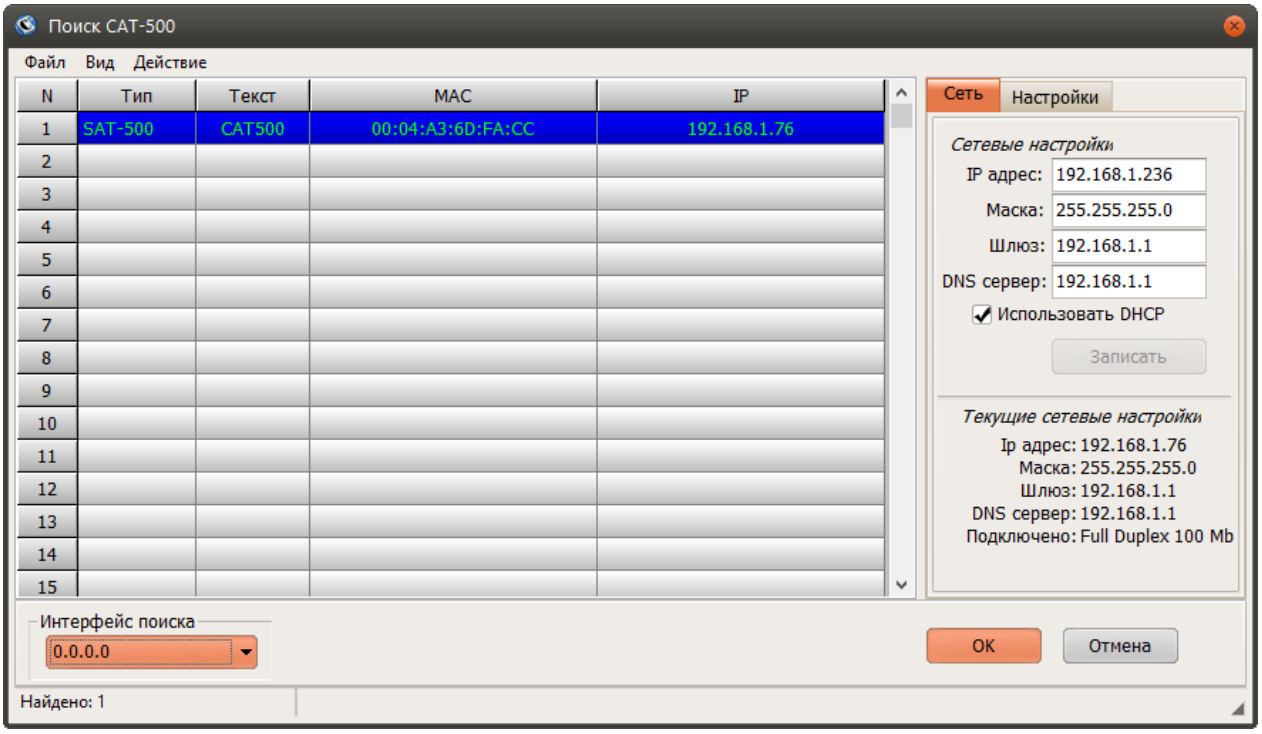


Откроется окно ввода IP адреса контроллера CAT-500:

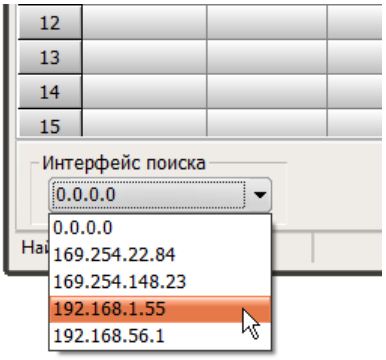


Если IP адрес контроллера CAT-500 известен, то следует ввести его в поле "*Адрес*" и нажать кнопку "*OK*".
IP адрес можно посмотреть непосредственно на дисплее контроллера. Для этого в режиме отображения основного экрана необходимо нажать кнопку "*ВВЕРХ*", появится информационный экран в котором есть строка с IP адресом:

Также можно выполнить поиск контроллера CAT-500 в локальной подсети. Этот способ сработает даже если сетевые настройки контроллера CAT-500 сделаны неправильно (не соответствуют локальной подсети в которой контроллер находится).
Для поиска контроллеров CAT-500 необходимо нажать кнопку "*Поиск*". Откроется новое окно "Поиск CAT - 500", содержащее список найденных контроллеров:



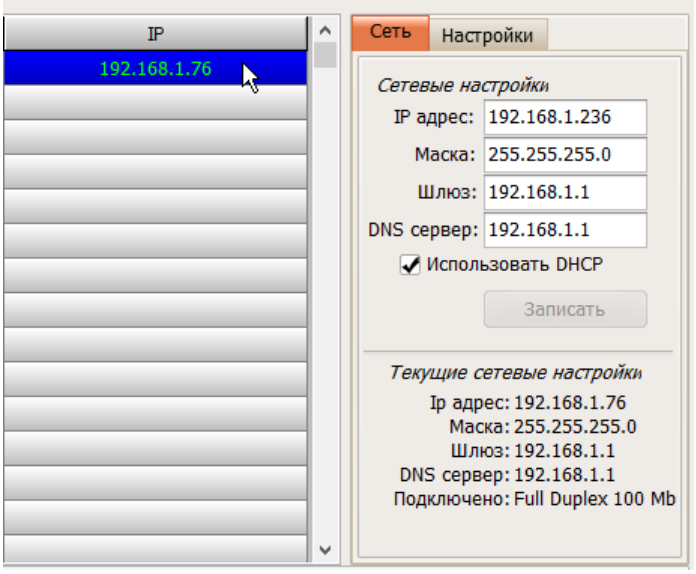
Контроллеры в списке найденных могут отсутствовать. В этом случае следует выбрать правильный интерфейс поиска в специальном поле выбора расположенном в нижней части окна:



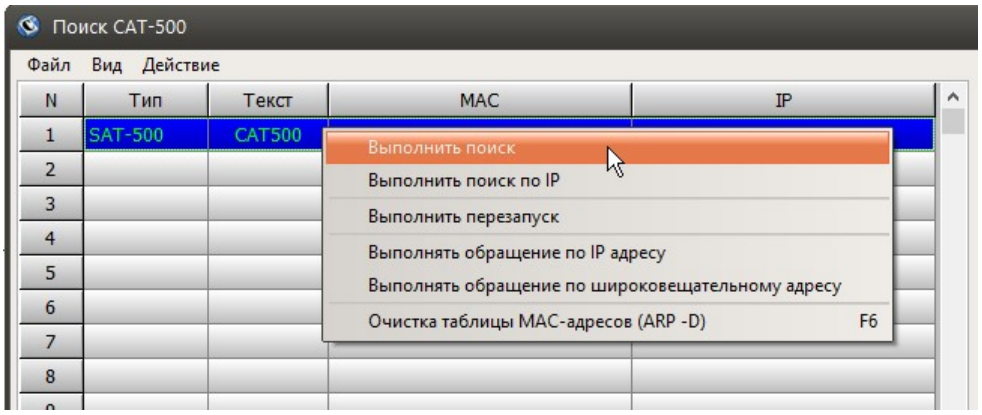
В списке "Интерфейс поиска" следует выбрать IP адрес локальной подсети в которой необходимо выполнить поиск.
Будет выполнен паспорт повторный поиск и контроллер CAT-500 появится в списке.

ВНИМАНИЕ! Поиск работает только в локальной подсети.

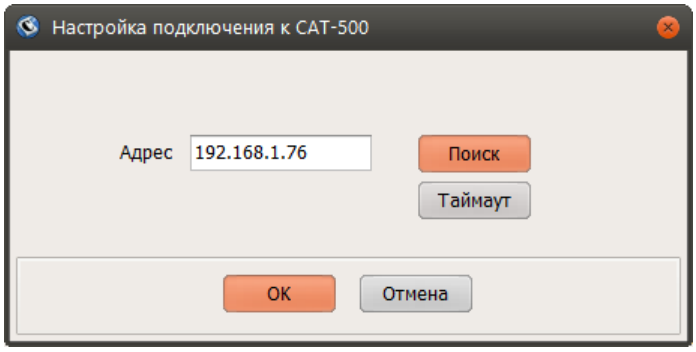
В случае, если IP адрес контроллера CAT-500, показанный в таблице найденных не соответствует локальной подсети, то необходимо выполнить настройку сетевого интерфейса контроллера CAT-500.
Для этого можно воспользоваться панелью "Сеть" в правой части окна:



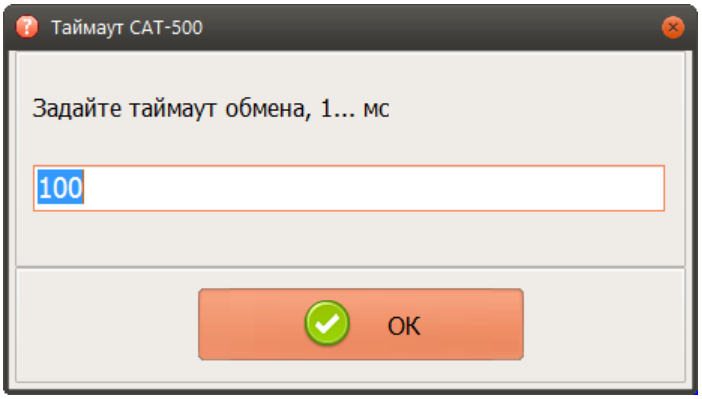
В нижней части панели показаны текущие сетевые настройки, которые могут отличаться от собственных сетевых настроек контроллера CAT-500, если для конфигурацией сети используется протокол DHCP.
На панели "Сеть" в сетевых настройках следует вести IP адрес, маску подсети, шлюз и адрес DNS сервера или же установить переключатель "Использовать DHCP". После этого необходимо нажать кнопку "Записать".
На вкладке "Настройки" находятся дополнительные настройки контроллера CAT-500, которые также можно изменить.
Если сетевые настройки контроллера были изменены, то необходимо выполнить повторный поиск выбрав пункт "Выполнить поиск" контекстного меню таблицы найденных контроллеров:



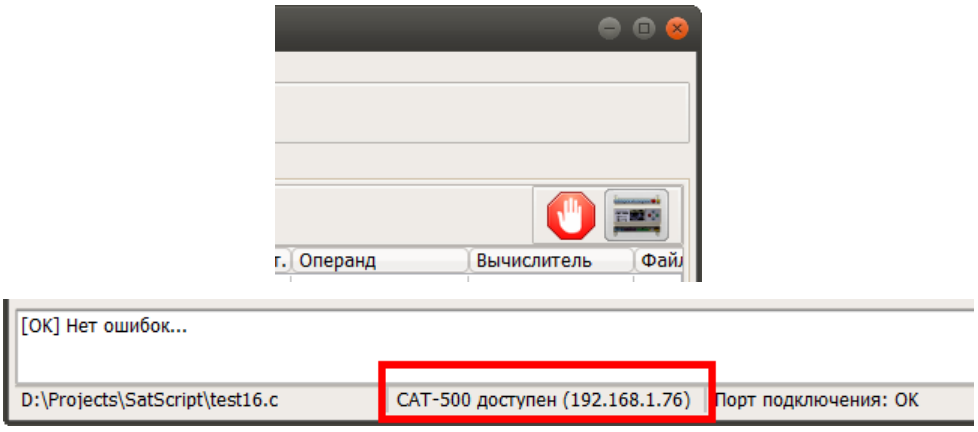
Далее следует выберать нужный контроллер CAT-500 в таблице найденных и нажать кнопку "OK".
Указанный IP адрес появится в окне настройка подключения CAT-500:



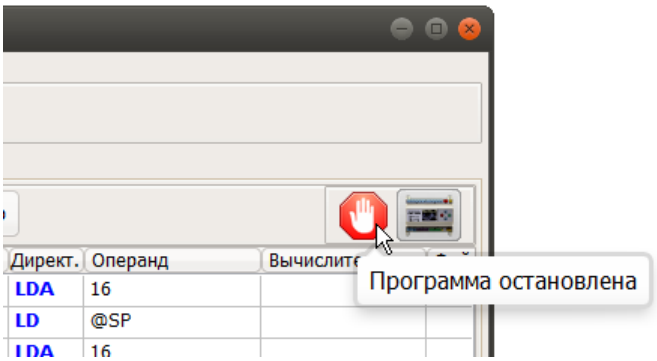
Далее нужно нажать кнопку "OK" для применения введенного адреса.
В окне "Настройка подключения к CAT-500" также можно установить таймаут обмена с контроллером CAT-500. Для этого следует нажать кнопку "Таймаут" и ввести нужное значение.



Если все сделано правильно и сетевые настройки контроллера CAT-500 соответствуют локальной подсети, то после нажатия "OK" кнопка с изображением контроллера CAT-500 изменит свой цвет с красного на серый и в статусной строке появится сообщение "CAT-500 доступен (*IP адрес*)":



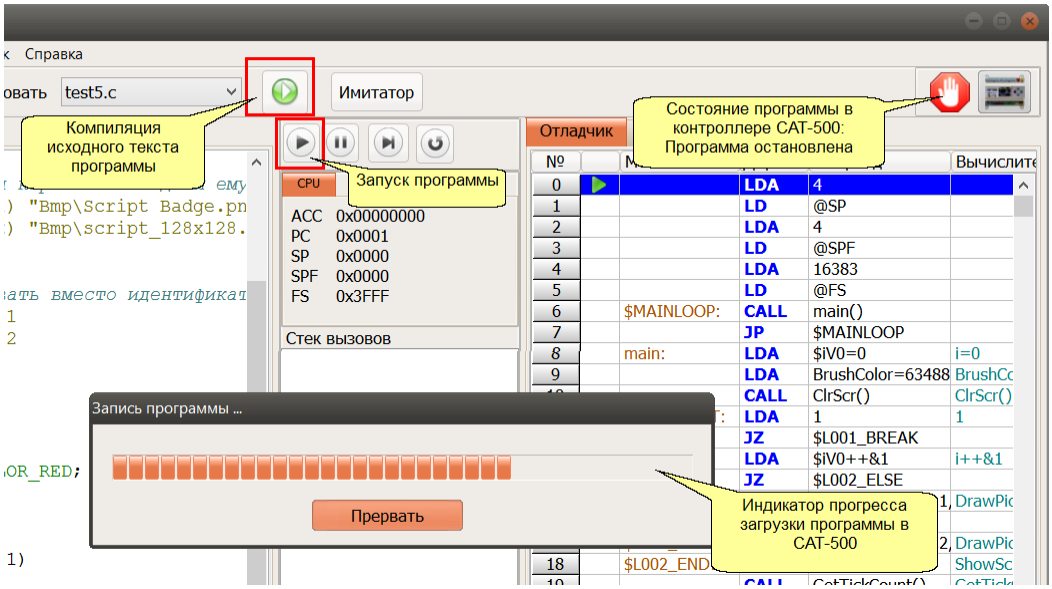
Левее кнопки "Контроллер CAT-500" появится дополнительная картинка, отображающая текущее состояние контроллера:



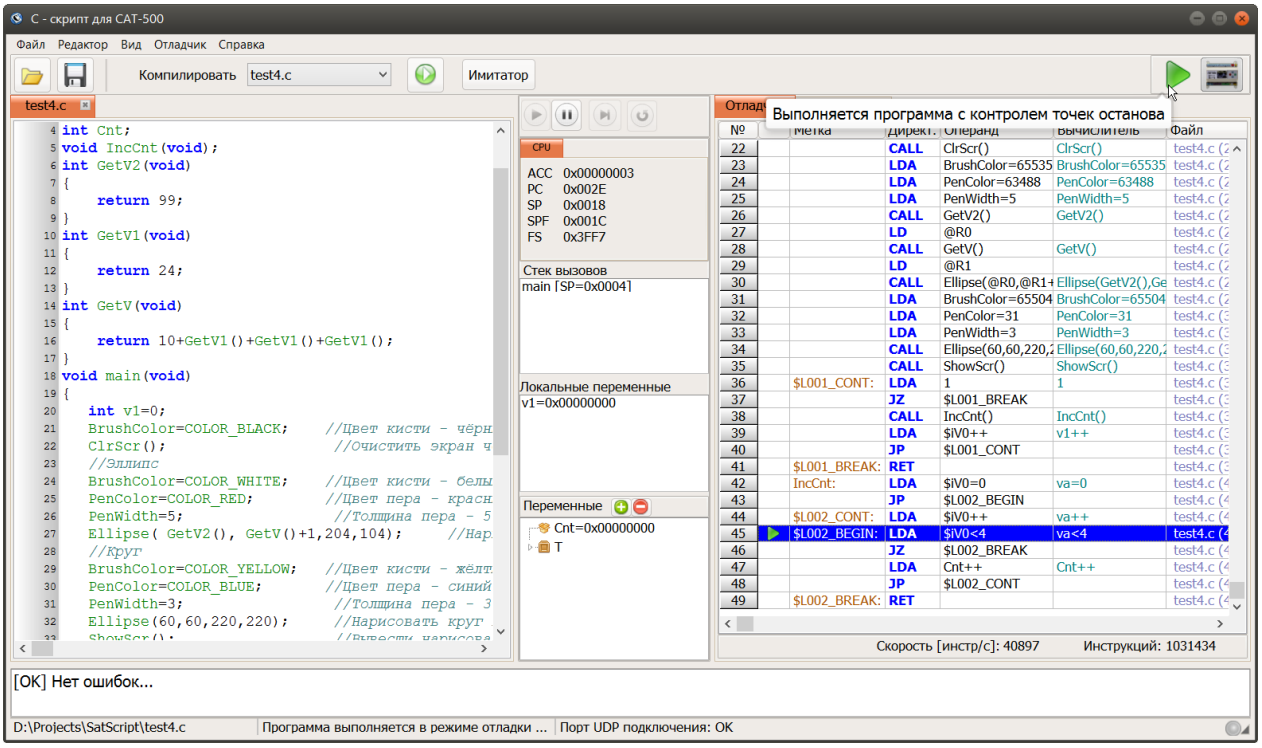
Если подвести к картинке указатель мышки, то появится подсказка с текущим состоянием контроллера.
Возможны следующие состояния контроллера:

Картинка	Состояние
	Программа не найдена
	Выполняется программа без отладки (нормальная работа после включения контроллера)
	Программа остановлена в режиме отладки
	Программа выполняется в режиме отладки
	Во время выполнения программы произошла ошибка

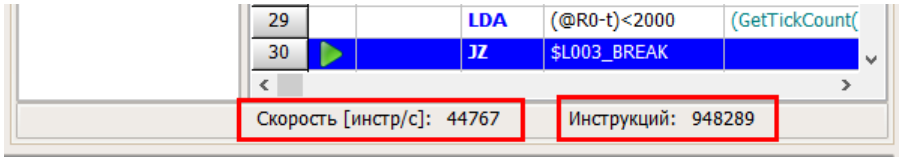
Теперь контроллер полностью готов к отладке.
Для начала отладки следует откомпилировать программу нажав на кнопку "*Компилировать программу*" или клавишу "*F2*" на клавиатуре компьютера.
Затем нажать кнопку "*Запуск программы*" или клавишу "*F9*" на клавиатуре компьютера.
Далее будет выполнена загрузка программы в контроллер, и загруженная программа автоматически запустится в режиме отладки, то есть программа будет отслеживать заданные точки останова и останавливаться при достижении такой точки.



В процессе выполнение программы в контроллере CAT-500 каждую секунду обновляется текущая выполняемая строка на вкладке "Отладчик":



В нижней части вкладки отладка отображается количество инструкций псевдоассемблера, выполняемых за секунду, а также общее количество инструкций выполненных процессором.

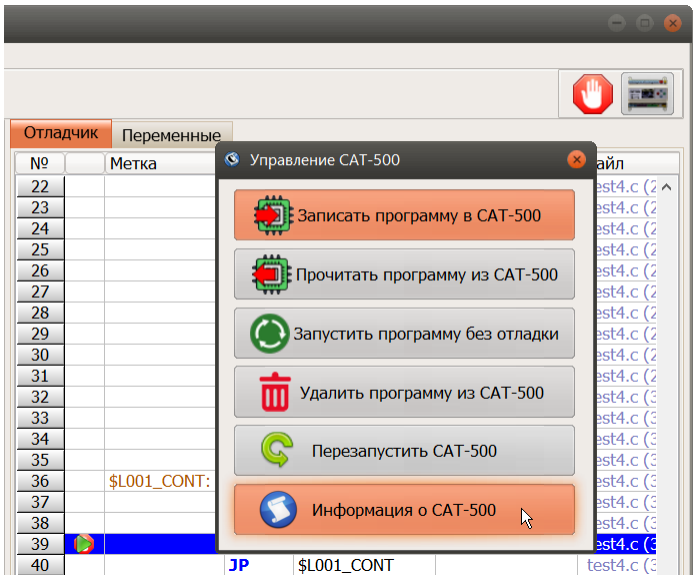


Выполнения программы можно остановить в любой момент времени нажав на кнопку "*Стоп*" или на клавишу "*F9*" на клавиатуре компьютера.
Также программа останавливается автоматически по достижении любой точки останова.
В момент останова считываются глобальные и локальные переменные, состояния регистров процессора, состояние стека и все это отображается на панелях вкладки "Отладка".

В общем процесс отладки практически не отличается от отладки в имитаторе.

4.9. УПРАВЛЕНИЕ CAT-500

Пользователь может выполнить некоторые дополнительные функции по управлению контроллером CAT-500.
Для вызова меню управления следует нажать левой кнопкой мышки по картинке отображающий состояние контроллера.



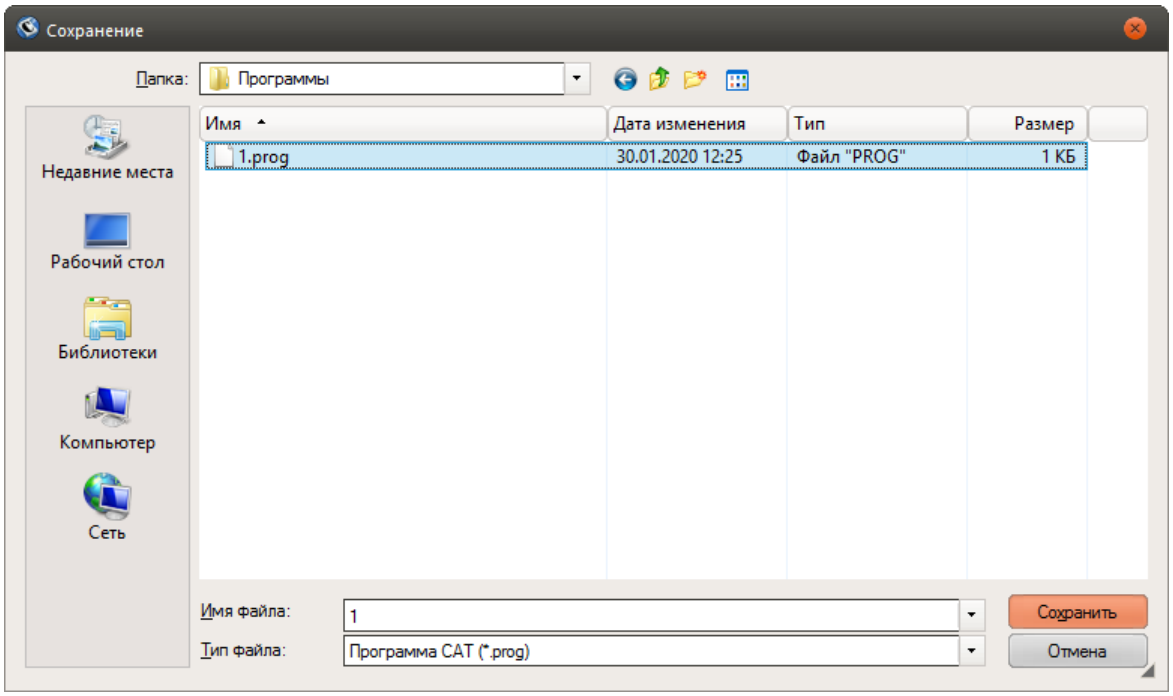
Открывается специальная панель "Управление CAT-500", состоящая из кнопок управления.

Нажав на любую из кнопок можно выполнить то или иное действие управлению контроллером CAT-500.
Доступны следующие действия:

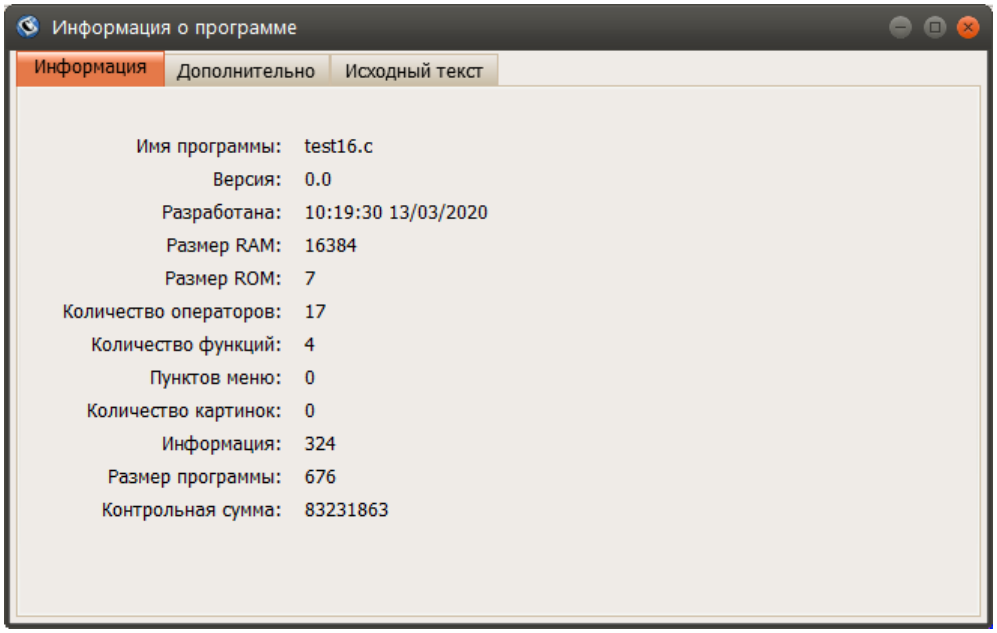
Название кнопки	***Действие***
Записать программу в CAT-500	Выполняется запись программы в контроллер без запуска отладки
Прочитать программу из CAT-500	Выполняется чтения программы, ранее загруженной в контроллер и показывается специальное окно, отображающее информацию о программе. Программа сохраняется в локальный файл и в дальнейшем её можно записать в другой контроллер CAT-500
Запустить программу без отладки	Программа ранее загруженная в контроллер CAT-500 запускается без отладки, то есть выполняется такое же действие как при подачи питания на контроллер
Удалить программу из CAT-500	Программа, ранее загруженная в контроллер CAT-500 удаляется
Перезапустить CAT-500	Выполняется программный перезапуск контроллера CAT-500
Информация о CAT-500	Считывается информация о контроллере CAT-500 и отображается в специальном дополнительном окне

Прочитать программу из CAT-500

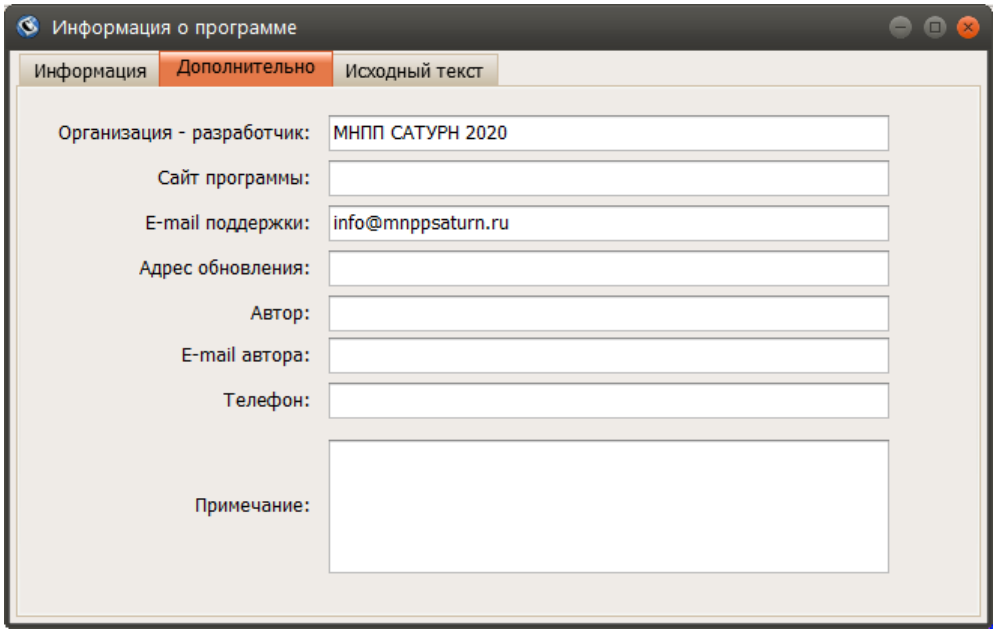
В случае выбора "Прочитать программу из CAT-500" появляется диалог выбора пути и имени файла для сохранения:



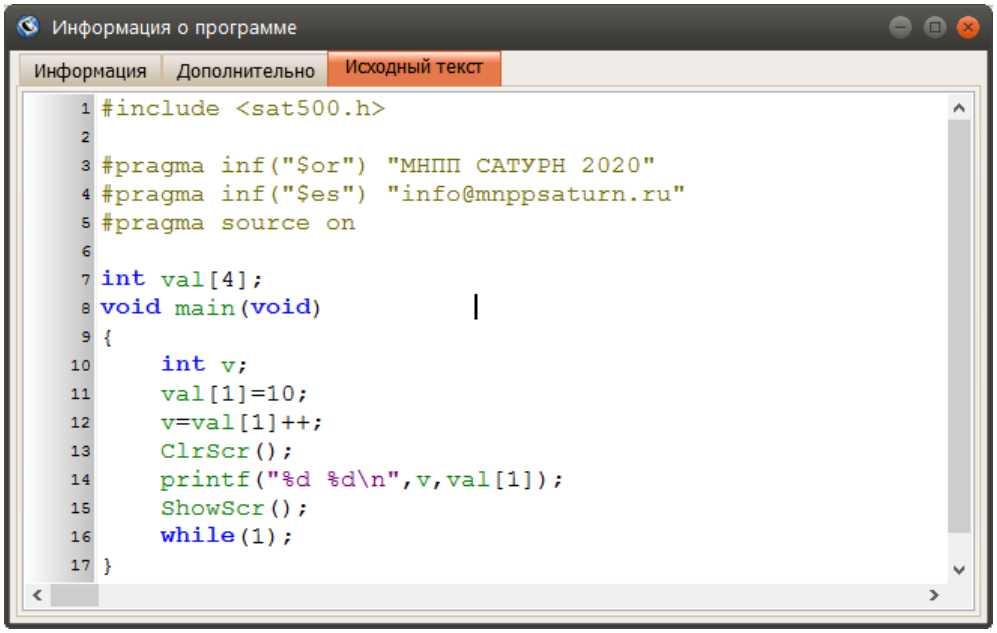
Следует ввести имя файла, которое будет использовано для прочитанной программы.
После нажатия кнопки "Сохранить" программа будет прочитана из CAT-500 и сохранена в указанном файле. Также будет показано следующее окно "Информация о программе":



На вкладке "Дополнительно" отображаются расширенная информация о программе, заданная при помощи директивы *#pragma inf* (см. раздел *Включение в программу дополнительной информации*)



И на вкладке "Исходный текст" может быть доступен исходный текст программы, если он включён при помощи директивы *#pragma source on* (см. раздел *Включение исходного текста программы в файл программы САТ-500*).



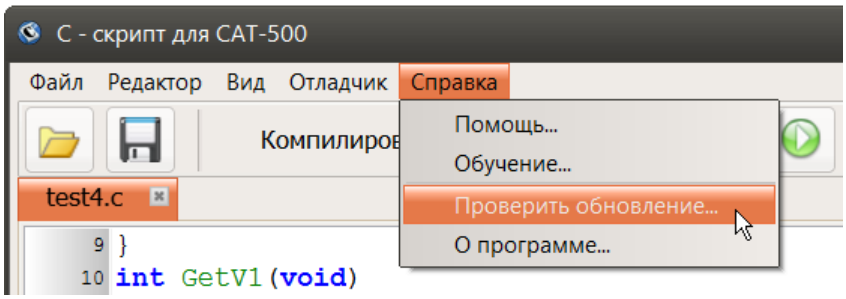
Информация о САТ-500

На рисунке ниже показано окно, возникающая при выборе пункта "Информация о САТ-500":

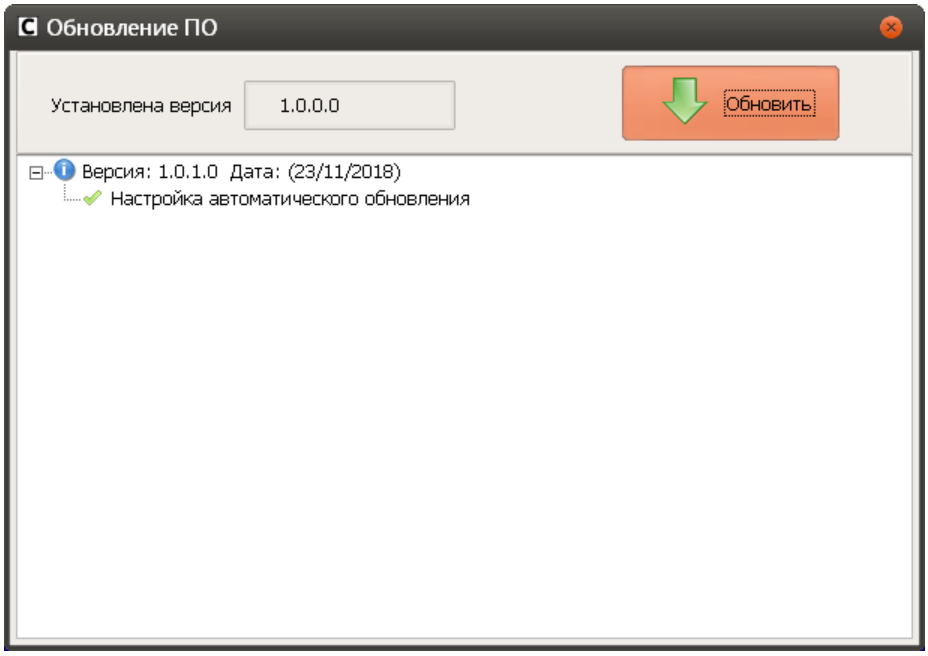
В окне отображается прочитанная информация о контроллере САТ-500.

4.10. ОБНОВЛЕНИЕ ПРОГРАММЫ КОМПИЛЯТОРА

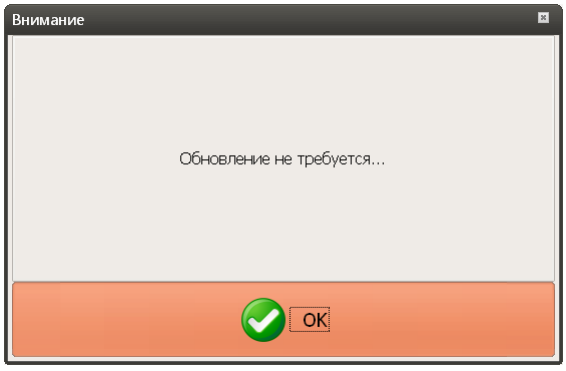
Программное обеспечение компилятор "С-Скрипт" постоянно совершенствуется. Чтобы работать с последней версией программы можно обновить программное обеспечение, выбрав пункт "**Справка/Проверить обновление...**" основного меню программы.



В случае если имеется обновление, то появится окно с предложением установить новую версию и описание изменений в новой версии:



Для начала обновления необходимо нажать кнопку "*Обновить*". Для отказа от обновления просто закройте окно "Обновление ПО".
В случае выбора "*Обновить*" автоматически будет загружен и запущен установщик программы, который обновит до самой новой версии программы.
Все настройки пользователя и данные будут сохранены.
Если программа не требует обновления, то появится следующее сообщение:

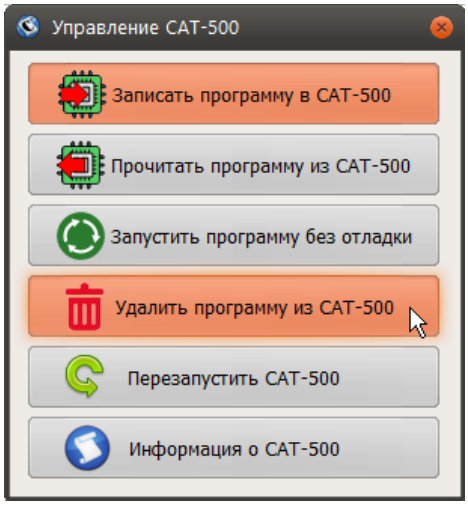


Так же можно настроить автоматическое обновление на вкладке *Обновление ПО* в окне настроек программы.

4.11. УДАЛЕНИЕ С-СКРИПТ ПРОГРАММЫ

Удаление С-скрипт программы из контроллера CAT-500 может быть выполнено двумя способами:

1. Подключиться к контроллеру CAT-500 в среде разработки, вызвать меню управления, нажав левой кнопкой мышки по картинке отображающий состояние контроллера. Затем нажать кнопку "**Удалить программу из CAT-500**"



2. Снять питание с контроллера CAT-500. Нажать и удерживать кнопку "*ВЛЕВО*" и кнопку "*ВПРАВО*" на клавиатуре контроллера. Подать питание на контроллер, продолжая удерживать кнопки до появления стартовой надписи С-скрипта.

Лист регистрации изменений

№ изменения	Номера листов				Всего листов в документе	№ документа	Подпись	Дата
	измененных	замененных	новых	аннулированных				