SIEMENS

SIMATIC

S7-SCL V5.1 для S7-300/S7-400

Руководство

Это руководство имеет заказной номер:
6ES7811-1CC04-8BA0

Редакция 09/2000
A5E00059543-01

Указания по безопасности

Это руководство содержит указания, которые вы должны соблюдать для обеспечения собственной безопасности, а также защиты продукта и подключенного оборудования. Эти указания выделены в руководстве предупреждающим треугольником и помечены следующим образом в соответствии с уровнем опасности:

Опасность
Указывает, что несоблюдение надлежащих предосторожностей приведет к смерти, тяжким телесным повреждениям или существенному повреждению имущества.

Предупреждение
Указывает, что несоблюдение надлежащих предосторожностей может привести к смерти, тяжким телесным повреждениям или существенному повреждению имущества.

Предостережение
Указывает, что несоблюдение надлежащих предосторожностей может привести к небольшим телесным повреждениям или порче имущества .

Замечание
привлекает внимание к особенно важной информации о продукте, его использовании или к конкретной части документации

Квалификация персонала

К установке и работе на данном оборудовании должен допускаться только квалифицированный персонал. К квалифицированному персоналу относятся лица, имеющие право пускать в эксплуатацию, заземлять и маркировать электрические цепи, оборудование и системы в соответствии с установленным порядком и стандартами.

Правильное использование

Не забудьте следующее:

Предупреждение
Это устройство и его компоненты могут использоваться только для приложений, описанных в каталоге или технических описаниях, и только в соединении с устройствами или компонентами других изготовителей, которые одобрены или рекомендованы Siemens.

Этот продукт может правильно и успешно функционировать только, если он транспортируется, хранится, монтируется и установливается правильно, обслуживается и поддерживается как рекомендуется.

Торговые марки

SIMATIC®, SIMATIC HMI® и SIMATIC NET® - зарегистрированные торговые марки SIEMENS AG.

Некоторые другие обозначения использованные в этих документах - также зарегистрированные фирменные знаки; права владельца могут нарушаться, если они используются третьими партиями для своих собственных целей.

Siemens AG
Департамент техники автоматизации и приводов
Сфера деятельности: промышленные системы автоматизации
Postfach 4848, D- 90327 Nuernberg

Отказ от ответственности
Мы проверили содержание этого руководства на соответствие с описанной аппаратурой и программным обеспечением. Так как отклонения не могут быть полностью предотвращены, мы не гарантируем полного соответствия. Однако данные, приведенные в этом руководстве, регулярно пересматриваются и необходимые исправления вносятся в последующие издания. Приветствуются предложения по улучшению.

Excellence in Automation & Drives: Siemens

Введение

Цель Руководства

Это руководство дает Вам полный обзор программирования на S7-SCL. Оно поддерживает Вас при установке программного обеспечения. Оно включает объяснения как создавать программу, структуру программ пользователя, и отдельные языковые элементы.

Руководство предназначается для программистов, пишущих SCL программы, и людей, занятых конфигурацией, установкой и обслуживанием программируемых логических контроллеров.

Мы рекомендуем Вам ознакомиться с примером, описанным в Главе 2 "Разработка SCL программы". Это поможет Вам быстро понять SCL.

Требования к квалификации

Чтобы понять руководство, Вы должны иметь общий опыт проектирования систем автоматизации.

Желательно, чтобы Вы были бы также знакомыми с работой на компьютерах или ЭВМ типа PC (например, программируя устройства с использованием операционных системы Windows 95/98/2000 или NT). Поскольку SCL использует, как платформу, STEP 7, требуется знакомство с работой со стандартным программным обеспечением, описанным в руководстве "Программирование на STEP 7 V5.1".

Область руководства

Руководство распространяется на программный пакет S7-SCL V5.1.

Пакеты документации для S7-SCL и стандартного пакета STEP 7

В следующей таблице дан обзор документации для STEP 7 и SCL:

Руководство	Назначение	Заказной номер
Основы SCL и справочник: • S7-SCL для S7-300/400, Программирующие блоки	Основная и справочная информация разъясняет, как создавать программу, структуру программ пользователя и отдельные языковых элементов.	6ES7811-1CC04-8XA0
Основы STEP 7: • Быстрый старт и упражнения для STEP 7 V5.1 • Программирование на STEP 7 V5.1 • Конфигурация оборудования и связи на STEP 7 V5.1 • Преобразование из S5 в S7	Основные сведения для технического персонала, описывающие как решать задачи управления с использованием STEP 7 и S7-300/400.	6ES7810-4CA05-8AA0
Справочники по STEP 7: • Руководство по LAD/FBD/STL для S7-300/400 • Стандартные и системные функции для S7-300/400	Справочники описывают стандартные языки программирования LAD, FBD и STL и системные функции как дополнение к основам STEP 7.	6ES7810-4CA05-8AR0

Встроенная помощь	Назначение	Заказной номер
Help (помощь) в S7-SCL	Основы и справочник для S7-SCL как встроенная помощь	Часть поставки программы S7-SCL
Help (помощь) в STEP 7	Основы программирования и конфигурирования аппаратуры в STEP 7 как встроенная помощь	Часть поставки программы STEP 7
Контекстная помощь по STL/LAD/FBD Контекстная помощь по SFB/SFC Контекстная помощь по организационным блокам Контекстная помощь по функциям IEC Контекстная помощь по системным атрибутам	Контекстная помощь	Часть поставки программы STEP 7

Встроенная помощь

В дополнение к руководствам, встроенная помощь (online help) интегрирована в программное обеспечение и поддерживает Вас непосредственно поддержкой при работе с программой.

Доступ к системе помощи, встроенной в программное обеспечение, осуществляется через различные интерфейсы:

- Меню **Help (Помощь)** обеспечивает ряд команд: **Contents (Содержание)** открывает содержание системы помощи SCL. **Introduction (Введение)** дает обзор программирования на SCL. **Using Help (Использование помощи)** дает подробные инструкции по работе с системой помощи.
- Система контекстно-зависимой помощи обеспечивает информацию о текущем контексте, например, помощь по использованию открытого диалогового окна или активного окна. Она показывается при нажатии кнопки "Help (Помощь)" и нажатии на клавишу F1.
- Строка состояния другая форма контекстно-зависимой помощи. В ней показано краткое объяснение каждой команды меню, когда Вы позиционируете указатель мыши в этой команде.
- Краткое объяснение кнопок в панели инструментов также отображается, если Вы позиционируете указатель мыши кратко над кнопкой.

Если Вы предпочитаете, иметь распечатку информации, содержащейся в системе помощи, Вы можете напечатать отдельные рубрики, книги или всю систему помощи.

В систему помощи SCL входит это руководство в виде файла формата HTML. Так как помощь в руководстве и программе имеет одинаковую структуру, Вы можете легко переключаться между руководством и помощью.

Документация SIMATIC в Internet и Intranet Siemens

Дополнительную информацию о документации SIMATIC Вы также найдете в Internet или Intranet SIEMENS.

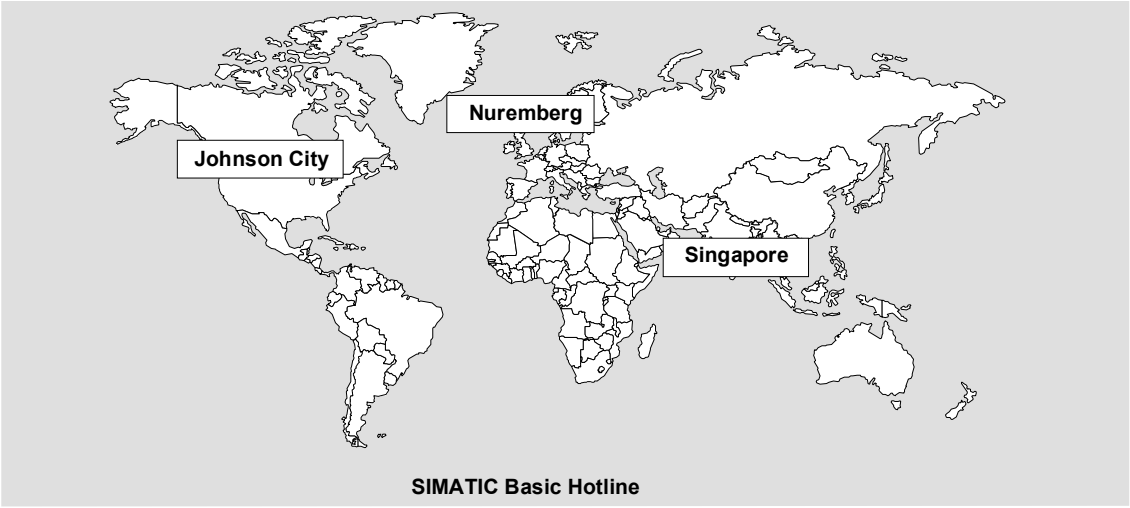
- Вы можете получить новейшую документацию по адресу
  - **Internet** http://www.ad.siemens.de/meta/html_00/support.shtml. Используйте Knowledge Manager для поиска требуемой документации.
- Вы можете задать вопросы по документации SIMATIC по следующему адресу. Вы быстро получите ответы на ваши проблемные вопросы.
  - **Internet** http://www4a.ad.siemens.de:8090/~SIMATIC/login
- Кроме того, посещайте страницу сопровождение документации SIMATIC. Здесь Вы можете узнать о новых продуктах и новшествах, послать вопросы о документации и передать Ваши просьбы, предложения, критику или благодарность.
  - **Siemens Intranet** http://intra1.khe.siemens.de/e8_doku/index.htm

Учебный центр SIMATIC

Пожалуйста обратитесь к вашему местному центру подготовки или центральному центру подготовки: D 90327 Nuremberg, Germany.
Phone: +49 (911) 895-3200.

Горячая линия поддержки пользователей SIMATIC

Доступна в любом время в любом часовом поясе:



SIMATIC Basic Hotline

Всемирная техническая поддержка (Nuremberg)	**Всемирная техническая поддержка** (Nuremberg)	
(Свободный контакт) Местное время: Пн.-Пят. 7:00 - 17:00 Тел: +49 (180) 5050 222 Факс: +49 (180) 5050 223 E-mail: techsupport@ad.siemens.de GMT: +1:00	(только по картам SIMATIC) Местное время: Пн.-Пят. 0:00 - 24:00 Тел: +49 (911) 895-7777 Факс: +49 (911) 895-7001 GMT: +01:00	
Европа / Африка (Nuremberg) **Авторизация**	**Америка** (Johnson City) **Техническая поддержка и авторизация**	**Азия / Австралия** (Singapore) **Техническая поддержка и авторизация**
Местное время: Пн.-Пят. 7:00 - 17:00 Тел: +49 (911) 895-7200 Факс: +49 (911) 895-7201 E-mail: authorization@nbgm.siemens.de GMT: +1:00	Местное время: Пн.-Пят. 8:00 -19:00 Тел: +1 423 461-2522 Факс: +1 423 461-2289 E-mail: simatic.hotline@sea.siemens.com GMT: -5:00	Местное время: Пн.-Пят. 8:30 - 17:30 Phone: +65 740-7000 Факс: +65 740-7001 E-mail: simatic.hotline@sae.siemens.com.sg GMT: +8:00

По-немецки и английски можно говорить на всех горячих линиях SIMATIC, по-французски, итальянски и испански можно говорить по горячей линии авторизации.

Служба технической поддержки пользователей SIMATIC

Служба технической поддержки пользователей SIMATIC обеспечивает Вас всеми видами дополнительной информации по продуктам SIMATIC:

- Вы можете получить общую свежую информацию:
  - **Internet** http://www.ad.siemens.de/simatic
- Бюллетени информации о выпускаемых продуктах и полезные советы:
  - **Internet** http://www.ad.siemens.de/simatic-cs
  - **Bulletin Board System** (BBS) в Nuremberg (*SIMATIC Customer Support Mailbox)* тел. +49 (911) 895-7100.

  Для подключения к почтовому ящику используйте модем до V.34 (28.8 Кбод) с следующими параметрами установки: 8, N, 1, ANSI, или абонент через ISDN (x.75, 64 Kbps).
- Вы найдете ваш местный контактный адрес для Automation & Drives в нашей базе данных контактов:
  - on the **Internet** at http://www3.ad.siemens.de/partner/search.asp

Содержание

1 Обзор и установка

1.1 Обзор S7-SCL

От современных программируемых контроллеров требуется, чтобы они, в дополнение к традиционным задачам управления, решали задачи управления данными и сложные математические задачи. В настоящее время для решения подобных задач мы предлагаем SCL (**S**tructured **C**ontrol **L**anguage - Структурный Управляющий Язык), язык программирования, который облегчает программирование для S7300/400 и соответствует DIN EN 61131-3.

SCL не только помогает Вам с "обычными" задачами управления, но также с расширенными приложениями, дополняя "традиционные" языки программирования следующими возможностями:

- Управление данными
- Оптимизация управления
- Управление рецептурами
- Математические и статистические операции

S7-SCL язык программирования

SCL является языком высокого уровня программирования, подобным PASCAL. Он базируется на стандарте для ПЛК (программируемых логических контроллеров).

Стандарт DIN EN-61131-3 (международный IEC 1131-3) определяет требования к языкам программирования для логических контроллеров. Язык программирования SCL подчиняется определяемому в этом стандарте уровню PLCOpen Basis ST (структурный текст). В файле NORM_TAB.WRI, Вы найдете точное определение соответствия стандарту DIN EN-61131-3.

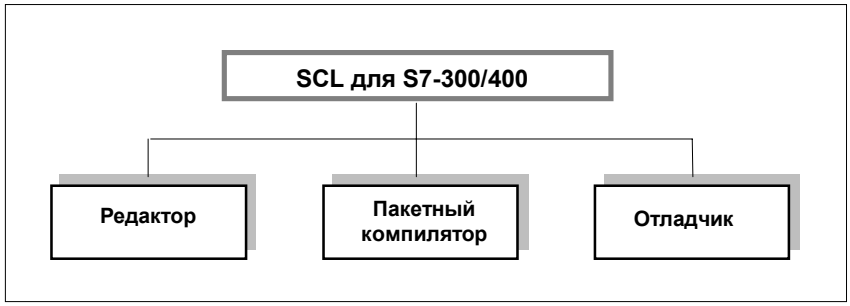
Кроме элементов языков высокого уровня SCL включает языковые элементы, типичные для ПЛК, как, например, входы, выходы, таймеры, меркеры, вызовы блоков и т.п. Другими словами, SCL дополняет и расширяет STEP 7 и его языки программирования: LAD (контактный план), FBD (функциональная блочная диаграмма), и STL (список команд).

Среда разработки

Для оптимального использования и практического применения SCL имеется мощная среда разработки, сочетающая специфические характеристики как SCL, так и STEP 7. Эта среда разработки состоит из следующих компонентов:

- **Редактор** для написания программ, состоящих из функций (FC), функциональных блоков (FB), организационных блоков (OB), блоков данных (DB) и определяемых пользователем типов данных (UDT). Редактор имеет развитые функции поддержки работы программиста.
- **Пакетный компилятор** для компиляции отредактированной программы в машинный код MC7. Сгенерированный код MC7 будет работать на всех CPU S7-300/400, начиная с CPU 314.
- **Отладчик** для поиска логических ошибок программирования в скомпилированной программе. Отладка выполняется на уровне входного языка.

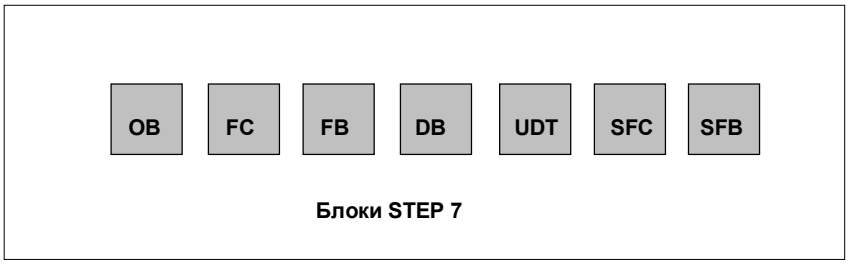
Компоненты SCL просты и удобны для пользователя с опытом работы в Windows и используют все преимущества этой операционной системы.



1.2 В чем преимущества S7-SCL?

SCL предоставляет Вам все преимущества языка программирования высокого уровня. SCL также имеет многие возможности, специально разработанные для поддержки структурного программирования, например,:

- SCL поддерживает концепцию блоков STEP 7 и, следовательно, допускает программирование блоков аналогичное программированию на языках Списка команд (STL), Контактного плана (LAD) и Функциональной диаграммы (FBD).



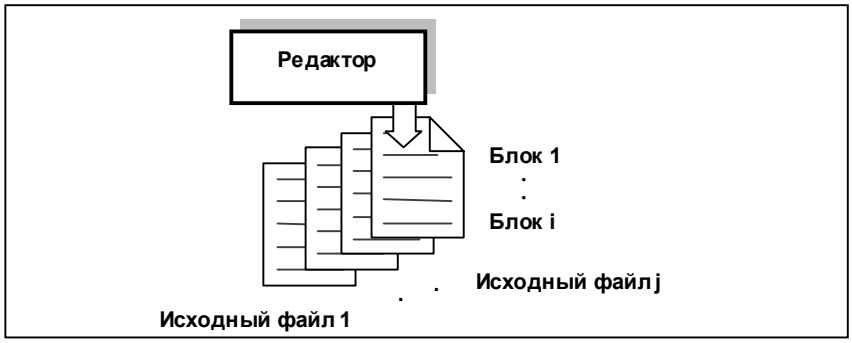
Блоки STEP 7

- Вам не нужно писать самим каждую функцию и можно использовать готовые блоки, например, системные функции или системные функциональные блоки, уже существующие в операционной системе CPU.
- Вы можете использовать блоки, запрограммированные на SCL, в комбинации с блоками, запрограммированными в списке команд (STL), контактном плане (LAD) и функциональной диаграмме (FBD). Это означает, что блок, написанный в SCL, может вызывать блоки, написанные в STL, LAD или FBD. Аналогично, блоки SCL могут вызываться из блоков, написанных на STL, LAD или FBD. Следовательно, языки программирования STEP 7 и SCL (дополнительный пакет) дополняют друг друга.
- Исходные объекты, которые Вы создаете на SCL для STEP 5, совместимы вверх с одним или двумя незначительными исключениями; иначе говоря, эти программы могут также редактироваться, компилироваться и отлаживаться с S7 SCL.
- Блоки SCL можно декомпилировать в язык списка команд STEP 7 (STL). Дальнейшая декомпиляция из STL на SCL не возможна.
- SCL может быть быстро изучен при некотором опыте работы на языках программирования высокого уровня.
- При написании программ, программист располагает развитыми средствами обработки исходного текста.
- Блоки, созданные при компиляции отредактированной программы могут выполняться во всех CPU S7 300/400, начиная с CPU 314.
- Применяя тестовые и отладочные функции SCL, Вы можете найти логические ошибки программирования в скомпилированной программе. Отладка выполняется на уровне входного языка.

1.3 Характеристики среды разработки

Редактор

Редактор SCL является текстовым редактором, который может использоваться для редактирования любых текстовых файлов. Основная цель - создание и редактирование исходных файлов для программ STEP 7. В исходном файле Вы можете запрограммировать один или более программных блоков. Редактор не проверяет синтаксис текста при его вводе.

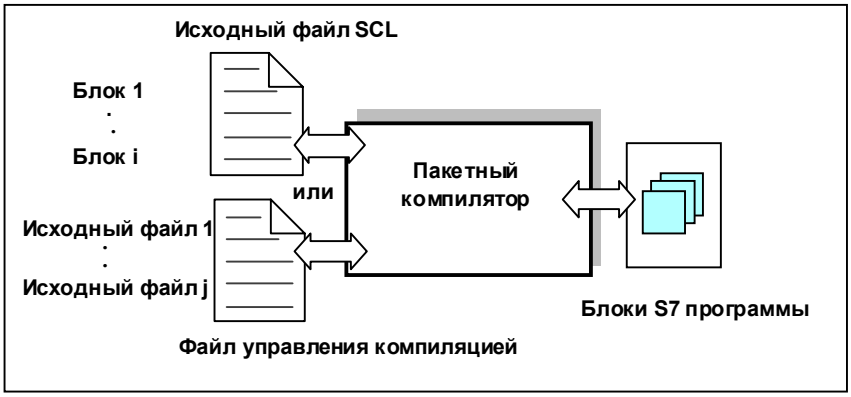


Используя редактор SCL, Вы можете:

- Редактировать целиком исходный файл, включающий один или более блоков
- Редактировать файл управления компиляцией, с которым Вы можете автоматизировать компиляцию серии исходных файлов
- Использовать дополнительные функции, которые упрощают задачу редактирования исходного файла, например, поиск и замену.
- Модифицировать установки редактора, чтобы удовлетворить Ваши требования, например, изменить окраску в соответствии с синтаксисом различных языковых элементов.

Компилятор

Как только Вы, используя редактор SCL, создали исходные файлы, Вы компилируете их в код MC7.

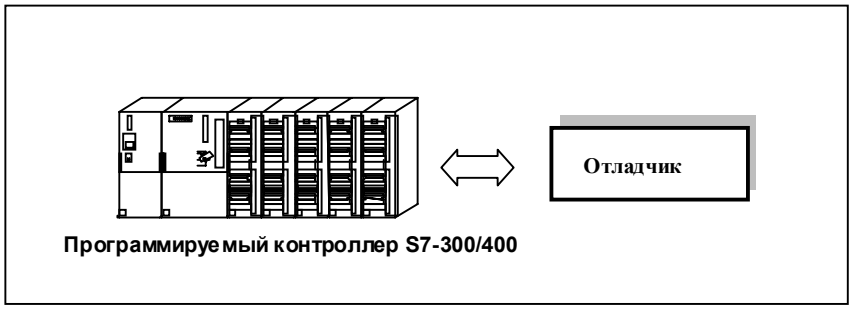


Компилятор SCL позволяет Вам:

- Компилировать за один прием исходный файл SCL, состоящий из многих блоков.
- Компилировать серию исходных файлов SCL, используя файл управления компиляцией, содержащий имена исходных файлов.
- Компилировать выбранные блоки из исходного файла.
- Изменить установки компилятора, в соответствии со своими специфическими требованиями.
- Просматривать ошибки и предупреждения, полученные при компиляции.
- Локализовать ошибки в исходном файле, получив при необходимости описания ошибок и советы по их исправлению.

Отладчик

Отладчик SCL обеспечивает проверку выполнения программ в ПЛК и идентифицирует любые логические ошибки.



SCL предусматривает два различных режима тестирования:

- **Пошаговый** просмотр - следует за логикой выполнения последовательности программы. Программа выполняется по одной команде и показывает в окне результатов изменение переменных;
- **Непрерывный** мониторинг - в этом режиме Вы можете протестировать группу команд в пределах блока исходного файла. В течение теста, величины переменных и параметров отображаются в хронологической последовательности и (где возможно) обновляются циклически.

1.4 Что нового в версии V5.1?

DIN EN 61131-1

Начиная с версии 5.0, S7-SCL подчиняется базисному уровню PLCOpen для ST (структурированного текста) в соответствии с DIN EN 61131-3 (ранее IEC 1131-3).

Расширение возможностей языка

- Динамический вызов таймеров и счетчиков
  При вызове функции таймера или счетчика, Вы можете определить переменную типа данных INT вместо абсолютного числа. При всяком выполнении программы Вы можете присвоить этим переменным другое число и сделать вызов функции динамическим.
- Запись констант с определением типа
  Прежде тип константы определялся только арифметической или логической операцией, в которой она использовалась. Например, в следующей команде '12345' получает тип данных INT:
  Int1:=Int2 + 12345
  Теперь Вы можете назначить тип данных константы явно, используя для константы нотацию, подобную приведенной ниже:
  INT#12345
- Альтернативная интерпретация типа переменной
  Чтобы иметь доступ к объявленной переменной, как к переменной другого типа данных, Вы можете определить несколько переменных разного типа по одному адресу.

Улучшенные функции редактора

- Исправление цветов и стилей в соответствии с синтаксисом
  Использование различных стилей и цветов для различных языковых элементов придает Вашим исходным SCL файлам профессиональный вид.
- Форматирование исходных файлов в соответствии с синтаксисом
  Автоматически устанавливаемые отступы строк увеличивают удобочитаемость исходных файлов SCL
- Возможность пошаговой отмены и восстановления действий при редактировании.
  Команды меню позволяют отменить или восстановить отдельные шаги редактирования.

Расширенные функции печати

- Выбор различных шрифтов и стилей для вашего принтера
  Когда Вы печатаете исходный файл на SCL, Вы можете выбрать стиль, который отличается от стиля, показанного на экране.
- Печать с номерами строк
  Вы можете печатать номера строк или печатать каждый новый блок с новой страницы.

Выборочная компиляция и загрузка

- Выборочная компиляция
  Используя функцию «Компиляция выбранного блока», Вы можете компилировать отдельные выбранные блоки исходного файла SCL , чтобы ускорить работу.
- Выборочная загрузка
  Используя функцию «Выборочная загрузка», Вы можете загружать выбранные блоки из исходного файла.

1.5 Установка программы и полномочий пользователя

Системные требования

S7-SCL V5.1 может работать на программаторах и ПК со следующими системами:

- Microsoft Windows 95/98/2000/NT
- Стандартный пакет STEP 7 V5, Service pack 3 или выше (для любой другой версии указано, что необходимо установить).

Аппаратные требования

Требования для S7-SCL такие же, как для стандартного пакета STEP 7. Дополнительное пространство на жестком диске, которое требуется для S7-SCL V5.1 приведено в файле readme.wri.

Запуск установки программы

S7-SCL имеет программу Setup для автоматической установки с диска программного обеспечения. На экране в окне подсказок Вы найдете описание процесса установки шаг за шагом.

Эти шаги описаны ниже:

1. Откройте панель управления (Control Panel) в Windows 95/98/2000/NT и двойным нажатием выберете иконку Добавить/Переустановить (Add/Remove Programs).
2. Выбор установки (Install).
3. Включите CD и нажмите кнопку "Next (Дальше)". Windows автоматически начнет установку с программы "Setup.exe".
4. Следуя инструкции на экране, установите программу.

Замечания для пользователя

Когда Вы установите программу, установите на своем жестком диске авторизацию (разрешение на использование) S7-SCL. Если авторизации нет, то появится соответствующая надпись. Если Вы хотите, то можете установить авторизацию позже. Для того, чтобы установить авторизацию во время установки, просто вставьте авторизационную дискету, когда на экране отображается приглашение.

Авторизационная дискета

Устанавливая авторизацию, надо иметь дискету (дискета не может быть копирована обычной операцией). Программа "AuthorsW", необходимая для установки разрешения, находится на том же CD ROM S7-SCL V5.1.

Число разрешенных использований определяется счетчиком на дискете. Каждый раз при установке счетчик уменьшает на единицу. Когда счетчик покажет ноль, с дискеты невозможно будет установить разрешение.

Внимание

Читайте инструкции в файле README.WRI в папке AuthorsW. Если Вы не будете следовать инструкциям, разрешение может быть потеряно.

Потеря авторизации

Вы можете потерять авторизацию, например, из-за дефектов диска.

Для восстановления авторизации требуется дискета. С утерянной авторизацией Вы можете пользоваться программой в течение зарегистрированного периода использования. В этом случае, при запуске программы, на экране отображается время использования. В течение этого периода Вам нужно получить замену потерянной авторизации. Чтобы произвести замену авторизации, пожалуйста, обратитесь к Вашему представителю компании SIEMENS.

Замечание

Вы найдете дальнейшие инструкции и правила, имеющие отношение к установке и модификации программного обеспечения в руководстве "Programming with STEP 7 V5.x" ("Программирование на STEP 7 V5.x").

1.6 Замечания о соответствии DIN EN 61131-3

Начиная с версии 5.0, S7-SCL подчиняется уровню PLCopen Basis для ST (структурный текст), в соответствии со стандартом DIN EN 61131-3 (ранее IEC 1131-3).

Если у Вас есть программа ST, Вы можете переместить ее как ASCII файл в STEP 7, используя SIMATIC Manager или используя функции копировать/вставить в редакторе SCL.

Настройки и требования

Вам требуются следующие настройки для создания системного окружения компиляции файлов выполненных в соответствии с стандартом :

- Выберите английскую мнемонику для проекта SIMATIC Manager **Options > Customize > Language** (Параметры>Настройки>Язык).
- В настройках SCL (**Options > Customize > Compiler** (Параметры>Настройки>Компилятор)) отмените опцию "Permit nested comments (разрешение комментариев)".
- Вместо слов FUNCTION_BLOCK / END_FUNCTION_BLOCK (Функциональный блок/ Конец функционального блока), могут быть использованы слова PROGRAM / END_PROGRAM (Программа/ Конец программы)
- Наименованиям программы, функционального блока или функции должно быть присвоены уникальные номера в таблице символов.

Изменения в синтаксисе и семантике

В результате согласования со стандартами в синтаксисе и семантике версии SCL Language Version 5.0 были сделаны следующие изменения:

- Символические имена более не чувствительны к регистру. Для имен из таблицы символов это справедливо начиная с версии STEP 7 V4.02.
- Строки END_VAR, END_CONST, END_LABEL и FUNCTION_BLOCK name, FUNCTION name и т.д. должны завершаться точкой с запятой. Точка с запятой интерпретируются как пустая инструкция, которая следует за текущей инструкцией.
- Списки переменных в графе CASE не должны более сортироваться в порядке возрастания. Однако, если Вы определите диапазон величин в формате "a .. b", должно быть истинно a<= b.
- Переменные типа INT и DINT больше не преобразуются в переменные реального типа при делении (/). Тип данных результата деления (/) теперь определяется типом данных наиболее значимого адреса. Если, например, поделены два целые числа, результат - также целого типа данных (например 10/3=3, поскольку 10.0/3=3.33).

2 Разработка SCL программ

2.1 Добро пожаловать в простейшую программу для начинающих – "Measured Value Acquisition"

Что Вы изучите

Простая программа для начинающих пользователей покажет, как эффективно использовать SCL. У Вас сразу возникают некоторые вопросы, например:

- Как создавать программы на SCL?
- Какие функции SCL пригодны для выполнения задач?
- Какие средства отладки существуют?

На эти и другие вопросы даются ответы в этом разделе.

Использование языковых элементов SCL

Программа вводит следующие языковые функции SCL:

- Структурирование и использование различных типов блоков SCL
- Вызов блока с просмотром параметров и их оценкой
- Различные форматы ввода и вывода
- Программирование с простыми типами данных и массивами
- Инициализация переменных
- Структурирование программы, использование ветвления и циклов

Необходимые аппаратные средства

Для запуска программ на SIMATIC S7-300 или SIMATIC S7-400 Вам нужны следующие устройства:

- Один 16-канальный входной модуль
- Один 16-канальный выходной модуль

Функции отладки

Программа построена так, что Вы можете быстро ее протестировать, задавая тумблерами сигналы, поступающие на входной модуль и наблюдая индикацию на выходном модуле. Для быстрого тестирования используйте возможности отладки SCL.

Кроме того, Вы можете использовать другие функции отладки, предусмотренные стандартным пакетом STEP 7.

2.2 Задача

Обзор

Измеряемые величины будут собираться входным модулем, отсортировываться и обрабатываться программой SCL. Результаты будут отображаться на выходном модуле.

Сбор данных по измеряемым величинам

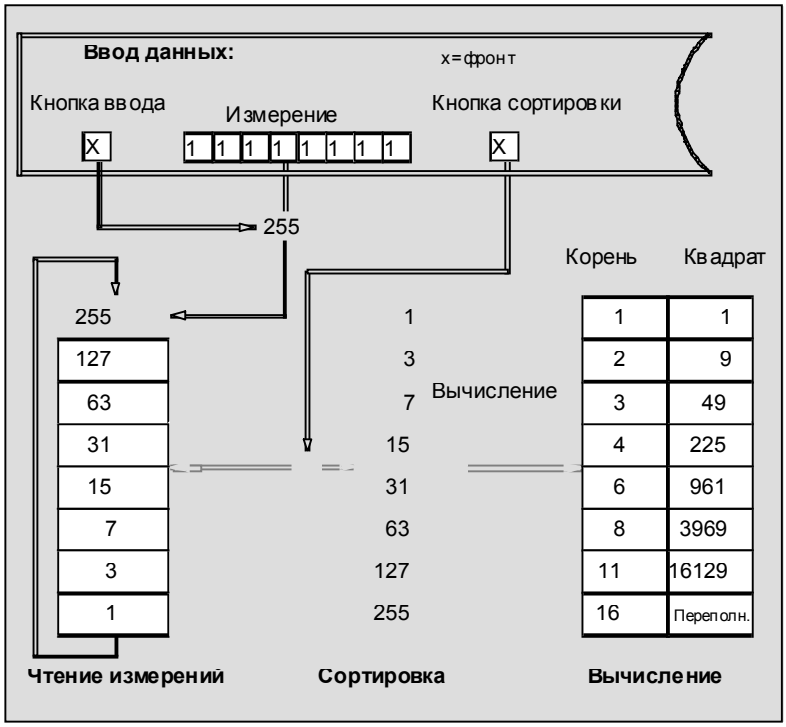
Измеряемая величина устанавливается, используя 8 входных тумблеров. Эта величина считывается в массив памяти при обнаружении фронта сигнала от кнопки ввода (смотрите следующую диаграмму).

Значение величины может быть от 0 до 255. Для нее необходим один байт.

Обработка измеряемой величины

Массив измеряемых величин будет организован как кольцевой буфер на восемь значений.

Когда сигнал поступает в программу сортировки, измеряемая величина записывается в массив в порядке возрастания. Затем вычисляется квадрат и квадратный корень каждого числа. Для хранения результатов обработки необходимо по одному слову.



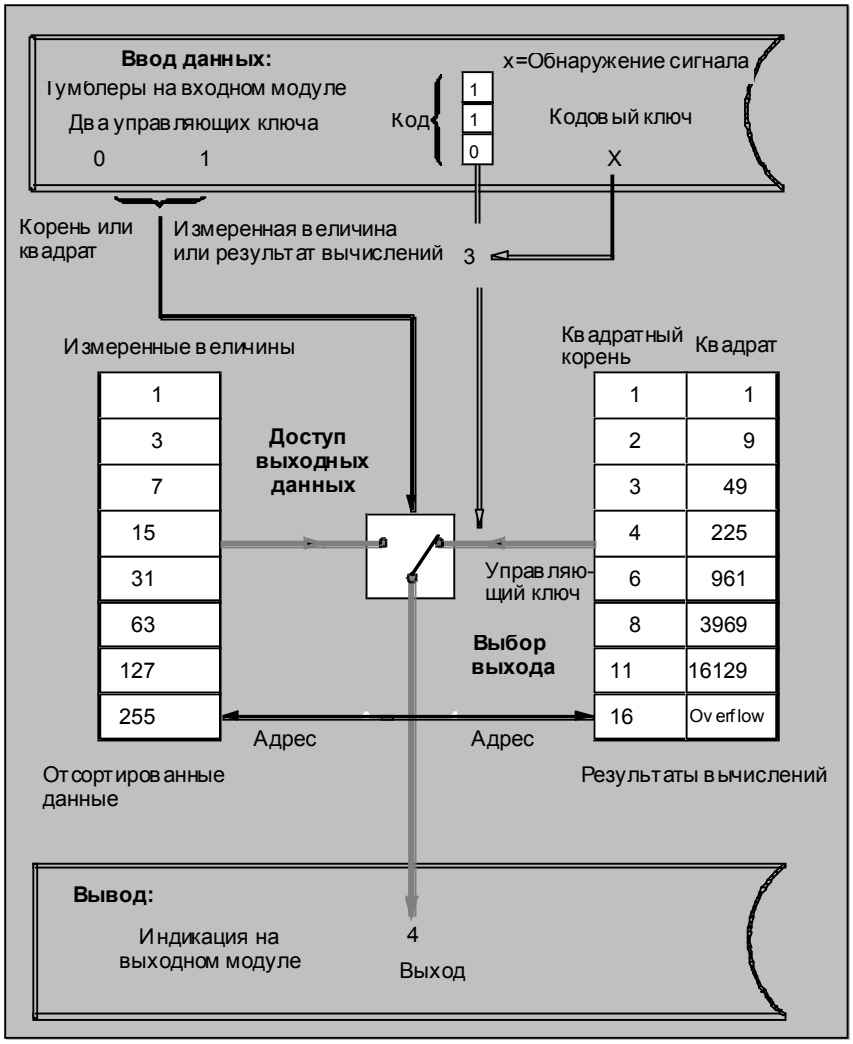
Выбор выходов

На выходном модуле может быть отражена только одна величина. Следовательно, необходимо сделать выбор между:

- элементами списка
- измеряемой величиной, квадратным корнем или квадратом.

Отображаемая величина будет выбираться следующим образом:

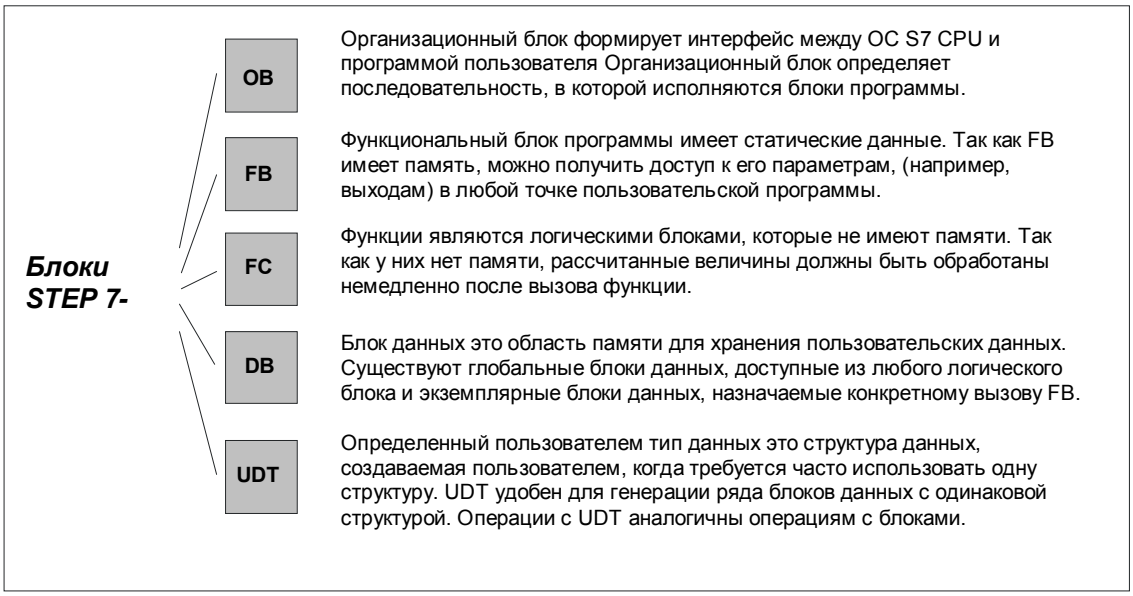
- Используются три переключателя, чтобы получить код, который вводится, если обнаружен сигнал на четвертом кодовом переключателе. Эти три переключателя определяют адрес выводимых данных.
- Одному адресу соответствуют три величины: измеряемая величина, ее квадрат и квадратный корень. Для выбора одной из этих величин, требуются два тумблера.



2.3 Проектирование структуры программы на SCL

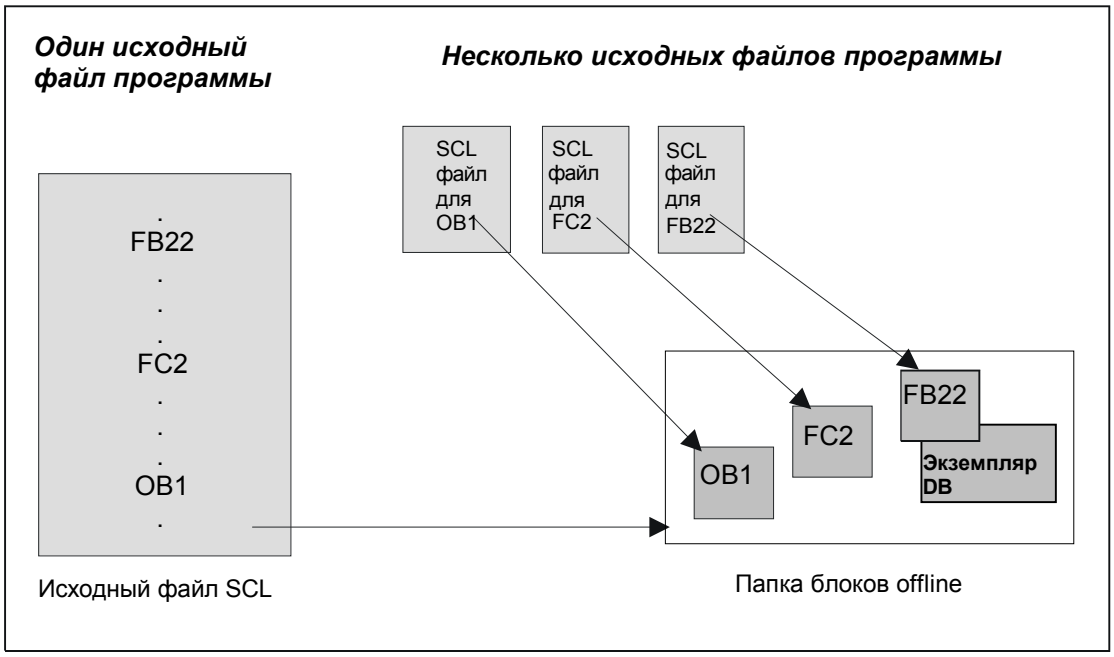
Типы блоков

Наилучшее решение задачи можно получить, используя структурированную **SCL** программу. Это означает использование модульного проектирования, когда программа делится на блоки, каждый из которых ответственен за специфическую задачу. В SCL, как и в других языках программирования STEP 7, Вы имеете следующие типы блоков.



Размещение блоков в исходных файлах SCL

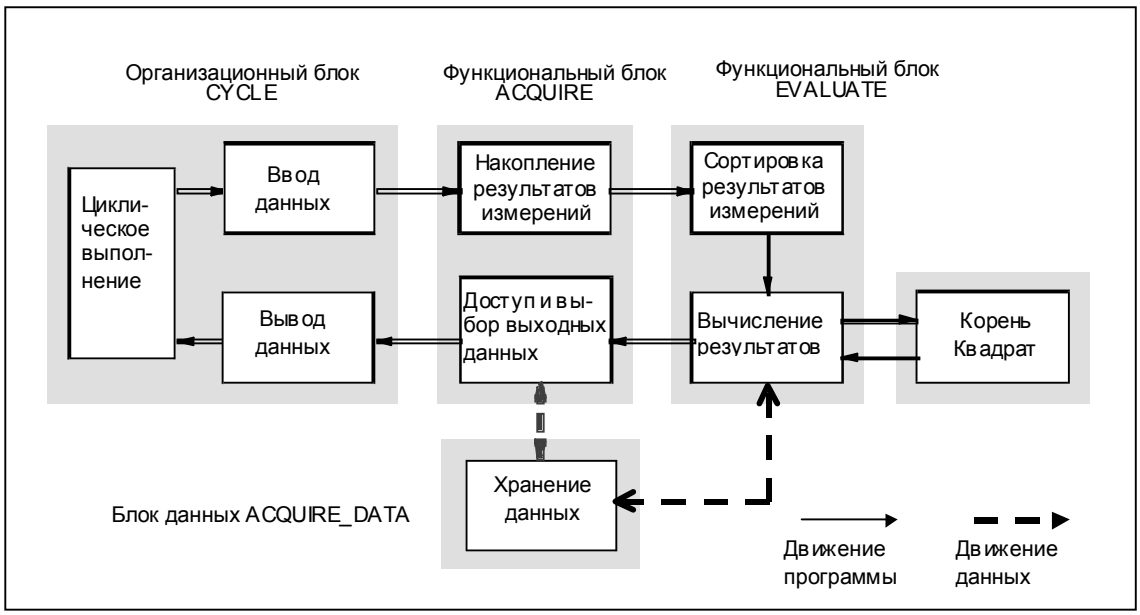
SCL программа состоит из одного или более исходных файлов. Исходный файл может содержать единственный блок или сложную программу из нескольких блоков.



2.4 Определение подзадач

Подзадачи

Подзадачи показаны на приведенной ниже схеме. Прямоугольные затемненные области представляют блоки программы. Логические блоки размещены слева направо в порядке вызова.



Выбор и распределение различных типов блоков

Каждый блок выбирается, исходя из следующих критериев:

Функция		**Название блока**
Пользовательская программа может начинаться только с блока OB. Поскольку измеряемые величины вводятся циклически, требуется использовать циклически вызываемый OB1. Часть программы - *ввод и вывод данных* – программируется в OB.	⇒	OB "Cycle"
Подзадаче *"Накопление результатов измерений"* требуется блок с памятью, другими словами, функциональный блок (FB), так как некоторый локальный блок данных (например, кольцевой буфер) должен сохранятся от одного программного цикла к другому. Место хранения данных (память) - экземплярный блок данных ACQUIRE_DATA. Тот же FB может оперировать *подзадачей доступа и выбора выходных данных*, так как данные находятся в нем.	⇒	FB "Acquire" (Сбор данных)

Функция		**Название блока**
Когда выбирается тип блоков для подзадач *сортировки измеряемых величин и вычисления результатов*, помните, что Вам необходимо сохранять в буфере вывода квадрат и квадратный корень измеряемых величин. Для этого есть только один пригодный блок FB. Поскольку он по иерархии выше, то не требует своего DB. Данные примера в этом случае могут загружаться в блок FB.	⇒	FB "Evaluate " (Вычисление)
Функция (FC) лучше всего подходит для подзадач вычисления квадрата и квадратного корня, результат которых получается как функциональная величина. Более того, никакие данные, используемые при вычислении, не сохраняются более одного программного цикла. Для вычисления квадратного корня используется стандартная функция SCL SQRT. Специальная функция SQUARE (квадрат) создана для вычисления квадрата, чтобы дополнительно проверить, находится ли величина в допустимых пределах.	⇒ ⇒	FC "SQRT" (квадратный корень) и FC "SQUARE" (квадрат)

2.5 Определение интерфейса между блоками

Обзор

Интерфейс блока определяет параметры, которые могут быть доступны другим блокам.

Параметры, объявленные в блоках, получают значение только при вызове блока. Эти параметры называются формальными и получают величину только при указании фактических параметров. При вызове блока формальным параметрам блока назначаются фактические параметры. После возвращения вызванного блока в программу, становятся доступными его выходные данные. Функция (FC) в результате получается как функциональная величина.

Параметры блоков могут быть подразделены на категории, указанные ниже:

Параметры блока	**Объяснение**	**Объявление**
Входные параметры	Параметры ввода принимают фактические значения только при вызове блока. Они только для чтения.	VAR_INPUT
Выходные параметры	При вызове блока передают текущие величины. Данные могут записываться и читаться.	VAR_OUTPUT
Вход/выходные параметры	Вход/выходные параметры принимают фактическую величину переменной, когда блок вызван, обрабатывают величину и возвращают результат в оригинальную переменную.	VAR_IN_OUT

Цикл OB

Цикл OB не имеет собственных формальных параметров. Он вызывает FB ACQUIRE и передает ему измеряемую величину и данные управления для его формальных параметров.

Сбор данных (FB ACQUIRE)

Имя параметра	Тип данных	Тип объявления	Описание
measval_in	INT	VAR_INPUT	Измеряемая величина
newval	BOOL	VAR_INPUT	Кнопка для ввода измеряемой величины в кольцевой буфер
resort	BOOL	VAR_INPUT	Ключ для сортировки и пересчетов измеряемых данных
funct_sel	BOOL	VAR_INPUT	Тумблер выбора квадратного корня или квадрата
selection	WORD	VAR_INPUT	Код для выбора выходной величины
newsel	BOOL	VAR_INPUT	Кнопка для чтения кода
result_out	DWORD	VAR_OUTPUT	Выход вычисленного результата
measval_out	DWORD	VAR_OUTPUT	Выход измеряемой величины

Нахождение численного значения (FB EVALUATE)

ACQUIRE FB вызывает EVALUATE FB. Данные в массиве измеряемых величин требуют сортировки. Следовательно, этот массив объявлен как вход/выходной параметр. Массив структур создается как параметр вывода для вычисленного результата квадратного корня и квадрата. Следующая таблица показывает формальные параметры:

Имя	Тип данных	Тип объявления	Описание
sortbuffer	ARRAY[..] OF REAL	VAR_IN_OUT	Массив измеряемой величины в виде кольцевого буфера
calcbuffer	ARRAY[..]OF STRUCT	VAR_OUTPUT	Массив результатов: Структура «квадратный корень» и «квадрат» из компонентов целого типа

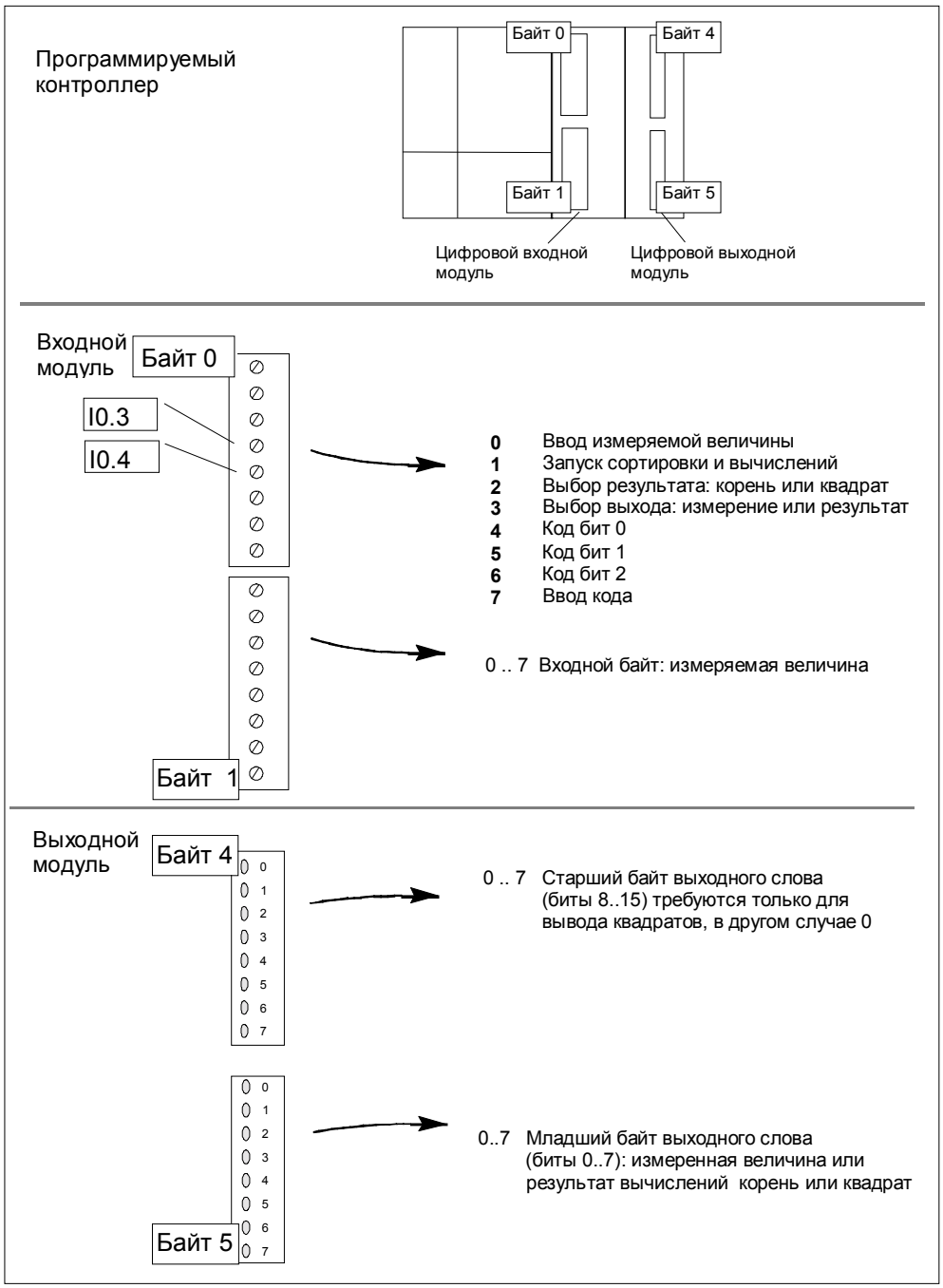
Квадратный корень и квадрат (FC SQRT и FC SQUARE)

Эти функции вызываются функцией EVALUATE. Они требуют ввода величины (аргумента) и возвращают результат как функциональную величину.

Наименование	Тип данных	Тип объявления	Описание
Value	REAL	VAR_INPUT	Ввод квадратного корня
SQRT	REAL	Function value	Квадратный корень вводимой величины
Value	INT	VAR_INPUT	Ввод квадрата
SQUARE	INT	Function value	Квадрат вводимой величины

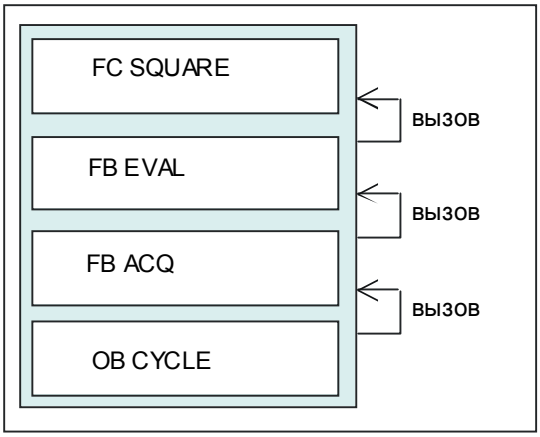
2.6 Определение интерфейса ввода/вывода

Схема, показанная ниже, - интерфейс ввода/вывода. Имейте в виду, что в пределах байта более низкий порядок имеет верхний бит, а более высокий - нижний. Порядок следования байт в словах противоположный – верхний байт содержит старшие разряды, а нижний – младшие.



2.7 Порядок следования блоков в исходном файле

Размещая блоки в исходном файле SCL, запомните, что блок должен появиться до того, как Вы его будете использовать; другими словами, вначале приводится вызываемый, а затем вызывающий блок. Блоки должны размещаться в исходном файле так, как показано ниже:



2.8 Определение символики

Используя символику для адресов модулей и блоков, Вы облегчите работу над программой. До использования символики в программе, Вы должны включить символические имена в таблицу символов.

Ниже приводится таблица символов, используемых в нашей программе. Она содержит символические имена, которые Вы объявляете в символьной таблице, чтобы исходный файл мог быть скомпилирован без ошибок:

	Symbol	Address	Data type
1	Coding	IW 0	WORD
2	Coding switch	I 0.7	BOOL
3	CYCLE	OB 1	OB 1
4	Entry	IB 1	BYTE
5	EVALUATE	FB 20	FB 20
6	Function switch	I 0.2	BOOL
7	Input 0.0	I 0.0	BOOL
8	Output	QW 4	INT
9	Output switch	I 0.3	BOOL
10	ACQUIRE	FB 10	FB 10
11	ACQUIRE_DATA	DB 10	FB 10
12	Sorting switch	I 0.1	BOOL
13	SQUARE	FC 41	FC 41

2.9 Создание функции вычисления квадрата

2.9.1 Раздел операторов функции вычисления квадрата

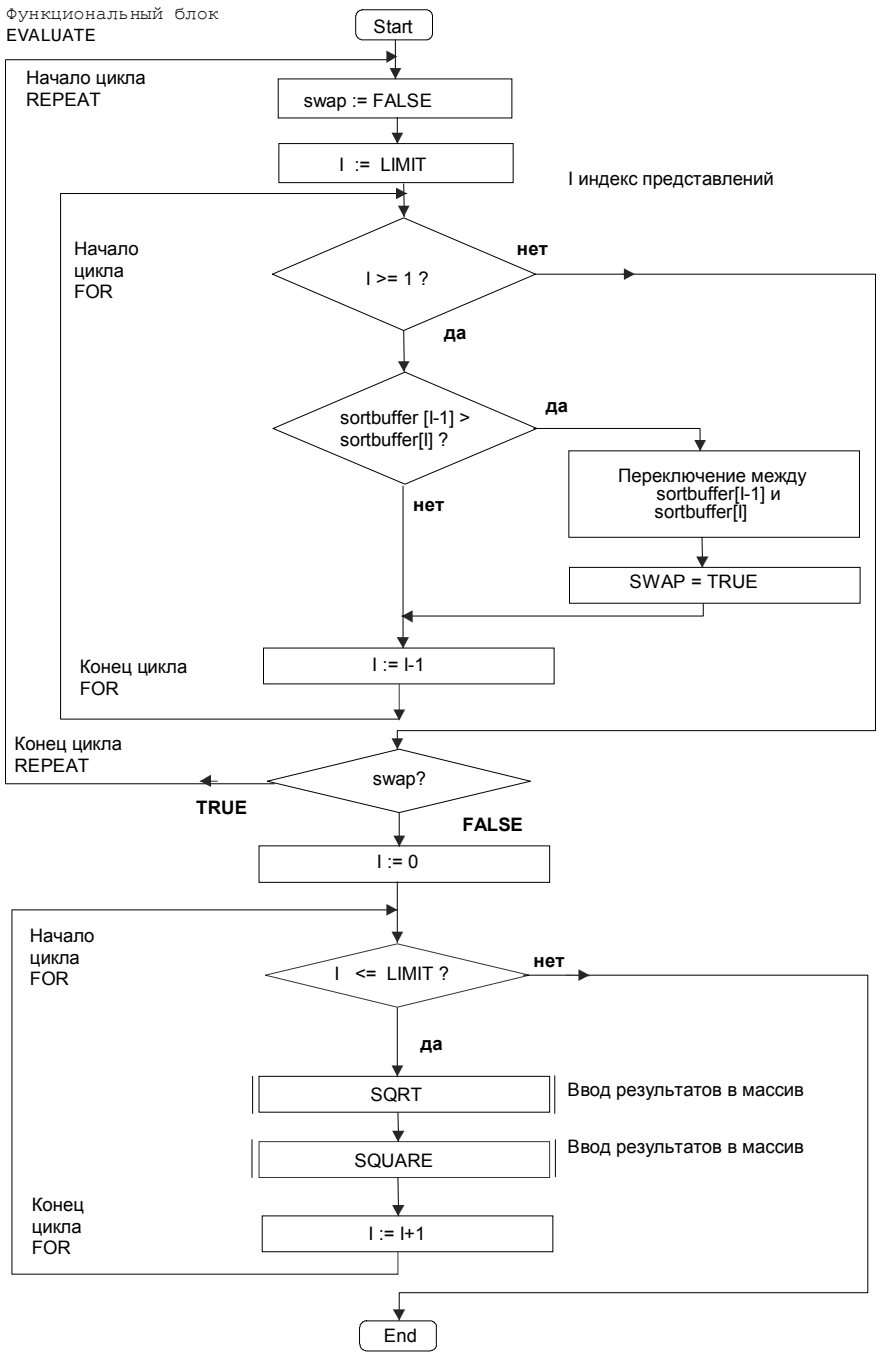
Раздел операторов

Сначала программа проверяет, превышает ли результат числовой диапазон. Если это так, то подставляется максимальная целая величина. В противном случае, вычисляется квадрат. Результат возвращается как величина функции.

```
FUNCTION SQUARE : INT
 VAR_INPUT
   value : INT;
END_VAR
BEGIN
IF value <= 181 THEN
   SQUARE   := value * value; //Вычисление функции
ELSE
   SQUARE   := 32_767; // При переполнении берется
                       // максимальное значение
END_IF;
END_FUNCTION
```

2.10 Создание функционального блока вычисления

2.10.1 Блок-схема алгоритма функционального блока вычисления



2.10.2 Раздел деклараций функционального блока вычисления

Структура раздела деклараций

Раздел деклараций этого блока состоит из следующих подразделов:

- Константы: между CONST и END_CONST.
- Параметры ввода/вывода между VAR_IN_OUT и END_VAR.
- Параметры вывода: между VAR_OUTPUT и END_VAR.
- Временные переменные: между VAR_TEMP и END_VAR.

```
CONST
   LIMIT := 7;
END_CONST

VAR_IN_OUT
   sortbuffer : ARRAY[0..LIMIT] OF INT;
END_VAR

VAR_OUTPUT
   calcbuffer  : ARRAY[0..LIMIT] OF
      STRUCT
         squareroot  : INT;
         square      : INT;
      END_STRUCT;
END_VAR

VAR_TEMP
   swap        : BOOL;
   index, aux   : INT;
   valr, resultr: REAL ;
END_VAR
```

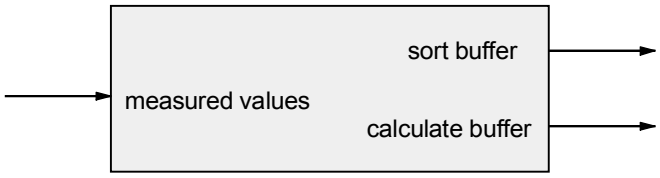
2.10.3 Раздел операторов функционального блока вычисления

Последовательность программы

Параметр ввода/вывода “sortbuffer" (буфер сортировки) связывается с кольцевым буфером “measvals" так, что оригинальное содержание буфера переписывается, сортируя измеряемые величины.

Новый массив “calcbuffer" (буфер вычисления) создается как выходной параметр с вычисленными результатами. Структура этих элементов такова, что они содержат квадратный корень и квадрат каждой измеряемой величины.

На схеме ниже показаны связи между массивами.



EVALUATION

Этот интерфейс показывает суть обмена данных для обработки измеренных величин. В этом случае, данные загружены в блок данных ACQUIRE_DATA, поскольку локальный пример для функционального блока EVALUTION создается в результате вызова блока ACQUIRE.

Раздел операторов блока EVALUATION

Во-первых, измеряемые величины в кольцевом блоке сортируются и затем вычисляются.

- Алгоритм сортировки
  Для сортировки в буфере используется метод постоянного обмена величин. Это значит, что величины последовательно сравниваются и перемещаются, пока не получен конечный результат. Используемый буфер является параметром ввода/вывода "sortbuffer" (буфер сортировки).
- Начало вычислений
  Как только сортировка завершена, выполняется цикл, в котором применяются функции SQUARE для возведения в квадрат и SQRT для извлечения квадратного корня. Эти результаты хранятся в структурном массиве "calcbuffer" (буфер вычисления).

Раздел операторов блока вычисления

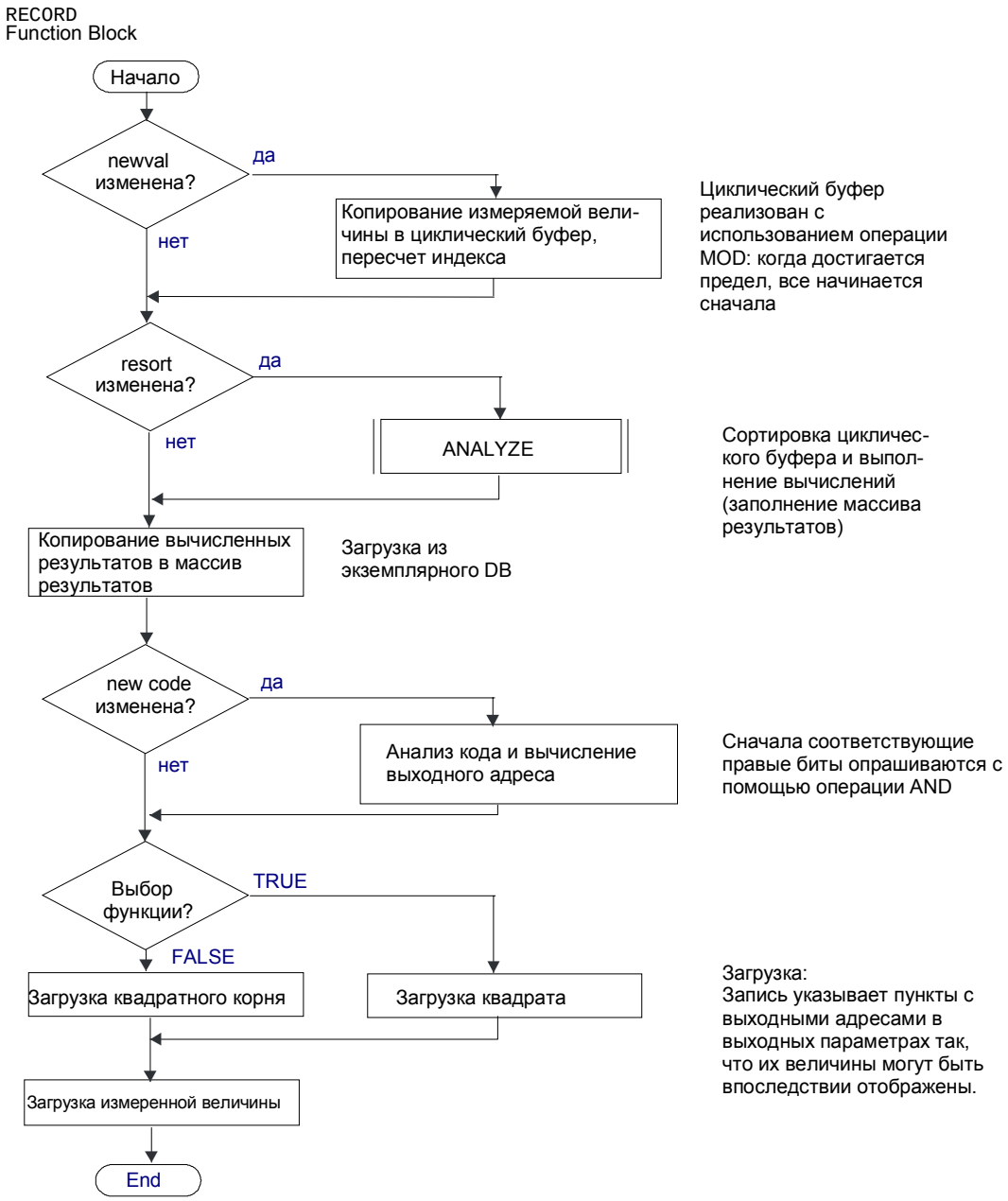
Этот раздел состоит из следующих логических блоков:

```
BEGIN
(*******************************************************
Часть 1 Сортировка : Применяется метод "пузырьковой сортировки": Замена пар
переменных пока измеряемые величины можно сортировать.
*******************************************************)
REPEAT
   swap := FALSE;
      FOR index := LIMIT TO 1 BY -1 DO
         IF sortbuffer[index-1] > sortbuffer[index]
            THEN  aux   :=sortbuffer[index];
            sortbuffer[index]     := sortbuffer[index-1];
            sortbuffer[index-1]   := aux;
            swap                  := TRUE;
         END_IF;
      END_FOR;
   UNTIL NOT swap
END_REPEAT;
(*********************************************************
Часть 2  Вычисление : Квадратный корень по стандартной функции SQRT и квадрат
по функции SQUARE.
*********************************************************)
FOR index := 0 TO LIMIT BY 1 DO
  valr   := INT_TO_REAL(sortbuffer[index]);
  resultr := SQRT(valr);
  calcbuffer[index].squareroot  := REAL_TO_INT(resultr);
  calcbuffer[index].square := SQUARE(sortbuffer[index]);
END_FOR;
END_FUNCTION_BLOCK
```

2.11 Создание функционального блока ACQUIRE

2.11.1 Диаграмма блока ACQUIRE

Следующая схема показывает алгоритм в форме диаграммы:



2.11.2 Раздел операторов функционального блока ACQUIRE

Структура раздела

Структура раздела состоит из следующих подразделов:

- Константы: между CONST и END_CONST.
- Параметры ввода: между VAR_INPUT и END_VAR.
- Параметры вывода: между VAR_OUTPUT и END_VAR.
- Статические переменные: между VAR и END_VAR. Они также включены в описание локального примера для блока Вычисления

```
CONST
   LIMIT := 7;
   QUANTITY := LIMIT + 1;
END_CONST
VAR_INPUT
 measval_in  :  INT ;   // Новая измеренная величина
 newval  : BOOL; // Измеренная величина в циклическом буфере
                          //  "measvals"
 resort  : BOOL; // Сортировка измеренных величин
 funct_sel   :  BOOL;  // Выбор вычисленного результата
                                    //  корень/квадрат
    newsel     :  BOOL; // Введенный выходной адрес
    selection   :  WORD; // Адрес выхода
   END_VAR
   VAR_OUTPUT
    result_out  :  INT;  // вычисленная величина
    measval_out  :  INT; // соответствующая измеренная
                                     //  величина
   END_VAR
   VAR
    measvals        :  ARRAY[0..LIMIT] OF INT := 8(0);
    resultbuffer     :  ARRAY[0..LIMIT] OF
    STRUCT
     squareroot     :  INT;
     square        :  INT;
    END_STRUCT;
    pointer         :  INT    := 0;
    oldval          :  BOOL  := TRUE;
    oldsort         :  BOOL  := TRUE;
    oldsel          :  BOOL  := TRUE;
    address  :  INT := 0;//Преобразование адреса выхода
    outvalues_instance: EVALUATE;//Определение локального
                                          //экземпляра
END_VAR
```

Статические переменные

Такой тип функционального блока выбран потому, что некоторые данные должны сохраняться от одного программного цикла к другому. Это статические переменные, описанные в подразделе “VAR, END_VAR".

Статические переменные это локальные переменные, величины которых сохраняются во время обработки каждого блока. Они используются для хранения величин в функциональном блоке и, например, в блоке данных.

Инициализация переменных

Имейте ввиду, что инициализация переменных проводится, когда инициализирован блок (после загрузки на CPU). Локальный экземпляр для блока EVALUATION также описан в подразделе“VAR, END_VAR". Впоследствии это наименование используется для вызова и доступа параметров вывода. Для хранения данных используется глобальный экземпляр ACQUIRE_DATA.

Наименование	**Тип данных**	**Инициализация переменных**	**Описание**
Measvals	ARRAY [..] OF INT	8(0)	Кольцевой буфер для измеряемых величин
Resultbuffer	ARRAY [..] OF STRUCT	-	Массив для структуры с компонентами квадрат и квадратный корень целого типа
Index	INT	0	Индекс для кольцевого буфера, опознающего позицию для следующей измеренной величины
Oldval	BOOL	FALSE	Предыдущая величина для чтения измеряемой величины "newval"
Oldsort	BOOL	FALSE	Предыдущая величина для сортировки результатов "resort"
Oldsel	BOOL	FALSE	Предыдущая величина для чтения кода "newsel"
Address	INT	0	Адрес для вывода измеряемой величины или результата
eval_instance	Local instance	-	Локальный пример для блока EVALUTION

2.11.3 Раздел операторов функционального блока ACQUIRE

Структура раздела

Операторы данного раздела делятся на три части:

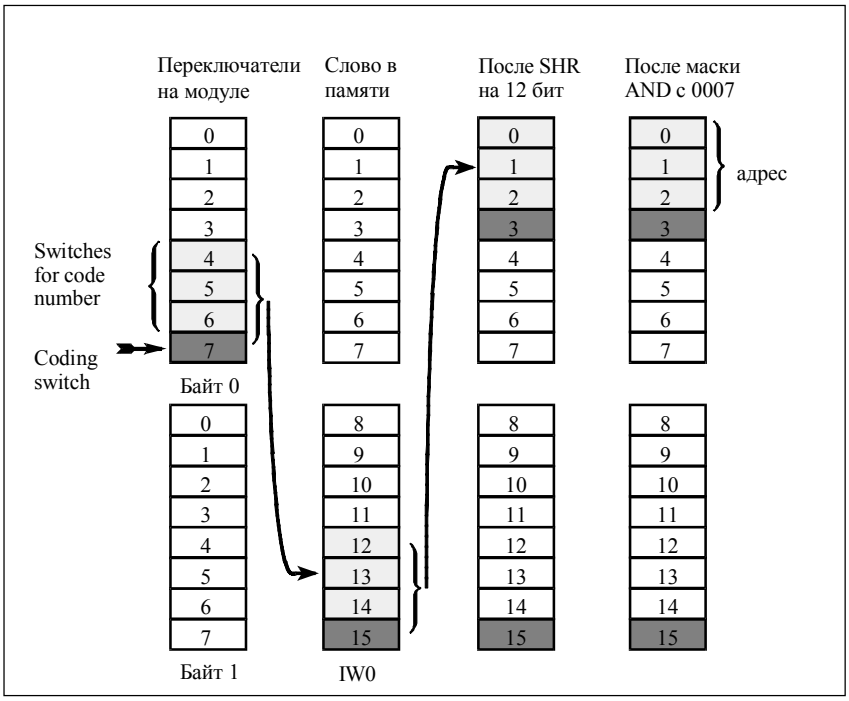
- Обработка измеряемых величин
  Если параметр ввода "newval" отличается от "oldval", новая измеряемая величина читается в кольцевом буфере.
- Начало сортировки и вычисления
  Сортировка и вычисление начинаются при вызове функционального блока Вычисления, когда параметр ввода "resort" меняется по сравнению с "oldsort".
- Вычисление кодирования и подготовка данных вывода
  Кодирование читается слово за словом. Согласно условным обозначениям SIMATIC, это значит, что верхняя группа переключателей (байта 0) содержит более высокий порядок восемь битов слова ввода и низкую группу переключателей (байта 1) с более низким порядком битов. Схема, расположенная ниже, показывает размещение кодовых переключателей.

Вычисление адресов

На схеме, расположенной ниже, показано, как вычисляются адреса: биты с 12 по 14 слова ввода IW0 содержат код, который читается, когда на переключателе кодов обнаруживается фронт сигнала (бит 15). Адрес получается при сдвиге вправо, используя стандартную функцию SHR и скрывая важные биты AND.

Этот адрес используется для написания массива элементов параметров вывода (вычисленные результаты и соответствующие измеряемые величины). Независимо от того, квадрат ли это или квадратный корень, вывод зависит от “funct_sel".

Фронт сигнала переключателя кодирования обнаруживается, когда “Newsel" отличается от “oldsel".



Раздел операторов

Этот раздел состоит из следующих логических блоков:

```
BEGIN
(*********************************************************
Часть 1 : Получение измеренных величин. Если переменная
"newval" изменена, вводится измеренная величина. Операция MOD
используется для реализации кольцевого буфера измеренных
величин.
*********************************************)
IF newval <> oldval THEN
   pointer         := pointer MOD QUANTITY;
   measvals[pointer]  := measval_in;
   pointer         := pointer + 1;
END_IF;
oldval := newval;
(*********************************************************
Часть 2 : Начало сортировки и вычислений
Если изменена переменная "resort", начинается сортировка
кольцевого буфера и выполнение вычислений с результатами
измерений. Результаты хранятся в новом массиве "calcbuffer".
*********************************************************)
IF resort <> oldsort THEN
  pointer := 0;     //Сброс указателя кольцевого буфера
  eval_instance(sortbuffer := measvals); //Вызов EVALUATE
END_IF;
oldsort     := resort;
resultbuffer := eval_instance.calcbuffer;//Квадрат и
                                                  //квадратный корень

(*********************************************************
Часть 3 : Анализ кодов и подготовка вывода: Если переменная
"newsel" изменилась, код для адресации элемента массива для
вывода переопределяется: Соответствующие биты  переменной
"selection" выделяются  маской и преобразуются в целое. Выход
выбирается в зависимости от установки ключей "funct_sel"  и
"squareroot"/"square".
*********************************************************)
IF newsel <> oldsel THEN
   address := WORD_TO_INT(SHR(IN := selection, N := 12) AND 16#0007);
END_IF;
oldsel := newsel;
IF funct_sel THEN
   result_out := resultbuffer[address].square;
ELSE
   result_out := resultbuffer[address].squareroot;
END_IF;
measval_out  := measvals[address];//Вывод измеренной величины
END_FUNCTION_BLOCK
```

2.12 Создание циклического организационного блока

Задачи циклического OB

OB1 выбирается, так как он вызывается циклически. Он выполняет следующие задачи в программе:

- Вызывает и поставляет блоку ACQUIRE данные ввода и данные управления.
- Читает данные, которые возвращает блок ACQUIRE.
- Выводит величины на дисплей

В начале раздела деклараций расположен массив из 20 байт временных данных “systemdata".

Программный код циклического OB

```
ORGANIZATION_BLOCK CYCLE
(*********************************************************
 CYCLE - название OB1, он вызывается циклически системой S7.
Часть 1 : Вызов функционального блока и передача входных величин
Часть 2 : Чтение выходных величин и вывод с переключением выходов
*********************************************************)
VAR_TEMP
   systemdata  :  ARRAY[0..20] OF BYTE; // Область для OB1
END_VAR
BEGIN
(* Part 1 : ***************************************************)
ACQUIRE.ACQUIRE_DATA(
  measval_in:=  WORD_TO_INT(input),
  newval       :=  "Input 0.0",  //Входной ключ как идентификатор сигнала
  resort       :=  Sort_switch,
  funct_sel:=  Function_switch,
  newsel       :=  Coding_switch,
  selection       :=  Coding);

(* Part 2 : ***************************************************)
IF Output_switch THEN                                  //Изменение выхода
  Output    :=  ACQUIRE_DATA.result_out;      //Квадратный корень или квадрат
ELSE
  Output    :=  ACQUIRE_DATA.measval_out;    //Измеряемая величина
END_IF;
END_ORGANIZATION_BLOCK
```

Преобразование типа данных

Измеряемые величины вводятся как данные байтового типа. Они должны быть преобразованы в данные целого типа. Они требуют преобразования из WORD в INT (первичное преобразование из типа BYTE в WORD делается компилятором). Вывод не требует никаких преобразований типов, так как в таблице символов объявлен целый тип данных.

2.13 Тестирование данных

Требования

Чтобы выполнить тест, Вам требуется входной модуль с начальным адресом 0 и выходной модуль с начальным адресом 4.

Перед выполнением теста, поверните все восемь переключателей верхней группы влево (“0") и все переключатели нижней группы вправо (“1").

Переключите блоки на CPU, так как начальные величины переменных должны быть протестированы.

Процедура тестирования

Запустите тест как показано в таблице .

Тест	Действия	Результат
1	Установите код в "111" (I0.4, I0.5 и I0.6) и введите его переключателем кода (I0.7).	Все на модуле вывода активизируется и зажгутся светодиоды.
2	Отобразите квадратный корень, повернув переключатель вывода с (I0.3) на "1".	Светодиоды на модуле вывода покажут бинарный номер "10000" (=16).
3	Отобразите квадрат, повернув функциональный переключатель с (I0.2) на "1".	15 светодиодов зажгутся на модуле вывода. Они покажут переполнение, если результат 255 x 255 выше диапазона целых.
4a	Верните переключатель вывода (I0.3) на "0".	Измеряемая величина снова отобразится. Все светодиоды на выходах байта низкого порядка будут установлены.
4b	Установите величину 3 (бинарную "11") как новую измеряемую величину при вводе.	Выходные данные на этом этапе не изменятся.
5a	Проверьте чтение измеряемой величины: Установите код в "000" и введите переключателем кода (I0.7) так, что позже Вы увидите введенную величину.	На модуле вывода 0; т.к. ни один из светодиодов не включен.
5b	Переключите входной тумблер «Вход 0.0" (I0.0). Это прочтение величины на 4 этапе теста.	На выходе отображена величина 3, двоичная "11".
6	Начните сортировку и вычисление, переключив ключ сортировки (I0.1).	На выходе будет снова 0, так как в процессе сортировки измеряемые величины переме-щаются на более высокую позицию массива.
7	На дисплее после сортировки: Установите код "110" (I0.6 = 1, I0.5 = 1, I0.4 = 0 IB0; переписывая на бит 14, бит 13 и бит 12 IW0) и прочтите его, поворачивая переключатель кода.	На выходе показана величина "11" как вторая величина массива.
8a	Переписанный результат будет следующим: Поворачивая переключатель вывода (I0.3) отобразится квадрат измеряемой величины на 7 шаге.	Отобразится выходная величина 9 (бинарная "1001").
8b	Переключите функциональный ключ (I0.2), чтобы получить квадратный корень.	Отобразится выходная величина 2 (двоичная "10").

Дополнительный тест

Следующая таблица описывает переключатели на модуле ввода и примеры квадрата и квадратного корня. Это описание поможет Вам в тестировании:

- Ввод производится переключателями. Вы можете контролировать программу верхними 8 переключателями и устанавливать измеряемые величины 8 нижними.
- Выходные данные показываются светодиодами. Верхняя группа отображает старший выходной байт, нижняя – младший выходной байт.

Переключатель	**Название параметра**	**Описание**
Канал 0	Переключатель ввода	Переключает для чтения измеряемой величины
Канал 1	Переключатель сортировки	Начало сортировки/вычисления
Канал 2	Функциональный переключатель	Влево ("0"): Квадратный корень, Вправо ("1"): квадрат
Канал 3	Переключатель вывода	Влево ("0"): Измеряемая величина, Вправо ("1"): Результат
Канал 4	Код	Адрес данных на выходе бит 0
Канал 5	Код	Адрес данных на выходе бит 1
Канал 6	Код	Адрес данных на выходе бит 2
Канал 7	Переключатель кода	Переключает для ввода кода

Следующая таблица содержит 8 примеров отсортированных измеряемых величин.

Вы можете вводить величины в любом порядке. Установите комбинации битов для каждой величины и передайте их переключателем ввода. Как только все величины будут введены, начните сортировку и вычисление поворотом переключателя сортировки. Затем Вы можете просмотреть отсортированные величины и результаты (квадратный корень или квадрат.

Измеряемая величина	**Квадратный корень**	**Квадрат**
0000 0001 = 1	0, 0000 0001 = 1	0000 0000, 0000 0001 = 1
0000 0011 = 3	0, 0000 0010 = 2	0000 0000, 0000 1001 = 9
0000 0111 = 7	0, 0000 0011 = 3	0000 0000, 0011 0001 = 49
0000 1111 = 15	0, 0000 0100 = 4	0000 0000, 1110 0001 = 225
0001 1111 = 31	0, 0000 0110 = 6	0000 0011, 1100 0001 = 961
0011 1111 = 63	0, 0000 1000 = 8	0000 1111, 1000 0001 = 3969
0111 1111 = 127	0, 0000 1011 = 11	0011 1111, 0000 0001 = 16129
1111 1111 = 255	0, 0001 0000 = 16	0111 111, 1111 1111 = Переполнение!

3 Использование SCL

3.1 Запуск программы SCL

Запуск из интерфейса Windows

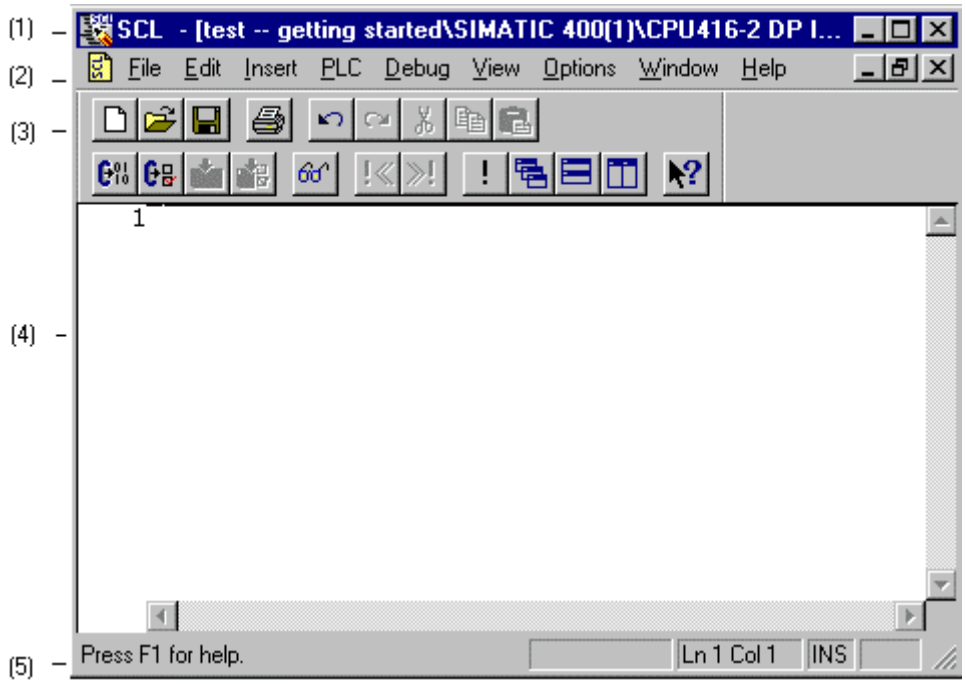
После установки программного обеспечения SCL на Вашем программаторе или ПК, Вы можете запустить программу SCL, используя кнопку Start (Пуск) в панели задач Windows (вход через “SIMATIC / STEP7").

Запуск из SIMATIC Manager

Самый быстрый путь запуска SCL - два раза щелкнуть мышью на иконке исходного файла SCL в SIMATIC Manager.

3.2 Пользовательский интерфейс

Окно SCL имеет следующие стандартные компоненты:



1. **Заголовок окна:**
   Содержит названия программы, проекта, станции, файла и кнопки управления.
2. **Строка меню**:
   Показывает все меню команд для открытого окна.
3. **Панель инструментов**:
   Содержит кнопки часто используемых команд.
4. **Рабочая область**:
   Содержит одно или несколько окон, в которых Вы можете редактировать текст программы, читать информацию компиляции или данные отладки
5. **Строка состояния**:
   Отображает состояние и другую информацию об активном объекте

3.3 Настройка интерфейса пользователя

Настройка редактора

Чтобы сделать установки редактора, выберите в меню команду **Options > Customize (Параметры > Настройка)** и закладку "Editor (Редактор)" в окне "Customize (Настройка)". На этой закладке есть следующие установки:

Настройки на закладке "Editor (Редактор)"	**Объяснение**
Fonts (Шрифты)	Определяет шрифт для исходного текста.
Tabulator length (Длина табуляции)	Определяет ширину колонки табулятора.
Display line numbers (Показ номеров строк)	Показывает в начале строки ее номер.
Save before compiling (Сохранение перед компиляцией)	Спрашивает перед компиляцией, не хотите ли Вы сохранить исходный файл.
Confirm before saving (Подтверждение перед сохранением)	Спрашивает подтверждение перед сохранением.

Адаптация стиля и цвета

Чтобы изменить стиль и цвет различных языковых элементов, выберите в меню команду **Options > Customize (Параметры > Настройка)** и закладку "Format (Формат)" в окне "Customize (Настройка)". Вы можете выбрать следующие установки:

Параметры в графе "Format"	**Объяснение**
Keywords in uppercase (Ключевые слова в верхнем регистре)	Ключевые слова SCL, такие как FOR, WHILE, FUNCTION_BLOCK, VAR или END_VAR, пишутся прописными буквами
Indent keywords (Отступ по ключевым словам)	Автоматическое форматирование отступов строк в разделах деклараций и в командах IF, CASE, FOR, WHILE и REPEAT.
Indent automatically (Автоматический отступ)	Новая строка автоматически делает такой же отступ, как у предыдущей. Эта установка применима только к новым строкам.
Style/Color (Стиль/Цвет)	Вы можете выбрать стиль и цвет для различных языковых элементов.

Эти установки действуют, если на закладке "Format (Формат)" выбрана опция "Use following Formats (Использовать следующее форматирование)".

Панель инструментов, панель контрольных точек, строка состояния

Вы можете включить или выключить изображение панели инструментов, панели контрольных точек или строки состояния. Просто выберите или отмените соответствующую команду в меню **View (Вид)**. Когда панель активирована, команда меню отмечена галочкой.

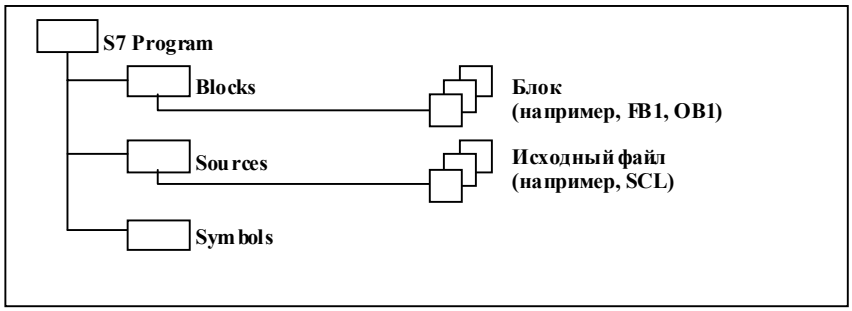
3.4 Создание и обработка исходного файла SCL

3.4.1 Создание нового исходного файла SCL

Перед написанием SCL программы, Вы должны создать новый исходный файл. Он создается в папке исходных файлов программы S7.

Структура программы S7 в SIMATIC Manager

Исходный файл, созданный в программе SCL, может быть интегрирован в структуру программы S7, как показано ниже:



Для этого проделайте следующие шаги:

1. Выберите команду меню "New (Новый)":
   - Нажмите кнопку "New (Новый)" на панели инструментов или
   - Выберите команду **File > New (Файл > Новый)**.
2. Командой "New (Новый)" выбирается
   - Проект
   - Установка фильтра (типа создаваемого файла) "SCL Source File (исходный файл SCL)" и
   - Папка исходных файлов в программе S7
3. Занесите имя исходного объекта в текстовый блок. Имя может иметь длину до 24 символов.
4. Нажмите "OK".

Исходный объект создается с выбранным именем и отображается в рабочем окне.

Замечание

Вы также можете создать исходный файл в SIMATIC Manager, выбрав папку исходных файлов, используя команду меню **Insert > S7 Software > SCL Source File (Вставить>Программы S7>Исходные файлы SCL)**.

3.4.2 Открытие исходного файла SCL

Вы можете открыть исходный файл для его компиляции или редактирования.

Проделайте следующие шаги:

1. Вызовите диалоговое окно "Open (Открыть)":
   - Нажмите кнопку "Open" на панели инструментов или
   - Выберите в меню команду **File > Open (Файл > Открыть)** .
2. Когда диалоговое окно будет открыто, выберите:
   - Необходимый проект
   - Необходимую программу S7
   - Соответствующую папку исходного файла
3. Выберите исходный файл SCL .
4. Нажмите кнопку "OK".

Замечание

Вы можете также открыть исходный файл SCL в SIMATIC Manager двойным щелчком на его иконке или в меню командой **Edit > Open Object (Файл > Открыть объект)**.

3.4.3 Открытие блоков

Невозможно открыть откомпилированные блоки, созданные приложением SCL. Однако, эти блоки можно открыть редактором LAD/STL/FBD и редактировать на языке программирования STL. Тем не менее, не следует модифицировать эти блоки в STL по следующим причинам:

- Отображаемые команды MC7 не обязательно представляют достоверный блок STL.
- Безошибочная компиляция компилятором STL включает модификации, которые требуют тщательного знания как STL, так и SCL..
- Блоки, компилированные STL, получают языковой идентификатор STL, языковый идентификатор SCL будет утерян.
- Исходный файл SCL и MC7 код не будут соответствовать друг другу.

Дальнейшая информация доступна в контекстной помощи STEP 7.

Замечание

Легче поддерживать программы CPU, делая изменения в исходном файле SCL, а затем снова его компилируя.

3.4.4 Закрытие исходного файла SCL

Выполните следующие шаги:

- Выберите в меню команду **File > Close (Файл > Закрыть)**.
- Нажмите кнопку "Close (Закрыть)" в заголовке окна.

Замечание

Если исходный файл был модифицирован, то перед его закрытием спрашивается, хотите ли Вы сохранить изменения. Если Вы не сохраните изменения, они будут утеряны.

3.4.5 Определение свойств объекта

Вы можете определить свойства программы, присваивая блокам атрибуты. Вы можете выбрать свойства исходного файла SCL (например, автора) в диалоговой панели "Properties (Свойства)" .

Проделайте следующие шаги:

1. Выберите в меню команду **File > Properties (Файл > Свойства)**.
2. Введите в панели "Properties (Свойства)" выбранную Вами опцию.
3. Нажмите "OK".

3.4.6 Создание исходного файла в стандартном редакторе

Для редактирования исходного файла SCL Вы можете также использовать стандартный редактор ASCII. Если Вы выберите этот метод, будут недоступны функции помощи SCL.

После того, как Вы создадите исходный файл и сохраните его, Вы должны переслать его в папку исходных файлов программы S7, используя SIMATIC Manager (смотрите документацию STEP 7). Следуя ей, Вы можете открыть исходный файл SCL и продолжить работу с ним.

3.4.7 Защита блока

Когда Вы программируете в исходном файле, Вы можете защитить блок специфическим атрибутом KNOW_HOW_PROTECT (защита ноу-хау).

Результат защиты блока

- Когда Вы открываете скомпилированный блок редактором STL, команды блока скрыты.
- Редактор STL показывает только переменные типов VAR_IN, VAR_OUT и VAR_IN_OUT. Переменные типа VAR и VAR_TEMP скрыты.

Правила использования защиты блока

- Существует ключевое слово KNOW_HOW_PROTECT. Введите его перед другими атрибутами блока.
- Этим способом могут быть защищены OB, FB, FC, и DB.

3.5 Основные правила для исходного файла

3.5.1 Общие правила для исходного файла SCL

Исходные файлы SCL должны подчиняться следующим правилам:

- В исходном файле SCL может быть отредактировано любое количество логических блоков (FB, FC, OB), блоков данных (DB), и пользовательских типов данных (UDT).
- Каждый тип блоков имеет типовую структуру.
- В конце каждой команды и каждого объявления переменных ставится точка с запятой (;).
- Между символами верхнего и нижнего регистров нет различий.
- Комментарии предназначаются только для программной документации. Они не влияют на исполнение программы.
- Экземплярные блоки данных создаются автоматически, когда вызван функциональный блок. Они не нуждаются в редактировании.
- DB0 имеет специальное назначение. Вы не можете создать блок данных с этим номером.

3.5.2 Порядок блоков

Когда Вы создаете исходный файл SCL, помните про следующий порядок блоков:

- Вызванный блок должен предшествовать вызываемому блоку.
- Определенный пользователем тип данных (UDT) должен предшествовать блоку, в котором он используется.
- Блок данных, в котором применяется определенный пользователем тип данных (UDT), находится после UDT.
- Глобальный блок данных предшествует всем блокам, имеющим к нему доступ.

3.5.3 Использование символьной адресации

В SCL программе Вы работаете с такими элементами, как входные/выходные сигналы, меркеры, счетчики, таймеры и блоки. Вы можете обратиться к этим элементам Вашей программы, используя абсолютные адреса (например, I1.1, M2.0, FB11), однако исходный файл SCL будет легче читаться, если использовать символику (например, Motor_ON). Абсолютный адрес будет доступен в Вашей программе и при использовании символов.

Локальные и глобальные символы

- Глобальные символы используются для областей памяти CPU и идентификаторов блоков. Они доступны во всей программе и поэтому должны быть уникальными. Таблицу символов Вы можете создать в STEP 7.
- Локальные символы известны только в блоке, в котором они объявлены. Вы должны назначить имена переменным, параметрам, константам и меткам переходов и можете использовать эти названия в других блоках для иных целей.

Замечание

Убедитесь, что имена символов уникальны и не совпадают с любым из ключевых слов.

3.6 Редактирование исходного файла SCL

3.6.1 Отмена последнего действия

Вы можете отменить одно или более действие командой **Edit > Undo (Правка > Отменить)**.

Отменяются не все действия. Например, команда меню **File > Save (Файл > Сохранить)** не отменяется.

3.6.2 Восстановление отмененного действия

После отмены одного или более действий, Вы можете использовать команду **Edit > Redo (Правка > Восстановить)**.

3.6.3 Нахождение и перемещение текстовых объектов

Если Вы редактируете исходный файл SCL, Вы можете сэкономить время поиском текстовых объектов и их заменой. Например, Вы можете искать ключевые слова, абсолютные идентификаторы, символические идентификаторы и т.д.

Выполните следующие шаги:

1. Выберите в меню команду Edit > Find and Replace (Правка >Найти и заменить)....
2. Введите настройки в диалоговом окне "Find and Replace (Найти и заменить)".
3. Начните поиск:
   - Нажмите кнопку "Find (Найти)" для поиска и выделения текста или
   - Нажмите кнопку "Replace (Заменить)" или "Replace All (Заменить все)" для нахождения текста и его замены на текст, введенный в текстовом окне "Replace with (заменить на)".

3.6.4 Выбор объектов текста

Вы можете выбрать текстовый объект и область кнопкой мыши.

Кроме того, Вы можете:

- Полностью пометить исходный текст командой меню **Edit > Select All (Правка > Выделить все)**
- Пометить слово двойным щелчком.
- Пометить целую строку, нажав на поле слева от нее.

Отменить выбор Вы можете командой **Edit > Undo Selection (Правка > Отменить выделение)**.

3.6.5 Копирование текстовых объектов

С помощью этой функции Вы можете копировать целую программу или ее часть. Копируемый текст размещается в буфере и затем может быть вставлен несколько раз в любое место текста.

Проделайте следующие шаги:

1. Выберите текстовый объект для копирования.
2. Копируйте объект:
   - Нажмите кнопку "Copy (Копировать)" на панели инструментов или
   - Выберите в меню команду **Edit > Copy (Правка > Копировать)**.
3. Переместите курсор в точку, в которую Вы хотите вставить объект (в этом или ином приложении).
4. Вставьте объект:
   - Нажмите кнопку "Paste (Вставить)" на панели инструментов или
   - Выберите в меню команду **Edit > Paste (Правка > Вставить)**.

3.6.6 Вырезание текста

С помощью этой функции Вы помещаете выделенный текст в буфер. Обычно эта команда используется вместе с командой меню **Edit > Paste (Правка > Вставить)**, которая вставляет вырезанный объект туда, куда указывает курсор.

Проделайте следующие шаги:

1. Выделите объект, который Вы хотите вырезать.
2. Вырежьте его:
   - Нажмите копку "Cut (Вырезать)" на панели инструментов или
   - Выберите в меню команду **Edit > Cut (Правка > Вырезать)**.

Замечание

- Выбранный объект не может быть вырезан командой меню **Edit > Cut (Правка > Вырезать)** , если она не активирована (залита серым цветом).
- Используя команду меню **Edit > Paste (Правка > Вставить)**, Вы можете вставить вырезанную часть в любую точку текста (в той же программе или в иной).
- Содержание буфера вырезанного текста будет оставаться таким до тех пор, пока Вы снова не примените команды **Edit > Cut (Правка > Вырезать)** или **Edit > Copy (Правка > Копировать)**.

3.6.7 Удаление текста

Вы можете удалить текст из исходного файла.

Проделайте следующие шаги:

1. Выберите текст, который Вы хотите удалить.
2. Примените команду меню **Edit > Delete (Правка > Удалить)**.

Удаленный текст не копируется в буфер вырезанного текста. Удаленный объект может быть восстановлен с помощью команд меню **Edit > Undo (Правка > Отменить)** или **Edit > Redo (Правка > Вернуть)**.

3.6.8 Размещение курсора в заданной строке

С помощью этой функции Вы можете поместить курсор в начале конкретной строки.

Проделайте следующие шаги:

1. Откройте диалоговую панель "Go To (Перейти)", выбрав команду меню **Edit > Go To Line (Правка > Перейти к строке)**.
2. Наберите на диалоговой панели в "Go To (Перейти)" номер строки.
3. Нажмите "OK".

3.6.9 Синтаксически правильный отступ строк

Имеются два варианта форматирования строк в исходном файле SCL:

- **Indent automatically (Автоматический отступ)**
  Когда действует эта функция, новая строка автоматически делает такой же отступ, как и предыдущая.
- **Indent keywords (Отступ по ключевым словам)**
  Когда действует эта функция, отступ делается в начале разделов деклараций и в управляющих структурах IF, CASE, FOR, WHILE и REPEAT.

Проделайте следующие шаги:

1. Выберите команду меню **Options > Customize (Параметры > Настройка)**.
2. Выберите в диалоговом окне закладку "Format (Формат)".
3. Удостоверьтесь, что действует опция "Use following Formats (Использовать следующее форматирование)".
4. Активируйте опцию "Indent automatically (Автоматический отступ)" или "Indent keywords (Отступ по ключевым словам)".

3.6.10 Установка шрифтов и цвета

Использование различных цветов и стилей для разных языковых элементов исходного файла SCL облегчает его прочтение и делает внешний вид файла более профессиональным. Форматируя исходный файл, Вы можете использовать следующие функции:

- **Keywords in uppercase (Ключевые слова в верхнем регистре)**:
  Когда действует эта функция, ключевые слова, такие как FOR, WHILE, FUNCTION_BLOCK, VAR или END_VAR пишутся заглавными буквами.
- **Defining the style and color (Определение стиля и цвета)**:
  Для различных языковых элементов, таких как операторы, комментарии или константы, установлены разные стили и цвета. Вы можете выбрать их из встроенных установок.
  Существуют следующие цвета:

Цвет шрифта	**Языковой элемент**	**Пример**
Синий	Ключевые слова	ORGANIZATION_BLOCK
	Тип встроенных данных	INT
	Встроенные идентификаторы	ENO
	Стандартные функции	BOOL_TO_WORD
Охра	Операторы	NOT
Розовый	Константы	TRUE
Сине-зеленый	Комментарии	//... or (*...*)
Фиолетовый	Глобальные символы (таблица символов) в кавычках	"Motor"
Черный	Обычный текст	Variables

Проделайте следующие шаги:

1. Выберите команду меню **Options > Customize (Параметры > Настройка)**.
2. Выберите в диалоговом окне закладку "Format (Формат)".
3. Убедитесь, что включена опция "Use following Formats for printing: (Использовать при печати следующие форматы:)"
4. Вы можете сделать необходимые установки. Подробную информацию о работе в диалоговом окне Вы можете получить, используя "Help (Помощь)".

3.6.11 Вставка шаблонов

3.6.11.1 Вставка вызова блоков

SCL поддерживает вызов программных блоков. Вы можете вызвать следующие блоки:

- Системные функциональные блоки (SFB) и системные функции (SFC) из библиотеки SIMATIC,
- Функциональные блоки и функции, созданные на SCL,
- Функциональные блоки и функции, созданные STEP 7.

Проделайте следующие шаги:

1. Выберите команду меню **Insert > Block Call (Вставить > Вызов блока)**.
2. Выберите необходимый блок SFC, SFB, FC или FB в диалоговом окне и подтвердите свой выбор "OK".
   SCL автоматически копирует вызванный блок в S7 программу и входные данные блока и формальные параметры с правильным синтаксисом в исходный файл.
3. Если Вы вызвали функциональный блок, добавляется информация о экземплярном DB.
4. Введите фактические параметры, необходимые для вызова блока. Помогая Вам выбрать фактические параметры, SCL показывает, как комментарий, требуемый тип данных.

3.6.11.2 Вставка шаблонов блоков

Редактор SCL позволяет вставлять шаблоны блоков OB, FB, FC, экземплярных DB, DB, DB, ссылок на UDT и UDT. Использование этих блочных шаблонов облегчает программирование и обеспечивает необходимый синтаксис.

Проделайте следующие шаги:

1. Установите курсор в ту точку, куда Вы хотите вставить шаблон блока.
2. Выберите команду меню **Insert > Block Template > OB/FB/FC/DB/IDB/DB Referencing UDT/UDT (Вставить> Шаблон блока > OB/FB/FC/DB/ IDB/ DB со ссылкой на UDT/UDT).**

3.6.11.3 Вставка шаблона комментариев

Редактор SCL позволяет вставить шаблон комментариев. Использование этого шаблона облегчает ввод информации и обеспечивает необходимый синтаксис.

Проделайте следующие шаги:

1. Установите курсор после заголовка необходимого блока.
2. Выберите в меню команду **Insert > Block Template > Comment (Вставить> Шаблон блока >Комментарий).**

3.6.11.4 Вставка шаблона параметров

Редактор SCL позволяет вставить шаблон объявления параметров. Использование этого шаблона облегчает набор Вашей программы и обеспечивает необходимый синтаксис. Вы можете объявить параметры в функциональных блоках и функциях.

Проделайте следующие шаги:

1. Установите курсор в раздел деклараций FB или FC.
2. Выберите команду меню **Insert > Block Template > Parameter (Вставить> Шаблон блока > Параметры).**

3.6.11.5 Вставка управляющих структур

Эта функция редактора SCL позволяет вставить в логические блоки шаблоны управляющих структур. Использование этих шаблонов облегчает ввод программы и обеспечивает необходимый синтаксис.

Проделайте следующие шаги:

1. Установите курсор в ту точку текста, куда Вы хотите вставить шаблон.
2. Выберите команду меню **Insert > Control Structure > IF/CASE/FOR/WHILE/REPEAT (Вставить> Управляющая структура > IF/CASE/FOR/WHILE/REPEAT)**.

3.7 Компиляция SCL программы

3.7.1 Что Вам необходимо знать о компиляции

Перед выполнением или тестированием Вашей программы, Вы должны выполнить ее компиляцию. Как только Вы начнете ее, компилятор запускается автоматически. Компилятор имеет следующие характеристики:

- Вы можете компилировать исходный файл SCL целиком или компилировать выбранные отдельные блоки.
- Все синтаксические ошибки, найденные компилятором, отобразятся на экране.
- Каждый раз при вызове функционального блока создается экземплярный блок данных, если он еще не существует.
- Вы можете компилировать совместно несколько исходных файлов SCL, создав файл управления компиляцией.
- Используйте команды меню **Options > Customize (Параметры > Настройки)** для установки настроек компиляции.

После создания пользовательской программы, которая откомпилирована и в ней исправлены ошибки, Вы можете предположить, что программа правильна. Тем не менее, при выполнении программы на ПЛК могут возникнуть проблемы. В этом случае используйте функции отладки SCL.

3.7.2 Настройка компилятора

Вы можете приспособить компилятор для Ваших потребностей.

Проделайте следующие шаги:

1. Выберите команды меню **Options > Customize (Параметры > Настройка)**, чтобы открыть диалоговое окно "Customize (Настройка)".
2. Выберите закладку "Compiler (Компилятор)" или "Create Block (Создать блок)".
3. Введите необходимые Вам настройки.

Настройки на закладке "Compiler (Компилятор) "

Create object code (Создание объектной программы)	С помощью этой опции Вы решите, хотите ли Вы или нет создать рабочую программу. Компиляция без этой опции работает просто как блок синтаксического контроля.
Optimize object code (Оптимизация объектной программы)	Когда выбрана эта опция, блоки оптимизируются по ячейкам памяти и по времени работы ПЛК. Желательно поддерживать эту опцию постоянно выбранной, поскольку оптимизация не имеет недостатков, влияющих на работу блока.
Monitor array limits (Контроль пределов массивов)	Если Вы выберите эту функцию, в течение всей работы программы S7 будет проверяться, не превышает ли индекс массива установленные рамки объявленного массива. Если индекс массива превысит их, флаг OK будет установлен в FALSE.
Create debug info (Создание отладочной информации)	Эта опция позволяет Вам протестировать программу отладчиком после ее компиляции и загрузки на CPU. Однако, при выборе этой опции, увеличиваются требования к памяти для программы и время работы CPU.
Set OK flag (Установка флага OK)	Эта опция позволяет использовать в исходном файле SCL флаг OK
Permit nested comments (Разрешение включен-ных комментариев)	Выберите эту опцию, если Вы хотите включить комментарии в Ваш исходный файл SCL.
Maximum string length (Максимальная длина строки)	Вы можете уменьшать стандартную длину строки данных типа STRING. Длина по умолчанию 254. Установка влияет на параметры ввода/ вывода, а также на возвращаемые величины функций. Заметьте, что величина должна быть не меньше, чем используемые в программе переменные типа STRING.

Настройки страницы "Create Block (Создание блока)"

Overwrite Blocks (Переписать блоки)	Переписывает существующие блоки в папке "Blocks (Блоки)" S7 программы, если блоки с теми же идентификаторами создаются в процессе компиляции. Блоки с одинаковыми именами, которые уже существуют в системе, также переписываются при загрузке блоков. Если эта опция не выбрана, то перед тем как переписать блок, потребуется подтверждение.
Display warnings (Показ предупреждений)	Вы можете решить, хотите ли Вы отображать предупреждения в дополнение к ошибкам, полученным при компиляции.
Display errors before warnings (Показ ошибок до предупреждений)	Вы можете показать ошибки перед предупреждениями.
Generate reference data (Создание справочных данных)	Когда выбрана эта опция, справочные данные создаются автоматически при создании блока. Справочные данные можно также создать или скорректировать командой меню **Options > Reference Data (Параметры > Справочные данные)**.
Include system attribute 'S7_server' (Включение системного атрибута 'S7_server')	Выберите эту опцию, если хотите, чтобы системный атрибут "S7_server" учитывался при создании блока. Вы назначаете этот атрибут, когда необходима конфигурация связей или сообщений. Он содержит связи или номера сообщений. Эта опция увеличивает время, необходимое для компиляции.

3.7.3 Компиляция программы

Перед тестированием или работой Ваша программа должна быть откомпилирована. Чтобы быть уверенным, что Вы компилируете последнюю сохраненную версию исходного файла, выберите команду меню **Options > Customize (Параметры > Настройка)** и установите опцию "Save before compiling (Сохранять перед компиляцией)" на закладке "Editor (Редактор)". Тогда команда меню **File > Compile (Файл > Компилировать)** сохранит исходный файл SCL.

Проделайте следующие шаги:

1. Сохраните исходный файл SCL перед компиляцией.
2. Для создания S7 программ, Вы должны выбрать опцию "Create Object code (Создание объектной программы)" на закладке "Compiler (Компилятор)" в окне "Customize (Настройка)".
3. Если требуется, установите иные модификации компилятора.
4. Проверьте таблицу символики в соответствующей программной папке.
5. Вы можете начать компиляцию так:
   - Команда меню **File > Compile (Файл > Компилировать)** компилирует исходный файл целиком.
   - Команда меню **File > Compile Selected Blocks (Файл > Компилировать выбранные блоки)** открывает диалоговую панель, в которой Вы выбираете нужные для компиляции блоки.
6. Диалоговое окно "Errors and Warnings (Ошибки и предупреждения)" показывает все ошибки и предупреждения, найденные в процессе компиляции. Исправьте все ошибки, выданные компилятором, а затем повторите процедуру, описанную выше.

3.7.4 Создание файла управления компиляцией

Если Вы создаете файл управления компиляцией, то Вы можете одновременно компилировать несколько исходных файлов, находящихся в папке. В файл управления Вы записываете имена всех исходных файлов, которые хотите откомпилировать.

Проделайте следующие шаги:

1. Выбором команды меню File > New (Файл > Новый) откройте диалоговое окно "New (Новый)".
2. Выберите в окне "New (Новый)"
   - Папку исходных файлов программы S7
   - Фильтр "SCL Compilation Control File (Файл управления компиляцией)"
3. Введите имя файла управления в соответствующей графе (максимум 24 символа) и нажмите "OK".

4. Файл будет создан и отображен в рабочем окне. □В рабочем окне введите в нужном порядке имена исходных SCL файлов, которые должны быть откомпилированы и запишите файл.
5. Начните компиляцию, выбрав команду меню **File > Compile (Файл > Компилировать)**.

3.7.5 Отладка программы после компиляции

Все синтаксические ошибки и предупреждения, обнаруженные в процессе компиляции, отображаются в окне " Errors and Warnings (Ошибки и предупреждения)". Если есть ошибка, блок не может компилироваться, пока она не будет исправлена. Тем не менее, и выполняя откомпилированный блок в ПЛК, Вы можете столкнуться с проблемами.

Исправление ошибок:

1. Выберите ошибку и нажмите клавишу F1, посмотрите в описании, как исправить ошибку.
2. Если отображены номер строки и номер колонки, Вы можете найти ошибку в исходном файле так:
   - Нажмите сообщение об ошибке в окне "Errors and Warnings (Ошибки и предупреждения)" правой кнопкой мыши и выберите команду **Display Errors (Показать ошибки)**.
   - Два раза щелкните по сообщению об ошибке, чтобы установить курсор в нужную точку (строка, колонка).
3. Найдите правильный синтаксис в описании языка SCL.
4. Проделайте в тексте необходимые исправления.
5. Сохраните исходный файл.
6. Откомпилируйте программу снова.

3.8 Сохранение и печать исходного файла SCL

3.8.1 Сохранение исходного файла SCL

Термин "saving" в SCL всегда указывает на сохранение исходных файлов. Программные блоки создаются в SCL при компиляции исходных файлов и автоматически записываются в соответствующую папку программ. Вы можете сохранить исходный файл в любое время в его текущем состоянии. При этом он не будет компилироваться. Все синтаксические ошибки также сохранятся.

Проделайте следующие шаги:

- Выберите команду меню **File > Save (Файл > Сохранить)** или нажмите кнопку "Save (Сохранить)" на панели инструментов.
  Исходный файл SCL изменится.
- Если Вы хотите создать копию рабочего исходного файла SCL, выберите команду меню **File > Save As (Файл > Сохранить как…)**. В диалоговом окне Save As (Сохранить как…) укажите новое имя и путь, куда скопировать файл.

3.8.2 Настройки форматирования страницы

Вы можете изменить вид распечатки следующим образом:

- Команда меню **File > Page Setup (Файл > Параметры страницы)** позволяет Вам выбрать формат страницы для печати.
- Вы можете установить верхние и нижние колонтитулы для вашего документа в SIMATIC Manager, используя команды меню **File > Headers and Footers (Файл > Колонтитулы)**.
- Вы также можете отобразить на экране и проверить страницу перед тем, как ее распечатать, используя команду меню **File > Print Preview (Файл > Просмотр перед печатью)**.

3.8.3 Печать исходного файла SCL

Печатается исходный файл SCL, который находится в рабочем окне редактора, иными словами, чтобы распечатать файл, его надо открыть.

Проделайте следующие шаги:

1. Активизируйте в окне редактора необходимый исходный файл.
2. Откройте окно диалога "Print (Печатать)":
   - Нажмите кнопку "Print (Печатать)" на панели инструментов или
   - Выберите команду меню **File > Print (Файл > Печатать)**.
3. В окне диалога выберите такие опции "Print (Печатать)", как число копий и диапазон печатаемых страниц.
4. Нажмите "OK".

3.8.4 Установка опций печати

Вы можете использовать следующие функции для форматирования Вашей распечатки:

- **Form feed at start of block (Загрузка страницы в начале блока)**
  Когда эта функция выбрана, каждый блок печатается с новой страницы.
- **Print line numbers (Печать номеров строк)**
  При выборе этой опции, в начале каждой строки печатается ее номер.
- **Define the font (Определение шрифта)**
  Шрифт по умолчанию - это Courier размер 8. Вы можете изменять эту установку.
- **Define style (Определение стиля)**
  Вы можете использовать различные стили для разных языковых элементов. Вы можете выбрать следующие стили:

Языковой элемент	**Пример**
Обычный текст	
Ключевое слово	ORGANIZATION_BLOCK
Идентификаторы данных встроенного типа	INT
Встроенные идентификаторы	ENO
Идентификаторы стандартных функций	BOOL_TO_WORD
Операторы	NOT
Константы	TRUE
Раздел комментариев	(* *)
Строчные комментарии	//...
Глобальные символы (таблица символов) в кавычках	"Motor"

Проделайте следующие шаги:

1. Выберите команду меню **Options > Customize (Параметры > Настройка)**.
2. Выберите закладку "Print (Печатать)" в диалоговом окне.
3. Убедитесь, что выбрана опция "Use following Formats (Использовать следующее форматирование)".
4. Сделайте необходимые установки. Вы можете получить детальную информацию, нажав кнопку "Help (Помощь)" в диалоговом окне.

3.9 Загрузка созданной программы

3.9.1 Очистка памяти CPU

Используя функцию Clear/Reset (Очистить память), в режиме online Вы можете удалить пользовательскую программу в памяти CPU.

Проделайте следующие шаги:

1. Выберите команду меню **PLC > Operating Mode (ПЛК > Режим работы)** и переключите CPU на STOP.
2. Выберите команду меню **PLC > Clear/Reset (ПЛК > Сброс памяти)**.
3. Подтвердите это действие в диалоговом окне.

Внимание

- CPU перезагружен.
- Все данные удалены.
- CPU завершил все связи.
- Если вставлена карта памяти, CPU копирует содержание карты памяти во внутреннюю загрузочную память после сброса памяти.

3.9.2 Загрузка пользовательской программы в CPU

Требования

Когда Вы компилируете исходный файл SCL, создаваемые блоки сохраняются в папке "Blocks (Блоки)" программы S7.

Блоки, вызываемые в первую очередь, автоматически копируются в папку "Blocks (Блоки)" и вводятся в список загрузки.

Далее с помощью SIMATIC Manager Вы можете загрузить программные блоки из программирующего устройства в CPU.

Перед загрузкой блоков, должна быть установлена связь между программирующим устройством и CPU. В SIMATIC Manager в online пользовательская программа должна назначаться CPU.

Процедура

Как только исходный файл откомпилирован, Вы должны начать загрузку таким образом.

- Команда меню **File > Download (Файл > Загрузить)** загружает все блоки исходного файла, и все вызываемые блоки загружаются в первую очередь.
- Команда меню **File > Compile Selected Blocks (Файл > Компилировать выбранные блоки)** открывает диалоговое окно, в котором Вы можете выбрать отдельные блоки для компиляции.

Блоки передаются на CPU. Если блок уже существует в RAM CPU, у Вас спросят, хотите ли Вы или нет переписать блок.

Замечание

Желательно загружать пользовательскую программу в режиме STOP, поскольку могут произойти ошибки, если Вы перезаписываете программу в модуль в режиме RUN.

3.10 Отладка созданной программы

3.10.1 Функции отладки SCL

Используя функции отладки SCL, Вы можете проверить выполнение программы на CPU и обнаружить любые ошибки в программе. Синтаксические ошибки выявляются при компиляции. Программные ошибки проявляются при работе программы и обнаруживаются, в некоторых случаях, через системные прерывания. Вы можете обнаружить ошибки программирования, используя функции отладки.

Функции отладки SCL

В SCL Вы можете воспользоваться следующими функциями теста:

- Monitor (Наблюдение) (S7-300/400-CPU)
  С помощью этой функции Вы можете отобразить имена и текущие величины переменных исходного файла SCL. В течение тестирования, значения переменных и параметров будут отображены в хронологическом порядке и циклически обновляться.
- Debug with Breakpoints/Single step (пошаговая отладка с контрольными точками) (только S7-400 CPU)
  С помощью этой функции Вы можете установить контрольные точки программы и отладить ее по шагам. Вы можете прогнать программный алгоритм шаг за шагом и посмотреть, как изменяются значения переменных.

Внимание

Если в программе есть ошибки, тестирование при включенной установке может привести к серьезным травмам персонала или к повреждению оборудования!

Перед запуском функций отладки убедитесь, что опасной ситуации не существует.

Требования к отладке

- Программа должна быть откомпилирована с опциями "Create Object code (Создание объектного кода)" и "Create debug info (Создание отладочной информации)". Вы можете выбрать эти параметры на странице "Compiler (Компилятор)" диалоговой панели "Customize (Настройка)" . Используйте команду меню **Options > Customize (Параметры > Настройка)**.
- Вы должны установить связь online от программатора или ПК к CPU.
- Программа должна быть загружена в CPU. Вы можете проделать это, используя команду меню **PLC > Download (ПЛК > Загрузить)**.

3.10.2 Функция отладки "Monitor (Наблюдение)"

Используя непрерывное наблюдение, Вы можете отладить группу операторов. Эта группа команд называется проверочный диапазон. В течение тестирования, значения переменных и параметров в этом диапазоне отображаются в хронологическом порядке и циклически обновляются. Если проверочный диапазон находится в программной секции, которая выполняется в каждом цикле, то значения наблюдаемых переменных обычно не могут получаться для всех циклов подряд.

Величины, которые обновляются в текущем цикле и величины, которые не обновляются, могут различаться цветом.

Диапазон команд, который может быть протестирован, зависит от производительности подключенного CPU. При компиляции команды SCL создается различное число команд в машинном коде, поэтому длина диапазона проверки переменная, она определяется и индицируется отладчиком SCL, когда Вы выбираете первую команду из необходимого диапазона проверки.

Запомните следующие ограничения для функции "Monitor (Наблюдение)":

- Переменные сложных типов данных не могут быть показаны. Элементы этих переменных могут наблюдаться, если они принадлежат к элементарному типу данных.
- Не отображаются переменные типа DATE_AND_TIME, STRING и параметрические типы BLOCK_FB, BLOCK_FC, BLOCK_DB, BLOCK_SDB, TIMER, COUNTER.
- Не отображается доступ к блокам данных в формате <символ>.<абсолютный адрес> (например, Data.DW4).

Обращение к этой информации обычно увеличивает время цикла. Чтобы повлиять на время цикла, SCL обеспечивает два различных режима функционирования.

Способ	**Объяснение**
Test Operation (режим тестирования)	В режиме "Test Operation (Режим тестирования)", проверочный диапазон ограничен только работой связи с CPU. Все функции отладки могут быть использованы без ограничений. Время цикла CPU может быть значительно увеличено в зависимости от команд, например, программными циклами.
Process Operation (режим функ-ционирования)	В "Process Operation (Рабочий режим)", SCL отладчик ограничивает максимальный проверочный диапазон так, что время цикла в течение тестирования не превышает реальное время работы программы или превышает незначительно.

3.10.3 Отладка с контрольными точками в пошаговом режиме

Если Вы тестируете с помощью контрольных точек, программа проверяется шаг за шагом. Вы можете выполнять алгоритм команда за командой и увидеть, как изменяются значения переменных.

После установки контрольных точек, Вы можете запустить программу и проверить ее шаг за шагом, начиная с контрольных точек.

Простые шаги функций:

Когда действует функция "Отладка с помощью контрольных точек", Вы можете использовать следующие команды:

- Next Statement (Следующая команда)
  Выполняется текущая выбранная команда.
- Resume (Возобновление)
  Возобновление выполнения программы до следующей активной точки контроля.
- To cursor (До позиции курсора)
  Продолжить выполнение программы до позиции курсора в исходном файле.
- Execute Call (Выполнение вызова)
  Переход в вызываемый SCL блок вниз по иерархии вызовов.

Контрольные точки

Вы можете установить контрольные точки в любой секции исходного файла.

Точки контроля передаются программируемому контроллеру и активируются, когда Вы выберете команду меню **Debug > Breakpoints Active (Отладка > Активировать точки останова)**. Программа будет выполняться с первой точки.

CPU допускают следующее максимальное число контрольных точек.

CPU 416	максимум 4 активные точки
CPU 414	максимум 4 активные точки
S7-300 CPU	не возможны точки контроля

Требования:

Открытый исходный файл прежде не может быть изменен в редакторе.

3.10.4 Шаги контроля

Как только Вы загрузите откомпилированную программу в программируемый контроллер, Вы можете протестировать ее в режиме "Monitor (Наблюдение)".

Проделайте следующие шаги:

1. Убедитесь, что выполнены требования к отладчику, а CPU находится в режимах RUN или RUN-P.
2. Выберите окно, содержащее требуемый исходный файл.
3. Если Вы хотите изменить установленный по умолчанию режим контроля (process operation), выберите команду меню **Debug > Operation > Test Operation (Отладка > Функционирование > Режим тестирования)**.
4. Установите курсор на ту строку исходного файла, где находится первая команда, которую нужно протестировать.
5. Убедитесь, что результат работы программы не может быть опасным и затем выберите команду меню **Debug > Monitor (Отладка > Монитор)**. **Результат:** Наибольший проверочный диапазон будет вычислен и выделен серым цветом в левом крае окна. Окно разделено, а имена и текущие значения величин из проверочного диапазона показаны строка за строкой в правой половине окна.
6. Выберите команду меню **View > Symbolic Representation (Вид > Символическое представление),** чтобы включить символические имена из таблицы символов в Вашу программу.
7. Выберите команду меню **Options > Customize (Параметры > Настройка)**, откройте закладку "Format (Формат)" и установите цвет для каждой показанной величины.
8. Выберите команду меню **Debug > Monitor (Отладка > Монитор)** для отладки.
9. Выберите команду меню **Debug > Finish Debugging (Отладка > Завершить отладку)** для завершения отладки.

Замечание

Количество команд, которые могут быть отлажены, (проверочный диапазон) зависит от производительности подключенного CPU.

3.10.5 Пошаговая отладка с помощью контрольных точек

3.10.5.1 Определение контрольных точек

Установите и определите контрольные точки:

1. Откройте исходный файл, который хотите отладить.
2. Откройте, если необходимо, панель инструментов контрольных точек, используя команду меню **View > Breakpoint Bar (Вид > Панель контрольных точек)**.
3. Установите курсор в нужную точку и выберите команду меню **Test > Set Breakpoint (Тест > Установить контрольную точку)** или нажмите соответствующую кнопку на панели контрольных точек. Контрольные точки будут показаны на левом краю окна как красный круг.
4. Если нужно, вызовите **Debug > Edit Breakpoints (Отладка > Редактирование контрольных точек)** и определите режим работы вызова. При этом определяется, будут ли активны или нет точки контроля, находящиеся в указанном блоке, в зависимости от следующих условий вызова блока:
   - При вызове данного блока из определенного блока более высокого уровня и/или
   - При вызове с использованием конкретного блока данных.

3.10.5.2 Начало тестирования с помощью точек контроля

Как только Вы загрузите откомпилированную программу в программируемый контроллер и установите точки контроля, Вы можете начать отладку в режиме " Test with Breakpoints (Тестирование с контрольными точками)".

Проделайте следующие шаги:

1. Откройте исходный файл SCL с программой, которую хотите отладить.
2. Убедитесь, что результаты работы не будут опасными и CPU находится в режиме RUN-P. Выберите команду меню **Debug > Breakpoints Active (Отладка > Активировать контрольные точки)** и затем **Debug > Monitor (Отладка > Монитор)**.
   Результат: Окно вертикально разделено на две половины. Программа выполнена до следующей контрольной точки. Когда она достигнута, CPU переходит в режим HOLD и красные точки контроля маркируются желтым указателем.
3. Продолжите работу одной из следующих команд:
   - Выберите команду меню **Debug > Resume (Отладка > Продолжить)** или нажмите кнопку "Resume (Продолжить)".
     CPU переходит в режим RUN. Когда будет достигнута следующая активная контрольная точка, снова установится режим HOLD и точки контроля отобразятся в окне справа.
   - Выберите команду меню **Debug > Next Statement (Отладка > Следущий оператор)** или нажмите кнопку "Next Statement (Следующий оператор)".

CPU переходит в режим RUN. После обработки следующей команды установится HOLD и справа в окне отобразится содержание обработанных к текущему времени переменных.

- Выберите команду меню **Debug > To cursor (Отладка > До курсора)** или нажмите кнопку "To cursor (До курсора)".
  CPU переходит в режим RUN. Когда выбранная точка будет достигнута, снова установится режим HOLD.
- Выберите команду меню **Debug > Execute call (Отладка > Выполнить вызов)**, если программа остановилась в строке, содержащей вызов блока.
  Если нижний в иерархии вызовов блок создан в SCL, он будет открыт и выполнен в режиме тестирования. После этого программа вернется обратно в точку вызова.
  Если блок будет создан на другом программном языке, вызов будет пропущен и будет выбрана следующая строка программы.

Замечание

Команды меню **Debug > Next Statement (Отладка > Следущий оператор)** или **Debug > To cursor (Отладка > До курсора)** сами устанавливают и активируют контрольные точки. Когда Вы выбираете эту функцию, убедитесь, что Вы не используете максимальное количество контрольных точек для данного CPU.

3.10.5.3 Остановка тестирования с помощью точек контроля

Чтобы вернуться к нормальному выполнению программы:

- Деактивируйте команду меню **Debug > Breakpoints Active (Отладка > Активировать контрольные точки),** чтобы прервать отладку или
- Выберите команду меню **Debug > Finish Debugging (Отладка > Завершить отладку),** чтобы завершить отладку.

3.10.5.4 Активирование, деактивирование и удаление контрольных точек

Вы можете активировать/деактивировать и удалять установки точек контроля индивидуально:

1. Выберите команду меню Debug > Edit Breakpoints (Отладка > Правка контрольных точек).
2. Вы можете
   - Активировать и деактивировать выбранные точки контроля с помощью маркера.
   - Удалять точки контроля.

Для удаления всех точек контроля выберите команду меню **Debug > Delete All Breakpoints (Отладка > Удалить все контрольные точки).**

3.10.5.5 Отладка в пошаговом режиме

Проделайте следующие шаги:

1. Установите контрольные точки перед той строкой, начиная с которой Вы хотите отлаживать программу по шагам.
2. Выберите команду меню **Debug > Breakpoints Active (Отладка > Активировать контрольные точки)**.
3. Выполняйте программу, пока не достигнете точки контроля.
4. Чтобы выполнить следующую команду, выберите команду меню **Debug > Next Statement (Отладка > Следующий оператор)** или нажмите кнопку на панели инструментов.
   - Если оператор - это вызов блока, то он будет выполнен и программа вернется к первому после вызова блока оператору.
   - С помощью команды меню **Debug > Execute Call (Отладка > Выполнить вызов)**, Вы можете перейти в вызываемый блок. Затем Вы можете продолжить отладку в пошаговом режиме или можете установить контрольную точку. В конце блока Вы вернетесь к оператору, который находится после вызова блока.

3.10.6 Использование функций отладки STEP 7

3.10.6.1 Создание и отображение справочных данных

Вы можете создать и использовать справочные данные для помощи в отладке и изменении программы.

Вы можете отобразить следующие справочные данные:

- Структуру пользовательской программы
- Список перекрестных ссылок
- Список назначений
- Список неиспользованных адресов
- Список адресов без символов

Создание справочных данных

Вы можете создать справочные данные следующим образом:

- Используя команду меню **Options > Reference Data > Display (Параметры > Справочные данные > Отобразить)**, Вы можете создать или исправить и показать необходимые данные.

Фильтрацией Вы можете ограничить число отображаемых справочных данных и значительно ускорить создание данных. Выберите команду меню **Options > Reference Data > Filter (Параметры > Справочные данные > Фильтровать)**.

- Используя команду меню **Options > Customize (Параметры > Настройка)**, Вы выбираете, создавать автоматически или нет справочные данные при компиляции исходного файла. Если Вы хотите, чтобы

справочные данные создавались автоматически, пометьте опцию “Create Reference Data (Создавать справочные данные)" на закладке “Create Block (Создать блок)". Помните, что автоматическое создание справочных данных будет увеличивать время, необходимое для компиляции программы.

3.10.6.2 Наблюдение и изменение переменных

Когда Вы тестируете свою программу с помощью функции «Monitoring and modifying variables (наблюдение и изменение переменных)», Вы можете сделать следующее:

- Отобразить текущие значения глобальных данных, содержащихся в программе (наблюдение)
- Присвоить переменным, использующимся в Вашей программе, установленные значения (изменение)

Проделайте следующие шаги:

- Выберите команду меню **PLC > Monitor/Modify Variables (ПЛК > Наблюдение/изменение переменных)**.
- Создайте таблицу переменных (VAT) в окне. Если Вы хотите изменить переменные, введите новые значения.
- Определите триггерные точки (точки включения) и условия.

Внимание

Выполнение тестирования в то время, когда действует Ваше оборудование, может повести за собой серьезные травмы персонала или повреждения оборудования, если программа содержит ошибки. Перед выполнением отладки убедитесь, что опасности не существует!

3.11 Отображение и изменение свойств CPU

3.11.1 Отображение и изменение режима работы CPU

Вы можете определить и изменить текущий режим работы CPU. Должна быть установлена связь с CPU (режим online).

Проделайте следующие шаги:

1. Выберите команду меню **PLC > Operating Mode (ПЛК > Режим работы)**.
2. В окне диалога выберите один из следующих режимов:
   - Warm restart (теплый рестарт)
   - Cold restart (холодный рестарт)
   - Hot restart (горячий рестарт)
   - STOP (стоп)

Внимание

Изменение режима работы при действующем оборудовании может привести к серьезным травмам персонала или повреждениям оборудования, если программа содержит ошибки!

Перед выполнением отладки убедитесь, что опасности не существует!

3.11.2 Отображение и установка даты и времени на CPU

Вы можете запросить у CPU время и изменить его. Должна быть установлена связь с CPU.

Проделайте следующие шаги:

1. Выберите команду меню **PLC > Set Date and Time (ПЛК > Установить дату и время)**.
2. В появившемся окне диалога установите дату и время для часов в CPU.

Если CPU не оснащено часами, в окне диалога будет показано "00:00:00" и в графе дата будет "00.00.00". Это значит, что Вы не можете внести изменения.

3.11.3 Считывание данных о CPU

Вы можете посмотреть следующую информацию о CPU:

- Системное семейство, тип, заказной номер, версия CPU.
- Размер оперативной и загрузочной памяти, максимально возможная конфигурация загрузочной памяти.
- Размер и адресация областей отображения входов и выходов, таймеров, счетчиков и меркеров.
- Размер локальных данных при работе CPU.
- Способно или нет CPU на многопроцессорную обработку.

При этом должна быть установлена связь с CPU.

Проделайте следующие шаги:

1. Выберите команду меню PLC > Module Information (ПЛК > Информация о модуле).
2. Выберите в диалоговом окне закладку "General (Общие свойства)".

3.11.4 Чтение диагностического буфера CPU

Если Вы читаете информацию из буфера диагностики, Вы можете найти причину перехода в режим STOP или проследить причину появления диагностического сообщения.

У Вас должна быть установлена связь с CPU.

Проделайте следующие шаги:

1. Выберите команду меню PLC > Module Information (ПЛК > Информация о модуле).
2. Выберите в диалоговом окне закладку "Diagnostic Buffer (Диагностический буфер)".

3.11.5 Отображение/Сжатие пользовательской памяти CPU

Используя эту функцию, Вы можете отобразить информацию о загрузочной памяти CPU и, если необходимо, реорганизовать память CPU. Это необходимо, когда наибольшее свободное пространство памяти не достаточно, чтобы загрузить новый объект на CPU с PG.

Должна быть установлена связь с CPU.

Проделайте следующие шаги:

1. Выберите команду меню PLC > Module Information (ПЛК > Информация о модуле).
2. Выберите в диалоговом окне закладку "Memory (Память)".

3.11.6 Отображение времени цикла на CPU

Между выбранными предельными величинами времени цикла представлены следующие интервалы времени:

- Длительность самого длинного цикла с момента последнего изменения STOP на RUN.
- Длительность самого короткого цикла с момента последнего изменения STOP на RUN.
- Длительность последнего цикла.

Если длительность последнего цикла приближается к контрольному времени, возможно, что контрольное время будет превышено, и CPU перейдет в режим STOP. Время цикла может увеличиться, например, за счет тестирования блоков. Для вывода времени цикла Вашей программы, Вы должны установить связь с CPU.

Проделайте следующие шаги:

1. Выберите команду меню PLC > Module Information (ПЛК > Информация о модуле).
2. Выберите в диалоговом окне закладку "Cycle Time (Время цикла) ".

3.11.7 Отображение системного времени CPU

Системное время CPU включает информацию о внутренних часах и времени синхронизации между несколькими CPU.

Вы должны установить связь с CPU.

Проделайте следующие шаги:

1. Выберите команду меню PLC > Module Information (ПЛК > Информация о модуле).
2. Выберите в диалоговом окне закладку "Time System (Система времени)".

3.11.8 Отображение блоков на CPU

Вы можете отобразить блоки, доступные в подключенном CPU.

Вы должны установить связь с CPU.

Проделайте следующие шаги:

1. Выберите команду меню PLC > Module Information (ПЛК > Информация о модуле).
2. В следующем диалоговом окне выберите закладку "Performance Data/Blocks (Данные о производительности/Блоки)".

3.11.9 Отображение информации о связи с CPU

Для каждого CPU Вы можете получить информацию о выбранной и максимальной скорости передачи данных, использовании ресурсов связи.

Вы должны установить связь с CPU.

Проделайте следующие шаги:

1. Выберите команду меню PLC > Module Information (ПЛК > Информация о модуле).
2. Выберите в диалоговом окне закладку "Communication (Связь)".

3.11.10 Отображение стековой памяти CPU

Выбрав эту закладку, Вы можете посмотреть информацию о содержании стеков CPU. CPU должен быть в режиме STOP или должен достигнуть контрольной точки при пошаговой отладке.

Отображение стеков чрезвычайно полезно для обнаружения ошибок, например, когда Вы тестируете блоки. Если перевести CPU в STOP, Вы можете проанализировать точку останова с помощью текущего значения слова состояния и содержания аккумуляторов в точке прерывания (I стек), чтобы найти причину программной ошибки.

Вы должны установить связь с CPU.

Проделайте следующие шаги :

1. Выберите команду меню PLC > Module Information (ПЛК > Информация о модуле).
2. Выберите в диалоговом окне закладку "Stacks (Стеки)".

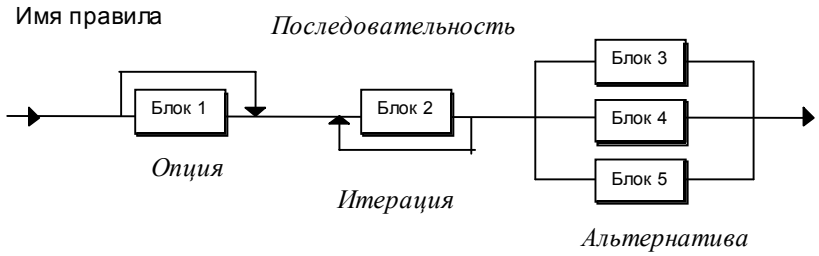
4 Основные понятия SCL

4.1 Интерпретация синтаксических диаграмм

Основное средство для описания языка в различных разделах - синтаксическая диаграмма. Она обеспечивает ясное понимание структуры синтаксиса SCL. Раздел "Описание языка" содержит все диаграммы с их языковыми элементами.

Что такое синтаксическая диаграмма?

Синтаксическая диаграмма - это графическое изображение структуры языка. Структура определяется рядом правил. Одно правило может базироваться на другом на более глубоком уровне.

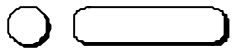


Синтаксическая диаграмма читается справа налево. Должны использоваться следующие структурные правила:

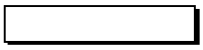
- Последовательность: порядок следования блоков
- Опция: пропущенная ветка
- Итерация: повторение ветки
- Альтернатива: выбор одной из нескольких ветвей

Что такое типы блоков?

Блок – это базовый элемент или элемент, создающий другие объекты. Диаграмма ниже показывает символы, которые представляют различные типы блоков.



Базовый элемент, который не требует никаких дальнейших объяснений. Это печатные или специальные символы, ключевые слова и предопределенные идентификаторы. Детали этих блоков копируются без изменений.



Сложный элемент, описанный другой синтаксической диаграммой.

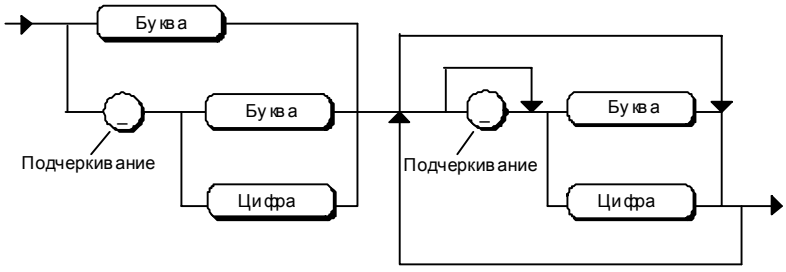
Что значит гибкий формат?

Когда пишется исходный код, программист должен соблюдать не только **синтаксические**, но и **лексические правила**.

Лексические и синтаксические правила детально описаны в разделе "Описание языка". Гибкий формат означает, что Вы можете включить между секциями правила такие символы форматирования, как пробелы, табуляцию и разрыв страницы, а также комментарии.

В лексическом правиле описан не гибкий формат! Когда Вы применяете лексическое правило, Вы должны использовать спецификацию точно по определению.

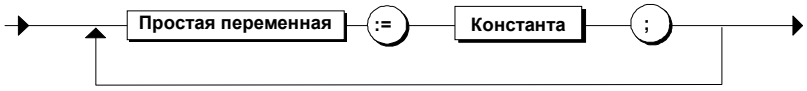
Лексическое правило



Следующие примеры соответствуют приведенному правилу:
R_CONTROLLER3
_A_FIELD
_100_3_3_10
Следующие примеры не соответствуют этому правилу:
1_1AB
RR__20
*#AB

Синтаксическое правило

В этом правиле используется гибкий формат.



Следующий пример демонстрирует приведенное правило:
VARIABLE_1 := 100; SWITCH:=FALSE;
VARIABLE_2 := 3.2;

4.2 Набор символов

Буквенные и числовые символы

SCL пользуется следующими символами как подмножествами символов ASCII:

- Буквами от A до Z (прописными и строчными).
- Арабскими цифрами от 0 до 9.
- Пробелами – собственно пробелами (код ASCII = 32) и всеми управляющими символами (коды ASCII 0-31), включая символ конца строки (код ASCII = 13).

Другие символы

Следующие символы имеют в SCL специфическое значение:

+	-	*	/	=	<	>	[	]	(	)
:	;	$	#	"	'	{	}	%	.	,

Замечание

В разделе "Описание языка", Вы можете найти детальный список всех символов с определенным назначением и информацию о том, как они интерпретируются в SCL.

4.3 Зарезервированные слова

Зарезервированные (ключевые) слова используются только для строго определенных целей. Между прописными и строчными буквами нет никаких различий.

Ключевые слова SCL

AND	END_CASE	ORGANIZATION_BLOCK
ANY	END_CONST	POINTER
ARRAY	END_DATA_BLOCK	PROGRAM
AT	END_FOR	REAL
BEGIN	END_FUNCTION	REPEAT
BLOCK_DB	END_FUNCTION_BLOCK	RETURN
BLOCK_FB	END_IF	S5TIME
BLOCK_FC	END_LABEL	STRING
BLOCK_SDB	END_TYPE	STRUCT
BLOCK_SFB	END_ORGANIZATION_BLOCK	THEN
BLOCK_SFC	END_REPEAT	TIME
BOOL	END_STRUCT	TIMER
BY	END_VAR	TIME_OF_DAY
BYTE	END_WHILE	TO
CASE	ENO	TOD
CHAR	EXIT	TRUE
CONST	FALSE	TYPE
CONTINUE	FOR	VAR
COUNTER	FUNCTION	VAR_TEMP
DATA_BLOCK	FUNCTION_BLOCK	UNTIL
DATE	GOTO	VAR_INPUT
DATE_AND_TIME	IF	VAR_IN_OUT
DINT	INT	VAR_OUTPUT
DIV	LABEL	VOID
DO	MOD	WHILE
DT	NIL	WORD
DWORD	NOT	XOR
ELSE	OF	Названия стандартных функций
ELSIF	OK	
EN	OR	

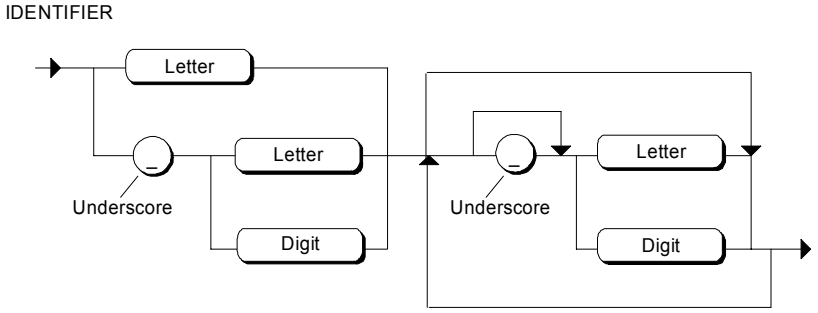
4.4 Идентификаторы

Описание

Идентификаторы – это наименования, которыми обозначаются объекты языка SCL, другими словами - константы, переменные или блоки.

Правила

Идентификаторы могут состоять максимум из 24 букв или чисел в любом порядке, но первым символом должна быть любая буква или символ подчеркивания. Разрешены символы верхнего и нижнего регистров, однако в идентификаторах они не различаются (например, AnNa и AnnA - синонимы).



Примеры

Следующие примеры показывают правильные идентификаторы:.

X	y12
Sum	Temperature
Name	Surface
Controller	Table

В этом примере приведены неправильные идентификаторы.

4th	//Первый символ должен быть буквой или символом //подчеркивания
Array	//ARRAY – ключевое слово
S Value	//Не допустим пробел (помните, что //пробел – также символ).

Замечание

- Убедитесь, что имена уже не использованы ключевыми словами или стандартными идентификаторами.
- Выбирайте уникальные имена для того, чтобы текст программы был понятнее.

4.5 Стандартные идентификаторы

В SCL есть множество встроенных стандартных идентификаторов:

- Идентификаторы блоков,
- Идентификаторы адреса для адресации областей памяти CPU,
- Идентификаторы таймеров и
- Идентификаторы счетчиков.

4.6 Идентификатор блока

Описание

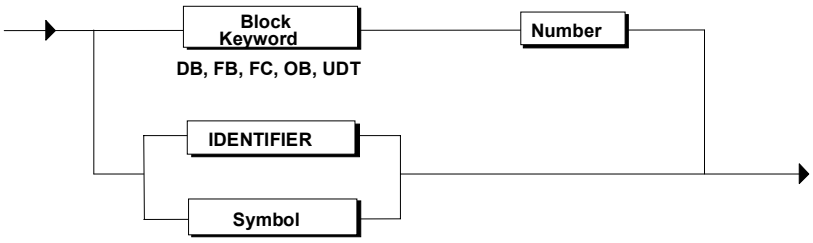
Эти стандартные идентификаторы используются для абсолютной адресации блоков.

Правила

Эта таблица отсортирована в порядке немецкой мнемоники, соответствующей международной мнемонике, показанной во второй графе. Буква x является меткой-заменителем для чисел от 0 до 65533 или от 0 до 65535 для таймеров и счетчиков.

Мнемоника (SIMATIC)	Мнемоника (IEC)	Идентификаторы
DBx	DBx	Блок данных. Идентификатор блока DB0 зарезервирован для SCL.
FBx	FBx	Функциональный блок
FCx	FCx	Функция
Obx	Obx	Организационный блок
SDBx	SDBx	Блок системных данных
SFCx	SFCx	Системная функция
SFBx	SFBx	Блок системной функции
Tx	Tx	Таймер
UDTx	UDTx	Тип данных, определенных пользователем
Zx	Cx	Счетчик

В SCL есть несколько способов, с помощью которых Вы можете определить идентификатор блока. Вы можете определить целое десятичное число как номер блока.



Пример

Это правильные идентификаторы:
FB10
DB100
T141

4.7 Идентификатор адреса

Описание

В любой точке Вашей программы Вы можете адресовать области памяти CPU, используя идентификаторы адреса.

Правила

Эта таблица отсортирована в порядке немецкой мнемоники, соответствующей международной мнемонике, показанной во второй графе. Идентификаторы адреса для блоков данных правильны только тогда, когда блок данных уже определен.

Мнемоника (SIMATIC)	Мнемоника (IEC)	Адреса	Тип данных
Ax.y	Qx,y	Выход (через образ процесса)	Bit
Abx	QBx	Выход (через образ процесса)	Byte
Adx	QDx	Выход (через образ процесса)	Double word
Awx	QWx	Выход (через образ процесса)	Word
AXx.y	QXx.y	Вывод (через образ процесса)	Bit
Dx.y	Dx.y	Блок данных	Bit
DBx	DBx	Блок данных	Byte
DDx	DDx	Блок данных	Double word
DWx	DWx	Блок данных	Word
DXx.y	DXx.y	Блок данных	Bit
Ex.y	Ix.y	Вход (через образ процесса)	Bit
Ebx	Ibx	Вход (через образ процесса)	Byte

Мнемоника (SIMATIC)	Мнемоника (IEC)	Адреса	Тип данных
EDx	IDx	Вход (через образ процесса)	Double word
Ewx	IWx	Вход (через образ процесса)	Word
EXx.y	IXx.y	Вход (через образ процесса)	Bit
Mx.y	Mx.y	Область меркеров	Bit
MBx	MBx	Область меркеров	Byte
MDx	MDx	Область меркеров	Double word
MWx	MWx	Область меркеров	Word
MXx	MXx	Область меркеров	Bit
PABx	PQBx	Выход (Адресация к периферии)	Byte
PADx	PQDx	Выход (Адресация к периферии)	Double word
PAWx	PQWx	Выход (Адресация к периферии)	Word
PEBx	PIBx	Вход (Адресация к периферии)	Byte
PEDx	PIDx	Вход (Адресация к периферии)	Double word
PEWx	PIWx	Вход (Адресация к периферии)	Word

x = число между 0 и 65535 (абсолютный адрес)

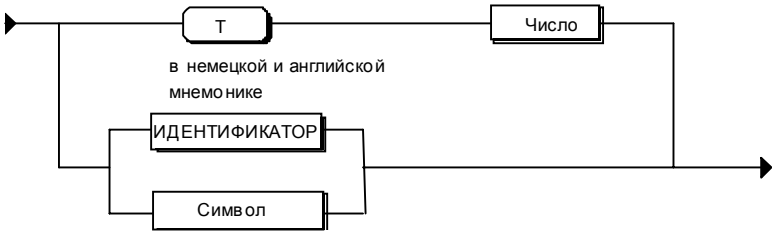
Примеры:

I1.0 MW10 PQW5 DB20.DW3

4.8 Идентификатор таймера

Правила

В SCL существует несколько способов, с помощью которых можно определить таймер. В качестве номера таймера Вы можете задать целое десятичное число.



4.9 Идентификатор счетчика

Правила

В SCL существует несколько способов, с помощью которых можно определить счетчик. В качестве номера счетчика Вы можете задать целое десятичное число.

4.10 Числа

В SCL есть несколько способов написания чисел. К ним применяются следующие правила:

- Число может дополнительно иметь знак, знак десятичной дроби и знак порядка (E).
- Число не может содержать запятые или пробелы.
- Чтобы улучшить удобочитаемость, можно использовать, как разделитель, подчеркивание (_).
- Числу может предшествовать плюс (+) или минус (-). Если числу не предшествует знак, оно считается положительным.
- Числа не должны выходить за пределы определенного диапазона.

Целые

Целое число не содержит ни знака десятичной дроби, ни знака порядка. Это значит, что целое число - это просто последовательность цифр, которым может предшествовать знак плюс или минус. В SCL есть два целых типа - INT и DINT, отличающихся диапазонами возможных значений.

Пример правильной записи целых чисел:

0	1	+1	-1
743	-5280	600_00	-32_211

Следующие целые числа записаны **неправильно** по причинам, указанным рядом:

123,456	Целые не могут содержать запятую.
36.	Целые не могут содержать знак десятичной дроби
10 20 30	Целые не должны содержать пробел.

В SCL Вы можете представлять целые числа в различных числовых системах, но целому должно предшествовать ключевое слово для числовой системы. Ключевое слово 2# стоит перед двоичной системой, 8# перед восьмеричной системой и 16# перед шестнадцатеричной системой.

Правильные целые для десятичного числа 15:

2#1111 8#17 16#F

Действительные числа

Действительные числа могут содержать знак десятичной дроби или порядок числа (или и то и другое). Знак десятичной дроби должен стоять между двумя цифрами. Это значит, что действительное число не может начинаться или заканчиваться знаком десятичной дроби.

Примеры правильной записи действительных чисел:

0.0	1.0	-0.2	827.602
50000.0	-0.000743	12.3	-315.0066

Эти действительные числа записаны **неправильно:**

1.	Должны быть цифры по обе стороны от знака десятичной дроби.
1,000.0	Целые числа не содержат запятой.
.3333	Должны быть цифры по обе стороны от знака десятичной дроби.

Действительные числа могут включать в себя порядок числа, который определяет позицию знака десятичной дроби. Если число не содержит знака десятичной дроби, предполагается положение знака десятичной дроби справа от цифры. Порядок числа должен быть положительным или отрицательным целым числом. Основание степени (10) обозначается буквой E.

Величина 3 x 10 в степени 10 может быть представлено в SCL следующим образом:

3.0E+10	3.0E10	3e+10	3E10
0.3E+11	0.3e11	30.0E+9	30e9

Следующие числа записаны **неправильно:**

3.E+10	Должны быть цифры по обе стороны от знака десятичной дроби.
8e2.3	Степень должна быть целым числом.
.333e-3	Должны быть цифры по обе стороны от знака десятичной дроби.
30 E10	Число не должно содержать пробел.

4.11 Символьные строки

Описание

Символьная строка – это последовательность символов (букв, чисел и специальных символов), заключенных в кавычки.

Примеры правильных символьных строк:
'RED' '76181 Karlsruhe' '270-32-3456'
'DM19.95' 'The correct answer is:'

Правила

Вы можете ввести специальные символы форматирования в кавычках (') или с символом $.

Исходный текст	После компиляции
'SIGNAL$'RED$"	SIGNAL'RED'
'50.0$$'	50.0$
'VALUE$P'	VALUE *page break*
'RUL$L'	RUL *line feed*
'CONTROLLER$R	CONTROLLER *carriage return*
'STEP$T'	STEP *tabulator*

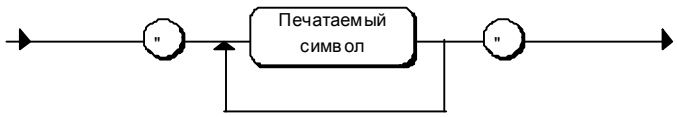
Чтобы ввести непечатаемые символы, используйте представление символа в шестнадцатеричном коде в виде $*hh*, где *hh* код ASCII символа, выраженный в шестнадцатеричной системе.

Чтобы ввести комментарии в символьной строке, которые не надо печатать или отображать на дисплее, используйте символы $> и $< , чтобы защитить комментарии.

4.12 Символ

Вы можете ввести символы в SCL, используя следующий синтаксис. В кавычках только то, что не подчиняется правилу Идентификатора.

Синтаксис:



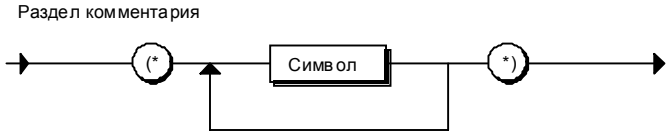
4.13 Раздел комментариев

Правила

- Раздел комментариев может занимать несколько строк и располагается между символами `(*' и `*)'.
- Настройка по умолчанию разрешает вложенные секции. Однако, Вы можете изменить установку и запретить вложенность комментариев.
- Комментарии не должны находиться в середине имени символа или константы. Однако, они могут находиться в середине строки.

Синтаксис

Раздел комментариев формально представлен этой диаграммой:



Пример

```
(* Это пример раздела комментариев
из нескольких строк.*)

SWITCH := 3 ;
END_FUNCTION_BLOCK
```

4.14 Строчный комментарий

Правила

- Строчный комментарий начинается с символов "//" и продолжается до конца строки.
- Длина комментария ограничена 254 символами, включая начальные символ '//'.
- Комментарии не могут находиться в середине имени символа или константы.

Синтаксис

Строчный комментарий формально показан на этой диаграмме:

Пример

```
VAR
     SWITCH : INT ; // строчный комментарий
END_VAR
```

Замечания

- Комментарии в разделе деклараций, начинающиеся с // , включаются в интерфейс блока и могут быть показаны в редакторе LAD/STL/CSF .
- Печатные символы приведены в главе "Описание языка".
- В строчном комментарии пара символов "(*" и "*)" не имеет значения.

4.15 Переменные

Идентификаторы, значения которых изменяются в течение программы, называются переменными. Каждая переменная должна быть индивидуально объявлена перед использованием в логическом блоке или блоке данных. Объявление переменных означает, что идентификатор является переменной (а не константой и т.д.), и устанавливает тип данных для переменной.

По области применения различаются следующие типы переменных:

- Локальные данные
- Глобальные данные пользователя
- Постоянные встроенные переменные (области памяти CPU)

Локальные данные

Локальные данные декларируются в логическом блоке (FC, FB, OB) и имеют значение только в контексте этого блока. Это следующие переменные:

Переменная	Объяснение
Статические переменные	Статические переменные это локальные переменные, величина которых сохраняется в течение, и после работы блока (память блока). Они используются для хранения данных функционального блока.
Временные переменные	Временные переменные принадлежат локально логическому блоку и не занимают статическую область памяти. Эти переменные сохраняются только во время работы блока. Временные переменные не могут быть использованы вне блока, в котором они объявлены.
Параметры блока	Параметры блока - это формальные параметры функционального блока или функции. Они являются локальными переменными, которые используются для передачи фактических параметров при вызове блока.

Глобальные данные пользователя

Эти данные или области данных могут быть доступны в любой точке программы. Чтобы использовать глобальные данные пользователя, мы должны создать блок данных (DB).

Когда Вы создаете DB, Вы определяете его структуру. Вместо объявления структуры Вы можете использовать ссылку на данные, определенные пользователем (UDT). Порядок, в котором Вы определяете структурные компоненты, определяет порядок расположения данных в DB.

Области памяти CPU

Из любой точки программы Вы можете иметь доступ к областям памяти CPU, непосредственно используя идентификаторы адреса, без объявления этих переменных.

Помните, что Вы можете адресовать эти области памяти символически. Назначение символов в этой случае делается глобально через таблицу символов STEP 7.

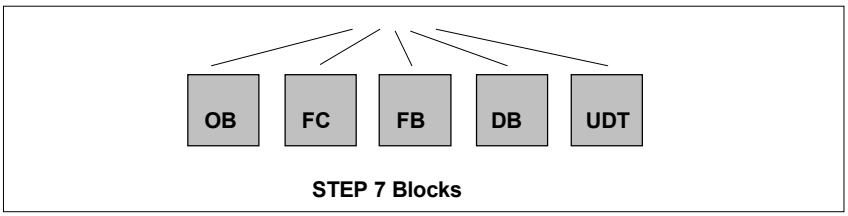
5 Структура SCL программы

5.1 Блоки в исходном файле SCL

В исходном файле SCL может быть несколько блоков. Блоки STEP 7 – это части пользовательской программы, различающиеся согласно их функции, их структуре или по их использованию.

Типы блоков

Существуют следующие типы блоков:



Готовые блоки

Вы не должны программировать сами каждую функцию. Вы можете использовать различные готовые блоки. Они доступны в операционной системе CPU или библиотеках *(S7lib)* стандартного пакета STEP 7 и могут быть использованы, например, для программирования задач связи ПЛК.

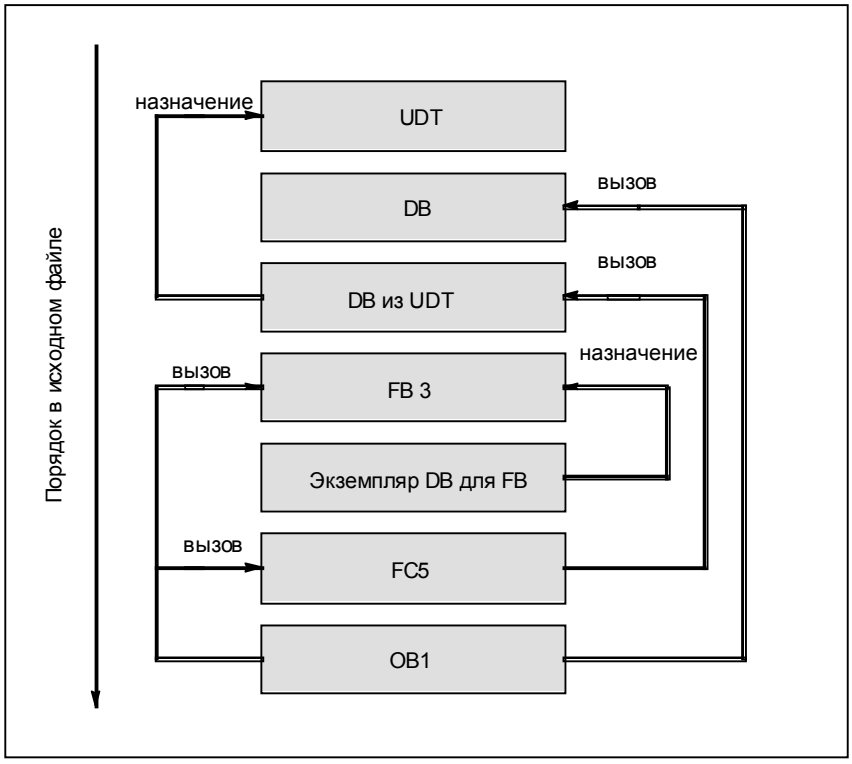
5.2 Порядок следования блоков

Здесь представлены общие правила:

Вызываемые блоки находятся перед вызывающими блоками.

Это означает следующее:

- Тип данных, определяемый пользователем (UDT), должен предшествовать блоку, в котором он используется.
- Блоки данных, в которых используются данные, определенные пользователем (UDT), должны следовать за UDT.
- Блоки данных, которые могут быть доступны логическим блокам, должны предшествовать им.
- Экземплярные блоки данных должны следовать после функциональных блоков, с которыми они связаны.
- Организационный блок OB1, который вызывает другие блоки, стоит в самом конце. Блоки, которые вызываются блоком OB1, должны предшествовать вызывающим блокам.
- Блоки, которые Вы вызываете из исходного файла, но не программируете в этом же исходном файле, должны быть созданы в пользовательской программе до компиляции файла.



5.3 Общая структура блока

Блок состоит из следующих областей:

- Блок начинается ключевым словом и номером блока или символическим именем блока, например, "ORGANIZATION_BLOCK OB1" для организационного блока.
  Для функции определяется тип возвращаемой величины. Если Вы не хотите возвращать величину, примените ключевое слово VOID.
- По желанию, блоку может быть присвоено заголовок, который следует за словом "TITLE =".
- По желанию, комментарий блока. Комментарий блока может быть занимать несколько строк, в начале каждой стоит "//".
- Определение атрибутов блока (дополнительно)
- Определение системных атрибутов блока (дополнительно)
- Раздел описания переменных (Зависит от типа блока)
- Раздел инструкций в логическом блоке или присвоение фактических величин в блоке данных (дополнительно)
- Блок оканчивается строкой END_ORGANIZATION_BLOCK, END_FUNCTION_BLOCK или END_FUNCTION

5.4 Начало и конец блока

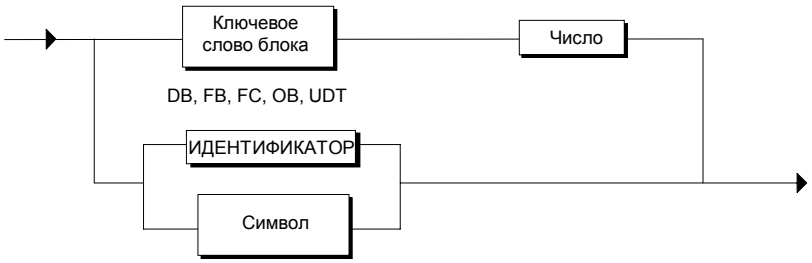
В зависимости от типа блока, исходный текст простого блока начинается стандартным идентификатором для начала и имени блока. Блок завершается стандартными идентификаторами конца.

Синтаксис для различных типов блоков находится в этой таблице:

Идентификатор	**Тип блока**	**Синтаксис**
Функциональный блок	FB	FUNCTION_BLOCK fb_name . . . END_FUNCTION_BLOCK
Функция	FC	FUNCTION fc_name : function type . . . END_FUNCTION
Организационный блок	OB	ORGANIZATION_BLOCK ob_name . . . END_ORGANIZATION_BLOCK
Блок данных	DB	DATA_BLOCK db_name . . . END_DATA_BLOCK
Глобальный тип данных	UDT	TYPE udt_name . . . END_TYPE

Имя блока

В таблице, *xx_name* – имя блока согласно следующему синтаксису:



Номер блока может быть числом от 0 до 65533, обозначение блока данных DB0, зарезервировано.

Имейте в виду, что идентификатор или символ Вы должны определить в таблице символов STEP 7.

Пример

```
FUNCTION_BLOCK FB10
FUNCTION_BLOCK Controller Block
FUNCTION_BLOCK "Controller.B1&U2"
```

5.5 Атрибуты блока

Определение

Атрибуты блока – это используемые Вами свойства блока, например, определение типа, версии, автора или защиты блока. В STEP 7 Вы можете посмотреть атрибуты в окне свойств, когда Вы выберете блок для использования.

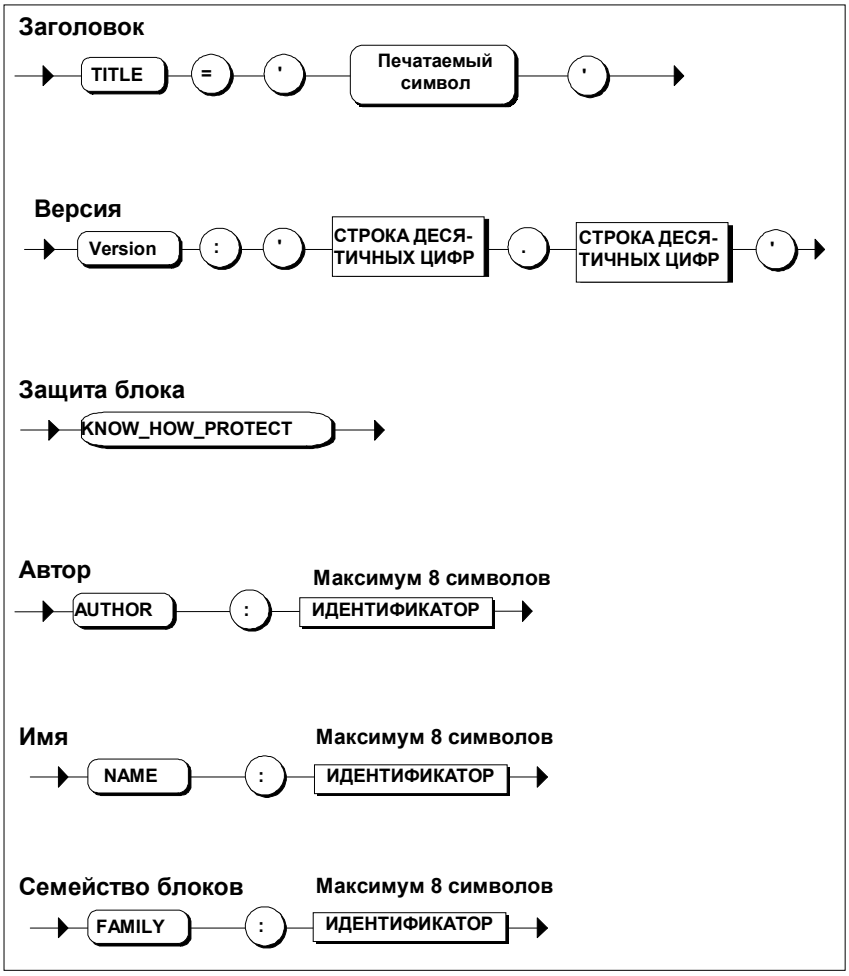
Вы можете присвоить следующие атрибуты:

Ключевое слово/атрибут	**Объяснение**	**Примеры**
TITLE = 'печатаемые символы'	Заголовок блока	TITLE='SORT'
VERSION :'Строка десятичных цифр. Строка десятичных цифр'	Номер версии блока(от 0 до 15) Замечание: В блоках данных (DB), атрибут версия не заключается в кавычки.	VERSION : '3.1' //With a DB: VERSION : 3.1
KNOW_HOW_PROTECT	Защита блока; блок, скомпилированный с этой функцией, не может быть открыт STEP 7.	KNOW_HOW_PROTECT
AUTHOR :	Имя автора: название компании, название отдела или другое имя	AUTHOR : Siemens
NAME :	Имя блока (идентификатор)	NAME : PID
FAMILY :	Название группы блоков: например, motors. Это сохраняет блок в группе блоков, чтобы он мог быть найден быстрее (идентификатор).	FAMILY : example

Правила

- Вы объявляете атрибуты блока, используя ключевые слова после оператора начала блока.
- Идентификатор не должен превышать 8 символов.

Синтаксис для ввода атрибутов блока показан ниже:



Примеры

```
FUNCTION_BLOCK FB10
TITLE = 'Mean_Value'
VERSION : '2.1'
KNOW_HOW_PROTECT
AUTHOR : AUT_1
```

5.6 Комментарии блока

Вы можете ввести комментарии для каждого блока в заголовке блока после строки "TITLE:". Вы используете строчные комментарии. Комментарии могут занимать несколько срок, в начале каждой "//".

Комментарий блока будет показан, например, в окне Properties (Свойства) в SIMATIC Manager или в редакторе LAD/STL/FBD.

Пример

```
FUNCTION_BLOCK FB15
TITLE=MY_BLOCK
//This is a block comment.
//It is entered as a series of line comments
//and can be displayed, for example, in the SIMATIC Manager.
AUTHOR...
FAMILY...
```

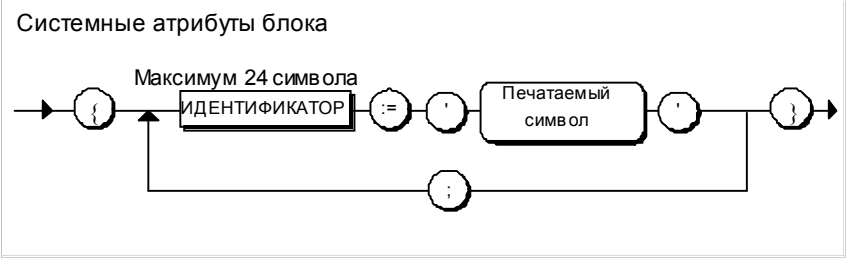
5.7 Системные атрибуты блока

Описание

Системные атрибуты – это атрибуты системного управления, которые верны за областью приложения. Системные атрибуты относятся к блоку в целом.

Правила

- Вы определяете системные атрибуты сразу после оператора начала блока.
- Синтаксис показан ниже:



Примеры

```
FUNCTION_BLOCK FB10
{S7_m_c := 'true' ;
S7_blockview := 'big'}
```

5.8 Раздел деклараций

Описание

Раздел деклараций используется для объявления локальных переменных, параметров, констант и меток.

- Локальные переменные, параметры, константы и метки верны в том логическом блоке, в разделе деклараций которого они объявлены .
- Область данных, которую Вы хотите сделать доступной для любых логических блоков, нужно объявлять в разделе деклараций блоков данных.
- В разделе деклараций UDT, Вы определяете тип данных, определяемый пользователем.

Структура

Раздел деклараций делится на подразделы, заключенные между парой своих ключевых слов. Каждый подраздел содержит список объявлений данных этого типа. Эти подразделы могут располагаться в любом порядке. Следующая таблица показывает возможные подразделы:

Данные	Синтаксис	FB	FC	OB	DB	UDT
Константы	CONST список объявлений END_CONST	X	X	X		
Метки	LABEL список объявлений END_LABEL	X	X	X		
Временные переменные	VAR_TEMP список объявлений END_VAR	X	X	X		
Статические переменные	VAR список объявлений END_VAR	X	X *)		X **)	X **)
Параметры входа	VAR_INPUT список объявлений END_VAR	X	X			
Параметры выхода	VAR_OUTPUT список объявлений END_VAR	X	X			
Параметры входа/выхода	VAR_IN_OUT список объявлений END_VAR	X	X			

*) Хотя объявление переменных между ключевыми словами VAR и END_VAR в функциях разрешено, переменные перемещаются во временную область, когда компилируется исходный файл.

**) В DB и UDT ключевые слова VAR и END_VAR заменяются STRUCT и END_STRUCT соответственно.

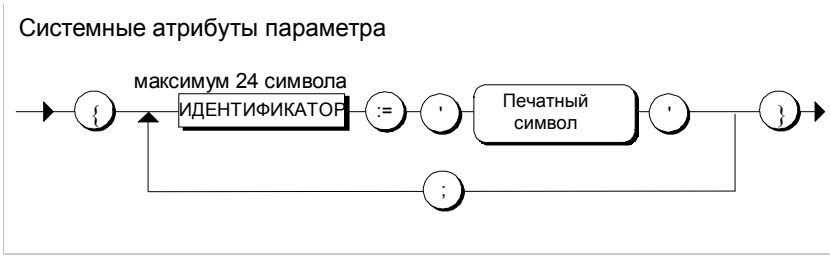
5.9 Системные атрибуты параметров

Описание

Системные атрибуты – это атрибуты системного управления, которые верны за областью приложения. Они используются, например, для конфигурации сообщений или связей. Системные атрибуты параметров применяются только для конфигурации отдельных параметров. Вы можете применить системные атрибуты к параметрам входа, выхода и параметрам входа /выхода.

Правила

- Вы назначаете системные атрибуты параметров в декларационном поле параметров входа, выхода и параметров входа /выхода.
- Идентификатор не должен превышать 24 символа.
- Синтаксис показан ниже:



Пример

```
VAR_INPUT
      in1    {S7_server:='alarm_archiv';
             S7_a_type:='ar_send'}: DWORD ;
END_VAR
```

5.10 Раздел операторов

Описание

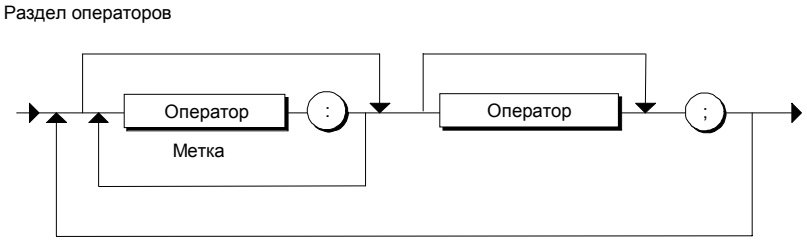
Раздел операторов содержит команды, которые выполняются, когда вызван логический блок. Эти команды используются для передачи данных и адресов.

Раздел операторов блока данных содержит команды для инициализации переменных.

Правила

- Если Вы хотите, то можете начать раздел операторов ключевым словом BEGIN. BEGIN обязателен для раздела операторов блока данных. Раздел операторов завершается ключевым словом конца блока.
- Каждый оператор заканчивается точкой с запятой.
- Идентификаторы, используемые в разделе операторов, должны быть заранее объявлены.
- Если требуется, Вы можете ввести метки перед каждым оператором.

Синтаксис показан ниже:



Пример

```
BEGIN
      INITIAL_VALUE       :=0;
      FINAL_VALUE         :=200;
.
.
STORE:    RESULT          :=SETPOINT;
.
.
END_FUNCTION_BLOCK
```

5.11 Операторы

Описание

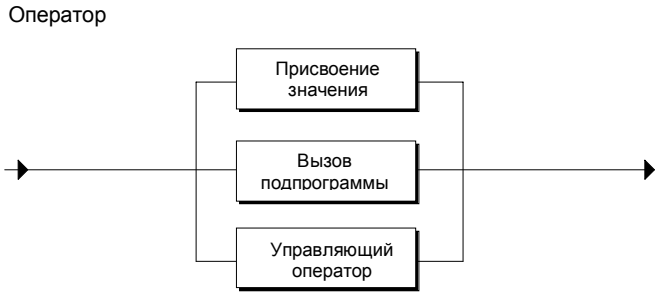
Оператор – это наименьшая замкнутая часть пользовательской программы. Она представляет инструкцию процессору, как производить конкретные операции.

В SCL используются следующие типы операторов:

- Присвоение значений используется для передачи результата выражения или величины одной переменной в другую.
- Операторы управления используются либо для повторения оператора или группы операторов, либо для ветвления программы.
- Подпрограммы вызова используются для вызова функций или функциональных блоков.

Правила

Синтаксис для ввода показан ниже:



Пример

Следующие примеры показывают различные виды операторов:

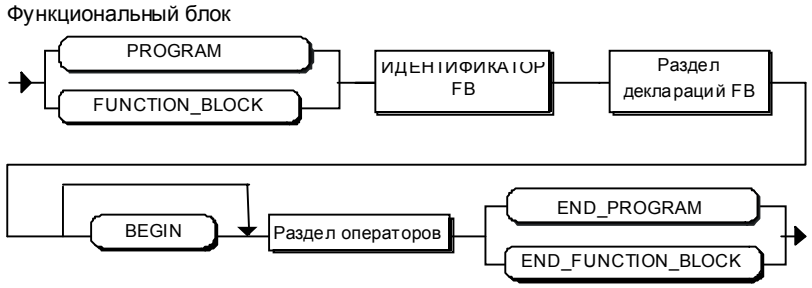
```
// пример присвоения значений
MEASVAL:= 0 ;

// пример вызова подпрограммы
FB1.DB11 (TRANSFER:= 10) ;

// пример оператора управления
WHILE COUNTER < 10 DO..
.
.
END_WHILE;
```

5.12 Структура функционального блока (FB)

Описание

Функциональный блок (FB) – это логический блок, который содержит часть программы и который использует определенную область памяти. Всякий раз, когда FB вызван, ему должен быть присвоен экземпляр DB. Структуру этого экземпляра DB Вы определяете в разделе деклараций FB.

Синтаксис



Идентификатор FB

После ключевого слова FUNCTION_BLOCK или PROGRAM, введите идентификатор функционального блока как ключевое слово “FB” с последующим номером блока или как символьное имя FB. Номер блока может быть от 0 до 65533.

Примеры:
FUNCTION_BLOCK FB10
FUNCTION_BLOCK MOTOR1

Раздел деклараций FB

Раздел деклараций FB используется для описания собственных данных блока. Возможные разделы деклараций подробно описаны в главе "Раздел деклараций". Помните, что раздел деклараций FB определяет и структуру экземпляров DB.

Пример

В этом примере показан исходный текст функционального блока. Параметру входа (в данном случае, V1) присвоено начальные значения.

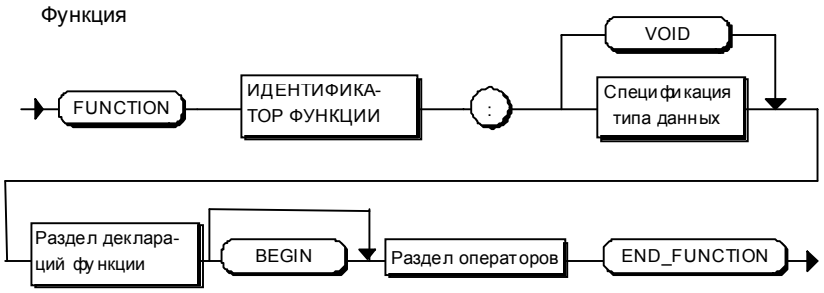
```
FUNCTION_BLOCK FB11
VAR_INPUT
   V1 : INT := 7 ;
END_VAR
VAR_OUTPUT
   V2 : REAL  ;
END_VAR
VAR
   FX1, FX2, FY1, FY2  :  REAL ;
END_VAR

BEGIN
  IF V1 = 7 THEN
    FX1 := 1.5 ;
    FX2 := 2.3 ;
    FY1 := 3.1 ;
    FY2 := 5.4 ;
// Вызов функции FC11 и использование статических переменных
// для входных параметров.
    V2 := FC11 (X1:= FX1, X2 := FX2, Y1 := FY1, Y2 := FY2) ;
   END_IF ;
END_FUNCTION_BLOCK
```

5.13 Структура функции (FC)

Описание

Функция (FC) – это логический блок, которому не соответствует собственная область памяти для хранения данных. Ей не требуется экземпляр DB. В отличие от FB, функция может возвращать свой результат (возвращаемая величина) в точку вызова. Поэтому функция также может быть использована в выражении подобно переменной. Функции типа VOID не имеют возвращаемой величины.

Синтаксис



Идентификатор FC

После ключевого слова "FUNCTION", введите идентификатор функции как ключевое слово “FC” с последующим номером блока или как символьное имя функции. Номер блока может быть от 0 до 65533.

Пример
FUNCTION FC17 : REAL
FUNCTION FC17 : VOID

Спецификация типа данных

Спецификация типа данных определяет тип данных возвращаемой функцией величины. Разрешены все типы данных, кроме STRUCT и ARRAY. Если возвращаемая величина не нужна, вместо типа данных приводится слово VOID.

Раздел деклараций FC

Раздел деклараций FC используется для объявления локальных данных функции (временных переменных, параметров входа, параметров выхода, параметров входа/выхода, констант, меток).

Раздел операторов FC

В разделе операторов имени функции должен быть присвоен результат ее выполнения. Это необязательно, если функция имеет тип VOID. Пример правильного оператора для функции FC31:
FC31:= VALUE;

Пример

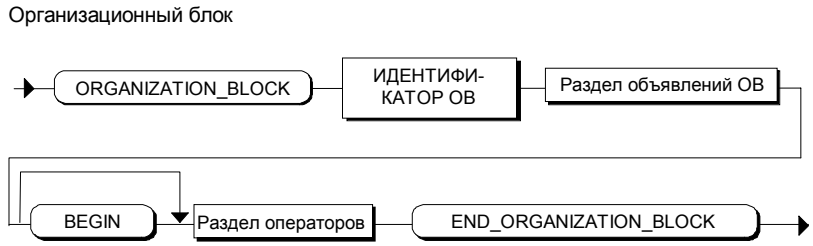
```
FUNCTION FC11: REAL
VAR_INPUT
   x1: REAL ;
   x2: REAL ;
   x3: REAL ;
   x4: REAL ;
END_VAR
VAR_OUTPUT
     Q2: REAL ;
END_VAR
BEGIN
 // возвращаемая функцией величина
 FC11:= SQRT( (x2 - x1)**2 + (x4 - x3) **2 ) ;
 Q2:= x1 ;
END_FUNCTION
```

5.14 Структура организационного блока (OB)

Описание

Организационный блок (OB), подобен FB или FC, это часть пользовательской программы, которая вызывается операционной системой циклически или как реакция на определенные события. Вызов OB обеспечивает интерфейс между пользовательской программой и операционной системой.

Синтаксис



Идентификатор OB

После ключевого слова "ORGANIZATION_BLOCK", введите идентификатор блока, как ключевое слово “OB” с последующим номером блока или как символьное имя OB. Номер блока может быть от 1 до 65533.

Примеры

```
ORGANIZATION_BLOCK OB1
ORGANIZATION_BLOCK ALARM
```

Раздел деклараций OB

Раздел деклараций OB используется для объявления локальных данных (временных переменных, констант, меток).

Каждый OB при выполнении требует 20 байт локальных данных для взаимодействия с операционной системой. Для этого Вы должны объявить соответствующий массив. Если Вы используете вставку шаблона OB, объявление этого массива уже предусмотрено.

Пример

```
ORGANIZATION_BLOCK OB1
VAR_TEMP
   HEADER : ARRAY [1..20] OF BYTE ; //20 байт для ОС
END_VAR
BEGIN
   FB17.DB10 (V1 := 7) ;
END_ORGANIZATION_BLOCK
```

5.15 Структура блока данных (DB)

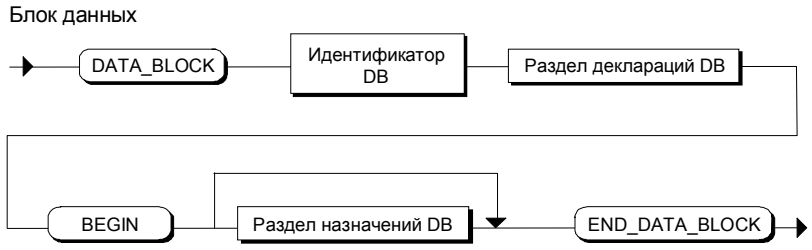
Описание

Глобальные данные, которые доступны всем блокам, хранятся в программе в блоках данных. Каждый логический блок, FB, FC или OB, может читать или переписывать эти блоки данных.

Есть два типа блоков данных:

- **Блоки данных**
  Блоки данных, которые доступны всем логическим блокам S7 программы. Каждый блок FB, FC или OB может читать или переписывать данные блоков.
- **Блоки данных, назначаемые FB (экземпляры DB)**
  Экземпляры блоков данных – это блоки данных, которые присвоены конкретному функциональному блоку (FB). Они содержат локальные данные для функционального блока, которому они присвоены. Эти блоки данных автоматически создаются компилятором SCL, если в пользовательской программе вызван FB.

Синтаксис



Идентификатор DB

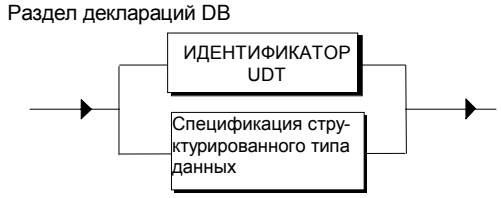
После ключевого слова "DATA_BLOCK" введите идентификатор блока как ключевое слово “DB” с последующим номером блока, или как символьное имя DB. Номер блока может быть от 1 до 65533.

Примеры:
DATA_BLOCK DB20
DATA_BLOCK MEASRANGE

Раздел деклараций DB

В разделе деклараций Вы определяете структуру DB. Есть два пути для этого:

- **Назначая тип данных, определенный пользователем**
  Вы можете сослаться на идентификатор определенного пользователем типа данных, приведенного ранее в программе. Блок данных получает структуру UDT. В разделе назначения DB Вы можете дать переменным первоначальные значения.
- **Определение типа данных STRUCT**
  С помощью типа данных STRUCT, Вы определяете тип данных и начальное значение для каждой переменной, которая хранится в DB.



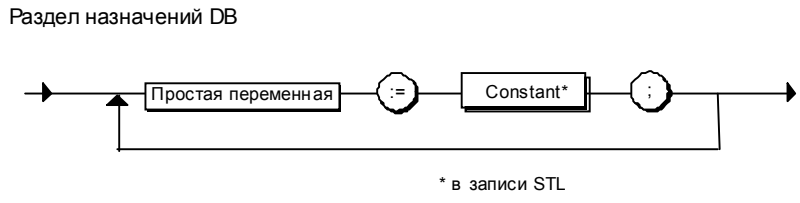
Пример

```
DATA_BLOCK DB20
   STRUCT // раздел деклараций
      VALUE:ARRAY  [1..100] OF INT;
   END_STRUCT
BEGIN                  // начало раздела назначения
:
END_DATA_BLOCK         // конец блока данных
```

Раздел инициализации DB

В разделе инициализации Вы можете адаптировать начальные данные, которые Вы объявили в разделе деклараций, так чтобы они имели разные значения в частных случаях.

Раздел назначения начинается ключевым словом BEGIN и затем идет последовательность назначений начальных значений.



Когда назначаются начальные величины, к атрибутам входа и комментариям применяется синтаксис STL. Для информации по написанию констант, атрибутов и комментариев, используйте online помощь STL или документацию по STEP 7.

Пример

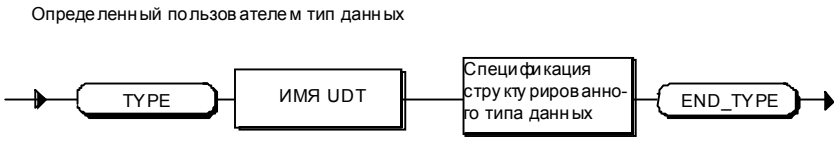
```
// Инициализация блока данных с данными типа STRUCT
DATA_BLOCK DB10
      STRUCT // Данные с первоначальными значениями
        VALUE      :      ARRAY [1..100] OF INT := 100 (1) ;
        SWITCH     :      BOOL          := TRUE ;
        S_WORD     :      WORD  := W#16#FFAA ;
        S_BYTE     :      BYTE   := B#16#FF ;
        S_TIME     :      S5TIME        := S5T#1h30m10s ;
      END_STRUCT

BEGIN      // Раздел назначения
   // Назначение величин для специфического массива элементов
   VALUE [1] := 5;
   VALUE [5] := -1 ;

END_DATA_BLOCK

// Блок данных с типом данных, определенным пользователем
DATA_BLOCK DB11
   UDT 51
BEGIN
END_DATA_BLOCK
```

5.16 Структура типа данных, определенного пользователем

Тип данных, определяемый пользователем (UDT) – специальная структура данных, которую Вы создаете сами. Как только будет создан тип данных, определяемый пользователем, определенные в нем структуры данных могут использоваться многократно. Они могут использоваться в любой точке программы CPU; другими словами, это глобальный тип данных. Данные также могут использоваться:

- В разделах деклараций блоков так же, как элементарные или комплексные типы данных, или
- Как шаблоны для создания блоков данных с подобной структурой.

Когда используете типы данных, определяемые пользователем, помните, что они должны находится в исходном файле SCL перед блоками, в которых они используются.



Идентификатор UDT

После ключевого слова TYPE, введите ключевое слово UDT с номером или символьное имя UDT. Номер блока может быть от 0 до 65533.

Примеры:

TYPE UDT10
TYPE SUPPLYBLOCK

Определение типа данных

Тип данных всегда определяется с помощью спецификации типа **STRUCT**. Тип данных UDT может использоваться в подразделах деклараций логических блоков, блоков данных или назначаться как структура DB.

Пример описания UDT

```
TYPE MEASVALUES
STRUCT
// UDT объявлено символическим идентификатором
   BIPOL_1 : INT := 5;
   BIPOL_2 : WORD := W#16#FFAA ;
   BIPOL_3 : BYTE  := B#16#F1 ;
   BIPOL_4 : WORD := B#(25,25) ;
   MEASURE : STRUCT
      BIPOLAR_10V : REAL ;
      UNIPOLAR_4_20MA : REAL ;
   END_STRUCT ;
END_STRUCT ;
END_TYPE

//использование UDT в FB
FUNCTION_BLOCK FB10
VAR
   MEAS_RANGE : MEASVALUES;
END_VAR
BEGIN
   // . . .
   MEAS_RANGE.BIPOL_1 := -4 ;
   MEAS_RANGE.MEASURE.UNIPOLAR_4_20MA  :=  2.7 ;
   // . . .
END_FUNCTION_BLOCK
```

6 Типы данных

6.1 Обзор типов данных в SCL

Типы данных определяют:

- Тип и интерпретацию элементов данных,
- Диапазон значений для элементов данных,
- Множество операций, которые могут выполняться с адресом типа данных
- Способ записи элементов данных

Элементарные типы данных

Элементарные типы данных определяют структуру элементов данных, которая не может быть разделена на меньшие части. Они соответствуют стандарту DIN EN 1131-3. Элементарные типы данных описывают области памяти фиксированной длины и включают бит, целые, действительные, период времени, время дня и символьные величины. В SCL представлены следующие элементарные типы данных:

Группа	Тип данных	Объяснение
Битовый типы данных	BOOL BYTE WORD DWORD	Элементы данных этих типов занимают 1 бит, 8 бит, 16 или 32 бита
Символьный тип	CHAR	Элементы данных этого типа занимают 1 символ в наборе символов ASCII
Численные типы	INT DINT REAL	Элементы данных этих типов доступны для обработки числовых величин.
Временные типы	TIME DATE TIME_OF_DAY S5TIME	Элементы данных этих типов представляют временные величины данных в соответствии с STEP 7.

Сложные типы данных

SCL поддерживает следующие виды сложных типов:

Тип данных	Объяснение
DATE_AND_TIME DT	Определяет область из 64 бит (8 байт). Этот тип данных содержит дату и время (в двоично-десятичных кодах). Это встроенный тип данных в SCL.
STRING	Определяет область для символьной строки, содержащей до 254 символов (тип данных CHAR).
ARRAY	Определяет массив, состоящий из элементов данных одного типа (элементарных или сложных).
STRUCT	Определяет группу данных любых типов, простых или сложных, в любых комбинациях.

Определенный пользователем тип данных

Вы можете создавать свои типы данных, путем их декларации. Каждый из них имеет уникальное имя и может использоваться в программе многократно. Как только тип определен, этот тип данных может использоваться для создания нескольких блоков данных с одинаковой структурой.

Параметрируемый тип

Параметрируемый тип – это специальный тип данных для таймеров, счетчиков и блоков, которые используются как формальные параметры.

Тип данных	Объяснение
TIMER	Используется для объявления таймеров как параметров.
COUNTER	Используется для объявления счетчиков как параметров.
BLOCK_xx	Используется для объявления FC, FB, DB и SDB как параметров.
ANY	Используется как параметр для доступа к областям памяти с данными определенного типа.
POINTER	Используется как параметр для доступа к области памяти.

Тип данных ANY

В SCL Вы можете использовать переменные типа ANY как формальный параметр блока. Вы также можете создать временные переменные этого типа и использовать их в назначении величин.

6.2 Элементарные типы данных

6.2.1 Битовые типы данных

Данные этих типов - комбинации битов, которые занимают 1 бит (тип данных BOOL), 8, 16 или 32 бита. Числовые данные не могут быть определены как данные типа: byte, word и double word. Это комбинации битов, которые используются только для формулирования логических выражений.

Тип	Ключевое слово	Битовая ширина	Выравнивание	Диапазон величин
Бит	BOOL	1 бит	Начинается с наименее значимого бита в байте	0, 1 или FALSE, TRUE
Байт	BYTE	8 битов	Начинается с наименее значимого байта в слове.	-
Слово	WORD	16 бит	Начинается с границы WORD .	-
Двойное слово	DWORD	32 бита	Начинается с границы WORD.	-

6.2.2 Символьный тип

Элементы данных этого типа представляют один символ из набора символов ASCII.

Тип	Ключевое слово	Битовая ширина	Диапазон величин
Единственный символ	CHAR	8	Расширенный набор символов ASCII

6.2.3 Численные типы данных

Эти типы служат для обработки численных величин (например, для вычисления арифметических выражений).

Тип	Ключевое слово	Битовая ширина	Выравнивание	Диапазон величин
Целый	INT	16	Начинается с границы WORD.	От -32_768 до 32_767
Двойной целый	DINT	32	Начинается с границы WORD.	От -2_147_483_648 до 2_147_483_647
Число с плавающей точкой	REAL	32	Начинается с границы WORD.	От -3.402822E+38 до -1.175495E-38 +/- 0 От 1.175495E-38 to 3.402822E+38

6.2.4 Типы времени

Данные этого типа представляют время и дату в STEP 7 (например, для установки даты или ввода времени).

Тип	Ключевое слово	Битовая ширина	Выравнивание	Диапазон величин
S5 время	S5TIME S5T	16	Начинается с границы WORD.	От T#0H_0M_0S_10MS до T#2H_46M_30S_0MS
Период времени: IEC время с шагом1 ms.	TIME T	32	Начинается с границы WORD.	От -T#24D_20H_31M_23S_647MS до T#24D_20H_31M_23S_647MS
Дата IEC дата с шагом1 день	DATE D	16	Начинается с границы WORD.	От D#1990-01-01 до D#2168-12-31
Время дня время с шагом 1 ms.	TIME_OF_DAY TOD	32	Начинается с границы WORD.	От TOD#0:0:0.0 до TOD#23:59:59.999

Если установленная величина выше, чем верхний предел диапазона, то используется верхний предел величины.
Для переменных типа S5TIME разрешающая способность ограничена, другими словами, доступно только время, основанное на 0.01 s, 0.1 s,1s, 10 s. Компилятор соответственно округляет величины. Если установленная величина выше, чем верхний предел диапазона, то используется верхний предел величины.

6.3 Сложные типы данных

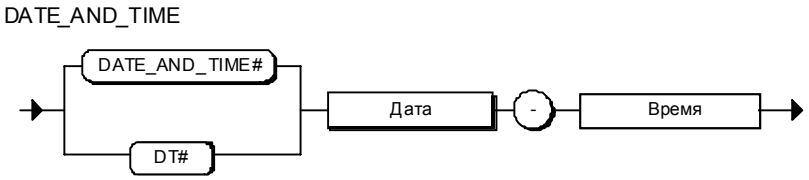
6.3.1 Тип данных DATE_AND_TIME

Описание

Этот тип данных определяет область из 64 бит (8 байт) для даты и времени, которая содержит в двоично-десятичных кодах следующую информацию:

год, месяц, день, час, минуты, секунды, миллисекунды.

Синтаксис



Точный синтаксис для записи даты и времени описан в главе "Объявление констант".

Диапазон величины

Тип	Ключевое слово	Битовая ширина	Выравнивание	Диапазон величины
Дата и время	DATE_AND_TIME DT	64	Начинается с границы WORD.	От DT#1990-01-01-0:0:0.0 до DT#2089-12-31-23:59:59.999

Тип данных Date_And_Time содержит следующие данные в формате BCD:

Байты	**Содержание**	**Диапазон**
0	Год	1990 до 2089
1	Месяц	01 до 12
2	День	1 до 31
3	Час	0 до 23
4	Минута	0 до 59
5	Секунда	0 до 59
6	Миллисекунды (2 старших разряда)	00 до 99
7 (4..7 биты)	Миллисекунды (младший разряд)	0 до 9
7 (0..3 биты)	День недели	1 до 7 (1 = воскресение)

Пример

Правильная запись даты и времени 20 октября 1995 12:20:30 и 10 миллисекунд показана ниже:

```
DATE_AND_TIME#1995-10-20-12:20:30.10
DT#1995-10-20-12:20:30.10
```

Замечание

Вы можете использовать стандартные функции (FC) из библиотеки STEP 7 для определения отдельных компонент DATE или TIME.

6.3.2 Тип данных STRING

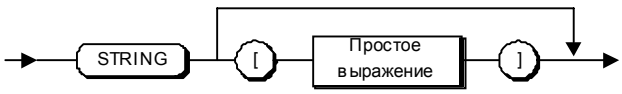
Описание

Тип данных STRING определяет строку символов с максимальной длиной 254 символа. Стандартная область, зарезервированная для строки символов, состоит из 256 байт. Такая область памяти требуется для хранения 254 символов и заголовка из 2 бай.

Вы можете уменьшить память, требующуюся для строки символов, определив максимальное число символов, сохраняемых в строке. Нуль-строка, другими словами, строка без символов, это наименьшая возможная величина строки.

Синтаксис

Спецификация типа данных STRING



Размер строки

Простое выражение задает максимальное количество символов в строке. В строке разрешены все символы ASCII. Строка может также включать специальные символы, например, символы управления и непечатаемые символы. Вы можете ввести их, используя знак $hh, где hh - код символа в ASCII, выраженный в шестнадцатеричной системе (например: '$0D$0AText').

Объявляя область памяти для строки, Вы можете определить максимальное число символов, которое может в ней содержаться. Если Вы не определите максимум, то длина строки будет 254.

Пример

```
VAR
  Text1   : String [123];
  Text2   : String;
END_VAR
```

Константа "123" в объявлении переменной "Text1" задает максимальное числе символов в строке. Для переменной "Text2", зарезервирована длина 254 символа.

Замечание

Для выходных, вход/выходных параметров и возвращаемой функцией величины, Вы можете уменьшить встроенную длину строки (254), используя ресурсы CPU. Для этого выберите команду меню **Options > Customize (Параметры > Настройка)** и введите требуемую длину в графе "Maximum String Length (максимальная длина строки)" на закладке "Compiler (Компилятор)". Помните, что эта установка влияет на все переменные типа STRING в исходном файле. Величина, которую Вы установите, не может быть меньше, чем переменные STRING, использующиеся в программе.

Инициализация строки символов

Строковые переменные, также как и другие, могут быть инициализированы строкой-константой при объявлении параметров функционального блока (FB). Это невозможно для инициализации параметров функций (FC).

Если инициализирующая строка меньше, чем максимальная длина, то оставшиеся символы не инициализируются. Когда переменная обрабатывается программой, во внимание принимаются только занятые к настоящему времени символьные позиции.

Пример

```
x : STRING[7]:='Address';
```

Если требуются временные переменные типа STRING, например, для буферных результатов, они должны быть уже инициализированы строковой константой, либо в объявлении переменных, либо присвоением величин перед тем, как они впервые будут использованы.

Замечание

Если функция из стандартной библиотеки возвращает величину типа STRING и Вы хотите, чтобы эта величина была назначена временной переменной, необходимо сначала инициализировать эту переменную.

Пример

```
FUNCTION Test : STRING[45]
VAR_TEMP
  x : STRING[45];
END_VAR
x := 'a';
x := concat (in1 := x, in2 := x);
Test := x;
END_FUNCTION
```

Без инициализации **x := 'a';** , функция будет возвращать неправильный результат.

Выравнивание

Переменные типа STRING начинаются и заканчиваются границей WORD.

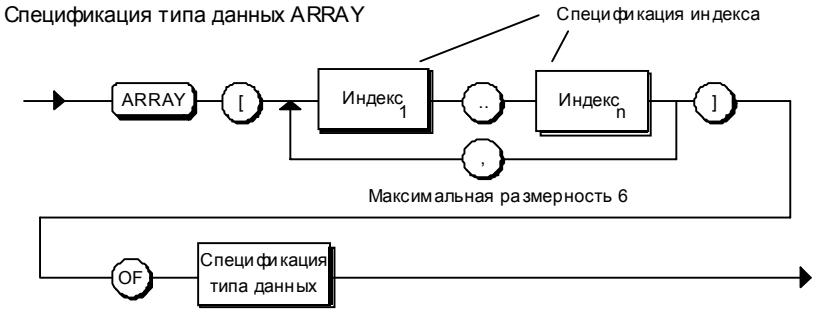
6.3.3 Тип данных ARRAY

Описание

Массив (ARRAY) содержит заданное число компонент одного типа данных. В SCL возможны следующие виды массивов:

- Одномерный массив. Это список элементов данных, расположенных в порядке возрастания.
- Двумерный массив. Это таблица данных, состоящая из строк и колонок. Первое измерение имеет отношение к номеру строки и второе - к номеру колонки.
- Многомерный массив. Это расширение двумерного массива с добавлением других измерений. Максимальное число измерений 6.

Синтаксис



Индексная спецификация

Возможны следующие описания измерений массива:

- Наименьший и наибольший возможный индекс (диапазон) для каждого измерения.
  Индекс может быть любой целой величиной (-32768 до 32767).
- Предел должен делиться на два периода. Отдельные диапазоны индекса должны разделяться запятой.
- Индексная спецификация в целом заключается в квадратные скобки.

Спецификация типа данных

Для спецификации типа данных, Вы объявляете тип данных компонентов. За исключением типа данных ARRAY, все другие типы могут использоваться для спецификации массива. Тип данных массива может быть сложным, например, STRUCT. В качестве компонентов массива не могут использоваться параметрические типы.

Пример

```
VAR
   CONTROLLER1 :
   ARRAY[1..3,1..4] OF INT :=  -54,   736,      -83,       77,
                      -1289,10362,          385,       2,
                         60,  -37,  -7,  103 ;
   CONTROLLER2 : ARRAY[1..10] OF REAL ;
END_VAR
```

Выравнивание

Переменные типа ARRAY размещаются в памяти колонка за колонкой. Каждое измерение переменной типов BOOL, BYTE или CHAR заканчиваются границей BYTE, все другие - границей WORD.

6.3.4 Тип данных STRUCT

Описание

Тип данных STRUCT описывает область, состоящую из фиксированного числа компонентов, которые могут быть данными различных типов. Эти элементы данных (поля) определяются за ключевым словом STRUCT при объявлении компонентов.

Основное свойство типа STRUCT - элемент данных может быть сложным. Это означает, что возможна расширенная (иерархическая) вложенность типа STRUCT.

Синтаксис

Декларация компонентов

Декларация компонентов – это список различных компонентов типа данных STRUCT. Он состоит из:

- От 1 до n идентификаторов назначенного типа и
- Дополнительной спецификации первоначальных величин

Идентификатор – это имя структуры элементов, к которой будет относиться последующая спецификация типа данных.

Все типы данных, за исключением параметрических, разрешены для спецификации.

У Вас есть возможность инициализации элементов после объявления типа данных, используя назначение величин.

Пример

```
VAR
    MOTOR : STRUCT
        DATA : STRUCT
            LOADCURR  : REAL ;
            VOLTAGE   : INT  := 5 ;
        END_STRUCT ;
    END_STRUCT ;
END_VAR
```

Выравнивание

Переменные типа STRUCT начинаются и заканчиваются границей WORD.

6.4 Определенный пользователем тип данных

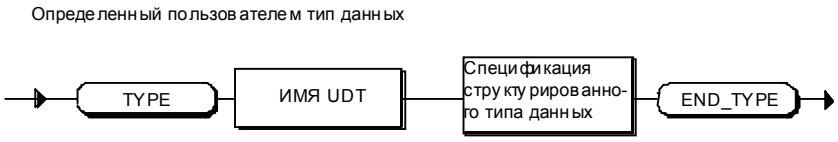
6.4.1 Определенный пользователем тип данных (UDT)

Описание

Вы определяете этот тип данных (UDT) как блок. Как только он будет определен, Вы можете использовать его в Вашей программе, другими словами, это глобальный тип данных. Вы можете использовать этот тип данных в разделе деклараций логического блока или блока данных, ссылаясь на его идентификатор, UDTx (x – это номер), или символическое имя.

Определенный пользователем тип данных может быть использован для объявления переменных, параметров, блоков данных и других типов, определенных пользователем. Компоненты массивов или структур также могут объявляться как определенный пользователем тип данных.

Синтаксис



Идентификатор UDT

Объявление определенного пользователем типа данных начинается ключевым словом TYPE, за которым следует имя типа данных (UDT идентификатор). Имя типа данных, определенных пользователем, может быть определено в абсолютной форме, то есть стандартным именем в виде UDTx (x - номер), или как символьное имя.

Примеры:

```
TYPE UDT10
TYPE MEASVALUES
```

Спецификация типа данных

За идентификатором UDT идет спецификация типа данных. Единственная спецификация типа данных, разрешенная в этом случае, STRUCT.

```
STRUCT
:
END_STRUCT
```

Замечание

В определенном пользователем типе данных должен быть использован синтаксис STL. Например, это относится к записи констант и назначению первоначальных величин (инициализации). Для информации о синтаксисе констант используйте online help STL.

Пример

```
// UDT определено символьным именем
TYPE MEASVALUES:
 STRUCT
  BIPOL_1 : INT := 5;
  BIPOL_2 : WORD := W#16#FFAA ;
  BIPOL_3 : BYTE  := B#16#F1 ;
  BIPOL_4 : WORD := W#16#1919 ;
  MEASURE : STRUCT
      BIPOLAR_10V : REAL ;
      UNIPOLAR_4_20MA : REAL ;
      END_STRUCT;
  END_STRUCT;
END_TYPE

//использование UDT в FB
FUNCTION_BLOCK FB10
VAR
    MEAS_RANGE : MEASVALUES;
END_VAR
BEGIN
  // . . .
  MEAS_RANGE.BIPOL_1 := -4 ;
  MEAS_RANGE.MEASURE.UNIPOLAR_4_20MA  :=  2.7 ;
  // . . .
END_FUNCTION_BLOCK
```

6.5 Параметрические типы данных

6.5.1 Параметрические типы данных

Определяя формальные параметры блоков FB и FC, Вы можете использовать в дополнение к уже знакомым типам данных параметрические типы.

Параметры	Размер	Описание
TIMER	2 байта	В вызываемом логическом блоке программой используется идентификатор таймера. Пример фактического параметра: T1
COUNTER	2 байта	В вызываемом логическом блоке программой используется идентификатор счетчика. Пример фактического параметра: C10
BLOCK_FB BLOCK_FC BLOCK_DB BLOCK_SDB	2 байта	В вызываемом логическом блоке программой используется идентификатор блока. Примеры фактического параметра: FC101, DB42
ANY	10 байт	Используется, если для фактического параметра должен быть разрешен любой тип данных, за исключением ANY.
POINTER	6 байт	Идентифицирует конкретную область памяти, использованную программой. Пример фактического параметра: M50.0

6.5.2 Типы данных TIMER и COUNTER

При вызове блока Вы определяете конкретный таймер или счетчик, использующиеся при работе этого блока. Типы данных TIMER и COUNTER разрешены только для входных параметров (VAR_INPUT).

6.5.3 Блоковые типы данных

С помощью этих типов Вы используете блок как входной параметр FB или FC. При объявлении входных параметров задается тип блока (FB, FC, DB). Для назначения параметров Вы определяете идентификатор блока. Разрешены абсолютный и символьный идентификаторы.

Вы можете иметь доступ к элементу типа BLOCK_DB, используя абсолютную адресацию (myDB.dw10). Но SCL не обеспечивает любые действия для других блочных типов данных. Параметры этих типов используются только при вызове блоков. Когда как параметрический тип используются функции, не могут быть переданы формальные параметры этих функций.

В SCL Вы можете назначить как фактические параметры адреса следующих типов данных:

- Функциональные блоки без фактических параметров
- Функции без формальных параметров или возвращаемых величин (функция VOID)
- Блоки данных и системные блоки данных.

6.5.4 Тип данных POINTER

Параметрический тип POINTER Вы можете использовать для объявления формальных параметров блоков. Если Вы вызываете такой блок, эти параметрам могут назначаться адреса других типов данных (исключая ANY).

SCL обеспечивает, однако, только один оператор для обработки данных типа POINTER, а именно, передачу в вызываемый блок.

Как фактические параметры, совместимые с типом POINTER, Вы можете назначить, следующие адреса:

- Абсолютные адреса
- Символьные имена
- Адреса данных типа POINTER
  Это возможно только тогда, когда адреса являются формальными параметрами с совместимым типом параметра.
- Константу NIL
  Вы определяете указатель на отсутствующий объект .

Ограничения

- Тип данных POINTER разрешен для формальных входных и вход/выходных параметров FB и FC, а также выходных параметров FC. Как фактические параметры не разрешены константы (за исключением нулевой константы NIL).
- Если Вы вводите формальные параметры типа POINTER для временных переменных, Вы не сможете использовать эти параметры в качестве фактических при вызове других FB или FC. Временные переменные при передаче теряют свою достоверность.

Пример

```
FUNCTION FC100 : VOID
VAR_IN_OUT
   N_out : INT;
   out     : POINTER;
END_VAR
VAR_TEMP
   ret     : INT;
END_VAR
BEGIN
   // ...
   ret := SFC79(N := N_out, SA := out);
   // ...
END_FUNCTION

FUNCTION_BLOCK FB100
VAR
   ii   : INT;
   aa : ARRAY[1..1000] OF REAL;
END_VAR

BEGIN
   // ...
   FC100( N_out := ii, out := aa);
   // ...
END_FUNCTION_BLOCK
```

6.6 Тип данных ANY

В SCL Вы можете объявить переменные типа ANY так:

- Как формальные параметры блока; этим параметрам могут при вызове блока назначаться фактические параметры любого типа данных .
- Как временные переменные, Вы можете назначить величины любого типа данных для этих переменных.

Вы можете использовать следующие данные как фактические параметры (величины, стоящие справа при присвоении):

- Локальные и глобальные переменные
- Переменные в DB (абсолютная или символьная адресация)
- Переменные в локальном экземпляре (абсолютная или символьная адресация)
- Константа NIL
  Вы определяете отсутствие объекта.
- Тип данных ANY
- Таймер, счетчик и блоки
  Вы определяете идентификатор (например, T1, C20 или FB6).

Ограничения

- Тип ANY разрешен для формальных входных и вход/выходных параметров FB и FC, а также для выходных параметров FC. Константы как фактические параметры не разрешены (исключая константу NIL).
- Нельзя, при вызове FB или FC, присваивать формальным параметрам типа ANY временные переменные этого типа. Временные переменные теряют свою достоверность при передаче.
- Переменные типа ANY не должны использоваться как компоненты структуры или как элемент массива.

6.6.1 Пример типа данных ANY

```
VAR_INPUT
   iANY : ANY;
END_VAR

VAR_TEMP
   pANY : ANY;
END_VAR


CASE ii OF
1:
  pANY := MW4;                  // pANY содержит адрес MW4

3..5:
  pANY:= aINT[ii];   //pANY содержит адрес  ii-того
                // элемента массива aINT;
100:
  pANY:= iANY;            //pANY содержит величину на входе iANY
  variable
ELSE
  pANY := NIL;                  // pANY содержит указатель NIL
 END_CASE;

SFCxxx(IN := pANY);
```

7 Объявление локальных переменных и параметров

7.1 Локальные переменные и параметры блока

Категории переменных

В этой таблице показаны категории локальных переменных:

Переменная	Объяснение
Статические переменные	Статические переменные – это локальные переменные, величина которых сохраняется от одного вызова к другому (память блока). Они используются для хранения переменных функциональных блоков и находятся в экземплярных блоках данных.
Временные переменные	Временные переменные - локальные переменные логического блока. Они не занимают статическую область памяти, поскольку хранятся в стеке CPU. Их величина сохраняется только во время работы блока. Временные переменные не доступны вне блока, в котором они объявлены.

Категории параметров блоков

Параметры блока - это метки-заменители, которым при вызове блока получают определенные значения. Метка-заменитель в блоке называется формальным параметром, а величины, присваиваемые им при вызове блока, называются фактическими параметрами. Формальные параметры блока могут рассматриваться как локальные переменные.

Параметры блока разделяются на категории, показанные ниже:

Параметр блока	Объяснение
Входные параметры	Входные параметры при вызове блока принимают текущие величины входов. Они доступны только для чтения.
Выходные параметры	Выходные параметры передают текущие величины на выходы вызываемого блока. Данные могут переписываться и читаться.
Вход/выходные параметры	Вход/выходные параметры принимают текущие величины входа, когда блок вызван. После обработки величин, они получают результат и возвращают его в вызывающий блок.

Флаги (флаг OK)

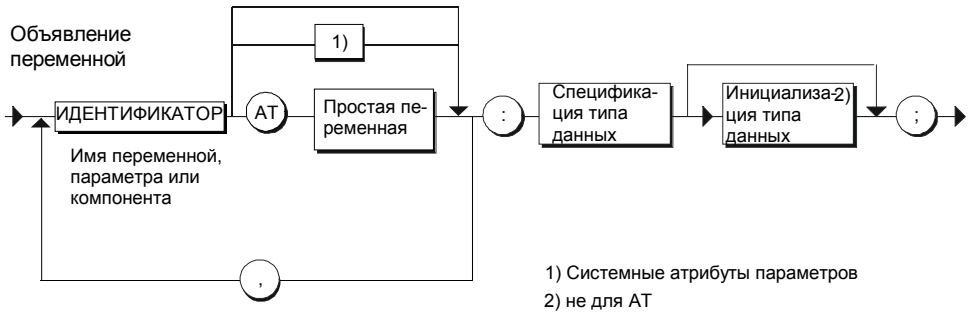
В SCL компилятор обеспечивает флаг, который используется для обнаружения ошибок, возникающих при выполнении программы CPU. Это локальная переменная типа BOOL со встроенные именем “OK".

7.2 Общий синтаксис переменной или объявления параметра

Переменные и параметры блока должны быть индивидуально объявлены перед тем, как они будут использованы в логических блоках или блоках данных. Объявление определяет, что идентификатор использован как параметр блока или как переменная и назначает ему тип данных.

Переменная или параметр состоит из идентификатора (имя для использования) и типа данных. Основной формат показан в синтаксической диаграмме.

Синтаксис переменной или объявления параметра



Примеры

```
VALUE1    :       REAL;
if there are several variables of the same type:

VALUE2, VALUE3,VALUE4,....: INT;
ARR       :       ARRAY[1..100, 1..10] OF REAL;
SET       :       STRUCT
      MEASARR : ARRAY[1..20] OF REAL;
      SWITCH : BOOL;
      END_STRUCT
```

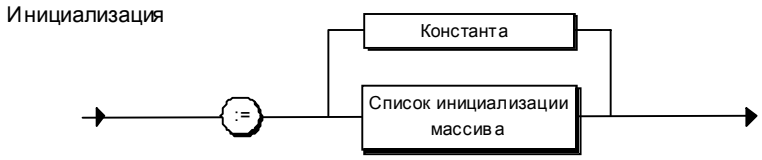
Замечание

Если Вы хотите использовать зарезервированные слова как идентификаторы, то им должен предшествовать символ "#" (например, #FOR).

7.3 Инициализация

При объявлении в FB статических переменных, а также параметров входа и выхода, им могут быть присвоены начальные значения. Параметрам входа/выхода могут также присваиваться начальные значения, но только для элементарных типов данных. Для простых переменных, начальное значение присваивается знаком (:=) после спецификации типа данных.

Синтаксис



Пример

```
VALUE      :REAL   := 20.25;
```

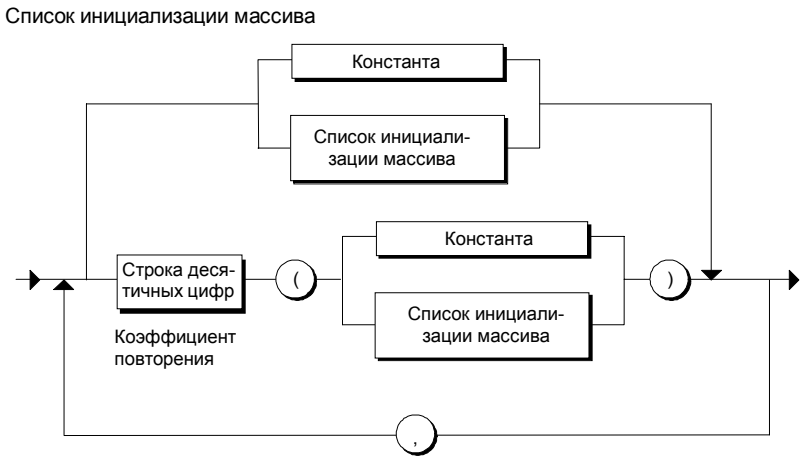
Замечание

Инициализация для списка переменных (A1, A2, A3,...: INT:=...) невозможна. В этом случае переменные должны инициализироваться индивидуально.

Инициализация массива

Инициализируя массив, Вы можете определить величину каждого компонента, разделив их запятыми, или, определив коэффициент повторения (целое число), Вы можете инициализировать несколько компонентов одинаковым значением.

Синтаксис инициализации массива



Примеры

```
VAR
// инициализация статических переменных:
  INDEX1 : INT := 3 ;
//инициализация массива:
  CONTROLLER1 : ARRAY [1..2, 1..2] OF INT := -54, 736, -83,
   77;
  CONTROLLER2 : ARRAY[1..10] OF REAL := 10(2.5);
//инициализация структуры:
   GENERATOR:      STRUCT
     DAT1 :   REAL                          := 100.5;
     A1 :       INT                                 := 10 ;
     A2 :       STRING[6]                           := 'FACTOR';
     A3 :       ARRAY[1..12] OF REAL := 0.0, 10(100.0), 1.0;
     END_STRUCT ;
END_VAR
```

7.4 Объявление многовариантного представления переменных

Чтобы иметь доступ к одной области памяти, как к переменным различных типов, Вы можете определить альтернативное представления переменной, используя ключевое слово "AT". Варианты представления действуют только в блоке; они не включаются в интерфейс. Представление может использоваться подобно другим переменным блока. Оно наследует все свойства переменной, на которую ссылается; только тип данных у него новый.

Пример

Следующий пример показывает несколько представлений одного параметра входа:

```
VAR_INPUT
    Buffer : ARRAY[0..255] OF BYTE;
    Frame1 AT Buffer : UDT100 ;
    Frame2 AT Buffer : UDT200 ;
END_VAR
```

Вызывающий блок обеспечивает параметр буфера, он не видит имена Frame1 и Frame2. Вызывающий блок имеет три пути интерпретации данных, а именно, массив с именем Buffer, или Frame1 и Frame2, с другими структурами (соответственно UDT100 и UDT200).

Правила

- Объявление представления переменной должна быть сделана за объявлением переменной, на которую он указывает в том же подразделе декларации.
- Инициализация невозможна.
- Тип данных представления должен быть совместим по размеру с типом данных переменной. Переменная определяет размер области памяти. Требования к памяти у представления могут быть равными этой области или быть меньше. Существуют следующие правила для комбинирования типов данных:

		Тип данных представления	**Тип данных переменной**		
			Элементарный	**Слож-ный**	**ANY/POINTER**
FB	Объявления представлений VAR, VAR_TEMP, VAR_IN или VAR_OUT	**Элементарный** **Сложный** **ANY/POINTER**	x x	x x x (1)	x (1)
	Объявления представлений VAR_IN_OUT	**Элементарный** **Сложный** **ANY/POINTER**	x	x	
FC	Объявления представлений VAR или VAR_TEMP	**Элементарный** **Сложный** **ANY/POINTER**	X X	x x x	x
	Объявления представлений VAR_IN, VAR_OUT или VAR_IN_OUT	**Элементарный** **Сложный** **ANY/POINTER**	X	x	

(1) ANY не разрешен в VAR_OUT.

Элементарный = BOOL, BYTE, WORD, DWORD, INT, DINT, DATE, TIME, S5TIME, CHAR
Сложный = ARRAY, STRUCT, DATE_AND_TIME, STRING

7.5 Использование мультиэкземпляров

Возможно, что Вам понадобится большее число экземплярных блоков данных, чем позволяют характеристики S7 используемого Вами CPU. Если существующие функциональные блоки вызываются в FB пользовательской программы (вложенный вызов FB), то Вы можете многократно вызвать функциональные блоки без увеличения числа экземпляров блоков данных.

Выполните следующие действия:

- Включите FB, которые Вы хотите вызвать, как статические переменные в объявления переменных вызывающего функционального блока.
- В этом функциональном блоке программируйте вызовы следующих по иерархии FB, используя созданные статические переменные, без указания соответствующего экземпляра блока данных.
- Это концентрирует все экземплярные данные в одном блоке, позволяя Вам более эффективно использовать доступное число блоков данных.

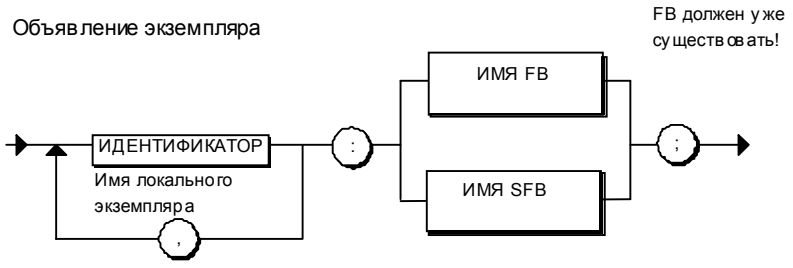
7.6 Объявление экземпляра

Для функциональных блоков Вы можете объявить переменные типа FB или SFB в разделе декларации статических переменных (VAR; END_VAR) дополнительно к переменным элементарного, сложного и определенного пользователем типа данных. Такие переменные называются локальными экземплярами FB или SFB.

Локальные экземплярные данные хранятся в экземплярном блоке данных вызывающего FB. Инициализация экземпляров невозможна.

Блоки, вызываемые как локальные экземпляры, не могут иметь нулевой размер. В таких блоках должна объявляться, по крайней мере, одна статическая переменная или параметр.

Синтаксис



Пример

```
Supply1     : FB10;
Supply2,Supply3,Supply4   : FB100;
Motor1      : Motor ;
```

где Motor –идентификатор FB, введенный в таблице символов.

7.7 Флаг (флаг OK)

Флаг OK информирует Вас о правильной или неправильной работе блока. Это локальная переменная типа BOOL с зарезервированным именем “OK".

В начале программы флагу OK присвоено значение TRUE. Он запрашивается из любой точки блока или устанавливается как TRUE / FALSE, используя операторы присвоения SCL. Если при работе оператора произойдет ошибка (например, деление на нуль), флаг OK автоматически устанавливается в FALSE. Когда блок будет завершен, значение флага OK сохранится в выходном параметре ENO и может использоваться в вызывающем блоке.

Декларация

Флаг OK - системная переменная. Объявлять ее необязательно. Однако, если Вы хотите использовать флаг OK в пользовательской программе, перед компиляцией необходимо установить опцию "Set OK flag (Установить флаг OK) ".

Пример

```
// проверьте флаг OK на TRUE
// для проверки
// правильности действия.
OK:= TRUE;
Division := 1 / IN;
IF OK THEN
     // деление правильно.

     // :
     // :

ELSE         // деление неправильно.

     // :

END_IF;
```

7.8 Подразделы декларации

7.8.1 Обзор подразделов

Каждая категория локальных переменных или параметров имеет свой подраздел декларации, отличающийся парой ключевых слов. Каждый подраздел содержит объявления, которые разрешены для этого подраздела. Подразделы могут располагаться в любом порядке.

В этой таблице показано, какие переменные или типы параметров Вы можете объявить в логических блоках:

Данные	Синтаксис	FB	FC	OB
Переменные: Статические переменные	VAR . . . END_VAR	X	X *)	
Временные переменные	VAR_TEMP . . . END_VAR	X	X	X
Параметры блока: Параметры входа	VAR_INPUT . . . END_VAR	X	X	
Параметры выхода	VAR_OUTPUT . . . END_VAR	X	X	
Параметры входа/выхода	VAR_IN_OUT . . . END_VAR	X	X	

*) Хотя декларация переменных между ключевыми словами VAR и END_VAR в функциях разрешена, при компиляции исходного файла они создаются во временной области.

7.8.2 Статические переменные

Статические переменные – это локальные переменные, значения которых сохраняются от одного вызова функционального блока к другому. Они используются для хранения переменных FB и находятся в соответствующем экземпляре блока данных.

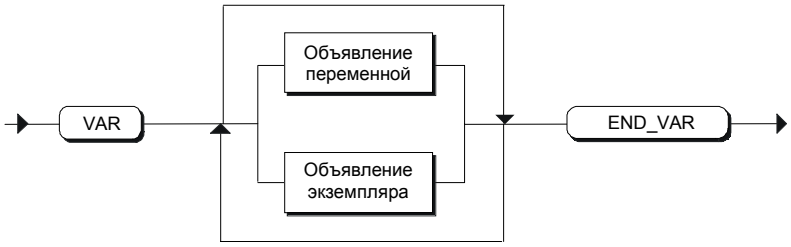
Синтаксис

Статические переменные объявляются в разделе VAR / END_VAR. Этот раздел является частью раздела деклараций FB. После компиляции этот подраздел и подразделы для параметров блока создают структуру назначаемого экземплярного блока данных.

В этом подразделе можно:

- Создавать переменные, назначать типы данных и инициализировать переменные.
- Объявить как статическую переменную FB, вызываемый из данного FB, если Вы хотите вызвать следующий по иерархии вызовов FB как локальный экземпляр.

Раздел статических переменных



Пример

```
VAR
RUN             :INT;
MEASARR         :ARRAY [1..10] OF REAL;
SWITCH          :BOOL;
MOTOR_1,MOTOR_2  :FB100;        //Объявление локального экземпляра
END_VAR
```

Доступ

Переменные доступны в разделе операторов:

- **Доступ в пределах блока**: Доступ к переменной Вы имеете в разделе операторов функционального блока, в котором эта переменная объявлена. Подробно это объяснено в разделе "Назначение величин".

- **Внешний доступ, использующий экземпляр DB:** Используя индексную запись, например, *DBx.variable,* Вы получаете доступ к статической переменной из других программных блоков

7.8.3 Временные переменные

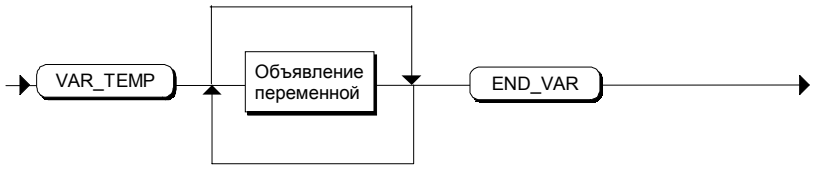
Временные переменные принадлежат только логическому блоку и не занимают статическую область памяти. Они находятся в локальном стеке CPU. Их значение сохраняется только во время работы блока. Временные переменные доступны только в том блоке, в котором они объявлены. Когда OB, FB или FC начинают свою работу, значение временных переменных не определено. Инициализация невозможна.

Вы можете объявить данные как временные, если требуется записать текущие результаты, используемые только пока работают блоки OB, FB или FC.

Синтаксис

Временные переменные объявляются в разделе VAR_TEMP / END_VAR. Этот подраздел – часть FB, FC или OB. Он используется для объявления имен переменных и назначения им типов данных.

Подраздел временных переменных



Инициализация невозможна

Пример

```
VAR_TEMP
  BUFFER 1        : ARRAY [1..10] OF INT ;
  AUX1, AUX2      : REAL ;
END_VAR
```

Доступ

Переменные доступны в разделе операторов логического блока, в котором они объявлены (внутренний доступ). Посмотрите раздел «Назначение переменных».

7.8.4 Параметры блока

Параметры – метки-заменители, которые доступны только при вызове блока. Метки-заменители объявляют в блоке как формальные параметры, величинам которых при вызове блока назначаются фактические параметры. Следовательно, параметры обеспечивают механизм обмена информацией между блоками.

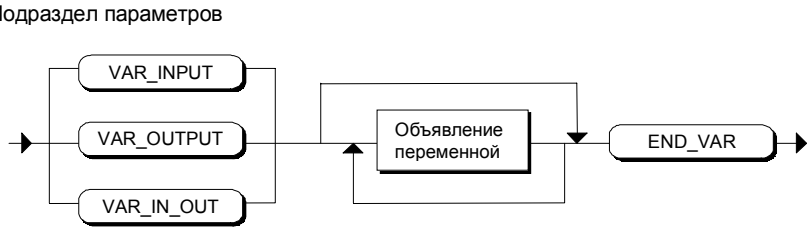
Типы параметров блоков

- Входным формальным параметрам назначаются фактические входные параметры (поток данных в блок).
- Выходные формальные параметры используются для передачи выходных данных (данные из блока)
- Формальные параметры входа/выхода имеют функции и входа, и выхода.

Синтаксис

Объявление формальных параметров осуществляется в одном их трех разделов функционального блока или функции, согласуясь с типом параметра. При объявлении Вы определяете имя параметра и его тип данных. Инициализация возможна только в блоке FB для параметров входа и выхода.

Для объявления формальных параметров, Вы можете использовать не только элементарные, комплексные и определенные пользователем, но и параметрические типы данных.



Инициализация возможна только для VAR_INPUT и VAR_OUTPUT

Пример

```
VAR_INPUT                    // параметры входа
   MY_DB  : BLOCK_DB ;
   CONTROLLER    : DWORD ;
   TIMEOFDAY     : TIME_OF_DAY ;
END_VAR

VAR_OUTPUT                   // параметры выхода
   SETPOINTS: ARRAY [1..10] of INT ;
END_VAR

VAR_IN_OUT                   // параметры входа_выхода
   SETTING : INT ;
END_VAR
```

Доступ

Параметры блоков доступны в разделе операторов логического блока:

- **Внутренний доступ**: Доступ в разделе операторов блока, в котором объявлен параметр. Это объяснено в разделах «Назначение величин» и «Выражения, операции и адрес».
- **Внешний доступ, использующий экземпляр блока данных**: Используя экземпляр DB, Вы имеете доступ к параметрам функционального блока из любого места программы.

8 Объявление констант и меток

8.1 Константы

Константы – это данные, которые имеют постоянную величину и не могут изменяться в процессе выполнения программы.

В SCL. Используются следующие типы констант

- Битовые константы
- Числовые константы
  - Целочисленные константы
  - Вещественные константы
- Символьные константы
  - Буквенные константы
  - Строковые константы
- Константы времени
  - Константы даты
  - Константы интервала времни
  - Константы времени дня
  - Костанты даты и времени

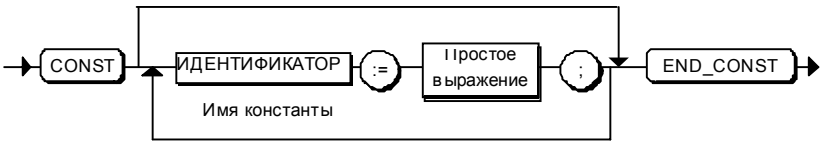
8.1.1 Объявление символьных имен для констант

Константы не обязательно описывать отдельно. Однако, не смотря на это, существует возможность присвоения константам символьных имен в разделе описания переменных.

Вы можете задать символьные имена констант, используя оператор CONST в разделе деклараций вашего логического блока. Это рекомендуется делать для всех констант, содержащихся в блоке. В таком виде блок легче читать и редактировать в случае, если Вы хотите изменить величину константы.

Синтаксис

Подраздел констант



В простом выражении могут содержаться только следующие семь простых математических действий (*, /, +, -, **, DIV, MOD).

Пример

```
CONST
  Number          := 10 ;
  TIMEOFDAY1      := TIME#1D_1H_10M_22S_2MS ;
  NAME            := 'SIEMENS' ;
  NUMBER2         := 2 * 5 + 10 * 4 ;
  NUMBER3         := 3 + NUMBER2 ;
END_CONST
```

8.1.2 Типы данных для констант

Сопоставление типа данных константе отличается от метода, применяемого в STL:

Константе сопоставляется ее тип с учетом арифметической или логической операции, в которой она используется, например:

```
Int1:=Int2 + 12345          //"12345" сопоставлен тип INT
Real1:=Real2 + 12345        //"12345" сопоставлен тип REAL
```

Константе сопоставляется тип данных с наименьшим диапазоном величин, который возможен при описании константы без потери информации. В нашем примере константа "12345" не всегда представляется в виде INT как в STL, а так же может быть представлена классом типов данных ANY_NUM; таким образом, в зависимости от использования, константе могут быть сопоставлены типы INT, DINT, или REAL.

Константы с определенным типом

Используя запись констант с определенным типом, Вы так же можете конкретно назначить константе один из следующих типов данных.

Примеры:

Тип данных	**Запись с определенным типом**	
BOOL	BOOL#1 Bool#false	bool#0 BOOL#TRUE
BYTE	BYTE#0 Byte#'ä'	B#2#101 b#16#f
WORD	WORD#32768 W#2#1001_0100	word#16#f WORD#8#177777
DWORD	DWORD#16#f000_0000 DW#2#1111_0000_1111_0000	dword#32768 DWord#8#37777777777
INT	INT#16#3f_ff Int#2#1111_0000	int#-32768 inT#8#77777
DINT	DINT#16#3fff_ffff DInt#2#1111_0000	dint#-1000_0000 dinT#8#17777777777
REAL	REAL#1 real#2e4	real#1.5 real#3.1
CHAR	CHAR#A	CHAR#49

8.1.3 Запись констант

Особенности записи или формата константы зависят от ее типа и формата данных. Тип и величина константы определяется непосредственно записью и не нуждается в объявлении. Примеры:

15	ЧИСЛО 15	целая константа в десятеричном формате
2#1111	ЧИСЛО 15	целая константа в двоичном формате
16#F	ЧИСЛО 15	целая константа в шестнадцатеричном формате

Обзор возможных записей

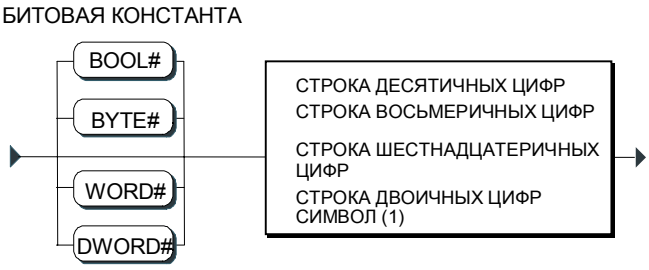
Тип данных	**Описание**	**Пример на SCL**	**Пример на STL (различия)**
BOOL	1 Бит	FALSE TRUE BOOL#0 BOOL#1 BOOL#FALSE BOOL#TRUE	
BYTE	8-битное шестнадцатеричное число	B#16#00 B#16#FF BYTE#0 B#2#101 Byte#'ä' b#16#f	
CHAR	8-бит (1 символ ASCII)	'A' CHAR#49	
STRING	До 254 ASCII символов	'Address'	
WORD	16-битное шестнадцатеричное число 16-битное восьмеричное число 16-битное двоичное число	W#16#0000 W#16#FFFF word#16#f WORD#8#177777 8#177777 W#2#1001_0100 WORD#32768	
DWORD	32-битное шестнад-цатеричное число 32-битное двоичное число 32-битное двоичное число	DW#16#0000_0000 DW#16#FFFF_FFFF Dword#8#37777777777 8#37777777777 DW#2#1111_0000_1111_0000 dword#32768	

Тип данных	**Описание**	**Пример на SCL**	**Пример на STL (различия)**
INT	16-битное число с фиксированной запятой	-32768 +32767 INT#16#3f_ff int#-32768 Int#2#1111_0000 inT#8#77777	
DINT	32-битное число с фиксированной запятой	-2147483648 +2147483647 DINT#16#3fff_ffff dint#-1000_0000 Dint#2#1111_0000 dinT#8#17777777777	L#-2147483648 L#+2147483647
REAL	32-битное число с плавающей запятой	Десятичный формат 123.4567 REAL#1 real#1.5 Экспоненциальный формат real#2e4 +1.234567E+02	
S5TIME	16-битная величина времени в формате SIMATIC	T#0ms TIME#2h46m30s T#0.0s TIME#24.855134d	S5T#0ms S5TIME#2h46m30s
TIME	32-битная величина времени в формате IEC	T#-24d20h31m23s647ms TIME#24d20h31m23s647ms T#0.0s TIME#24.855134d	
DATE	16-битная дата	D#1990-01-01 DATE#2168-12-31	
TIME_OF_DAY	32-битное время дня	TOD#00:00:00 TIME_OF_DAY#23:59:59.999	
DATE_AND_ TIME	Дата и время	DT#95-01-01-12:12:12.2	

8.1.3.1 Битовые константы

Битовыми константами могут быть величины длиной 1, 8, 16 или 32 бита. В SCL программе, в зависимости от длины, используется один из типов BOOL, BYTE, WORD или DWORD.

Синтаксис



(1) только типа BYTE

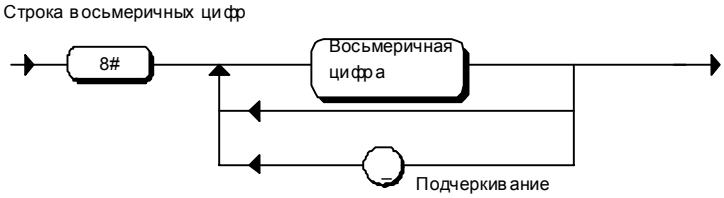
Десятичная строка

Константа, являющаяся десятичным числом, представляет собой строку цифр (при необходимости цифры могут быть разделены подчеркиванием). Подчеркивания используются для того, чтобы улучшить читаемость длинных чисел. Пример правильного описания строки десятичных цифр для констант показан ниже:

```
dword#32768
```

Двоичные, Восьмеричные и Шестнадцатеричные величины

Вы можете определить битовую константу целым числом в системе счисления, отличной от десятичной используя префиксы **2#, 8#** или **16#** перед числом, записанным в выбранной Вами системе счисления. Это показано на рисунке внизу на примере восьмеричной константы:



Пример

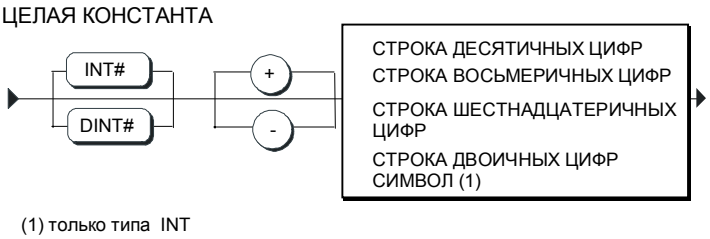
Следующие примеры показывают запись битовых констант:

```
Bool#false
8#177777
DW#16#0000_0000
```

8.1.3.2 Целочисленные константы

Целочисленные константы представляют собой целые величины длиной 16 или 32 бита. В SCL программе, в зависимости от длины, они могут относиться к типам INT или DINT.

Синтаксис



(1) только типа INT

Десятеричная строка

Константа, заданная десятичным числом, представляет собой строку десятичных разрядов (при необходимости разряды могут быть разделены подчеркиванием). Подчеркивания используются для того, чтобы улучшить читаемость длинных чисел. Примеры правильного описания строк десятичных разрядов для констант показаны ниже:

```
1000
1_120_200
666_999_400_311
```

Двоичные, Восьмеричные и Шестнадцатеричные величины

Вы можете определить целую константу в системе исчисления, отличной от десятичной, используя префиксы **2#, 8#** или **16#** перед числом, записанным в выбранной Вами системе исчисления.

Пример

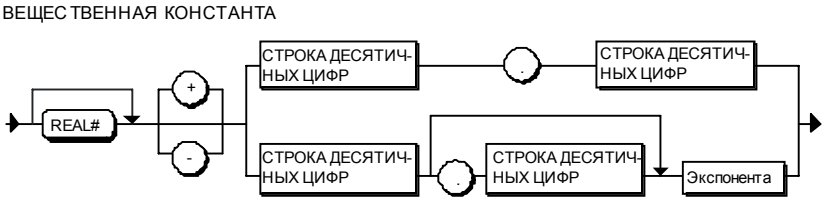
Следующие примеры иллюстрируют запись для целочисленных констант:

```
Value_2:=2#0101;     // Двоичное число, десятичное 5
Value_3:=8#17;               // Восьмеричное число, десятичное 14
Value_4:=16#F;               // 16-теричное число, десятичное 15
Value_5:=INT#16#3f_ff // 16-теричное число, запись,
                // определяющая тип
```

8.1.3.3 Вещественные константы

Вещественные константы это величины с десятичной точкой. Им сопоставляется тип данных REAL.

Синтаксис



Использование знаков плюс и минус не обязательно. Если знак не определен, число считается положительным.

Константа, заданная десятичным числом, представляет собой строку разрядов (при необходимости разряды могут быть разделены подчеркиванием). Подчеркивания используются для того, чтобы улучшить читаемость длинных чисел. Примеры правильного описания строк десятичных цифр для констант показаны ниже:

```
1000
1_120_200
666_999_400_311
```

Экспоненциальная запись

При записи чисел с плавающей запятой можно так же использовать экспоненциальную запись. Порядок обозначается буквой "E" или "e" с целым числом порядков.

В SCL вещественные числа могут быть записаны следующим образом:

```
3.0E+10     3.0E10   3e+10          3E10
0.3E+11     0.3e11   30.0E+9 30e9
```

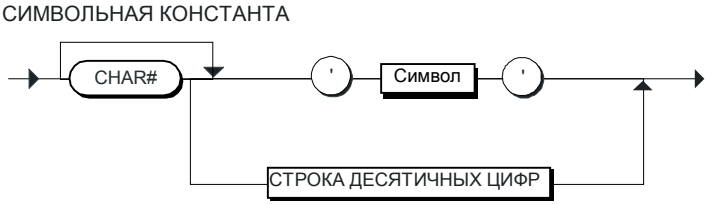
Примеры

```
NUM4:= -3.4 ;
NUM5:= 4e2 ;
NUM6:= real#1.5;
```

8.1.3.4 Символьные константы (Одиночные символы)

Символьная константа представляет собой один символ. Символ заключается в одиночные кавычки ('). Символьные константы не могут использоваться в математических выражениях.

Синтаксис



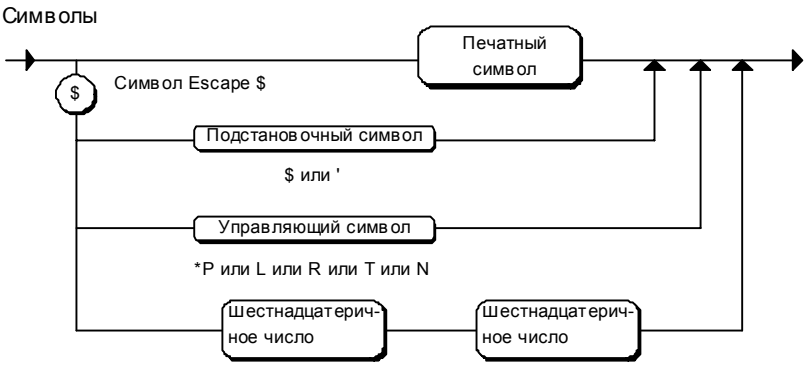
Пример

```
Charac_1 := 'B';
Charac_2 := char#43;
Charac_3 := char#'B';
```

Синтаксис символа

Можно использовать любые символы полного и расширенного набора кодов ASCII. Спецсимволы, такие как одиночные кавычки (') или символ $, можно ввести, используя символ $.

Вы можете так же использовать непечатные символы полного и расширенного набора ASCII. Для их представления используется шестнадцатеричный код символа.



*P = Подача страницы
L = Перевод строки
R = Возврат каретки
T = Табулятор
N = Новая строка

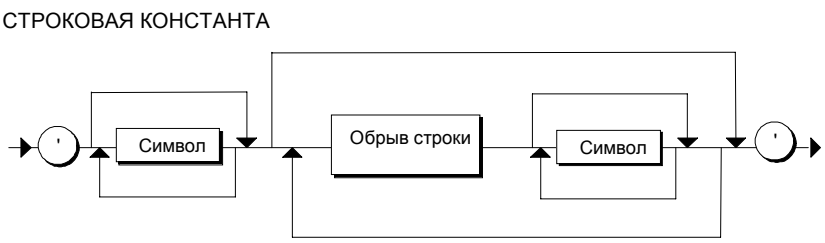
Пример шестнадцатеричных кодов символов

```
CHARACTER       := '$41' ;          //Определяет символ 'A'
Blank           :='$20';  //Определяет символ '_'
```

8.1.3.5 Строковые константы

Строковая константа это строка, состоящая максимум из 254 символов. Символы заключаются в одиночные кавычки. Строковые константы не могут использоваться в математических выражениях.

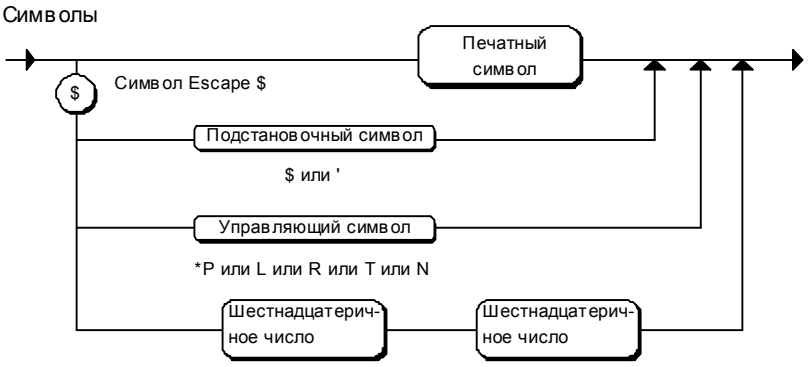
Синтаксис



Синтаксис символа

Можно использовать любые символы основного и расширенного набора ASCII. Спецсимволы, такие как одиночные кавычки (') или символ $, можно ввести, используя символ $.

Вы можете также использовать непечатные символы полного и расширенного набора ASCII. Для их представления используется шестнадцатеричный код символа.



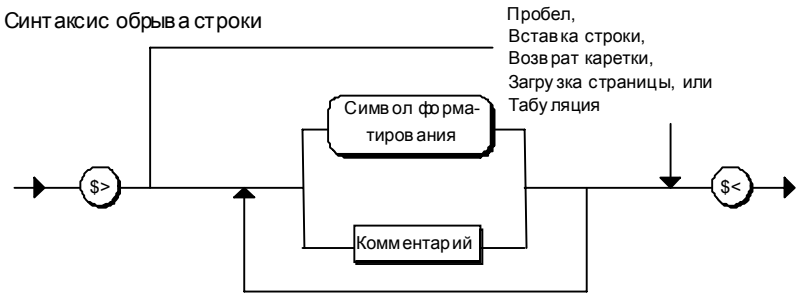
Альтернативное представление в шестнадцатеричном коде

*P = Подача страницы
L = Перевод строки
R = Возврат каретки
T = Табулятор
N = Новая строка

Прерывание строки

Записывая длинную строку в программе, Вы можете прервать ее и вернуться к ней несколько раз.

Строковая константа может располагаться в любой строчке блока SCL программы или в нескольких строчках с использованием специальных символов. Для прерывания строки используется символ $>, а для продолжения символ $<. В промежутках могут содержаться любые комментарии или пропуски длиной до нескольких строк.



Примеры

```
// Строковая константа:
NAME := 'SIEMENS';
//Прерывание строковой константы
MESSAGE1:= 'MOTOR- $>
$< Controller';
// строка в шестнадцатеричных кодах:
MESSAGE1:= '$41$4E' (*character string AN*);
```

8.1.3.6 Константы типа DATE

Дата обозначается префиксами DATE# или D# . Запись даты включает целые числа: год (4 разряда), месяц и день, разделенные тире.

Синтаксис

Пример

```
TIMEVARIABLE1:= DATE#1995-11-11 ;
TIMEVARIABLE2:= D#1995-05-05 ;
```

8.1.3.7 Константы интервала времени

Интервал времени обозначается префиксами TIME# или T#. Интервал времени может быть записан двумя различными способами:

- В десятичном формате
- В смешанном формате

Синтаксис

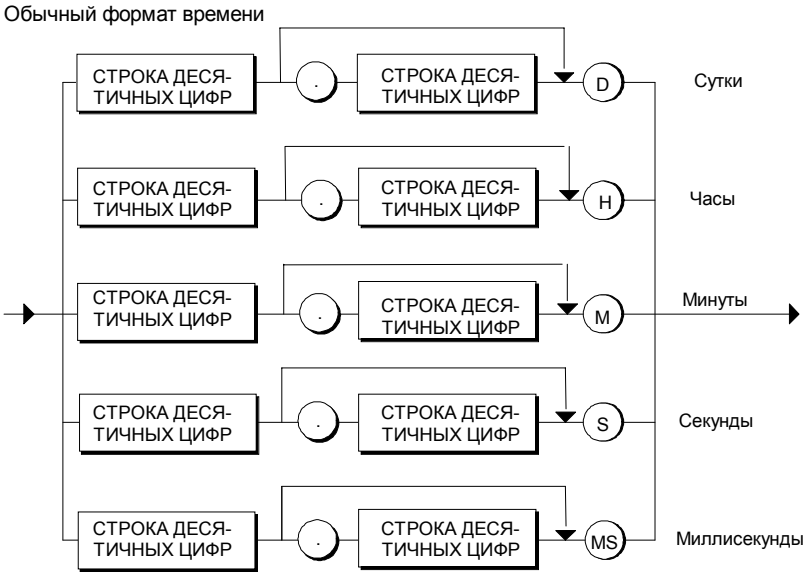
- Каждая единица времени (часы, минуты, и т.д.) должна быть задана однократно.
- Должен соблюдаться порядок перечисления: дни, часы, минуты, секунды, миллисекунды.

Переход от смешанного к десятичному формату возможен только в случае, когда эта единица измерения времени еще не задана.

После вступительных приставок T# или TIME# Вы должны указать хотя бы одну единицу времени.

Десятичный формат

Используйте десятичный формат, если хотите указать компоненты времени, например, часы и минуты, в десятичном виде.

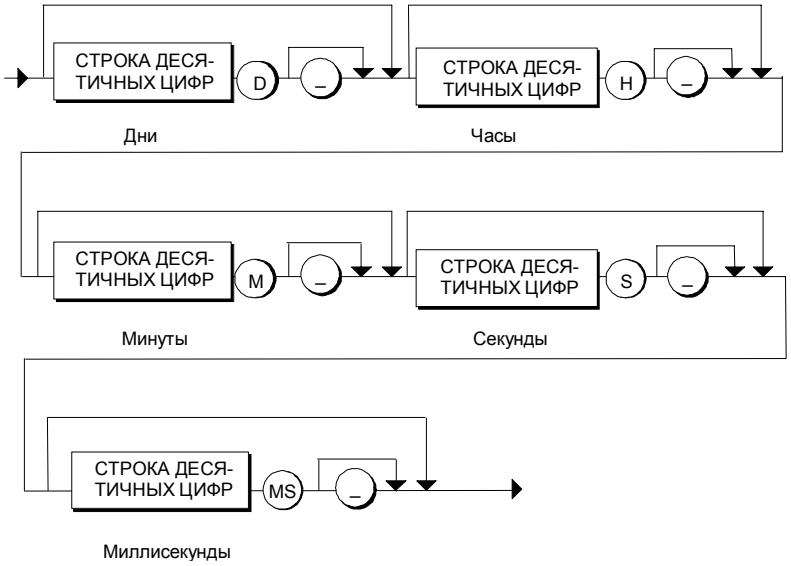


Использование обычного формата времени возможно только для еще не определенных единиц времени.

Смешанный формат

Смешанный формат – это последовательность отдельных компонент времени. Сначала задается день, потом час и. т. д. Записи разделяются подчеркиванием. Вы можете пропускать в последовательности ненужные компоненты. При этом надо указать хотя бы одну единицу времени.

Составной формат времени



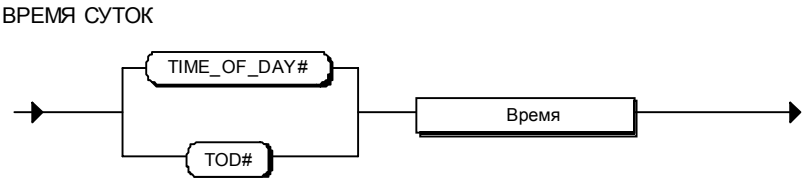
Пример

```
// Десятичный формат
Interval1:= TIME#10.5S ;

// Смешанный формат
Interval2:= T#3D_2S_3MS ;

// Смешанный и десятичный формат
Interval3 := T#2D_2.3s ;
```

8.1.3.8 Константы времени дня

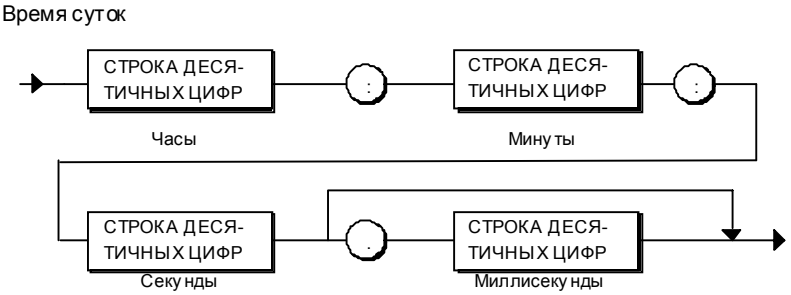
Время дня описывается префиксами TIME_OF_DAY# или TOD#.

Синтаксис



Время дня определяется записью часов, минут и секунд, разделенных двоеточиями. Задание миллисекунд необязательно. Миллисекунды отделяются от других чисел десятичной точкой.

Определение времени дня



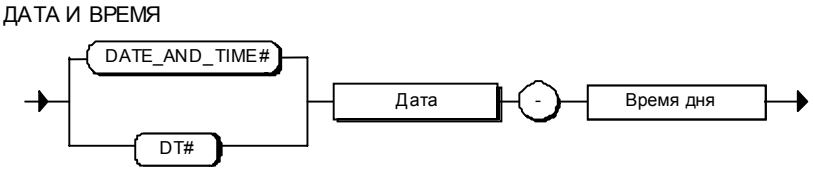
Пример

```
TIMEOFDAY1:= TIME_OF_DAY#12:12:12.2 ;
TIMEOFDAY2:= TOD#11:11:11 ;
```

8.1.3.9 Константы даты и времени

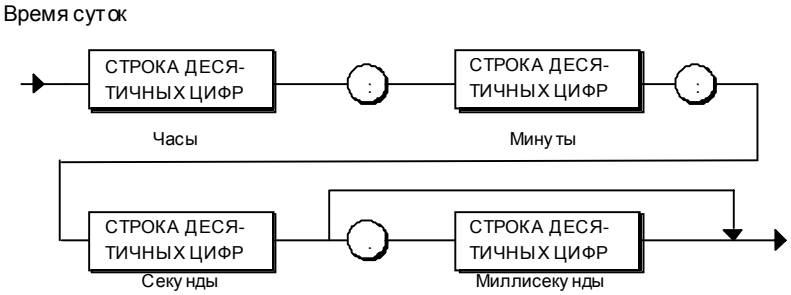
Дата и время описывается префиксами DATE_AND_TIME# или DT#. Эта константа состоит из даты и времени дня.

Синтаксис



Дата

Время дня



Пример

```
TIMEOFDAY1:= DATE_AND_TIME#1995-01-01-12:12:12.2 ;
TIMEOFDAY2:= DT#1995-02-02-11:11:11;
```

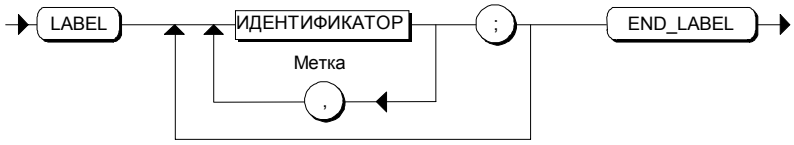
8.2 Объявление меток

8.2.1 Объявление меток

Метки используются для определения места адресации оператора GOTO. Метки объявляются в разделе деклараций логического блока, где им присваиваются символьные имена.

Синтаксис

Подраздел меток



Пример

```
LABEL
   LAB1, LAB2, LAB3;
END_LABEL
```

9 Глобальные данные

9.1 Обзор глобальных данных

В SCL предоставляется доступ к глобальным данным. Существуют следующие два типа глобальных данных:

- **Области памяти CPU**

  Это области системной памяти, например области отображения входов-выходов и меркеры. Размеры доступных областей памяти зависят от используемого CPU.

- **Совместно используемые данные в форме загружаемых блоков**

  Эти области данных находятся в блоках данных. Для того чтобы их использовать, Вы должны сначала создать блоки данных и объявить данные в них. Экземплярные блоки данных основаны на спецификации функциональных блоков и создаются автоматически.

Доступ к совместно используемым данным

Вы можете получить доступ к совместно используемым данным следующим образом:

- **Методом абсолютной адресации:** Используя идентификатор адреса и абсолютный адрес.
- **Методом символической адресации:** Указав символ, ранее определенный в таблице символов**.**
- **Индексированно:** Используя идентификатор адреса и индекс массива.
- **Структурно:** Используя переменные.

Тип доступа	Области памяти CPU	Совместно используемые данные пользователя
Абсолютный	Да	Да
Символический	Да	Да
Индексированный	Да	Да
Структурный	Нет	Да

9.2 Области памяти CPU

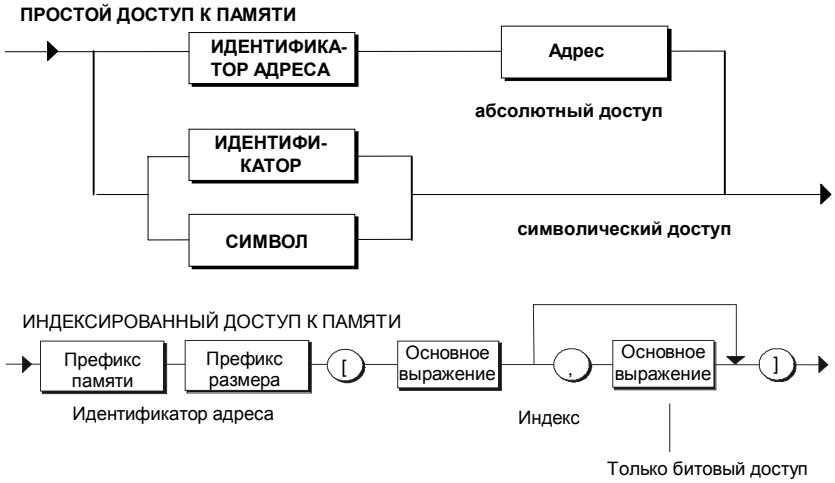
9.2.1 Обзор областей памяти CPU

Области памяти CPU – это области памяти, объявленные во всей системе. По этой причине эти области не нуждаются в объявлении в вашем логическом блоке. Каждый CPU имеет следующие области памяти, с диапазонами адресов, зависящими от типа CPU:

- Области отображения входов и выходов (например, Q1.0)
- Периферийные входы и выходы (например, PQ1.0)
- Меркеры (например, M1.0)
- Таймеры, счетчики (например, C1)

Синтаксис доступа

- В разделе операторов вашего логического блока Вы получаете доступ к области памяти CPU, используя задание величин следующим образом: при помощи простого доступа, когда Вы можете указать абсолютный адрес или символ, или
- Используя индексную адресацию.



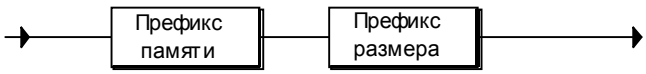
9.2.2 Абсолютный доступ к областям памяти CPU

Для доступа к области памяти CPU с заданием абсолютного адреса используйте, например, абсолютный идентификатор для обозначения переменной соответствующего типа.

```
STATUS_2:= IB10;
   |          \
   |           Абсолютный идентификатор
Переменная с соответствующим
типом
```

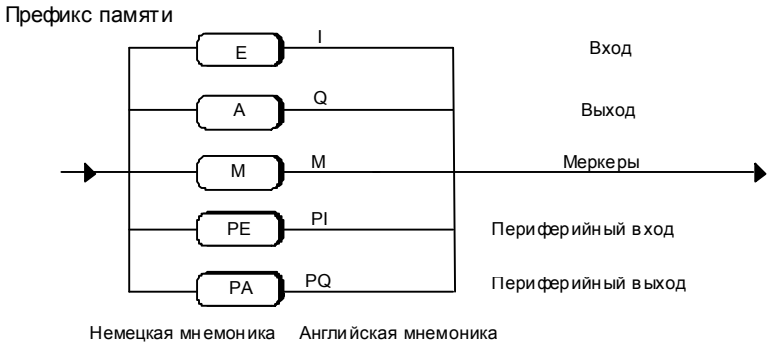
Абсолютный идентификатор указывает на область памяти CPU. Эта область определяется идентификатором адреса (в нашем случае это IB), приведенным перед адресом (в нашем случае 10).

Синтаксис абсолютного идентификатора



Префикс памяти

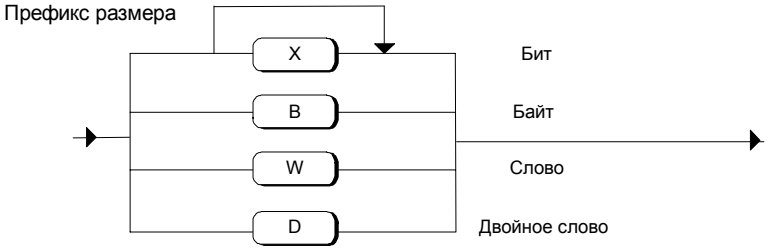
Префикс памяти определяет тип памяти, к которой Вы хотите обратиться.



Идентификатор адреса имеет зарезервированные значения, зависящие от выбранной Вами мнемоники, английской или немецкой.

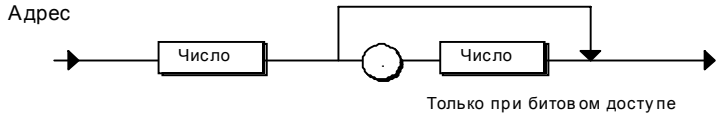
Префикс размера

Префикс размера определяет длину области памяти. Можно, например, прочитать байт или слово. Использование префикса размера не обязательно в случае, если Вы хотите определить один бит.



Адрес

Для задания адреса сначала определяется абсолютный адрес байта, а затем, через точку, адрес бита. Задание адреса бита необязательно.



Примеры

```
STATUSBYTE      :=IB10;
STATUS_3 :=I1.1;
MEASVAL :=IW20;
```

9.2.3 Символический доступ к областям памяти CPU

При символической адресации для определения области памяти CPU, вместо абсолютного идентификатора используется символическое имя.

В таблице символики Вы присваиваете символические имена отдельным адресам в Вашей программе. Для добавления символов в таблицу, Вы можете открыть ее в любое время при помощи команды меню **Options > Symbol Table (Параметры > Таблица символов)**.

В таблице символики для определения типа данных можно использовать любой элементарный тип данных, применение которого не противоречит объявленному размеру элемента данных. Приведенная таблица показывает, как образуется таблица символики.

Symbol	Absolute Address	Data Type	Comments
Motor_contact_1	I1.7	BOOL	Переключатель 1 мотора
Input1	IW 10	INT	Слово статуса

Доступ

Адрес становится доступным путем присвоения величине переменной соответствующего типа вместе с объявленным символическим именем.

Пример

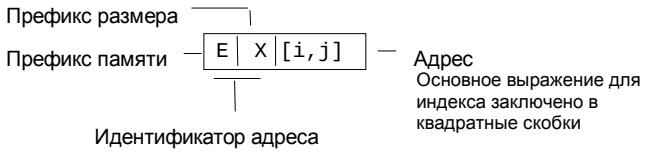
```
MEASVAL_1 := Motor_contact_1;
Status_Motor1 := Input1 ;
```

9.2.4 Индексная адресация к областям памяти CPU

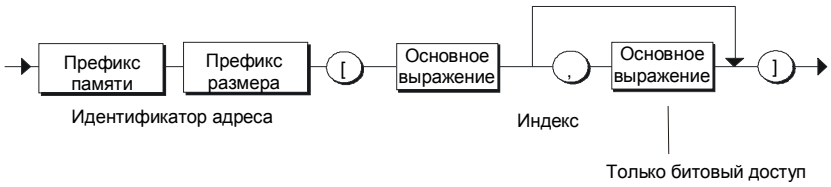
Доступ к областям памяти CPU можно также получить, используя индексную адресацию. Преимущество этого метода перед методом абсолютной адресации заключается в том, что Вы можете задавать адрес динамически, используя различные индексы. Можно, например, использовать в качестве адреса переменную цикла FOR.

Индексированный доступ к областям памяти осуществляется методом похожим на метод абсолютной адресации. Отличие состоит только в особенности определения адреса. Вместо абсолютного адреса задается индекс, который может быть константой, переменной или арифметическим выражением.

При индексированном доступе идентификатор адреса состоит из абсолютного идентификатора (префиксов памяти и размера) и основного выражения для индексирования.



Синтаксис абсолютного идентификатора



Индексирование (основное выражение) должно удовлетворять следующим правилам

- Каждый индекс должен быть арифметическим выражением типа INT.
- При доступе к данным типа BYTE, WORD или DWORD можно использовать только один индекс. Индекс интерпретируется как адрес байта. Область доступа определяется префиксом размера.
- При доступе к данным типа BOOL используются два индекса. Первый индекс определяет адрес байта, а второй положение бита в байте.

Пример

```
MEASVAL_1       :=IW[COUNTER];
OUTLABEL        :=I[BYTENO, BITNO];
```

9.3 Блоки данных

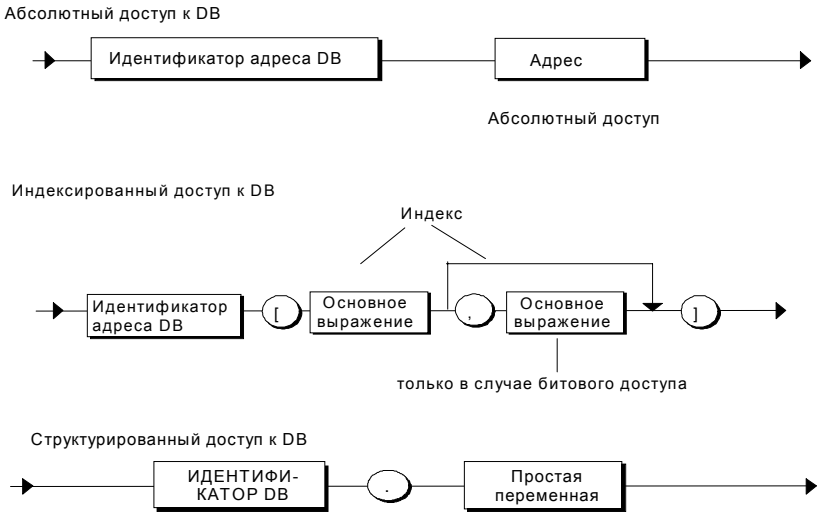
9.3.1 Обзор блоков данных

В блоках данных Вы можете хранить и изменять данные ваших приложений, которые могут относиться ко всей программе или ко всему проекту. Каждый логический блок может считывать или записывать общие пользовательские данные.

Доступ

Доступ к глобальному блоку данных осуществляется следующими способами:

- абсолютно или непосредственно,
- структурно,
- индексированно.



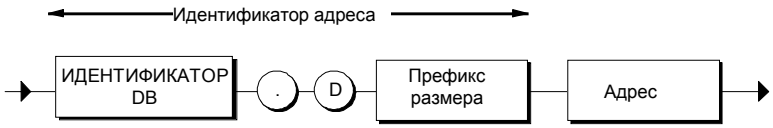
9.3.2 Абсолютный доступ к блокам данных

При программирования с абсолютным доступом к блоку данных, Вы задаете величину переменной таким же образом, как и в случае областей данных CPU. Сначала задается идентификатор DB, сопровождаемый ключевым словом "D", потом префикс размера (например, X – для бита) и, наконец, адрес относительно начала DB (например 13.1).

Синтаксис

Доступ определяется идентификатором DB вместе с префиксом размера и адресом.

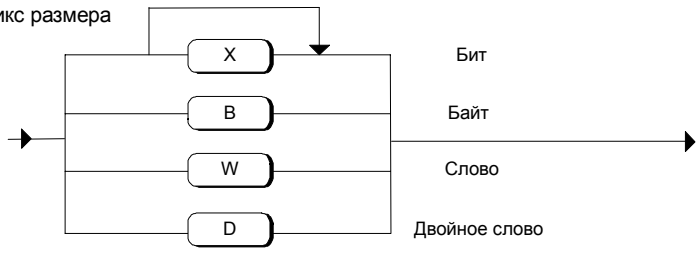
Абсолютный доступ к DB



Префикс размера

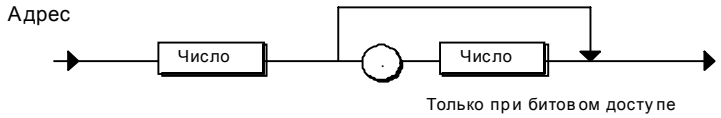
Префикс размера отображает размер области памяти в блоке данных, к которой идет адресация. Можно, например, прочитать байт или слово из DB. Использование префикса размера необязательно в случае, если Вы хотите задать один бит.

Префикс размера



Адрес

Когда Вы задаете адрес, сначала задается абсолютный адрес байта, а затем, через точку, адрес бита (только при побитном доступе).



Пример

Примеры абсолютного доступа к данным приведены ниже. Сам блок данных задается в абсолютных выражениях в первой части и в символических выражениях во второй части.

```
STATUSBYTE      :=DB101.DB10;
STATUS_3        :=DB30.D1.1;
MEASVAL         :=DB25.DW20;

STATUSBYTE      :=Status_data.DB10;
STATUS_3        :="New data".D1.1;
MEASVAL         :=Measdata.DW20;

STATUS_1        :=WORD_TO_BLOCK_DB (INDEX).DW10;
```

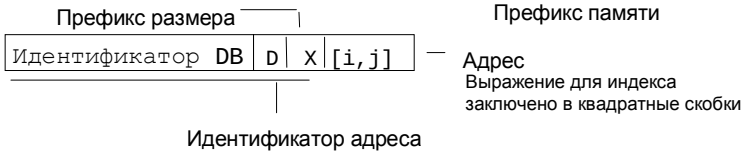
9.3.3 Индексированный доступ к блокам данных

Вы также можете получить доступ к блоку данных с использованием индекса. Преимущество этого метода перед методом абсолютной адресации заключается в том, что он позволяет обращаться к областям, адрес которых определяется после запуска программы. Можно, например, использовать в качестве адреса переменную цикла FOR.

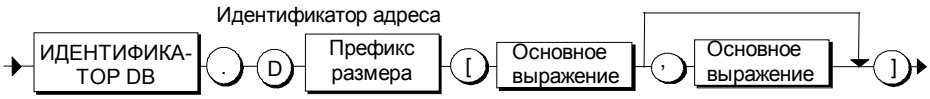
Индексированный доступ к блокам данных похож на абсолютный доступ. Различие состоит только в задании адреса.

Вместо абсолютного адреса задается индекс, который может быть константой, переменной или арифметическим выражением.

Индексированный доступ состоит из идентификатора DB, идентификатора адреса (ключевое слово "D" и префикс размера) и основного выражения для индексирования.



Синтаксис



При использовании индексов должны быть соблюдены следующие правила:

- При доступе к данным типа BYTE, WORD или DWORD можно использовать только один индекс. Индекс интерпретируется как адрес байта. Область доступа определяется префиксом.
- При доступе к данным типа BOOL используются два индекса. Первый индекс определяет адрес байта, а второй положение бита в байте.
- Каждый индекс должен быть арифметическим выражением типа INT (0 - 32767).

Пример

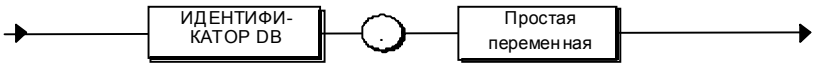
```
STATUS_1:= DB11.DW[COUNTER];
STATUS_2:= DB12.DX[WNO, BITNO];
STATUS_1:= Database1.DW[COUNTER];
STATUS_2:= Database2.DX[WNO, BITNO];
STATUS_1:= WORD_TO_BLOCK_DB(INDEX).DW[COUNTER];
```

9.3.4 Структурированный доступ к блокам данных

При структурированном доступе используется идентификатор переменной, объявленной в блоке данных. Вы можете сопоставить переменную любой переменной того же типа.

Ссылка на переменную в блоке данных определяется именем DB и именем простой переменной, разделенными точкой.

Синтаксис



Простая переменная обозначает переменную, которой сопоставлен элементарный или комплексный тип данных при объявлении DB.

Если при инициализации доступа к блоку данных используется параметр типа BLOCK_DB или результат преобразования функции WORD_TO_BLOCK_DB, то возможны только индексированный и абсолютный доступы, а структурированный доступ запрещен.

Пример

В разделе деклараций FB10:

```
VAR
Result:    STRUCT RES1 : INT;
RES2 : WORD;
END_STRUCT
END_VAR
```

Пользовательский тип данных UDT1

```
TYPE UDT1   STRUCT RES1 : INT;
RES2 : WORD;
END_STRUCT
```

DB20 с пользовательским типом данных:

```
DB20
UDT1
BEGIN ...
```

DB30 без пользовательского типа данных:

```
DB30      STRUCT RES1 : INT;
RES2 : WORD;
END_STRUCT
BEGIN ...
```

FB со следующими доступами:

```
..
FB10.DB10();
RESWORD_A   :=   DB10.Result.RES2;
RESWORD_B   :=   DB20.RES2;
RESWORD_C   :=   DB30.RES2;
```

10 Выражения, операции и адреса

10.1 Обзор выражений, операций и адресов

Выражение обозначает величину, которая вычисляется при компиляции или в процессе выполнения и состоит из адресов (например, констант, переменных или вызовов функций) и операций (например, *, /, + или -).

Типы данных адресов и операций определяют тип выражения. В SCL возможны следующие типы выражений:

- Арифметические выражения
- Неравенства
- Логические выражения

Выражение рассчитывается по следующим правилам:

- Операции выполняются в порядке старшинства
- По порядку слева направо,
- Операции в скобках выполняются в первую очередь.

С результатами выражений можно производить следующие действия:

- Присвоить значение переменной.
- Использовать как условие для условного оператора.
- Использовать как параметр при вызове функции или функционального блока.

10.2 Операции

Выражение состоит из адресов и операций. Большинство операций в SCL действуют с двумя адресами и называются, поэтому, *бинарными* операторами. Остальные операции действуют только с одним адресом и поэтому называются унарными операторами.

Бинарные операции записываются между адресами (например, A + B). Унарные операции всегда записываются непосредственно перед адресом (например, -B).

Приоритет операций показан в таблице ниже, операции со приоритетом `1' выполняются в первую очередь.

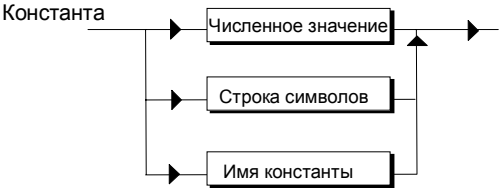
Класс	Операция	Символ	Приоритет
Операция присвоения:	Присвоение	: =	11
Арифметические операции:	Возведение в степень	**	2
	Унарные операции		
	Унарный плюс	+	3
	Унарный минус	-	3
	Основные арифметические операции		
	Умножение	*	4 4
	Деление	/	4
	Модуль	MOD	4
	Деление нацело	DIV	5
	Сложение	+	5
	Вычитание	-	
Операции сравнения:	Меньше	<	6
	Больше	>	6
	Меньше или равно	<=	6
	Больше или равно	>=	6
	равно	=	7
	не равно	<>	7
Логические операции:	Отрицание	NOT	3
	Основные логические операции		
	И	AND or &	8
	Исключающее или	XOR	9
	Или	OR	10
Скобки:	Скобки	( )	1

10.3 Адреса

Адреса – это объекты, из которых может быть составлено выражение. В адресах могут содержаться следующие элементы:

Константы

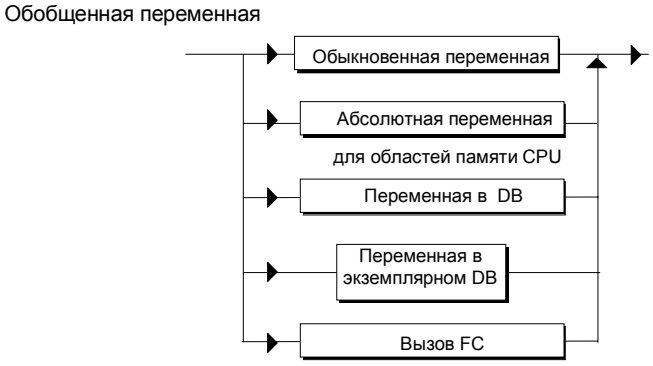
Константа может быть численным значением, символьным именем или строкой символов.



Далее представлены примеры правильных констант:
4_711
4711
30.0
'CHARACTER'
FACTOR

Обобщенные переменные

Обобщенные переменные – это общее название для последовательности переменных, более детально описанных в разделе “Присвоение значений”.



Некоторые примеры правильных переменных:

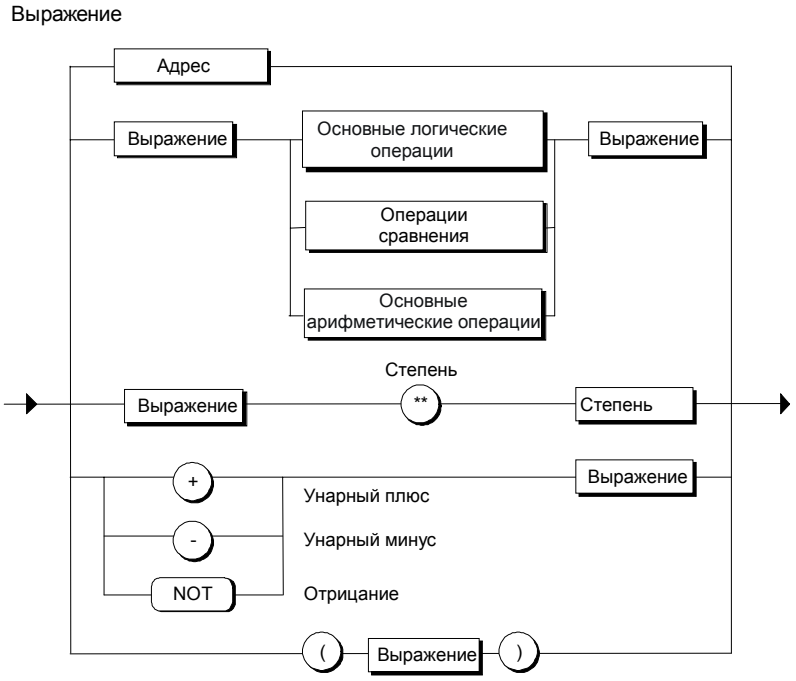
SETPOINT	простая переменная
IW10	абсолютная переменная
I100.5	абсолютная переменная
DB100.DW [INDEX]	переменная в DB
MOTOR.SPEED	переменная в локальном экземпляре
SQR (20)	стандартная функция
FC192 (SETPOINT)	вызов функции

Примечание

В случае вызова функции, результат ее вычисления – возвращаемая величина, вставляется в выражение на место имени функции. Поэтому функция VOID, которая не возвращает значения, не может определять адрес в выражении.

10.4 Синтаксис выражения

Синтаксис



Результат выражения

С результатами выражений можно производить следующие действия:

- Присвоить значение переменной.
- Использовать как условие для условного оператора.
- Использовать как параметр при вызове функции или функционального блока.

Порядок вычисления

Порядок вычисления значения выражения определяется следующими правилами:

- С учетом приоритета операций
- По порядку слева направо
- С учетом скобок (если операции имеют одинаковый приоритет).

Правила

Вычисление значения выражения подчиняется следующим правилам:

- Адрес, находящийся между двумя операциями, всегда соответствует операции с большим приоритетом.
- Операции выполняются в иерархическом порядке.
- Операции с одинаковым приоритетом выполняются в порядке слева направо.
- Запись знака минус перед идентификатором эквивалентна умножению его на -1.
- Арифметические операции не могут следовать одна за другой. Запись a * - b неправильна, в этом случае требуется a*(-b).
- Скобки используются для изменения порядка вычисления по приоритету; другими словами скобки имеют наивысший приоритет.
- Выражение в скобках считается одним адресом и всегда вычисляется в первую очередь.
- Количество левых скобок должно быть равно количеству правых скобок.
- В арифметических операциях не могут применяться символы или логические данные. Выражения типа 'A' + 'B' и (n <= 0) + (m > 0) неправильны.

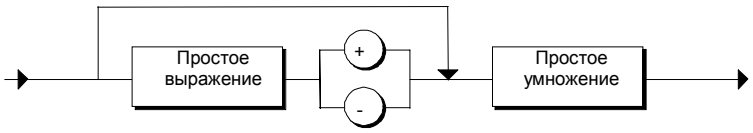
Примеры выражений

```
IB10                    // адрес
A1 AND (A2)             // логическое выражение
(A3) < (A4)     // выражение сравнения
3+3*4/2                 // арифметическое выражение
```

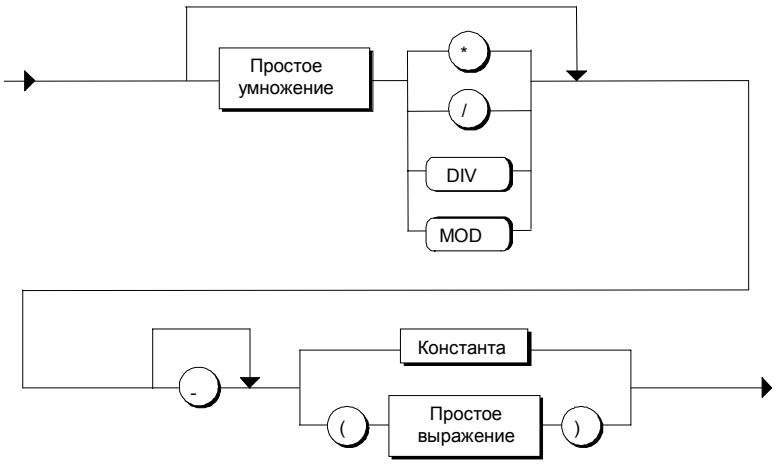
10.5 Простые выражения

В SCL простые выражения это простые арифметические выражения. Вы можете попарно умножать или делить величины констант и добавлять или исключать эти пары.

Синтаксис простого выражения



Синтаксис простого умножения



Пример

```
SIMP_EXPRESSION= A * B + D / C - 3 * VALUE1;
```

10.6 Арифметические выражения

Арифметическое выражение – это выражение, составленное при помощи арифметических операций. В выражении могут использоваться различные числовые типы данных.

Следующая таблица иллюстрирует все возможные *операции* и показывает тип, к которому будет относиться результат, в зависимости от адреса. Были использованы следующие сокращения:

ANY_INT	Для типов данных	INT, DINT
ANY_NUM	Для типов данных	ANY_INT and REAL

Операция	Запись	1й адрес	2й адрес	Результат	Приоритет
Степень	**	ANY_NUM	ANY_NUM	REAL	2
Унарный плюс	+	ANY_NUM	-	ANY_NUM	3
		TIME	-	TIME	3
Унарный минус	-	ANY_NUM	-	ANY_NUM	3
		TIME	-	TIME	3
Умножение	*	ANY_NUM	ANY_NUM	ANY_NUM	4
		TIME	ANY_INT	TIME	4
Деление	/	ANY_NUM	ANY_NUM	ANY_NUM	4
		TIME	ANY_INT	TIME	4
Деление нацело	DIV	ANY_INT	ANY_INT	ANY_INT	4
		TIME	ANY_INT	TIME	4
Деление по модулю	MOD	ANY_INT	ANY_INT	ANY_INT	4
Сложение	+	ANY_NUM	ANY_NUM	ANY_NUM	5
		TIME	TIME	TIME	5
		TOD	TIME	TOD	5
		DT	TIME	DT	5
Вычитание	-	ANY_NUM	ANY_NUM	ANY_NUM	5
		TIME	TIME	TIME	5
		TOD	TIME	TOD	5
		DATE	DATE	TIME	5
		TOD	TOD	TIME	5
		DT	TIME	DT	5
		DT	DT	TIME	5

Примечание

При определении типа адреса помните, что тип адреса зависит от типа результата операции. Если операция возвращает, например, тип INTEGER, Вы не должны использовать тип REAL при задании адреса.

Правила

Операции в арифметических выражениях выполняются в порядке старшинства.

- Для лучшей ясности рекомендуется брать отрицательные величины в скобки даже в случаях, когда это не является синтаксически необходимо.
- Если в операции деления участвуют два целых числа, то результаты операций “DIV" и “/" совпадают (см. пример ниже).
- При операциях деления (`/', `MOD' and `DIV'), делитель не может быть равен нулю.
- Если одно из чисел принадлежит к типу INT (целое), а все остальные числа принадлежат к типу REAL (вещественное число), результат всегда будет принадлежать типу REAL.

Примеры

```
// Результат арифметического выражения (11)
// присвоен переменной "VALUE"
VALUE1 := 3 + 3 * 4 / 2 - (7+3) / (-5) ;
// Переменная VALUE2 в следующем выражении равна 1
VALUE2 := 9 MOD 2 ;
```

10.7 Логические выражения

Логическое выражение – это выражение, составленное при помощи логических операций.

Основные логические операции

В логических выражениях могут быть использованы операции AND, &, XOR и OR, логические адреса (типа BOOL) или переменные типа BYTE, WORD или DWORD. Для отрицания логического адреса используется оператор NOT.

Логические операции:

Результат логического выражения над булевскими адресами всегда TRUE или FALSE, либо набор двоичных разрядов при логической операции сравнения двух адресов.

Следующая таблица представляет допустимые логические выражения и типы данных для результата в зависимости от типа адресов. Использовались следующие сокращения:

ANY_BIT Для типов данных BOOL, BYTE, WORD, DWORD

Операция	Запись	1й адрес	2й адрес	Результат	Приоритет
Отрицание	NOT	ANY_BIT	-	ANY_BIT	3
Конъюнкция	AND	ANY_BIT	ANY_BIT	ANY_BIT	8
Исключающая дизъюнкция	XOR	ANY_BIT	ANY_BIT	ANY_BIT	9
Дизъюнкция	OR	ANY_BIT	ANY_BIT	ANY_BIT	10

Результат:

Результат логического выражения может быть:

- 1 (*true*) или 0 (*false*), если оперируют булевскими адресами, или
- Набор двоичных разрядов, отвечающий результату побитового сравнения двух адресов.

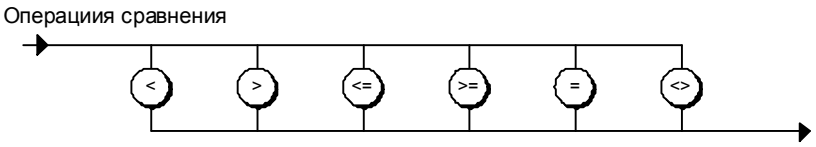
Пример

```
// Отрицание результата сравнения.
    IF NOT (COUNTER > 5) THEN . . . ;
// Результат первого сравнения отрицается и
// комбинируется с результатом второго
    A := NOT (COUNTER1 = 4) AND (COUNTER2 = 10) ;
// Дизъюнкция двух результатов сравнения
    WHILE  (A >= 9) OR (SCAN <> "n") DO.... ;
// Маскирования входного байта (битовая операция)
    Result := IB10 AND 2#11110000 ;
```

10.8 Выражения сравнения

Выражения сравнения сравнивают величины двух адресов и возвращают булевскую величину. Результат равен TRUE, если условие сравнения истинно, или FALSE, если оно ложно.

Синтаксис



Правила

К выражениям сравнения применимы следующие правила:

- Можно сравнивать любые величины следующих типов:
  - INT, DINT, REAL
  - BOOL, BYTE, WORD, DWORD
  - CHAR, STRING
- Величины следующих типов могут сравниваться только сами с собой:
  - DT, TIME, DATE, TOD
- При сравнении символов (тип CHAR), используется величина ASCII кодов символов.
- Величины формата S5TIME не могут участвовать в операциях сравнения. Формат S5TIME должен быть преобразован в формат TIME с использованием функций IEC.
- Выражения сравнения могут комбинироваться в соответствии с правилами булевской логики при создании выражений типа "if a < b and b < c then ...".
  (Пример: Value_A > 20 AND Value_B < 20)
  Операции выполняются в порядке приоритета. Приоритет операций может быть изменен скобками.
  Пример:
  A<>(B AND C)

Примеры

```
// Сравнение 3 меньше или равно 4.
// Результат "TRUE" (Истина)
   A := 3 <= 4
// Сравнение 7 не равно 7. Результат "FALSE" (Ложь)
   7 <> 7
// Вычисление выражения сравнения в операторе IF
   IF COUNTER < 5 THEN ...
```

11 Операторы

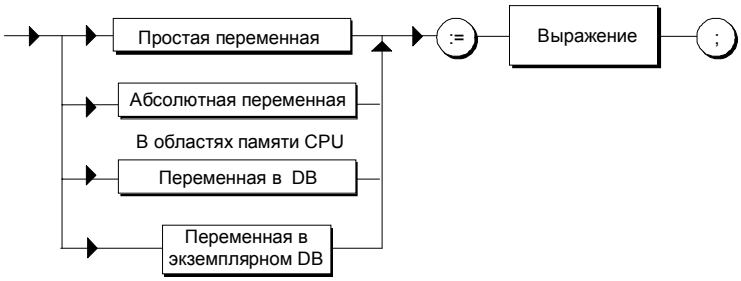
11.1 Присвоение значений

При присвоении текущая величина переменной заменяется новым значением, заданным в выражении. При записи этих выражений в них можно также включать функции, возвращающие соответствующие значения (возвращаемая величина).

Как показано на диаграмме, переменной, имя которой записано слева от оператора присвоения, присваивается вычисленное значение выражения записанного справа от оператора присвоения. На рисунке указаны типы переменных, применимых в данном случае.

Синтаксис присвоения величин

Присвоение значения



Тип выражения должен совпадать с типом адреса слева от оператора присвоения. Простая переменная может иметь простой или сложный тип данных.

11.1.1 Присвоение величин переменных простых типов

Значение любого выражение и любой переменной простого типа может быть присвоено другой переменной того же типа.

```
Identifier := Expression ;
Identifier := Variable ;
```

Пример

```
FUNCTION_BLOCK FB12
VAR
  SWITCH_1          : INT ;
  SWITCH_2          : INT ;
  SETPOINT_1        : REAL ;
  SETPOINT_2        : REAL ;
  QUERY_1 : BOOL ;
  TIME_1            : S5TIME ;
  TIME_2            : TIME ;
  DATE_1            : DATE ;
  TIMEOFDAY_1       : TIME_OF_DAY ;
END_VAR
BEGIN

// Присвоение переменным констант
  SWITCH_1      := -17 ;
  SETPOINT_1    := 100.1 ;
  QUERY_1       := TRUE ;
  TIME_1        := T#1H_20M_10S_30MS ;
  TIME_2        := T#2D_1H_20M_10S_30MS ;
  DATE_1        := D#1996-01-10 ;

// Присвоение переменным переменных
  SETPOINT_1  := SETPOINT_2 ;
  SWITCH_2    := SWITCH_1 ;

// Присвоение переменной результата выражения
  SWITCH_2  := SWITCH_1 * 3 ;
END_FUNCTION_BLOCK
```

11.1.2 Присвоение величин переменным типа STRUCT и UDT

Переменные типов STRUCT и UDT – это структурные переменные, которые представляют любую полную структуру или компонент структуры.

Далее представлены примеры правильно заданных структурных переменных:

```
Image                  //Идентификатор структуры
Image.element          //Идентификатор компонента структуры
Image.arr              //Идентификатор массива «в целом» в
                       //структуре
Image.arr[2,5]         //Идентификатор элемента массива
                       //в структуре
```

Задание структур в целом

Структура в целом может быть задана только другой структурой, компоненты которой совпадают друг с другом по имени и типу данных. Будет правильным следующее присвоение:

```
structname_1 := structname_2 ;
```

Задание компонентов структуры

Любой компонент структуры можно задать с помощью переменной того же типа, выражения того же типа или другого компонента структуры.

Вы можете сослаться на компонент структуры, задав идентификаторы структуры и ее компонента, разделенные точкой. Будут правильными следующие присвоения:

```
structname_1.element1          := Value ;
structname_1.element1          := 20.0 ;
structname_1.element1          := structname_2.element1 ;
structname_1.arrname1          := structname_2.arrname2 ;
structname_1.arrname[10]       := 100 ;
```

Пример

```
FUNCTION_BLOCK FB3
VAR
   AUXVAR : REAL ;
   MEASVAL : STRUCT    //Целевая структура
        VOLTAGE:REAL ;
        RESISTANCE:REAL ;
        SIMPLEARR : ARRAY [1..2, 1..2] OF INT ;
       END_STRUCT ;
   PROCVAL  : STRUCT    //Структура-источник
        VOLTAGE : REAL ;
        RESISTANCE : REAL ;
        SIMPLEARR : ARRAY [1..2, 1..2] OF INT ;
       END_STRUCT ;
END_VAR

BEGIN
//Присвоение структуры в целом
   MEASVAL := PROCVAL ;
//Присвоение компонента структуры компоненту структуры
   MEASVAL.VOLTAGE := PROCVAL.VOLTAGE ;
//Присвоение компонента структуры переменной того же типа
   AUXVAR := PROCVAL.RESISTANCE ;
//Присвоение константы компоненту структуры
   MEASVAL.RESISTANCE := 4.5;
//Присвоение константы элементу массива в структуре
   MEASVAL.SIMPLEARR[1,2] := 4;
END_FUNCTION_BLOCK
```

11.1.3 Присвоение значений переменных типа ARRAY

Массив может иметь размерность от одного до шести и должен состоять из элементов одинакового типа. При назначении значений массиву возможны два варианта доступа. Вы можете ссылаться на массивы в целом или на элементы массива.

Задание массивов в целом

Массив в целом может быть присвоен другому массиву в случае, если совпадают как тип компонентов массива, так и размеры массива (наименьший и наибольший индексы массива). В этом случае, после оператора присвоения, задается идентификатор массива. Будет правильным следующее присвоение:

```
arrname_1 := arrname_2 ;
```

Задание элементов массива

Обращение к элементу массива осуществляется при помощи имени массива вместе с соответствующими индексами в квадратных скобках. Каждому измерению массива соответствует свой индекс. Они разделены запятыми и заключены в общие квадратные скобки. Индекс должен быть арифметическим выражением типа INT.

Для присвоения величин подмассиву, исключаются индексы массива в квадратных скобках после его имени, начиная справа. В этом случае Вы обращаетесь к подмассиву, число измерений которого равно числу исключенных индексов. Будут правильными следующие присвоения:

```
arrname_1[ i ]      := arrname_2[ j ] ;
arrname_1[ i ]      := expression ;
identifier_1        := arrname_1[ i ] ;
```

Пример

```
FUNCTION_BLOCK FB3
VAR
 SETPOINTS          :ARRAY [0..127] OF INT ;
 PROCVALS           :ARRAY [0..127] OF INT ;
// Объявление матрицы (двумерного массива)
// с 3 строками и 4 столбцами
 CRTLLR : ARRAY [1..3, 1..4] OF INT ;
// Объявление вектора (одномерного массива) с 4 элементами
 CRTLLR_1 : ARRAY [1..4] OF INT ;
END_VAR

BEGIN
// Присвоение массива в целом
 SETPOINTS := PROCVALS ;
// Присвоение вектора второй строке матрицы CRTLLR    CRTLLR[2] := CRTLLR_1 ;
// Присвоение элемента массива элементу массива
 CRTLLR [1,4] := CRTLLR_1 [4] ;
END_FUNCTION_BLOCK
```

11.1.4 Присвоение значений переменным типа STRING

Переменная типа STRING представляет собой строку с длиной не более 254 символов. Любая переменная типа STRING может быть присвоена переменной того же типа. Будут правильными следующие присвоения:

```
stringvariable_1 := stringconstant;
stringvariable_1 := stringvariable_2 ;
```

Пример

```
FUNCTION_BLOCK FB3
VAR
  DISPLAY_1          : STRING[50] ;
  STRUCTURE1         : STRUCT
     DISPLAY_2       : STRING[100] ;
     DISPLAY_3       : STRING[50] ;
  END_STRUCT ;
END_VAR

BEGIN
// Присвоение константы переменной типа STRING
        DISPLAY_1 := 'Error in module 1' ;
// Присвоение компоненты структуры переменной типа STRING
        DISPLAY_1 := STRUCTURE1.DISPLAY_3 ;
// Присвоение переменной типа STRING переменной типа STRING
        If DISPLAY_1 <> STRUCTURE1.DISPLAY_3 THEN
                DISPLAY_1 := STRUCTURE1.DISPLAY_3 ;
        END_IF;
END_FUNCTION_BLOCK
```

11.1.5 Присвоение значений переменным типа DATE_AND_TIME

Тип данных DATE_AND_TIME определяет область, состоящую из 64 бит (8 байт), в которой задается дата и времени. Любой переменной типа DATE_AND_TIME может быть присвоена любая переменная или константа того же типа. Будут правильными следующие присвоения:

```
dtvariable_1 := date_and_time_constant;
dtvariable_1 := dtvariable_2 ;
```

Пример

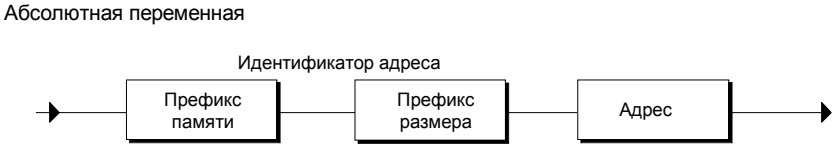
```
FUNCTION_BLOCK FB3
VAR
   TIME_1        : DATE_AND_TIME ;
   STRUCTURE1    :        STRUCT
         TIME_2 : DATE_AND_TIME ;
         TIME_3 : DATE_AND_TIME ;
   END_STRUCT ;
END_VAR

BEGIN
// Присвоение константы типа DATE_AND_TIME переменной
      TIME_1 := DATE_AND_TIME#1995-01-01-12:12:12.2 ;
      STRUCTURE1.TIME_3 := DT#1995-02-02-11:11:11 ;
// Присвоение компонента структуры переменной типа
// DATE_AND_TIME
      TIME_1 := STRUCTURE1.TIME_2 ;
// Присвоение переменной типа DATE_AND_TIME
// переменной того же типа
      If TIME_1 < STRUCTURE1.TIME_3 THEN
            TIME_1 := STRUCTURE1.TIME_3 ;
      END_IF ;
END_FUNCTION_BLOCK
```

11.1.6 Присвоение значений с абсолютной адресацией в областях памяти CPU

Абсолютная переменная ссылается на области памяти CPU с глобальной областью действия. Для получения доступа к этим областям существуют следующие три метода:

- Абсолютный доступ
- Индексированный доступ
- Символический доступ

Синтаксис абсолютной переменной



Пример

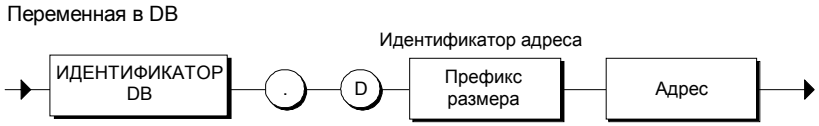
```
FUNCTION_BLOCK FB3
VAR
  STATUSWORD1   : WORD ;
  STATUSWORD2   : BOOL ;
  STATUSWORD3   : BYTE ;
  STATUSWORD4   : BOOL ;
  ADDRESS: INT ;
END_VAR
BEGIN
  ADDRESS := 10 ;
// Присвоение входного слова переменной (элементарный
// доступ)
  STATUSWORD1 := IW4 ;
// Присвоение переменной выходному биту (элементарный
// доступ)
  a1.1 := STATUSWORD2 ;
// Присвоение входного байта переменной (индексный
// доступ)
  STATUSWORD3 := IB[ADDRESS] ;
// Присвоение входного бита переменной (индексный
// доступ)
  FOR ADDRESS := 0 TO 7 BY 1 DO
    STATUSWORD4 := e[1, ADDRESS] ;
  END_FOR ;
END_FUNCTION_BLOCK
```

11.1.7 Присвоение глобальных переменных

Доступ к переменным в блоке данных также осуществляется присвоением величин переменным того же типа или наоборот. Вы можете присвоить любую глобальную переменную, переменную или выражение того же типа. Существует несколько способов доступа к этим данным:

- Структурный доступ
- Абсолютный доступ
- Индексированный доступ

Синтаксис переменной в DB



Пример

```
FUNCTION_BLOCK FB3
VAR
  CRTLLR_1          : ARRAY [1..4] OF INT ;
  STATUSWORD1   : WORD ;
  STATUSWORD2   : ARRAY [0..10] OF WORD ;
  STATUSWORD3   : INT ;
  STATUSWORD4   : WORD ;
  ADDRESS: INT ;
END_VAR
VAR_INPUT
  ADDRESSWORD  : WORD ;
END_VAR
BEGIN
// Присвоение слова 1 из DB11
// переменной (элементарный доступ)
    STATUSWORD1 := DB11.DW1 ;
// Первому элементу массива присвоено значение
// переменной "NUMBER" из DB11 (структурный доступ):
    CRTLLR_1[1] := DB11.NUMBER ;
// Присвоение компонента "NUMBER2" из структуры
// "NUMBER1" переменной STATUSWORD3
    STATUSWORD3  := DB11.NUMBER1.NUMBER2 ;
// Присвоение слова с индексным доступом
// из DB11 переменной
    FOR
      ADDRESS := 1 TO 10 BY 1 DO
      STATUSWORD2[ADDRESS] := DB11.DW[ADDRESS] ;
 // Здесь входной параметр ADDRESSWORD – номер
 // DB и индекс ADDRESS использован для определения адреса  // слова в DB.
       STATUSWORD4 := WORD_TO_BLOCK_DB(ADDRESSWORD).DW[ADDRESS]
;
      END_FOR ;
END_FUNCTION_BLOCK
```

11.2 Управляющие операторы

11.2.1 Обзор управляющих операторов

Условные операторы

Условные операторы позволяют Вам направлять выполнение программы по различным последовательностям операторов.

Тип перехода	Функция
Оператор IF	Оператор IF позволяет направлять выполнение программы по одной из двух ветвей в зависимости от выполнения или невыполнения условия.
Оператор CASE	Оператор CASE позволяет направлять выполнение программы по одному из n альтернативных путей, в зависимости от величины переменной.

Циклы

Вы можете управлять циклами, используя операторы повтора. Оператор повтора определяет, какие части программы должны повториться в зависимости от различных условий.

Тип перехода	Функция
Оператор FOR	Используется для повтора последовательности операторов до тех пор, пока величина управляющей переменной остается в пределах определенных границ
Оператор WHILE	Используется для повтора последовательности операторов до тех пор, пока удовлетворяется условие выполнения
Оператор REPEAT	Используется для повтора последовательности операторов до тех пор, пока не будет соблюдено условие завершения

Программный переход

Программный переход – это непосредственный переход к определенной цели, то есть к другому оператору внутри того же блока.

Тип перехода	Функция
Оператор CONTINUE	Используется для прерывания выполнения текущего прохода тела цикла.
Оператор EXIT	Используется для прерывания цикла в независимости от выполнения условия завершения
Оператор GOTO	Вызывает переход программы на ранее заданную метку
Оператор RETURN	Вызывает выход программы из выполняемого в данный момент блока

11.2.2 Условия

Условие может быть задано любым выражением сравнения, логическим выражением или арифметическим выражением. Оно имеет тип BOOL и может принимать значения TRUE или FALSE. Арифметическое выражение считается равным TRUE, если его результат отличен от нуля, и FALSE, если равен нулю. В следующей таблице даны примеры условий:

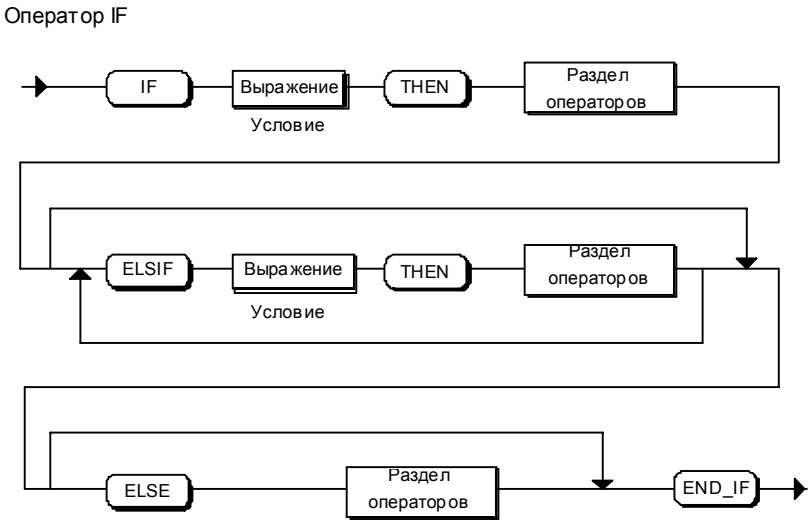
Тип	**Пример**
Выражение сравнения	TEMP > 50 COUNTER <= 100 CHAR1 < 'S'
Выражения сравнения и логические выражения	(ALPHA <> 12) AND NOT BETA
Булевский адрес	I 1.1
Арифметическое выражение	ALPHA = (5 + BETA)

11.2.3 Операторы IF

Оператор IF – это условный оператор. Он предусматривает одну или несколько возможностей и выбирает для исполнения один (или ни одного) из операторных компонентов.

При исполнении условного оператора вычисляется определенные логические выражения. Если величина выражения равна TRUE, то условие считается выполненным, если FALSE, то не выполненным.

Синтаксис



Оператор IF исполняется в соответствии со следующими правилами:

- Выполняется первая последовательность операторов, для которых логическое выражение равно TRUE. Остальные последовательности не исполняются.
- Если ни одно из булевских выражений не равно TRUE, то выполняется последовательность операторов заданная оператором ELSE (либо, в случае если переход ELSE отсутствует, не выполняется ни одна из последовательностей операторов).
- Можно использовать любое количество операторов ELSIF.

Примечание

Использование перехода ELSIF имеет следующую особенность: Логические выражения, следующие за верными выражениями, больше не вычисляются, в отличие от последовательности оператора IF. Следовательно, время выполнения программы может быть уменьшено.

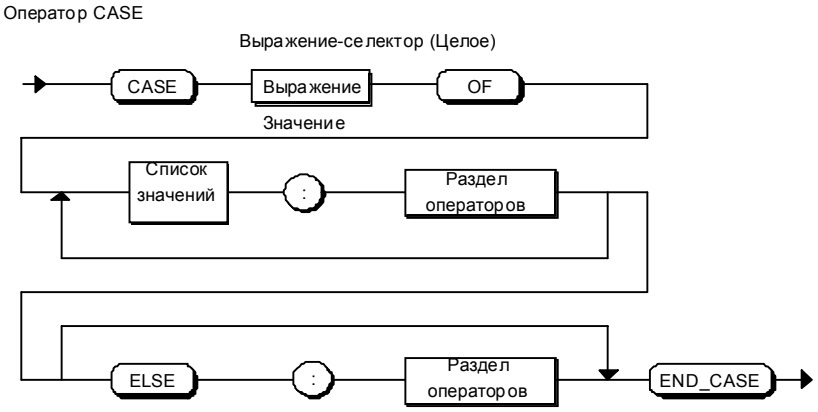
Пример

```
IF  I1.1 THEN
    N := 0 ;
    SUM := 0 ;
    OK := FALSE ;  // Установка флага OK в FALSE
ELSIF  START = TRUE THEN
    N := N + 1 ;
    SUM := SUM + N ;
ELSE
    OK := FALSE ;
END_IF ;
```

11.2.4 Оператор CASE

Оператор CASE используется для выбора одного из нескольких альтернативных разделов программы. Выбор зависит от текущей величины условного выражения.

Синтаксис

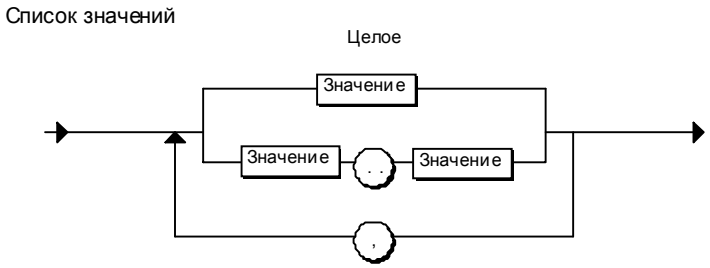


Оператор CASE исполняется в соответствии со следующими правилами:

- Условное выражение должно возвращать величину типа INTEGER.
- При исполнении оператора CASE, программа проверяет, какой величине из определенного списка переменных равна величина условного выражения. Если совпадающие величины найдены, выполняются операторы из соответствующего списка.
- Если совпадений не найдено, то выполняется часть программы соответствующая переходу ELSE, либо, в случае если переход ELSE не определен, не выполняет никаких операций.

Список величин

Данный рисунок содержит величины, которые можно использовать в списке условий.

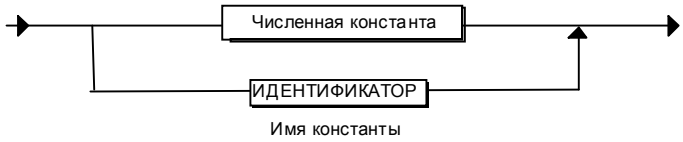


Список величин должен удовлетворять следующим правилам:

- Каждая величина в списке начинается с константы, списка констант или интервала констант.
- Величины из списка должны соответствовать типу INTEGER.
- Каждая величина может встречаться один раз.

Значение

Значение имеет синтаксис приведенный ниже:



Пример

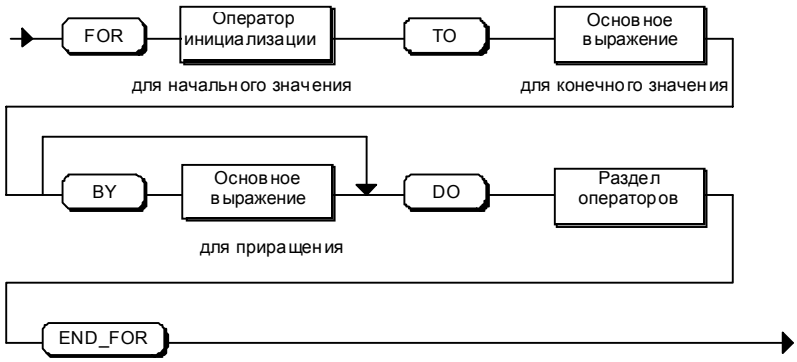
```
CASE TW OF
   1 :            DISPLAY:= OVEN_TEMP;
   2 :            DISPLAY:= MOTOR_SPEED;
   3 :            DISPLAY:= GROSS_TARE;
                         QW4:= 16#0003;
   4..10:          DISPLAY:= INT_TO_DINT (TW);
                  QW4:= 16#0004;
   11,13,19:       DISPLAY:= 99;
                  QW4:= 16#0005;
ELSE:
      DISPLAY:= 0;
      TW_ERROR:= 1;
END_CASE ;
```

11.2.5 Оператор FOR

Используется для повтора последовательности операторов до тех пор, пока управляющая переменная остается в заданных границах. Управляющей переменной должен быть идентификатор локальной переменной типа INT или DINT. При определении цикла оператора FOR необходимо указать величины начального и конечного значения управляющей переменной. Обе величины должны быть того же типа, что и управляющая переменная.

Синтаксис

Оператор FOR



Оператор FOR исполняется следующим образом:

- В начале цикла управляющей переменной присваивается начальное значение (начальное присвоение) и каждый раз при выполнении одного витка цикла она увеличивается на заданную величину (положительное приращение) или уменьшается (отрицательное приращение), до тех пор, пока не будет достигнуто конечное значение.
- При каждом витке цикла проверяется условие того, что управляющая переменная меньше конечного значения. Если условие удовлетворено, то выполняется последовательность операторов, в противном случае цикл прекращается.

Правила

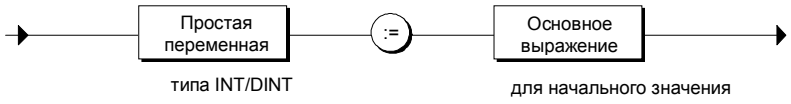
Правила для записи оператора FOR

- Управляющая переменная может принадлежать только к типам INT или DINT.
- Можно пропускать оператор BY [приращение]. Если величина приращения не задана, она автоматически задается равной +1.

Начальное значение

Величина начального значения управляющей переменной должна быть записана, как показано на рисунке. Простая переменная слева от оператора присвоения должна быть типа INT или DINT.

Оператор инициализации

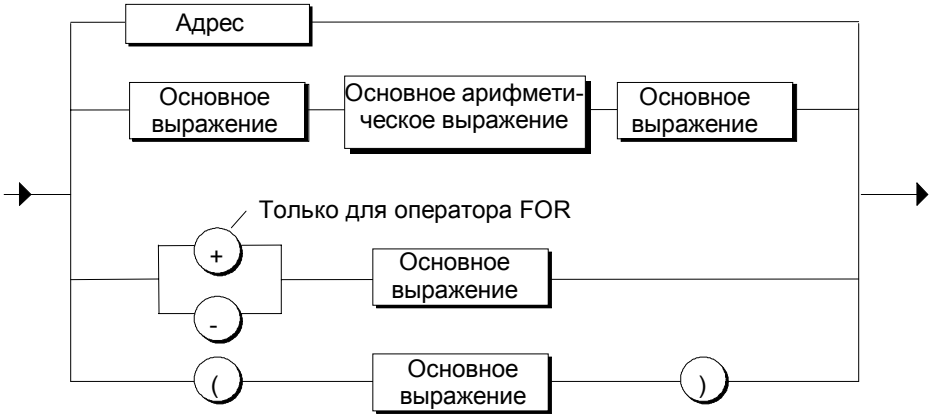


Примеры правильного присвоения:

```
FOR I := 1 TO 20
FOR I := 1 TO (START + J)
```

Конечное значение и приращение

Вы можете записать основное выражение для конечной величины и требуемого приращения. Это основное выражение должно быть записано следующим образом:



- Можно пропускать оператор BY [••••••••••]. Если величина приращения не задана, она автоматически задается равной *+1*.
- Начальное значение, конечное значение и приращение – это выражения (см. "Выражения, операции и адреса"). Они вычисляются перед началом выполнения цикла FOR.
- В процессе исполнения цикла нельзя изменять величину конечного значения и приращения.

Пример

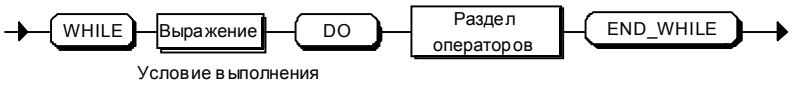
```
FUNCTION_BLOCK FOR_EXA
VAR
  INDEX: INT ;
  IDWORD: ARRAY [1..50] OF STRING;
END_VAR
BEGIN
FOR INDEX := 1 TO 50 BY 2 DO
  IF IDWORD [INDEX] = 'KEY' THEN
                EXIT;
  END_IF;
END_FOR;

END_FUNCTION_BLOCK
```

11.2.6 Оператор WHILE

Оператор WHILE предназначен для многократного исполнения последовательности операторов, управляемого условием исполнения. Условие исполнения формируется в соответствии с правилами для логических выражений.

Синтаксис

Оператор WHILE



Оператор WHILE исполняется в соответствии со следующими правилами:

- Условие исполнения вычисляется заново при каждом витке цикла.
- Тело цикла, находящееся после DO, выполняется до тех пор, пока условие исполнения будет иметь величину TRUE.
- Как только возникает величина FALSE, цикл прекращается, и исполняются следующий за циклом оператор.

Пример

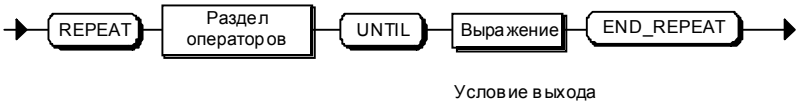
```
FUNCTION_BLOCK WHILE_EXA
VAR
  INDEX: INT ;
  IDWORD: ARRAY [1..50] OF STRING ;
END_VAR
BEGIN
INDEX := 1 ;
WHILE INDEX <= 50 AND IDWORD[INDEX] <> 'KEY' DO
  INDEX := INDEX + 2;
END_WHILE ;
END_FUNCTION_BLOCK
```

11.2.7 Оператор REPEAT

Оператор REPEAT вызывает повторное исполнение последовательности операторов находящихся между REPEAT и UNTIL вплоть до удовлетворения условия прекращения. Условие прекращения формируется в соответствии с правилами для логических выражений.

Синтаксис

Оператор REPEAT



Условие вычисляется после исполнения тела цикла. Это значит, что тело цикла исполняется хотя бы единожды, даже в случае, когда условие прекращения удовлетворено при начале цикла.

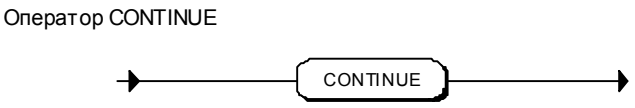
Пример

```
FUNCTION_BLOCK REPEAT_EXA
VAR
  INDEX: INT ;
  IDWORD: ARRAY [1..50] OF STRING ;
END_VAR

BEGIN
INDEX := 0 ;
REPEAT
  INDEX := INDEX + 2 ;
UNTIL INDEX > 50 OR IDWORD[INDEX] = 'KEY'
END_REPEAT ;

END_FUNCTION_BLOCK
```

11.2.8 Оператор CONTINUE

Оператор CONTINUE предназначен для прерывания исполнения текущего повтора операторов цикла (FOR, WHILE или REPEAT).

Синтаксис



Оператор CONTINUE исполняется в соответствии со следующими правилами:

- Этот оператор немедленно прерывает исполнение тела цикла.
- В зависимости от того удовлетворено ли условие продолжения тела цикла, тело цикла исполняется снова или оператор повторения прерывается и исполняется следующий оператор.
- В случае оператора FOR, к управляющей переменной сразу же после оператора CONTINUE прибавляется заданное приращение.

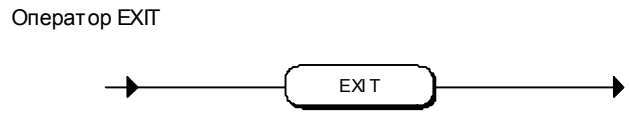
Пример

```
FUNCTION_BLOCK CONTINUE_EXA
VAR
  INDEX     :INT ;
  ARRAY     :ARRAY[1..100] OF INT ;
END_VAR

BEGIN
INDEX := 0 ;
WHILE INDEX <= 100 DO
  INDEX := INDEX + 1 ;
  // Если ARRAY[INDEX] равно INDEX,
  // то ARRAY [INDEX] не изменяется:
     IF ARRAY[INDEX] = INDEX THEN
       CONTINUE ;

       END_IF ;
     ARRAY[INDEX] := 0 ;
  // Другие инструкции
END_WHILE ;
END_FUNCTION_BLOCK
```

11.2.9 Оператор EXIT

Оператор EXIT предназначен для выхода из цикла (FOR, WHILE или REPEAT) в любой момент, вне зависимости от того удовлетворено ли условие окончания цикла.

Синтаксис



Оператор EXIT исполняется в соответствии со следующими правилами:

- Этот оператор вызывает немедленное прерывание оператора того цикла, внутри которого он находится.
- Выполнение программы продолжается с места окончания цикла (например, после END_FOR).

Пример

```
FUNCTION_BLOCK EXIT_EXA
VAR
  INDEX_1  : INT ;
  INDEX_2  : INT ;
  INDEX_SEARCH   : INT ;
  IDWORD  : ARRAY[1..51] OF STRING ;
END_VAR

BEGIN
INDEX_2    := 0 ;
FOR INDEX_1         := 1 TO 51 BY 2 DO
  // Выход из цикла FOR, если
  // IDWORD[INDEX_1] равно 'KEY':
  IF IDWORD[INDEX_1] = 'KEY' THEN
    INDEX_2 := INDEX_1 ;
    EXIT ;
  END_IF ;
END_FOR ;
// Другие операции присвоения
// после выполнения EXIT или после обычного
// окончания цикла FOR:
INDEX_SEARCH := INDEX_2 ;
END_FUNCTION_BLOCK
```

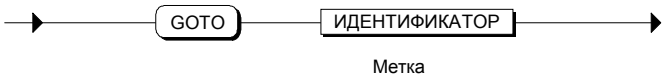
11.2.10 Оператор GOTO

Вы можете реализовать переход в программе, используя оператор GOTO. Это значит, что выполнение программы немедленно перейдет к заданной метке, и, следовательно, к другому оператору внутри того же блока.

Оператор GOTO должен использоваться только в специфических случаях, например при обработке ошибок. В соответствии с правилами структурного программирования оператор GOTO не должен использоваться.

Синтаксис

Оператор GOTO



Используемая здесь метка должна быть объявлена в разделе деклараций LABEL/END_LABEL. Она записывается перед оператором, который должен исполняться после оператора GOTO.

Если Вы пользуетесь оператором GOTO, помните следующие правила:

- Цель перехода должна находиться в том же блоке.
- Цель перехода должна быть однозначно определена.
- Нельзя перейти внутрь цикла, но из цикла переход возможен.

Пример

```
FUNCTION_BLOCK GOTO_EXA
VAR
  INDEX     : INT ;
  A : INT ;
  B : INT ;
  C : INT ;
  IDWORD  : ARRAY[1..51] OF STRING ;
END_VAR
LABEL
  LAB1, LAB2, LAB3 ;
END_LABEL

BEGIN
IF A > B THEN
  GOTO LAB1 ;
ELSIF A > C THEN
  GOTO LAB2 ;
END_IF ;
// . . .
LAB1: INDEX := 1 ;
     GOTO LAB3 ;
LAB2: INDEX := 2 ;
// . . .
LAB3:
// . . .
```

11.2.11 Оператор RETURN

Оператор RETURN совершат выход из активного в данный момент блока (OB, FB, FC) и возвращается к вызывающему блоку или в операционную систему, когда происходит выход из OB.

Синтаксис

Оператор RETURN

Примечание

Оператор RETURN в конце раздела кода логического блока или раздела объявления блока данных излишен, так как в этих случаях он выполняется автоматически.

11.3 Вызов функций и функциональных блоков

11.3.1 Вызов и передача параметров

Вызов FC и FB

Для того чтобы облегчить чтение и коррекцию пользовательских программ, функции программы делят на более мелкие индивидуальные задания, которые выполняются функциональными блоками (FB) и функциями (FC). Из SCL блока Вы можете вызывать другие FC и FB. Вы можете вызывать следующие блоки:

- Функциональные блоки и функции, созданные в SCL
- Функциональные блоки и функции, созданные на других языках STEP 7 (LAD, FBD, STL)
- Системные функции (SFC) и системные функциональные блоки (SFB), доступные в операционной системе CPU.

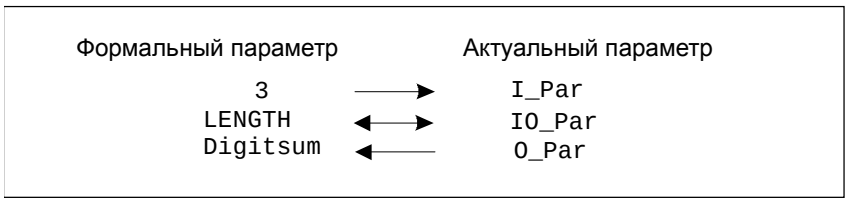
Основные принципы передачи параметров

Когда вызывается функция или функциональный блок, происходит обмен данными между вызывающим и вызываемым блоком. Параметры, с которыми работает блок, определены в интерфейсе вызываемого блока. Эти параметры называются формальными параметрами. Они являются всего лишь “заменителями" для параметров, которые поступают в блок тогда, когда его вызывают. Параметры, которые поступают в блок, называются фактическими параметрами.

Синтаксис передачи параметров

Передаваемые параметры должны быть определены при вызове в форме списка параметров. Параметры заключаются в скобки и разделяются запятыми.

В примере, приведенном ниже, определены входной, вход/выходной и выходной параметры.



Определение параметров имеют форму присвоения величин. Это присвоение передает величину (фактический параметр) параметру, определенному в разделе объявления вызываемого блока (формальному параметру).

Абсолютный вызов функции:

```
FB10.DB20 (X1:=5,X2:=78,......);
```

Назначение параметров

Символический вызов функции:

```
DRIVE.ON (X1:=5,X2:=78,......);
```

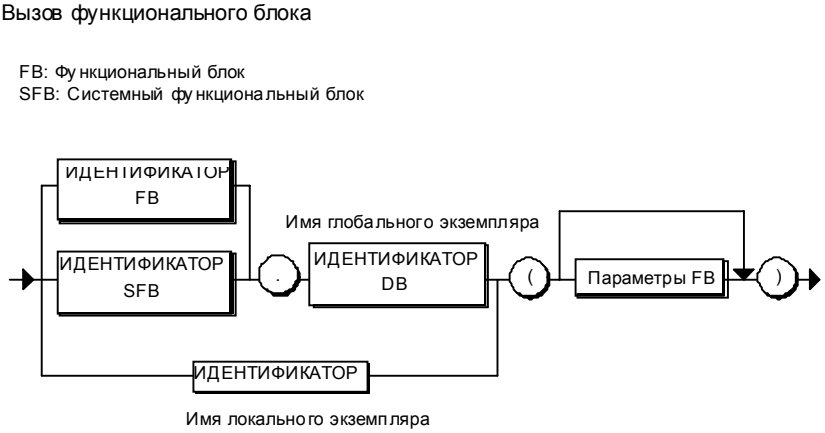
11.3.2 Вызов функциональных блоков

Когда Вы вызываете функциональный блок, Вы можете использовать как глобальные экземплярные блоки, так и локальные экземплярные области активного в данный момент экземплярного блока.

Вызов FB как локального экземпляра отличается от вызова FB как глобального экземпляра методом хранения данных. В данном случае данные хранятся не в собственном DB, а в экземплярном блоке вызывающего FB.

Синтаксис

Вызов функционального блока

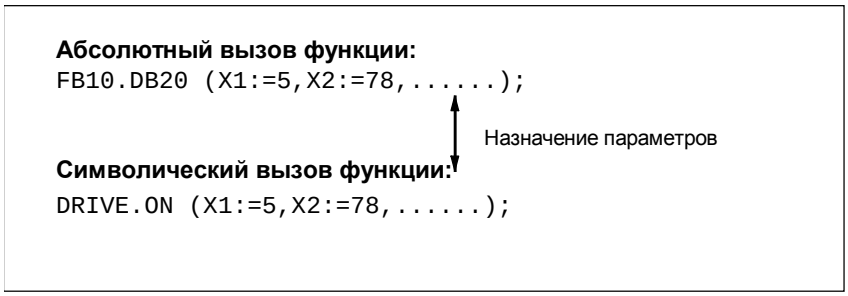


Вызов как глобального экземпляра

В команде вызова определяются:

- Имя функционального или системного функционального блока (идентификатор FB или SFB),
- Экземплярный блока данных (идентификатор DB),
- Параметры FB.

Вызов функции для глобального экземпляра может быть абсолютным или символическим.



Вызов как локального экземпляра

В команде вызова определяются:

- Имя локального экземпляра (Идентификатор)
- Параметры FB

Вызов локального экземпляра всегда символический. Вы должны объявить символическое имя в разделе объявления вызывающего блока.

11.3.2.1 Задание параметров FB

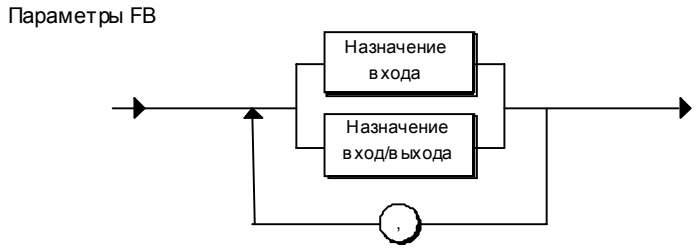
При вызове функционального блока (как глобального или локального экземпляра) Вы можете задать следующие параметры:

- Входные параметры
- Вход/выходные параметры

При вызове FB выходные параметры определять не нужно.

Синтаксис присвоения величин при определении параметров FB

Синтаксис параметров FB одинаков в случае вызова глобальных и локальных экземпляров.



При задании параметров должны удовлетворяться следующие правила:

- Присвоения могут идти в любом порядке.
- Типы формальных и фактических параметров должны совпадать.
- Присвоения разделены запятыми.
- При вызове FB невозможны выходные присвоения. Значение объявленного выходного параметра хранится в экземплярных данных. Отсюда они доступны любым FB, в которых Вы должны определить доступ к этим параметрам.
- Помните об особенностях параметров типа ANY и POINTER.

Результат исполнения блока

После исполнения блока:

- Фактические параметры передаются неизменными.
- Переданные и измененные величины вход/выходных параметров обновляются; вход/выходные параметры элементарных типов данных являются исключением.
- Выходные параметры могут быть прочитаны вызывающим блоком из глобальных экземплярных блоков или локальной экземплярной области

Вызов с присвоением входных и вход/выходных параметров может выглядеть следующим образом:

```
FB31.DB77(I_Par:=3, IO_Par:=LENGTH);
```

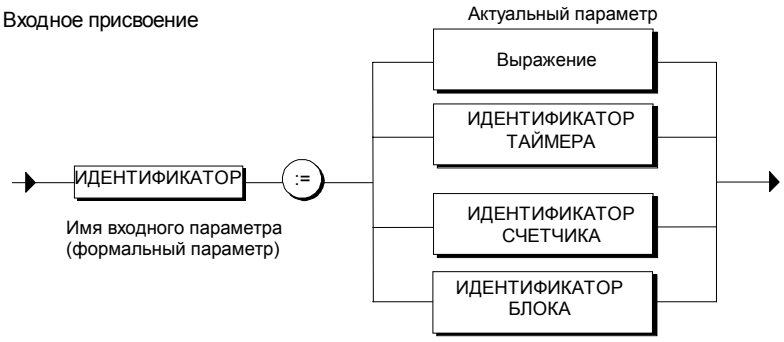
11.3.2.2 Входное присвоение (FB)

Входное присвоение копирует значения фактических параметров в формальные параметры. FB не может изменять эти фактические параметры. Присвоение входных фактических параметров необязательно. Если не задано входных фактических параметров, то используются величины, заданные при последнем вызове, и сохраненные в DB.

Ниже приведены возможные фактические параметры:

Фактический параметр	**Пояснение**
Выражение	• Арифметическое, логическое или выражение сравнения • Константа • Расширенная переменная
Идентификатор таймера/счетчика	Определяет конкретный таймер или счетчик для использования в функциональном блоке.
Идентификатор блока	Определяет конкретный блок для использования в качестве входного параметра. Тип блока (FB, FC или DB) определяется при объявлении входных параметров. Когда присвоены величины параметров, задайте номер блока. Вы можете задать его в абсолютной или символической форме.

Синтаксис



11.3.2.3 Вход/выходное присвоение (FB)

Вход/выходное присвоение нужно для присвоения фактических параметров формальным вход/выходным параметрам. Вызываемый FB может изменять вход/выходные параметры. Новая величина параметра, которая получается в результате выполнения FB, записывается обратно в фактические параметры. Начальное значение переписывается.

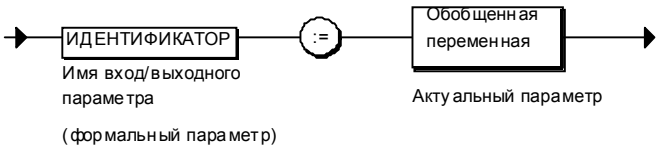
Если в вызываемом FB объявлены вход/выходные параметры сложного типа, для них должны быть заданы фактические параметры во время первого вызова блока. Когда он исполняется снова, задание фактических параметров необязательно. В случае вход/выходных параметров элементарного типа не происходит обновления фактических параметров, если при вызове блока они не заданы.

Так как заданный фактический вход/выходной параметр может измениться в процессе обработки блока, он должен быть переменной.

Фактический параметр	**Пояснение**
Расширенная переменная	Возможны следующие типы расширенных переменных: • Простые переменные и параметры • Доступ к абсолютным переменным • Доступ к блокам данных • Вызовы функций

Синтаксис

Назначение вход/выхода



Примечание

- Для задания величин типа ANY и POINTER существуют специальные правила.
- Нельзя использовать в качестве фактических параметров для вход/выходных параметров неэлементарного типа:
  - Вход/выходные параметры FB
  - Параметры FC

11.3.2.4 Чтение выходных величин (Вызов FB)

После того как вызванный блок был исполнен, выходные параметры могут быть считаны из глобального экземплярного блока или локальной экземплярной области с использованием присвоения величин.

Пример

```
RESULT:= DB10.CONTROL;
```

11.3.2.5 Пример вызова глобального экземпляра

Функциональный блок с циклом FOR может выглядеть так, как показано на приведенных примерах. В этих примерах предполагается, что в таблице символов FB17 присвоено символическое имя TEST.

Функциональный блок

```
FUNCTION_BLOCK TEST

VAR_INPUT
  FINALVAL:         INT;      //Входной параметр
END_VAR
VAR_IN_OUT
  IQ1          :    REAL;   // Вход/выходной параметр
END_VAR
VAR_OUTPUT
  CONTROL:          BOOL; // Выходной параметр
END_VAR
VAR
  INDEX:    INT;
END_VAR

BEGIN
CONTROL  :=FALSE;
FOR INDEX           := 1 TO FINALVAL DO
  IQ1          :=IQ1*2;
    IF IQ1 > 10000 THEN
      CONTROL        := TRUE;
    END_IF;
END_FOR;
END_FUNCTION_BLOCK
```

Вызов

Для вызова FB, Вы можете выбрать один из следующих вариантов. Предполагается, что переменная VARIABLE1 была объявлена в вызывающем блоке как REAL.

```
//Абсолютный вызов, глобальный экземпляр:
FB17.DB10 (FINALVAL:=10, IQ1:=VARIABLE1);

//Символический вызов, глобальный экземпляр:
TEST.TEST_1 (FINALVAL:=10, IQ1:= VARIABLE1);
```

Результат:

После исполнения блока величина, вычисленная для вход/выходного параметра IQ1, доступна через переменную VARIABLE1.

Чтение выходной величины

Два следующих примера показывают два возможных способа чтения выходного параметра `CONTROL`.

```
// Выходной параметр доступен через:
    RESULT:= DB10.CONTROL;

//Вы можете также использовать выходной параметр
//непосредственно при вызове другого FB,
//используя его как входной:
    FB17.DB12 (INP_1:=DB10.CONTROL);
```

11.3.2.6 Пример вызова как локального экземпляра

Функциональный блок с простым циклом FOR может быть запрограммирован так, как показано здесь в примере "Вызов как локального экземпляра". Предполагается, что символ TEST объявлен в таблице символов для FB17.

Этот FB может быть вызван так, как показано ниже. Предполагается, что переменная VARIABLE1 была объявлена в вызывающем блоке как REAL.

Вызов

```
FUNCTION_BLOCK CALL
VAR
// Объявление локального экземпляра
   TEST_L : TEST ;
   VARIABLE1 : REAL ;
   RESULT : BOOL ;
END_VAR
BEGIN
. . .

// Вызов локального экземпляра:
TEST_L (FINALVAL:= 10, IQ1:= VARIABLE1) ;
```

Чтение выходной величины

Выходной параметр CONTROL может быть прочитан следующим образом:

```
// Доступ к выходному параметру через:
RESULT := TEST_L.CONTROL ;
END_FUNCTION_BLOCK
```

11.3.3 Вызов функций

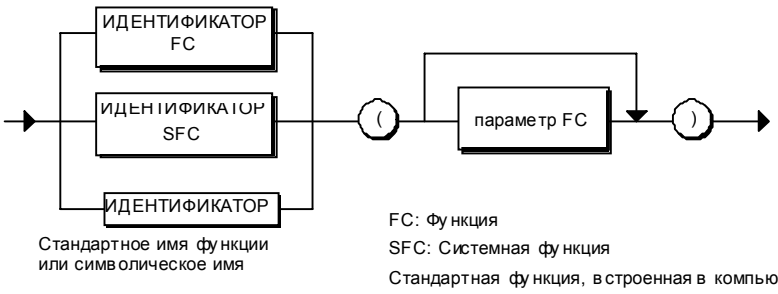
Вы вызываете функцию, написав ее имя (идентификатор FC, SFC) и список параметров. Имя функции определяет возвращаемую величину в абсолютной или символической форме:

```
FC31 (X1:=5, Q1:=Checksum)  ;          // Абсолютно
DISTANCE (X1:=5, Q1=:Checksum) ;   // Символически
```

После вызова, результаты выполнения функции доступны как возвращаемые величины или как выходные и вход/выходные параметры (фактические параметры).

Синтаксис

Вызов функции



.

Примечание

Если функция вызывается в SCL, и возвращаемая величина в ней не была задана, это может привести к некорректному выполнению пользовательской программы:

- Это может произойти с функцией, запрограммированной на SCL, если возвращаемая величина задана в программе, но этот оператор не был выполнен.
- Это может произойти с функцией, запрограммированной на STL/LAD/FBD, в случае, если функция была запрограммирована без определения возвращаемой величины, либо соответствующий оператор не был выполнен.

11.3.3.1 Возвращаемая величина (FC)

В отличие от функциональных блоков, функции имеют вычисленное значение, известное как возвращаемая величина, По этой причине функции могут обрабатываться как адреса (Исключение: Функции типа VOID).

Функция рассчитывает возвращаемую величину, которая имеет то же имя, что и функция и возвращает его вызывающему блоку. Там возвращаемая величина подставляется вместо вызова функции.

Например, в следующем операторе присвоения вызывается функция `DISTANCE`, а ее результат присваивается переменной `LENGTH`:

```
LENGTH:= DISTANCE (X1:=-3, Y1:=2);
```

Возвращаемая величина может быть использована в следующих элементах FC или FB:

- При присвоении величин,
- В логических, арифметических или выражениях сравнения, или
- Как фактический параметр для вызовов последующих функциональных блоков или функций.

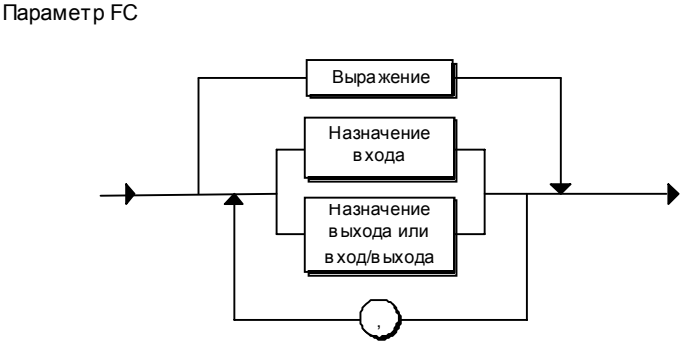
Примечание

- Если функция имеет возвращаемую величину типа ANY, как минимум один входной или вход/выходной параметр должен также принадлежать к типу ANY. Если определены один или более параметров типа ANY, Вы должны задать для них фактические параметры соответствующего класса типов (например, INT, DINT и REAL). Тогда возвращаемая величина автоматически будет принадлежать к набольшему типу в данном классе типов.
- Максимальная длина типа STRING может быть уменьшена с 254 символов до любой длины.

11.3.3.2 Параметры FC

В отличие от функциональных блоков, у функций нет областей памяти, в которых они могли бы хранить величины параметров. Локальные данные хранятся в памяти только в то время, когда функция активна. По этой причине, когда Вы вызываете функцию, всем формальным входным, вход/выходным и выходным параметрам, объявленным в разделе деклараций, должны быть присвоены величины фактических параметров.

Синтаксис



Правила

Правила для назначения параметров

- Присвоения могут следовать в любом порядке.
- Типы данных формальных и фактических параметров должны совпадать.

Пример

Вызов с заданием входных, выходных и вход/выходных параметров может выглядеть следующим образом:

```
FC32 (E_Param1:=5,D_Param1:=LENGTH,
    A_Param1:=Checksum)
```

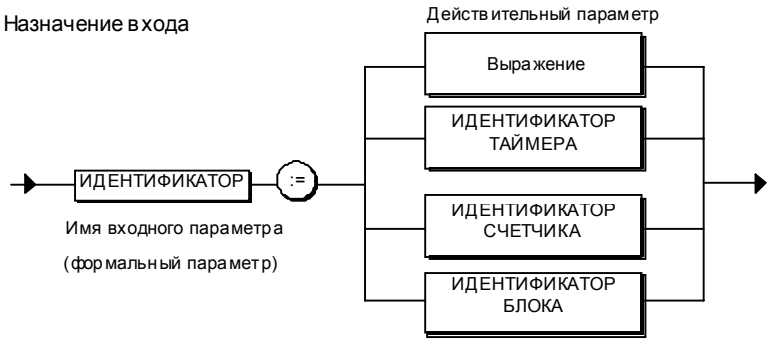
11.3.3.3 Входные присвоения (FC)

При использовании входных присвоений, формальным входным параметрам вызываемой функции присваиваются величины (фактические параметры). FC может работать с этими фактическими параметрами, но не может их менять. В отличие от вызова FB, при вызове FC это присвоение обязательно.

При входных присвоениях могут быть заданы следующие фактические параметры:

Фактический параметр	**Пояснение**
Выражение	Выражение обозначает величину и может состоять из адресов и операций. Возможны следующие типы выражений: • Арифметические, логические или выражения сравнения • Константы • Расширенные переменные
Идентификатор таймера/счетчика	Определяет конкретный таймер или счетчик для использования во время обработки блока.
Идентификатор блока	Определяет конкретный блок для использования в качестве входного параметра. Тип блока (FB, FC или DB) определяется при объявлении входных параметров. Когда присвоены величины параметров, задайте адрес блока. Вы можете задать его в абсолютной или символической форме.

Синтаксис



Примечание

В случае входных параметров неэлементарного типа в качестве фактических параметров не могут использоваться вход/выходные параметры FB и параметры FC. Помните об особенностях типов ANY и POINTER.

11.3.3.4 Выходные и вход/выходные присвоения (FC)

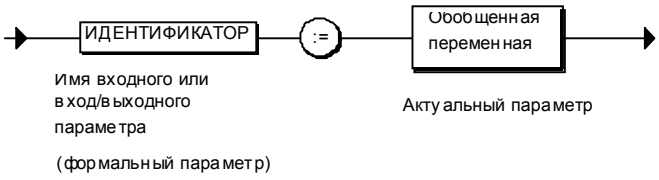
При выходных присвоениях, Вы определяете переменную вызывающего блока, в которую будут записаны выходные величины, являющиеся результатом выполнения функции. Вход/выходные присвоения используются при присвоении фактических величин вход/выходным параметрам.

При выходных и вход/выходных присвоениях фактические параметры должны изменяться в тот момент, когда FC записываются формальные параметры. Поэтому при вход/выходных присвоениях, входные параметры не могут быть заданы (величина не может быть записана). Это значит, что при выходных и вход выходных присвоениях могут задаваться только расширенные переменные.

Фактический параметр	**Пояснение**
Расширенная переменная	Возможны следующие типы расширенных переменных: • Простые переменные и параметры • Доступ к абсолютным переменным • Доступ к блокам данных • Вызовы функций

Синтаксис

Назначение входов и вход/выходов



Примечание

Следующие фактические параметры не могут быть использованы вместе с формальными выходными вход/выходными параметрами:

- Входные параметры FC/FB
- Вход/выходные параметры FB неэлементарного типа
- Вход/выходные и выходные параметры FC неэлементарного типа
- Помните об особенностях типов ANY и POINTER.
- Максимальная длина типа STRING может быть уменьшена с 254 символов до любой длины.

11.3.3.5 Пример функционального вызова

Вызываемая функция

Функция DISTANCE, использующаяся для подсчета расстояния между двумя точками (X1,Y1) и (X2,Y2), лежащими в одной плоскости, в декартовой системе координат, имеет следующую форму (пример подразумевает, что в таблице символов FC37 присвоено символическое имя DISTANCE).

```
FUNCTION DISTANCE: REAL // символическое имя
VAR_INPUT
   X1: REAL;
   X2: REAL;
   Y1: REAL;
   Y2: REAL;
END_VAR
VAR_OUTPUT
   Q2: REAL;
END_VAR
BEGIN
   DISTANCE:= SQRT( (X2-X1)**2 + (Y2-Y1)**2 );
        Q2:= X1+X2+Y1+Y2;
END_FUNCTION
```

Вызывающий блок

Этот пример демонстрирует дополнительные возможности использования величины функции:

```
FUNCTION_BLOCK CALL
VAR
    LENGTH : REAL ;
    CHECKSUM : REAL ;
    RADIUS : REAL;
    Y : REAL;
END_VAR
BEGIN
. . .
// ••••• ••• ••••••••• ••••••• •••••••••:
LENGTH := DISTANCE (X1:=3, Y1:=2, X2:=8.9, Y2:= 7.4,
          Q2:=CHECKSUM) ;

// ••••• • ••••••••••••• ••• ••••••••• ••••••••, ••••••••,
Y := RADIUS + DISTANCE (X1:=-3, Y1:=2, X2:=8.9, Y2:=7.4,
Q2:=Checksum)

// ••• ••••••••• ••••••••• ••• ••••• ••••••• •••••:
FB32.DB32 (DIST:= DISTANCE (X1:=-3, Y1:=2, X2:=8.9, Y2:=7.4),
Q2:=Checksum)
    . . .
    END_FUNCTION_BLOCK
```

11.3.4 Неявно заданные параметры

11.3.4.1 Входной параметр EN

У каждого функционального блока и у каждой функции есть неявно заданный входной параметр EN. Параметр EN имеет тип BOOL и хранится во временной области данных блока. Если EN равен TRUE, то вызванный блок исполняется. В противном случае не исполняется. Задание величины параметра EN необязательно. Однако помните, что он не должен быть объявлен в разделе деклараций блока или функции.

Так как EN – это входной параметр, Вы не можете изменять его внутри блока.

Примечание

Возвращаемая функцией величина не определяется, если функция не была вызвана (EN : FALSE).

Пример

```
FUNCTION_BLOCK FB57
VAR
  MY_ENABLE: BOOL ;
  Result : REAL;
END_VAR
// . . .
BEGIN
// . . .
MY_ENABLE:= FALSE ;

// Вызов функции и назначение параметра EN:
Result := FC85 (EN:= MY_ENABLE, PAR_1:= 27) ;
// FC85 не выполнено, так как MY_ENABLE равно FALSE

END_FUNCTION_BLOCK
```

11.3.4.2 Выходной параметр ENO

У каждого функционального блока и у каждой функции есть неявно заданный выходной параметр ENO, который имеет тип BOOL. Он хранится во временной области данных блока. Когда блок выполнен, текущая величина флага OK копируется в ENO.

Сразу после вызова блока, Вы можете проверить значение ENO, чтобы проверить, все ли операции в блоке выполнены корректно, или где-то возникла ошибка.

Пример

```
// Вызов функционального блока:
FB30.DB30 ([Назначение параметров]);
```

```
// Проверка правильности выполнения вызванного блока:
IF ENO THEN
// Когда все OK
// . . .
ELSE
// Была ошибка, требуется ее обработка
// ..
END_IF;
```

12 Счетчики и таймеры

12.1 Счетчики

12.1.1 Функции счетчиков

В STEP 7 Вам предоставляется ряд стандартных функций счетчиков. В SCL их можно использовать без предварительного объявления. Вы должны просто задать для них требуемые параметры. В STEP 7 имеются следующие функции счетчиков:

Функция счетчика	Пояснение
S_CU	Счет в прямом порядке
S_CD	Счет в обратном порядке
S_CUD	Счет в прямом и обратном порядке

12.1.2 Вызов функций счетчиков

Функции счетчиков вызываются как обычные функции. Поэтому идентификатор функции можно использовать вместо адреса в выражениях во всех случаях, когда тип возвращаемой величины соответствует типу замещаемого адреса.

Вызывающему блоку возвращается значение функции (возвращаемая величина), являющаяся текущим результатом счета (в формате BCD) типа WORD.

Абсолютный или динамический вызов

При вызове Вы должны передать функции номер счетчика, как абсолютное значение (например, C_NO:=C10). Эта величина не может изменяться в процессе выполнения программы.

Вместо абсолютного номера счетчика, Вы можете так же задать переменную или константу типа INT. Преимущество этого метода состоит в том, что при этом вызов счетчика становится динамическим, и ему, при каждом вызове, могут передаваться отличающиеся значения.

При динамическом вызове, Вы так же должны задать переменную типа COUNTER.

Примеры

```
//Пример абсолютного вызова:
S_CUD (C_NO:=C12,
  CD:=I0.0,
  CU:=I0.1,
  S:=I0.2 & I0.3,
  PV:=120,
  R:=FALSE,
  CV:=binVal,
  Q:=actFlag);

//Пример динамического вызова: В каждой итерации
//цикла FOR, вызывается разный счетчик:
FUNCTION_BLOCK COUNT
VAR_INPUT
    Count: ARRAY [1..4] of STRUCT
                C_NO: INT;
                PV  : WORD;
               END_STRUCT;
.
.
END_VAR
.
.
FOR I:= 1 TO 4 DO
  S_CD(C_NO := Count[I].C_NO, S := true, PV:= Count[I].PV);
END_FOR;

//Пример динамического вызова с использованием переменной
//типа COUNTER:
FUNCTION_BLOCK COUNTER
VAR_INPUT
  MYCounter : COUNTER;
END_VAR
.
.
CurrVal := S_CD (C_NO := MyCounter,.....);
```

Примечание

Имена функций и параметров совпадают в немецкой и английской мнемонике. Различаются только идентификаторы счетчиков (Немецкая: Z, Английская: C).

12.1.3 Задание параметров для функций счетчиков

В следующей таблице дан обзор параметров для функции счетчиков.

Параметр	Тип	Описание
C_NO	COUNTER INT	Номер счетчика (ИДЕНТИФИКАТОР СЧЕТЧИКА); размер области памяти зависит от CPU
CD	BOOL	CD вход: Обратный счет
CU	BOOL	CU вход: Прямой счет
S	BOOL	Сигнал предустановки счетчика
PV	WORD	Величина от 0 до 999 для инициализации счетчика (передается 16#<величина>, с величиной в формате BCD)
R	BOOL	Вход сброса
Q	BOOL	Выход: Состояние счетчика
CV	WORD	Выход: Содержимое счетчика в двоичном коде

Правила

Так как величина параметра хранится глобально (например, CD:=I0.0), необязательно определять ее в любых ситуациях. При задании величин параметров должны быть соблюдены следующие общие правила:

- При вызове функции должен быть задан параметр для идентификатора счетчика C_No. Вместо абсолютного номера счетчика (например, C12), можно задать переменную или константу типа INT или входной параметр типа COUNTER.
- Должен быть задан хотя бы один из параметров CU (прямой счет) или CD (обратный счет).
- Параметры PV (начальная величина) и S (установка) можно пропускать совместно.
- Функцией всегда возвращается результирующая величина в формате BCD.

Пример

```
FUNCTION_BLOCK FB1
VAR
  CurrVal, binVal: word;
  actFlag: bool;
END_VAR

BEGIN
CurrVal      :=S_CD (C_NO: C10, CD:=TRUE, S:=TRUE, PV:=100,  R:=FALSE,
CV:=binVal,Q:=actFlag);
CurrVal      :=S_CU (C_NO: C11, CU:=M0.0, S:=M0.1, PV:=16#110,
     R:=M0.2, CV:=binVal,Q:=actFlag);
CurrVal      :=S_CUD(C_NO: C12, CD:=I0.0, CU:=I0.1, S:=I0.2              &I0.3,
PV:=120, R:=FALSE, CV:=binVal,Q:=actFlag);
CurrVal      :=S_CD (C_NO: C10, CD:=FALSE, S:=FALSE, PV:=100,
       R:=TRUE, CV:=binVal,Q:=actFlag);
END_FUNCTION_BLOCK
```

12.1.4 Ввод и вычисление величины счетчика

Для установки начальной величины счета или расчетов с результатом функции необходимо внутреннее представление величины счета. Величина счета принадлежит типу WORD, при этом биты с 0 по 11 содержат величину счета в коде BCD, а биты с 12 по 15 не используются.

При установке счетчика заданная Вами величина записывается в счетчик. Величины имеют диапазон от 0 до 999. Вы можете изменять содержимое счетчика в пределах этого диапазона, вызовом операций счета в прямом/обратном порядке (S_CUD), счета в прямом порядке (S_CU) и счета в обратном порядке (S_CD).

Формат

Рисунок ниже показывает расположение битов в величине счета.

Ввод

- Целая десятеричная величина: Например, 295, предполагается, что эта величина соответствует формату BCD (если BCD код 127, интерпретировать как обычный двоичный, получим 295).
- В формате BCD (ввод шестнадцатеричной константы): например 16#127

Вычисление

- Как результат функции (типа WORD): в формате BCD
- Как выходной параметр CV (типа WORD): в двоичном коде

12.1.5 Прямой счет (S_CU)

При использовании функции прямого счета (S_CU), исполняются операции только возрастающего счетчика. В таблице показано, как работает счетчик.

Операция	**Пояснение**
Прямой счет	Величина счета возрастает на "1" если состояние сигнала на входе CU изменяется с "0" на "1" и, при этом, величина счета меньше 999.
Установка счетчика	Когда состояние сигнала на входе **S** меняется с "0" на "1", счетчик устанавливается в положение входной величины **PV**. Такое изменение сигнала необходимо для установки счетчика.
Сброс	Счетчик сбрасывается, когда задан вход **R** = 1. Сброс счетчика устанавливает величину счета равную "0".
Запрос счетчика	Состояние сигнала запроса на выходе Q возвращает "1" в случае, если величина счета больше "0". Запрос возвращает "0", если величина счета равна "0".

12.1.6 Обратный счет (S_CD)

При использовании функции обратного счета (S_CD), исполняются операции только убывающего счетчика. В таблице показано, как работает счетчик.

Операция	**Пояснение**
Обратный счет	Величина счета убывает на "1" если состояние сигнала на входе **CD** изменяется с "0" на "1" и, при этом, величина счета больше "0".
Установка счетчика	Когда состояние сигнала на входе **S** меняется с "0" на "1", счетчик устанавливается в положение входной величины **PV**. Такое изменение сигнала необходимо для установки счетчика.
Сброс	Счетчик сбрасывается, когда задан вход **R** = 1. Сброс счетчика устанавливает величину счета равную "0".
Запрос счетчика	Состояние сигнала запроса на выходе Q возвращает "1" в случае если величина счета больше "0". Запрос возвращает "0", если величина счета равна "0".

12.1.7 Прямой/обратный счет (S_CUD)

При использовании функции прямого/обратного счета (S_CUD), Вы можете исполнять функции как прямого, так и обратного счетчика. Если импульсы прямого и обратного счета приходят одновременно, исполняются оба вида операций. Величина счета остается неизменной. В таблице показано, как работает счетчик.

Операция	**Пояснение**
Прямой счет	Величина счета возрастает на "1" если состояние сигнала на входе CU изменяется с "0" на "1" и, при этом, величина счета меньше 999.
Обратный счет	Величина счета убывает на "1" если состояние сигнала на входе **CD** изменяется с "0" на "1" и, при этом, величина счета больше "0".
Установка счетчика	Когда состояние сигнала на входе **S** меняется с "0" на "1", счетчик устанавливается в положение входной величины **PV**. Такое изменение сигнала необходимо для установки счетчика.
Сброс	Счетчик сбрасывается, когда задан вход **R** = 1. Сброс счетчика устанавливает величину счета равную "0".
Запрос счетчика	Состояние сигнала запроса на выходе Q возвращает "1" в случае, если величина счета больше "0". Запрос возвращает "0", если величина счета равна "0".

12.1.8 Пример функций счетчиков

Присвоение параметров

Приведенная ниже таблица иллюстрирует присвоение параметров для функции S_CD.

Параметр	Описание
C_NO	MYCOUNTER:
CD	INPUT I0.0
S	SET
PV	INITIALVALUE 16#0089
R	RESET
Q	Q0.7
CV	BIN_VALUE

Пример

```
FUNCTION_BLOCK COUNT
VAR_INPUT
  MYCOUNTER       : COUNTER ;
END_VAR
VAR_OUTPUT
  RESULT    : INT ;
END_VAR
VAR
  SET           : BOOL ;
  RESET  : BOOL ;
  BCD_VALUE       : WORD ;// Величина счета в BCD коде
  BIN_VALUE       : WORD ;// Двоичная величина счета
  INITIALVALUE: WORD ;
END_VAR
BEGIN
  Q0.0          := 1 ;
  SET           := I0.2 ;
  RESET           := I0.3 ;
  INITIALVALUE    := 16#0089 ;
//Счет вниз
    BCD_VALUE  := S_CD (C_NO  := MYCOUNTER,
       CD     := I0.0 ,
       S          := SET ,
       PV     := INITIALVALUE,
       R          := RESET ,
       CV     := BIN_VALUE ,
       Q          := Q0.7) ;
//Продолжение обработки выходного параметра
   RESULT        := WORD_TO_INT (BIN_VALUE) ;
   QW4           := BCD_VALUE ;
END_FUNCTION_BLOCK
```

12.2 Таймеры

12.2.1 Функции таймеров

Таймеры – это функциональные элементы Вашей программы, которые исполняют функции, зависящие от времени и выполняют их мониторинг. В STEP 7 представлены следующие стандартные функции таймеров, которые Вы можете использовать в Вашей SCL программе.

Функция таймера	Пояснение
S_PULSE	Запускает таймер как импульсный таймер
S_PEXT	Запускает таймер как таймер удлиненного импульса
S_ODT	Запускает таймер как задержку включения
S_ODTS	Запускает таймер как задержку включения, с памятью
S_OFFDT	Запускает таймер как задержку выключения

12.2.2 Вызов функций таймеров

Функции таймеров вызываются как обычные функции. Поэтому идентификатор функции может использоваться вместо адреса в выражениях во всех случаях, когда тип возвращаемой величины соответствует типу замещаемого адреса.

Вызывающему блоку возвращается величина функции (возвращаемое значение), которая является текущим временем в формате S5TIME.

Абсолютный или динамический вызов

При вызове Вы можете задать абсолютную величину типа TIMER, как номер таймера (например, T_NO:=T10). Подобные величины не могут изменяться в процессе исполнения программы.

Вместо абсолютного номера счетчика, Вы можете так же задать переменную или константу типа INT. Преимущество этого метода состоит в том, что при этом вызов счетчика становится динамическим, и ему могут передаваться различные значения при каждом новом вызове.

При динамическом вызове Вы так же можете использовать переменную типа TIMER.

Примеры

```
//Пример абсолютного вызова:
S_ODT (T_NO:=T10,
      S:=TRUE,
      TV:=T#1s,
      R:=FALSE,
      BI:=biVal,
      Q:=actFlag);

//Пример динамического вызова: В каждой итерации
//цикла FOR, вызывается различный таймер:
FUNCTION_BLOCK TIME
VAR_INPUT
  MY_TIMER: ARRAY [1..4] of STRUCT
           T_NO: INT;
           TV  : WORD;
          END_STRUCT;
.
.
END_VAR
.
.
FOR I:= 1 TO 4 DO
  S_ODT(T_NO:=MY_TIMER[I].T_NO, S:=true,
    TV:= MY_TIMER[I].TV);
END_FOR;

//Пример динамического вызова, использующего переменную
//типа TIMER:
FUNCTION_BLOCK TIMER
VAR_INPUT
  mytimer:TIMER;
END_VAR
.
.
CurrTime:=S_ODT (T_NO:=mytimer,.....);
```

Примечание

Имена функций совпадают в немецкой и английской мнемонике.

12.2.3 Задание параметров для функций таймеров

В следующей таблице дан обзор параметров для функций таймеров:

Параметр	Тип	Описание
T_NO	TIMER INTEGER	Идентификационный номер таймера; диапазон зависит от CPU
S	BOOL	Начальный вход
TV	S5TIME	Инициализация величины таймера (в формате BCD)
R	BOOL	Сброс входа
Q	BOOL	Состояние таймера
BI	WORD	Оставшееся время (двоичный код)

Правила

Так как величина параметра хранится глобально (в таймерной памяти CPU), необязательно определять ее в любых ситуациях. При задании величин параметров должны быть соблюдены следующие общие правила:

- При вызове функции должен быть задан параметр для идентификатора таймера T_No. Вместо абсолютного номера таймера (например, T12), можно задать переменную или константу типа INT или входной параметр типа TIMER.
- Параметры PV (начальная величина) и S (установка) можно пропускать совместно.
- Функция всегда возвращает оставшееся время в формате S5TIME.

Пример

```
FUNCTION_BLOCK FB2
VAR
  CurrTime   : S5time;
  BiVal                : word;
  ActFlag    : bool;
END_VAR

BEGIN
CurrTime    :=S_ODT   (T_NO:= T10, S:=TRUE, TV:=T#1s, R:=FALSE,
        BI:=biVal,Q:=actFlag);
CurrTime    :=S_ODTS (T_NO:= T11, S:=M0.0, TV:= T#1s, R:=M0.1,
        BI:=biVal,Q:=actFlag);
CurrTime    :=S_OFFDT(T_NO:= T12, S:=I0.1 & actFlag, TV:= T#1s,
        R:=FALSE, BI:=biVal,Q:=actFlag);
CurrTime    :=S_PEXT (T_NO:= T13, S:=TRUE, TV:= T#1s, R:=I0.0,
        BI:=biVal,Q:=actFlag);
CurrTime    :=S_PULSE(T_NO:= T14, S:=TRUE, TV:= T#1s, R:=FALSE,
        BI:=biVal,Q:=actFlag);
END_FUNCTION_BLOCK
```

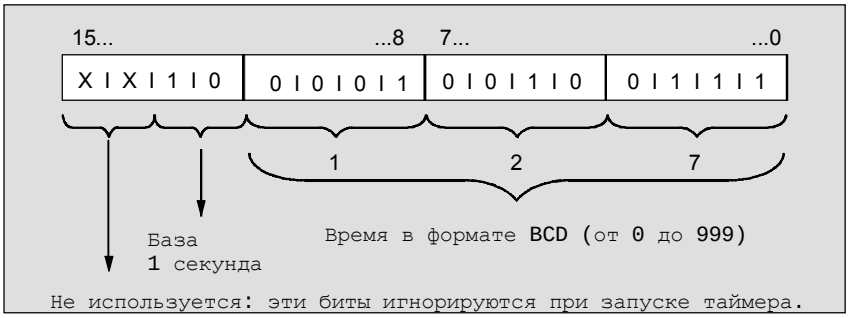
12.2.4 Ввод и вычисление величины времени

Для ввода начальной величины и вычисления значения функции в коде BCD, необходимо внутреннее представление величины времени. Величина времени принадлежит типу WORD, при этом биты с 0 по 11 содержат величину времени в формате BCD, а биты 12 и 13 единицу измерения времени. Биты 14 и 15 не используются.

Изменение времени уменьшает величину на таймере на 1 единицу в течение 1 интервала, определенного единицей измерения времени. Величина на таймере уменьшается до тех пор, пока не достигнет "0". Возможный диапазон величин времени от 0 до 9990 секунд.

Формат

Рисунок ниже показывает расположение битов в величине времени.



Ввод

Вы можете задать предопределенные величины времени в следующих форматах:

- В смешанном формате времени
- В десятеричном формате времени

Величина единицы измерения времени в обоих случаях выбирается автоматически и величина на счетчике округляется в меньшую сторону до следующего числа в этих единицах измерения времени.

Вычисление

Вы можете получить результат в следующих форматах:

- Как результат функции (типа S5TIME): в формате BCD
- Как выходной параметр (время типа WORD без единицы измерения): в двоичном коде

Единицы измерения величин времени

Для ввода и обработки величины времени, Вам потребуются единицы измерения (12 и 13 биты слова на таймере). Единица измерения времени определяет интервал, за который величина времени уменьшается на одну единицу (см. таблицу). Наименьшая единица измерения равна 10 мс; наибольшая 10 с.

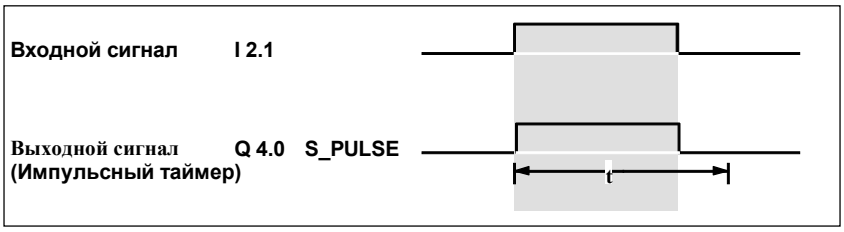
Единица измерения времени	Двоичный код единицы измерения времени
10 мс	00
100 мс	01
1 с	10
10 с	11

Примечание

Так как величины времени хранятся только как набор интервалов, величины, не вписывающиеся в данный диапазон, обрезаются. Величины со слишком высокой точностью округляются в меньшую сторону, таким образом, достигаются требуемый диапазон, но не требуемая точность.

12.2.5 Запуск таймера как импульсного таймера (S_PULSE)

Величина времени на таймере равна максимальному интервалу времени, в течение которого сигнал на выходе будет равен "1". Если, в процессе работы таймера, на входе возникает нулевой сигнал, то таймер сбрасывается на "0". Это означает преждевременное завершение работы таймера.



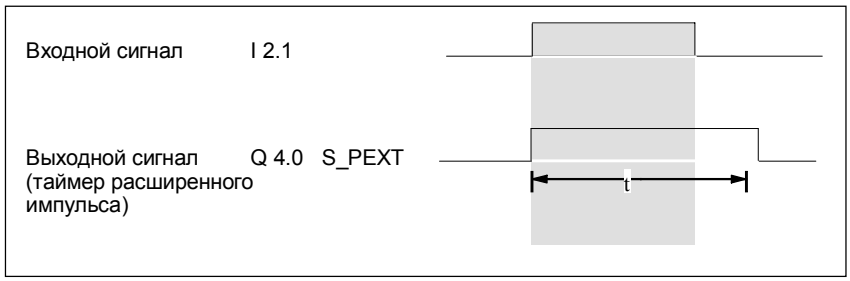
Как функционирует таймер

Приведенная таблица показывает, как работает функция "импульсный таймер":

Операция	Пояснение
Запуск	Операция "импульсный таймер" запускает таймер, когда состояние сигнала на начальном входе (**S**) изменяется с "0" на "1". Для включения таймера изменение сигнала необходимо.
Определение времени работы	Если на входе **S** сохраняется сигнал “1”, таймер работает, начиная с величины, заданной на входе **TV,** до тех пор, пока установленное время не истечет.
Прерывание работы	Если сигнал на входе **S** изменяется с "1" на "0" до истечения запрограммированного времени, таймер останавливается.
Сброс	Таймер сбрасывается в случае если сигнал на входе сброса (**R**) изменяется с "0" на "1" в процессе работы таймера. При этом как величина времени на таймере, так и единица измерения времени сбрасываются на ноль. Эффект сигнала "1" на входе **R** отсутствует, если таймер не работает.
Запрос состояния сигнала	Пока таймер работает, запрос состояния сигнала с выхода **Q** возвращает "1". Если таймер прерывается, запрос состояния сигнала с выхода **Q** возвращает "0".
Запрос текущей величины на таймере	Текущая величина оставшегося времени может быть запрошена с выхода BI и используя функциональную величину S_PULSE.

12.2.6 Запуск таймера как таймера расширенного импульса (S_PEXT)

Выходной сигнал остается равным "1" в течение заданного времени (t) вне зависимости от того, сколько времени входной сигнал равен "1". Повторение стартового импульса начинает заново отсчет времени, при этом выходной импульс расширяется (повторный запуск).



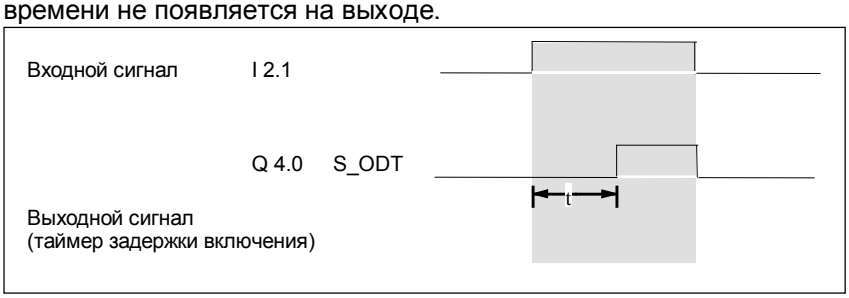
Как функционирует таймер

Приведенная таблица показывает, как работает функция "таймер расширенного импульса":

Операция	**Пояснение**
Запуск	Операция "расширенный импульсный таймер" (S_PEXT) запускает таймер, когда на начальном входе (**S**) состояние сигнала изменяется с "0" на "1". Для включения таймера необходимо изменение сигнала.
Перезапуск счетчика времени	Если состояние сигнала на воде **S** снова переходит из "0" в "1" в процессе работы таймера, таймер перезапускает текущую величину времени.
Определение времени работы	Таймер работает, начиная с величины на входе **TV,** до тех пор пока запрограммированное время не истечет.
Сброс	Таймер сбрасывается в случае, если сигнал на входе сброса (**R**) изменяется с "0" на "1" в процессе работы таймера. При этом как величина времени на таймере, так и единица измерения времени сбрасываются на ноль. Эффект сигнала "1" на входе **R** отсутствует, если таймер не работает.
Запрос состояния сигнала	До тех пор пока таймер работает, запрос состояния сигнала с выхода **Q** возвращает "1", вне зависимости от длины входного сигнала.
Запрос текущей величины на таймере	Текущая величина на таймере может быть запрошена с выхода BI и используя функциональную величину S_PEXT.

12.2.7 Запуск таймера как таймера задержки включения (S_ODT)

Выходной сигнал меняется с "0" на "1" только в случае, если запрограммированное время истекло, а входной сигнал равен "1". Это означает, что выход активируется после задержки. Входной сигнал, остающийся активным в течение времени, короче запрограммированного времени не появляется на выходе.



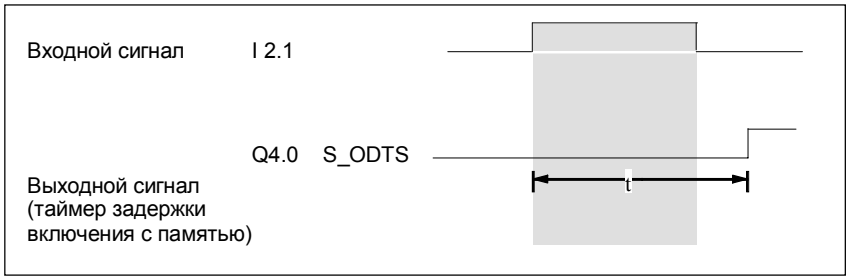
Как функционирует таймер

Приведенная таблица показывает, как работает "реле времени":

Операция	Пояснение
Запуск	Операция задержка включения запускает таймер, когда на стартовом входе (**S**) состояние сигнала изменяется с "0" на "1". Для запуска таймера необходимо изменение сигнала.
Остановка	Если в процессе работы таймера сигнал на входе **S** изменяется с "1" на "0", таймер останавливается.
Определение времени работы	Таймер работает, используя величину на входе **TV,** пока на входе **S** сохраняется сигнал “1”.
Сброс	Таймер сбрасывается в случае, если сигнал на входе сброса (**R**) изменяется с "0" на "1" в процессе работы таймера. При этом как величина времени на таймере, так и единица измерения времени сбрасываются на ноль. Если задано **R** = 1, то таймер сбрасывается даже в случае, если он не работает.
Запрос состояния сигнала	Запрос состояния сигнала с выхода **Q** возвращает "1", когда время истекло без возникновения ошибок, а на входе **S** сохраняется "1". Если таймер был остановлен, то запрос состояния сигнала всегда возвращает "0". Запрос состояния сигнала с выхода **Q** также возвращает "0" когда таймер не работает, а состояние сигнала на входе **S** все еще "1".
Запрос текущей величины на таймере	Текущая величина на таймере может быть запрошена с выхода BI и используя функциональную величину S_ODT.

12.2.8 Запуск таймера как таймера задержки включения с памятью (S_ODTS)

Выходной сигнал меняется с "0" на "1", когда истекает запрограммированное время, вне зависимости от того, сколько времени сигнал на входе остается равным "1".



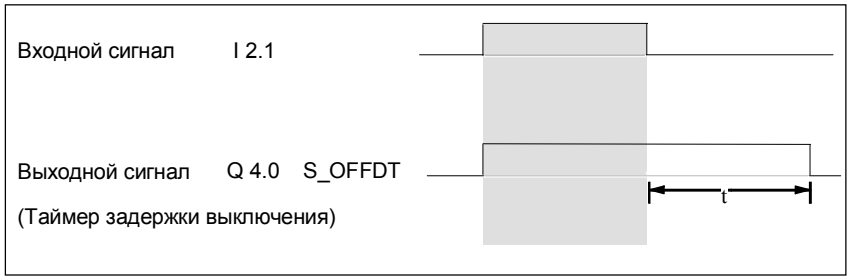
Как функционирует таймер

Приведенная таблица показывает, как работает "задержка включения с памятью":

Операция	**Пояснение**
Запуск	Операция " задержка включения с памятью" запускает таймер, когда на начальном входе (**S**) состояние сигнала изменяется с "0" на "1". Для включения таймера необходимо изменение сигнала.
Перезапуск таймера	Таймер перезапускает заданную величину времени, когда вход **S** изменяется с "0" на "1" в процессе работы таймера.
Определение времени работы	Таймер продолжает работать с заданной на входе **TV** начальной величиной**,** даже если состояние сигнала на входе **S** изменяется на "0" до истечения времени.
Сброс	Если вход сброса (**R**) изменяется с "0" на "1", таймер сбрасывается вне зависимости от состояния сигнала на входе **S**.
Запрос состояния сигнала	Сигнал на выходе **Q** равен "1", если таймер был запущен и отсчитал заданное время, независимо от сигнала на входе **S**.
Запрос текущей величины на таймере	Текущая величина на таймере может быть запрошена с выхода BI и используя функциональную величину S_ODTS.

12.2.9 Запуск таймера как таймера задержки выключения (S_OFFDT)

При изменении состояния сигнала на начальном входе S с "0" на "1", на выходе Q устанавливается "1". Если начальный вход изменяется с "1" на "0", таймер запускается. На выходе снова устанавливается "0" только после истечения времени. Поэтому выход деактивируется после заданной задержки.



Как функционирует таймер

Приведенная таблица показывает, как работает "таймер задержки выключения":

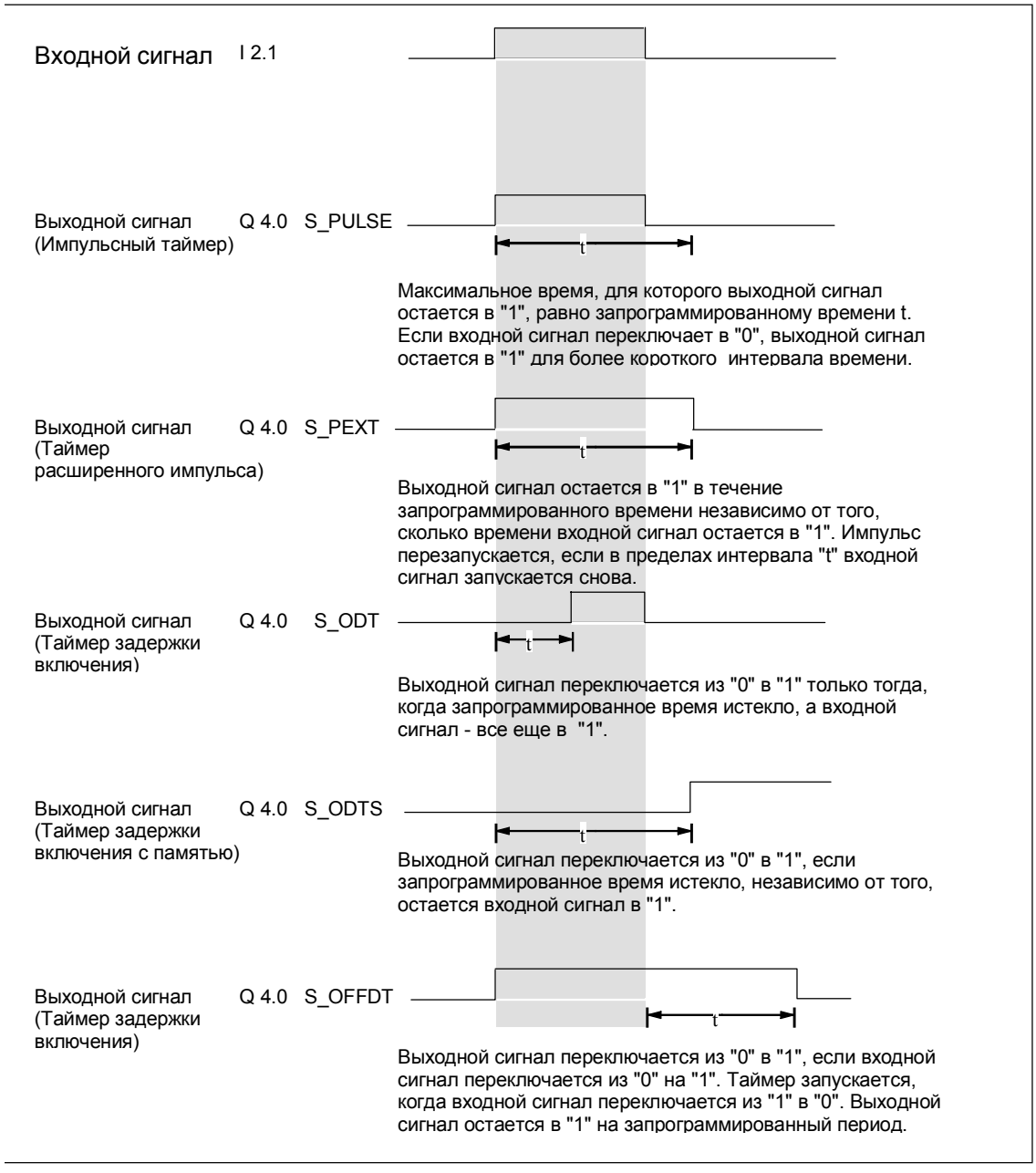
Операция	Пояснение
Запуск	Операция "таймер задержки выключения" запускает таймер, когда на начальном входе (**S**) состояние сигнала изменяется с "0" на "1". Для включения таймера необходимо изменение сигнала.
Перезапуск таймера	Таймер перезапускается, когда состояние сигнала на входе **S** снова изменяется с "1" на "0" (например, после сброса).
Определение времени работы	Таймер работает с величиной определенной на выходе **TV**.
Сброс	Таймер сбрасывается в случае, если сигнал на входе сброса (**R**) изменяется с "0" на "1".
Запрос состояния сигнала	Запрос состояния сигнала с выхода **Q** возвращает "1" если состояние сигнала на выходе **S** = 1, либо таймер работает.
Запрос текущей величины на таймере	Текущая величина на таймере может быть запрошена с выхода BI и используя функциональную величину S_OFFDT.

12.2.10 Пример функций таймеров

```
FUNCTION_BLOCK TIMER
VAR_INPUT
  mytime      : TIMER ;
END_VAR
VAR_OUTPUT
  result        : S5TIME ;
END_VAR
VAR
  set     : BOOL ;
  reset     : BOOL ;
  bcdvalue  : S5TIME ; //Время в BCD и база времени
  binvalue  : WORD ;   //Время в двоичном коде
  initialvalue : S5TIME ;
END_VAR
BEGIN
  Q0.0   := 1;
  set     := I0.0 ;
  reset       := I0.1;
  initialvalue := T#25S ;
  bcdvalue   := S_PEXT (T_NO  := mytime ,
              S    := set ,
              TV  := initialvalue ,
              R    := reset ,
              BI  := binvalue ,
              Q    := Q0.7) ;
//Продолжение обработки выходного параметра
  result := bcdvalue ;
//вывод на дисплей
  QW4 := binvalue ;
END_FUNCTION_BLOCK
```

12.2.11 Правильный выбор таймера

Следующий рисунок представляет обзор пяти различных функций таймеров, описанных в этом разделе. Этот обзор должен помочь Вам выбрать функцию таймера наилучшим образом отвечающую Вашей задаче.



13 Стандартные функции SCL

13.1 Функции преобразования типов данных

13.1.1 Преобразование типов данных

При выполнении логических операций с двумя адресами, Вы должны быть уверенны, что типы адресов совместимы. Если адреса имеют различный тип, то необходимо преобразование типов данных. В SCL возможны следующие преобразования типов данных:

- Неявное преобразование типов данных

  Типы данных сгруппированы по классам. Если адреса относятся к одному и тому же классу, SCL совершает неявное преобразование типов данных. Функции, используемые компилятором в этом случае, сгруппированы в класс "Функции преобразования Класса A".

- Явное преобразование типов данных

  В случае, если адреса не принадлежат к одному классу, Вы должны сами запустить функцию преобразования классов. Для реализации явного преобразования типов данных в SCL представлены стандартные функции, разбитые на следующие классы:

  - Функции преобразования Класса B
  - Функции округления

13.1.2 Неявное преобразование типов данных

13.1.2.1 Неявное преобразование типов данных

Внутри классов типов данных, определенных в таблице, компилятор совершает неявное преобразование типов данных в указанном порядке. Общий формат для двух адресов подбирается из расчета наименьшего типа данных с интервалом, включающим оба адреса. Например, общий формат для типов BYTE и INTEGER – это INTEGER.

Помните, что преобразование типов данных внутри класса ANY_BIT приводит к заполнению нулями старших разрядов.

Классы	Порядок преобразования
ANY_BIT	BOOL > BYTE > WORD > DWORD
ANY_NUM	INT > DINT > REAL

Пример неявного преобразования типов данных

```
VAR
  PID_CTRLLER_1 : BYTE ;
  PID_CTRLLER_2 : WORD ;
END_VAR
BEGIN
IF (PID_CTRLLER_1 <> PID_CTRLLER_2) THEN ...
(* В инструкции IF вверху PID_CTRLLER_1 неявно преобразован из типа BYTE в
тип WORD. *)
```

13.1.2.2 Функции преобразования класса A

В данной таблице приведены функции преобразования типов данных класса A. Эти функции неявно исполняются компилятором, однако, Вы можете их использовать и в явном виде. Результат всегда определен.

Имя функции	**Правило преобразования**
BOOL_TO_BYTE	Обнуляет старшие разряды
BOOL_TO_DWORD	Обнуляет старшие разряды
BOOL_TO_WORD	Обнуляет старшие разряды
BYTE_TO_DWORD	Обнуляет старшие разряды
BYTE_TO_WORD	Обнуляет старшие разряды
CHAR_TO_STRING	Преобразование в строку (длины 1), содержащую тот же символ.
DINT_TO_REAL	Преобразование в REAL в соответствии со стандартом IEEE. Величина может изменяться (из-за различной точности типа REAL).
INT_TO_DINT	В случае отрицательного параметра слово высшего порядка функциональной величины заполняется 16#FFFF, в противном случае оно заполняется нулями. Величина при этом не меняется.
INT_TO_REAL	Преобразование в REAL в соответствии со стандартом IEEE. Величина при этом не меняется.
WORD_TO_DWORD	Обнуляет старшие разряды

13.1.3 Стандартные функции для явного преобразования типов данных

Общее описание вызова функции можно найти в разделе "Вызов функций".

При вызове функций преобразования помните следующие правила:

- Входные параметры:
  Каждая функция преобразования типов данных имеет только один входной параметр с именем IN. Так как это функции одного параметра, то имя IN не надо определять.
- Функциональная величина
  Результат всегда является функциональной величиной.
- Имена функций
  Тип входного параметра и функциональной величины однозначно определяется именем функции, приведенном в обзоре классов A и B. Например, для функции BOOL_TO_BYTE, тип входного параметра BOOL, а тип функциональной величины BYTE.

13.1.3.1 Функции преобразования класса B

В данной таблице приведены функции преобразования типов данных класса B. Эти функции должны быть заданы явным образом. Кроме того, результат может быть неопределенным, если результирующий тип недостаточен для представления величины.

Вы можете сами контролировать такие ситуации, проверяя пределы типов, либо контроль будет осуществлять система, проставляя перед компиляцией значения "флага OK". В ситуациях, когда результат не определен, система устанавливает флаг OK в положение FALSE.

Имя функции	Правило преобразования	OK
BYTE_TO_BOOL	Копирует младший значащий разряд	Y
BYTE_TO_CHAR	Копирует разрядную строку	N
CHAR_TO_BYTE	Копирует разрядную строку	N
CHAR_TO_INT	Разрядная строка входного параметра записывается в младшие разряды функциональной величины, а старшие разряды заполняются нулями.	N
DATE_TO_DINT	Копирует разрядную строку	N
DINT_TO_DATE	Копирует разрядную строку	Y
DINT_TO_DWORD	Копирует разрядную строку	N
DINT_TO_INT	Копирует знаковый разряд. Величина входного параметра преобразуется к типу INT. Если величина менее -32_768 или более 32_767, флаг OK устанавливается в положение FALSE.	Y
DINT_TO_TIME	Копирует разрядную строку	N
DINT_TO_TOD	Копирует разрядную строку	Y
DWORD_TO_BOOL	Копирует младший значащий разряд	Y
DWORD_TO_BYTE	Копирует 8 младших значащих разрядов	Y

DWORD_TO_DINT	Копирует разрядную строку	N
DWORD_TO_REAL	Копирует разрядную строку	N
DWORD_TO_WORD	Копирует 16 младших значащих разрядов	Y
INT_TO_CHAR	Копирует разрядную строку	Y
INT_TO_WORD	Копирует разрядную строку	N
REAL_TO_DINT	Округляет величину IEEE REAL в DINT. Если величина менее -2_147_483_648 или более 2_147_483_647, флаг OK устанавливается в положение FALSE.	Y
REAL_TO_DWORD	Копирует разрядную строку	N
REAL_TO_INT	Округляет величину IEEE REAL в INT. Если величина менее -32_768 или более 32_767, флаг OK устанавливается в положение FALSE.	Y
STRING_TO_CHAR	Копирует первый символ строки. Если переменная типа STRING имеет длину не равную 1, флаг OK устанавливается в положение FALSE.	Y
TIME_TO_DINT	Копирует разрядную строку	N
TOD_TO_DINT	Копирует разрядную строку	N
WORD_TO_BOOL	Копирует младший значащий разряд	Y
WORD_TO_BYTE	Копирует 8 младших значащих разрядов	Y
WORD_TO_INT	Копирует разрядную строку	N
WORD_TO_BLOCK_DB	Разрядная сетка WORD воспринимается как номер блока данных	N
BLOCK_DB_TO_WORD	Номер блока данных воспринимается как разрядная сетка WORD	N

Примечание

Вы можете использовать и другие IEC функции преобразования типов. Информация о функциях содержится в руководстве "System Software for S7-300 and S7-400, System and Standard Functions" ("Системное программное обеспечение для S7-300 и S7-400, Системные и стандартные функции").

13.1.3.2 Функции округления

Функции округления чисел также относятся к функциям преобразования типов данных. Данная таблица показывает имена, типы данных (для входных параметров и функциональных величин) и задачи этих функций:

Имя функции	Тип входного параметра	Тип выходной величины	Задача
ROUND	REAL	DINT	Округление (возвращает число типа DINT) В соответствии с DIN EN 61131-3, функция всегда округляет 0,5 в сторону ближайшего четного числа; другими словами, 1.5 возвращает 2 и 2.5 также возвращает 2.
TRUNC	REAL	DINT	Truncates (возвращает число типа DINT)

Примечание

Вы так же можете использовать и другие IEC функции преобразования типов. Информация о функциях содержится в руководстве "System Software for S7-300 and S7-400, System and Standard Functions"("Системное программное обеспечение для S7-300 и S7-400, Системные и стандартные функции»).

Пример

```
// Округление вниз (результат: 3)
   ROUND (3.14) ;

// Округление вверх (результат: 4)
   ROUND (3.56) ;

// Обрезание (результат: 3)
   TRUNC (3.14) ;

// Обрезание (результат: 3)
   TRUNC (3.56) ;
```

13.1.3.3 Примеры преобразования со стандартными функциями

В приведенном примере необходимо явное преобразование, так как результирующий тип имеет более низкий порядок, чем исходный тип.

```
FUNCTION_BLOCK FB10
VAR
  SWITCH : INT;
  CTRLLER : DINT;
END_VAR

(* INT имеет меньший порядок чем DINT *)
SWITCH := DINT_TO_INT (CTRLLER) ;
// . . .
END_FUNCTION_BLOCK
```

В приведенном примере необходимо явное преобразование, так как тип REAL не может использоваться в арифметических выражениях с операцией MOD.

```
FUNCTION_BLOCK FB20
VAR
  SWITCH   : REAL
  INTVALUE          : INT := 17;
  CONV2     : INT ;
END_VAR

(* MOD может использоваться только с данными типа INT или DINT *)
CONV2 := INTVALUE MOD REAL_TO_INT (SWITCH);
// . . .
END_FUNCTION_BLOCK
```

В следующем примере преобразование необходимо, так как приведенный тип не может использоваться в логических операциях. Операция NOT может использоваться только с типами BOOL, BYTE, WORD или DWORD.

```
FUNCTION_BLOCK FB30
VAR
  INTVALUE : INT := 17;
  CONV1     : WORD ;
END_VAR

(* NOT не может применяться к данным типа INT *)
CONV1 := NOT INT_TO_WORD(INTVALUE);
// . . .
END_FUNCTION_BLOCK
```

Следующий пример иллюстрирует преобразование типов данных для внешних входов/выходов.

```
FUNCTION_BLOCK FB40
VAR
  Radius_in    : WORD ;
  Radius       : INT;
END_VAR

  Radius_in    := %IB0;
  Radius       := WORD_TO_INT (radius_in);
(* Преобразование при изменении классов типов. Переменная полученная со входа
преобразована для дальнейшей обработки.*)

  Radius   := Radius (area:= circledata.area)
  %QB0     :=WORD_TO_BYTE (INT_TO_WORD(RADIUS));
(* Радиус вычислен из площади и является целым. Для выдачи на выход, результат
сначала преобразуется в другой класс данных (INT_TO_WORD), а затем в данные
меньшего диапазона (WORD_TO_BYTE).*)
// . . .
END_FUNCTION_BLOCK
```

13.2 Числовые стандартные функции

13.2.1 Общие арифметические стандартные функции

Ниже представлены функции для подсчета модуля величины, квадрата и квадратного корня величины.

Тип данных ANY_NUM означает INT, DINT или REAL. Обратите внимание на то, что входные параметры типа INT или DINT преобразуются внутри функции в переменную типа REAL, в случае, если возвращаемая функцией величина имеет тип REAL.

Имя функции	Тип входного параметра	Тип выходной величины	Описание
ABS	ANY_NUM	ANY_NUM	Модуль
SQR	ANY_NUM	REAL	Возведение в квадрат
SQRT	ANY_NUM	REAL	Квадратный корень

Примечание

Вы так же можете использовать и другие IEC функции преобразования типов. Информация о функциях содержится в руководстве "System Software for S7-300 and S7-400, System and Standard Functions" ("Системное программное обеспечение для S7-300 и S7-400, Системные и стандартные функции»).

13.2.2 Логарифмические функции

Далее представлены функции для расчета показательных величин и логарифмов.

Тип данных ANY_NUM означает INT, DINT или REAL. Обратите внимание на то, что входные параметры типа INT или DINT преобразуются внутри функции в переменную типа REAL, в случае, если функция имеет тип REAL.

Имя функции	Тип входного параметра	Тип выходной величины	Описание
EXP	ANY_NUM	REAL	e в степени IN
EXPD	ANY_NUM	REAL	10 в степени IN
LN	ANY_NUM	REAL	Натуральный логарифм
LOG	ANY_NUM	REAL	Логарифм по произвольному основанию

Примечание

Вы так же можете использовать и другие IEC функции преобразования типов. Информация о функциях содержится в руководстве "System Software for S7-300 and S7-400, System and Standard Functions" ("Системное программное обеспечение для S7-300 и S7-400, Системные и стандартные функции»).

13.2.3 Тригонометрические функции

Тригонометрические функции, представленные в таблице, используют в качестве аргумента угол, измеряемый в радианах.

Тип данных ANY_NUM означает INT, DINT или REAL. Обратите внимание на то, что входные параметры типа INT или DINT преобразуются внутри функции в переменную типа REAL, в случае, если функция имеет тип REAL.

Имя функции	Тип входного параметра	Тип выходной величины	Описание
ACOS	ANY_NUM	REAL	Арккосинус
ASIN	ANY_NUM	REAL	Арксинус
ATAN	ANY_NUM	REAL	Арктангенс
COS	ANY_NUM	REAL	Косинус
SIN	ANY_NUM	REAL	Синус
TAN	ANY_NUM	REAL	Тангенс

Примечание

Вы так же можете использовать и другие IEC функции преобразования типов. Информация о функциях содержится в руководстве "System Software for S7-300 and S7-400, System and Standard Functions"("Системное программное обеспечение для S7-300 и S7-400, Системные и стандартные функции»).

13.2.4 Примеры числовых стандартных функций

Вызов	РЕЗУЛЬТАТ
RESULT := ABS (-5) ;	//5
RESULT := SQRT (81.0);	//9
RESULT := SQR (23);	//529
RESULT := EXP (4.1);	//60.340 ...
RESULT := EXPD (3);	//1_000
RESULT := LN (2.718 281) ;	//1
RESULT := LOG (245);	//2.389_166 ...
PI := 3. 141 592 ; RESULT := SIN (PI / 6) ;	//0.5
RESULT := ACOS (0.5);	//1.047_197 (=PI / 3)

13.3 Стандартные функции битовых переменных

Каждая стандартная функция битовых переменных имеет два входных параметра, названных IN и N. Результат всегда возвращается через величину функции. В следующей таблице представлен обзор типов обоих входных параметров и функциональной величины. Описание входных параметров:

- Входной параметр IN: буфер, в котором совершается операция над битовой переменной. Тип этого параметра определяет тип функциональной величины.
- Входной параметр N: число циклов циклических буферных функций ROL и ROR, либо число позиций, на которые необходимо произвести сдвиг в случае функций SHL и SHR.

В таблице показаны возможные стандартные функции битовых переменных.

Имя функции	**Тип входного параметра IN**	**Тип входного параметра N**	**Тип выходной величины**	**Задача**
ROL	BOOL BYTE WORD DWORD	INT INT INT INT	BOOL BYTE WORD DWORD	Величина параметра IN циклически сдвигается влево на количество разрядов, определенных параметром N.
ROR	BOOL BYTE WORD DWORD	INT INT INT INT	BOOL BYTE WORD DWORD	Величина параметра IN циклически сдвигается вправо на количество разрядов, определенных параметром N.
SHL	BOOL BYTE WORD DWORD	INT INT INT INT	BOOL BYTE WORD DWORD	Величина параметра IN сдвигается на столько разрядов влево и столько разрядов справа заменяются нулями, сколько задано параметром N.
SHR	BOOL BYTE WORD DWORD	INT INT INT INT	BOOL BYTE WORD DWORD	Величина параметра IN сдвигается на столько разрядов вправо и столько разрядов слева заменяются нулями, сколько задано параметром N.

Примечание

Вы так же можете использовать и другие IEC функции преобразования типов. Информация о функциях содержится в руководстве "System Software for S7-300 and S7-400, System and Standard Functions"("Системное программное обеспечение для S7-300 и S7-400, Системные и стандартные функции»).

13.3.1 Примеры стандартных функций битовых сток

Вызов	**Результат**
RESULT := ROL (IN:=BYTE#2#1101_0011, N:=5);	//2#0111_1010 //(= 122 десятичное)
RESULT := ROR (IN:=BYTE#2#1101_0011, N:=2);	//2#1111_0100 //(= 244 десятичное)
RESULT := SHL (IN:=BYTE#2#1101_0011, N:=3);	//2#1001_1000 //(= 152 десятичное)
RESULT := SHR (IN:=BYTE#2#1101_0011, N:=2);	//2#0011_0100 //(= 52 десятичное)

13.4 Функции для обработки символьных строк

13.4.1 Функции для работы со строками

LEN

Функция LEN (FC21) возвращает текущую длину строки (число символов в строке). Пустая строка ('') имеет нулевую длину. Функция не выводит сообщений об ошибках.
Пример LEN (S:= 'XYZ')

Параметр	Объявление	Тип	Область памяти	Описание
S	INPUT	STRING	D, L	Входная переменная в формате STRING
Возвращаемая величина		INT	I, Q, M, D, L	Текущее число символов

CONCAT

Функция CONCAT (FC2) комбинирует две переменных типа STRING в одну строку. Если результирующая строка длиннее, чем переменная выходного параметра, результирующая строка ограничивается максимальной длиной. Ошибки могут быть запрошены через флаг OK.
Пример CONCAT (IN1:= 'Valve', IN2:= ' open')

Параметр	Объявление	Тип	Область памяти	Описание
IN1	INPUT	STRING	D, L	Входная переменная в формате STRING
IN2	INPUT	STRING	D, L	Входная переменная в формате STRING
Возвращаемая величина		STRING	D, L	Результирующая строка

LEFT или RIGHT

Функции LEFT или RIGHT (FC20 и FC32) возвращают первые или последние L символов в строке. Если L больше, чем текущая длина переменной типа STRING, возвращается строка целиком. Если L = 0, возвращается пустая строка. Если L отрицательно, возвращается пустая строка и флаг OK устанавливается в положение "0".
Пример LEFT (IN:= 'Valve', L:= 4)

Параметр	Объявление	Тип	Область памяти	Описание
IN	INPUT	STRING	D, L	Входная переменная в формате STRING
L	INPUT	INT	I, Q, M, D, L, const.	Длина строки слева
Возвращаемая величина		STRING	D, L	Выходная переменная в формате STRING

MID

Функция MID (FC26) возвращает часть строки. L – это длина строки которая будет считана, P – это позиция первого считываемого символа.
Если сумма L и (P-1) длиннее, чем текущая длина переменной типа STRING, возвращается часть строки, начинающаяся с P-го символа и до конца входной величины. В любой другой ситуации (P лежит за пределами текущей длины строки, P и/или L равны нулю или отрицательны), возвращается пустая строка, и флаг OK устанавливается в положение "0".
Пример MID (IN:= 'Temperature', L:= 2, P:= 3)

Параметр	Объявление	Тип	Область памяти	Описание
IN	INPUT	STRING	D, L	Входная переменная в формате STRING
L	INPUT	INT	I, Q, M, D, L, const.	Длина части строки
P	INPUT	INT	I, Q, M, D, L, const.	Положение первого символа
Возвращаемая величина		STRING	D, L	Выходная переменная в формате STRING

INSERT

Функция INSERT (FC17) вставляет символьную строку параметра IN2 в строку параметра IN1 после символа, определенного переменной P. Если P равно нулю, то вторая строка вставляется перед первой строкой. Если P больше чем текущая длина первой строки, вторая строка приписывается к первой. Если P отрицательно, возвращается пустая строка и флаг OK устанавливается в положение "0". Флаг OK находится в положении "0" всегда, когда выходная строка длиннее, чем переменная, определенная выходным параметром; в этом случае длина результирующей строки ограничена максимальной возможной длиной.
Пример INSERT (IN1:= 'Participant arrived', IN2:='Miller':= 2, P:= 11)

Параметр	**Объявление**	**Тип**	**Область памяти**	**Описание**
IN1	INPUT	STRING	D, L	Переменная типа STRING, в которую будет производиться вставка
IN2	INPUT	STRING	D, L	Вставляемая переменная типа STRING
P	INPUT	INT	I, Q, M, D, L, const.	Позиция вставки
Возвращаемая величина		STRING	D, L	Результирующая строка

DELETE

Функция DELETE (FC 4) стирает L символов в строке, начиная с символа, определенного переменной P (включая). Если L и/или P равны нулю, или если P больше чем текущая длина входной строки, возвращается входная строка. Если сумма L и P больше чем длина входной строки, то стирается вся строка до конца. Если L и/или P отрицательны, возвращается пустая строка, и флаг OK устанавливается в положение "0".
Пример: DELETE (IN:= 'Temperature ok', L:= 6, P:= 5)

Параметр	**Объявление**	**Тип**	**Область памяти**	**Описание**
IN	INPUT	STRING	D, L	Переменная типа STRING, из которой будут стираться символы
L	INPUT	INT	I, Q, M, D, L, const.	Число стираемых символов
P	INPUT	INT	I, Q, M, D, L, const.	Положение первого стираемого символа
Возвращаемая величина		STRING	D, L	Результирующая строка

REPLACE

Функция REPLACE (FC31) заменяет L символов первой строки (IN1), начиная с символа, определенного переменной P (включительно) второй строкой (IN2). Если L равно нулю, возвращается первая строка. Если P равно нулю или единице символы заменяются, начиная с первого (включительно). Если P находится за пределами первой строки, вторая строка приписывается к первой. Если L или/и P отрицательны, возвращается пустая строка, и флаг OK устанавливается в положение "0". Флаг OK находится в положении "0" всегда, когда выходная строка длиннее чем переменная определенная выходным параметром; в этом случае длина результирующей строки ограничена максимальной возможной длиной.
Пример REPLACE (IN1:= 'Temperature', IN2:= ' high' L:= 6, P:= 5)

Параметр	Объявление	Тип	Область памяти	Описание
IN1	INPUT	STRING	D, L	Переменная типа STRING, символы которой будут заменены
IN2	INPUT	STRING	D, L	Вставляемая переменная типа STRING
L	INPUT	INT	I, Q, M, D, L, const.	Число заменяемых символов
P	INPUT	INT	I, Q, M, D, L, const.	Положение первого заменяемого символа
Возвращаемая величина		STRING	D, L	Результирующая строка

FIND

Функция FIND (FC11) возвращает положение второй строки (IN2) внутри первой строки (IN1). Поиск начинается слева; выводится результат первого обнаружения строки. Если вторая строка не повторяется внутри первой, возвращается ноль. Функция не выводит сообщений об ошибках.
Пример FIND (IN1:= 'Processingstation', IN2:='station')

Параметр	Объявление	Тип	Область памяти	Описание
IN1	INPUT	STRING	D, L	Переменная типа STRING, в которой осуществляется поиск
IN2	INPUT	STRING	D, L	Искомая величина типа STRING
Возвращаемая величина		INT	I, Q, M, D, L	Положение найденной строки

13.4.2 Функции сравнения строк

В SCL Вы можете сравнивать строки друг с другом, используя стандартные операции сравнения ==, <>, <, >, <= и >=. Компилятор включает требуемый вызов автоматически. Следующие функции перечислены для создания цельной картины.

EQ_STRNG и NE_STRNG

Функции EQ_STRNG (FC10) и NE_STRNG (FC29) сравнивают содержимое двух переменных в формате STRING, проверяя эквивалентность (FC10) или неэквивалентность (FC29) и возвращают результат сравнения. Если строка параметра S1 эквивалентна (неэквивалентна) строке параметра S2, возвращается "1". Функция не выводит сообщений об ошибках.

Параметр	Объявление	Тип	Область памяти	Описание
S1	INPUT	STRING	D, L	Входная переменная в формате STRING
S2	INPUT	STRING	D, L	Входная переменная в формате STRING
Возвращаемая величина		BOOL	I, Q, M, D, L	Результат сравнения

GE_STRNG и LE_STRNG

Функции GE_STRNG (FC13) и LE_STRNG (FC19) сравнивают содержимое двух переменных в формате STRING, проверяя соотношение больше (меньше) или равно, и возвращают результат сравнения. Если строка параметра S1 больше (меньше) или равна строке параметра S2, возвращается "1". Символы сравниваются, начиная слева, с использованием кодировки ASCII (например, 'a' больше чем 'A'). Результат сравнения вычисляется по первому различающемуся символу. Если левая часть более длинной строки идентична более короткой строке, более длинная строка считается большей. Функция не выводит сообщений об ошибках.

Параметр	Объявление	Тип	Область памяти	Описание
S1	INPUT	STRING	D, L	Входная переменная в формате STRING
S2	INPUT	STRING	D, L	Входная переменная в формате STRING
Возвращаемая величина		BOOL	I, Q, M, D, L	Результат сравнения

GT_STRNG и LT_STRNG

Функции GT_STRNG (FC15) и LT_STRNG (FC24) сравнивают величины двух переменных формата STRING, проверяя соотношение больше (меньше) и возвращают результат сравнения. Если строка параметра S1 больше (меньше) строки параметра S2, возвращается "1". Символы сравниваются, начиная слева, с использованием кодировки ASCII (например, 'a' больше чем 'A'). Результат сравнения вычисляется по первому различающемуся символу. Если левая часть более длинной строки идентична более короткой строке, более длинная строка считается большей. Функция не выводит сообщений об ошибках.

Параметр	Объявление	Тип	Область памяти	Описание
S1	INPUT	STRING	D, L	Входная переменная в формате STRING
S2	INPUT	STRING	D, L	Входная переменная в формате STRING
RET_VAL		BOOL	I, Q, M, D, L	Результат сравнения

13.4.3 Функции преобразования формата данных

I_STRNG и STRNG_I

Функции I_STRNG (FC16) и STRNG_I (FC38) преобразуют переменную в формате INT в символьную строку, или строку в переменную типа INT. Строка представляется со знаком. Если переменная, определенная в возвращаемой величине, слишком мала, то преобразования не совершается, а флаг OK устанавливается в положение "0".

Параметр	Объявление	Тип	Область памяти	Описание
I_STRNG				
I	INPUT	INT	I, Q, M, D, L, const.	Входная величина
Возвращаемая величина		STRING	D, L	Результирующая строка
STRNG_I				
S	INPUT	STRING	D, L	Входная строка
Возвращаемая величина		INT	I, Q, M, D, L	Результат

DI_STRNG и STRNG_DI

Функции DI_STRNG (FC5) и STRNG_DI (FC37) преобразуют переменную в формате DINT в символьную строку, или строку в переменную типа DINT. Строка представляется со знаком. Если переменная, определенная в возвращаемой величине слишком мала, то преобразование не выполняется, а флаг OK устанавливается в положение "0".

Параметр	Объявление	Тип	Область памяти	Описание
DI_STRNG				
I	INPUT	DINT	I, Q, M, D, L, const.	Входная величина
Возвращаемая величина		STRING	D, L	Результирующая строка
STRNG_DI				
S	INPUT	STRING	D, L	Входная строка
Возвращаемая величина		DINT	I, Q, M, D, L	Результат

R_STRNG и STRNG_R

Функции R_STRNG (FC30) и STRNG_R (FC39) преобразуют переменную в формате REAL в символьную строку, или строку в переменную типа REAL. Строка представляется первыми 14 символами в формате:
±v.nnnnnnnE±xx
Если переменная, определенная в возвращаемой величине слишком мала или не существует правильного числа с плавающей запятой для параметра IN parameter, то преобразование не выполняется, а флаг OK устанавливается в положение "0".

Параметр	Объявление	Тип	Область памяти	Описание
R_STRNG				
IN	INPUT	REAL	I, Q, M, D, L, const.	Входная величина
Возвращаемая величина		STRING	D, L	Результирующая строка
STRNG_R				
S	INPUT	STRING	D, L	Входная строка
Возвращаемая величина		REAL	I, Q, M, D, L	Результат

13.4.4 Примеры обработки символьных строк

Складывание текстов сообщений

```
// Поместите вместе тексты сообщений, управляемые процессом,
// и сохраните их
////////////////////////////////////////////////////////////
// Блок содержит тексты необходимых сообщений и генерирует 20
// последних сообщений
////////////////////////////////////////////////////////////

DATA_BLOCK Messagetexts

 STRUCT
  Index   : int;
  textbuffer  : array [0..19] of string[34];
  HW     : array [1..5] of string[16]; //5 различных
                                       //устройств
statuses : array [1..5] of string[12]; // 5 различных
                                       //состояний
   END_STRUCT
BEGIN
  Index  :=0;
  HW[1]  := 'Motor ';
  HW[2]  := 'Valve ';
  HW[3]  := 'Press ';
  HW[4]  := 'Weldingstation ';
  HW[5]  := 'Burner ';
  Statuses[1]  := ' problem';
  Statuses[2]  := ' started';
  Statuses[3]  := ' temperature';
  Statuses[4]  := ' repaired';
  Statuses[5]  := ' maintained';
END_DATA_BLOCK

////////////////////////////////////////////////////////////
///
//Функция выводит тексты сообщений вместе и вводит их в
//DB текстов сообщений. Тексты сообщений хранятся в
//циклическом буфере
//Следующий индекс буфера текстов находится также
//в DB текстов сообщений и обновляется функцией.
////////////////////////////////////////////////////////////
///
FUNCTION Textgenerator : bool
VAR_INPUT
  unit      : int;   //Индекс текста устройства
  no          : int; // Номер устройства
  status  : int;
  value      : int;
END_VAR
VAR_TEMP
```

```
  text  : string[34];
  i       : int;
END_VAR
//initialization of the temporary variables
  text := '';
    Textgenerator := true;
    Case unit of
      1..5 : case status of
        1..5 : text := concat( in1 := Messagetexts.HW[unit],
                in2 := right(l:=2,in:=I_STRNG(no)));
             text := concat( in1 := text,
                in2 := Messagetexts.statuses[status]);
            if value <> 0 then
             text := concat( in1 := text,
                    in2 := I_STRNG(value));
        end_if;
       else Textgenerator := false;
  end_case;
else Textgenerator := false;
end_case;
i := Messagetexts.index;
  Messagetexts.textbuffer[i] := text;
  Messagetexts.index := (i+1) mod 20;
END_FUNCTION

//////////////////////////////////////////////////////////////
//Функция вызывается в циклической программе по фронту
//на %M10.0 и сообщение вводится однократно при изменении параметра.
//////////////////////////////////////////////////////////////

Organization_block Cycle
Var_temp
  Opsy_ifx : array [0..20] of byte;
  error: BOOL;
End_var;

//////////////////////////////////////////////////////////////
//Следующий вызов вводит сообщение "Motor 12 started" в
//текстовый буфер DB текстов сообщений, при установке %MW0 в 1, %IW2
//в 12 и %MW2 в 2. *)
//////////////////////////////////////////////////////////////

if %M10.0 <> %M10.1 then
    error := Textgenerator (unit   := word_to_int(%MW0),
                no     := word_to_int(%IW2),
                status := word_to_int(%MW2),
                value   := 0);
                %M10.1:=M10.0;
end_if;
end_organization_block
```

13.5 SFC, SFB и Стандартная Библиотека

В CPU S7 встроены стандартные функции операционной системы, которые Вы можете использовать, программируя на SCL. К ним так же относятся:

- Организационные блоки(OB)
- Системные функции (SFC)
- Системные функциональные блоки (SFB)

Интерфейс вызова (SFC/SFB)

Вы можете обращаться к блокам в символической или абсолютной форме. Вам необходимо либо символическое имя, которое должно быть объявлено в символьной таблице, либо номер абсолютного идентификатора блока.

Когда вызываются эти функции и блоки, Вы должны назначить фактические параметры (величины, которые будет использовать блок в процессе исполнения программы) формальным параметрам, имена и типы данных которых были определены, когда создавался реконфигурируемый блок.

SCL ищет вызываемый блок в следующих директориях и библиотеках:

- Каталог "Programs"
- Стандартные библиотеки Simatic
- Стандартные библиотеки IEC

Если SCL находит блок, он копируется в пользовательскую программу. Исключения составляют блоки, которые должны вызываться с записью (" ... ") через свое имя и блоки, вызываемые с абсолютным идентификатором. После этого SCL ищет их имена только в символьной таблице пользовательской программы. Вы должны самостоятельно скопировать эти функции в вашу пользовательскую программу, используя SIMATIC Manager. Это применимо к следующим функциям IEC.

- "DATE and TOD to DT"
- "DT to DATE"
- "DT to DAY"
- "DT to TOD"

Условный вызов (SFB/SFC)

Для условного вызова функции, Вы можете присвоить предопределенному входному параметру EN значение 0 (например, через вход I0.3). После этого блок не вызывается. Если EN равен 1, функция вызывается. Выходной параметр ENO также выставляется равным "1" (в противном случае "0") в случае, если при исполнении блока не возникало ошибок.

Условные вызовы SFC не рекомендуется использовать, так как переменные, которые обычно должны получать возвращаемую функцией величину, не определяются, если функция не вызывается.

Примечание

Если Вы в программе используете следующие операции для типов TIME, DATE_AND_TIME и STRING, SCL неявно вызывает соответствующие стандартные блоки.

Поэтому символические имена и номера этих стандартных блоков зарезервированы и не должны использоваться для других блоков. Нарушение этого правила не всегда распознается SCL и может привести к ошибкам компилятора.

Следующая таблица содержит обзор стандартных функций IEC, неявно используемых в SCL.

Операция	**DATE_AND_TIME**	**STRING**
==	EQ_DT (FC9)	EQ_STRING (FC10)
<>	NE_DT (FC28)	NE_STRING (FC29)
>	GT_DT (FC14)	GT_STRING (FC15)
>=	GE_DT (FC12)	GE_STRING (FC13)
<=	LE_DT (FC18)	LE_STRING (FC19)
<	LT_DT (FC23)	LT_STRING (FC24)
DATE_AND_TIME + TIME	AD_DT_TM (FC1)	
DATE_AND_TIME + TIME	SB_DT_TM (FC35)	
DATE_AND_TIME + DATE_AND_TIME	SB_DT_DT (FC34)	
TIME_TO_S5TIME(TIME)	TIM_S5TI (FC40)	
S5TIME_TO_TIME(S5TIME)	S5TI_TIM (FC33)	

За детальной информацией о возможных SFB, SFC и OB и детальным описанием интерфейса, Обращайтесь к руководству "System Software for S7-300 and S7-400, System and Standard Functions" ("Системное программное обеспечение для S7-300 и S7-400, Системные и стандартные функции").

13.5.1 Интерфейс взаимодействия с операционной системой через OB

Организационные блоки

Организационные блоки формируют интерфейс между операционной системой CPU и пользовательской программой. OB могут использоваться для исполнения определенных разделов программы в следующих ситуациях:

- Когда CPU загружается
- При циклических или временных операциях
- В особое время или особые дни
- По истечении определенного периода времени
- В случае возникновения ошибки
- Если срабатывают аппаратные или коммуникационные прерывания

Организационные блоки обрабатываются в соответствии с приоритетом их задания.

Доступные OB

Не все CPU могут исполнять все OB, предоставляемые S7. Обращайтесь к спецификации Вашего CPU, чтобы узнать, какие OB Вы можете использовать.

14 Описание языка

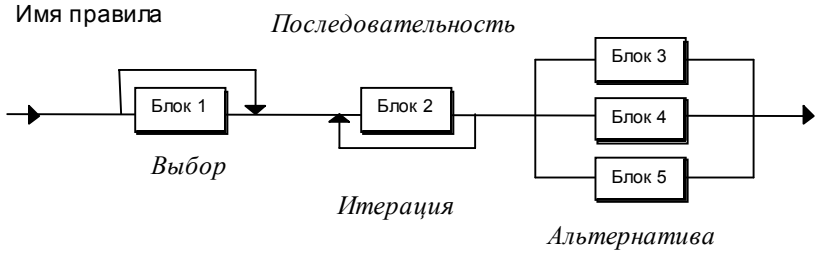
14.1 Формальное описание языка

14.1.1 Обзор синтаксических диаграмм

Основной способ описания языка в различных разделах – это синтаксические диаграммы. Они обеспечивают ясное понимание структуры языка SCL. В разделах "Лексические правила" и "Синтаксические правила" Вы найдете полный набор всех диаграмм языковых элементов.

Что такое синтаксическая диаграмма?

Синтаксическая диаграмма – это графическое представление структуры языка. Структура определяется рядом правил. Одно правило может основываться на другом на более глубоком уровне.

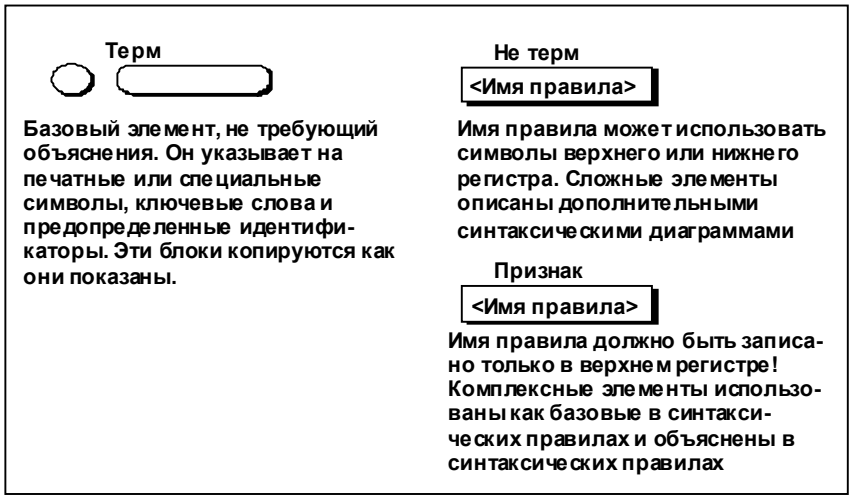


Синтаксическая диаграмма читается справа налево. Основные элементы диаграммы:

- Последовательность: порядок следования блоков
- Выбор: возможность попустить необязательный элемент
- Итерация: повторение веток
- Альтернатива: выбор из нескольких ветвей

Что такое тип блоков?

Блок – это базовый элемент или элемент, создающий другие объекты. На диаграмме ниже показаны различные типы блоков



14.1.2 Правила

Правила, которые Вы применяете к структуре программы SCL, подразделяются на **лексические** и **синтаксические**.

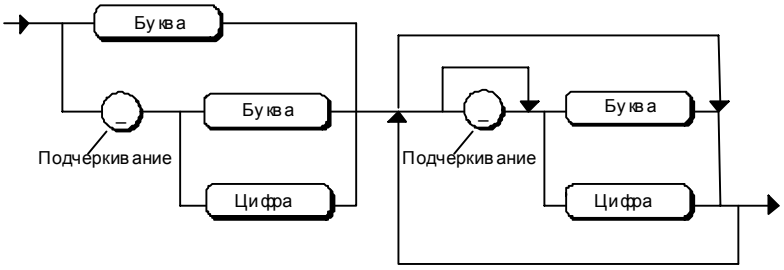
Лексические правила

Лексические правила описывают структуру элементов (признаки), обрабатывающуюся в течение лексического анализа, выполняемого компилятором. По этой причине лексические правила не допускают гибкого форматирования и должны строго соблюдаться. Конкретно это значит, что:

- Форматирование, включая символьное, не разрешено,
- Нельзя включить разделы или строки комментариев.
- Не разрешено включение атрибутов идентификаторов.

Этот пример показывает лексическое правило для идентификатора. Он определяет структуру идентификатора (имя), например:

MEAS_FIELD_12
SETPOINT_B_1



Синтаксические правила

Синтаксические правила строятся на лексических и определяют структуру SCL. В пределах действия этих правил, структура Вашей SCL программы имеет гибкий формат.

Формальные аспекты

Каждое правило имеет имя, которое предшествует описанию. Если это правило используется на более высоком уровне правил, то имя появляется в этом правиле не как терм.

Если имя правила записано прописными буквами, значит оно - лексическое.

Семантика

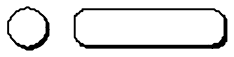
Правила могут представлять только формальную структуру языка. Значение (семантика) не всегда очевидна из правил. По этой причине, где это важно, вместе с правилом приводится дополнительная информация. Следующие примеры таких ситуаций:

- Когда элементы одного типа имеют различное значение, дается дополнительное имя: например, в правиле спецификации даты СТРОКА ДЕСЯТИЧНЫХ ЦИФР это год, месяц или день. Имя указывает использование.
- Важные ограничения отмечены рядом правил: Например, вы можете найти замечание, что символическое имя должно быть определено в таблице символов.

14.1.3 Термы, использующиеся в лексических правилах

Описание

Терм – это базовый элемент, который не может быть объяснен другими правилами, только устно. В синтаксической диаграмме он показан следующим символом:



Терм представлен прямоугольником с закругленными углами или кругом. Пункт показывается как символ или как имя (буквами верхнего регистра). Это определяет диапазон символов кода ASCII, которые могут быть использованы.

В таблице ниже термы определены на основе диапазона символов из набора ASCII.

Буквы и числовые символы

Буквы и числовые символы - главные использующиеся символы. Идентификатор, например, состоит из букв, числовых символов и символа подчеркивания.

Символ	Подгруппа	Диапазон символов
Letter (Буква)	Строчная и прописная	От A до Z от a до z
Digit (Цифра)	Десятичные цифры	0.. 9
Octal digit (Восьмеричная цифра)	Восьмеричные цифры	0.. 7
Hexadecimal digit (Шестнадцатеричная цифра)	Шестнадцатеричные цифры	От 0 до 9, от A до F, от a до f
Bit (Бит)	Двоичные цифры	0, 1

Печатные символы и специальные символы

Полный, расширенный набор символов ASCII, может использоваться в строках, комментариях и символах.

Символ	Подгруппа	Диапазон символов
Печатаемые символы	Зависят от кодовой таблицы. В ASCII коде, например, символы начинаются с эквивалента десятичного кода 32 без DEL и без следующих символов замены:	Все печатные символы
Символы замены	Знак доллара Запятая	$ '
Символы управления	$P или $p $L или $l $R или $r $T или $t $N или $n	Прогон страницы перевод строки возврат каретки табуляция новая строка
Замена в шестнадцатерич-ном коде	$hh	Любые символы, способные к представлению в шестнадцатеричном коде (hh)

14.1.4 Символы форматирования, разделения и операций

Использование лексических правил

Следующая таблица показывает символы таблицы ASCII, использующиеся как символы форматирования и разделители в лексических правилах.

Символы	Описание
:	Ставятся между часом, минутами и секундами Атрибут
.	Разделяет абсолютные адреса в действительных числах или представление периода времени
' '	Символы и символьная строка
" "	Вводный символ для правил редактирования
_ подчерки-вание	Разделяет числа в константах и может быть использован в идентификаторах
$	Символ перехода для специфических символов управления или символов замены
$> $<	Разрыв строки, в случае, если строка занимает не в одну строку в тексте исходного файла, или при включении комментария.

Для констант

Следующая таблица показывает использование индивидуальных символов и символов строки для констант в лексических правилах. Таблица содержит английскую и немецкую мнемонику.

Префикс	Представление	Лексическое правило
BOOL#	Константа типа BOOL	BIT константа
BYTE#	Константа типа BYTE	BIT константа
WORD#	Константа типа WORD	BIT константа
DWORD#	Константа типа DWORD	BIT константа
INT#	Константа типа INT	Целая константа
DINT#	Константа типа DINT	Целая константа
REAL#	Константа типа REAL	REAL константа
CHAR#	Константа типа CHAR	CHAR константа
2#	Цифровая константа	Строка двоичных цифр
8#	Цифровая константа	Строка восьмеричных цифр
16#	Цифровая константа	Строка шестнадцатеричных цифр
D#	Время	DATE
DATE#	Время	DATE
DATE_AND_TIME#	Время	DATE AND TIME
DT#	Время	DATE AND TIME
E	Разделитель для числовой константы типа REAL	Степень
e	Разделитель для числовой константы типа REAL	Степень

Префикс	Представление	Лексическое правило
D	Разделитель для отрезка времени (день)	дни (правило: сложный формат)
H	Разделитель для отрезка времени (час)	Часы: (правило: сложный формат)
M	Разделитель для отрезка времени (минуты)	Минуты : (правило: сложный формат)
MS	Разделитель для отрезка времени (миллисекунды)	Миллисекунды: (правило: сложный формат)
S	Разделитель для отрезка времени (секунды)	Секунды: (правило: сложный формат)
T#	Время	TIME PERIOD
TIME#	Время	TIME PERIOD
TIME_OF_DAY#	Время	TIME OF DAY
TOD#	Время	TIME OF DAY

В синтаксических правилах

Следующая таблица показывает использование индивидуальных символов как символов форматирования и разделения в синтаксических правилах, в комментариях и атрибутах.

Символ	Описание	Синтаксические правила, ремарки и атрибуты
:	Разделитель перед спецификацией типа, в операторе после метки	Декларация переменной, декларация экземпляра, раздел функционального кода, оператор CASE
;	Завершает декларацию или оператор	Константы и декларации переменных, раздел кода, раздел назначения DB, подраздел констант, подраздел меток, декларация компонент
,	Разделитель для списка и подраздела меток	Декларация переменных, спецификация типа данных массив, инициализация массива, параметры FB, параметры FC, список величин, декларация экземпляров
..	Спецификация диапазона	Спецификация типа данных массив, список величин
.	Разделяет имена FB и DB, абсолютный адрес	Вызов FB, структуры переменных
( )	Функция и вызываемый функциональный блок заключается в скобки в выражениях Список инициализаций для массивов	Вызов функции, вызов FB, выражение, Список инициализаций массива, простое умножение, степенное выражение
[ ]	Декларация массива, раздел структурных переменных массива, индексация глобальных переменных и строк	Спецификация типа данных массива, спецификация типа данных STRING
(* *)	Раздел комментариев	Смотрите "Лексические правила"
//	Строчный комментарий	Смотрите "Лексические правила"
{ }	Поле атрибута	Для специфических атрибутов
%	Вводит прямой идентификатор	Чтобы программировать в соответствии с IEC, %M4.0 может использоваться вместо M4.0.
#	Вводит не ключевое слово	Показывает, что идентификатор не является ключевым словом, например, #FOR.

Операции

Следующая таблица содержит список SCL операций, ключевых слов, например AND и общие операции как простые символы. В этой таблице применима для английской и немецкой мнемоники.

Операция	Описание	Пример, синтаксическое правило
:=	Назначение операций, первоначальное назначение, инициализация типа данных	Назначение величин, раздел назначений DB, подраздел констант, назначение выхода и входа/выхода, назначение входа, назначение входа/выхода
+, -	Арифметические операции: одноместные операции, знак	Выражения, простое выражение, степенное выражение
+, -, *, / MOD; DIV	Основные арифметические операции	Основные арифметические операции, простое умножение
**	Арифметические операции: степенные операции	Выражение
NOT	Логические операции: отрицание	Выражение
AND, &, OR; XOR,	Основные логические операции	Основные логические операции
<,>,<=,>=,=,<>	Операция сравнение	Операция сравнение

14.1.5 Ключевые слова и встроенные идентификаторы

В следующей таблице находится список ключевых слов SCL и встроенных идентификаторов в алфавитном порядке. К каждому из них прилагается описание и лексическое правило, в котором они применяются. Ключевые слова обычно не зависят от мнемоники.

Ключевое слово	Описание	Пример, лексическое правило
AND	Логический оператор	Основной логический оператор
ANY	Идентификатор типа данных ANY	Спецификация типа данных параметра
ARRAY	Вводит спецификацию массива и следует перед индексным списком, заключенным в "[" и "]".	Спецификация типа данных массива
AT	Объявляет альтернативное представление типа переменой	Объявление переменной
BEGIN	Начинает раздел кода в логическом блоке или раздел инициализации в блоке данных	Организационный блок, функция, функциональный блок, блок данных
BLOCK_DB	Идентификатор типа данных BLOCK_DB	Спецификация параметрического типа данных
BLOCK_FB	Идентификатор типа данных BLOCK_FB	Спецификация параметрического типа данных
BLOCK_FC	Идентификатор типа данных BLOCK_FC	Спецификация параметрического типа данных
BLOCK_SDB	Идентификатор типа данных BLOCK_SDB	Спецификация параметрического типа данных
BOOL	Элементарный тип для двоичных данных	Битовый тип данных
BY	Вводит спецификацию приращения	Оператор FOR
BYTE	Элементарный тип данных	Битовый тип данных
CASE	Вводит оператор управления для выбора	Оператор CASE
CHAR	Элементарный тип данных	Символьный тип
CONST	Вводит описание констант	Подраздел констант
CONTINUE	Оператор управления для циклов FOR, WHILE и REPEAT	Оператор CONTINUE
COUNTER	Тип данных счетчиков, используемый только в подразделе деклараций параметров	Спецификация типа данных параметра
DATA_BLOCK	Вводит блок данных	Блок данных
DATE	Элементарный тип данных для даты	Временной тип
DATE_AND_TIME	Составной тип данных для даты и времени	DATE_AND_TIME
DINT	Элементарный тип данных для двойных целых	Цифровой тип данных
DIV	Оператор деления	Базовый арифметический оператор, простое умножение
DO	Вводит раздел операторов для оператора FOR	Оператор FOR, оператор WHILE
DT	Элементарный тип данных для даты и времени	DATE_AND_TIME

Ключевое слово	Описание	Пример, лексическое правило
DWORD	Элементарный тип данных для двойных слов	Битовый тип данных
ELSE	Вводит инструкции, выполняемые, если условие не выполнено	Операторы IF, CASE
ELSIF	Вводит альтернативные условия	Оператор IF
EN	Флаг разрешения выполнения блока	
ENO	Флаг ошибки блока	
END_CASE	Завершает оператор CASE	Оператор CASE
END_CONST	Завершает описание констант	Раздел констант
END_DATA_BLOCK	Завершает блок данных	Блок данных
END_FOR	Завершает оператор FOR	Оператор FOR
END_FUNCTION	Завершает функцию	Функция
END_FUNCTION_BLOCK	Завершает функциональный блок	Функциональный блок
END_IF	Завершает оператор IF	Оператор IF
END_LABEL	Завершает описание меток	Подраздел меток
END_TYPE	Завершает UDT	Определенный пользователем тип данных
END_ORGANIZATION_BLOCK	Завершает организационный блок	Организационный блок
END_REPEAT	Завершает оператор REPEAT	Оператор REPEAT
END_STRUCT	Завершает спецификацию структуры	Спецификация структуры типа данных
END_VAR	Завершает блок декларации	Подразделы временных переменных, статических переменных, параметров
END_WHILE	Завершает оператор WHILE	Оператор WHILE
EXIT	Выполняет немедленный выход из цикла	Оператор EXIT
FALSE	Встроенная булевская константа: логическое условие не истина, величина 0	
FOR	Вводит оператор контроля для обработки цикла	Оператор FOR
FUNCTION	Вводит функцию	Функция
FUNCTION_BLOCK	Вводит функциональный блок	Функциональный блок
GOTO	Инструкция перехода к метке	Программный переход
IF	Вводит управляющий оператор для выбора одной из ветвей алгоритма	Оператор IF
INT	Элементарный тип данных для целых чисел, простых решений	Цифровой тип данных
LABEL	Вводит декларацию подраздела меток	Подраздел меток
MOD	Арифметическая операция для получения остатка от деления	Основная арифметическая операция, простое умножение
NIL	Нулевой указатель	
NOT	Логический одноместный оператор	Выражение
OF	Вводит спецификацию типа данных	Спецификация типа данных массива, оператор CASE
OK	Флаг, указывающий на отсутствие ошибок в блоке	
OR	Логический оператор	Основные логические операции
ORGANIZATION_BLOCK	Вводит организационный блок	Организационный блок
POINTER	Тип данных Pointer, допустимый только в декларации параметров в подразделе параметров, не обрабатывающихся в	См. раздел 10

Ключевое слово	Описание	Пример, лексическое правило
	SCL	
PROGRAM	Вводит раздел операторов FB (синоним FUNCTION_BLOCK)	Функциональный блок
REAL	Элементарный тип данных	Цифровой тип данных
REPEAT	Вводит оператор цикла	Оператор REPEAT
RETURN	Оператор управления, который выполняет возврат из подпрограммы	Оператор RETURN
S5TIME	Элементарный тип данных для спецификации времени, специальный формат S5	Временной тип
STRING	Тип данных для символьной строки	Спецификация типа данных STRING
STRUCT	Вводит спецификацию структуры и предшествует списку компонент	Спецификация типа данных STRUCT
THEN	Вводит действия, выполняемые, если условие удовлетворено	Оператор IF
TIME	Элементарный тип данных для спецификации времени	Временной тип
TIMER	Тип данных таймера, используемый только в подразделе параметров	Спецификация типа данных параметра
TIME_OF_DAY	Элементарный тип данных для времени и даты	Временной тип
TO	Вводит конечную величину	Оператор FOR
TOD	Элементарный тип данных для времени и даты	Временной тип
TRUE	Встроенная булевская константа: выполненное логическое условие, величина не равна 0	
TYPE	Вводит UDT	Определенный пользователем тип данных
VAR	Вводит подраздел декларации	Подраздел статических переменных
VAR_TEMP	Вводит подраздел декларации	Подраздел временных переменных
UNTIL	Вводит условие окончания выполнения оператора REPEAT	Оператор REPEAT
VAR_INPUT	Вводит подраздел декларации	Подраздел параметра
VAR_IN_OUT	Вводит подраздел декларации	Подраздел параметра
VAR_OUTPUT	Вводит подраздел декларации	Подраздел параметра
WHILE	Вводит оператор управления для обработки цикла	Оператор WHILE
WORD	Элементарный тип данных Word	Битовый тип данных
VOID	Вызываемая функция не возвращает величины	Спецификация типа функции
XOR	Логический оператор	Основные логические операции

14.1.6 Идентификаторы адреса и ключевые слова блока

Глобальные системные данные

В следующей таблице показан список SIMATIC мнемоники SCL идентификаторов адреса в алфавитном порядке с описанием каждого их них.

- Спецификация идентификатора адреса:

  Префикс памяти (Q, I, M, PQ, PI) или блока данных (D)

- Спецификация размера элемента данных:

  Префикс размера (дополнительно или B, D, W, X)

Мнемоника представляет комбинацию идентификаторов адреса (префикс памяти или D для блока данных) и префикс размера. Оба являются лексическими правилами. В таблице немецкая мнемоника и во второй колонке соответствующая английская мнемоника.

Немецкая мнемоника	Английская мнемоника	Префикс памяти или блок данных	Префикс размера
A	Q	Выход (через образ процесса)	Бит
AB	QB	Выход (через образ процесса)	Байт
AD	QD	Выход (через образ процесса)	Двойное слово
AW	QW	Выход (через образ процесса)	Слово
AX	QX	Выход (через образ процесса)	Бит
D	D	Блок данных	Бит
DB	DB	Блок данных	Байт
DD	DD	Блок данных	Двойное слово
DW	DW	Блок данных	Слово
DX	DX	Блок данных	Бит
E	I	Вход (через образ процесса)	Бит
EB	IB	Вход (через образ процесса)	Байт
ED	ID	Вход (через образ процесса)	Двойное слово
EW	IW	Вход (через образ процесса)	Слово
EX	IX	Вход (через образ процесса)	Бит
M	M	Область меркеров	Бит
MB	MB	Область меркеров	Байт
MD	MD	Область меркеров	Двойное слово
MW	MW	Область меркеров	Слово
MX	MX	Область меркеров	Бит
PAB	PQB	Выход (Адресация к периферии)	Байт
PAD	PQD	Выход (Адресация к периферии)	Двойное слово
PAW	PQW	Выход (Адресация к периферии)	Слово
PEB	PIB	Вход (Адресация к периферии)	Байт
PED	PD	Вход (Адресация к периферии)	Двойное слово
PEW	PIW	Вход (Адресация к периферии)	слово

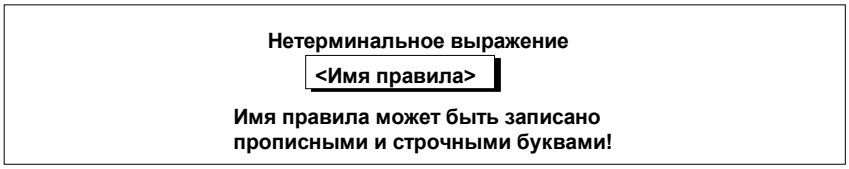
Ключевые слова блока

Используется абсолютная адресация блока. В таблице в первой графе находится немецкая мнемоника и во второй – соответствующая английская.

Немецкая мнемоника	Английская мнемоника	Префикс памяти или блок данных
DB	DB	Блок данных
FB	FB	Функциональный блок
FC	FC	Функция
OB	OB	Организационный блок
SDB	SDB	Системный блок данных
SFC	SFC	Системная функция
SFB	SFB	Системный функциональный блок
T	T	Таймер
UDT	UDT	Определенный пользователем тип данных
Z	C	Счетчик

14.1.7 Нетерминальные выражения

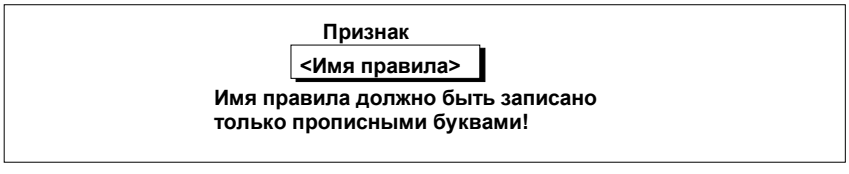
Нетерминальный – это сложный элемент, описанный в последующем правиле. Графически нетерминальное выражение представлено прямоугольным блоком. Имя блока – это имя последующего, более детального правила.



Этот элемент употребляется в лексических и синтаксических правилах.

14.1.8 Обзор признаков

Признак – это базовый элемент, использующийся в синтаксических правилах и объясненный лексическими правилами. Признак представлен прямоугольником блоком. Имя, написанное прописными буквами, это имя поясняющего лексического правила (не показанного в блоке).



Определенные признаки представляют идентификаторы, полученные на основе лексических правил. Такие признаки описывают:

- идентификаторы
- обозначения SCL
- встроенные константы и флаги

14.1.9 Идентификаторы

Идентификатор

У Вас есть доступ к объектам SCL , использующим идентификаторы. В следующей таблице показаны классы идентификаторов.

Тип идентификатора	Комментарии, примеры
Ключевые слова	Например, операторы управления BEGIN, DO,WHILE
Встроенные имена	Имена • Стандартные типы данных (например, BOOL, BYTE, INT) • Встроенные стандартные функции, например ABS • Стандартные константы TRUE и FALSE
Абсолютные идентификаторы адреса	Для глобальных системных данных и блоков данных: например, I1.2, MW10, FC20, T5, DB30, DB10.D4.5
Определенные пользователем имена, основанные на правиле идентификатора IDENTIFIER	Имена • Декларированных переменных • Структурных компонентов • Параметров • Объявленных констант • меток
Символы редактора	Соответствуют лексическому правилу идентификатора или лексическому правилу символа; другими словами, заключены в кавычки, например, "xyz"

Прописные и строчные буквы

Верхний или нижний регистры для записи ключевых слов не важны. Начиная с версии S7 SCL 4.0, нотация встроенных имен и свободно выбираемых имен, таких как переменные и символы из символьной таблицы, более не чувствительна к регистру.

Тип идентификатора	**Чувствителен к регистру?**
Ключевые слова	Нет
Встроенные имена для стандартных типов данных	Нет
Имена стандартных функций	Нет
Встроенные имена для стандартных констант	Нет
Абсолютные идентификаторы адреса	Нет
Определенные пользователем имена	Нет
Символы редактора	Нет

Имена стандартных функций, такие как BYTE_TO_WORD и ABS, могут писаться в нижнем регистре. Это также применимо к параметрам времени и функции счетчика, например, SV, SE или CV.

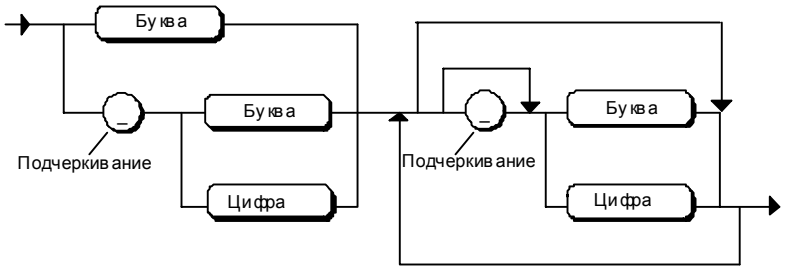
14.1.10 Назначение имен в SCL

Назначение выбранных имен

Вы можете назначить имена двумя основными способами:

- Вы можете назначить имена в SCL сами. Эти имена должны соответствовать правилу идентификатора (смотрите схему). Идентификатор – это важный терм, который Вы можете использовать для любых имен в SCL.
- Как вариант, Вы можете назначить имена в STEP 7, используя таблицу символов. В этом случае также применяется правило Идентификатора или, дополнительно, правило Символа. Вводя имя в кавычках, Вы можете написать символ из всех печатных символов (например, использовать пробел).

ИДЕНТИФИКАТОР



СИМВОЛ

Символы определяются в символьной таблице.

Правила для назначения имен

Пожалуйста, помните следующие пункты:

- Выбирайте имя, которое является однозначным, понятным и улучшает макет программы.
- Проверьте, не используется ли уже это имя, например, идентификатором типа данных или стандартной функцией.
- Область: Если Вы используете имена в глобальной области, она охватывает всю программу. Имена с локальной областью верны только в блоке. Это позволяет Вам использовать те же имена в других блоках. В следующей таблице список возможных выборов.

Ограничения

Когда назначаете имя, помните о следующих ограничениях:

Имя должно быть уникальным в диапазоне своей собственной применимости, то есть, имена, уже использованные в пределах конкретного блока, не могут быть использованы снова в пределах того же блока. Не могут быть использованы имена, зарезервированные системой:

- Имена ключевых слов: например, CONST, END_CONST, BEGIN
- Имена операторов: например, AND, XOR
- Имена встроенных идентификаторов: например, имена типов данных BOOL, STRING, INT
- Имена встроенных констант TRUE и FALSE
- Имена стандартных функций: например, ABS, ACOS, ASIN, COS, LN
- Имена абсолютных идентификаторов адреса для глобальных системных данных: например, IB, IW, ID, QB, QW, QD MB, MD

Использование идентификаторов

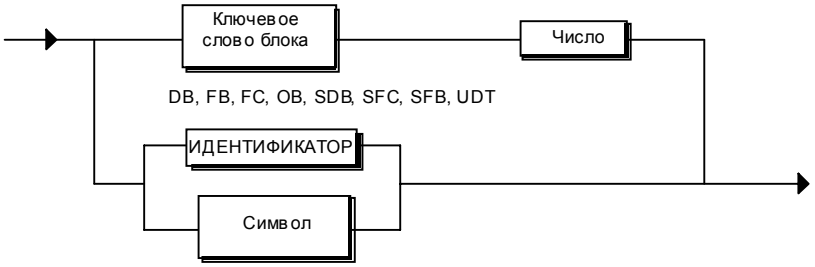
В следующей таблице показаны ситуации, в которых Вы можете использовать имена, соответствующие правилу Идентификатора.

Идентификатор	**Описание**	**правило**
Имя блока	Символьное имя блока	Идентификатор блока, функция вызова
Имя таймера или счетчика	Символьное имя таймера или счетчика	Идентификатор таймера, идентификатор счетчика
Имя атрибута	Имя атрибута	Назначение атрибута
Имя константы	декларация/использование символьной константы	Подраздел констант
Метка	Декларация метки, использование метки	Подраздел метки, оператор GOTO
Имя переменной	Декларация временных или статических переменных	Декларация переменной, простая переменная, Структурная переменная
Локальное имя экземпляра	Декларация локального экземпляра	Декларация экземпляра, имя FB вызова

Идентификатор блока

В правиле ИДЕНТИФИКАТОРА БЛОКА, Вы можете использовать идентификатор или символ:

ИДЕНТИФИКАТОР БЛОКА



Правила Идентификатор таймера и Идентификатор счетчика аналогичны правилу Идентификатора блока.

14.1.11 Встроенные константы и флаги

Таблица применима для английской и немецкой мнемоник.

Константы

Мнемоника	Описание
FALSE	Встроенная булевская константа (стандартная константа) со значением 0. Это логическое значение, если условие не удовлетворено.
TRUE	Встроенная булевская константа (стандартная константа) со значением 1. Это логическое значение, если условие удовлетворено.

Флаги

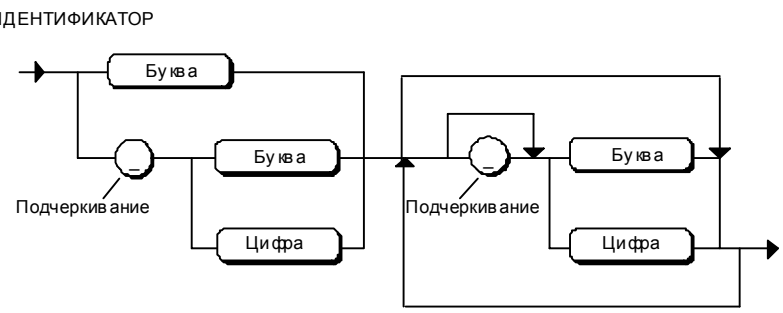
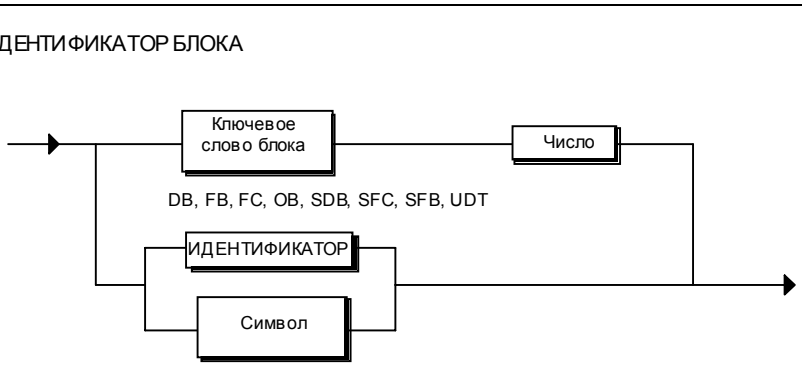
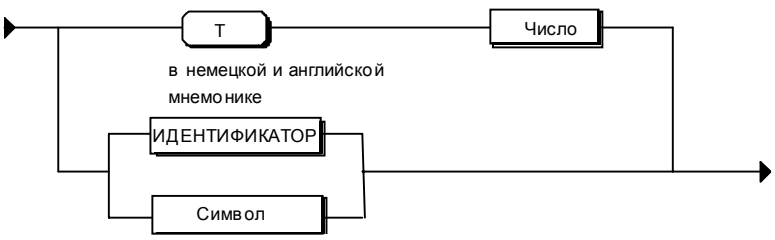
Мнемоника	Описание
EN	Допустимый флаг блока
ENO	Флаг ошибки блока
OK	Флаг, установленный на FALSE, если оператор выполнен неправильно.

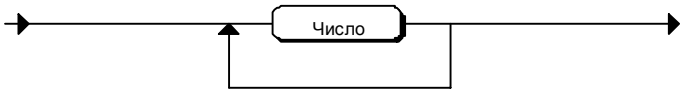
14.2 Лексические правила

Лексические правила описывают структуру элементов (признаков), обрабатывающуюся в течение лексического анализа, выполняемого компилятором. По этой причине лексические правила не имеют гибкого формата и должны строго соблюдаться. Это значит, что

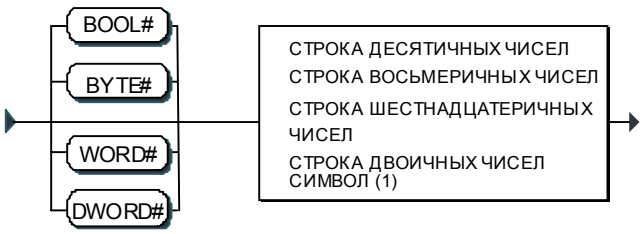
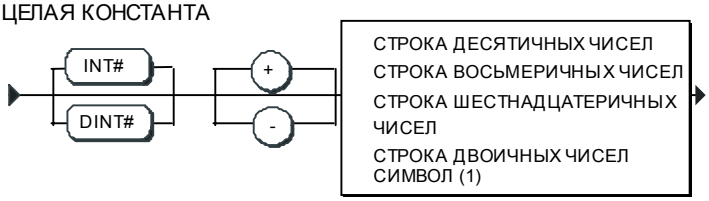
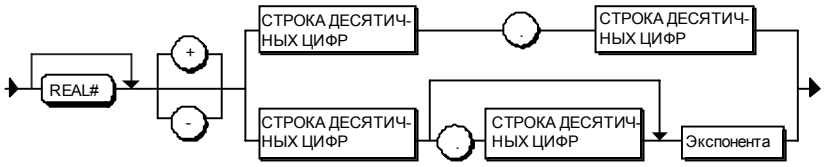
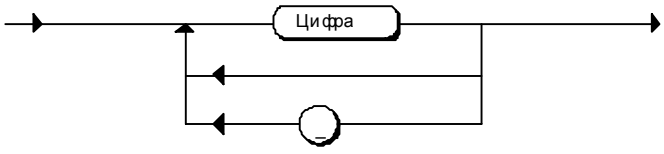
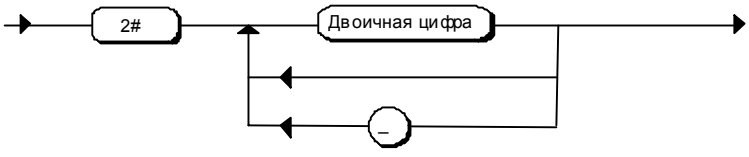
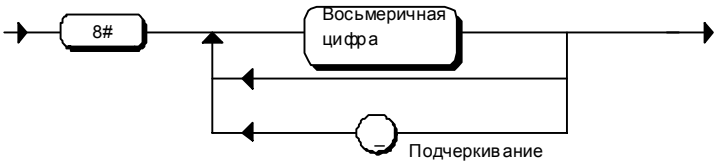
- Форматирование, включая символьное, не разрешено,
- Разделы и строки комментариев не могут быть включены.
- Атрибуты для идентификаторов не разрешены.

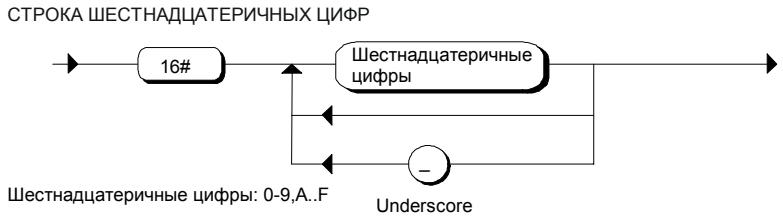
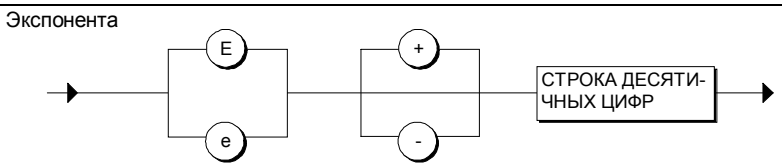
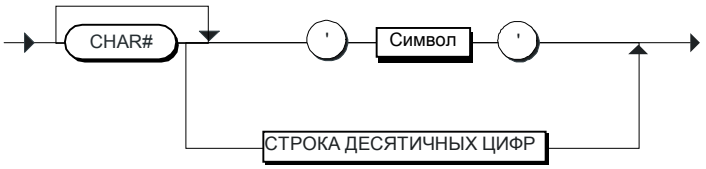
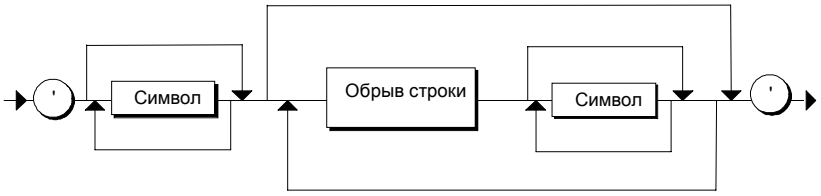
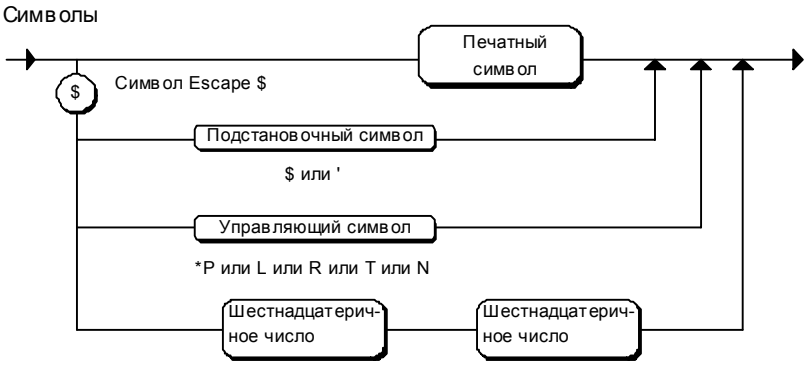
14.2.1 Идентификаторы

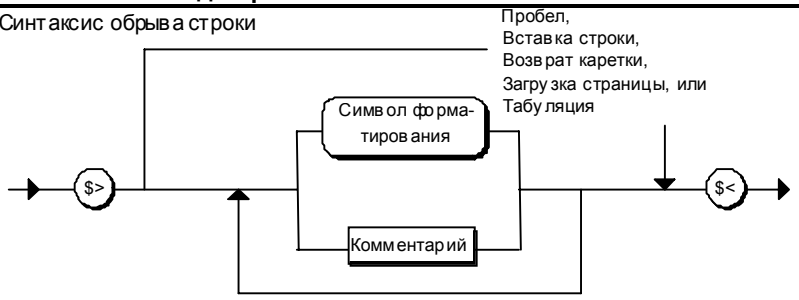
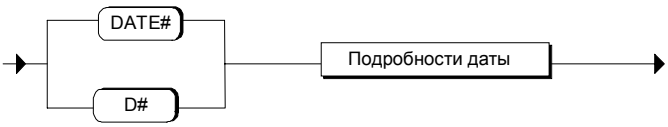
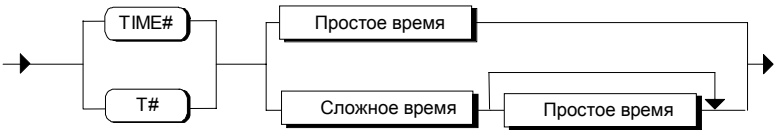
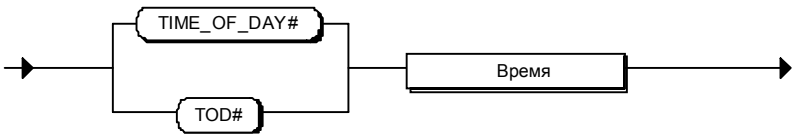
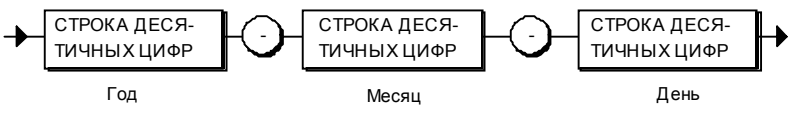
Правило	**Синтаксическая диаграмма**
Идентификатор	
Идентификатор блока	
Идентификатор таймера	

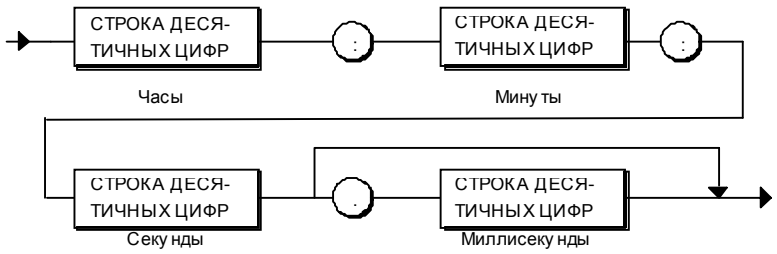
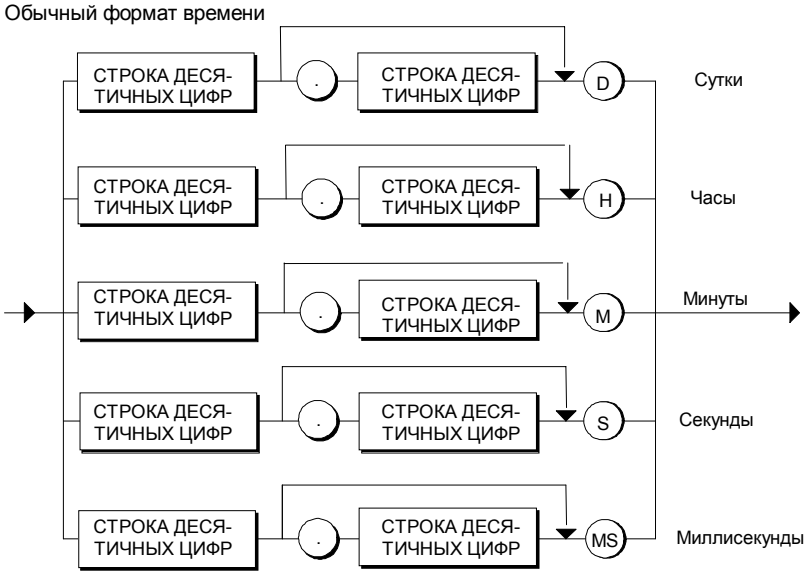
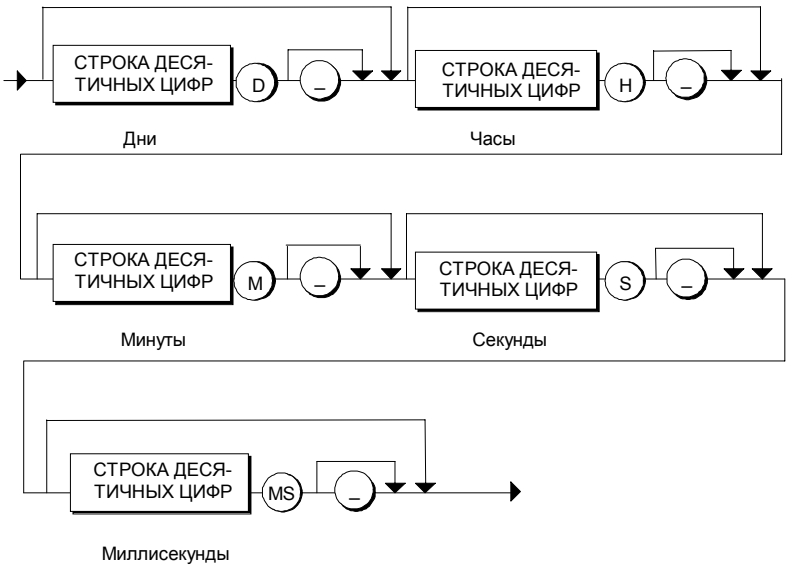
Правило	**Синтаксическая диаграмма**
Идентификатор счетчика	
Ключевые слова блока	
Символ	
Число	

14.2.2 Константы

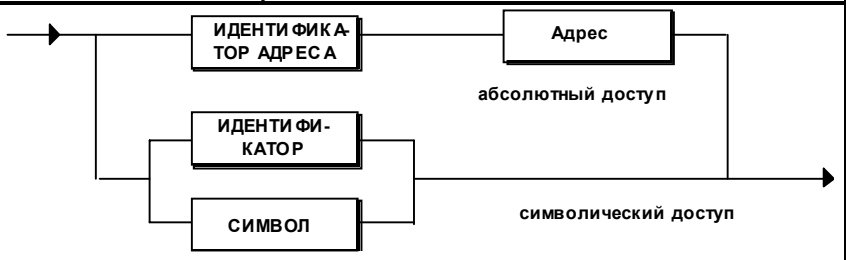
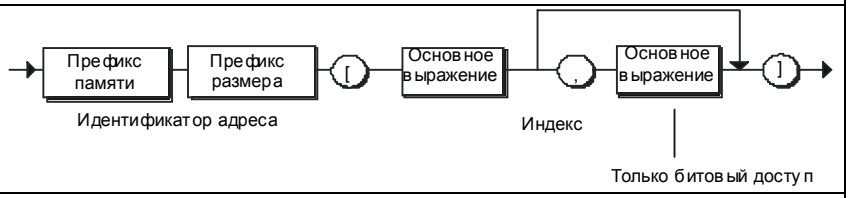
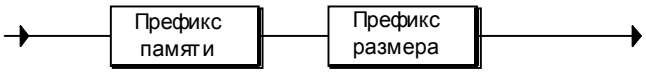
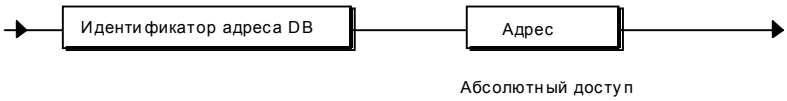
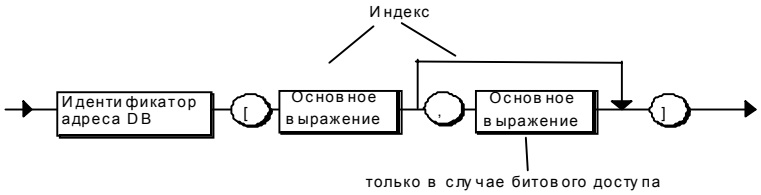
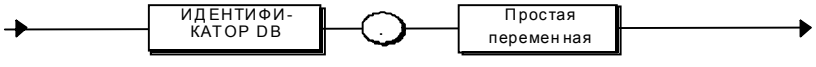
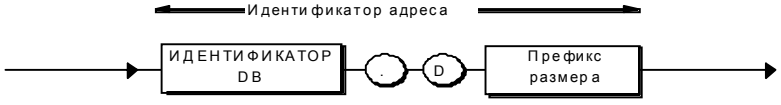
Правило	Синтаксическая диаграмма
Битовая константа	БИТОВАЯ КОНСТАНТА  (1) только типа BYTE
Целая константа	ЦЕЛАЯ КОНСТАНТА  (1) только типа INT
Действительная константа	ВЕЩЕСТВЕННАЯ КОНСТАНТА 
Строка десятичных цифр	Строка десятичных цифр  Десятичные цифры: 0-9
Строка двоичных цифр	Строка двоичных цифр  Двоичные цифры: 0 or 1
Строка восьмеричных цифр	Строка восьмеричных цифр 

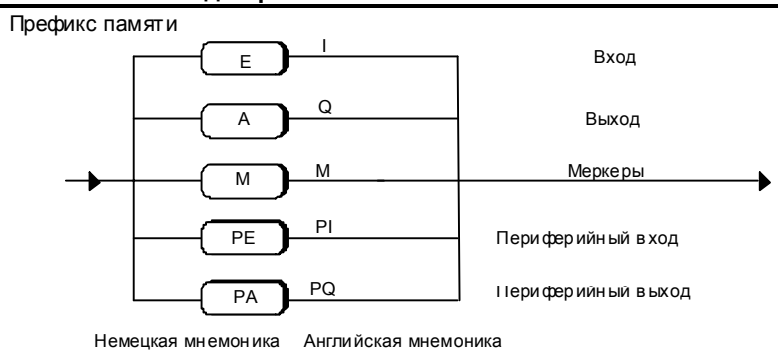
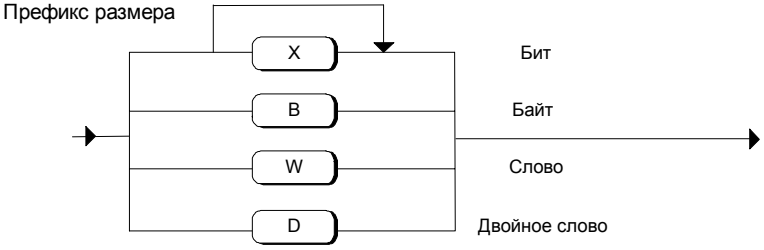
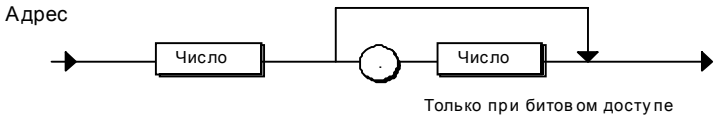
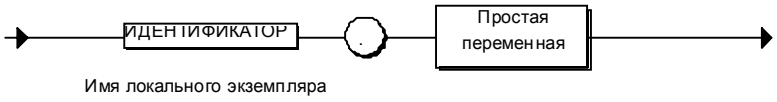
Правило	**Синтаксическая диаграмма**
Строка шестнадцатеричных цифр	
СТЕПЕНЬ	
Символьная константа	
Строковая константа	
СИМВОЛЫ	

Правило	**Синтаксическая диаграмма**
Строка разрыва	Синтаксис обрыва строки 
ДАТА	Дата 
Период времени	ПЕРИОД ВРЕМЕНИ  - Каждая единица времени (часы, минуты, и т.д.) должна быть задана однократно. - Должен соблюдаться порядок перечисления: дни, часы, минуты, секунды, миллисекунды.
Время суток	ВРЕМЯ СУТОК 
Дата и время	ДАТА И ВРЕМЯ 
ДАТА	Дата 

Правило	**Синтаксическая диаграмма**
ВРЕМЯ СУТОК	Время суток 
ДЕСЯТИЧНОЕ ПРЕДСТАВЛЕНИЕ	Обычный формат времени  Использование обычного формата времени возможно только для еще не определенных единиц времени.
СОСТАВНОЙ ФОРМАТ ВРЕМЕНИ	Составной формат времени 

14.2.3 Абсолютная адресация

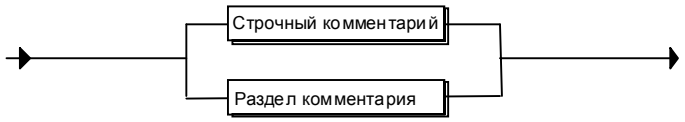
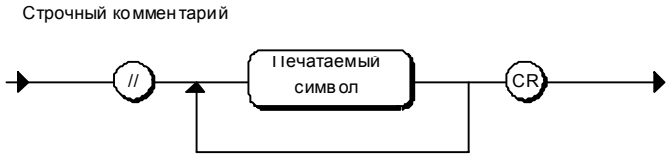
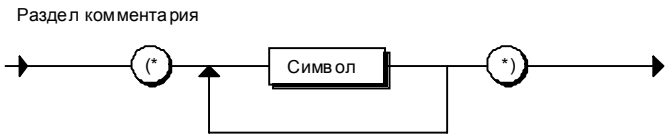
Правило	Синтаксическая диаграмма
Простой доступ к памяти	
Индексированный доступ памяти	
Идентификатор адреса для памяти	
Абсолютный доступ DB	
Индексированный доступ DB	
Структурированный доступ DB	
Идентификатор адреса DB	

Правило	**Синтаксическая диаграмма**
Префикс памяти	
Префикс размера для памяти и DB	
Адрес для памяти и DB	
Доступ к локальному экземпляру	

14.2.4 Комментарии

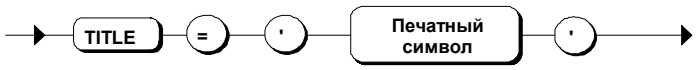
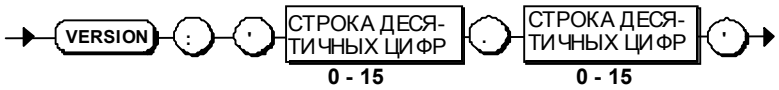
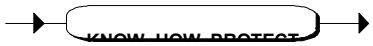
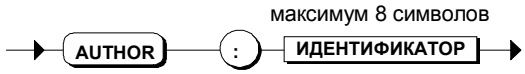
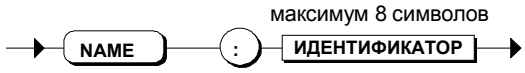
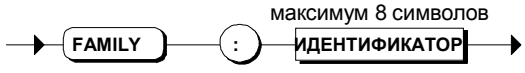
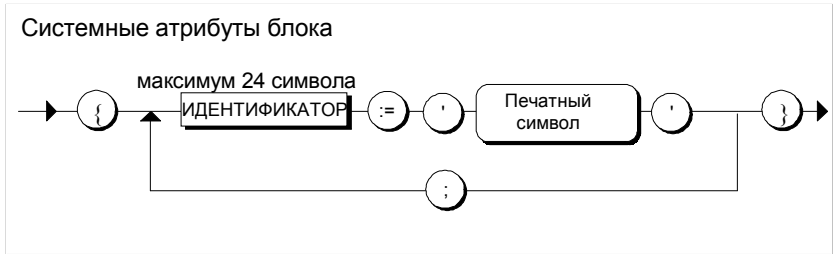
Важные случаи, которые нужно помнить при работе с комментариями:

- Вложенность комментариев разрешена, если активирован "доступ вложенных комментариев".
- Комментарии могут быть в любой точке синтаксического правила, но не в лексическом правиле.

Правило	**Синтаксическая диаграмма**
Комментарий	
Строка комментария	
Раздел комментария	

14.2.5 Атрибуты блока

Атрибуты блока могут располагаться после Идентификатора блока и перед подразделом декларации первой переменной или параметра. Использующийся синтаксис показан ниже.

Правило	Синтаксическая диаграмма
Заголовок	
Версия	
Защита блока	
Автор	
Имя параметра	
Серия блоков	
Системные атрибуты блоков	

14.3 Синтаксические правила

Синтаксические правила строятся на лексических и определяют структуру SCL. С некоторыми ограничениями, структура программы SCL гибкая.

У каждого правила есть наименование, которое предшествует описанию. Если правило используется как высокоуровневое, то имя не является не терм.

Если имя находится в прямоугольнике, написанное заглавными буквами, это значит, что это признак, описанный в лексическом правиле.

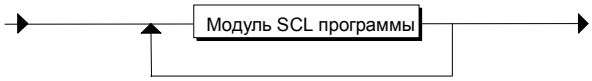
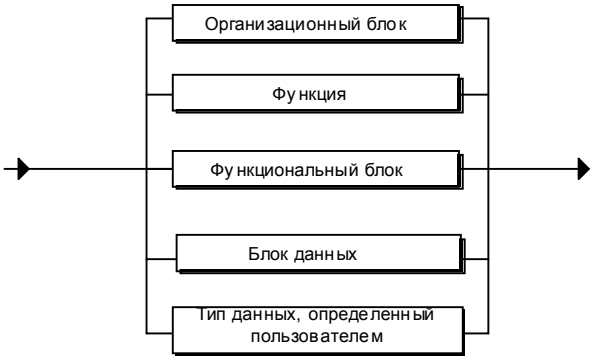
Вы найдете информацию о названиях правил в кругах или овалах в разделе "Формальное описание языка ".

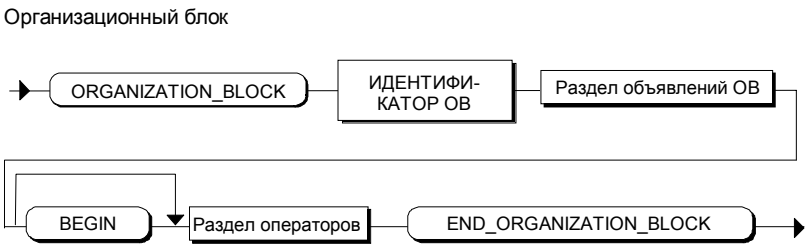
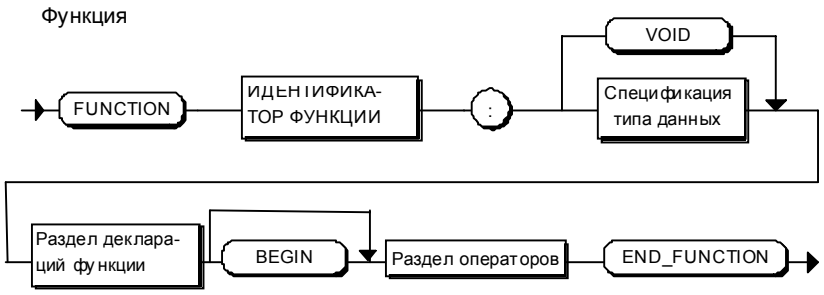
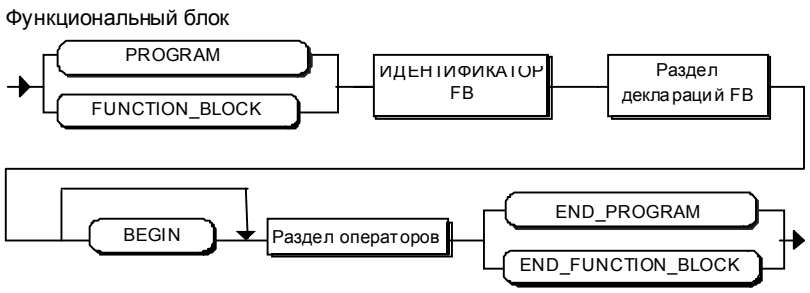
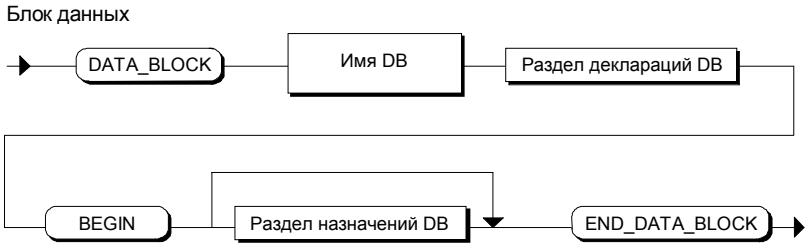
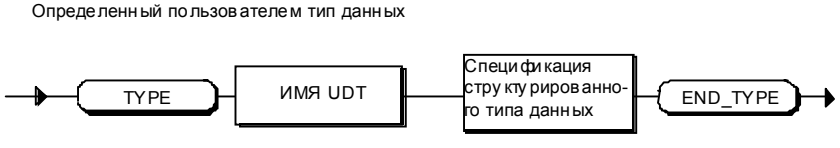
Гибкий формат

Гибкий формат значит:

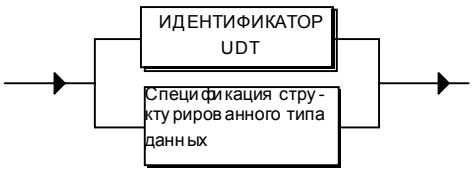
- Вы можете включить символы форматирования в любой точке программы.
- Вы можете включить комментарии и разделы комментариев

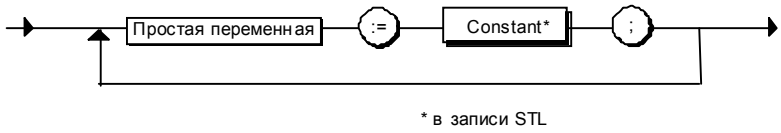
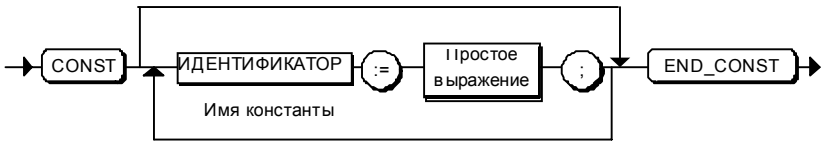
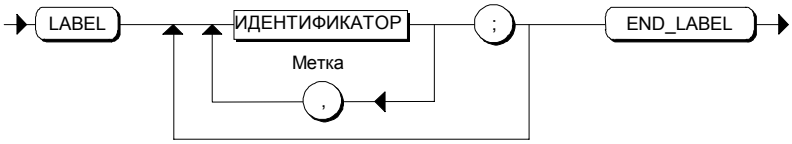
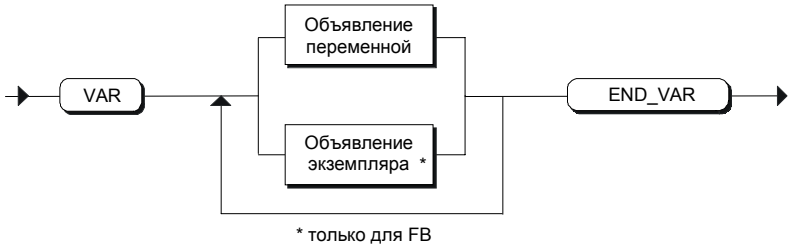
14.3.1 Структура исходного файла SCL

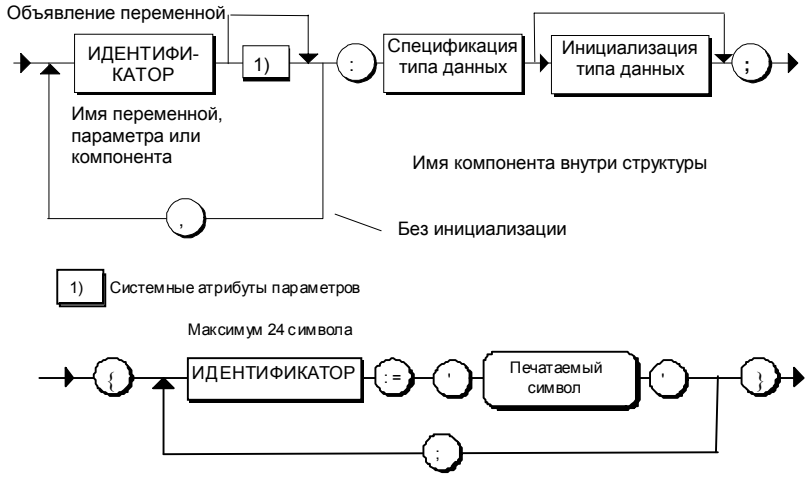
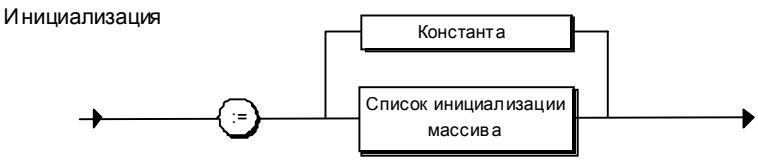
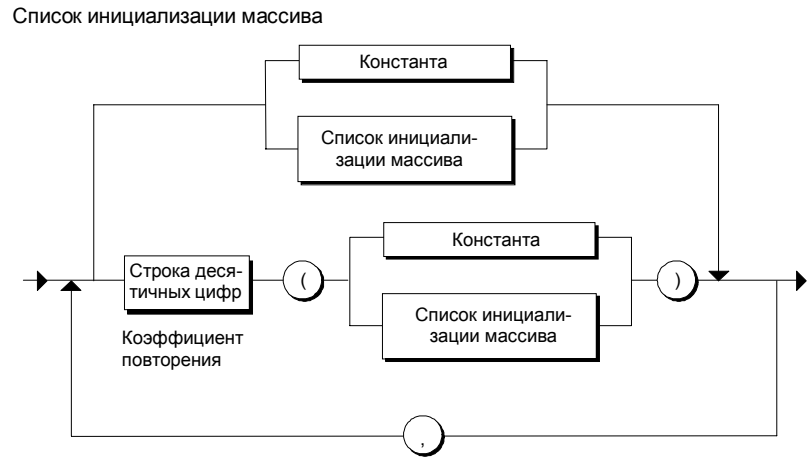
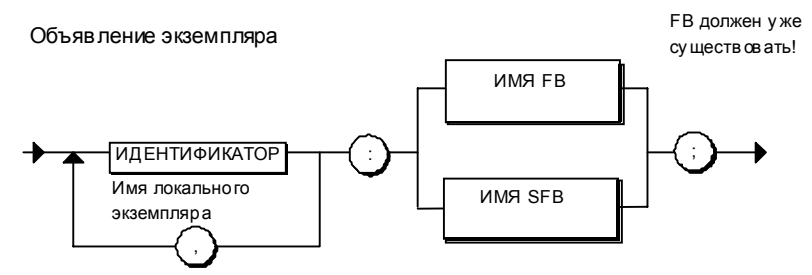
Правило	**Синтаксическая диаграмма**
SCL программа	
Модуль программы SCL	

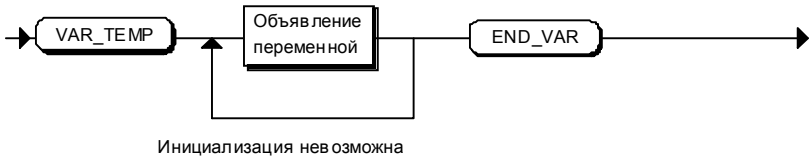
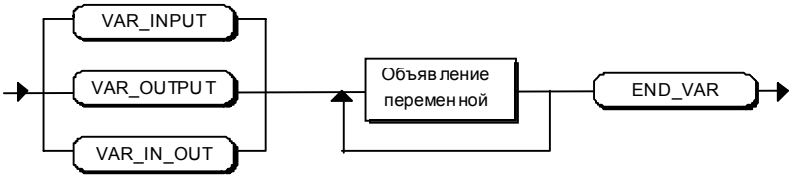
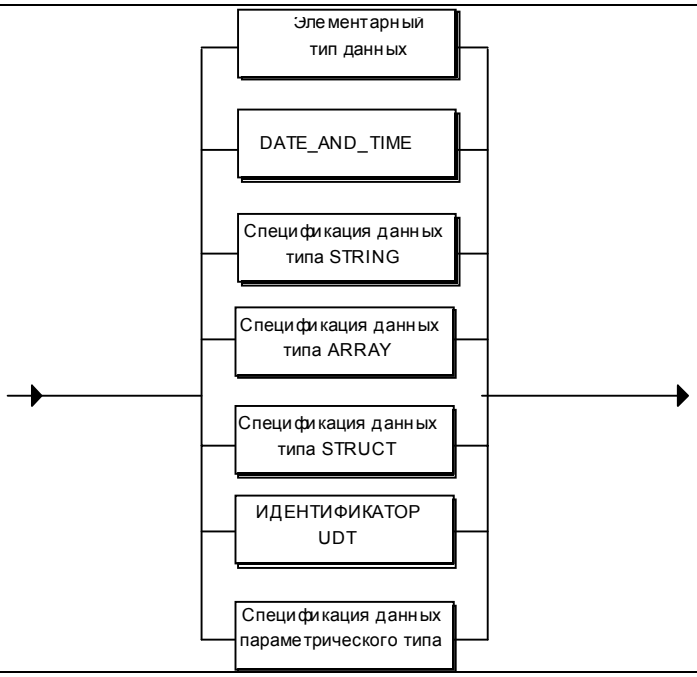
Правило	**Синтаксическая диаграмма**
Организационный блок	
Функция Заметьте, что функция не может быть VOID в кодовом разделе, обратная величина должна присваиваться имени функции.	
Функциональный блок	
Блок данных	
Определенный пользователем тип данных	

14.3.2 Структура раздела декларации

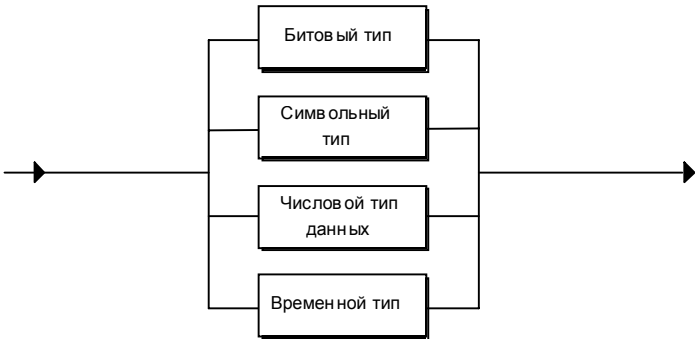
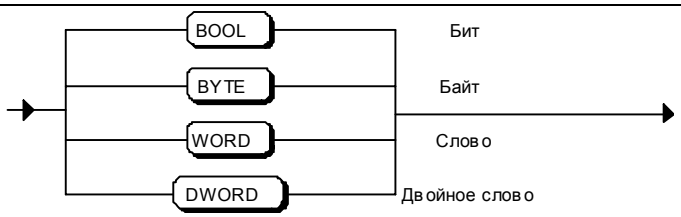
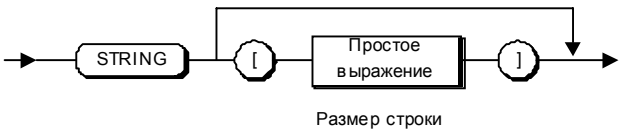
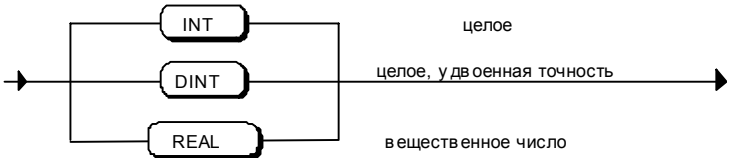
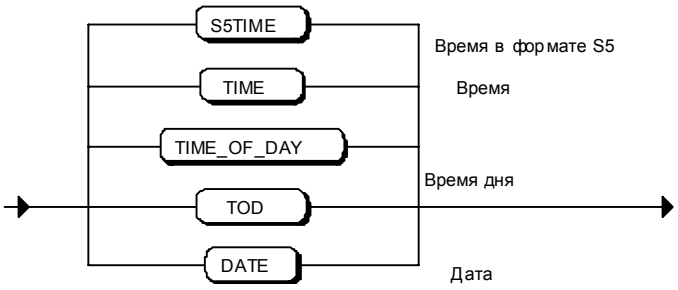
Правило	Синтаксическая диаграмма
Раздел декларации OB	
Раздел декларации FC	
Раздел декларации FB	
Раздел декларации DB	

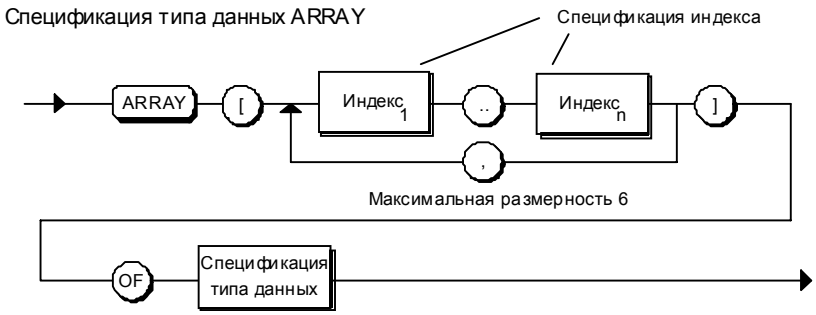
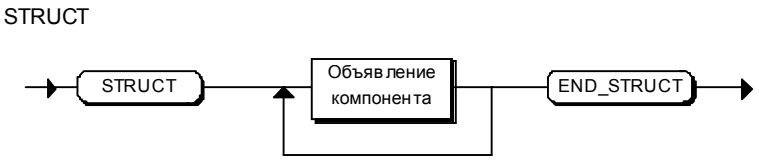
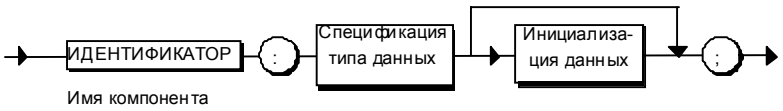
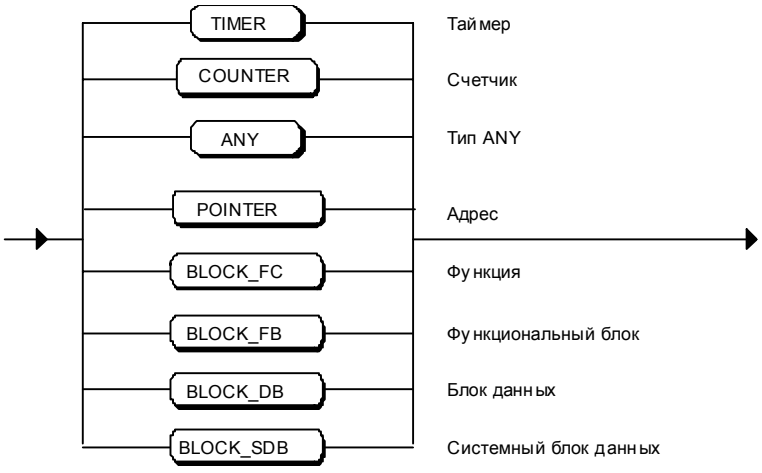
Правило	**Синтаксическая диаграмма**
Раздел назначений DB	Раздел назначений DB 
Подраздел констант	Подраздел констант 
Подраздел меток	Подраздел меток 
Подраздел статических переменных	Раздел статических переменных 

Правило	**Синтаксическая диаграмма**
Объявление переменной	
Инициализация типа данных	
Список инициализаций массива	
Декларация экземпляра (возможна только в разделе VAR блока FB)	

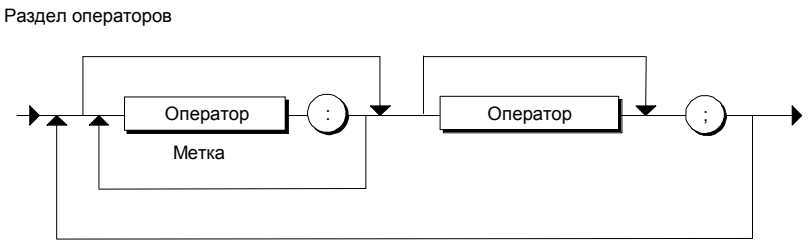
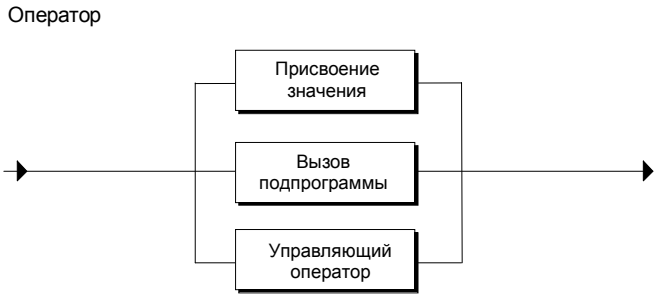
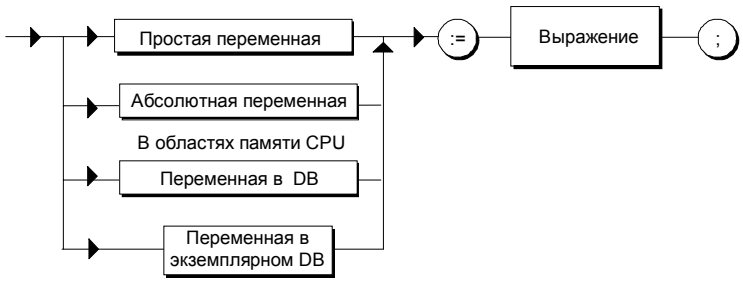
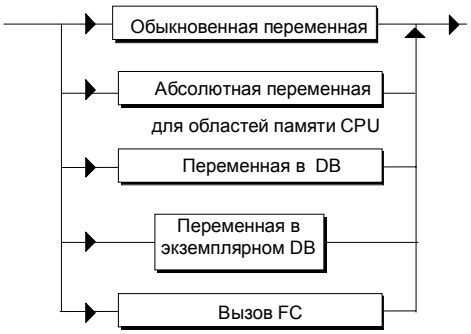
Правило	**Синтаксическая диаграмма**
Подраздел временных переменных	Подраздел временных переменных  Инициализация невозможна
Подраздел параметров	Блок параметров  Инициализация возможна только для VAR_INPUT и VAR_OUTPUT
Спецификация типа данных	

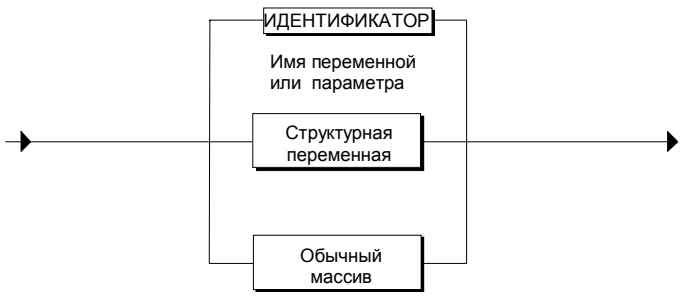
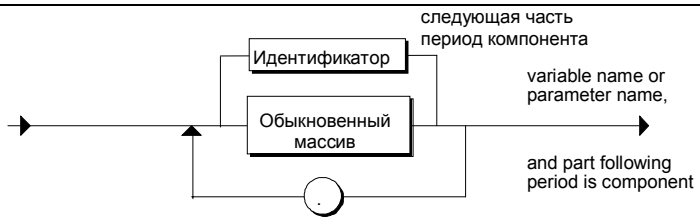
14.3.3 Типы данных в SCL

Правило	**Синтаксическая диаграмма**
Элементарный тип данных	
Битовый тип данных	
Символьный тип	
Спецификация типа данных STRING	Спецификация типа данных STRING 
Цифровой тип данных	
Временной тип	
DATE_AND_TIME	DATE_AND_TIME 

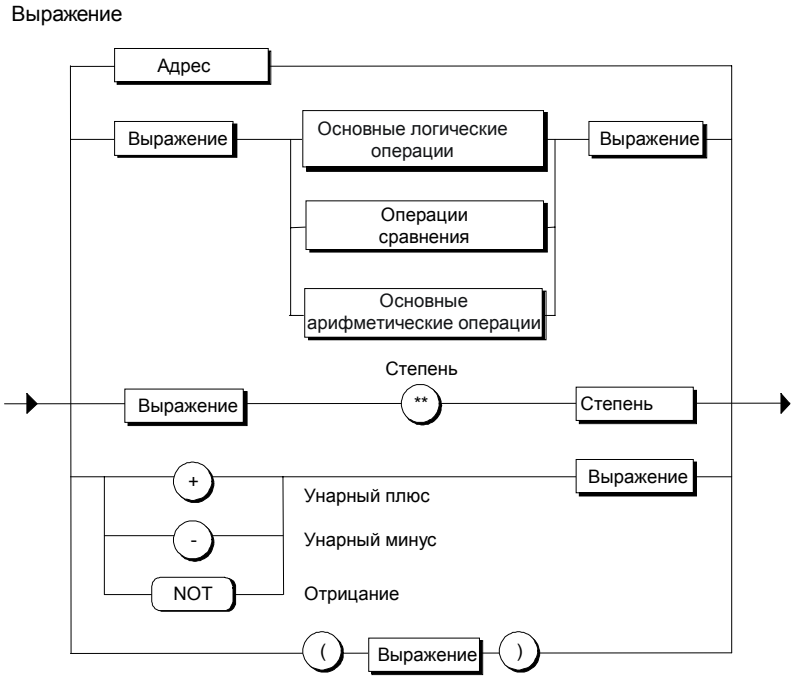
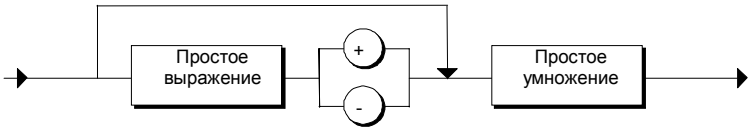
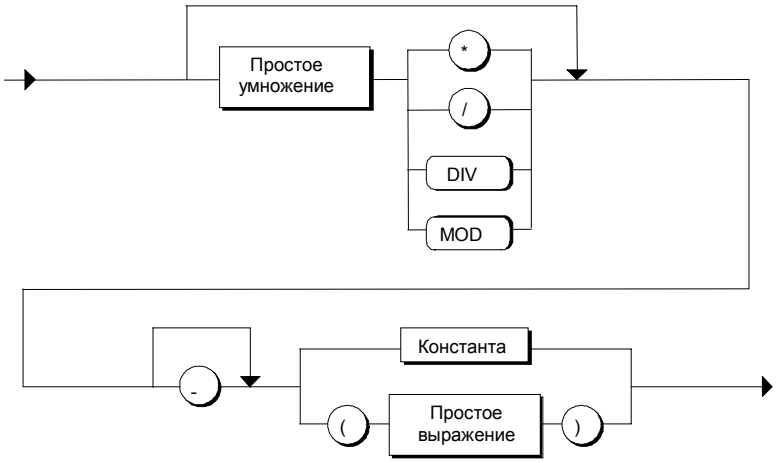
Правило	**Синтаксическая диаграмма**
Спецификация типа данных ARRAY	
Спецификация типа данных STRUCT Помните, что ключевое слово END_STRUCT должно заканчиваться точкой с запятой.	
Объявление компонента	
Спецификация типа параметра	

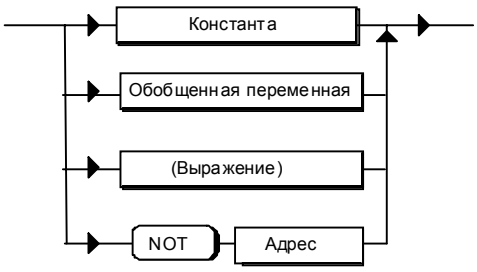
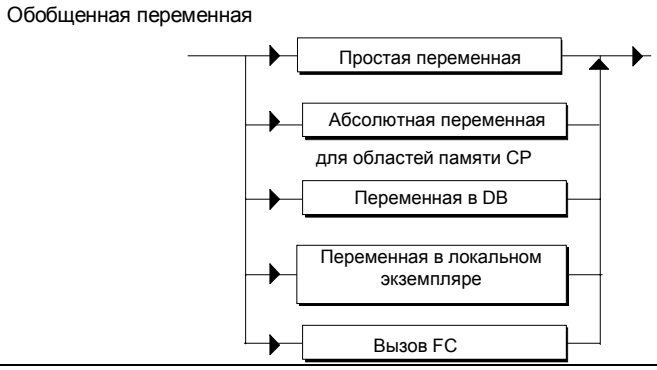
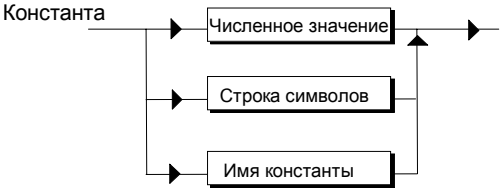
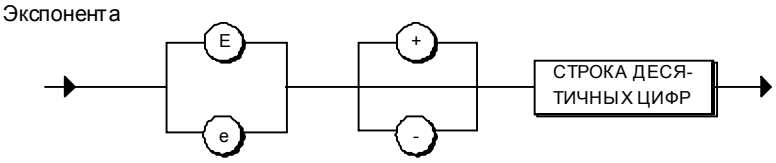
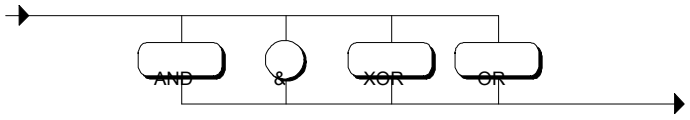
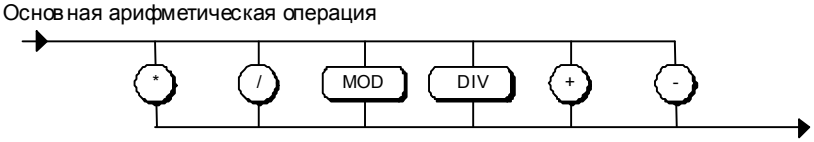
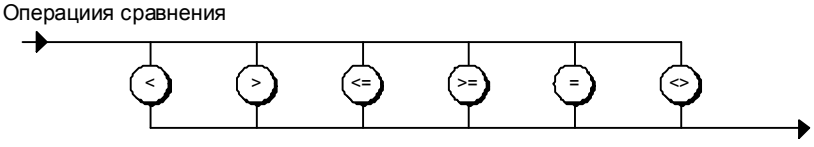
14.3.4 Раздел операторов

Правило	Синтаксическая диаграмма
Раздел операторов	
Оператор	
Присвоение значения	
Обобщенная переменная	

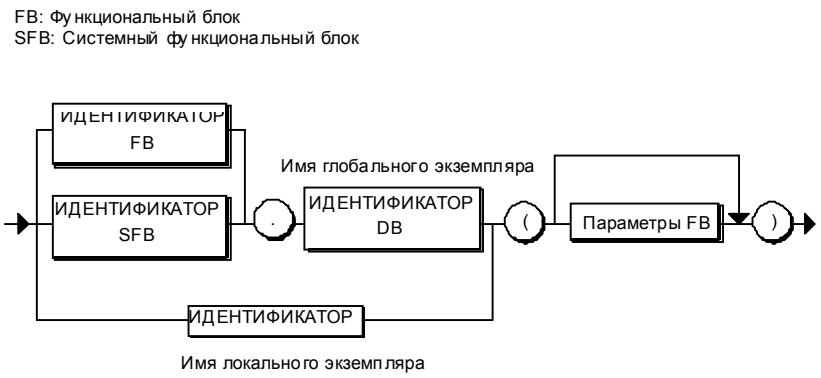
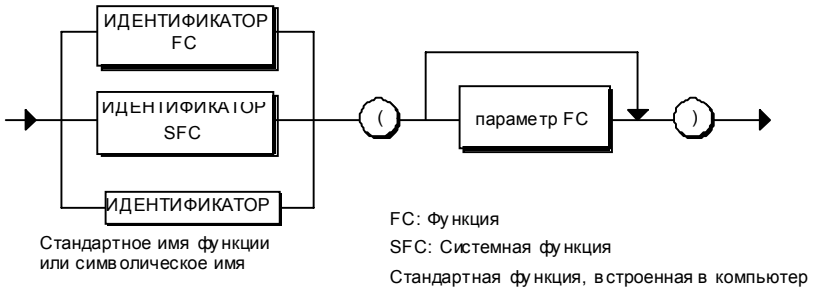
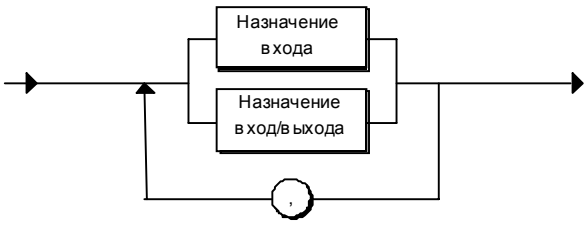
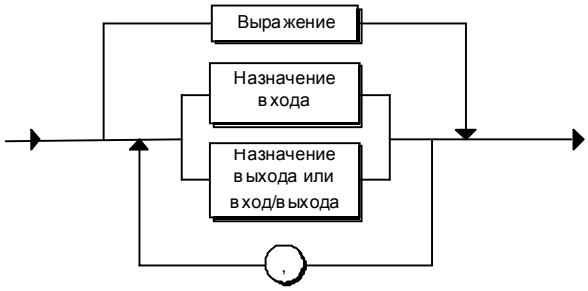
Правило	**Синтаксическая диаграмма**
Обыкновенная переменная	
Структурная переменная	

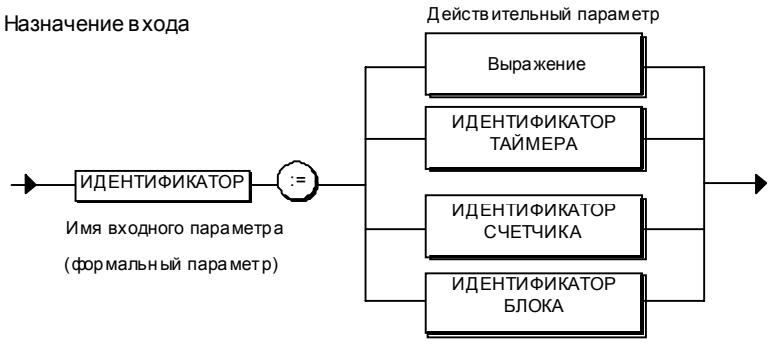
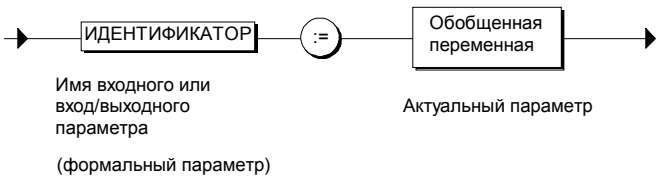
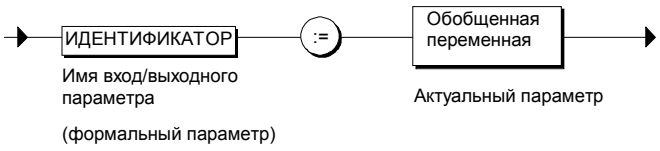
14.3.5 Назначения величин

Правило	Синтаксическая диаграмма
Выражение	
Простое выражение	
Простое умножение	

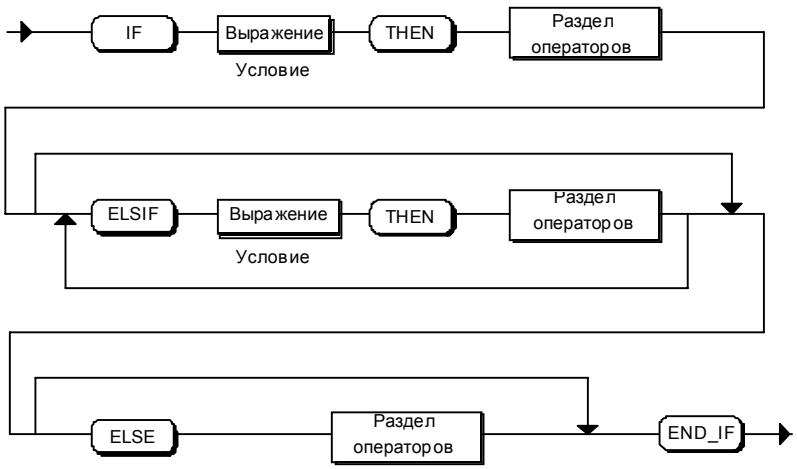
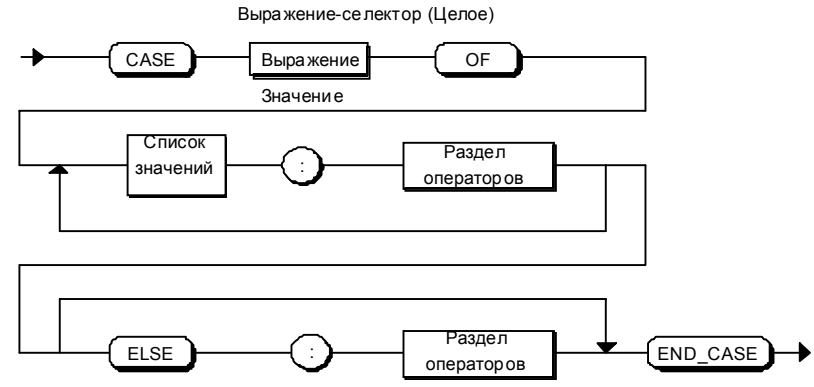
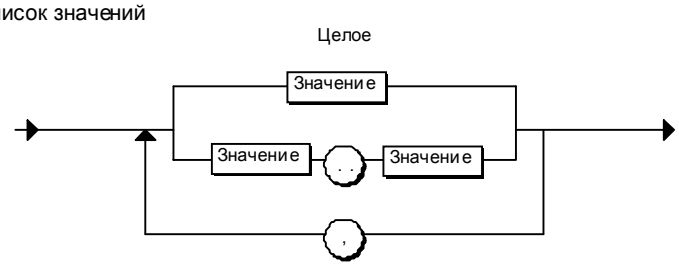
Правило	**Синтаксическая диаграмма**
Ад\рес	
Обобщенная переменная	
Константа	
Экспонента	
Основная логическая операция	
Основная арифметическая операция	
Действие сравнения	

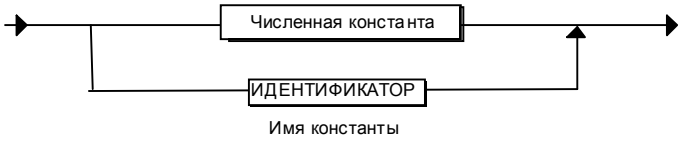
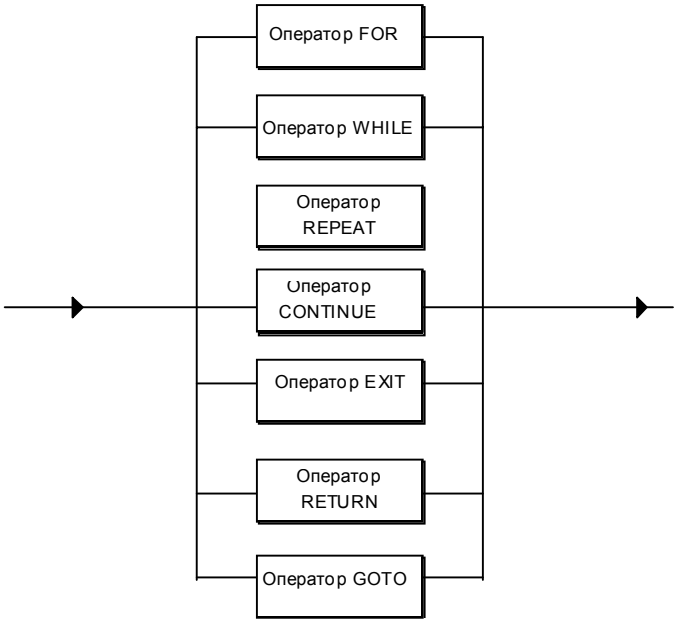
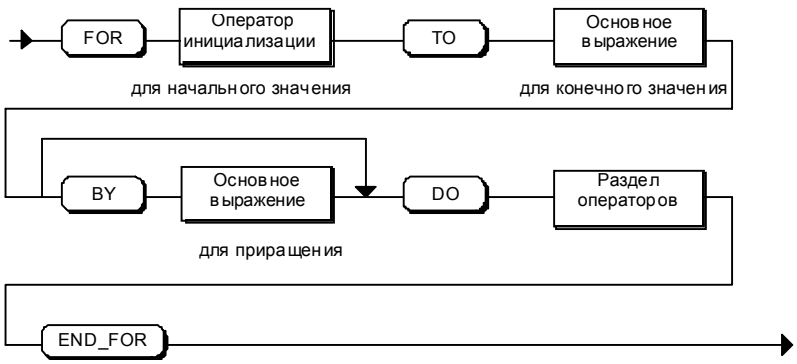
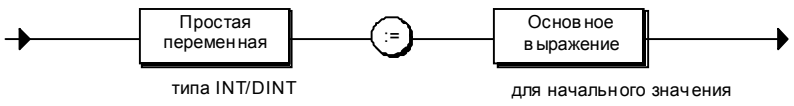
14.3.6 Вызов функций и функциональных блоков

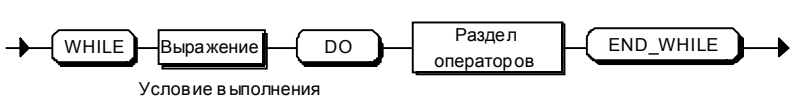
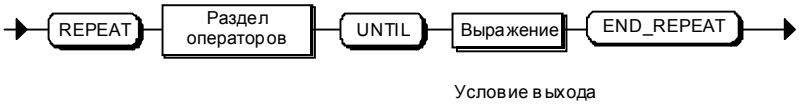
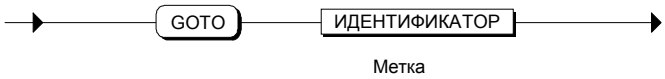
Правило	Синтаксическая диаграмма
Вызов FB	Вызов функционального блока 
Вызов функции	Вызов функции 
Параметры FB	Параметры FB 
Параметр FC	Параметр FC 

Правило	**Синтаксическая диаграмма**
Назначение входа	
Назначение выхода или входа/выхода	Назначение входов и вход/выходов 
Назначение входа/выхода	Назначение вход/выхода 

14.3.7 Операторы управления

Правило	Синтаксическая диаграмма
Оператор IF Помните, что ключевое слово END_IF должно заканчиваться точкой с запятой.	Оператор IF 
Оператор CASE Помните, что ключевое слово END_CASE должно заканчиваться точкой с запятой.	Оператор CASE 
Список значений	Список значений 

Правило	**Синтаксическая диаграмма**
Значение	
Итерация и оператор перехода	
Оператор FOR Помните, что ключевое слово END_FOR должно заканчиваться точкой с запятой.	Оператор FOR 
Оператор инициализации	Оператор инициализации 

Правило	**Синтаксическая диаграмма**
Оператор WHILE Помните, что ключевое слово END_WHILE должно заканчиваться точкой с запятой.	Оператор WHILE 
Оператор REPEAT Помните, что ключевое слово END_REPEAT должно заканчиваться точкой с запятой.	Оператор REPEAT 
Оператор CONTINUE	Оператор CONTINUE 
Оператор RETURN	Оператор RETURN 
Оператор EXIT	Оператор EXIT 
Программный переход	Оператор GOTO 

15 Полезные советы

Деление двух целых величин с результатом в формате REAL

Вы программируете следующее утверждение в SCL:

Fraction := Dividend/Divisor;

Когда Fraction – действительная величина, то Dividend и Divisor - целые.

Помните, что когда компилятор SCL обнаруживает такие действия, он выполняет преобразование неявного типа данных implicit и компилирует следующим образом:

Fraction := INT_TO_REAL(Dividend/Divisor);

Это значит, что деление всегда возвращает как результат округленную величину, например,1/3 = 0 или 3/2 = 1.

Время работы оптимального кода, когда доступная структура в блоке данных

Если Вам нужен неоднократный доступ к структуре в блоке данных, то рекомендуется следующий метод:

1. Создайте локальную переменную с типом структуры.
2. Присвойте структуре из блока данных значение переменной
3. Затем Вы можете использовать переменную неоднократно без доступа к DB.

Пример

DB100.array[i].value :=
DB100.array[i].value1 * DB100.array[i].value2 / DB100.array[i].value3 ;

Пример, приведенный ниже, потребует меньше памяти и короче по времени работы:

```
VAR_TEMP
  tmp : STRUCT
     value : REAL;
     value1 : REAL;
     value2 : REAL;
     value3 : REAL;
   END_STRUCT;
END_VAR
tmp := DB100.array[i];
DB100.array[i].value := tmp.value1 * tmp.value2 / tmp.value3;
```

Замечание

Используя VAR_TEMP Вы сохраняете переменные в стеке CPU. При маломощных CPU это может привести к переполнению стека. Следовательно, Вы должны умеренно использовать временные переменные!

Проблемы распределения L стека в малом CPU

Проблемы распределения являются следствием небольшого размера L стека CPU. В большинстве случаев проблема может решаться с помощью мер, очерченных ниже:

- Умеренное использование временных переменных (в VAR_TEMP или VAR разделах).
- Не декларировать переменные higher Data type и сводить количество переменных элементарного типа к минимуму.
- Использовать статические переменные:
  - Когда Ваша программа в FB, Вы можете использовать раздел VAR вместо VAR_TEMP.
  - Когда Ваша программа в OB или FC, используйте глобальный блок данных или область меркера.
- Избегайте сложных выражений. Когда выражение компилируется, компилятор сохраняет временные результаты в стеке. В зависимости от типа и количества временных результатов, доступный размер стека может быть превышен.
  Средство:
  Разбейте ваше выражение на несколько маленьких частей и назначьте временные результаты в переменные.

Выход действительных чисел во время контроля

Тест "Контроль" может выдать следующие образцы, когда отображаются непечатаемые действительные числа:

Величина	**Символ**
+ бесконечность	1.#INFrandom-digits
- бесконечность	-1.#INFrandom-digits
неопределенная	digit.#INDrandom-digits
NaN digit.	#NANrandom-digits

Отображение SCL программы в STL представлении

Вы можете открыть блок SCL с помощью редактора STL/LAD/FBD и отобразить команды MC7. Однако не следует корректировать блок в STL по следующим причинам:

- Отображаемая команда MC7не обязательно представляет правильный блок STL.
- Безошибочная компиляция STL требует значения обоих модификаций STL и SCL.
- Блок, откомпилированный STL, затем имеет значения идентификатора языка STL, он не длиннее идентификатора SCL.
- Исходный файл SCL и код MC7 уже не соответствуют друг другу.

Обработка меток времени интерфейса и кода

Если вы создаете новый блок (FB, FC и OB), интерфейс (параметры блока) и код имеют метки времени компиляции.

Если блок уже существует, у кода уже есть метка времени, когда он откомпилирован. Интерфейс может сохранять свою старую метку времени. Метка времени интерфейса изменяется только тогда, когда структура интерфейса модифицируется, другими словами:

- Метка времени сохраняется, если модификация выполняется в кодовом разделе, в атрибутах, в комментариях, в разделе VAR_TEMP (с FC также VAR) или в записи имен параметров или переменных. Это также относится к подключаемому (как мультиэкземпляры) интерфейсу.
- Метка времени интерфейса корректируется, когда изменяется тип данных или инициализация параметра или переменной, когда удалены или добавлены параметры, когда изменено название FB и когда включены мультиэкземпляры. Это также применимо к подключаемым (как мультиэкземпляры) интерфейсам.

Возвращаемая величина в стандарте STEP 7 и системные функции

Стандарт STEP 7 и системные функции имеют функциональное значение целого типа и содержат код ошибки. В справочнике для этих функций, возможные коды ошибок определены как константы WORD типа "W#16#8093".

S7 SCL – это язык, который строг относительно смешивания типов, так что INT и WORD не могут смешиваться. Следующее выражение, например, не приводит к требуемому результату.

```
IF SFCxx(..) = 16#8093 THEN ...
```

Однако Вы можете сообщить компилятору S7 SCL, что константа WORD должна считаться целой.

- Определенный тип константы. В этом случае, запрос делается так:
  `IF SFCxx(..) = INT#16#8093 THEN ...`
- Преобразование WORD_TO_INT(). Запрос формулируется так:
  `IF SFCxx(..) = WORD_TO_INT(16#8093) THEN ...`

Переконфигурация блоков

Вы больше не можете переконфигурировать блок вызова в блоках SCL, используя **Options > Rewire (Параметры > Переключить)** в SIMATIC Manager. Вы можете редактировать вызов в исходном файле SCL , воздействуя вручную.

Рекомендации:

- Определите символьные имена блоков в таблице символов и вызовите блоки, использующие их символьные имена.
- Определите символьные имена для абсолютных адресов (I, M, Q т.д.) в таблице символов и используйте символьные имена в Вашей программе.

Если позже Вы захотите переконфигурировать блок, Вам необходимо только изменить назначение в таблице символов, делать изменения в исходном файле SCL не требуется.

Словарь

А

Абсолютная адресация (Addressing, Absolute)

Абсолютная адресация непосредственно определяет ячейку памяти адреса, которая должна обрабатываться. Пример: адрес Q4.0 описывает бит 0 в байте 4 в области отображения выходов.

Адрес (Address)

Адрес - часть команды, определяющая данные, для которых должна быть выполнена операция. Возможны абсолютное и символическое определения адреса.

Адресация (Addressing)

Назначение позиции памяти в пользовательской программе. Позиции памяти могут быть назначены специфические адреса или области адреса (примеры: вход I 12.1, слово памяти MW25)

Атрибут (Attribute)

Атрибут является характеристикой, которая может применяется, например, к идентификатору блока или имени переменной. В SCL есть атрибуты для следующих пунктов информации: заголовка блока, номер версии, защиты блока, автора, имени блока, семейства блоков.

Б

Блок (Block)

Блоки являются субмодулями программы пользователя и различаются их функцией, их структурой или их целью. В STEP 7 есть логические блоки (FB, FC, OB, SFC и SFB), блоки данных (DB и SDB) и определенный пользователем тип данных (UDT).

Блок данных (Data Block) (DB)

Блоки Данных являются блоками, содержащими данные и параметры, с которыми работает программа пользователя. В отличие от всех других типов блоков, они не содержат никаких команд.

В

Вид (View)

Чтобы получить доступ к объявленной переменной с другим, альтернативным типом данных, Вы можете определить виды переменной или областей в пределах переменных. Вид может использоваться подобно любой другой переменной в блоке. Он наследует все особенности переменной, на которую он ссылается, только тип данных - новый.

Возвращаемая величина (RET_VAL) (Return Value (RET_VAL))

В противоположность функциональным блокам, функции возвращают результат, известный как возвращаемая величина.

Время цикла (Scan Cycle Time)

Время цикла - время требующееся ПЛК, чтобы однократно выполнить программу пользователя.

Вход/выходной параметр (In/Out Parameter)

Вход/выходные параметры существуют в функциях и функциональных блоках. Они используются, чтобы передать данные в вызываемый блок, где они обрабатываются, и возвращать результат в исходную переменную из вызванного блока.

Входной параметр (Input Parameters)

Входные параметры существуют только в функциях и функциональных блоках. Входные параметры используются, чтобы передать в вызванный блок данные для обработки.

Вызов блока (Block Call)

Вызов блока запускает блок в программе пользователя STEP 7. Организационные блоки вызываются только операционной системой; все другие блоки вызываются программой пользователя STEP 7.

Выражение (Expression)

Выражение в SCL - средство обработки данные. Различаются выражения арифметические, логические и выражения сравнения.

Выходной параметр (Output Parameter)

Выходные параметры блока в программе пользователя используются, чтобы вернуть результаты после вызова блока.

Г

Глобальные данные (Shared Data)

Глобальными являются данные, которые могут быть доступны любым логическим блоком (FC, FB или OB). В частности они включают битовую память (M), входы (I), выходы (O), таймеры, счетчики и элементы блоков данных (DB). Глобальные данные могут адресоваться в абсолютных или символических условиях.

Д

Данные, статические (Data, Static)

Статические данные являются локальными данными функционального блока, которые загружены в экземплярный блок данных и следовательно сохранены до следующего вызова функционального блока.

Данные, временные (Data, Temporary)

Временные данные -локальные данные блока, которые вводятся в локальный стек (стек L) на время выполнения блока. После выполнения блока эти данные не сохраняются.

Данные пользователя (User Data)

Данные пользователя осуществляют обмен между CPU и сигнальными, функциональными или коммуникационными модулями через область отображения процесса или прямым доступом. Примеры данных пользователя: цифровые или аналоговые входные и выходные сигналы сигнальных модулей, управляющие и контрольные сигналы функциональных модулей.

Двоично-десятичный код (BCD)

Двоично-десятичный код. В STEP 7, внутренний код таймеров и счетчиков в CPU – только в формате BCD.

Действительное число (Real Number)

Действительное число является положительным или отрицательным числом, представляя десятичную дробь, например 0.339 или -11.1.

З

Загрузка (Download)

Передача загружаемых объектов (например, логических блоков) из устройства программирования в загрузочную память CPU.

Защита блока (Block Protection)

Используя защиту блока, Вы можете защитить отдельные блоки от декомпиляции. Вы включаете эту защиту, назначая ключевое слово "KNOW_HOW_PROTECTED" при компиляции исходного файла блока.

И

Идентификатор (Identifier)

Комбинация букв, цифр и символа подчеркивания, которые идентифицируют языковой элемент.

Идентификатор адреса (Address Identifier)

Идентификатор адреса - часть адреса, которая содержит детальную информацию об области памяти, в которой команда получает доступ к переменной (данным), с которой он должен выполнить операцию или величину переменной (данные) с которой он должен выполнить операцию. В команде "Value := IB10", "IB" - идентификатор адреса ("I" указывает на область памяти для входов, а "B" указывает, что в этой области берется один байт).

Иерархия вызовов (Call Hierarchy)

Для выполнения блоков они должны быть вызваны. Порядок и вложенность последовательности вызовов блоков называется иерархией вызовов.

Интерфейс вызова (Call Interface)

Интерфейс вызова определяется входными, выходными и вход-выходными параметрами (формальные параметры) блока в пользовательской программе STEP 7. Когда вызывается блок, эти параметры заменяются фактическими параметрами

Исходный файл (Source File)

Часть программы, созданная графическим или текстовым редактором, из которой может быть скомпилирована исполняемая программа.

Исходный файл SCL (SCL Source File)

Исходный файл SCL является файлом, в котором программа записана на языке SCL. Исходный файл SCL позже переводится в машинный код компилятором SCL.

К

Класс блоков (Block Class)

Согласно типу информации, которую они содержат, блоки подразделяются в следующие два класса: логические блоки и блоки данных.

Ключевое слово (Keyword)

Зарезервированное слово специальный языковой элемент, например, "IF".

Ключевые слова используются в SCL, чтобы выделить начало блока, чтобы выделить подразделы в разделе декларации и идентифицировать команды. Они также используются для атрибутов и комментариев.

Команда CASE (CASE Statement)

Эта команда обеспечивает выбор ветви алгоритма. Она выбирает из набора n веток определенную программную ветку, основываясь на величине выражения выбора.

Команда CONTINUE (CONTINUE Statement)

Команда CONTINUE используется в SCL, чтобы завершить выполнение текущей итерации команды цикла (FOR, WHILE или REPEAT).

Команда EXIT (Statement EXIT)

Языковая конструкция в пределах программы пользователя для выхода из цикла в любой точке независимо от условий.

Команда FOR (Statement FOR)

Языковая конструкция в пределах программы. Команда FOR используется, чтобы выполнить последовательность команд в цикле пока управляющая переменная непрерывно назначена величины.

Компилятор SCL (SCL Compiler)

Компилятор SCL - пакетный компилятор, который используется, чтобы перевести программу, написанную в текстовом редакторе (исходный файл SCL) в машинный код M7. Скомпилированные блоки хранятся в папке "Blocks" программы S7.

Команда GOTO (GOTO Statement)

Языковая конструкция в пределах программы. Команда GOTO заставляет программу непосредственно перейти к определенной метке и следовательно к другой команде в пределах одного блока.

Команда REPEAT (REPEAT Statement)

Языковая конструкция программы пользователя для повторения последовательности команд до тех пор, пока не достигнуто условие завершения.

Команда RETURN (RETURN Statement)

Языковая конструкция в пределах программы, с которой Вы можете выйти из текущего блока.

Комментарии (Comments)

Языковая конструкция, позволяющая включить в программу пояснительный текст, не влияющий на ее выполнение.

Комментарий блока (Block Comment)

Вы можете ввести дополнительную информацию о блоке (например, чтобы описать автоматизированный процесс). Эти комментарии не загружаются в рабочую память программируемых контроллеров SIMATIC S7.

Компиляция (Compilation)

Процесс генерации из исходного файла исполняемой программы пользователя.

Компиляция, ориентированная на источник (Compilation, Source-Oriented)

При исходном-ориентированном вводе, исходник компилируется в исполняемую программу пользователя только тогда, когда введены все команды. Компилятор проверяет ошибки ввода.

Константа (Constant)

Заменители постоянных величин в логических блоках. Константы используются для улучшения удобочитаемости программы. Например: Вместо определения величины (например, 10) определен заменитель "Max_loop_iterations". Когда вызывается блок, величина константы (например, 10) заменяет заменитель.

Контрольное время цикла (Scan Cycle Monitoring Time)

Если время, необходимое для выполнения программа пользователя превышает контрольное время цикла, операционная система генерирует сообщение ошибки и переключает ПЛК в режим STOP.

Контрольная точка (Breakpoint)

Эта функция может использоваться, чтобы переключить CPU в режим HOLD в заданных точках программы. Когда программа достигает контрольной точки, становятся доступными такие отладочные функции как пошаговое выполнение программы с возможность наблюдения и управления переменными.

Л

Лексическое правило (Lexical Rule)

Нижний уровень правил в формальном языковом описании SCL состоит из лексических правил. Они не допускают гибкое форматирование; другими словами, не разрешено дополнение пробелами и управляющими символами.

Литерал (Literal)

Формальная нотация, которая определяет величину и тип константы.

Локальные данные (Local Data)

Локальные данные являются данными отдельного логического блока, которые объявляются в своих разделах деклараций или в своих описаниях переменных. В зависимости от конкретного блока, он состоит из формальных параметров, статических и временных данных.

Логический блок (Logic Block)

Логический блок в SIMATIC S7 - блок, который содержит раздел программы пользователя STEP 7. В отличие от этого блок данных содержит только данные. Есть следующие типы логических блоков: организационные блоки (OB), функциональные блоки (FB), функции (FC), системные функциональные блоки (SFB) и системные функции (SFC).

M

Массив (Array)

Массив является сложным типом данных, состоящим из множества однотипных элементов. Эти элементы данных могут быть, в свою очередь, элементарными или сложными.

Меркеры Bit Memory (M)

Область системной памяти в CPU SIMATIC S7. Эта область доступна для записи или чтения в формате бит, байт, слово и двойное слово. Битовая область памяти может использоваться, чтобы загрузить временные результаты.

Мнемоника (Mnemonics)

Мнемоника - краткое представление адресов и программирования операций в программе. STEP 7 поддерживает английское (в котором, например, "I" - представляемый ввод) и немецкое представление (где, например, представление входа - "E" (Eingang по-немецки)).

Мониторинг (Monitoring)

Используя мониторинг программы, Вы можете проверить, как программа выполняется в CPU. При мониторинге, например, отображаются имена и фактические значения переменных и параметров в хронологическом порядке и обновляются циклически.

Мультиэкземплярные блоки (Multiple Instance)

Когда используются мультиэкземплярные блоки, блок данных экземпляра содержит данные для ряда функциональных блоков в пределах иерархии вызовов.

N

Назначение (Assignment)

Механизм для связывания величины с переменной.

Начальное значение (Initial Value)

Величина присвоенная переменной при запуске системы.

Нетерминальное выражение (Non Term)

Нетерминальное выражение является сложным элементом в синтаксическом описании, который описан другим лексическим или синтаксическим правилом.

O

Область отображения входов (PII) (Process-Image Input Table (PII))

Перед выполнением программы пользователя изображение процесса входовв читается операционной системой из входных модулей.

Область отображения выходов (PIQ) (Process-Image Output Table (PIQ))

В конце программы пользователя отображение выходов процесса передается операционной системой в выходные модули.

Объявление (Declaration)

Механизм для определения языкового элемента. Декларация включает связь идентификатора с элементом языка назначением атрибутов и типов данных.

Объявление переменной: (Variable Declaration)

Объявление переменной содержит спецификацию символьного имени, типа данных и, если требуется, их начальное значение и комментарий.

Области памяти (Memory Area)

CPU S7 имеет три области памяти: загрузочная, рабочая и системная области.

Оператор (Statement)

Оператор является минимальным неделимым элементом программы пользователя записанным на языке, ориентированным на текст. Оно представляет команду процессору для выполнения конкретного действия.

Операция (Operation)

Операция - часть команды, определяющая действие, которое должен выполнить процессор.

Организационный блок (Organization Block (OB))

Организационный блок - форма интерфейса между операционной системой CPU S7 и программой пользователя. Организационный блок определяет последовательность, в которой выполняются блоки программы пользователя.

Отладчик SCL (SCL Debugger)

Отладчик SCL - отладчик языка высокого уровня, используемый для обнаружения логических ошибок программирования в программах пользователя созданных на SCL.

Отображение процесса (Process Image)

Сигнальные состояния дискретных входных и выходных модулей хранятся в CPU в областях отображений процесса. Это две отдельные области отображения: входов (PII) и выходов (PIQ).

П

Параметрический тип (Parameter Type)

Параметрический тип является специальным типом данных для таймеров, счетчиков и блоков. Он может использоваться для описания входных параметров функциональных блоков и функций, и для вход/выходных параметров функциональных блоков только для передачи таймеров и счетчиков в вызываемый блок.

Переменная (Variable)

Переменные - это элемент данных с переменным содержанием, которые могут использоваться в пользовательской программе STEP 7. Переменная характеризуется адресом (например, M3.1) и типом данных, (например, BOOL), и могут обозначаться символическим именем (например, TAPE_ON). Переменные описываются в разделе объявлений.

Пользовательская программа S7 (S7 User Program)

Папка блоков загружаются в программируемый модуль S7 (например ПЛК или FM) и может выполняться в модуле как часть программы управления системой или процессом.

Помощь (Online Help)

Работая с программным обеспечением STEP 7, Вы можете вызвать на экран контекстную подсказку.

Пошаговая отладка (Single Step)

Пошаговая отладка способ отладки управления, выполняемый отладчиком SCL. При этом способе отладки, Вы можете выполнить программу одна команда за другой, рассматривая результаты каждого шага.

Преобразование типов данных (Data Type Conversion)

Преобразование типов данных необходимо, когда фв операции участвуют в две переменных различных типов.

Программа пользователя (User Program)

Программа пользователя содержит все команды, декларации и данные, необходимые для обработки сигналов и управления установками или технологическим процессом. Программа назначается программируемому модулю (например, CPU, FM) и может быть структурирована в виде меньших модулей (блоков.)

Проект (Project)

Папка для хранения всех объектов, имеющих отношение к конкретному решению автоматизации независимо от количества станций, модулей или их отношения к сетевой структуре.

P

Раздел объявлений (Declaration Section)

Объявление переменной блока разделяется на различные разделы для объявления различных блочных параметров. Раздел декларации IN содержит, например, объявления входных параметров, раздел объявления OUT содержит объявления выходных параметров.

Разрешение (Enable) (EN)

Каждый функциональный блок и каждая функция в STEP 7 имеет неявно определенный входной параметр "Enable (разрешение)" (EN), который может использоваться при вызове блока. Если EN = TRUE, вызванный блок выполняется. В противном случае он не выполняется.

Разрешение выхода (Enable Output) (ENO)

Каждый блок в STEP 7 имеет параметр "Enable Output (разрешение выхода)" (ENO). Когда выполнение блока завершено текущая величина флага OK устанавливается в ENO. Сразу после вызова блока Вы можете проверить величину ENO, чтобы увидеть, все ли операции в блоке выполнены правильно или при выполнении происходили ошибки.

Редактор SCL (SCL Editor)

Редактор SCL (SCL Editor) - текстовый редактор, специально разработанный для работы с SCL, в котором создаются исходные файлы SCL.

Режим HOLD

CPU переходит в состояние HOLD из режима RUN по команде программатора. В этом режиме возможны специальные тестирующие функции.

Режим Offline

Режим работы, в котором устройство программирования не связано (физически или логически) с ПЛК.

Режим Online

Режим работы, в котором устройство программирования связано (физически или логически) с ПЛК.

Режим RUN

В режиме RUN выполняется программа пользователя и циклически обновляется образ процесса. Все цифровые выходы разрешены.

Режим RUN-P

Режим RUN-P – такой же рабочий режим как и режим RUN, кроме того в режиме RUN-P разрешаются без ограничения все функции программатора.

C

Семантика (Semantics)

Отношение между символическими элементами языка программирования и их значением, интерпретацией и применением.

Символ (Symbol)

Символ является определенным потребителем именем, которое должно соответствовать определенным синтаксическим правилам. Это имя может использоваться при программировании, управлении и мониторинге, как только Вы определите его (например, как переменная, тип данных, метка перехода, или блок). Пример: Адрес: I5.0, тип данных: Bool, символ: Emer_Off_Switch

Символическая адресация (Addressing, Symbolic)

При использовании символической адресации, адреса операндов вводятся как символ и не как конкретный адрес. Соответствие символа и конкретного адреса устанавливается в таблице идентификаторов или при использовании файла символики.

Символическая константа (Constant, symbolic)

Константы с символическими именами - -заменители для постоянных величин в логических блоках. Символические константы используются для улучшения удобочитаемости программы.

Символьное программирование (Programming, Symbolic)

Язык программирования SCL позволяет Вам использовать символьные строки вместо адресов: Например, адрес Q1.1 может заменяться "valve_17". Таблица символов создает связь между адресом и назначенной символической строкой.

Синтаксическое правило (Syntax Rule)

Наиболее высокий уровень правил в формальном описании языка SCL состоит из синтаксических правил. Они допускают гибкое форматирование; другими словами, могут быть добавлены пробелы и управляющие символы.

Системный блок данных (SDB) (System Data Block (SDB))

Системные блоки данных являются областями данных в S7 CPU, которые содержат системные установочные параметры и системные параметры модулей. Системные блоки данных создаются и редактируются, с использованием стандартного программного обеспечение STEP 7.

Системная функция (SFC) (System Function (SFC))

Системная функция (SFC) является встроенной функцией операционной системы CPU, которая может при необходимости вызываться пользовательской программой STEP 7.

Системный функциональный блок (SFB) (System Function Block (SFB))

Системный функциональный блок (SFB) является функциональным блоком встроенным в операционную систему CPU, который может при необходимости вызываться пользовательской программой STEP 7.

Системная область памяти (System Memory (System Area))

Системная память встраивается в CPU S7 как RAM. В системной памяти хранятся адресные области (таймеры, счетчики, битовая память и т.п.) и области данных, непосредственно используемые операционной системой (например, копии данных для связи).

Слово состояния (Status Word)

Слово состояния является одним из регистров CPU. Слово состояния содержит информацию о состоянии и информацию о ошибках в связи с

обработкой команд STEP 7. Биты состояния могут быть прочитаны и записаны программистом. Биты ошибки могут только быть прочитаны.

Структура (STRUCT) (Structure (STRUCT))

Сложный тип данных, состоящий из любых элементов данных типов других данных. Типы данных в пределах структур могут быть элементарного или другого сложного типа.

Структурное программирование (Programming, Structured)

Для того, чтобы облегчить решение сложных задач автоматизации, программа пользователя подразделяется на отдельные, самостоятельные модули (блоки). Разбиение программы пользователя базируется на функциональных соображениях или технологической структуре системы.

Счетчик (Counter)

Счетчики являются компонентами системной памяти CPU. Содержимое счетчика корректируются программой пользователя. Команды STEP 7 используются, чтобы определить точную функцию счетчика (например, счет на увеличение) и выполнить их (например, запустить).

T

Таблица переменных (Variable Table)

Таблица переменных включает переменные, которые мы хотим наблюдать и модифицировать, и формат их представления.

Таблица символов (Symbol Table)

Таблица используется, чтобы назначить символы (или символические имена) адресам глобальных данных и блоков. Примеры: Emer_Off (Символ), I1.7 (Address), Controller (Символ), SFB24 (Блок)

Таймеры (Timers)

Таймеры являются компонентами системной памяти CPU. Содержание таймеров корректируются операционной системой в пользовательской программе асинхронно. Вы можете использовать команды STEP 7, чтобы определить точно функцию таймера (например, таймер задержки включения) и запуск таймера (Start).

Терм (Term)

Терм - основной элемент лексического или синтаксического правила, он не может раскрываться через другие правила, но представлен литерами языка. Терм может быть ключевым словом или даже единственным символом.

Тип блока (Block Type)

Блочная архитектура STEP 7 включает следующие типы блоков: организационные блоки, функциональные блоки, функции, блоки данных, а также системные функциональные блоки, системные функции, системные блоки данных и тип данных, определенный пользователем.

Тип данных (Data Type)

Типы данных определяют:

- Тип и интерпретацию элементов данных
- Выделенная память и диапазон значений элементов данных
- Комплект операций, которые могут выполняться с адресом типа данных
- Нотация элементов данных

Тип данных, определенный пользователем (Data Type, User-defined)

Типы данных, определенные пользователем (UDT), являются типами данных, которые Вы можете создать используя описание типа данных. Каждый пользовательский тип данных имеет уникальное имя и может быть использован любое число раз. Тип данных, определенный пользователем, полезен для генерации ряда блоков данных с одной и той же структурой (например, контроллер).

Тип данных, сложный (Data Type, Complex)

Сложные типы данных состоят из элементов данных элементарных типов. Различают структуры и массивы. Типы данных STRING и DATE_AND_TIME - также сложные типы.

Тип данных, элементарный (Data Type, Elementary)

Элементарные типы данных - встроенные типы данных в соответствии с IEC 1131-3. Примеры: тип данных "BOOL" определяет двоичную переменную ("бит"); тип данных "INT" определяет 16-битную целую переменную

Ф

Фактический параметр (Actual Parameter)

Фактический параметр заменяет формальный параметр при вызове функционального блока (FB) или функции (FC).

Пример: формальный параметр "Start" заменяется фактическим параметром "I3.6".

Флаг OK (OK Flag)

Флаг OK используется, чтобы указать правильно или неправильно выполняется последовательность команд в блоке. Это - глобальная переменная типа BOOL.

Формальный параметр (Formal Parameter)

Формальный параметр является заменителем фактического параметра в перестраиваемых логических блоках. Для FB и FC формальные параметры объявляются программистом, в случае SFB и SFC они уже существуют. Когда вызывается блок, формальным параметрам назначаются фактические параметры, в результате чего вызванный блок работает с фактическими величинами. Формальные параметры рассматриваются как локальные блочные данные и подразделяются на входные, выходные и вход-выходные параметры.

Функция (Function (FC))

A function allows you to pass parameters in the user program, which means they are suitable for programming complex functions that are required frequently, for example, calculations.

Согласно стандарту International Electrotechnical Commission (Международная электротехническая комиссия) IEC 1131-3, функции являются логическими блоками, которые не имеют статических данные. Функция позволяет Вам передавать параметры в программу пользователя, в которой программируются сложных и часто используемые вычисления вычисления.

Функциональный блок (Function Block (FB))

Согласно Международному Electrotechnical Commission IEC 1131-3 стандарту, функциональные блоки являются логическими блоками со статическими данными (Данные, Статические). Поскольку FB имеет "память" (экземплярный блок данных), возможно иметь доступ к ее параметрам (например, выходам) в любое время и в любой точке в программе пользователя.

Ц

Целое (Integer (INT))

Целое (INT) является одним из элементарных типов данных. Величины являются всеми целыми 16-битными числами.

Э

Экземпляр (Instance)

Термин "экземпляр" имеет отношение к вызову функционального блока. Вызываемому функциональный блок назначен экземплярный блок данных или локальный экземпляр. Если функциональный блок в программе пользователя STEP 7 назван время n раз, всякий раз, когда, используя другие параметры и другой экземплярный блок данных, получим n экземпляров.

Экземплярный блок данных (Instance Data Block (Instance DB))

Экземплярный блок данных хранит формальные параметры и статические локальные данные функционального блока. Экземплярный блок данных может назначаться при вызове FB или функциональной блочной иерархии вызова.

Язык SCL

Язык высокого уровня, основанный на PASCAL, который соответствует стандарту DIN EN-61131-3 (международный IEC 1131-3) и используется, чтобы запрограммировать на ПЛК сложные операции, например, алгоритмы и задачи обработки данных. Сокращение от " Structured Control Language (Структурный управляющий язык)".

Указатель