ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р МЭК 61131-3— 2016

КОНТРОЛЛЕРЫ ПРОГРАММИРУЕМЫЕ

Часть 3

Языки программирования

(IEC 61131-3:2013, IDT)

Издание официальное

Москва
Стандартинформ
2016

Предисловие

1 ПОДГОТОВЛЕН Негосударственным образовательным частным учреждением «Новая Инженерная Школа» (НОЧУ «НИШ») на основе перевода на русский язык англоязычной версии указанного в пункте 4 стандарта, который выполнен Российской комиссией экспертов МЭК/ТК 65, и Федеральным государственным унитарным предприятием «Всероссийский научно-исследовательский институт стандартизации и сертификации в машиностроении» («ВНИИНМАШ»)

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 306 «Измерения и управление в промышленных процессах»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 13 мая 2016 г. № 313-ст

4 Настоящий стандарт идентичен международному стандарту МЭК 61131-3:2013 «Контроллеры программируемые. Часть 3. Языки программирования (IEC 61131-3:2013, «Programmable controllers — Part 3: Programming languages», IDT).

При применении настоящего стандарта рекомендуется использовать вместо ссылочных международных стандартов соответствующие им национальные стандарты, сведения о которых приведены в дополнительном приложении ДА

5 В настоящем стандарте часть его содержания может быть объектом патентных прав

6 ВВЕДЕН ВПЕРВЫЕ

Правила применения настоящего стандарта установлены в ГОСТ Р 1.0—2012 (раздел 8). Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (www.gost.ru)

Содержание

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

КОНТРОЛЛЕРЫ ПРОГРАММИРУЕМЫЕ

Часть 3

Языки программирования

Programmable controllers. Part 3. Programming languages

Дата введения — 2017—04—01

1 Область применения

Настоящий стандарт устанавливает синтаксис и семантику языков программирования программируемых контроллеров, определенных в МЭК 61131 (часть 1).

Функции ввода программы, тестирования, мониторинга, операционной системы и т. п. определены в МЭК 61131 (часть 1).

Настоящий стандарт устанавливает синтаксис и семантику унифицированного набора языков программирования для программируемых контроллеров (PC). Данный набор состоит из двух текстовых языков программирования, списка инструкций (IL) и структурированного текста (ST), и двух графических языков, релейно-контактных схем (LD) и функциональных блоковых диаграмм (FBD).

Дополнительный набор графических и эквивалентных текстовых элементов, именуемый последовательная функциональная схема (SFC), определяется для структурирования внутренней организации программ и функциональных блоков программируемого контроллера. Определены также элементы конфигурации, поддерживающие установку программ программируемого контроллера в системы программируемого контроллера. Кроме того, определены средства, облегчающие взаимодействие между программируемыми контроллерами и другими компонентами автоматизированных систем.

2 Нормативные ссылки

В настоящем стандарте использованы нормативные ссылки на следующие стандарты. Для недатированных ссылок применяют последнее издание ссылочного документа (включая изменения).

МЭК 61131-1 Программируемые контроллеры. Часть 1. Общие положения (IEC 61131-1, Programmable controllers — Part 1: General information)

МЭК 61131-5 Программируемые контроллеры. Часть 5. Взаимодействия (IEC 61131-5, Programmable controllers — Part 5: Communications)

ИСО/МЭК 10646:2012 Информационная технология. Универсальный набор символов (UCS) (ISO/IEC 10646:2012, Information technology — Universal Coded Character Set (UCS)

ИСО/МЭК/IEEE 60559 Информационная технология. Микропроцессорные системы. Арифметика с плавающей точкой (ISO/IEC/IEEE 60559, Information technology — Microprocessor Systems — Floating-Point arithmetic)

Издание официальное

3 Термины и определения

В настоящем стандарте применены следующие термины с соответствующими определениями:

3.1 **абсолютное время** (absolute time): Комбинация времени суток и даты.

3.2 **путь доступа** (access path): Связь символического имени с переменной для реализации открытого взаимодействия.

3.3 **действие** (action): Логическая переменная или набор подлежащих выполнению операций вместе со связанной управляющей конструкцией.

3.4 **блок действий** (action block): Элемент графического языка, который использует входную логическую переменную для определения значения выходной логической переменной или разрешающее условие для действия в соответствии с предопределенной управляющей конструкцией.

3.5

агрегат (aggregate): Структурированная совокупность объектов данных, образующая тип данных.

[ИСТОЧНИК: ISO/AFNOR:1989]

3.6

массив (array): Агрегат, состоящий из объектов данных с идентичными атрибутами, на каждый из объектов данных можно уникально ссылаться с помощью индекса.

[ИСТОЧНИК: ISO/AFNOR:1989]

3.7

присваивание (assignment): Механизм для придания значения переменной или агрегату.

[ИСТОЧНИК: ISO/AFNOR:1989]

3.8 **базовый тип** (base type): Тип данных, тип функционального блока или класс, из которых наследуются или производятся дальнейшие типы.

3.9 **число с основанием** (based number): Число, представленное с конкретным основанием, отличным от 10.

3.10 **двоично-десятичный код** (binary coded decimal; BCD): Код десятичного числа, в котором каждая цифра представлена ее двоичным значением.

3.11 **бистабильный функциональный блок** (bistable function block): Функциональный блок с двумя устойчивыми состояниями, управляемый одним или более входами.

3.12 **битовая строка** (bit string): Элемент данных, состоящий из одного или более битов.

3.13 **битово-строковый литерал** (bit string literal): Литерал, который прямо представляет значение битовой строки типов данных BOOL, BYTE, WORD, DWORD или LWORD.

3.14 **тело** (body): Набор операций программного компонента.

3.15 **вызов** (call): Языковая конструкция, вызывающая выполнение функции, функционального блока или метода.

3.16 **строка символов** (character string): Агрегат, состоящий из упорядоченной последовательности символов.

3.17 **символьно-строковый литерал** (character string literal): Литерал, прямо представляющий значение символа или строки символов типов данных CHAR, WCHAR, STRING или WSTRING.

3.18 **класс** (class): программный компонент, состоящий из:

- определения структуры данных;
- набора методов, выполняемых над структурой данных.

3.19

комментарий (comment): Языковая конструкция для включения текста, не влияющего на выполнение программы.

[ИСТОЧНИК: ISO/AFNOR:1989]

3.20 **конфигурация** (configuration): Элемент языка, соответствующий системе программируемого контроллера.

3.21 **константа** (constant): Элемент языка, указывающий на элемент данных с фиксированным значением.

3.22 **функциональный блок счетчика** (counter function block): Функциональный блок, который накапливает значение числа изменений, определяемых на одном или более указанных выходов.

3.23

тип данных (data type): Набор значений вместе с набором допустимых операций.
[ИСТОЧНИК: ISO/AFNOR:1989]

3.24 **дата и время** (date and time): Дата с годом и время суток, представленные как отдельный элемент данных.

3.25 **объявление** (declaration): Механизм для определения элемента языка.

3.26 **разделитель** (delimiter): Символ или комбинация символов, используемая для разделения элементов языка программирования.

3.27 **производный класс** (derived class): Класс, создаваемый наследованием из другого класса.

Примечание — Производный класс также называют расширенным классом или порожденным классом.

3.28 **производный тип данных** (derived data type): Тип данных, созданный с использованием другого типа данных.

3.29 **производный тип функционального блока** (derived function block type): Тип функционального блока, созданный наследованием из другого типа функционального блока.

3.30 **прямое представление** (direct representation): Средства представления переменной в программе программируемого контроллера, из которых может быть прямо определено физическое или логическое расположение переменной.

3.31 **двойное слово** (double word): Элемент данных, содержащий 32 бита.

3.32 **динамическое связывание** (dynamic binding): Ситуация, в которой экземпляр вызова метода возвращается во время выполнения в соответствии с фактическим типом экземпляра или интерфейса.

3.33 **оценка** (evaluation): Процесс установления значения выражения, функции, выходных переменных сети или экземпляра функционального блока во время выполнения программы.

3.34 **элемент управления выполнением** (execution control element): Элемент языка, контролирующий поток выполнения программы.

3.35 **задний фронт** (falling edge): Часть временной диаграммы сигнала, где происходит переход логической переменной из 1 в 0.

3.36 **функция** (function): Элемент языка, который во время выполнения обычно вырабатывает результат в виде одного элемента данных и, возможно, дополнительные выходные переменные.

3.37 **экземпляр функционального блока** (function block instance): Экземпляр типа функционального блока.

3.38 **тип функционального блока** (function block type): Элемент языка, состоящий из:

- определения структуры данных, разделенной на входные, выходные и внутренние переменные; и
- набора операций или набора методов, выполняемых над элементами структуры данных при вызове типа функционального блока.

3.39 **функциональная блоковая диаграмма** (function block diagram): Сеть, узлы которой являются экземплярами функциональных блоков, графически представленные функции или вызовы метода, переменные, литералы и метки.

3.40 **родовой тип данных** (generic data type): Тип данных, представляющий более одного типа данных.

3.41 **глобальная переменная** (global variable): Переменная с глобальной областью действия.

3.42 **иерархическая адресация** (hierarchical addressing): Прямое представление элемента данных как члена физической или логической иерархии.

Пример — Точка в модуле, который хранится на стеллаже, который, в свою очередь, помещен в стенд и т. д.

3.43 **идентификатор** (identifier): Комбинация букв, цифр и символов подчеркивания, которая начинается с буквы или символа подчеркивания и которая именует элемент языка.

3.44 **реализация** (implementation): Версия программируемого логического контроллера (PLC) или программного или отладочного инструмента, предоставленная разработчиком.

3.45 **разработчик** (Implementer): Изготовитель PLC или программного или отладочного инструмента, предоставленного пользователю для разработки приложений PLC.

3.46 **наследование** (inheritance): Создание нового класса, типа функционального блока или интерфейса на основе существующего класса, типа функционального блока или интерфейса, соответственно.

3.47 **начальное значение** (initial value): Значение, присвоенное переменной при запуске системе.

3.48 **входная и выходная переменная** (in-out variable): Переменная, используемая для передачи значения программному компоненту и, дополнительно, для возврата значения из программного компонента.

3.49 **входная переменная** (input variable): Переменная, используемая для передачи значения программному компоненту, отличному от класса.

3.50 **экземпляр** (instance): Отдельная, именованная копия структуры данных, связанная с типом функционального блока, классом или программным типом, которая сохраняет свои значения от одного вызова соответствующей операции до другого.

3.51 **имя экземпляра** (instance name): Идентификатор, связанный с конкретным экземпляром.

3.52 **создание экземпляра** (instantiation): Создание экземпляра.

3.53 **целое число** (nteger): Целое число, которое может содержать положительные, нулевые или отрицательные значения.

3.54 **целый литерал** (integer literal): Литерал, прямо представляющий целое значение.

3.55 **интерфейс** (interface): Элемент языка в контексте объектно-ориентированного программирования, содержащий набор прототипов метода.

3.56 **ключевое слово** (keyword): Лексическая единица, которая характеризует элемент языка.

3.57 **метка** (label): Конструкция языка, именующая инструкцию, сеть или группу сетей, включая идентификатор.

3.58 **элемент языка** (language element): Любая единица, идентифицированная символом в левой части порождающего правила в формальной спецификации.

3.59

литерал (literal): Лексическая единица, которая прямо представляет значение.
[ИСТОЧНИК: ISO/AFNOR:1989]

3.60 **логическое расположение** (logical location): Расположение иерархически адресуемой переменной в схеме, которая может быть связана или может быть не связана с физической структурой входных и выходных переменных и памятью программируемого контроллера.

3.61 **длинное действительное число** (long real): Действительное число, представленное в двойном слове.

3.62 **двойное слово** (long word): 64-битовый элемент данных.

3.63 **метод** (method): Элемент языка, подобный функции, который может быть определен типом функционального блока и имеет неявный доступ к статическим переменным экземпляра функционального блока или экземпляра класса.

3.64 **прототип метода** (method prototype): Элемент языка, содержащий только сигнатуру метода.

3.65 **именованный элемент** (named element): Элемент структуры, именуемый своим связанным идентификатором.

3.66 **сеть** (network): Совокупность узлов и соединяющих ветвей.

3.67 **числовой литерал** (numeric literal): Литерал, прямо представляющий численное значение, то есть целый литерал или действительный литерал.

3.68 **операция** (operation): Элемент языка, который представляет элементарную функциональность, присущую программному компоненту или методу.

3.69 **операнд** (operand): Элемент языка, на котором выполняется операция.

3.70 **оператор** (operator): Символ, представляющий действие, выполняемое в операции.

3.71 **переопределение** (override): Ключевое слово override, использованное с методом или типом функционального блока для метода с такой же сигнатурой, как метод базового класса или тип функционального блока, использующие новое тело метода.

3.72 **выходная переменная** (output variable): Переменная, используемая для возврата значения из программного компонента, отличного от класса.

3.73 **параметр** (parameter): Переменная, которая используется для предоставления значения программному компоненту (как входной или входной-выходной параметр), или переменная, которая используется для возврата значения из программного компонента (как выходной или входной-выходной параметр).

3.74 **ссылка** (reference): Определяемые пользователем данные, содержащие адрес размещения переменной или экземпляра функционального блока заданного типа.

3.75 **поток энергии** (power flow): Символический поток электроэнергии в релейно-контактной схеме, используемый для указания продвижения логического решающего алгоритма.

3.76 **прагма** (pragma): Конструкция языка для включения в программный компонент текста, который может влиять на подготовку программы к выполнению.

3.77 **программа** (program): Разработка, написание и тестирование программ пользователя.

3.78 **программный компонент** (program organization unit): Функция, функциональный блок, класс или программа.

3.79 **действительный литерал** (real literal): Литерал, прямо представляющий значения типа REAL или LREAL.

3.80 **ресурс** (resource): Элемент языка, соответствующий «функции обработки сигналов» и ее «человеко-машинному интерфейсу» и «функциям интерфейса с датчиками и исполнительными механизмами», при наличии таковых.

3.81 **результат** (result): Значение, возвращаемое как результат выполнения программного компонента.

3.82 **возврат** (return): Конструкция языка в программном компоненте, обозначающая конец последовательности выполнения в компоненте.

3.83 **передний фронт** (rising edge): Часть временной диаграммы сигнала, где происходит переход логической переменной из 0 в 1.

3.84 **область видимости** (scope): Набор программных компонент, в которых применяется объявление или метка.

3.85 **семантика** (semantics): Отношения между символическими элементами языка программирования и их значениями, интерпретацией и использованием.

3.86 **полуграфическое представление** (semigraphic representation): Представление графической информации с использованием ограниченного набора символов.

3.87 **сигнатура** (signature): Набор информации, однозначно определяющий идентичность интерфейса параметров МЕТОДА, состоящий из его имени и имен, типов и порядка всех его параметров (то есть входных, выходных и входных-выходных переменных и типа результата).

3.88 **одноэлементная переменная** (single-element variable): Переменная, представляющая единственный элемент данных.

3.89 **статическая переменная** (static variable): Переменная, значение которой сохраняется от одного вызова до другого.

3.90 **шаг** (step): Ситуация, в которой поведение программного компонента в отношении его входных и выходных переменных следует набору правил, определенных связанными действиями шага.

3.91 **структурированный тип данных** (structured data type): Агрегированный тип данных, который был определен, используя определение STRUCT или FUNCTION_BLOCK.

3.92 **индексирование** (subscripting): Механизм для ссылки к элементу массива посредством ссылки на массив и одного или более выражений, которые, после их вычисления, определяют положение элемента.

3.93 **задача** (task): Элемент контроля выполнения, обеспечивающий периодическое или управляемое выполнение группы связанных программных компонентов.

3.94 **литерал дат и времени** (time literal): Литерал, представляющий данные типов TIME, DATE, TIME_OF_DAY или DATE_AND_TIME.

3.95 **переход** (transition): Условие, посредством которого управление переходит от одного или более предшествующих шагов к одному или более последующих шагов по направленной связи.

3.96 **целое число без знака** (unsigned integer): Целое число, которое может содержать положительные и нулевые значения.

3.97 **литерал целого числа без знака** (unsigned integer literal): Целый литерал, не содержащий спереди знака (+) или минус (-).

3.98 **пользовательский тип данных** (user-defined data type): Тип данных, определенный пользователем.

Пример — Перечисление, массив или структура.

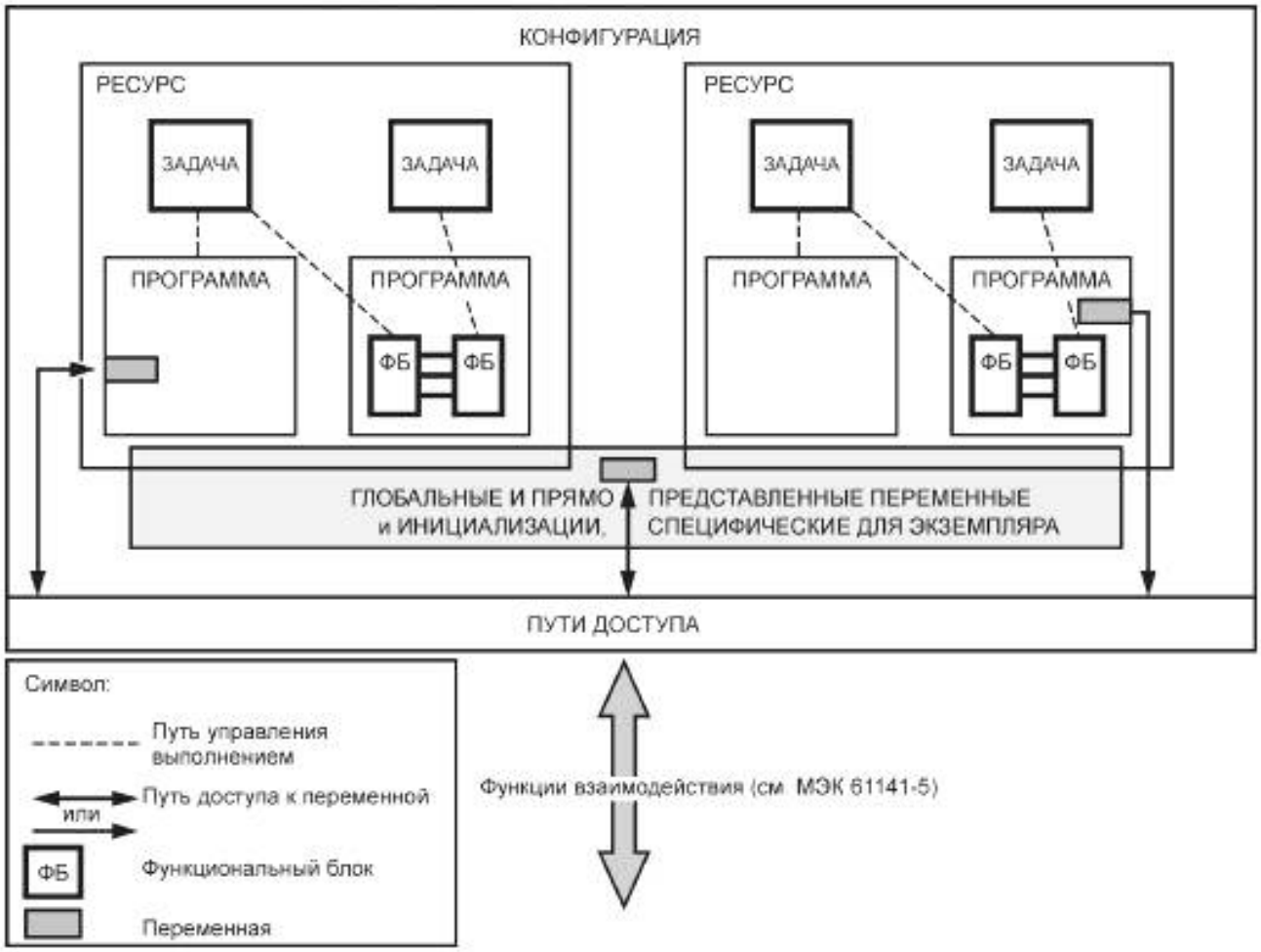
3.99 **переменная** (variable): Объект программного обеспечения, который может принимать различные значения, в каждый момент времени только одно значение.

4 Структурные модели

4.1 Модель программного обеспечения

Основные элементы языка программирования высокого уровня и их взаимосвязи приведены на рисунке 1.

Данные элементы программируются на языках, определенных в настоящем стандарте, т. е. это — программы и типы функциональных блоков, классы, функции и элементы конфигурации, а именно, ресурсы, задачи, глобальные переменные, пути доступа и инициализации экземпляров, которые поддерживают установку программ программируемых контроллеров в системы программируемых контроллеров.



Примечание 1 — Рисунок 1 предназначен только для иллюстрации. Графическое представление не является нормативным.

Примечание 2 — В конфигурации с единственным ресурсом необязательно явно представлять ресурс.

Рисунок 1 — Модель программного обеспечения

Конфигурация является элементом языка, который соответствует системе программируемого контроллера, как определено в МЭК 61131-1. Ресурс соответствует «функции обработки сигналов» и ее «человеко-машинному интерфейсу» и «функциям интерфейса с датчиками и исполнительными механизмами» (при наличии таковых), как определено в МЭК 61131-1.

Конфигурация содержит один или более ресурсов, каждый из которых содержит одну или более программ, выполняемых под контролем нуля или более задач.

Программа может содержать нуль или более экземпляров функциональных блоков или других элементов языка, как определено в настоящем стандарте.

Задача способна вызывать (например, на периодической основе) выполнение набора программ и экземпляров функциональных блоков.

Конфигурации и ресурсы могут запускаться и останавливаться через функции «интерфейс оператора», «программирование, тестирование и мониторинг» или «операционная система», определенные в МЭК 61131-1. Запуск конфигурации будет вызывать инициализацию ее глобальных переменных с последующим запуском всех ресурсов конфигурации. Запуск ресурса будет вызывать инициализацию всех переменных в ресурсе с последующей активацией всех задач в ресурсе. Останов ресурса будет вызывать прекращение всех его задач, в то время как останов конфигурации будет вызывать останов всех ее ресурсов.

Механизмы управления задачами определены в 6.8.2, а механизмы запуска и останова конфигураций и ресурсов через функции взаимодействия определены в МЭК 61131-5.

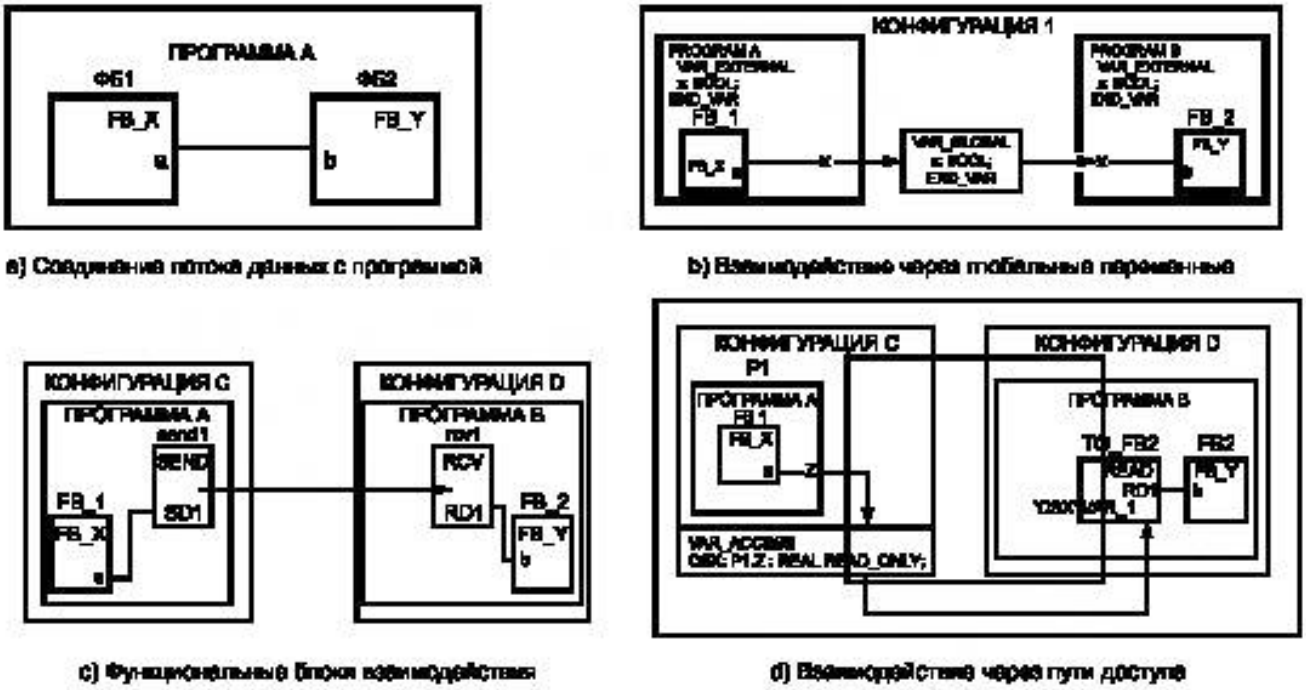
Программы, ресурсы, глобальные переменные, пути доступа (и соответствующие привилегии доступа) и конфигурации могут быть загружены или удалены «функцией взаимодействия», определенной в МЭК 61131-1. Загрузка или удаление конфигурации или ресурса будет эквивалентно загрузке или удалению всех элементов, которые там содержатся.

Пути доступа и их соответствующие привилегии доступа определяются в настоящем стандарте.

Отображение элементов языка на объекты взаимодействия определено в МЭК 61131-5.

4.2 Модель взаимодействия

Способы связи значений переменных с элементами программного обеспечения иллюстрируются на рисунке 2.



a) Соединения потока данных с программой

b) Взаимодействие через глобальные переменные

c) Функциональные блоки взаимодействия

d) Взаимодействие через пути доступа

Примечание 1 — Рисунок 2 предназначен только для иллюстрации. Графическое представление не является нормативным.

Примечание 2 — В данном примере предполагается, что конфигурации C и D имеют один ресурс.

Примечание 3 — На рисунке 2 не показаны детали функционального блока взаимодействия.

Примечание 4 — Пути доступа могут быть объявлены в прямо представленных переменных, входных, выходных или внутренних переменных программ или экземпляров функционального блока.

Примечание 5 — В МЭК 61131-5 определены средства, с помощью которых системы с РС и без РС могут использовать пути доступа для чтения и записи переменных.

Рисунок 2 — Модель взаимодействия

Как показано на рисунке 2a), значения переменных в программе могут связываться прямо, соединением выхода одного программного элемента ко входу другого. Данное соединение явно показывается в графических языках и неявно в тестовых языках.

Значения переменных могут передаваться между программами в одной конфигурации через глобальные переменные, как переменная x, показанная на рисунке 2b). Такие переменные будут объявляться в конфигурации как GLOBAL, и в программах как EXTERNAL.

Как показано на рисунке 2c), значения переменных могут передаваться между различными частями программы, между программами в одной или различных конфигурациях или между программой PC и системой без PC, используя функциональные блоки взаимосвязи, определенные в МЭК 61131-5.

Кроме того, системы с и PC и системы без PC могут передавать данные, которые делаются доступными путями доступа, как показано на рисунке 2d), используя механизмы, определенные в МЭК 61131-5.

4.3 Модель программирования

На рисунке 3 показана сводка элементов языков программирования PLC. Комбинация этих элементов должна подчиняться следующим правилам:

1 Типы данных объявляются с использованием стандартных типов данных и любых ранее определенных типов данных.

2 Функции объявляются с использованием стандартных или определенных пользователем типов данных, стандартных функций и любых ранее определенных функций.

Данные объявления должны использовать механизмы, определенные для языков IL, ST, LD или FBD.

3 Типы функциональных блоков объявляются, используя стандартные и определенные пользователем типы данных, функции, стандартные типы функциональных блоков и любые ранее определенные типы функциональных блоков.

Данные объявления используют механизмы, определенные для языков IL, ST, LD или FBD, и могут включать в себя элементы последовательных функциональных схем (SFC).

Дополнительно, можно определять объектно-ориентированные типы функциональных блоков или классы, которые используют методы и интерфейсы.

4 Программа объявляется, используя стандартные или определенные пользователем типы данных, функции, функциональные блоки и классы.

Данные объявления используют механизмы, определенные в языках IL, ST, LD или FBD и могут в себя включать элементы последовательных функциональных схем (SFC).

5 Программы могут собираться в конфигурации, используя элементы, то есть: глобальные переменные, ресурсы, задачи и пути доступа.

Ссылка на «ранее определенные» типы данных, функции и функциональные блоки означает, что после того как некоторый элемент был объявлен, его определение доступно (например, в «библиотеке» ранее определенных элементов) для использования в дальнейших определениях.

Для программирования функций, типов функциональных блоков и методов может использоваться язык программирования, отличный от языков, определенных в настоящем стандарте.

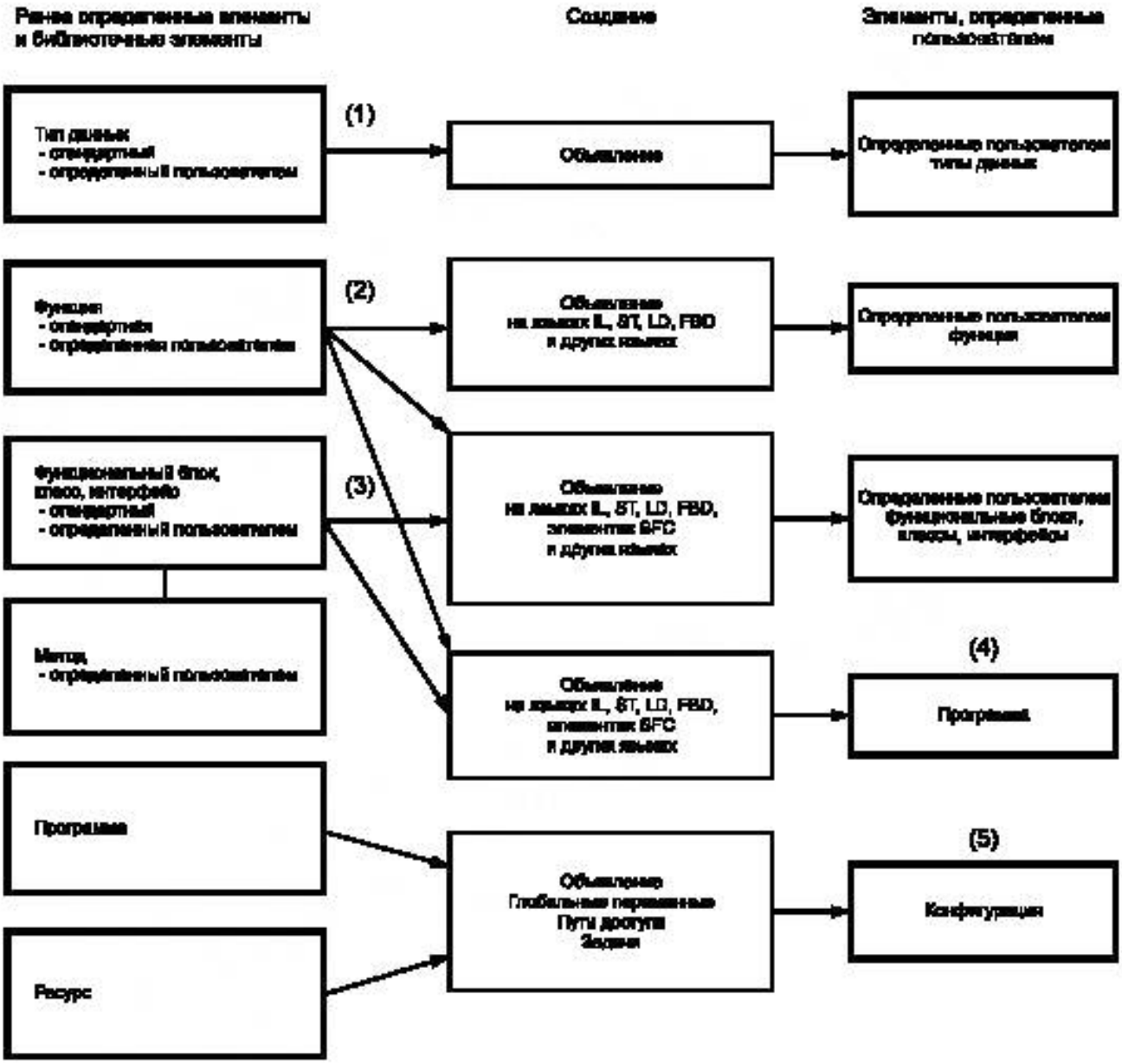


Рисунок 3 — Сочетание элементов языка программируемых контроллеров, лист 1

где LD — язык релейно-контактных схем;

FBD — язык функционально-блоковых диаграмм;

IL — язык списка инструкций;

ST — язык структурированного текста;

Другие — другие языки программирования.

Примечание 1 — Числа от (1) до (5) в скобках относятся к соответствующим параграфам 1)—5) выше.

Примечание 2 — Типы данных используются во всех способах создания. Для четкости, соответствующие связи опущены на данном рисунке.

Рисунок 3 — Сочетание элементов языка программируемых контроллеров, лист 2

5 Совместимость

5.1 Общие положения

Средство программирования и отладки PLC (PADT), как определено в МЭК 61131-1, которое удовлетворяет полностью или частично требованиям настоящего стандарта и должно:

a) обеспечивать подмножество свойств и предоставлять декларацию соответствия разработчика как описано ниже;

b) не требовать включения альтернативных или дополнительных элементов языка для достижения какого-либо свойства;

c) предоставлять документ, определяющий все конкретные расширения разработчика. Сюда входят любые принятые системой свойства, которые запрещены или точно не определены;

d) предоставлять документ, определяющий все специфические зависимости разработчика. В данный документ включают все зависимости реализации, явно определенные в настоящем стандарте, и ограничивающие параметры, такие как максимальная длина, количество, размер и диапазон изменений, которые на заданы явно;

e) предоставлять документ, который устанавливает все ошибки, обнаруживаемые и сообщаемые при реализации. В данный документ включаю ошибки, установленные в настоящем стандарте, и ошибки, обнаруживаемые во время подготовки программы к выполнению и во время ее выполнения.

Примечание — В настоящем стандарте только частично установлены ошибки, случающиеся во время выполнения программы, приведенной в МЭК 61131;

f) не использовать стандартные имена типов данных, функций или имен функциональных блоков, установленных в настоящем стандарте для определенных в реализации свойств, функциональность которых отличается от функциональности свойств, описанных в настоящем стандарте.

5.2 Таблицы свойств

Все таблицы настоящего стандарта, используемые для конкретной цели, представлены единообразно. В первой графе содержится «номер свойства», во второй графе дается «описание свойства», следующие графы могут содержать примеры или дополнительную информацию. В декларации соответствия разработчика используется следующая структура таблицы.

5.3 Декларация соответствия разработчика

Разработчик может определить любое согласующееся подмножество свойств, перечисляемых в таблицах свойств и будет объявлять предоставляемое подмножество как «Декларацию соответствия разработчика».

Декларация соответствия разработчика будет включена в документацию, сопровождающую систему, или будет создаваться самой системой.

Формат декларации соответствия разработчика будет предоставлять следующую информацию (пример декларации соответствия приведен на рисунке 4):

- общая информация, включающая наименование и адрес разработчика, наименование и версию продукта и дату выпуска;
- номер соответствующей таблицы соответствия, номер свойства и используемый язык программирования для каждого реализованного свойства.

Заголовок и подзаголовок таблицы свойств, описание свойства, примеры, примечания разработчика и другая информация являются необязательными.

Нереализованные таблицы и свойства могут быть опущены.

МЭК 61131-3 «Языки программирования PLC»						
Разработчик: Наименование компании, адрес и пр.						
Продукт: Наименование продукта, версия и т. д. Тип контроллера и т. д.						
Дата: 2012-05-01						
Настоящий продукт удовлетворяет требованиям для следующих свойств: языка:						
Номер свойства	Номер и заголовок таблицы/ Описание свойства	Соответствующим образом реализовано в языке (√)				Примечание разработчика
		LD	FBD	ST	IL	
	Таблица 1 — Набор символов					
1	ИСО/МЭК 10646:2012, Информационная технология — Универсальный набор символов (UCS)	√	√	√	√	
2a	Символы нижнего регистра: a, b, c, …	√	√	√		Отсутствуют символы «ß, ü, ä, ö»
2b	Знак числа: # См. таблицу 5	√				
2c	Знак доллара $ См. таблицу 6		√			
	Таблица 2 — Идентификаторы					
1	Буквы верхнего регистра и цифры IW215					
2	Буквы верхнего и нижнего регистров, цифры и внутренние символы подчеркивания					
3	Буквы верхнего и нижнего регистров, цифры и ведущие или внутренние символы подчеркивания					
	Таблица 3 — Комментарии					
1	Однострочный комментарий //…					
2a	Многострочный комментарий (* … *)					
2b	Многострочный комментарий /* … */					
3a	Вложенный комментарий (* ..(* .. *) ..*)					
3b	Вложенный комментарий /* .. /* .. */ .. */					

Рисунок 4 — Декларация соответствия разработчика (пример)

	Таблица 4 — Прагма					
1	Прагма в фигурных скобках { … }					
	Таблица 5 — Числовые литералы					
1	Целый литерал: -12					
2	Действительный литерал: -12.0					
3	Действительные литералы с экспонентой: -1.34E-12					
4	Двоичный литерал: 2#1111_1111					
5	Восьмеричный литерал: 8#377					
6	Шестнадцатеричный литерал: 16#FF					

7	Логический ноль и единица					
8	Логические FALSE и TRUE					
9	Типизированный литерал: INT#-123					
	И т. д.					

Рисунок 4

6 Общие элементы

6.1 Использование печатных символов

6.1.1 Набор символов

Набор символов текстовых языков и текстовых элементов графических языков приведен в таблице 1. Символы представлены по ИСО/МЭК 10646.

Таблица 1 — Набор символов

Номер	Описание		
1	ИСО/МЭК 10646		
2a	Символы нижнего регистра[a]:		a, b, c
2b	Знак числа:	#	См. таблицу 5
2c	Знак доллара:	$	См. таблицу 6

[a] Когда поддерживаются буквы нижнего регистра, регистр букв в элементах языка не учитывается за исключением комментариев, как определено 6.1.5, строковых литералах 6.3.3, и переменных типов STRING и WSTRING, как определено в 6.3.3.

6.1.2 Идентификаторы

Идентификатор — это строки букв, цифр и символов подчеркивания, начинающаяся с буквы или символа подчеркивания.

Регистр букв не имеет значения в идентификаторах, например идентификаторы abcd, ABCD и aBCd будут интерпретироваться одинаково.

Символ подчеркивания является существенным в идентификаторах, например, A_BCD и AB_CD будут интерпретироваться, как различные идентификаторы. Множественные ведущие или множественные внутренние символы подчеркивания не допустимы, например последовательности символов _LIM_SW5 и LIM_SW5 не являются допустимыми идентификаторами. Завершающие символы подчеркивания не допустимы, например, последовательность символов LIM_SW5_ не является допустимым идентификатором.

Во всех системах, которые поддерживают использование идентификаторов, по меньшей мере, 6 символов будет учитываться при определении уникальности идентификатора, например, во всех таких системах, ABCDE1 будет интерпретироваться отличным от ABCDE2. Максимально допустимое число символов, разрешенное в идентификаторе, определяется разработчиком.

Свойства и примеры идентификаторов приведены в таблице 2.

Таблица 2 — Идентификаторы

Номер	Описание	Пример
1	Буквы верхнего регистра и цифры IW215	IW215 IW215Z QX75 IDENT
2	Буквы верхнего и нижнего регистров, цифры и внутренние символы подчеркивания	Все приведенные выше плюс: LIM_SW_5 LimSw5 abcd ab_Cd
3	Буквы верхнего и нижнего регистров, цифры и ведущие или внутренние символы подчеркивания	Все приведенные выше плюс: _MAIN _12V7

6.1.3 Ключевые слова

Ключевые слова — это уникальные комбинации символов, используемых как отдельные синтаксические элементы. Ключевые слова не содержат внутренних пробелов. В ключевых словах регистр символов не учитывается.

Например, ключевые слова FOR и for синтаксически эквивалентны. Они не должны использоваться в любых других целях, например, как имена переменных или расширения.

6.1.4 Использование символов-разделителей

Пользователю разрешено вставлять один или более «символов-разделителей» в любом месте текста программ программируемого контроллера, только не внутри ключевых слов, литералов, перечисленных значений, идентификаторов, прямо представленных переменных или разделительных сочетаний, например, для комментариев. «Символ-разделитель» определяется как символ SPACE с кодированным числовым значением 32, а также как непечатаемые символы, такие как символ табуляции, символ перевода строки и т. п., которым в МЭК/ИСО 10646 не придано закодированного значения.

6.1.5 Комментарии

Имеются различные виды комментариев, приведенные в таблице 3.

1 Однострочные комментарии начинаются с комбинации символов // и заканчиваются на следующем символе перевода строки, новой строки, подачи (прогона) страницы или возврата каретки.

В однострочном комментарии специальные комбинации символов (* и *) или /* и */ не имеют специального значения.

2 Многострочном комментарии разделяются вначале и в конце специальными комбинациями символов (* и *), соответственно.

Альтернативно, многострочный комментарий может предоставляться, используя специальные комбинации символов /* и */.

В многострочном комментарии специальная комбинация символов // не имеет специального значения.

Комментарии разрешены в любом месте программы, где разрешены пробелы, только не внутри символьно-строковых литералов.

Комментарии не имеют никакого синтаксического и семантического значения ни в одном из языков, определенных в данном стандарте. Они трактуются как символы-разделители.

Вложенные комментарии используют соответствующие

- пары (*, *), например, (* ... (* ВЛОЖЕННЫЙ КОММЕНТАРИЙ *)... *) или
- пары /*, */,например, /* ... /* ВЛОЖЕННЫЙ КОММЕНТАРИЙ */... */.

Таблица 3 — Комментарии

Номер	Описание	Пример

1	Однострочный комментарий
	// ...
		X:= 13; // комментарий для одной строки
		// однострочный комментарий может начинаться
		// с позиции первого символа

2a	Многострочный комментарий с (* ... *)	(* текст комментария *) (********** Комментарий в рамке на трех строках **********)
2b	Многострочный комментарий с /* ... */	/* комментарий на одной или более строк */
3a	Вложенный комментарий с (* .. (* .. *) ..*)	(* (* ВЛОЖЕННЫЙ КОММЕНТАРИЙ *) *)
3b	Вложенный комментарий с /* .. /* .. */ .. */	/* /* ВЛОЖЕННЫЙ КОММЕНТАРИЙ */ */

6.2 Прагма

Как показано в таблице 4, прагмы ограничиваются в начале и в конце фигурными скобками { и }, соответственно. Синтаксис и семантика конструкций конкретной прагмы определяются разработчиком.

Комментарии разрешены в любом месте программы, где разрешены пробелы, только не внутри символьно-строковых литералов.

Таблица 4 — Прагма

Описание	Пример
Прагма с фигурными скобками { ... }	{ВЕРСИЯ 2.0} {АВТОР JHC} {x:= 256, y:= 384}

6.3 Литералы — внешнее представление данных

6.3.1 Общие положения

Внешние представления данных в различных языках программирования программируемых контроллеров состоят из числовых литералов, символьно-строковых литералов и литералов дат и времени.

Признана необходимость в обеспечении внешних представлений для двух различных типов данных, связанных со временем:

- данные о продолжительности времени при измерении и контроле событий;
- данные о времени суток, которые могут также включать в себя информацию о дате — для синхронизации начала и окончания событий в абсолютной временной шкале.

6.3.2 Числовые литералы и строковые литералы

Имеется два типа числовых литералов: целые литералы и действительные литералы. Числовой литерал определяется как десятичное число или число с основанием. Максимальное количество цифр для каждого вида числовых литералов должно быть достаточным для выражения всего диапазона значений с требуемой точностью для всех типов данных, которые представляются литералами в заданной реализации.

Единичные символы подчеркивания «_», вставленные между цифрами числового литерала не являются существенными. Никакое иное использование символов подчеркивания в числовых литералах не разрешается.

Десятичные литералы представляются в обычной десятичной нотации. Действительные литералы характеризуются наличием десятичной точки. Экспонента указывает на целую степень 10, на которую должно умножаться предшествующее число, чтобы достичь представленного значения. Десятичные литералы и экспоненты литералов могут содержать предшествующий знак «+» или «–».

Литералы могут также представляться с основаниями 2, 8 и 16. Основание указывается в десятичной нотации. Для основания 16 используется расширенный набор цифр, состоящий из букв от A до F, с оговоренным десятичным значением от 10 до 15, соответственно. Числа с основанием не содержат ведущего знака «+» или «–». Они интерпретируются как битово-строковые литералы.

Числовые литералы, представляющие положительные целые значения, могут использоваться как битово-строковые литералы.

Логические данные представляются числовыми литералами со значением (0) или один (1), или ключевыми словами FALSE или TRUE, соответственно.

Свойства и примеры числовых литералов приведены в таблице 5.

Тип данных логических или числовых литералов может указываться добавлением префикса типа к литералу, состоящего из имени элементарного типа данных и символа «#». Примеры приведены в свойстве 9 таблицы 5.

Таблица 5 — Числовые литералы

Номер	Описание	Пример	Объяснение
1	Целый литерал	-12, 0, 123_4, +986	
2	Действительный литерал	0.0, 0.4560, 3.14159_26	
3	Действительные литералы с экспонентой	-1.34E-12, -1.34e-12 1.0E+6, 1.0e+6 1.234E6, 1.234e6	
4	Двоичный литерал	 2#1111_1111 2#1110_0000	Литерал с основанием 2 для представления: десятичного числа 255 десятичного числа 224
5	Восьмеричные литералы	 8#377 8#340	Литерал с основанием 8 для представления: десятичного числа 255 десятичного числа 224
6	Шестнадцатеричный литерал	 16#FF или 16#ff 16#E0 или 16#e0	Литерал с основанием 16 для представления: десятичного числа 255 десятичного числа 224
7	Логический ноль и единица	0 или 1	
8	Логические FALSE и TRUE	FALSE TRUE	
9	Типизированный литерал	INT#-123	Представление INT десятичного значения –123
		INT#16#7FFF	Представление INT десятичного значения 32767
		WORD#16#AFF	Представление WORD шестнадцатиричного значения 0AFF
		WORD#1234	Представление WORD десятичного значения 1234=16#4D2
		UINT#16#89AF	Представление UINT шестнадцатиричного значения 89AF
		CHAR#16#41	Представление CHAR символа 'A'
		BOOL#0	
		BOOL#1	
		BOOL#FALSE	
		BOOL#TRUE	

Примечание 1 — Ключевые слова FALSE и TRUE соответствуют логическим значениям 0 и 1, соответственно.

Примечание 2 — Свойство 5 «Восьмеричные литералы» не рекомендуется и может не быть включено в следующую редакцию настоящего стандарта.

6.3.3 Символьно-строковые литералы

Символьно-строковые литералы содержат однобайтовые или двухбайтовые кодированные символы:

- символьно-строковый литерал однобайтовых символов является последовательностью нуля или более символов, с предшествующим и завершающим символом одиночной кавычки ('). В строках однобайтовых символов, трехсимвольная комбинация символа доллара ($) с двумя следующими шестнадцатиричными символами интерпретируется как шестнадцатиричное представление восьмибитового кода символа, как показано в свойстве 1 таблицы 6;

- символьно-строковый литерал двухбайтовых символов является последовательностью 0 или более символов из набора символов ИСО/МЭК 10646, с предшествующим и завершающим символом двойной кавычки («). В строках однобайтовых символов, трехсимвольная комбинация символа доллара ($) с двумя следующими шестнадцатиричными символами интерпретируется как шестнадцатиричное представление восьмибитового кода символа, как показано в свойстве 2 таблицы 6.

Примечание — Отношение стандартов ИСО/МЭК 10646 и Юникод:

Несмотря на то, что коды символов и формы кодирования стандартов Юникод и ИСО/МЭК 10646 синхронизированы, стандарт Юникод налагает дополнительные ограничения на реализации, чтобы гарантировать, что они трактуют символы одинаково во всех платформах и приложениях. В связи с этим, данный стандарт предоставляет широкий набор спецификаций функциональных символов, данных символов, алгоритмов и обширный справочный материал, который отсутствует в ИСО/МЭК 10646.

Двухсимвольные комбинации, начинающиеся с символа доллара интерпретируются, как показано в таблице 7, когда они встречаются в строках символов.

Таблица 6 — Символьно-строковые литералы

Номер	Описание	Пример
	Односимвольные символы и строки символов с ' '	
1a	Пустая строка (длины ноль)	''
1b	Строка длины 1 или символ CHAR, содержащий единственный символ	'A'
1c	Строка длины один или символ CHAR, содержащий символ пробела	' '
1d	Строка длины один или символ CHAR, содержащий символ одиночной кавычки	$
1e	Строка длины один или символ CHAR, содержащий символ двойной кавычки	'»'
1f	Поддержка двухсимвольных комбинаций таблицы 7	'RL'
1g	Поддержка представления символа с знаком доллара '$' и двумя шестнадцатиричными цифрами	'$0A'
	Двухбайтовые символы или симаольные строки с «» (Примечание)	
2a	Пустая строка (длины ноль)	«»
2b	Строка длины один или символ WCHAR, содержащий единственный символ	«A»
2c	Строка длины один или символ WCHAR, содержащий символ пробела	« »
2d	Строка длины один или символ WCHAR, содержащий символ одиночной кавычки	«'»
2e	Строка длины один или символ WCHAR, содержащий символ двойной кавычки	«$"»
2f	Поддержка двухсимвольных комбинаций таблицы 7	«RL»
2h	Поддержка представления символа с знаком доллара '$' и четырьмя шестнадцатиричными цифрами	«$00C4»
	Типизированные однобайтовые символы или строковый литерал с #	
3a	Типизированная строка	STRING#'OK'
3b	Типизированный символ	CHAR#'X'

Окончание таблицы 6

Номер	Описание	Пример
	Двухбайтовые типизированные строковые литералы с # (NOTE)	
4a	Типизированные двухбайтовые строки (с использованием символа двойной кавычки)	WSTRING#«OK»
4b	Типизированный двухбайтовый символ (с использование символа двойной кавычки)	WCHAR#«X»
4d	Типизированный двухбайтовый символ (с использованием символа одиночной кавычки)	WCHAR#'X'
Примечание — Если конкретная реализация поддерживает свойство 4, но не поддерживает свойство 2, Реализатор может определить синтаксис и семантику для использования символа двойной кавычки.		

Таблица 7 — Двухсимвольные комбинации в символьных строках

Номер	Описание	Комбинации
1	Знак доллара	$$
2	Единичная кавычка	$'
3	Перевод строки	$L или $l
4	Новая строка	$N или $n
5	Прогон (перевод) страницы	$P или $p
6	Возврат каретки	$R или $r
7	Табуляция	$T или $t
8	Двойная кавычка	$»
Примечание 1 — Символ новой строки предоставляет независимые от реализации средства определения конца строки данных. При печати эффект таков, что заканчивается текущая строка данных и печать возобновляется в начале следующей строки. Примечание 2 — Комбинация $' действительна только внутри строковых литералов с одиночными кавычками. Примечание 3 — Комбинация $» действительна только внутри строковых литералов с двойными кавычками.		

6.3.4 Литерал продолжительности времени

Данные продолжительности времени ограничиваются слева ключевым словом T#, TIME# или LTIME#. Представление данных о продолжительности времени в терминах дней, часов, минут, секунд и долей секунды, или любой их комбинации поддерживается как показано в таблице 8. Наименьшая единица времени может быть записана в нотации действительных чисел без экспоненты.

Единицы литералов продолжительности времени могут разделяться символами подчеркивания.

Разрешается «переполнение» самой большой единицы продолжительности времени, например нотация T#25h_15m является допустимой.

Единицы времени (например, секунды, миллисекунды и т. д.) могут быть представлены буквами верхнего или нижнего регистра.

Как показано в таблице 8, для продолжительности времени разрешены как положительные, так и отрицательные значения.

Таблица 8 — Литералы продолжительности времени

Номер	Описание	Пример
	Сокращения продолжительности времени	
1a	d	День
1b	h	Час
1c	m	Минута
1d	s	Секунда
1e	ms	Миллисекунда
1f	us (если символ μ недоступен)	Микросекунда
1g	ns	Наносекунда
	Литералы продолжительности времени без символов подчеркивания	
2a	короткий префикс	T#14ms T#-14msLT#14.7s T#14.7m T#14.7h t#14.7d t#25h15m lt#5d14h12m18s3.5ms t#12h4m34ms230us400ns
2b	длинный префикс	TIME#14ms TIME#-14ms time#14.7s
	Литералы продолжительности времени с символами подчеркивания	
3a	короткий префикс	t#25h_15m t#5d_14h_12m_18s_3.5ms LTIME#5m_30s_500ms_100.1us
3b	длинный префикс	TIME#25h_15m ltime#5d_14h_12m_18s_3.5ms LTIME#34s_345ns

6.3.5 Литерал даты и времени суток

Ключевые слова префикса литералов для времени суток и даты приведены в таблице 9.

Таблица 9 — Литералы даты и времени суток

Номер	Описание	Пример
1a	Литерал даты (длинный префикс)	DATE#1984-06-25, date#2010-09-22
1b	Литерал даты (короткий префикс)	D#1984-06-25
2a	Длинный литерал даты (длинный префикс)	LDATE#2012-02-29
2b	Длинный литерал даты (короткий префикс)	LD#1984-06-25
3a	Литерал времени суток (длинный префикс)	TIME_OF_DAY#15:36:55.36
3b	Литерал времени суток (короткий префикс)	TOD#15:36:55.36
4a	Длинный литерал времени суток (короткий префикс)	LTOD#15:36:55.36
4b	Длинный литерал времени суток (длинный префикс)	LTIME_OF_DAY#15:36:55.36
5a	Литерал даты и времени (длинный префикс)	DATE_AND_TIME#1984-06-25-15:36:55.360227400
5b	Литерал даты и времени (короткий префикс)	DT#1984-06-25-15:36:55.360_227_400

Окончание таблицы 9

Номер	Описание	Пример
6a	Длинный литерал даты и времени (длинный префикс)	LDATE_AND_TIME#1984-06-25-15:36:55.360_227_400
6b	Длинный литерал даты и времени (короткий префикс)	LDT#1984-06-25-15:36:55.360_227_400

6.4 Типы данных

6.4.1 Общие положения

Тип данных — это классификация, которая определяет возможные значения для литералов и переменных, операции, которые можно выполнять и способ хранения значений.

6.4.2 Элементарные типы данных (BOOL, INT, REAL, STRING и т. д.)

6.4.2.1 Спецификация элементарных типов данных

Настоящий стандарт устанавливает набор (предопределенных) элементарных типов данных.

Элементарные типы данных, ключевое слова для каждого типа данных, число битов на элемент данных и диапазон значений для каждого элементарного типа данных приведены в таблице 10.

Таблица 10 — Элементарные типы данных

Номер	Описание	Ключевое слово	Неявное начальное значение	Длина (бит)[a]
1	Логический	BOOL	0, FALSE	1[h]
2	Короткое целое	SINT	0	8[c]
3	Целое	INT	0	16[c]
4	Двойное целое	DINT	0	32[c]
5	Длинное целое	LINT	0	64[c]
6	Короткое целое без знака	USINT	0	8[d]
7	Целое без знака	UINT	0	16[d]
8	Двойное целое без знака	UDINT	0	32[d]
9	Двойное целое без знака	ULINT	0	64[d]
10	Действительные числа	REAL	0.0	32[e]
11	Длинные целые	LREAL	0.0	64[f]
12a	Продолжительность времени	TIME	T#0s	--[b]
12b	Продолжительность времени	LTIME	LTIME#0s	64[m], q]
13a	Дата (отдельно)	DATE	Примечание	-- [b]
13b	Длинная дата (отдельно)	LDATE	LDATE#1970-01-01	64[n]
14a	Время суток (отдельно)	TIME_OF_DAY или TOD	TOD#00:00:00	--[b]
14b	Время суток (отдельно)	LTIME_OF_DAY или LTOD	LTOD#00:00:00	64[o], q]
15a	Дата и время суток	DATE_AND_TIME или DT	Примечание	--[b]
15b	Дата и время суток	LDATE_AND_TIME или LDT	LDT#1970-01-01-00:00:00	64[p], q]
14a	Время суток (отдельно)	TIME_OF_DAY или TOD	TOD#00:00:00	--[b]
14b	Время суток (отдельно)	LTIME_OF_DAY или LTOD	LTOD#00:00:00	64[o], q]

Окончание таблицы 10

Номер	Описание	Ключевое слово	Неявное начальное значение	Длина (бит)[a]
15a	Дата и время суток	DATE_AND_TIME или DT	Примечание	--[b]
15b	Дата и время суток	LDATE_AND_TIME или LDT	LDT#1970-01-01-00:00:00	64[p], [q]
16a	Строка однобайтовых символов переменной длины	STRING	'' (пустая)	8[i], [g], [k], [l]
16b	Строка двухбайтовых символов переменной длины	WSTRING	'' (пустая)	16[i], [g], [k], [l]
17a	Однобайтовый символ	CHAR	'$00'	8[g], [l]
17b	Двухбайтовый символ	WCHAR	«$0000»	16[g], [l]
18	Битовая строка длины 8	BYTE	16#00	8[j], [g]
19	Битовая строка длины 16	WORD	16#0000	16[j], [g]
20	Битовая строка длины 32	DWORD	16#0000_0000	32[j], [g]
21	Битовая строка длины 64	LWORD	16#0000_0000_0000_0000	64[j], [g]

Примечание — Определяется разработчиком, так как специальное стартовое значение отлично от 0001-01-01.

[a] Значения в данной колонке интерпретируются как описано в подстрочных примечаниях к таблице.

[b] Диапазон значений и точность представления в данных типах данных определяются разработчиком.

[c] Диапазон значений переменных данного типа данных от — (2^{N-1}) до (2^{N-1}) -1.

[d] Диапазон значений переменных данного типа данных от 0 до (2^{N}) -1.

[e] Диапазон значений переменных данного типа данных определяется в МЭК 60559 для основного формата с плавающей точкой одинарной точности. Результаты арифметических команд с ненормализованными значениями, бесконечным значением и нечисловыми значениями определяются разработчиком.

[f] Значение переменных данного типа данных определяется в МЭК 60559 для основного формата с плавающей точкой двойной точности. Результаты арифметических команд с ненормализованными значениями, бесконечным значением и нечисловыми значениями определяются разработчиком.

[g] Числовой диапазон значений не применяется к данному типу данных.

[h] Возможные значения переменных этого типа данных: 0 и 1, соответствующие ключевым словам FALSE и TRUE соответственно.

[i] Значение N указывает на число битов или символов для этого типа данных.

[j] Значение N указывает на число битов в битовой строке для этого типа данных.

[k] Допустимая переменных типов STRING и WSTRING определяется разработчиком.

[l] Типов CHAR, STRING, WCHAR и WSTRING используется кодировка по ИСО/МЭК 10646 (см. 6.3.3).

[m] Тип данных LTIME является 64-битовым целым числом со знаком, значение задается в наносекундах.

[n] Тип данных LDATE является 64-битовым целым числом со знаком, значение задается в наносекундах, с начальной датой 1970-01-01.

[p] Тип данных LTOD является 64-битовым целым числом со знаком, значение задается в наносекундах, начальное время с полуночи TOD#00:00:00.

[q] Точность обновления значений данного формата времени определяется разработчиком; то есть значение указывается в наносекундах, но оно может обновляться через микросекунду или миллисекунду.

6.4.2.2 Элементарные строковые типы данных (STRING, WSTRING)

Максимальная поддерживаемая длина элементов типа STRING и WSTRING задается разработчиком и определяет максимальную длину STRING и WSTRING, которая поддерживается средствами программирования и отладки.

Явная максимальная длина определяется максимальной длиной (которая не должна превышать поддерживаемое максимальное значение, определенное разработчиком), приведенной в скобках в соответствующем объявлении данных.

Доступ к отдельным символам строки в элементах данных CHAR или WCHAR осуществляется указанием в квадратных скобках позиции символа в строке, начиная с позиции 1.

Ошибка возникает, если к строкам двухбайтовых символов осуществляется доступ с использованием однобайтовых символов или если к строкам однобайтовых символов осуществляется доступ с использование двухбайтовых символов.

Пример 1 — Типы STRING, WSTRING и CHAR, WCHAR

a) Объявление

```
VAR
    String1: STRING[10]:= 'ABCD';
    String2: STRING[10]:= '';
    aWStrings: ARRAY [0..1] OF WSTRING:= ["1234", "5678"];
    Char1: CHAR;
    WChar1: WCHAR;
END_VAR
```

b) Использование типов STRING и CHAR

```
Char1:= String1[2]; // эквивалентно Char1:= 'B';
String1[3]:= Char1; // приводит к String1:= 'ABBD'
String1[4]:= 'B'; // приводит к String1:= 'ABBB'
String1[1]:= String1[4]; // приводит к String1:= 'BBBB'
String2:= String1[2]; (* приводит к String1:= 'BBBB'
если было выполнено неявное преобразование CHAR_TO_STRING*)
```

c) Использование типов WSTRING и WCHAR

```
WChar1:= aWStrings[1][2]; // эквивалентно WChar1:= 'B';
aWStrings[1][3]:=WChar1; // приводит к aWStrings[1]:= «5668»
aWStrings[1][4]:= «6»; // приводит к aWStrings[1]:= "5666"
WStrings[1][1]:= aWStrings[1][4]; // приводит к String1:= "6666"
aWStrings[0]:= aWStrings[1][4]; (* приводит aWStrings[0]:= "6";
если было выполнено неявное преобразование WCHAR_TO_WSTRING*)
```

d) Эквивалентные функции (см. 6.6.2.5.11)

```
Char1:= String1[2];
эквивалентно
Char1:= STRING_TO_CHAR(Mid(IN:= String1, L:= 1, P:= 2));
aWStrings[1][3]:= WChar1;
эквивалентно
REPLACE(IN1:= aWStrings[1], IN2:= WChar1, L:= 1, P:=3 );
```

e) Случаи ошибки

```
Char1:= String1[2]; // смешивание типов WCHAR,
    STRING String1[2]:= String2;
// требует неявного преобразования STRING_TO_CHAR, которое не разрешено
```

Примечание — Типы данных для отдельных символов (CHAR и WCHAR) могут содержать только один символ. Строки могут содержать несколько символов; поэтому строки могут содержать дополнительную информацию для управления, которая не нужна для отдельных символов.

Пример 2 — Если тип STR10 объявлен как

```
TYPE STR10: STRING[10]:= 'ABCDEF'; END_TYPE,
```

то максимальная длина STR10 равна 10 символам, начальное значение по умолчанию равно 'ABCDEF', и начальная длина элементов данных типа STR10 равна шести символам.

6.4.3 Родовые типы данных

В дополнение к элементарным типам данных, приведенным в таблице 10, в спецификации входных и выходных переменных стандартных функций и функциональных блоков можно использовать иерархию родовых типов данных, показанных на рисунке 5. Родовые типы данных определяются по префиксу «ANY».

При использовании родовых типов данных следует соблюдать следующие правила:

1 Родовой тип прямо порожденного типа является таким же, как родовой тип элементарного типа, из которого он порожден.

2 Порожденным типом типа-диапазона является ANY_INT.

Родовым типом всех других порожденных типов, приведенных в таблице 11, является ANY_DERIVED.

Использование родовых типов данных в определенных пользователем программных компонентов находится вне области действия настоящего стандарта.

Родовые типы данных	Родовые типы данных	Группы элементарных типов данных
ANY	g)	
ANY_DERIVED		
ANY_ELEMENTARY		
ANY_MAGNITUDE		
ANY_NUM		
ANY_REAL	h)	REAL, LREAL
ANY_INT	ANY_UNSIGNED	USINT, UINT, UDINT, ULINT
	ANY_SIGNED	SINT, INT, DINT, LINT
ANY_DURATION		TIME, LTIME
ANY_BIT		BOOL, BYTE, WORD, DWORD, LWORD
ANY_CHARS		
ANY_STRING		STRING, WSTRING
ANY_CHAR		CHAR, WCHAR
ANY_DATE		DATE_AND_TIME, LDT, DATE, TIME_OF_DAY, LTOD

Рисунок 5 — Иерархия родовых типов данных

6.4.4 Определенные пользователем типы данных

6.4.4.1 Объявление (TYPE)

6.4.4.1.1 Общие положения

Назначение определенных пользователем типов данных — это их использование в объявлении других типов данных и в объявлениях переменных.

Определенный пользователем тип данных может использоваться везде, где может использоваться базовый тип.

Определенные пользователем типы данных объявляются, используя текстовую конструкцию TYPE...END_TYPE. Объявление типа состоит из следующих элементов:

- имя типа;
- символ двоеточия «:»;
- объявление собственно типа, как определено в следующих предложениях.

Пример — Объявление типа

TYPE

myDatatype1: <объявление типа с необязательной инициализацией>;

END_TYPE

6.4.4.1.2 Инициализация

Определенные пользователем типы данных могут быть инициализированы определенными пользователем значениями. Такая инициализация имеет приоритет над неявным начальным значением.

Определенная пользователем инициализация следует за объявлением типа и начинается оператором присваивания «:=», за которым следует начальное значение (значения).

Могут использоваться литералы (например, -123, 1.55, «abc») или константные выражения (например, 12*24). Используемые начальные значения должны иметь совместимый тип, то есть тот же самый тип или тип, который может быть конвертирован, используя неявное преобразование типа.

Для инициализации типов данных следует применять правила, приведенные на рисунке 6.

Родовой тип данных	Инициализировано литералом	Результат
ANY_UNSIGNED	Неотрицательный целый литерал или неотрицательное константное выражение	Неотрицательное целое значение
ANY_SIGNED	Целый литерал или константное выражение	Целое значение
ANY_REAL	Числовой литерал или константное выражение	Числовое значение
ANY_BIT	Целый литерал без знака или константное выражение без знака	Целое значение без знака
ANY_STRING	Битово-строковый литерал	Строковое значение
ANY_DATE	Литерал даты и времени суток	Значение даты и времени суток
ANY_DURATION	Литерал продолжительности времени	Значение продолжительности времени

Рисунок 6 — Инициализация литералами и константными выражениями (правила)

В таблице 11 определены свойства объявления типов данных и их инициализации, определенных пользователем.

Таблица 11 — Объявление определенных пользователем типов данных и их инициализации

Номер	Описание	Пример	Объяснение
1a 1b	Перечислимые типы данных	TYPE ANALOG_SIGNAL_RANGE: (BIPOLAR_10V, UNIPOLAR_10V, UNIPOLAR_1_5V, UNIPOLAR_0_5V, UNIPOLAR_4_20_MA, UNIPOLAR_0_20_MA) := UNIPOLAR_1_5V; END_TYPE	Инициализация
2a 2b	Типы данных с именованными значениями	TYPE Colors: DWORD (Red := 16#00FF0000, Green:= 16#0000FF00, Blue := 16#000000FF, White:= Red OR Green OR Blue, Black:= Red AND Green AND Blue) := Green; END_TYPE	Инициализация

Продолжение таблицы 11

Номер	Описание	Пример	Объяснение
3a 3b	Тип — диапазоны	TYPE ANALOG_DATA: INT(-4095 .. 4095):= 0; END_TYPE	
4a 4b	Типы данных — массивы	TYPE ANALOG_16_INPUT_DATA: ARRAY [1..16] OF ANALOG_DATA := [8(-4095), 8(4095)]; END_TYPE	ANALOG_DATA см. выше. Инициализация
5a 5b	Типы функциональных блоков и классы как элементы массива	TYPE TONs: ARRAY[1..50] OF TON := [50(PT:=T#100ms)]; END_TYPE	Инициализация функционального блока TON как элемента массива
6a 6b	Структурированный тип данных	TYPE ANALOG_CHANNEL_CONFIGURATION: STRUCT RANGE: ANALOG_SIGNAL_RANGE; MIN_SCALE: ANALOG_DATA:= -4095; MAX_SCALE: ANALOG_DATA:=4095; END_STRUCT; END_TYPE	см. выше ANALOG_SIGNAL_RANGE
7a 7b	Типы функциональных блоков и классы как элементы структуры	TYPE Cooler: STRUCT Temp: INT; Cooling: TOF:= (PT:=T#100ms); END_TYPE	Функциональный блок TOF как элемент структуры
8a 8b	Структурированный тип данных с относительной адресацией AT	TYPE Com1_data: STRUCT head AT %B0: INT; length AT %B2: USINT:= 26; flag1 AT %X3.0: BOOL; end AT %B25: BYTE; END_STRUCT; END_TYPE	Явное расположение без перекрытия
9a	Структурированный тип данных с относительной адресацией AT и OVERLAP	TYPE Com2_data: STRUCT OVERLAP head AT %B0: INT; length AT %B2: USINT; flag2 AT %X3.3: BOOL; data1 AT %B5: BYTE; data2 AT %B5: REAL; end AT %B19: BYTE; END_STRUCT; END_TYPE	Явное расположение с перекрытием

Окончание таблицы 11

Номер	Описание	Пример	Объяснение

10a
10b
	Прямо представленные элементы структуры — частично определенные, используя «*»	TYPE
		HW_COMP: STRUCT;
		IN AT %I*: BOOL;
		OUT_VAR AT %Q*: WORD:= 200;
		ITNL_VAR: REAL:= 123.0; // not located
		END_STRUCT;
		END_TYPE
			Присваивает компоненты структуры еще не локализованным входным и выходным переменным, см. примечание 2

11a 11b	Прямо производный тип данных	TYPE CNT: UINT; FREQ: REAL:= 50.0; ANALOG_CHANNEL_CONFIG: ANALOG_CHANNEL_CONFIGURATION := (MIN_SCALE:= 0, MAX_SCALE:= 4000); END_TYPE	Инициализация Новая инициализация
12	Инициализация с использованием константных выражений	TYPE PIx2: REAL:= 2 * 3.1416; END_TYPE	Использует константное выражение

Примечание 1 — Возможно объявление типа данных без инициализации (свойство «a») или с инициализацией (свойство «b»). Если поддерживается свойство «a», тип данных инициализируется с неявным начальным значением. Если поддерживается свойство «b», тип данных инициализируется с данным значением или неявным начальным значением, если начальное значение не дано.

Примечание 2 — Переменные с прямо представленными элементами — частично определенными, используя «*», не могут использоваться в секциях VAR_INPUT или VAR_IN_OUT.

6.4.4.2 Перечислимый тип данных

6.4.4.2.1 Общие положения

Объявление перечислимого типа данных означает, что каждый элемент данных этого типа может принимать только значения, указанные в соответствующем перечне идентификатора, как показано в таблице 11.

Перечень перечисления определяет упорядоченное множество перечислимых значений, начиная с первого идентификатора и оканчивая последним.

Различные перечислимые типы данных могут использовать одинаковые идентификаторы для перечислимых значений. Максимально допустимое число перечислимых значений определяется разработчиком.

Для обеспечения уникальной идентификации при использовании в конкретном контексте, перечислимые литералы могут уточняться префиксом, состоящим из имени ассоциированного типа данных и символа номера «#», аналогично типизированным литералам. В перечне перечисления префиксы не используются.

Происходит ошибка, если в перечислимом литерале недостаточно информации для однозначного определения его значения (см. пример ниже).

Пример — Перечислимый тип данных

```
TYPE
    Traffic_light: (Red, Amber, Green);
    Painting_colors: (Red, Yellow, Green, Blue):= Blue;
END_TYPE
VAR
    My_Traffic_light: Traffic_light:= Red;
```

```
END_VAR
    IF My_Traffic_light = Traffic_light#Amber THEN ... // OK
    IF My_Traffic_light = Traffic_light#Red THEN ... // OK
    IF My_Traffic_light = Amber THEN ... // OK — идентификатор Amber уникален
    IF My_Traffic_light = Red THEN ... // ОШИБКА — идентификатор Red не является уникальным
```

6.4.4.2.2 Инициализация

Неявное начальное значение перечислимого типа данных — первый идентификатор в связанном перечне перечисления.

Пользователь может инициализировать тип данных определенным пользователем значением из перечня перечислимых значений данного типа. Такая инициализация имеет приоритет.

Как показано в таблице 11 для ANALOG_SIGNAL_RANGE, определенное пользователем начальное значение перечислимого типа данных — это присвоенное значение UNIPOLAR_1_5V.

Определенное пользователем присваивание начального значения типа данных является свойством в таблице 11.

6.4.4.3 Тип данных с именованными значениями

6.4.4.3.1 Общие положения

Связанным с перечислимым типом данных, где перечислимые идентификаторы не заданы пользователем, является перечислимый тип данных с именованными значениями. Объявление определяет тип данных и присваивает значения именованных переменных, как показано в таблице 11.

Объявление именованных значений не ограничивает диапазон значений переменных этого типа, то есть переменной могут быть присвоены другие константы, или значение может определяться вычислением.

Для обеспечения уникальной идентификации при использовании в конкретном контексте, именованные значения могут уточняться префиксом, состоящим из имени ассоциированного типа данных и символа номера «#», аналогично типизированным литералам.

В перечне объявления префиксы не используются. Происходит ошибка, если в перечислимом литерале недостаточно информации для однозначного определения его значения (см. пример).

Пример — Тип данных с именованными значениями

```
TYPE
    Traffic_light: INT (Red:= 1, Amber := 2, Green:= 3):= Green;
    Painting_colors: INT (Red:= 1, Yellow:= 2, Green:= 3, Blue:= 4):= Blue;
END_TYPE

VAR
    My_Traffic_light: Traffic_light;
END_VAR

My_Traffic_light:= 27; // Присваивание константы IF
My_Traffic_light = Amber THEN ...// Присваивание выражения
                                 // Примечание — Это невозможно для перечислимых значений
My_Traffic_light:= Traffic_light#Red + 1;

    IF My_Traffic_light      = 123 THEN ...                    // OK
    IF My_Traffic_light      = Traffic_light#Amber  THEN ...   // OK
    IF My_Traffic_light      = Traffic_light#Red    THEN ...   // OK
    IF My_Traffic_light      = Amber THEN ...                  // OK — идентификатор Amber
    уникален
    IF My_Traffic_light      = Red THEN ...                    // ОШИБКА — идентификатор
    Red не является уникальным
```

6.4.4.3.2 Инициализация

Неявное значение для типа данных с именованными значениями — это первый элемент данных в перечне перечисления. В приведенном выше примере для Traffic_light таким элементом является Red.

Пользователь может инициализировать тип данных определенным пользователем значением. Инициализация не ограничивается именованными значениями — может использоваться любое значение из диапазона базового типа. Такая инициализация имеет приоритет.

В пример, определенным пользователем начальным значением перечислимого типа для Traffic_light является Green.

Определенное пользователем присваивание начального значения типа данных является свойством в таблице 11.

6.4.4.4 Тип-диапазон

6.4.4.4.1 Общие положения

Декларацией типа-диапазона определено, что значение любого элемента данных этого типа может принимать значения между указанными верхними и нижними пределами (включительно), как показано в таблице 11.

Пределы в типе-диапазоне должны быть литералами или константными выражениями.

Пример —

```
TYPE
    ANALOG_DATA: INT(-4095 .. 4095):= 0;
END_TYPE
```

6.4.4.4.2 Инициализация

Неявные начальные значения для типов данных с диапазоном — это первый (нижний) предел диапазона.

Пользователь может инициализировать тип данных определенным пользователем значением из диапазона. Такая инициализация имеет приоритет.

Например, как показано в примере, приведенном в таблице 11, неявное начальное значение элементов типа ANALOG_DATA равно -4095, в то время, как при явной инициализации, неявное начальное значение равно нулю (как объявлено).

6.4.4.5 Типы данных — массивы

6.4.4.5.1 Общие положения

Объявление типа данных — массива определяет, что должно быть выделено достаточное количество памяти для каждого элемента этого типа, чтобы хранить все данные, которые могут быть индексированы указанным поддиапазоном (поддиапазонами) индексов, как показано в таблице 11.

Массив — это совокупность элементов данных одинакового типа. В качестве типа элемента массива могут использоваться элементарные и определенные пользователем типы данных, типы функциональных блоков и классы. На данную совокупность элементов данных ссылаются с помощью одного или более индексов, заключенных в квадратные скобки и разделенных запятыми. Если значение индекса выходит за пределы, указанные в объявлении массива, возникает ошибка.

Примечание — Для вычисляемых индексов такая ошибка может быть обнаружена только во время выполнения.

Максимальное число индексов массива, максимальный размер массива и максимальный диапазон значений индекса определяются разработчиком.

Пределы в диапазона индекса должны быть литералами или константными выражениями. Массивы переменной длины определены в 6.5.3.

В языке ST индекс является выражением, производящим значение, соответствующее одному из подтипов родового типа ANY_INT.

Форма индексов в языке IL и графических языках, определенных в разделе 8, ограничена одноэлементными переменными или целыми литералами.

Пример —

а) Объявление массива

```
VAR myANALOG_16: ARRAY [1..16] OF ANALOG_DATA;
    := [8(-4095), 8(4095)];        // определенные пользователем начальные значения
END_VAR
```

b) Использование переменных массива в языке ST может быть следующим:

OUTARY[6,SYM]:= INARY[0] + INARY[7] — INARY[i] * %IW62.

6.4.4.5.2 Инициализация

Неявное начальное значение каждого элемента массива — это начальное значение, определенное для типа данных элементов массива.

Пользователь может инициализировать тип массива значением, определенным пользователем. Такая инициализация имеет приоритет.

Определенное пользователем начальное значение массива назначается в форме списка, в котором могут использоваться скобки для обозначения повторений.

Во время инициализации типов данных — массивов, самый правый индекс массива изменяется быстрее остальных при наполнении массива из списка начальных значений.

Пример — Инициализация массива

A: ARRAY [0..5] OF INT:= [2(1, 2, 3)]

эквивалентно последовательности инициализации 1, 2, 3, 1, 2, 3.

Если число начальных значений, данных в перечне инициализации превышает число входов массива, лишние (самые правые начальные значения будут отброшены. Если число начальных значений меньше, чем число входов массива, оставшиеся входы массива будут заполнены неявными начальными значениями для соответствующего типа данных. В любом случае, пользователь будет предупрежден об этой ситуации во время подготовки программы для выполнения.

Определенное пользователем присваивание начального значения типа данных является свойством в таблице 11.

6.4.4.6 Структурированный тип данных

6.4.4.6.1 Общие положения

Объявление структурированного типа данных (STRUCT) указывает, что этот тип данных содержит совокупность подэлементов определенных типов, к которым можно осуществлять доступ под определенным именем, как показано в таблице 11.

Элемент структурированного типа данных представляется двумя или более идентификаторами, разделенными точкой «.».

Первый идентификатор представляет имя структурированного элемента, а последующие идентификаторы представляют последовательность имен элементов для доступа к конкретному элементу данных в структуре данных.

В качестве типа элемента структуры могут использоваться элементарные и определенные пользователем типы данных, типы функциональных блоков и классы.

Например, элемент типа данных ANALOG_CHANNEL_CONFIGURATION, объявленный таблице 11, будет содержать подэлемент RANGE типа ANALOG_SIGNAL_RANGE, подэлемент MIN_SCALE типа ANALOG_DATA и подэлемент MAX_SCALE типа ANALOG_DATA.

Максимальное число элементов структуры, максимальное количество данных, которое может содержаться в структуре и максимальное число вложенных уровней адресации структурного элемента определяются разработчиком.

Две структурированных переменных являются совместимыми по присваиванию, только если они имеют одинаковый тип данных.

Пример — Объявление и использование структурированного типа данных и структурированной переменной.

a) Объявление структурированного типа данных

```
TYPE
   ANALOG_SIGNAL_RANGE: (BIPOLAR_10V,
                         UNIPOLAR_10V);
   ANALOG_DATA: INT (-4095 .. 4095);
   ANALOG_CHANNEL_CONFIGURATION:
      STRUCT
         RANGE:       ANALOG_SIGNAL_RANGE;
         MIN_SCALE:   ANALOG_DATA:
         MAX_SCALE: ANALOG_DATA;
```

```
    END_STRUCT;
END_TYPE
```

b) Объявление структурированной переменной

```
VAR
    MODULE_CONFIG:ANALOG_CHANNEL_CONFIGURATION;
    MODULE_8_CONF:          ARRAY [1..8] OF ANALOG_CHANNEL_CONFIGURATION;
END_VAR
```

c) Использование переменных массива в языке ST:

```
MODULE_CONFIG.MIN_SCALE:= -2047;
MODULE_8_CONF[5].RANGE:= BIPOLAR_10V.
```

6.4.4.6.2 Инициализация

Неявные значения компонентов структуры даются их индивидуальными типами данных.

Пользователь может инициализировать компоненты структуры значениями, определенными пользователем. Такая инициализация имеет приоритет.

Пользователь может также инициализировать ранее определенную структуру, используя перечень присваиваний компонентам структуры. Данная инициализация имеет более высокий приоритет, чем неявная инициализация и инициализация компонентов.

Пример — Инициализация структуры

a) Объявление с инициализацией структурированного типа данных

```
TYPE
    ANALOG_SIGNAL_RANGE:
            (BIPOLAR_10V,
            UNIPOLAR_10V):= UNIPOLAR_10V;
    ANALOG_DATA: INT (-4095 ..4095);
    ANALOG_CHANNEL_CONFIGURATION;
STRUCT
        RANGE:          ANALOG_SIGNAL_RANGE;
        MIN_SCALE: ANALOG_DATA:= -4095;
        MAX_SCALE: ANALOG_DATA:= -4096;
END_STRUCT;
    ANALOG_8BI_CONFIGURATION:
      ARRAY [1..8] OF ANALOG_CHANNEL_CONFIGURATION
          := [8((RANGE:= BIPOLAR_10V))];
END_TYPE
```

b) Объявление с инициализацией структурированной переменной

```
VAR
    MODULE_CONFIG:ANALOG_CHANNEL_CONFIGURATION
      := (RANGE:= BIPOLAR_10V, MIN_SCALE:= -1023);
    MODULE_8_SMALL: ANALOG_8BI_CONFIGURATION
      := [8 ((MIN_SCALE:= -2047, MAX_SCALE:= 2048))];
END_VAR
```

6.4.4.7 Относительное положение элементов структурированных типов данных (AT)

6.4.4.7.1 Общие положения

Положения (адреса) элементов структурированного типа могут быть определены относительно начала структуры.

В этом случае, за именем компонента этой структуры следует ключевое слово AT и относительный адрес.

Объявление может содержать разрывы в расположении памяти.

Относительный адрес состоит из символа процента «%», определителя битового или байтового адреса. Байтовый адрес — это целый литерал без знака, обозначающий смещение в байтах. Битовый адрес состоит из смещения в байтах, следующего символа точки «.» и смещения в битах, являющегося

целым литералом без знака в диапазоне от 0 до 7. В относительном адресе не допускаются пробельные символы.

Компоненты структуры не должны перекрываться в расположении памяти, за исключением ситуации, когда в объявлении имеется ключевое слово OVERLAP.

Перекрытие строк находится вне области применения настоящего стандарта.

Примечание — Отсчет битового смещения начинается от самого правого бита с 0. Отсчет битового смещения начинается от начала структуры с 0.

Пример — Относительные адреса и перекрытие в структуре

```
TYPE
    Com1_data: STRUCT
head    AT %B0:     INT;          // в положении 0
length  AT %B2:     USINT=26; // в положении 2
flag1   AT %X3.0:   BOOL;     // в положении 3.0
end     AT %B25:    BYTE;     // в положении 25, оставляя разрыв
    END_STRUCT;

    Com2_data: STRUCT OVERLAP
head    AT %B0:     INT;      // в положении 0
length  AT %B2:     USINT;    // в положении 2
flag2   AT %X3.3:   BOOL;     // в положении 3.3
data1   AT %B5:     BYTE;     // в положении 5, перекрывается
data2   AT %B5:     REAL;     // в положении от 5 до 8
end     AT %B19:    BYTE;     // по адресу 19, оставляя разрыв
    END_STRUCT;

Com_data: STRUCT OVERLAP // C1 и C2 перекрываются
    C1 at %B0: Com1_data:
    C2 at %B0: Com2_data;
  END_STRUCT;
END_TYPE
```

6.4.4.7.2 Инициализация

Структуры с перекрытием не могут явно инициализироваться.

6.4.4.8 Прямо представленные компоненты структуры — частично определенные с использованием «*»

Символ звездочки «*» в таблице 11 может использоваться, чтобы обозначить еще не полностью определенные адреса для прямо представленных компонентов структуры.

Пример — Присваивание компонентов структуры еще не локализованным входным и выходным переменным.

```
TYPE
  HW_COMP: STRUCT;
    IN          AT %I*: BOOL;
    VAL         AT %I*: DWORD;
    OUT         AT %Q*: BOOL; OUT_VAR          AT %Q*: WORD;
    ITNL_VAR: REAL; // еще не локализован END_STRUCT;
END_TYPE
```

В случае, когда прямо представленный компонент структуры используется для назначения расположения в части объявлений программы, типа функционального блока или класса, на месте префикса размера и целого со знаком может использоваться звездочка «*» для указания того, что прямое представление еще не полностью определено.

Использование этого свойства требует, чтобы положение структурированной переменной, объявленной таким образом, было полностью определено внутри конструкции VAR_CONFIG...END_VAR конфигурации для каждого экземпляра охватывающего типа.

Переменные такого типа не могут использоваться в секциях VAR_INPUT, VAR_IN_OUT и VAR_TEMP.

Ошибка возникает, если отсутствует какая-либо полная спецификация в конструкции VAR_CONFIG...END_VAR для какой-либо неполной спецификации адреса, выраженной символом «*» в любом экземпляре программы или функционального блока, который содержит такие неполные спецификации.

6.4.4.9 Прямо порожденный тип данных

6.4.4.9.1 Общие положения

Определенные пользователем типы данных могут быть прямо порождены из элементарного типа данных или определенного пользователем типа данных.

Это может быть использовано для определения специфических для типа начальных значений.

Пример — Прямо порожденный тип данных

```
TYPE
    myInt1123:   INT:= 123;
    myNewArrayType: ANALOG_16_INPUT_DATA := [8(-1023), 8(1023)];
    Com3_data: Com2_data:= (head:= 3, length:=40);
END_TYPE
    .R1: REAL:= 1.0;
    R2: R1;
```

6.4.4.9.2 Инициализация

Неявное начальное значение равно начальному значению типа данных, из которого порожден новый тип. Пользователь может инициализировать тип данных определенным пользователем значением. Такая инициализация имеет приоритет.

Определенное пользователем начальное значение элементов структуры может быть объявлено в перечне, заключенном в скобки и следующим за идентификатором типа данных. Элементы, начальное значение которых не перечислено в перечне инициализации, имеют неявные начальные значения, объявленные для них в объявлении оригинального типа данных.

Пример 1 — Использование определенных пользователем типов данных

С учетом объявлений ANALOG_16_INPUT_DATA в таблице 11 и объявления VAR INS: ANALOG_16_INPUT_DATA; END_VAR переменные от INS[1] до INS[16] могут использоваться везде, где могут использоваться переменные типа INT.

Пример 2

Аналогично, с учетом объявления Com_data в таблице 11 и, дополнительно, объявления VAR telegram: Com_data; END_VAR переменная telegram.length может использоваться везде, где может использоваться тип USINT.

Пример 3

Это правило может применяться рекурсивно:

С учетом объявления ANALOG_16_ INPUT_CONFIGURATION, ANALOG_CHANNEL_CONFIGURATION и ANALOG_DATA в таблице 11 и объявления VAR CONF: ANALOG_16_INPUT_CONFIGURATION; END_VAR переменная CONF.CHANNEL[2].MIN_SCALE может использоваться везде, где может использоваться тип INT.

6.4.4.10 Указатели

6.4.4.10.1 Объявление указателя

Указатель — это переменная, которая содержит только ссылку на переменную или на экземпляр функционального блока. Указатель может иметь значение NULL, то есть он не ссылается ни на что.

Указатели объявляются для определенных типов данных, используя ключевое слово REF_TO и ссылочный тип данных — тип данных, на который производится ссылка. Ссылочный тип данных уже должен быть определен. Им может являться элементарный тип данных или определенный пользователем тип данных.

Примечание — Указатели без привязки к типу данных выходят за пределы настоящего стандарта.

Пример 1

```
TYPE
    myArrayType:      ARRAY[0..999] OF INT;
    myRefArrType:     REF_TO myArrayType;              // определение указателя
    myArrOfRefType:   ARRAY [0..12] OF myRefArrType; // определение массива ссылок
END_TYPE
VAR
    myArray1:         myArrayType;
    myRefArr1:        myRefArrType;        // определение указателя
    myArrOfRef:       myArrOfRefType;      // определение массива указателей
END_VAR
```

Ссылка должна ссылаться только на переменные указанного ссылочного типа данных. Указатели на прямо порождаемые типы данных обрабатываются как псевдонимы указателей на базовый тип данных. Прямое порождение может применяться несколько раз.

Пример 2

```
TYPE
    myArrType1: ARRAY[0..999] OF INT;
    myArrType2: myArrType1;
    myRefType1: REF_TO myArrType1;
    myRefType2: REF_TO myArrType2;
END_TYPE
```

myRefType1 и myRefType2 могут ссылаться на переменные типа ARRAY[0..999] OF INT и производных типов данных.

Ссылочный тип данных указателя может также являться типом функционального блока или классом. Указатель базового типа может также ссылаться на экземпляры, порожденные из этого типа данных.

Пример 3

```
CLASS F1 ...              END_CLASS;
CLASS F2 EXTENDS F1 ...   END_CLASS;
TYPE
    myRefF1:   REF_TO F1;
    myRefF2:   REF_TO F2;
END_TYPE
```

Указатели типа myRefF1 могут ссылаться на экземпляры классов F1, F2 и на производные от них классы. Однако указатели типа myRefF2 не могут ссылаться на экземпляры класса F1, а могут ссылаться только на экземпляры класса F2 и производные от него, так как класс F1 может не поддерживать методы и переменные расширенного класса F2.

6.4.4.10.2 Инициализация указателей

Указатели могут инициализироваться значением NULL (неявно) или адресом уже объявленных переменных, экземпляров функционального блока или класса.

Пример —

```
FUNCTION_BLOCK F1 ... END_FUNCTION_BLOCK;
VAR
    myInt:     INT;
    myRefInt:  REF_TO INT:= REF(myInt);
    myF1:  F1;
    myRefF1:   REF_TO F1:= REF(myF1);
END_VAR
```

6.4.4.10.3 Операции с указателями

Оператор REF() возвращает указатель на заданную переменную или экземпляр. Ссылочным типом данных возвращенного указателя является тип данных заданной переменной. Применение оператора REF() к временной переменной (например, переменным любой секции VAR_TEMP или любым переменным внутри функций) не разрешается.

Указатель может быть присвоен другому указателю, если его ссылочный тип данных эквивалентен базовому типу или является ссылочным типом данных присвоенного указателя.

Указатели могут присваиваться параметрам функций, функциональных блоков и методов в вызове, если ссылочный тип данных параметра эквивалентен базовому типу или является базовым типом ссылочного типа данных. Ссылки не могут использоваться как входные-выходные переменные.

Если указатель присвоен указателю такого же типа данных, то последний ссылается на ту же самую переменную. В таком случае, прямо порожденный тип данных рассматривается также, как его базовый тип.

Если указатель присваивается указателю на такой же тип функционального блока или базовый тип функционального блока, то затем этот указатель указывает на тот же самый экземпляр, но является все еще связанным со своим типом функционального блока, то есть может использовать только переменные и методы своего ссылочного типа данных.

Разыменование указателей осуществляется явно.

Указатель разыменовывается использованием предшествующего символа крышки «^».

Разыменованный указатель может использоваться так же, как прямо используется переменная. Разыменованный указатель на NULL является ошибкой.

Примечание 1 — Возможные проверки указателей на NULL может производиться во время компиляции, системой поддержки выполнения программы или прикладной программой.

Конструкция REF() и оператор разыменования «^» используются в графических языках при определении операндов.

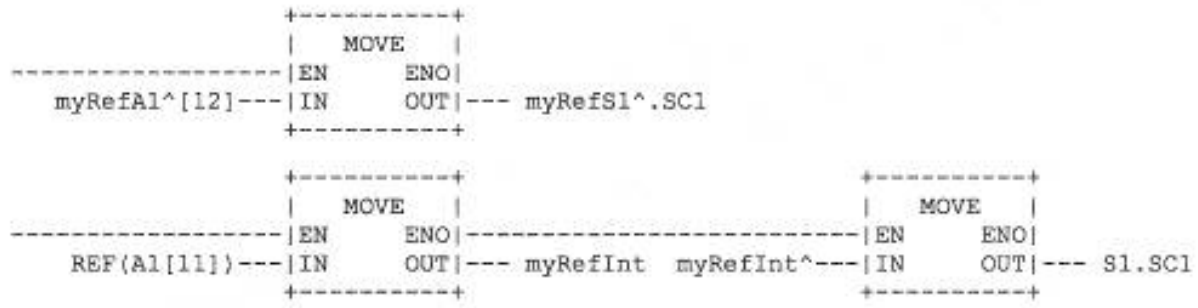
Примечание 2 — Арифметические операции с указателями не рекомендуются и не входят в задачу настоящего стандарта.

Пример 1

```
TYPE
    S1: STRUCT
    SC1: INT;
    SC2: REAL;
    END_STRUCT;
A1: ARRAY[1..99] OF INT;
END_TYPE
VAR
    myS1: S1;
    myA1: A1;
    myRefS1: REF_TO S1:= REF(myS1);
    myRefA1: REF_TO A1:= REF(myA1);
    myRefInt: REF_TO INT:= REF(myA1[1]);
END_VAR
    myRefS1^.SC1:= myRefA1^[12]; // в данном случае, это эквивалентно S1.SC1:= A1[12];
    myRefInt:= REF(A1[11]);
    S1.SC1:= myRefInt^; // присваивает значение переменной A1[11] элементу структуры S1.SC1
```

Пример 2

Графическое представление операторов из примера 1



В таблице 12 приведены свойства операций с указателями.

Таблица 12 — Операции с указателями

Номер	Описание	Пример
	Объявление	
1	// определение типа указателя	TYPE myRefType: REF_TO INT; END_TYPE
	Присваивание и сравнение	
2a	Присваивание указателя указателю	<указатель>:= <указатель> myRefType1:= myRefType2;
2b	Присваивание указателя параметру функции, функционального блока или метода	myFB (a:= myRefS1); Типы должны быть эквивалентными
2c	Сравнение с NULL	IF myInt = NULL THEN ...
	Создание ссылки	
3a	REF(<переменная>) Предоставляет типизированную ссылку на переменную	myRefA1:= REF (A1);
3b	REF(<экземпляр функционального блока>) Предоставляет типизированную ссылку на экземпляр функционального блока или класса	myRefFB1:= REF(myFB1)
	Разыменование	
4	<указатель>^ Предоставляет содержимое переменной или содержимое экземпляра, на которые ссылается переменная указателя	myInt:= myA1Ref^[12];

6.5 Переменные

6.5.1 Объявление и инициализация переменных

6.5.1.1 Общие положения

Переменные предоставляют средства идентификации объектов данных, содержание которых может изменяться. Например, данные, связанные с входами, выходами или памятью программируемого контроллера.

В отличие от литералов, которые являются внешним представлением данных, переменные могут изменять свое значение с течением времени.

6.5.1.2 Объявление

Переменные объявляются внутри одной из секций переменных.

Переменные можно объявлять, используя:

- элементарный тип данных; или
- предварительно определенный пользователем тип; или
- тип указателя; или
- прямо определенный пользователем тип.

Возможны следующие виды переменной:

- одноэлементная переменная, то есть переменная, тип которой либо:
  - элементарный тип данных; или
  - определенное пользователем перечисление или тип-диапазон; или
  - определенный пользователем тип, происхождение которого, определяемое рекурсивно, прослеживается до элементарного типа, типа перечисления или типа-диапазона;
- многоэлементная переменная, то есть переменная, которая представляет массив ARRAY или структуру STRUCT;
- указатель, то есть переменную, которая ссылается на другую переменную или экземпляр функционального блока.

Объявление переменной состоит из следующих элементов:

- списка имен объявляемых переменных;
- символа двоеточия «:»;
- типа данных с необязательной инициализации, специфичной для различных видов переменных.

Пример —

```
TYPE
    myType: ARRAY [1..9] OF INT; // предварительно определенный пользователем тип
END_TYPE
VAR
    myVar1, myVar1a: INT;          // две переменные, используя элементарный тип
    myVar2: myType;                // используя предварительно определенный пользователем тип
    myVar3: ARRAY [1..8] OF REAL;  // используя прямо определенный пользователем тип
END_VAR
```

6.5.1.3 Инициализация переменных

Неявным начальным значением переменной (переменных) являются:

1 Неявное начальное значение (значения) лежащих в основе элементарных типов данных, как определено в таблице 10.

2 NULL, если переменная является указателем.

3 Определенное пользователем значение (значения) назначенного типа данных.

Это значение факультативно может быть определено использованием оператора присваивания «:=» в определении типа TYPE, как показано в таблице 11.

4 Пользователем значение (значения) переменной.

Это значение факультативно может быть определено использованием оператора присваивания «:=» в объявлении переменной VAR (см. таблицу 14).

Определенное пользователем значение может быть литералом (например, -123, 1.55, «abc») или константным выражением (например, 12*24).

Начальные значения не могут задаваться в объявлениях VAR_EXTERNAL.

Начальные значения могут также определяться с использованием определяемого экземпляром свойства инициализации, предоставляемого конструкцией VAR_CONFIG...END_VAR. Определяемые экземпляром начальные значения всегда замещают специфические для типа начальные значения.

Таблица 13 — Определение переменных

Номер	Описание	Пример	Объяснение
1	Переменная элементарного типа данных	VAR MYBIT: BOOL; OKAY: STRING[10]; VALVE_POS AT %QW28: INT; END_VAR	Распределяет бит памяти логической переменной MYBIT Распределяет память для хранения строки с максимальной длиной 10 символов
2	Переменная с определенным пользователем типом данных	VAR myVAR: myType; END_VAR	Описание переменных с пользовательским типом данных
3	Массив	VAR BITS: ARRAY[0..7] OF BOOL; TBT: ARRAY [1..2, 1..3] OF INT; OUTA AT %QW6: ARRAY[0..9] OF INT; END_VAR	
4	Указатель	VAR myRefInt: REF_TO INT; END_VAR	Определение переменной, являющейся указателем

Таблица 14 — Инициализация переменных

Номер	Описание	Пример	Объяснение
1	Инициализация переменной с элементарным типом данных	VAR MYBIT: BOOL := 1; OKAY: STRING[10] := 'OK'; VALVE_POS AT %QW28: INT:= 100; END_VAR	Распределяет бит памяти логической переменной MYBIT с начальным значением от 1. Распределяет память для хранения строки с максимальной длиной десяти символов После инициализации строка имеет длину 2 и содержит двухбайтовую последовательность символов «ОК» (десятичные 79 и 75, соответственно), в порядке, подходящем для печати символьной строки
2	Инициализация переменной с определенным пользователем типом данных	TYPE myType: ... END_TYPE VAR myVAR: myType:= ... // инициализация END_VAR	Объявление определенного пользователем типа с инициализацией и без инициализации Описание с предварительной инициализацией переменной с определенным пользователем типом данных
3	Массив	VAR BITS: ARRAY[0..7] OF BOOL :=[1,1,0,0,0,1,0,0]; TBT: ARRAY [1..2, 1..3] OF INT := [9,8,3(10),6]; OUTARY AT %QW6: ARRAY[0..9] OF INT := [10(1)]; END_VAR	Распределяет 8 битов памяти для хранения начальных значений BITS[0]:= 1, BITS[1]:= 1,..., BITS[6]:= 0, BITS[7]:= 0. Распределяет целый массив TBT размером 2*3 с начальными значениями TBT[1,1]:= 9, TBT[1,2]:= 8, TBT[1,3]:= 10, TBT[2,1]:= 10, TBT[2,2]:= 10, TBT[2,3]:= 6.

Окончание таблицы 14

Номер	Описание	Пример	Объяснение
4	Объявление и инициализация констант	VAR CONSTANT PI: REAL:= 3.141592; PI2: REAL:= 2.0*PI; END_VAR	Константа Символическая константа PI
5	Инициализация с использованием константных выражений	VAR PIx2: REAL:= 2.0 * 3.1416; END_VAR	Использует константное выражение
6	Инициализация указателя	VAR myRefInt: REF_TO INT := REF(myINT); END_VAR	Инициализирует переменную-указатель myRefInt ссылкой на переменную myINT.

6.5.2 Секции переменных

6.5.2.1 Общие положения

Каждая декларация программного компонента (POU), то есть функционального блока, функции или программы и, дополнительно, метода, начинается частями от нуля или более объявлений, которые определяют имена, типы (и, если необходимо, физическое или логическое расположение и инициализацию) переменных, используемых в организационной единице.

Часть деклараций программного компонента POU может содержать различные секции VAR, в зависимости от вида программного компонента POU.

Переменные могут объявляться в различных текстовых конструкциях VAR ... END_VAR, включающих квалификаторы, такие как RETAIN или PUBLIC, при необходимости. Квалификаторы секций переменных кратко приведены на рисунке 7.

Ключевое слово	**Использование переменной**
Секции VAR: зависящие от типа программного компонента POU (функция, функциональный блок, программа) или метода	
VAR	Внутренние по отношению к объекту (функция, функциональный и т. д.).
VAR_INPUT	Предоставленные извне, не модифицируются внутри объекта.
VAR_OUTPUT	Предоставляемых объектом для внешних объектов.
VAR_IN_OUT	Предоставляются внешними объектами, могут модифицироваться внутри объекта, предоставляться для внешнего объекта.
VAR_EXTERNAL	Предоставляемые конфигурацией через VAR_GLOBAL.
VAR_GLOBAL	Объявление глобальной переменной.
VAR_ACCESS	Объявление пути доступа.
VAR_TEMP	Временное хранилище для переменных в функциональных блоках, методах и программах.
VAR_CONFIG	Специфическая для экземпляра инициализация и назначение расположения.
(END_VAR)	Заканчивает описанные выше секции VAR.

Рисунок 7 — Ключевые слова объявлений переменных (сводка), лист 1

Квалификаторы: могут следовать за описанными выше ключевыми словами	
RETAIN	Сохраняемые переменные.
NON_RETAIN	Несохраняемые переменные.

PROTECTED	Доступ только изнутри собственного объекта и его производных объектов (неявно).
PUBLIC	Разрешен доступ из всех объектов.
PRIVATE	Доступ только из собственно объекта.
INTERNAL	Доступ только внутри одного пространства имен.
CONSTANT[a)]	Константа (неизменяемая переменная).

Примечание — Использование данных ключевых слов является свойством программного компонента и элемента конфигурации, в котором они используются.

[a)] Экземпляры функциональных блоков не объявляются в секциях переменных с квалификатором CONSTANT.

Рисунок 7, лист 2

- VAR

Переменные, объявленные в секции VAR ... END_VAR сохраняются от одного вызова программы или экземпляра функционального блока до другого.

В пределах функций, переменные, объявленные в этой секции, не сохраняются между вызовами функций.

- VAR_TEMP

В пределах программных компонент, переменные могут объявляться только секции VAR_TEMP... END_VAR.

Для функций и методов ключевые слова VAR и VAR_TEMP эквивалентны.

Данные переменные распределяются и инициализируются специфическими для типа неявными значениями, и не сохраняются между вызовами.

- VAR_INPUT, VAR_OUTPUT и VAR_IN_OUT

Переменные, объявленные в данных секциях, являются формальными параметрами функций, типов функциональных блоков и методов.

- VAR_GLOBAL и VAR_EXTERNAL

Переменные, объявленные в секции VAR_GLOBAL, могут использоваться в других программных компонентах, если они повторно объявлены там в секции VAR_EXTERNAL .

На рисунке 8 показано использование ключевых слов VAR_GLOBAL, VAR_EXTERNAL и CONSTANT.

Объявление в элементе, содержащем переменную	Объявление в элементе, использующем переменную			Разрешено?
VAR_GLOBAL X	VAR_EXTERNAL	CONSTANT	X	Да
VAR_GLOBAL X	VAR_EXTERNAL	X		Да
VAR_GLOBAL CONSTANT X	VAR_EXTERNAL	CONSTANT	X	Да
VAR_GLOBAL CONSTANT X	VAR_EXTERNAL	X		П/п
Примечание — Использование секции VAR_EXTERNAL в содержащемся элементе может приводить к непредвиденному поведению. Например, когда значение внешней переменной изменяется другим содержащимся элементом в одном и том же содержащемся элементе.				

Рисунок 8 — Использование VAR_GLOBAL, VAR_EXTERNAL и CONSTANT (правила)

- VAR_ACCESS

Доступ к переменным, объявленным в секции VAR_ACCESS, может производиться с использованием пути доступа, заданного в объявлении.

- VAR_CONFIG

Конструкция VAR_CONFIG...END_VAR предоставляет средства для назначения специфического для экземпляра размещения символически представленных переменных, используя символ «*» или для присвоения специфических для экземпляра начальных значений символически представленным переменным, или и для того и для другого.

6.5.2.2 Область действия объявлений

Область действия (диапазон применимости) деклараций, содержащихся в разделе деклараций, является локальной для программных компонент, в которых данный раздел деклараций содержится. То есть объявленные переменные не будут доступны для других программных компонентов, за исключением явных параметров, передаваемых через переменные, которые объявлены как входы и выходы этих компонент.

Исключением из данного правила являются переменные, объявленные как глобальные. Такие переменные доступны для программных компонент только через объявление VAR_EXTERNAL. Тип переменных, объявленных в блоке VAR_EXTERNAL, должен быть согласован с типом, объявленным в блоке VAR_GLOBAL, связанных программ, конфигурации и ресурсе.

Ошибка возникает, если:

- какая-либо программная компонента пытается изменить значение переменной, которая была объявлена с квалификатором CONSTANT или в секции VAR_INPUT;
- переменная, объявленная как VAR_GLOBAL CONSTANT, в элементе конфигурации или программном компоненте («содержащем элементе») используется в объявлении VAR_EXTERNAL (без квалификатора CONSTANT) любого элемента, содержащегося в пределах охватывающего элемента, как показано ниже.

Максимальное число переменных, допустимых в блоке объявления переменных, определяется разработчиком.

6.5.3 Переменные типа ARRAY переменной длины

Массивы переменной длины могут использоваться только как:

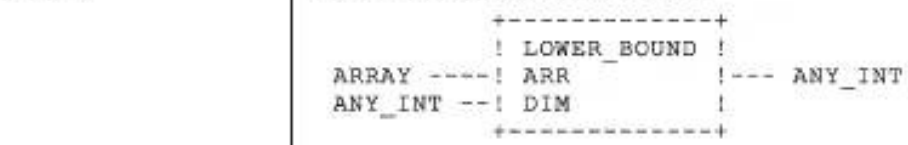
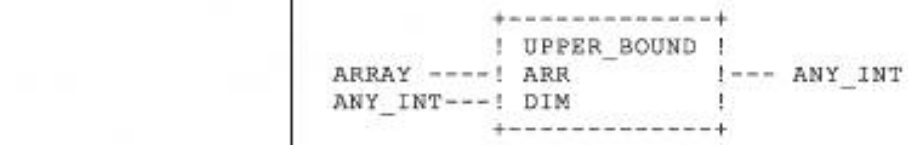
- входные, выходные или входные-выходные переменные функций и методов;
- входные-выходные переменные функциональных блоков.

Число размерностей массива и фактических и формальных параметров должны быть одинаковыми. Они определяются, используя символ звездочки как спецификацию неопределенного диапазона для границ индекса.

Массивы переменной длины предоставляют программам, функциям, функциональным блокам и методам средства использовать массивы с различными диапазонами индекса.

Для работы с массивами переменной длины предоставляются следующие стандартные функции (см. таблицу 15).

Таблица 15 — Переменные типа ARRAY переменной длины

Номер	Описание	Примеры
1	Декларация с использованием * ARRAY [*, *, . . .] OF тип данных	VAR_IN_OUT A: ARRAY [*, *] OF INT; END_VAR;
	Стандартные функции LOWER_BOUND и UPPER_BOUND	
2a	Графическое представление	Получить нижнюю границу массива:  Получить верхнюю границу массива: 

2b	Текстовое представление	Получить нижнюю границу 2-го измерения массива A:
		low2:= LOWER_BOUND (A, 2);
		Получить верхнюю границу 2-го измерения массива A:
		up2:= UPPER_BOUND (A, 2);

Пример 1

```
A1: ARRAY [1..10] OF INT:= [10(1)];

A2: ARRAY [1..20, -2..2] OF INT:= [20(5(1))];
    // в соответствии с инициализацией массива, см. 6.4.4.5.2
LOWER_BOUND (A1, 1)          →      1
UPPER_BOUND (A1, 1)          →     10
LOWER_BOUND (A2, 1)          →      1
UPPER_BOUND (A2, 1)          →     20
LOWER_BOUND (A2, 2)          →     -2
UPPER_BOUND (A2, 2)          →      2
LOWER_BOUND (A2, 0)          →   ошибка
LOWER_BOUND (A2, 3)          →   ошибка
```

Пример 2 — Суммирование массивов

```
FUNCTION SUM: INT;
VAR_IN_OUT A: ARRAY [*] OF INT; END_VAR;
VAR i, sum2: DINT; END_VAR;

sum2:= 0;
FOR i:= LOWER_BOUND(A,1) TO UPPER_BOUND(A,1)
    sum2:= sum2 + A[i]; END_FOR;
SUM:= sum2; END_FUNCTION

// SUM (A1)       →10
// SUM (A2[2]) →5
```

Пример 3 — Умножение матриц

```
FUNCTION MATRIX_MUL
VAR_INPUT
    A: ARRAY [*, *] OF INT;
    B: ARRAY [*, *] OF INT;
END_VAR;

VAR_OUTPUT C: ARRAY [*] OF INT; END_VAR;
VAR i, j, k, s: INT; END_VAR;

FOR i:= LOWER_BOUND(A, 1) TO UPPER_BOUND(A, 1)
    FOR j:= LOWER_BOUND(B, 2) TO UPPER_BOUND(B, 2)
    s:= 0;
      FOR k:= LOWER_BOUND(A,2) TO UPPER_BOUND(A, 2)
      s:= s + A[i,k] * B[k,j];
      END_FOR;
  C[i,j]:= s;
  END_FOR;
END_FOR;
END_FUNCTION
```

```
// Использование:
VAR
    A: ARRAY [1..5, 1..3] OF INT;
    B: ARRAY [1..3, 1..4] OF INT;
    C: ARRAY [1..5, 1..4] OF INT;
END_VAR
MATRIX_MUL (A, B, C);
```

6.5.4 Константные переменные

Константные переменные — это переменные, определенные в секции переменных, которая содержит ключевое слово CONSTANT. Применяются правила, определенные для выражений.

Пример — Константные переменные

```
VAR CONSTANT
    PI: REAL:= 3.141592;
    TwoPi: REAL:= 2.0*Pi;
END_VAR
```

6.5.5 Прямо представленные переменные (%)

6.5.5.1 Общие положения

Прямое представление одноэлементной переменной обеспечивается специальным символом, сформированных конкатенацией следующих элементов:

- знак процента «%»; и
- префиксы расположения I, Q или M; и
- префикс размера X (или никакого), B, W, D или L; и
- одно или более (см. ниже иерархическую адресацию) целых без знака, разделенных точками «.».

Пример —

```
%MW1.7.9
%ID12.6
%QL20
```

Разработчик определяет соответствие между прямым представлением переменной и физическим или логическим расположением адресуемой единицы в памяти на входе или на выходе.

Примечание — Использование прямо представленных переменных в телах функций, типов функциональных блоков, методов и типов программ ограничивает возможность многократного использования типов данных программных компонентов. Например, в системах программируемых контроллеров, где физические входы и выходы используются для различных целей.

Использование прямо представленных переменных разрешено в теле функций, функциональных блоках, программах, методах и в конфигурациях и ресурсе.

В таблице 16 представлены свойства прямо представленных переменных.

Использование прямо представленных переменных в теле программных компонентов и методов является не рекомендуемой функциональной возможностью.

Таблица 16 — Прямо представленные переменные

Номер	Описание		Пример	Объяснение
	Расположение (примечание 1)			
1	Расположение на входе	I	%IW215	Входное слово 215
2	Расположение на выходе	Q	%QB7	Выходной байт 7
3	Расположение в памяти	M	%MD48	Двойное слово по адресу памяти 48
	Размер			
4a	Размер одного байта	X	%IX1	Тип входных данных BOOL

Окончание таблицы 16

Номер	Описание		Пример	Объяснение
4b	Размер одного байта	Отсутствует	%I1	Тип входных данных BOOL
5	Размер байта (8 битов)	B	%IB2	Тип входных данных BYTE
6	Размер слова (16 битов)	W	%IW3	Тип входных данных WORD
7	Размер двойного слова (32 бита)	D	%ID4	Тип входных данных DWORD
8	Размер длинного слова (64 бита)	L	%IL5	Тип входных данных LWORD
	Адресация			
9	Простая адресация	%IX1	%IB0	Один уровень
10	Иерархическая адресация с использованием «.»	%QX7.5	%QX7.5 %MW1.7.9	Определенная разработчиком, например: два уровня, диапазоны 0..7; три уровня, диапазоны 1..16
11	Частично определенные переменные с использованием «*» (примечание 2)		%M*	

Примечание 1 — Национальные организации по стандартизации могут публиковать таблицы переводов этих префиксов.

Примечание 2 — Для использования символа звездочки «*» в этой таблице требуется наличие свойства VAR_CONFIG и наоборот.

6.5.5.2 Прямо представленные переменные — иерархическая адресация

Когда простое прямое представление (одного уровня) расширяется дополнительными цифровыми полями, разделенными точками, оно интерпретируется как иерархический физический или логический адрес. Самое левое поле представляет верхний уровень иерархии, уровень понижается при переходе вправо.

Пример — Иерархический адрес

%IW2.5.7.1

Например, данная переменная представляет первый «канал» (слово) седьмого «модуля» на пятом «стеллаже» второй «шины ввода/вывода» этой системы программируемого контроллера. Максимальное число уровней иерархической адресации определяется разработчиком.

Использование иерархической адресации (для разрешения доступа программы из одной системы программируемого контроллера к данным другого программируемого контроллера) считается расширением языка, специфическим для разработчика.

6.5.5.3 Объявление прямо представленных переменных (AT)

Объявлению прямо представленных переменных в соответствии с таблицей 16 (например, %IW6) может даваться символическое имя, используя ключевое слово AT.

Переменным с определенными пользователем типами данных (например, массиву) может быть назначен «абсолютный» адрес в памяти, используя AT. Расположение переменной определяет начальный адрес памяти и не требует размера, равного или превышающего размер данного прямого представления (то есть допустимы пустая память и перекрытие).

Пример — Использование прямого представления

```
VAR                                    Имя и тип для входа
  INP_0 AT %I0.0: BOOL;
  AT %IB12: REAL;
  PA_VAR AT %IB200: PA_VALUE;          Имя и определенный пользователем тип для размещения входа, начиная с %IB200
  OUTARY AT %QW6: ARRAY[0..9] OF INT;  Массив из 10 целых для размещения в смежных выходных адресах, начиная с %QW6
END_VAR
```

Для всех видов переменных, определенных в таблице 13, явное (определенное пользователем) расположение в памяти может быть объявлено, используя ключевое слово AT в сочетании с прямо представленными переменными (например, %MW10).

Если в одном или нескольких объявлениях это свойство не поддерживается, это должно быть указано в декларации соответствия разработчика.

Примечание — Инициализация входов системы (например, %IW10) определяется Разработчиком.

6.5.5.4 Прямо представленные переменные — частично определенные с использованием «*»

Запись с символом звездочки «*» может использоваться в назначениях адреса внутри программ и типов функциональных блоков для обозначения еще не полностью определенного расположения для прямо представленных переменных.

Пример —

```
VAR
  C2 AT %Q*: BYTE;
END_VAR
```

Назначает еще не расположенный выходной байт переменной типа битовой строки C2, длиной 1 байт.

В случае, когда прямо представленная переменная используется для назначения расположения внутренней переменной в части объявления программы, типа функционального блока, на месте префикса размера и целого со знаком может использоваться звездочка «*» для указания того, что прямое представление еще не полностью определено.

Переменные такого типа не могут использоваться в секциях VAR_INPUT и VAR_IN_OUT.

Использование этого свойства требует, чтобы положение структурированной переменной, объявленной таким образом, было полностью определено внутри конструкции VAR_CONFIG...END_VAR конфигурации для каждого экземпляра содержащего типа.

Ошибка возникает, если отсутствует какая-либо полная спецификация в конструкции VAR_CONFIG...END_VAR для какой-либо неполной спецификации адреса, выраженной символом «*» в любом экземпляре программы или функционального блока, который содержит такие неполные спецификации.

6.5.6 Сохраняемые переменные (RETAIN, NON_RETAIN)

6.5.6.1 Общие положения

Когда элемент конфигурации (ресурс или конфигурация) «запускается», как «теплый рестарт» или «холодный рестарт» в соответствии с МЭК 61131-1, каждая переменная, связанная с элементом конфигурации и ее программами, имеет значение, зависящее от операции запуска элемента конфигурации и объявления свойств переменной в части сохранения.

Свойства в части сохранения могут объявлять переменные, содержащиеся в секциях переменных VAR_INPUT, VAR_OUTPUT и VAR функциональных блоков и программ, сохраняемыми или несохраняемыми, используя квалификаторы RETAIN or NON_RETAIN, представленные на рисунке 7. Использование этих ключевых слов необязательно.

На рисунке 9 показан алгоритм назначения начальных значений переменным.

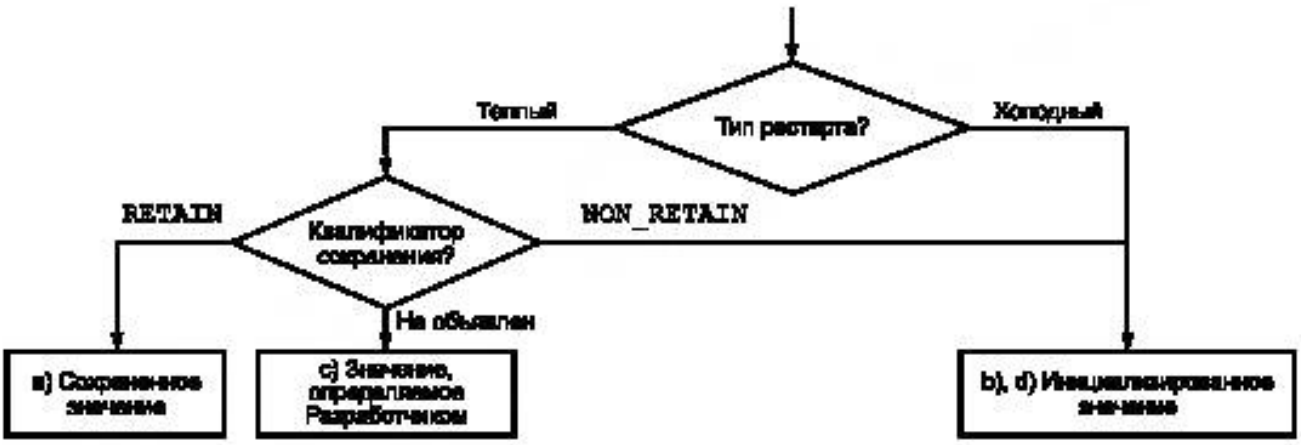


Рисунок 9 — Алгоритм назначения начального значения переменной (правила)

1 Если тип рестарта — «теплый рестарт», как определено в МЭК 61131-1, то начальные значения всех переменных в секции переменных с квалификатором RETAIN будут сохраненными значениями. Данные значения — значения переменных в момент остановки ресурса или конфигурации.

2 Если тип рестарта — «теплый рестарт», то начальные значения всех переменных в секции переменных с квалификатором NON_RETAIN инициализируются.

3 Если тип рестарта — «теплый рестарт», и квалификаторы RETAIN и NON_RETAIN не отсутствуют, то начальные значения определяются разработчиком.

4 Если тип рестарта — «холодный рестарт», начальные значения переменных в секции VAR с квалификаторами RETAIN и NON_RETAIN инициализируются, как описано ниже.

6.5.6.2 Инициализация

Переменные инициализируются, используя определенные пользователем значения, специфические для переменной.

Если никакого значения для инициализации переменной явно не определено, используется определенное пользователем начальное значение, специфическое для переменной. Если ничего не определено, используется специфическое для типа неявное начальное значение, определенное в таблице 10.

Затем применяются следующие правила:

- переменные, которые представляют входы системы программируемого контроллера, как определено в МЭК 61131-1, инициализируются специфическим для разработчика способом;
- квалификаторы RETAIN и NON_RETAIN могут использоваться для переменных, объявленных в статических секциях VAR, VAR_INPUT, VAR_OUTPUT и VAR_GLOBAL, но не в секции VAR_IN_OUT;
- разрешено использование квалификаторов RETAIN и NON_RETAIN в объявлениях экземпляров функционального блока, класса и программы. Поэтому, все переменные образца обрабатываются как RETAIN или NON_RETAIN, за исключением следующего:
  - переменная явно объявлена, как RETAIN или NON_RETAIN в объявлении функционального блока, класса или типа программы;
  - переменная является типом функционального блока или классом. В этом случае применяется декларация сохранения используемого типа функционального блока или класса.

Разрешено использование квалификаторов RETAIN и NON_RETAIN для экземпляров типов структурированных данных. Поэтому, все элементы структуры, а также все элементы вложенных структур обрабатываются как RETAIN или NON_RETAIN.

Пример —

```
VAR RETAIN
    AT %QW5: WORD:= 16#FF00;
    OUTARY AT %QW6: ARRAY[0..9] OF INT:= [10(1)];
    BITS: ARRAY[0..7] OF BOOL:= [1,1,0,0,0,1,0,0];
END_VAR

VAR NON_RETAIN
    BITS: ARRAY[0..7] OF BOOL;
    VALVE_POS AT %QW28: INT:= 100;
END_VAR
```

6.6 Программные компоненты (POU)

6.6.1 Общие свойства программных компонентов

6.6.1.1 Общие положения

Программными компонентами (POU), установленными в настоящем стандарте, являются функция, функциональный блок, класс и программа. Функциональные блоки и классы могут содержать методы.

Для достижения модуляризации и структурирования программные компоненты состоят из четко сформулированных частей программы. Программные компоненты имеют определенный интерфейс с входами и выходами и может вызываться и выполняться много раз.

Примечание — Упомянутый выше параметрический интерфейс не совпадает с интерфейсом, определенным в рамках объектно-ориентированного подхода.

Программные компоненты и методы могут поставляться разработчиком или программироваться пользователем.

Ранее объявленный программный компонент может использоваться в объявлении других программных компонентов, как показано на рисунке 3.

Рекурсивные выходы программных компонентов и методы определяются разработчиком.

Максимальное число программных компонентов, методов и экземпляров для данного ресурса определяется разработчиком.

6.6.1.2 Присваивание и выражение

6.6.1.2.1 Общие положения

Языковые конструкции присваивания и выражения используются в текстовых и (частично) графических языках.

6.6.1.2.2 Присваивание

Присваивание используется для записи значения литерала, константы или выражения (см. ниже) другой переменной. Данная переменная может быть переменной любого вида, например, входной или выходной переменной функции, метода, функционального бока и т. д.

Переменные одного типа всегда могут присваиваться. Дополнительно применяются следующие правила:

- переменная или константа типа STRING или WSTRING может быть присвоена другой переменной типа STRING или WSTRING, соответственно. Если исходная строка длиннее, чем целевая строка, результат определяется реализатором;
- переменная типа-диапазона может использоваться везде, где может использоваться переменная базового типа. Если значение типа-диапазона выходит за пределы указанных значений, возникает ошибка;
- переменная производного типа может использоваться везде, где может использоваться переменная ее базового типа.

Дополнительные правила для массивов могут быть определены разработчиком.

Для адаптации типа данных источника к типу данных адресата может использоваться неявное или явное преобразование типа:

a) в текстовой форме (частично применимой и к графическим языкам) оператор присваивания может быть следующим:

«:= » который означает, что значение выражения в правой стороне оператора записывается в переменную в левой стороне оператора; или

« => » который означает, что значение в левой стороне оператора записывается в переменную в правой стороне оператора.

Оператор «=>» используется только для списка параметров вызова функций, методов, функциональных блоков и т. п. и только для передачи параметра VAR_OUTPUT назад вызывающему объекту.

Пример —

A:= B + C/2;

Func (in1:= A, out2 => x); A_struct1:= B_Struct1;

Примечание — Присваивание определенных пользователем типов данных (STUCTURE, ARRAY) рассматривается в таблице 72;

b) в графической форме:

присваивание изображается как линия соединения от источника к адресату, в основном, слева направо. Например, от выхода функционального блока к входу функционального блока, или от графического «расположения» переменной (константы) к входу функции, или от выхода функции к графическому «расположению» переменной.

Стандартная функция MOVE является одним из графических представлений присваивания.

6.6.1.2.3 Выражение

Выражение — это языковая конструкция, которая состоит из определенной конфигурации операндов (таких как литералы, переменные, вызовы функций) и операторов, (таких как +, -, *, /) и которая производит одно значение, которое может быть многозначным.

Для адаптации типов данных операции в выражении может использоваться неявное или явное преобразование типа:

a) в текстовой форме (а также частично в графических языках), выражение вычисляется в определенном порядке, зависящем от приоритетов, заложенных в языке.

Пример — ... B + C / 2 * SIN(x) ...;

b) в графической форме, выражение показывается как сеть графических блоков (функциональных блоков, функций и т. п.), связанных линиями.

6.6.1.2.4 Константное выражение

Константное выражение — это языковая конструкция, состоящая из определенной комбинации операндов (таких как +, -, *) и производит одно значение, которое может быть многозначным.

6.6.1.3 Частичный доступ к переменным типа ANY_BIT

Для переменных типа данных ANY_BIT (BYTE, WORD, DWORD, LWORD), частичный доступ к биту, байту, слову и двойному слову переменной определен в таблице 17.

Для адресации части переменной используются символ «%» и префикс размера, определенный как и для прямо представленных переменных в таблице 16 (X, B, W, D, L) используются в сочетании с целым литералом (со значением от 0 до max) для адреса внутри переменной. Литерал 0 указывает на самую младшую часть, max — на самую старшую часть. Префикс «%X» может факультативно использоваться при доступе к битам.

Пример — Частичный доступ к переменным ANY_BIT

```
VAR
    Bo: BOOL;
    By: BYTE;
    Wo: WORD;
    Do: DWORD;
    Lo: LWORD;
END_VAR;
    Bo:= By.%X0; // бит 0 переменной By
    Bo:= By.7; // бит 7 переменной By; %X используется по умолчанию и может быть опущен.
    Bo:= Lo.63 // бит 63 переменной Lo;
    By:= Wo.%B1; // байт 1 переменной Wo;
    By:= Do.%B3; // байт 3 переменной Do.
```

Таблица 17 — Частичный доступ к переменным типа ANY_BIT

Номер	Описание	Тип данных	Пример и синтаксис (примечание 2)
	Тип данных — доступ к		myVAR_12.%X1; yourVAR1.%W3;
1a	BYTE — бит VB2.%X0	BOOL	от <имя_переменной>.%X0 до <имя_переменной>.%X7
1b	WORD — бит VW3.%X15	BOOL	от <имя_переменной>.%X0 до <имя_переменной>.%X15
1c	DWORD — бит	BOOL	от <имя переменной>.%X0 до <имя_переменной>.%X31
1d	LWORD — бит	BOOL	от <имя_переменной>.%X0 до <имя_переменной>.%X63
2a	WORD — байт VW4.%B0	BYTE	от <имя_переменной>.%B0 до <имя_переменной>.%B1
2b	DWORD — байт	BYTE	от <имя_переменной>.%B0 до <имя_переменной>.%B3
2c	LWORD — байт	BYTE	от <имя_переменной>.%B0 до <имя_переменной>.%B7
3a	DWORD — слово	WORD	от <имя_переменной>.%W0 до <имя_переменной>.%W1
3b	LWORD — слово	WORD	от <имя_переменной>.%W0 до <имя_переменной>.%W3
4	LWORD — двойное слово VL5.%D1	DWORD	от <имя_переменной>.%D0 до <имя_переменной>.%D1
Префикс доступа к биту %X может быть опущен в соответствии с таблицей 16, например, By1.%X7 эквивалентно By1.7. Частичный доступ не должен использоваться с прямо представленными переменными, например, %IB10.			

6.6.1.4 Представление и правила вызова

6.6.1.4.1 Общие положения

Вызов используется для выполнения функции, экземпляра функционального блока или метода функционального блока или класса. Как показано на рисунке 10, вызов может быть представлен в текстовой или графической форме.

1 Там, где не заданы входные переменные стандартных функций, применяются неявные имена IN1, IN2, ... в порядке сверху вниз. Если стандартная функция имеет один вход без имени, применяется неявное имя IN.

2 Если какая-либо переменная VAR_IN_OUT какого-либо вызова в программном компоненте «неправильно отображается», возникает ошибка.

Переменная VAR_IN_OUT «отображена правильно», если:

- она графически соединена в левой части; или
- она присваивается оператором «:=» в текстовом вызове, переменной, объявленной (без квалификатора CONSTANT) в блоке VAR_IN_OUT, VAR, VAR_TEMP, VAR_OUTPUT или VAR_ EXTERNAL содержащего программного компонента или «правильно отображенной» в блоке VAR_IN_OUT другого содержащегося вызова.

3 «Правильно отображенная» (как показано в правиле выше) переменная VAR_IN_OUT вызова может

- графически соединяться в правой части; или
- присваиваться, используя оператор «:=» в текстовом операторе присваивания переменной, объявленной в блоке VAR, VAR_OUTPUT или VAR_EXTERNAL содержащего программного компонента.

Если такое соединение будет приводить к неопределенному значения переменной, соединенной таким образом, то возникает ошибка.

4 Имя экземпляра функционального блока может использоваться как вход, если оно объявлено как VAR_INPUT, или как VAR_IN_OUT.

Экземпляр может быть использован внутри вызванного объекта следующим образом:

- если он объявлен как VAR_INPUT, переменные функционального блока могут только читаться;
- если он объявлен как VAR_IN_OUT, переменные функционального блока могут читаться и записываться, и функциональный блок может вызываться.

6.6.1.4.2 Текстовые языки

Свойства текстового вызова определяются в таблице 20. Текстовый вызов состоит из имени вызываемого объекта и последующего списка параметров.

В языке ST параметры разделяются запятыми, и этот перечень ограничивается слева и справа скобками.

Перечень параметров вызова предоставляет фактические значения и может присваивать их соответствующим именам формальных параметров (если они имеются):

- Формальный вызов

Перечень параметров имеет форму набора операторов присваивания фактических значений формальным параметрам (перечню формальных параметров), то есть:

a) присваивание значений входным и входным-выходным переменным, используя оператор «:=»; и

b) присваивание значений выходным переменным, используя оператор «=>».

Перечень формальных параметров может быть полным или неполным. Каждая переменная, которой в перечне на назначено значение, имеет начальное значение, присвоенное в объявлении вызванного объекта или неявное значение соответствующего типа данных.

Порядок параметров в перечне не имеет значения. Могут использоваться параметры управления выполнением EN и ENO.

Пример 1

```
A:= LIMIT(EN:= COND, IN:= B, MN:= 0, MX:= 5, ENO => TEMPL); // полный перечень параметров
A:= LIMIT(IN:= B, MX:= 5);          // неполный перечень параметров
```

- Неформальный вызов

Перечень параметров содержит точно такое число параметров, и точно в том порядке и тех же типов данных, как задано в определении функции, исключая параметры управления выполнением EN и ENO.

Пример 2

A:= LIMIT(B, 0, 5);

Данный вызов эквивалентен полному вызову в примере 1, но без параметров EN и ENO.

6.6.1.4.3 Графические языки

В графических языках вызов функций представляется в виде графических блоков в соответствии со следующими правилами:

1 Все блоки — прямоугольные.

2 Размер и пропорции блока могут изменяться в зависимости от числа входов и другой, показываемой информации.

3 Направление обработки блока — слева направо (входные параметры в левой стороне и выходные параметры — в правой).

4 Имя или символ вызываемого объекта, как описано ниже, расположено внутри блока.

5 Предусмотрено место для входных и выходных переменных, появляющихся на левой и правой сторонах блока, соответственно.

6 Могут использоваться дополнительные входная EN и выходная ENO переменные. Если они присутствуют, то показываются в самой верхней позиции слева и справа от блока, соответственно.

7 Результат функции показывается в верхней позиции с правой стороны блока, кроме случая, когда присутствует выходной параметр ENO. В этом случае результат функции показывается в позиции, следующей за выходным параметром ENO. Так как имя вызванного объекта само используется для присваивания своего выходного значения, никаких имен выходных переменных не показывается в правой стороне блока для результата функции.

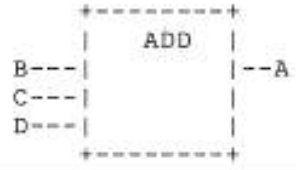
8 Соединения параметров (включая результат функции) показываются линиями передачи сигналов.

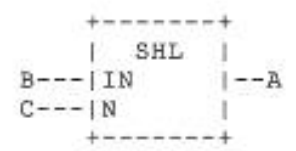
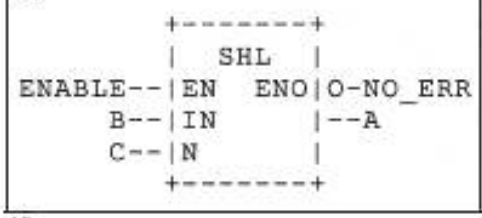
9 Отрицание логического сигнала показывается помещением светлого кружка вблизи от пересечения входной и выходной линии с блоком. В наборе символов это может быть представлено буквой «O» верхнего регистра, как показано в таблице 20. Отрицание выполняется за пределами программного компонента.

10 Все входы и выходы (включая результат функции) графически представленных функций представляются одной линией с соответствующей стороны блока, даже когда элемент данных является многоэлементной переменной.

Результаты и выходы (VAR_OUTPUT) могут соединяться с переменной, используемой как входная переменная к другим вызовам, или могут оставаться без соединения.

Графический пример (язык FBD)	Текстовый пример (язык ST)	Объяснение

a)
 
 	
	A:= ADD(B,C,D); // функция или
	A:= B + C + D; // операторы
		Неформальный перечень параметров
		(B, C, D)

b)  	A:= SHL(IN:= B, N:= C);	Имена формальных параметров IN, N
c)  	A:= SHL(EN:= ENABLE, IN:= B, N := C, NOT ENO => NO_ERR);	Имена формальных параметров Использование входного параметра EN и отрицания выходного параметра ENO

Графический пример (язык FBD)	Текстовый пример (язык ST)	Объяснение
d)  	A:= INC(V:= X);	Определенная пользователем функция INC Имена формальных параметров V для VAR_IN_OUT

Рисунок 10 — Формальное и неформальное представление вызова (примеры), лист 1

В примере показывается графическое и текстовое представление вызова, включая вызов стандартной функции (ADD) без определенных имен формальных параметров; вызов стандартной функции (SHL) с определенными именами формальных параметров; вызов этой же функции с использованием входного параметра EN и выходного параметра ENO с отрицанием; и вызов определенной пользователем функции (INC) с определенными именами формальных параметров.

Рисунок 10

6.6.1.5 Управление выполнением (EN, ENO)

Как показано в таблице 18, дополнительная логическая входная переменная EN (Разрешить) и дополнительная логическая выходная переменная ENO (Разрешить выход) могут предоставляться разработчиком или пользователем в соответствии с объявлением.

VAR_INPUT	EN:	BOOL:= 1;	END_VAR
VAR_OUTPUT	ENO:	BOOL;	END_VAR

Когда используются эти переменные, выполнение операций, определенных программным компонентом, контролируется в соответствии со следующими правилами:

1 Если значение EN равно FALSE, то программный компонент не будет выполняться. Кроме того, значение ENO будет установлено в FALSE. Разработчик подробно определяет поведение в этом случае, см. примеры ниже.

2 В противном случае, если значение EN равно TRUE, значение ENO устанавливается в TRUE, и реализация программного компонента будет выполняться. Программный компонент может устанавливать ENO в логическое значение в соответствии с результатами выполнения.

3 Если во время выполнения одного из программных компонентов возникает ошибка, выходная переменная ENO этого программного компонента устанавливается в FALSE (0) системой программированного контроллера.

4 Если выходная переменная ENO установлена FALSE (0), значения всех других выходных переменных (VAR_OUTPUT, VAR_IN_OUT и результат функции) определяются разработчиком.

5 Входная переменная EN устанавливается в фактическое значение только во время вызова программного компонента.

6 Выходная переменная ENO передается только как во время вызова программного компонента.

7 Выходная переменная ENO устанавливается только внутри программного компонента.

8 Использование параметров EN или ENO в функции REF() для получения указателя на EN или ENO является ошибкой.

В случае, когда EN равно FALSE, можно выполнять другие действия вместо нормального выполнения программного компонента. Данные действия определяются разработчиком. См. примеры ниже.

Пример 1 — Внутренняя реализация

Входная переменная EN оценивается внутри программного компонента.

Если EN равно FALSE, то ENO устанавливается в False, и программный компонент немедленно завершает выполнение или выполняет подмножество операций в зависимости от ситуации.

Все заданные входные и входные-выходные параметры оцениваются и устанавливаются в экземпляре программного компонента (за исключением функций). Проверяется достоверность входных-выходных параметров.

Пример 2 — Внешняя реализация

Входная переменная EN оценивается вне программного компонента. Если EN равно False, то только происходит установка ENO в значение False, и программный компонент не вызывается.

Входные и входные-выходные параметры не оцениваются и не устанавливаются в экземпляре программного компонента. Достоверность входных-выходных параметров не оценивается.

Входной параметр EN не устанавливается вне программного компонента отдельно от вызова.

На следующем рисунке и в примерах иллюстрируется использование программного компонента с параметрами EN и ENO и без них:

Пример 3 — Внутренняя реализация

myInst (EN:= cond, A:= v1, C:= v3, B=> v2, ENO=> X);

где тело экземпляра функционального блока myInst начинает выполнение с параметрами

IF NOT EN THEN... // выполняет подмножество операций

// в зависимости от ситуации

ENO:= 0; RETURN; END_IF;

Пример 4 — Внешняя реализация

IF cond THEN myInst (A:= v1, C:= v3, B=> v2, ENO=> X)

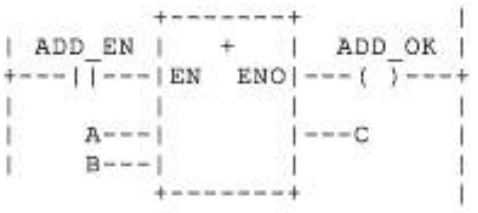
ELSE X:= 0; END_IF;

В таблице 18 приведены свойства при вызове программного компонента с параметрами EN и ENO и без них.

Таблица 18 — Управление выполнением графически с использованием EN и ENO

Номер	Описание [a]	Пример [b]
1	Использование без EN и ENO	Показано для функции в языках FBD и ST  C:= ADD(IN1:= A, IN2:= B);
2	Использование только EN (без ENO)	Показано для функции в языках FBD и ST  C:= ADD(EN:= ADD_EN. IN1:= A, IN2:= B);

Окончание таблицы 18

Номер	Описание [a]	Пример [b]
3	Использование только ENO (без EN)	Показано для функции в языках FBD и ST  C:= ADD(IN1:= A, IN2:= B, ENO => ADD_OK);
4	Использование EN и ENO	Показано для функции в языках FBD и ST  C:= ADD(EN:= ADD_EN, IN1:= a, IN2:= IN2, EN => ADD_OK);
[a] Разработчик указывает в каком из языков поддерживается свойство, то есть в реализации может быть запрещено использование EN и/или ENO. [b] Языки, выбранные для демонстрации свойств выше, даны только для примера.		

6.6.1.6 Преобразование типов данных

Преобразование типов данных используется для настройки типов данных к использованию в выражениях, присваиваниях и назначении параметров.

Представление и интерпретация информации, хранящейся в переменной зависит от объявленного типа данных переменной. Имеется два случая, где используется преобразование типов данных.

- В присваивании значения переменной другой переменной с другим типом данных.

Это применимо к операторам присваивания «:=» и «=>» и присваивании переменным, объявленным как параметры, то входным и выходным переменным функций, функциональных блоков, методов и программ. На рисунке 11 показаны правила преобразования исходного типа данных в целевой тип данных;

Пример 1

```
A:= B;                    // Присваивание переменной
FB1 (x:= z, v => W);      // Присваивание параметрам
```

- В выражении (см. 7.3.2 для языка ST), состоящем из операторов, таких как «+», и операндов, таких как литералы и переменные такого же типа данных или других типов данных.

Пример 2

```
... SQRT( B + (C * 1.5)); // Выражение
```

- Явное преобразование типа данных выполняется использованием функции преобразования.
- Неявное преобразование типа данных имеет следующие правила применения:
  1) должно сохранять значение и точность типов данных;
  2) может применяться для типизированных функций;
  3) может применяться к присваиваниям выражений переменным;

Пример 3

```
myUDInt:= myUInt1 * myUInt2;
/* Умножение имеет результат типа UINT
который затем неявно преобразуется в тип UDINT при присваивании */
```

4) может применяться к присваиванию входного параметра;

5) может применяться к присваиванию входного параметра;

6) не применяется к присваиванию входного-выходного параметра;

7) может применяться так, что операнды и результаты операции или перегруженной функции получает одинаковый тип данных;

Пример 4

```
myUDInt:= myUInt1 * myUDInt2;
// myUInt1 неявно конвертируется в тип данных UDINT, умножение имеет результат типа данных UDINT
```

8) правила для нетипизированных литералов определяются разработчиком.

Примечание — Для предотвращения неопределенностей, пользователь может использовать типизированные литералы.

Пример 5

```
IF myWord = NOT (0) THEN ...;          // Неопределенное сравнение с 16#FFF, 16#0001, 16#00FF и т. д.
IF myWord = NOT (WORD#0) THEN ...; // Неопределенное сравнение с 16#FFFF
```

На рисунке 11 показаны два альтернативных «явных» и «неявных» преобразования исходного типа данных к целевому типу данных.

Исходный тип данных		Целевой тип данных																									
		действительный		целый				без знака				битовый					дата и время								символьный			
		LREAL	REAL	LINT	DINT	INT	SINT	ULINT	UDINT	UINT	USINT	LWORD	DWORD	WORD	BYTE	BOOL	LTIME	TIME	LDT	DT	LDATE	DATE	LTOD	TOD	WSTRING	STRING	WCHAR	CHAR
действительный	LREAL		e	e	e	e	e	e	e	e	e	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	REAL	i		e	e	e	e	e	e	e	e	-	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
целый	LINT	e	e		e	e	e	e	e	e	e	e	e	e	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	DINT	i	e	i		e	e	e	e	e	e	e	e	e	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	INT	i	i	i	i		e	e	e	e	e	e	e	e	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	SINT	i	i	i	i	i		e	e	e	e	e	e	e	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-
без знака	ULINT	e	e	e	e	e	e		e	e	e	e	e	e	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	UDINT	i	e	i	e	e	e	i		e	e	e	e	e	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	UINT	i	i	i	i	e	e	i	i		e	e	e	e	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	USINT	i	i	i	i	i	e	i	i	i		e	e	e	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-
битовый	LWORD	e	-	e	e	e	e	e	e	e	e		e	e	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	DWORD	-	e	e	e	e	e	e	e	e	e	i		e	e	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	WORD	-	-	e	e	e	e	e	e	e	e	i	i		e	-	-	-	-	-	-	-	-	-	-	-	-	e	-
	BYTE	-	-	e	e	e	e	e	e	e	e	i	i	i		-	-	-	-	-	-	-	-	-	-	-	-	-	e
	BOOL	-	-	e	e	e	e	e	e	e	e	i	i	i	i		-	-	-	-	-	-	-	-	-	-	-	-	-
дата и время	LTIME	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		e	-	-	-	-	-	-	-	-	-	-	-
	TIME	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	i		-	-	-	-	-	-	-	-	-	-	-
	LDT	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		e	e	e	e	e	-	-	-	-	-
	DT	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	i		e	e	e	e	-	-	-	-	-
	LDATE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		e	-	-	-	-	-	-	-
	DATE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	i				-	-	-	-	-	-
	LTOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			e	-	-	-	-
	TOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	i			-	-	-	-
символьный	WSTRING	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	X	e	-	-
	STRING (Примечание)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		-	-	e		-	e
	WCHAR	-	-	-	-	-	-	-	-	-	-	e	e	e	-	-	-	-	-	-	-	-		-	-	i	-		e
	CHAR (Примечание)	-	-	-	-	-	-	-	-	-	e	e	e	e	e	-	-	-	-	-	-	-		-	-	-	i	e	

Рисунок 11 — Правила преобразования типов данных — явные и неявные (сводка)

Обозначения:

— преобразование типа данных не требуется;

- — данным стандартом не определены явные или неявные преобразования типов данных. Реализация может поддерживать дополнительные преобразования типов данных, специфичные для разработчика;

i — неявное преобразование типов данных; однако дополнительно разрешено явное преобразование типов;

e — явное преобразование типов данных, применяемое пользователем (стандартные функции преобразования), могут использоваться для предотвращения потери данных, несоответствия диапазонов или воздействия возможных функциональных возможностей, реализованных разработчиком.

Примечание — Преобразование STRING в WSTRING и CHAR в WCHAR не являются неявными, во избежание конфликтов с используемыми наборами символов.

Рисунок 11, лист 2

На рисунке 12 показаны преобразования типов данных, поддерживаемые неявным преобразованием типов данных. Стрелки представляют возможные пути преобразования. Например, BOOL может быть преобразована в BYTE, BYTE может быть преобразована в WORD и т. д.

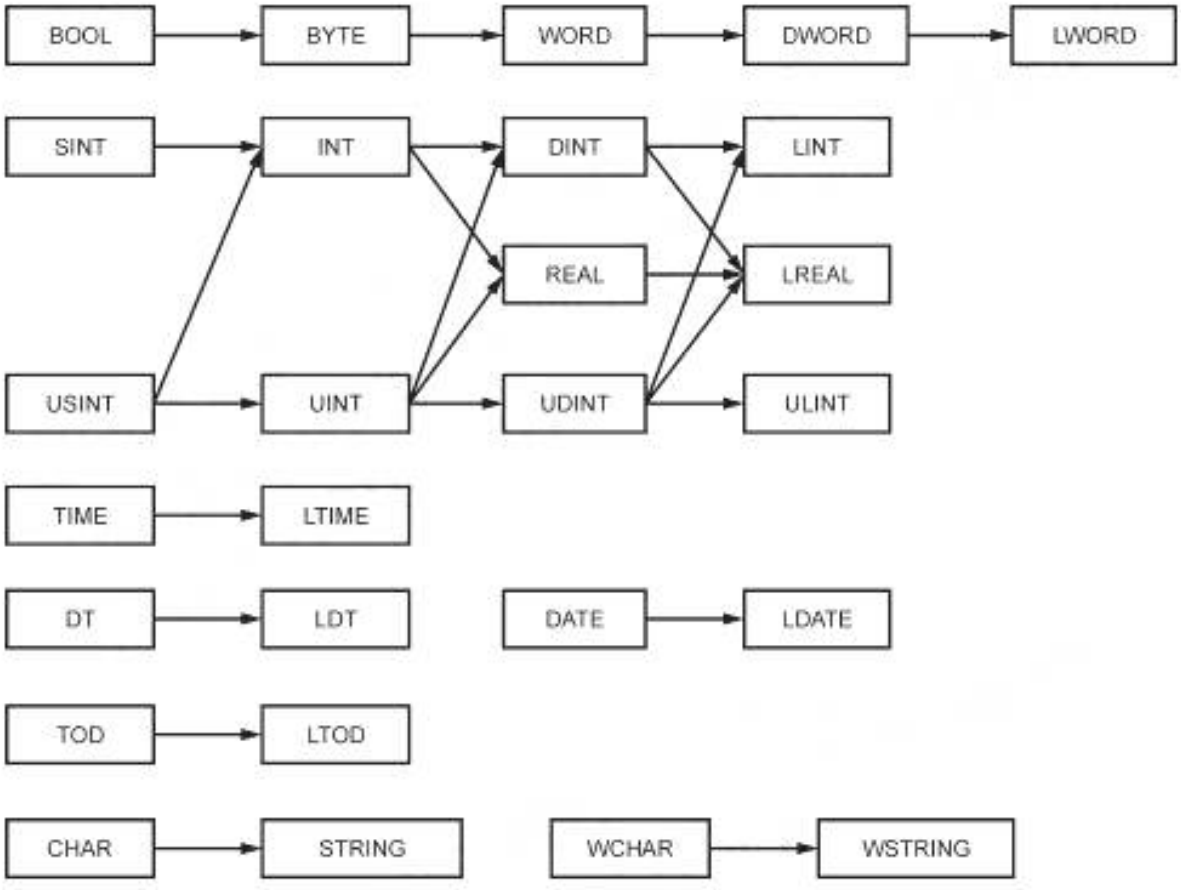


Рисунок 12 — Поддерживаемые неявные преобразования типов

В следующих примерах показываются примеры преобразования типов данных.

Пример 6 — Сравнение явных и неявных преобразований типов

1) Объявление типа

```
VAR
    PartsRatePerHr:     REAL;
    PartsDone:          INT;
    HoursElapsed:       REAL;
    PartsPerShift:      INT;
    ShiftLength: SINT;
END_VAR
```

2) Использование в языке ST

a) Явное преобразование типа данных

```
PartsRatePerHr:= INT_TO_REAL(PartsDone) / HoursElapsed;
PartsPerShift := REAL_TO_INT(SINT_TO_REAL(ShiftLength)*PartsRatePerHr);
```

b) Явное преобразование перегруженного типа

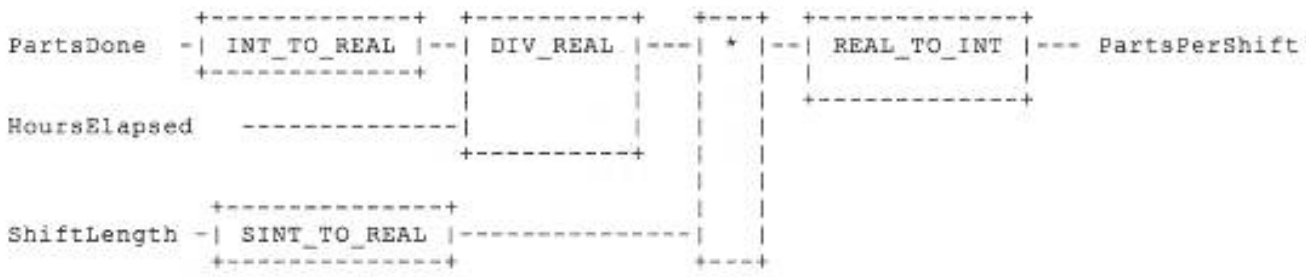
```
PartsRatePerHr:= TO_REAL(PartsDone) / HoursElapsed; PartsPerShift := TO_INT(TO_REAL(ShiftLength)*
PartsRatePerHr);
```

c) Неявное преобразование типа данных

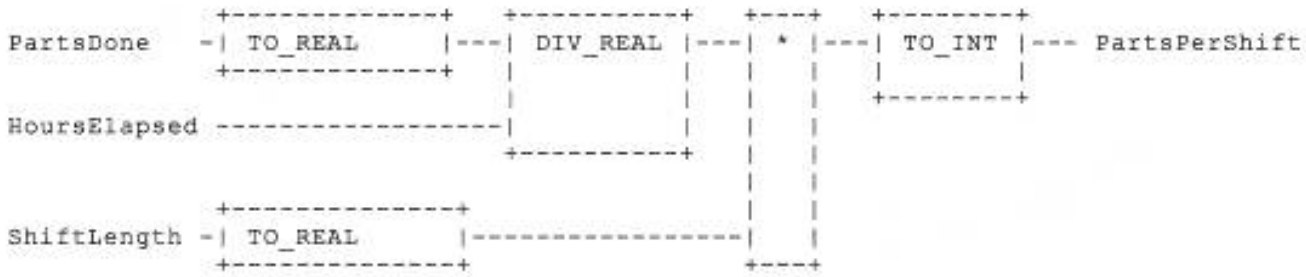
```
PartsRatePerHr:= PartsDone / HoursElapsed;
PartsPerShift := TO_INT(ShiftLength * PartsRatePerHr);
```

3) Использование в языке FBD

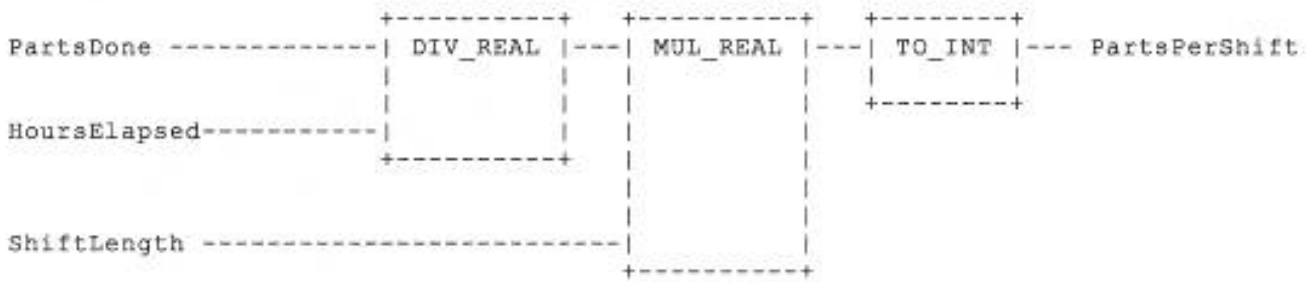
a) Явное преобразование типа данных



b) Явное преобразование перегруженного типа



c) Неявное преобразование типов типизированными функциями



6.6.1.7 Перегрузка

6.6.1.7.1 Общие положения

Говорят, что элемент языка перегруженный, когда он может оперировать с элементами входных данных различных типов в пределах родового типа данных, например ANY_NUM, ANY_INT.

Следующие стандартные элементы языка, предоставляемые изготовителем, могут иметь родовую перегрузку как специальное свойство:

- стандартные функции

Это — перегруженные стандартные функции (например, ADD, MUL) и перегруженные стандартные функции преобразования (например, TO_REAL, TO_INT);

- стандартные методы

Настоящий стандарт не определяет стандартные методы в пределах стандартных классов и типов функциональных блоков. Однако они могут быть предоставлены разработчиком;

- функциональные блоки

Настоящий стандарт не определяет стандартные функциональные блоки, за исключением некоторых простых блоков, таких как счетчики.

Однако они могут быть определены другими частями МЭК 61131, и могут предоставляться разработчиком;

- стандартные классы

Настоящий стандарт не определяет стандартных классов. Однако они могут быть определены в других частях МЭК 61131, и могут предоставляться разработчиком;

- операции

Это, например, «+» и «*» в языке ST; ADD, MUL в языке IL.

6.6.1.7.2 Преобразование типов данных

Когда система программированного контроллера поддерживает перегруженные элементы языка, данный элемент языка применяется ко всем подходящим типам данных этого родового типа, которые поддерживаются системой.

Подходящие типы данных для каждого элемента языка определены в соответствующих таблицах свойств. Следующие примеры иллюстрируют детали:

Пример 1

Настоящий стандарт определяет для функции ADD родовой тип данных ANY_NUM для многих входных переменных одного вида и одного выходного результата.

Разработчик определяет для этого родовой тип данных ANY_NUM для связанных элементарных типов данных REAL и INT системы PLC.

Пример 2

Настоящий стандарт определяет функцию битового сдвига LEFT для родового типа данных ANY_BIT для одной входной переменной и выходного результата и родового типа данных ANY_INT для другой входной переменной.

Разработчик определяет следующие два родовых типа данных для системы PLC:

ANY_BIT представляет, например, элементарные типы данных BYTE и WORD;

ANY_INT представляет, например, элементарные типы данных INT и LINT.

Перегруженный элемент языка оперирует с определенными элементарными типами данных в соответствии со следующими правилами:

- типы данных входных переменных и результата имеют одинаковый тип, это применимо к входным переменным и результату одинакового вида.

«Одинаковый вид» означает, что параметры, операнды и результат одинаково используются при сложении и умножении.

Более сложные комбинации определяются разработчиком;

- если типы данных входных и выходных данных одинакового вида имеют разный тип, то преобразование типов в элементе языка определяется разработчиком;

- неявное преобразование типов выражения и присваивания следует за последовательностью вычисления выражения. См. примеры ниже;

- тип данных переменной для хранения результата перегруженной функции не влияет на тип данных результата функции или результата.

Примечание — Пользователь может явно задать тип результата операции, используя типизированные функции.

Пример 3

int3 := int1 + int2 (* Сложение выполняется как целочисленная операция *)

dint1:= int1 + int2; (* Сложение выполняется как целочисленная операция, когда результат преобразуется в тип DINT и присваивается переменной dint1 *)

dint1:= dint2 + int3; (* int3 преобразуется в тип DINT, сложение выполняется как сложение DINT *)

6.6.2 Функции

6.6.2.1 Общие положения

Функция — это программный компонент, который не сохраняет свое состояние, то есть входные параметры, внутренние переменные, выходные параметры и результат.

Если не оговорено иное, к функциям применяются общие свойства программных компонентов.

Выполнение функции:

- обычно предоставляет временный результат, который может быть одним элементом, многоэлементным массивом или структурой;
- возможно предоставляет выходные переменные, которые могут быть многоэлементными;
- может изменять значение входных-выходных переменных и переменных VAR_EXTERNAL.

Функция с результатом может вызываться в выражении или как оператор. Функция без результата не должна вызываться внутри выражения.

6.6.2.2 Объявление функции

Объявление функции состоит из следующих элементов, как определено в таблице 19. Данные свойства объявляются так же, как описано для функциональных блоков.

При объявления функции применяются следующие правила, заданные в таблице 19:

1 Объявление начинается с ключевого слова FUNCTION, за которым следует идентификатор, указывающий имя функции.

2 Если функция предоставляет результат, то далее следует символ «:» и тип данных значения, возвращаемого функцией. Если функция не предоставляет результата, двоеточие и тип данных опускаются.

3 Конструкции с VAR_INPUT, VAR_OUTPUT и VAR_IN_OUT, если требуются, указывающие имена и типы данных параметров функции.

4 Значения переменных, которые передаются функции через конструкцию VAR_EXTERNAL, могут изменяться из функции.

5 Значения констант, которые передаются функции через конструкцию VAR_EXTERNAL CONSTANT, не могут изменяться из функции.

6 Значения переменных, которые передаются функции через конструкцию VAR_EXTERNAL, могут изменяться из функции.

7 Массивы переменной длины могут использоваться как VAR_INPUT, VAR_OUTPUT и VAR_IN_OUT.

8 Входные-выходные и временные переменные могут инициализироваться.

9 Могут использоваться входная переменная EN и выходная переменная ENO как описано.

10 Если требуется, конструкция VAR...END_VAR, а также последовательность VAR_TEMP...END_VAR используются для определения имен и типов внутренних временных переменных.

В отличие от функциональных блоков, переменные, объявленные в секции VAR, не сохраняются.

11 Если в определении переменных стандартной функции используются родовые типы данных (например, ANY_INT), то правила использования фактических типов параметров таких функций являются частью определения функции.

12 Конструкции инициализации переменных могут использоваться для объявления начальных значений входных параметров функции, внутренних и выходных переменных.

13 Ключевое слово END_FUNCTION завершает объявление.

Таблица 19 — Объявление функции

Номер	Описание	Пример
1a	Без результата FUNCTION ... END_FUNCTION	FUNCTION myFC ... END_FUNCTION
1b	С результатом FUNCTION <name>: <data type> END _FUNCTION	FUNCTION myFC: INT ... END_FUNCTION
2a	Входные параметры VAR_INPUT...END_VAR	VAR_INPUT IN:
2b	Выходные параметры VAR_OUTPUT...END_VAR	VAR_OUTPUT OUT: BOOL; ET_OFF: TIME; END_VAR
2c	Входные-выходные параметры VAR_IN_OUT...END_VAR	VAR_IN_OUT A: INT; END_VAR
2d	Временные переменные VAR_TEMP...END_VAR	VAR_TEMP I: INT; END_VAR
2e	Временные переменные VAR...END_VAR	VAR B: REAL; END_VAR Различие с функциональными блоками из-за проблем совместимости в функциональных блоках VAR являются статическими (сохраняются)!
2f	Внешние переменные VAR_EXTERNAL...END_VAR	VAR_EXTERNAL B: REAL; END_VAR Соответствует следующему: VAR_GLOBAL B: REAL...
2g	Внешние константы VAR_EXTERNAL CONSTANT...END_VAR	VAR_EXTERNAL CONSTANT B: REAL; END_VAR Соответствует следующему: VAR_GLOBAL B: REAL
3a	Инициализация входных параметров	VAR_INPUT MN: INT:= 0;
3b	Инициализация выходных параметров	VAR_OUTPUT RES: INT:= 1;
3c	Инициализация временных переменных	VAR I: INT:= 1;
--	Входной параметр EN и выходной параметр ENO	Определено в таблице 18

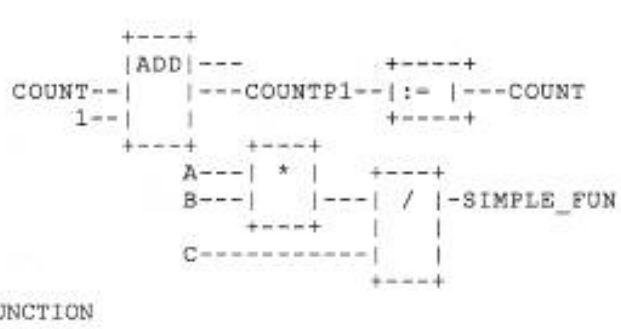
Пример —

```
// Спецификация интерфейсов параметра
FUNCTION SIMPLE_FUN: REAL
  VAR_INPUT
    A, B: REAL;
C: REAL:= 1.0; END_VAR
VAR_IN_OUT COUNT: INT;
  END_VAR
```

```
// Спецификация интерфейсов параметра
                FUNCTION
             +-------------+
             |  SIMPLE_FUN |
REAL-------|A            |----REAL
REAL-------|B            |
REAL-------|C            |
INT--------|COUNT---COUNT|----INT
             +-------------+
```

```
// Спецификация тела функции
VAR COUNTP1: INT; END_VAR COUNTP1:=
ADD(COUNT, 1);
COUNT := COUNTP1
SIMPLE_FUN:= A*B/C; // результат
END_FUNCTION
```

// Спецификация тела функции

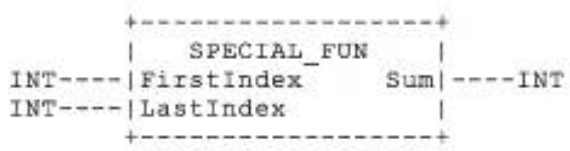


a) Объявление и тело функции (языки ST и FBD) — (см. Примечание)

```
VAR_GLOBAL DataArray: ARRAY [0..100] OF INT;
END_VAR
FUNCTION SPECIAL_FUN
VAR_INPUT
  FirstIndex: INT;
  LastIndex: INT;
END_VAR
VAR_OUTPUT Sum:
  INT;
END_VAR
VAR_EXTERNAL DataArray:
  ARRAY [0..100] OF INT;
END_VAR
VAR I: INT; Sum: INT:= 0; END_VAR
  FOR i:= FirstIndex TO LastIndex DO Sum:=
    Sum + DataArray[i];
END_FOR
END_FUNCTION
```

// Внешний интерфейс

// функция без результата, но есть выходная переменная Sum



// Тело функции — графически не показано

b) Объявление и тело функции (функция без результата — с выходом Var)

***Примечание** — В примере а) входной переменной дано определенное неявное значение 1.0, чтобы предотвратить ошибку «деление на ноль», если вход не указан при вызове функции, например, если графический вход в функцию слева не соединен.*

6.6.2.3 Вызов функции

Вызов функции может быть представлен в текстовой или графической форме.

Так как входные переменные, выходные переменные и результат функции не сохраняется, присваивание входным параметрам, доступ к выходным переменным и результату происходит мгновенно при вызове функции.

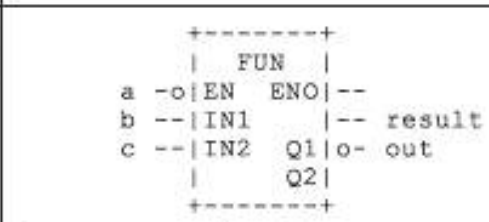
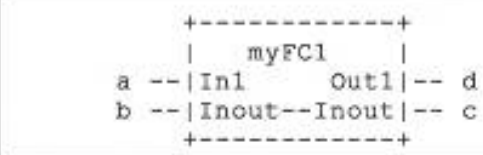
Если массив переменной длины используется как параметр, параметр должен быть соединен к статической переменной.

Функция не содержит информацию о внутреннем состоянии, то есть она не сохраняет никакие входные, внутренние (временные) и выходные элементы от одного вызова до другого:

- вызов функции с одинаковыми параметрами (VAR_INPUT и VAR_IN_OUT) и одинаковыми значениями переменных VAR_EXTERNAL всегда будет изготавливать одинаковые значения выходных переменных, входных-выходных переменных, внешних переменных и результат функции, если он имеется.

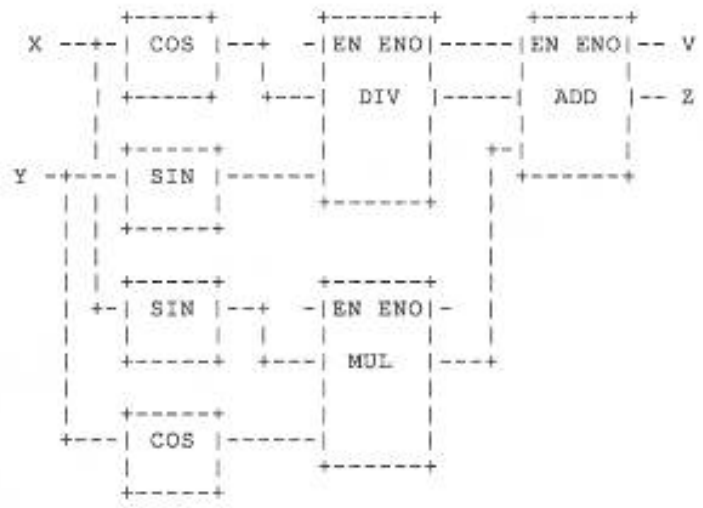
Примечание — Некоторые функции, обычно предоставляемые как системные функции от разработчкиа могут производить различные значения, например, функции TIME(), RANDOM().

Таблица 20 — Вызов функции

Номер	Описание	Пример
1a	Полный формальный вызов (только текстовый) Примечание 1 — Такой вызов используется, если указание параметров EN и ENO в вызове является обязательным.	A:= LIMIT(EN: = COND, IN:= B, MN:= 0, MX:= 5, ENO => TEMPL);
1b	Неполный формальный вызов (только текстовый) Примечание 2 — Используется, если использование параметров EN и ENO в вызове не является обязательным.	A:= LIMIT(IN:= B, MX:= 5); Примечание 3 — Переменная MN будет иметь неявное значение 0 (ноль).
2	Неформальный вызов (только текстовый) (с фиксированным порядком параметров и полный) Примечание 4 — Используется для вызова стандартных функций без формальных имен.	A:= LIMIT(B, 0, 5); Примечание 4 — Данный вызов эквивалентен вызову в примере 1a, но без параметров EN и ENO.
3	Функция без результата функции	FUNCTION myFun // нет объявления типа VAR_INPUT x: INT; END_VAR; VAR_OUTPUT y: REAL; END_VAR; myFun(150, var); // Вызов
4	Графическое представление	
5	Использование логического входа с отрицанием и логического выхода с отрицанием в графическом представлении	 Примечание 6 — Использование таких конструкций запрещено для входных-выходных переменных.
6	Графическое использование VAR_IN_OUT	

Пример — Вызов функции

```
Вызов
VAR
    X, Y, Z, Res1, Res2: REAL;
    En1, V: BOOL;
    END_VAR
Res1:= DIV(In1:= COS(X), In2:= SIN(Y), ENO => EN1);
Res2:= MUL(SIN(X), COS(Y));
Z := ADD(EN:= EN1, IN1:= Res1, IN2:= Res2, ENO => V);
```



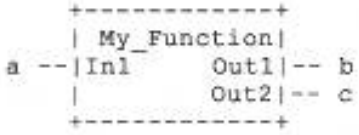
a) Вызов стандартных функций с результатом и параметрами EN и ENO

Объявление

```
FUNCTION My_function                                    //нет типа, нет результата
    VAR_INPUT In1:              REAL;  END_VAR
    VAR_OUTPUT Out1,    Out2:   REAL;  END_VAR
    VAR_TEMP Tmp1:              REAL;  END_VAR  //разрешено использование VAR_TEMP
    VAR_EXTERNAL Ext:           BOOL;  END_VAR
      // Тело функции
END_FUNCTION
```

Текстовый и графический вызов

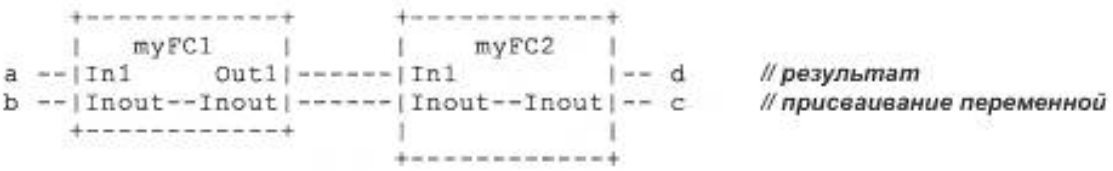
```
My_Function (In1:= a, Out1 => b; Out2 => c);
                                        // без результата
                                        // с двумя выходными переменными
```



b) Объявление и вызов функции без результата, но с двумя выходными переменными

Текстовый и графический вызов

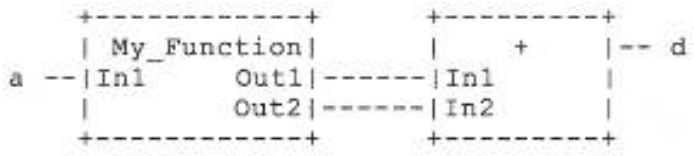
```
myFC1 (In1:= a, Inout:= b, Out1 => Tmp1);   // использование временной переменной
d:= myFC2 (In1:= Tmp1, Inout:= b);          // переменная b сохраняется в входной-выходной переменной inout; Присваивание переменной
c:= b; // значение переменной b присвоено переменной c
```



c) Вызов функции с графическим представлением входных-выходных переменных

Текстовый и графический вызов

```
~~My_Function (In1:= a, Out1+Out2 => d);~~ // не разрешен в языке ST
My_Function (In1:= a, Out1 => Tmp1, Out2 => Tmp2);
d:= Tmp1 + Tmp2;
```



d) Вызов функции без результата, но с выражением из выходных переменных

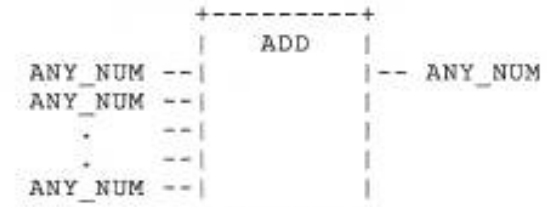
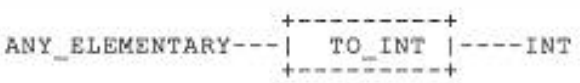
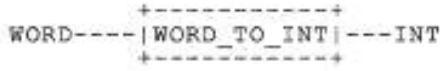
Примечание 2 — Данные примеры представляют различные представления одной и той же фукциональности. Не требуется поддерживать какое-либо автоматическое преобразование между двумя формами представления.

6.6.2.4 Типизированные и перегруженные функции

Функция, которая нормально представляет перегруженный оператор, должна быть типизированной. Это можно сделать добавлением символа подчеркивания «_» с последующим требуемым типом, как показано в таблице 21. Типизированная функции выполняется, используя тип данных для своих входных и выходных переменных. Может применяться неявное или явное преобразование типов.

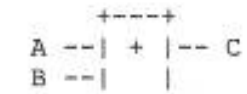
Перегруженная функция преобразования TO_xxx или TRUNC_xxx с xxx , указывающим на типизированный элементарный выходной тип, может быть уточнена предшествующим требуемым элементарным типом входных данных и следующим символом подчеркивания.

Таблица 21 — Типизированные и перегруженные функции

Номер	Описание	Пример
1a	Перегруженная функция ADD (ANY_Num to ANY_Num)	
1b	Преобразование входных переменных ANY_ELEMENT TO_INT	
2a[a)]	Типизированные функции: ADD_INT	
2b[a)]	Преобразование типов: WORD_TO_INT	
Примечание — Перегрузка нестандартных функций или типов функциональных блоков не входит в задачу настоящего стандарта.		
[a)] Если поддерживается свойство 2, разработчик предоставляет дополнительную таблицу, показывающую, какие функции являются перегруженными и какие являются типизированными в реализации.		

Пример 1 — Типизированные и перегруженные функции

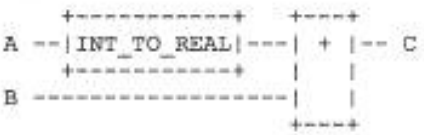
```
VAR
   A: INT;
   B: INT;
   C: INT;
END_VAR
```




```
C:= A+B;
```

Примечание 1 — Преобразование типов в показанном выше примере не требуется.

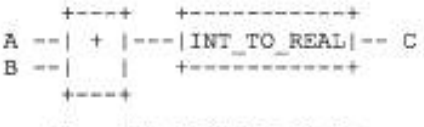
```
VAR
   A: INT;
   B: REAL;
   C: REAL;
END_VAR
```



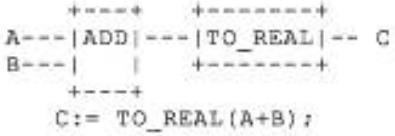

```
C:= INT_TO_REAL(A)+B;
```


```
C:= TO_REAL(A) + B;
```

```
VAR
   A: INT;
   B: INT;
   C: REAL;
END_VAR
```




```
C:= INT_TO_REAL(A+B);
```




```
C:= TO_REAL(A+B);
```

a) Объявление типа (язык ST) *b) Использование (языки FBD и ST)*

Пример 2 — Явное и неявное преобразование типов типизированными функциями

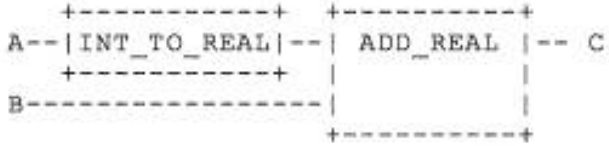
```
VAR
  A: INT;
  B: INT;
  C: INT;
END_VAR
```


```
C:= ADD_INT(A, B);
```

Примечание 2 — Преобразование типов в показанном выше примере не требуется.

Явное преобразование типа данных

```
 VAR
   A: INT;
   B: REAL;
   C: REAL;
 END_VAR
```




```
C:= ADD_REAL(INT_TO_REAL(A), B);
```

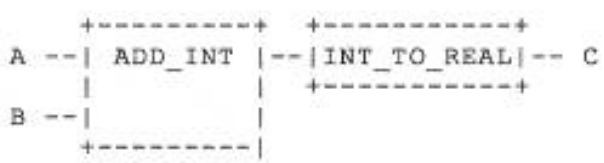
Неявное преобразование типа данных

```
  VAR
    A: INT;
    B: REAL;
    C: REAL;
  END_VAR
```


```
C:= ADD_REAL (A,B);
```

Явное преобразование типа данных

```
VAR
  A: INT;
  B: INT;
  C: REAL;
END_VAR
```




```
C:= INT_TO_REAL(ADD_INT(A, B));
```

Неявное преобразование типа данных

```
VAR
  A: INT;
  B: INT;
  C: REAL;
END_VAR
```


```
C:= ADD_INT(A, B);
```

a) Объявление типов (язык ST) ***b) Использование (языки FBD и ST)***

6.6.2.5 Стандартные функции

6.6.2.5.1 Общие положения

Стандартная функция, определенная в этом подпункте расширяемой, может иметь две или более входных переменных, к которым может быть применена указанная операция. Например, расширяемое сложение дает в качестве выхода сумму всех ее входов. Максимальное число входных переменных расширяемой функции определяется разработчиком. Фактическое число входных переменных в формальном вызове расширяемой функции определяется именем формальной входной переменной с самым большим индексом в последовательности имен переменной.

Пример 1 —

Оператор X:= ADD (Y1, Y2, Y3);

эквивалентен оператору X:= ADD (IN1:= Y1, IN2:= Y2, IN3:= Y3);

Пример 2 —

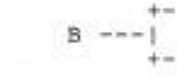
Оператор I:= MUX_INT (K:=3, IN0:= 1, IN2:= 2, IN4:= 3);

эквивалентен оператору I:= 0;

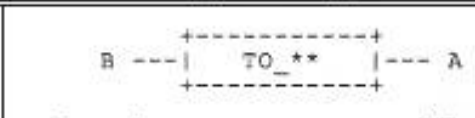
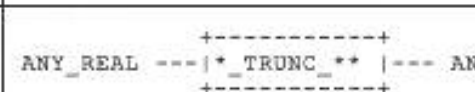
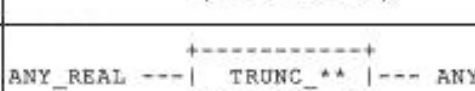
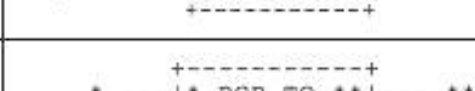
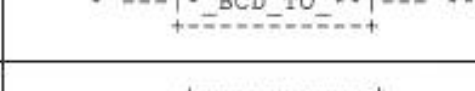
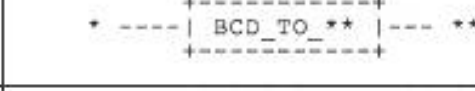
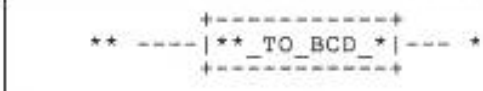
6.6.2.5.2 Функции преобразования типов данных

Как показано в таблице 22, функции преобразования типов *_TO_**, где «*» — тип входной переменной IN, а «**» — тип выходной переменной OUT, например, INT_TO_REAL. Влияние преобразований типов на точность и типы ошибок, которые могут возникать во время выполнения операций преобразования типов, определяется разработчиком.

Таблица 22 — Функция преобразования типов данных

Номер	Описание	Графическая форма	Пример использования
1a	Типизированное преобразование вход_TO_выход	 (*) — Входной тип данных, например, INT (**) — Выходной тип данных, например, REAL	A:= INT_TO_REAL(B);

Продолжение таблицы 22

Номер	Описание	Графическая форма	Пример использования
1b[a),b),e)]	Перегруженное преобразование TO_выход	 — Входной тип данных, например, INT (**) — Выходной тип данных, например, REAL	A:= TO_REAL(B);
2a[c)]	«Старое» перегруженное усечение данных TRUNC		Не рекомендуется
2b[c)]	Типизированное усечение данных вход_TRUNC_выход		A:= REAL_TRUNC_INT(B);
2c[c)]	Перегруженное усечение данных TRUNC_выход		A:= TRUNC_INT(B);
3a[d)]	Типизированная функция вход_BCD_TO_выход		A:= WORD_BCD_TO_INT(B);
3b[d)]	Перегруженная функция BCD_TO_выход		A:= BCD_TO_INT(B);
4a[d)]	Типизированная функция вход_TO_BCD_выход		A:= INT_TO_BCD_WORD(B);
4b[d)]	Перегруженная функция TO_BCD_выход		A:= TO_BCD_WORD(B);

Примечание — Примеры использования даны на языке ST.

[a)] Декларация соответствия на свойство 1 этой таблицы должна включать перечень поддерживаемых специфических преобразований типов и описание эффектов выполнения каждого преобразования.

[b)] Преобразования типа REAL или LREAL в тип SINT, INT, DINT или LINT производить, округлять в соответствии с МЭК 60559, согласно которому, если два ближайших целых значения одинаково близки, результатом является ближайшее целое число, например:

REAL_TO_INT (1.6) эквивалентно 2;

REAL_TO_INT (-1.6) эквивалентно -2;

REAL_TO_INT (1.5) эквивалентно 2;

REAL_TO_INT (-1.5) эквивалентно -2;

REAL_TO_INT (1.4) эквивалентно 1;

REAL_TO_INT (-1.4) эквивалентно -1;

REAL_TO_INT (2.5) эквивалентно 2;

REAL_TO_INT (-2.5) эквивалентно -2.

[c)] Функция TRUNC_* используется для усечения по направлению к нулю типов REAL или LREAL, выдавая один из целых типов, например:

TRUNC_INT (1.6) эквивалентно INT#1;

TRUNC_INT (-1.6) эквивалентно INT#-1;

TRUNC_SINT (1.4) эквивалентно SINT#1;

TRUNC_SINT (-1.4) эквивалентно SINT#-1.

Окончание таблицы 22

[d] Функции преобразования *_BCD_TO_** и **_TO_BCD_* выполняют преобразования между переменными типа BYTE, WORD, DWORD и LWORD и переменными типа USINT, UINT, UDINT и ULINT (представленными «*» и «**», соответственно), когда соответствующие переменные типа битовой строки закодированы в формате BCD. Например, значением USINT_TO_BCD_BYTE(25) будет 2#0010_0101, а значением WORD_BCD_TO_UINT (2#0011_0110_1001) будет 396.

[e] Когда входом или выходом функции преобразования типов является тип STRING или WSTRING, данные символьной строки соответствуют внешнему представлению соответствующих данных, как указано в 6.3.3, в наборе символов, определенном в 6.1.1.

6.6.2.5.3 Преобразование числовых типов данных

В преобразовании числовых типов данных используются следующие правила:

1 Тип данных источника расширяется до самого большого типа данных этой категории типов данных.

2 Затем результат преобразуется в самый большой тип данных категории типов данных, к которой принадлежит целевой тип данных.

3 Затем этот результат преобразуется в целевой тип данных.

Если значение исходной переменной не вмещается в целевой тип данных, то есть диапазон значений слишком мал, то значение целевой переменной определяется разработчиком.

Примечание — Реализация функции преобразования может использовать более эффективную процедуру.

Пример — X:= REAL_TO_INT (70_000.4)

1 Значение (70_000.4) типа REAL преобразуется в значение (70_000.400_000..) типа LREAL.

2 Значение (70_000.4000_000..) типа LREAL преобразуется в значение (70_000) типа LINT. Здесь значение округлено до целого.

3 Значение (70_000) типа LINT преобразуется в значение типа INT. Здесь окончательное значение определяется разработчиком, поскольку максимальное значение, которое может хранить тип INT равно 65536.

Затем результат записывается в переменную целевого типа данных. Теперь данная переменная хранит то же значение, что и исходная переменная, если целевой тип данных в состоянии хранить это значение.

При преобразовании чисел с плавающей точкой применяются нормальные правила округления, то есть округление до ближайшего целого и, если результат неоднозначен, до ближайшего четного целого.

Тип данных BOOL, используемый в качестве исходного типа данных, рассматривается как тип данных целого без знака, который может хранить значения 0 и 1.

В таблице 23 описаны функции преобразования с деталями, вытекающими из применения описанных выше правил.

Таблица 23 — Преобразование числовых типов данных

Номер	Функция преобразования типов данных	Детали преобразования
1	LREAL _TO_ REAL	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
2	LREAL _TO_ LINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
3	LREAL _TO_ DINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
4	LREAL _TO_ INT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком

Продолжение таблицы 23

Номер	Функция преобразования типов данных	Детали преобразования
5	LREAL _TO_ SINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
6	LREAL _TO_ ULINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
7	LREAL _TO_ UDINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
8	LREAL _TO_ UINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
9	LREAL _TO_ USINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
10	REAL _TO_ LREAL	Преобразование, сохраняющее значение
11	REAL _TO_ LINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
12	REAL _TO_ DINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
13	REAL _TO_ INT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
14	REAL _TO_ SINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
15	REAL _TO_ ULINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
16	REAL _TO_ UDINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
17	REAL _TO_ UINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
18	REAL _TO_ USINT	Преобразование с округлением, ошибки дают результат, определяемый разработчиком
19	LINT _TO_ LREAL	Преобразование с потенциальной потерей точности
20	LINT _TO_ REAL	Преобразование с потенциальной потерей точности
21	LINT _TO_ DINT	Ошибки диапазона значений дают результат, определяемый разработчиком
22	LINT _TO_ INT	Ошибки диапазона значений дают результат, определяемый разработчиком
23	LINT _TO_ SINT	Ошибки диапазона значений дают результат, определяемый разработчиком
24	LINT _TO_ ULINT	Ошибки диапазона значений дают результат, определяемый разработчиком
25	LINT _TO_ UDINT	Ошибки диапазона значений дают результат, определяемый разработчиком
26	LINT _TO_ UINT	Ошибки диапазона значений дают результат, определяемый разработчиком
27	LINT _TO_ USINT	Ошибки диапазона значений дают результат, определяемый разработчиком

Продолжение таблицы 23

Номер	Функция преобразования типов данных	Детали преобразования
28	DINT _TO_ LREAL	Преобразование, сохраняющее значение
29	DINT _TO_ REAL	Преобразование с потенциальной потерей точности
30	DINT _TO_ LINT	Преобразование, сохраняющее значение
31	DINT _TO_ INT	Ошибки диапазона значений дают результат, определяемый разработчиком
32	DINT _TO_ SINT	Ошибки диапазона значений дают результат, определяемый разработчиком
33	DINT _TO_ ULINT	Ошибки диапазона значений дают результат, определяемый разработчиком
34	DINT _TO_ UDINT	Ошибки диапазона значений дают результат, определяемый разработчиком
35	DINT _TO_ UINT	Ошибки диапазона значений дают результат, определяемый разработчиком
36	DINT _TO_ USINT	Ошибки диапазона значений дают результат, определяемый разработчиком
37	INT _TO_ LREAL	Преобразование, сохраняющее значение
38	INT _TO_ REAL	Преобразование, сохраняющее значение
39	INT _TO_ LINT	Преобразование, сохраняющее значение
40	INT _TO_ DINT	Преобразование, сохраняющее значение
41	INT _TO_ SINT	Ошибки диапазона значений дают результат, определяемый разработчиком
42	INT _TO_ ULINT	Ошибки диапазона значений дают результат, определяемый разработчиком
43	INT _TO_ UDINT	Ошибки диапазона значений дают результат, определяемый разработчиком
44	INT _TO_ UINT	Ошибки диапазона значений дают результат, определяемый разработчиком
45	INT _TO_ USINT	Ошибки диапазона значений дают результат, определяемый разработчиком
46	SINT _TO_ LREAL	Преобразование, сохраняющее значение
47	SINT _TO_ REAL	Преобразование, сохраняющее значение
48	SINT _TO_ LINT	Преобразование, сохраняющее значение
49	SINT _TO_ DINT	Преобразование, сохраняющее значение
50	SINT _TO_ INT	Преобразование, сохраняющее значение
51	SINT _TO_ ULINT	Ошибки диапазона значений дают результат, определяемый разработчиком
52	SINT _TO_ UDINT	Ошибки диапазона значений дают результат, определяемый разработчиком
53	SINT _TO_ UINT	Ошибки диапазона значений дают результат, определяемый разработчиком

Продолжение таблицы 23

Номер	Функция преобразования типов данных	Детали преобразования
54	SINT _TO_ USINT	Ошибки диапазона значений дают результат, определяемый разработчиком
55	ULINT _TO_ LREAL	Преобразование с потенциальной потерей точности
56	ULINT _TO_ REAL	Преобразование с потенциальной потерей точности
57	ULINT _TO_ LINT	Ошибки диапазона значений дают результат, определяемый разработчиком
58	ULINT _TO_ DINT	Ошибки диапазона значений дают результат, определяемый разработчиком
59	ULINT _TO_ INT	Ошибки диапазона значений дают результат, определяемый разработчиком
60	ULINT _TO_ SINT	Ошибки диапазона значений дают результат, определяемый разработчиком
61	ULINT _TO_ UDINT	Ошибки диапазона значений дают результат, определяемый разработчиком
62	ULINT _TO_ UINT	Ошибки диапазона значений дают результат, определяемый разработчиком
63	ULINT _TO_ USINT	Ошибки диапазона значений дают результат, определяемый разработчиком
64	UDINT _TO_ LREAL	Преобразование, сохраняющее значение
65	UDINT _TO_ REAL	Преобразование с потенциальной потерей точности
66	UDINT _TO_ LINT	Преобразование, сохраняющее значение
67	UDINT _TO_ DINT	Ошибки диапазона значений дают результат, определяемый разработчиком
68	UDINT _TO_ INT	Ошибки диапазона значений дают результат, определяемый разработчиком
69	UDINT _TO_ SINT	Ошибки диапазона значений дают результат, определяемый разработчиком
70	UDINT _TO_ ULINT	Преобразование, сохраняющее значение
71	UDINT _TO_ UINT	Ошибки диапазона значений дают результат, определяемый разработчиком
72	UDINT _TO_ USINT	Ошибки диапазона значений дают результат, определяемый разработчиком
73	UINT _TO_ LREAL	Преобразование, сохраняющее значение
74	UINT _TO_ REAL	Преобразование, сохраняющее значение
75	UINT _TO_ LINT	Преобразование, сохраняющее значение
76	UINT _TO_ DINT	Преобразование, сохраняющее значение
77	UINT _TO_ INT	Ошибки диапазона значений дают результат, определяемый разработчиком
78	UINT _TO_ SINT	Ошибки диапазона значений дают результат, определяемый разработчиком
79	UINT _TO_ ULINT	Преобразование, сохраняющее значение

Окончание таблицы 23

Номер	Функция преобразования типов данных	Детали преобразования
80	UINT _TO_ UDINT	Преобразование, сохраняющее значение
81	UINT _TO_ USINT	Ошибки диапазона значений дают результат, определяемый разработчиком
82	USINT _TO_ LREAL	Преобразование, сохраняющее значение
83	USINT _TO_ REAL	Преобразование, сохраняющее значение
84	USINT _TO_ LINT	Преобразование, сохраняющее значение
85	USINT _TO_ DINT	Преобразование, сохраняющее значение
86	USINT _TO_ INT	Преобразование, сохраняющее значение
87	USINT _TO_ SINT	Ошибки диапазона значений дают результат, определяемый разработчиком
88	USINT _TO_ ULINT	Преобразование, сохраняющее значение
89	USINT _TO_ UDINT	Преобразование, сохраняющее значение
90	USINT _TO_ UINT	Преобразование, сохраняющее значение

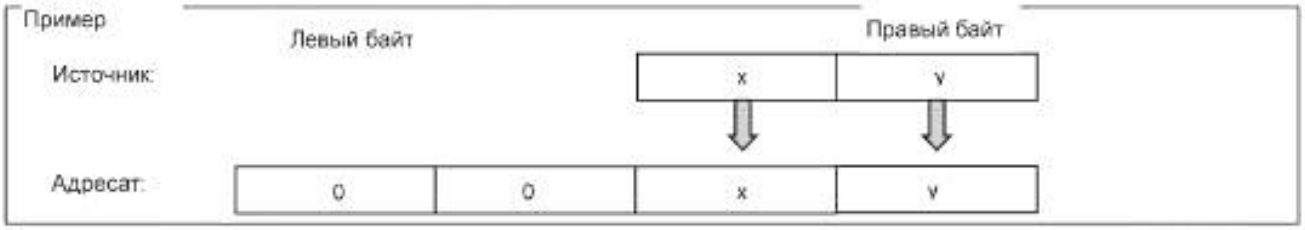
6.6.2.5.4 Преобразование типов битовых типов данных

При преобразовании этого типа данных используются следующие правила:

1 Преобразование типов данных осуществляется как передача двоичных данных.

2 Если исходный тип данных меньше, чем целевой тип данных, исходное значение хранится в самых правых битах целевой переменной, а самые левые биты устанавливаются в ноль.

3 Если исходный тип данных меньше, чем целевой тип данных, только самые правые биты исходной переменной сохраняются в целевом типе данных.



В таблице 24 описаны функции преобразования с деталями, вытекающими из применения описанных выше правил.

Таблица 24 — Преобразование битовых типов данных

Номер	Функция преобразования типов данных			Детали преобразования
1	LWORD	_TO_	DWORD	Двоичная передача самых правых байтов в адресат
2	LWORD	_TO_	WORD	Двоичная передача самых правых байтов в адресат
3	LWORD	_TO_	BYTE	Двоичная передача самых правых байтов в адресат
4	LWORD	_TO_	BOOL	Двоичная передача самого правого бита в адресат

Окончание таблицы 24

Номер	Функция преобразования типов данных			Детали преобразования
5	DWORD	_TO_	LWORD	Двоичная передача самых правых байтов в адресат, самые левые байты устанавливаются в нуль
6	DWORD	_TO_	WORD	Двоичная передача самых правых байтов в адресат
7	DWORD	_TO_	BYTE	Двоичная передача самых правых байтов в адресат
8	DWORD	_TO_	BOOL	Двоичная передача самого правого бита в адресат
9	WORD	_TO_	LWORD	Двоичная передача самых правых байтов в адресат, самые левые байты устанавливаются в нуль
10	WORD	_TO_	DWORD	Двоичная передача самых правых байтов в адресат, самые левые байты устанавливаются в нуль
11	WORD	_TO_	BYTE	Двоичная передача самых правых байтов в адресат
12	WORD	_TO_	BOOL	Двоичная передача самого правого бита в адресат
13	BYTE	_TO_	LWORD	Двоичная передача самых правых байтов в адресат, самые левые байты устанавливаются в нуль
14	BYTE	_TO_	DWORD	Двоичная передача самых правых байтов в адресат, самые левые байты устанавливаются в нуль
15	BYTE	_TO_	WORD	Двоичная передача самых правых байтов в адресат, самые левые байты устанавливаются в нуль
16	BYTE	_TO_	BOOL	Двоичная передача самого правого бита в адресат
17	BYTE	_TO_	CHAR	Передача двоичных данных
18	BOOL	_TO_	LWORD	Дает результат 16#0 или 16#1
19	BOOL	_TO_	DWORD	Дает результат 16#0 или 16#1
20	BOOL	_TO_	WORD	Дает результат 16#0 или 16#1
21	BOOL	_TO_	BYTE	Дает результат 16#0 или 16#1
22	CHAR	_TO_	BYTE	Передача двоичных данных
23	CHAR	_TO_	WORD	Двоичная передача самых правых байтов в адресат, самые левые байты устанавливаются в нуль
24	CHAR	_TO_	DWORD	Двоичная передача самых правых байтов в адресат, самые левые байты устанавливаются в нуль
25	CHAR	_TO_	LWORD	Двоичная передача самых правых байтов в адресат, самые левые байты устанавливаются в нуль
26	WCHAR	_TO_	WORD	Передача двоичных данных
27	WCHAR	_TO_	DWORD	Двоичная передача самых правых байтов в адресат, самые левые байты устанавливаются в нуль
28	WCHAR	_TO_	LWORD	Двоичная передача самых правых байтов в адресат, самые левые байты устанавливаются в нуль

6.6.2.5.5 Преобразование битовых типов данных в числовые типы данных

При преобразовании этого типа данных используются следующие правила:

1 Преобразование типов данных осуществляется как передача двоичных данных.

2 Если исходный тип данных меньше, чем целевой тип данных, исходное значение хранится в самых правых битах целевой переменной, а самые левые биты устанавливаются в ноль.

Пример 1

X: SINT:= 18; W: WORD; W:= SINT_TO_WORD(X); и W получает значение 16#0012.

3 Если исходный тип данных меньше, чем целевой тип данных, только самые правые байты исходной переменной сохраняются в целевом типе данных.

Пример 2

W: WORD: = 16#1234; X: SINT; X:= W; и X получает значение 54 (=16#34).

В таблице 25 описаны функции преобразования с деталями, вытекающими из применения описанных выше правил.

Таблица 25 — Преобразование битовых и числовых типов данных

Номер	Функция преобразования типов данных			Детали преобразования
1	LWORD	_TO_	LREAL	Передача двоичных данных
2	DWORD	_TO_	REAL	Передача двоичных данных
3	LWORD	_TO_	LINT	Передача двоичных данных
4	LWORD	_TO_	DINT	Двоичная передача самых правых байтов в адресат
5	LWORD	_TO_	INT	Двоичная передача самых правых байтов в адресат
6	LWORD	_TO_	SINT	Двоичная передача самого правого байта в адресат
7	LWORD	_TO_	ULINT	Передача двоичных данных
8	LWORD	_TO_	UDINT	Двоичная передача самых правых байтов в адресат
9	LWORD	_TO_	UINT	Двоичная передача самых правых байтов в адресат
10	LWORD	_TO_	USINT	Двоичная передача самого правого байта в адресат
11	DWORD	_TO_	LINT	Двоичная передача в самые правые байты адресата
12	DWORD	_TO_	DINT	Передача двоичных данных
13	DWORD	_TO_	INT	Двоичная передача самых правых байтов в адресат
14	DWORD	_TO_	SINT	Двоичная передача самого правого байта в адресат
15	DWORD	_TO_	ULINT	Двоичная передача в самые правые байты адресата
16	DWORD	_TO_	UDINT	Передача двоичных данных
17	DWORD	_TO_	UINT	Двоичная передача самых правых байтов в адресат
18	DWORD	_TO_	USINT	Двоичная передача самого правого байта в адресат
19	WORD	_TO_	LINT	Двоичная передача в самые правые байты адресата
20	WORD	_TO_	DINT	Двоичная передача в самые правые байты адресата
21	WORD	_TO_	INT	Передача двоичных данных
22	WORD	_TO_	SINT	Двоичная передача самого правого байта в адресат
23	WORD	_TO_	ULINT	Двоичная передача в самые правые байты адресата
24	WORD	_TO_	UDINT	Двоичная передача в самые правые байты адресата
25	WORD	_TO_	UINT	Передача двоичных данных
26	WORD	_TO_	USINT	Двоичная передача самого правого байта в адресат
27	BYTE	_TO_	LINT	Двоичная передача в самые правые байты адресата
28	BYTE	_TO_	DINT	Двоичная передача в самые правые байты адресата

Продолжение таблицы 25

Номер	Функция преобразования типов данных			Детали преобразования
29	BYTE	_TO_	INT	Двоичная передача в самые правые байты адресата
30	BYTE	_TO_	SINT	Передача двоичных данных
31	BYTE	_TO_	ULINT	Двоичная передача в самые правые байты адресата
32	BYTE	_TO_	UDINT	Двоичная передача в самые правые байты адресата
33	BYTE	_TO_	UINT	Двоичная передача в самые правые байты адресата
34	BYTE	_TO_	USINT	Передача двоичных данных
35	BOOL	_TO_	LINT	Дает результат 0 или 1
36	BOOL	_TO_	DINT	Дает результат 0 или 1
37	BOOL	_TO_	INT	Дает результат 0 или 1
38	BOOL	_TO_	SINT	Дает результат 0 или 1
39	BOOL	_TO_	ULINT	Дает результат 0 или 1
40	BOOL	_TO_	UDINT	Дает результат 0 или 1
41	BOOL	_TO_	UINT	Дает результат 0 или 1
42	BOOL	_TO_	USINT	Дает результат 0 или 1
43	LREAL	_TO_	LWORD	Передача двоичных данных
44	REAL	_TO_	DWORD	Передача двоичных данных
45	LINT	_TO_	LWORD	Передача двоичных данных
46	LINT	_TO_	DWORD	Двоичная передача самых правых байтов в адресат
47	LINT	_TO_	WORD	Двоичная передача самых правых байтов в адресат
48	LINT	_TO_	BYTE	Двоичная передача самого правого байта в адресат
49	DINT	_TO_	LWORD	Двоичная передача в самые правые байты адресата, остальные байты = 0
50	DINT	_TO_	DWORD	Передача двоичных данных
51	DINT	_TO_	WORD	Двоичная передача самых правых байтов в адресат
52	DINT	_TO_	BYTE	Двоичная передача самого правого байта в адресат
53	INT	_TO_	LWORD	Двоичная передача в самые правые байты адресата, остальные байты = 0
54	INT	_TO_	DWORD	Двоичная передача в самые правые байты адресата, остальные байты = 0
55	INT	_TO_	WORD	Передача двоичных данных
56	INT	_TO_	BYTE	Двоичная передача самого правого байта в адресат
57	SINT	_TO_	LWORD	Двоичная передача в самые правые байты адресата, остальные байты = 0
58	SINT	_TO_	DWORD	Двоичная передача в самые правые байты адресата, остальные байты = 0
59	SINT	_TO_	WORD	Передача двоичных данных
60	SINT	_TO_	BYTE	Передача двоичных данных

Окончание таблицы 25

Номер	Функция преобразования типов данных			Детали преобразования
61	ULINT	_TO_	LWORD	Передача двоичных данных
62	ULINT	_TO_	DWORD	Двоичная передача самых правых байтов в адресат
63	ULINT	_TO_	WORD	Двоичная передача самых правых байтов в адресат
64	ULINT	_TO_	BYTE	Двоичная передача самого правого байта в адресат
65	UDINT	_TO_	LWORD	Двоичная передача в самые правые байты адресата, остальные байты = 0
66	UDINT	_TO_	DWORD	Передача двоичных данных
67	UDINT	_TO_	WORD	Двоичная передача самых правых байтов в адресат
68	UDINT	_TO_	BYTE	Двоичная передача самого правого байта в адресат
69	UINT	_TO_	LWORD	Двоичная передача в самые правые байты адресата, остальные байты = 0
70	UINT	_TO_	DWORD	Двоичная передача в самые правые байты адресата, остальные байты = 0
71	UINT	_TO_	WORD	Передача двоичных данных
72	UINT	_TO_	BYTE	Двоичная передача самого правого байта в адресат
73	USINT	_TO_	LWORD	Двоичная передача в самые правые байты адресата, остальные байты = 0
74	USINT	_TO_	DWORD	Двоичная передача в самые правые байты адресата, остальные байты = 0
75	USINT	_TO_	WORD	Передача двоичных данных
76	USINT	_TO_	BYTE	Передача двоичных данных

6.6.2.5.6 Преобразование типов данных даты и времени

В таблице 26 показывается преобразование типов данных даты и времени.

Таблица 26 — Преобразование типов данных даты и времени

Номер	Функция преобразования типов данных			Детали преобразования
1	LTIME	_TO_	TIME	Ошибки диапазона значений дают результат, определяемый разработчиком, и может происходить потеря точности
2	TIME	_TO_	LTIME	Ошибки диапазона значений дают результат, определяемый разработчиком, и может происходить потеря точности
3	LDT	_TO_	DT	Ошибки диапазона значений дают результат, определяемый разработчиком, и может происходить потеря точности
4	LDT	_TO_	DATE	Преобразует только содержащуюся дату, ошибки диапазона значений дают результат, определяемый разработчиком
5	LDT	_TO_	LTOD	Преобразует только содержащееся время суток
6	LDT	_TO_	TOD	Преобразует только содержащееся время суток, может происходить потеря точности
7	DT	_TO_	LDT	Ошибки диапазона значений дают результат, определяемый разработчиком, и может происходить потеря точности

Окончание таблицы 26

Номер	Функция преобразования типов данных			Детали преобразования
8	DT	_TO_	DATE	Преобразует только содержащуюся дату, ошибки диапазона значений дают результат, определяемый разработчиком
9	DT	_TO_	LTOD	Преобразует только содержащееся время суток, ошибки диапазона значений дают результат, определяемый разработчиком
10	DT	_TO_	TOD	Преобразует только содержащееся время суток, ошибки диапазона значений дают результат, определяемый разработчиком
11	LTOD	_TO_	TOD	Преобразование, сохраняющее значение
12	TOD	_TO_	LTOD	Ошибки диапазона значений дают результат, определяемый разработчиком, и может происходить потеря точности

6.6.2.5.7 Преобразование символьных типов данных

В таблице 27 показывается преобразование символьных типов данных.

Таблица 27 — Преобразование символьных типов данных

Номер	Функция преобразования типов данных			Детали преобразования
1	WSTRING	_TO_	STRING	Преобразуются только символы, поддерживаемые разработчиком в типе данных STRING, преобразование остальных символов определяется разработчиком
2	WSTRING	_TO_	WCHAR	Передается первый символ строки, если строка пустая, целевая переменная имеет неопределенное значение
3	STRING	_TO_	WSTRING	Преобразует символы строки как определено разработчиком в соответствующие символы набора символов ИСО/МЭК 10646 (UTF-16)
4	STRING	_TO_	CHAR	Передается первый символ строки, если строка пустая, целевая переменная имеет неопределенное значение
5	WCHAR	_TO_	WSTRING	Дает строку с фактической длиной в один символ
6	WCHAR	_TO_	CHAR	Преобразуются только символы, поддерживаемые разработчиком в типе данных CHAR, преобразование остальных символов определяется разработчиком
7	CHAR	_TO_	STRING	Дает строку с фактической длиной в один символ
8	CHAR	_TO_	WCHAR	Преобразует символ как определено разработчиком в соответствующий символ набора символов UTF-16

6.6.2.5.8 Числовые и арифметические функции

Стандартное графическое представление, имена функций, типы входных и выходных переменных и описания функций одной числовой переменной определяются в таблице 28. Данные функции перегружаются на определенных родовых типах данных и могут быть типизированными. В таких функциях вход и выход имеют одинаковый тип.

Стандартное графическое представление, имена и символы функций и описания арифметических функций двух и более переменных показываются в таблице 29. Данные функции перегружаются на всех определенных числовых типах данных, и могут быть типизированными.

Точность числовых функций выражается в терминах одной или более зависимостей, определяемых разработчиком.

Ошибка возникает, если результат вычисления одной из таких функций превышает диапазон значений, указанных для типа данных выхода функции, или если предпринимается попытка деления на ноль.

Таблица 28 — Числовые и арифметические функции

Номер	Описание (имя функции)	Тип входной/выходной переменной	Объяснение
	Графическая форма +----------+ * --| ** |-- * +----------+ (*) — Тип входной/выходной переменной (**) — Имя функции		**Пример использования в языке ST** A:= SIN(B); (язык ST)
	Общие функции		
1	ABS(x)	ANY_NUM	Абсолютная величина
2	SQRT(x)	ANY_REAL	Квадратный корень
	Логарифмические функции		
3	LN(x)	ANY_REAL	Натуральный логарифм
4	LOG(x)	ANY_REAL	Десятичный логарифм
5	EXP(x)	ANY_REAL	Экспонента
	Тригонометрические функции		
6	SIN(x)	ANY_REAL	Синус от входного значения в радианах
7	COS(x)	ANY_REAL	Косинус от входного значения в радианах
8	TAN(x)	ANY_REAL	Тангенс от входного значения в радианах
9	ASIN(x)	ANY_REAL	Главное значение арксинуса
10	ACOS(x)	ANY_REAL	Главное значение арккосинуса
11	ATAN(x)	ANY_REAL	Главное значение арктангенса
12	ATAN2(yx) +--------+ | ATAN2 | ANY_REAL--|Y |--ANY_REAL ANY_REAL--|X | +--------+	ANY_REAL	Угол между положительным направлением оси x плоскости и точкой, заданной координатами (x, y). Значение угла является положительным для углов против часовой стрелки (верхняя полуплоскость, y > 0), и отрицательным для углов по часовой стрелке (нижняя полуплоскость, y < 0).

Таблица 29 — Арифметические функции

Номер	Описание	Название	Символ (оператор)	Объяснение
	Графическая форма			**Пример использования в языке**
	+------+ ANY_NUM --| *** |-- ANY_NUM ANY_NUM --| | . --| | . --| | ANY_NUM --| | +------+ (***) — Название или символ			как вызов функции: A:= ADD(B, C, D); или как оператор (символ) A:= B + C + D;

Окончание таблицы 29

Номер	Описание	Название	Символ (оператор)	Объяснение
	Расширяемые арифметические функции			
1[c]	Сложение	ADD	+	OUT:= IN1 + IN2 +... + INn
2	Умножение	MUL	*	OUT:= IN1 * IN2 *... * INn
	Нерасширяемые арифметические функции			
3[c]	Вычитание	SUB	-	OUT:= IN1 * IN2 *... * INn
4[d]	Деление	DIV	/	OUT:= IN1 / IN2
5[e]	Остаток по модулю	MOD		OUT:= IN1 modulo IN2
6[f]	Экспонента	EXPT	**	OUT:= IN1IN2
7[g]	Пересылка	MOVE	:=	OUT:= IN

Примечание 1 — Непустые значения в графе «Символ» могут использоваться как операторы в текстовых языках.

Примечание 2 — Запись IN1, IN2, ..., INn ссылается на входные переменные в нисходящем порядке; OUT ссылается на выходную переменную.

Примечание 3 — Примеры использования и объявления даны на языке ST.

[a] Когда представление функции поддерживается именем, это отмечается суффиксом «n» в декларации соответствия.

Например, «1n» представляет запись «ADD».

[b] Когда представление функции поддерживается символом, это отмечается суффиксом «s» в декларации соответствия. Например, «1s» представляет запись «+».

[c] Входные и выходные переменные для этих функций имеют тип ANY_MAGNITUDE.

[d] Результатом деления целых чисел является целое число того же типа с усечением значения по направлению к нулю, например, 7/3 = 2 и (-7)/3 = -2.

[e] Для этой функции, IN1 и IN2 имеют родовой тип ANY_INT. Результат выполнения этой функции MOD эквивалентен вычислению следующих операторов языка ST:

```
IF (IN2 = 0)
   THEN OUT:=0;
   ELSE OUT:=IN1 - (IN1/IN2)*IN2;
END_IF
```

[f] Для функции EXPT, IN1 имеет тип ANY_REAL, а IN2 — тип ANY_NUM. Тип выходной переменной — такой же, как тип переменной IN1.

[g] Функция MOVE имеет ровно одну входную переменную (IN) типа ANY и одну входную переменную (OUT) типа ANY.

6.6.2.5.9 Битовые строки и поразрядные логические функции

Стандартное графическое представление, имена функций и описания функций сдвига для одной переменной типа битовой строки определяются в таблице 30. Данные функции перегружаются для типов битовой строки и могут быть типизированы.

Стандартное графическое представление, имена функций и символов и описания поразрядных логических функций определяются в таблице 31. Данные функции являются расширяемыми (за исключением функции NOT), перегружаются для всех типов битовых строк, и могут быть типизированными.

Таблица 30 — Функции битового сдвига

Номер	Описание	Название	Объяснение
	Графическая форма		**Пример использования a**
	 (***) — Имя функции		A:= SHL(IN:=B, N:=5); (язык ST)
1	Сдвиг влево	SHL	OUT:= IN, сдвинутому влево на N бит, биты справа заполняются нулями
2	Сдвиг вправо	SHR	OUT:= IN, сдвинутому вправо на N бит, биты слева заполняются нулями
3	Циклический сдвиг влево	ROL	OUT:= IN, циклически сдвинутому влево на N бит
4	Циклический сдвиг вправо	ROR	OUT:= IN, циклически сдвинутому вправо на N бит
	Примечание 1 — Запись OUT ссылается на результат функции. ***Пример —*** ***IN:= 2#0001_1001 of type BYTE, N = 3*** ***SHL(IN, 3) = 2#1100_1000*** ***SHR(IN, 3) = 2#0000_0011*** ***ROL(IN, 3) = 2#1100_1000*** ***ROR(IN, 3) = 2#0010_0011*** Примечание 2 — IN типа BOOL (один бит) не имеет смысла.		
	[a] Если входная переменная N меньше нуля, возникает ошибка.		

Таблица 31 — Поразрядные логические функции

Номер	Описание	Название	Символ	Объяснение (см. примечание 3)
	Графическая форма  (***) — Название или символ			**Примеры использования** (см. примечание 5) A:= AND(B, C, D); или A:= B & C & D;
1	И	AND	& (см. примечание 1)	OUT:= IN1 & IN2 &... & INn
2	Или	OR	>= 1 (см. примечание 2)	OUT:= IN1 OR IN2 OR... OR INn
3	Исключающее или	XOR	= 2k+1 (см. примечание 2)	OUT:= IN1 XOR IN2 XOR... XOR INn
4	Отрицание	NOT		OUT:= NOT IN1 (см. примечание 4)

Окончание таблицы 31

Примечание 1 — Данный символ подходит для использования в качестве оператора в текстовых языках, как показано в таблицах 68 и 71. Примечание 2 — Данный символ не подходит для использования в качестве оператора в текстовых языках. Примечание 3 — Запись IN1, IN2, ..., INn ссылается на входные переменные в нисходящем порядке; OUT ссылается на выходную переменную. Примечание 4 — Графическое отрицание сигналов типа BOOL также может быть осуществлено. Примечание 5 — Примеры использования и объявления даны на языке ST.
[a] Когда представление функции поддерживается именем, это отмечается суффиксом «n» в декларации соответствия. Например, «1n» представляет запись «AND». [b] Когда представление функции поддерживается символом, это отмечается суффиксом «s» в декларации соответствия. Например, «1s» представляет запись «&».

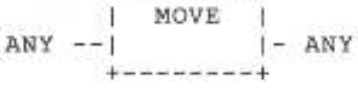
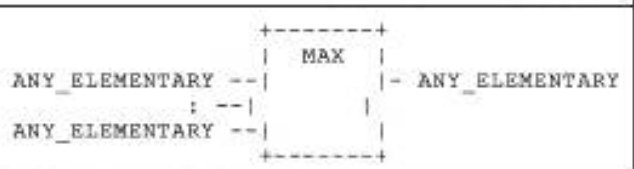
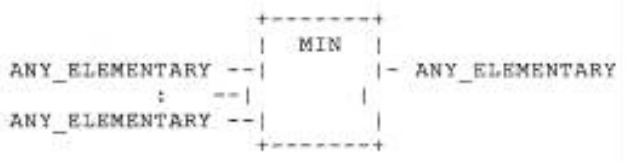
6.6.2.5.10 Функции выбора и сравнения

Функции выбора и сравнения перегружены для всех типов данных. Стандартное графическое представление, имена функций и символов и описания функций сравнения показываются в таблице 32.

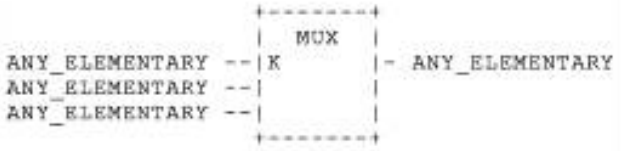
Стандартное графическое представление, имена функций и символов и описания функций сравнения показываются в таблице 33. Все функции сравнения (за исключением функции NE) являются расширяемыми.

Сравнение битовых строк выполняется поразрядно от самого левого бита к самому правому. Предполагается, что более короткие битовые строки заполнены слева нулями при сравнении с более длинными битовыми строками, то есть сравнение переменных типа битовой строки будет иметь такой же результат, как сравнение целых чисел без знака.

Таблица 32 — Функции выбора[d]

Номер	Описание	Имя	Графическая форма	Объяснение/Пример
1	Пересылка[a], [d] (присваивание)	MOVE		OUT:= IN
2	Двоичный выбор[d]	SEL		OUT:= IN0 if G = 0 OUT:= IN1 if G = 1 ***Пример 1 —*** ***A:= SEL (G := 0,*** ***IN0:= X,*** ***IN1:= 5);***
3	Расширяемая функция максимума	MAX		OUT:= MAX(IN1, IN2, ..., INn); ***Пример 2 —*** ***A:= MAX(B, C , D);***
4	Расширяемая функция минимума	MIN		OUT:= MIN (IN1, IN2,..., Nn) ***Пример 3 —*** ***A:= MIN(B, C, D);***

Окончание таблицы 32

Номер	Описание	Имя	Графическая форма	Объяснение/Пример
5	Ограничитель	LIMIT		OUT:= MIN (MAX(IN, MN),MX); ***Пример 4 —*** ***A:= LIMIT(IN:= B,*** ***MN:= 0,*** ***MX:= 5);***
6	Расширенный мультиплексор [b], [c], [d], [e]	MUX		a, b, c: Выбирает одну из N входных переменных в зависимости от входной переменной K ***Пример 5 —*** ***A:= MUX(0, B, C, D);*** ***имеет такой же эффект как*** ***A:= B;***

Примечание 1 — Запись IN1, IN2, ..., INn ссылается на входные переменные в нисходящем порядке; OUT ссылается на выходную переменную.

Примечание 2 — Примеры использования и объявления даны на языке ST.

[a] Функция MOVE имеет ровно одну входную переменную IN типа ANY и одну входную переменную OUT типа ANY.

[b] Неименованные входные переменные функции MUX имеют неявные имена IN0, IN1,..., INn-1 в нисходящем порядке, где n из общего числа входных переменных. Данные имена могут (но необязательно) быть показаны в графическом представлении.

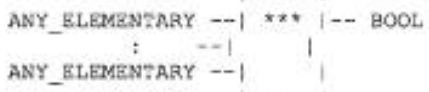
[c] Функция MUX может быть типизирована в форме MUX_*_**, где * — тип входной переменной K и ** — тип других входных переменных и выхода.

[d] Разработчику разрешается (но необязательно) поддерживать выбор среди переменных определенных пользователем типов данных, чтобы подтвердить соответствие этому свойству.

[e] Если фактическое значение входной переменной K функции MUX находится вне диапазона {0 ... n-1}, возникает ошибка.

Таблица 33 — Функции сравнения

Номер	Описание	Имя[a]	Символ[b]	Объяснение (расширяемая: 2 или большее число операндов)
	Графическая форма			**Пример использования**

	
	(***) Имя или символ
				A:= GT(B, C, D); // Имя функции
				или
				A:= (B>C) & (C>D); // Символ

1	Убывающая последовательность	GT	>	OUT:= (IN1>IN2)& (IN2>IN3) &.. & (INn-1 > INn)
2	Монотонная последовательность	GE	>=	OUT:= (IN1>=IN2)&(IN2>=IN3)&.. & (INn-1 >= INn)

Окончание таблицы 33

Номер	Описание	Имя[a]	Символ[b]	Объяснение (расширяемая: 2 или большее число операндов)
3	Equality	EQ	=	OUT:= (IN1=IN2)&(IN2=IN3) &.. & (INn-1 = INn)
4	Монотонная последовательность	LE	<=	OUT:= (IN1<=IN2)&(IN2<=IN3)&.. & (INn-1 <= INn)
5	Increasing sequence	LT	<	OUT:= (IN1<IN2)& (IN2<IN3) &.. & (INn-1 < INn)
6	Inequality	NE	<>	OUT:= (IN1<>IN2) (нерасширяемая)

Примечание 1 — Запись IN1, IN2, ..., INn ссылается на входные переменные в нисходящем порядке; OUT ссылается на выходную переменную.

Примечание 2 — Все символы, показанные в данной таблице, могут использоваться как операторы в текстовых языках.

Примечание 3 — Примеры использования и объявления даны на языке ST.

Примечание 4 — Стандартные функции сравнения могут также определяться зависящими от языка, например, на языке релейно-контактных схем.

[a] Когда представление функции поддерживается именем, это отмечается суффиксом «n» в декларации соответствия. Например, «1n» представляет запись «GT».

[b] Когда представление функции поддерживается символом, это отмечается суффиксом «s» в декларации соответствия. Например, «1s» представляет запись «>».

К строкам символов применяется таблица 33. Вместо однобайтовой строки может использоваться переменная типа CHAR или WCHAR, соответственно.

При сравнении двух строк разной длины более короткая строка рассматривается расширенной справа символами с нулевым значением до длины более длинной строки. Сравнение осуществляется слева направо на базе числовых значений кодов символа в наборе символов.

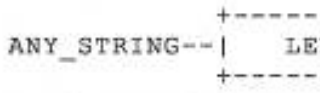
Пример — Строка символов 'Z' больше строки символов 'AZ' ('Z' > 'A') и строка символов 'AZ' больше чем строка 'ABC' ('A' = 'A' и 'Z' > 'B').

Стандартное графическое представление, имена и символы функций и описания дополнительных функций сравнения строк символов показываются в таблице 34. При выполнении данных операций, позиции символов в строке считаются пронумерованными 1, 2, ..., L, начиная с самого левого символа, где L — длина строки.

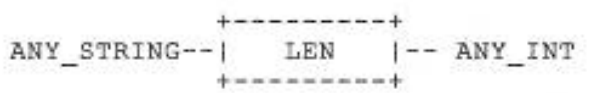
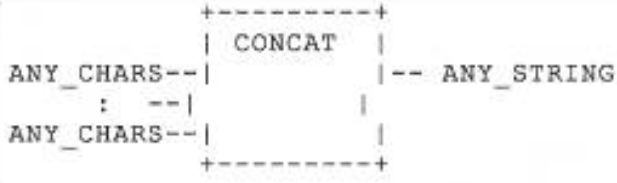
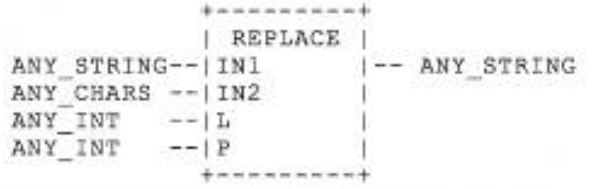
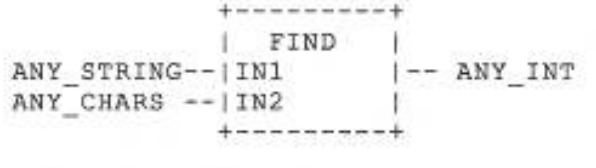
Ошибка возникает, если:

- фактическое значение какой-либо входной переменной типа ANY_INT в таблице 34 меньше нуля;
- вычисление функции приводит к попытке (1) получить доступ к несуществующей позиции в строке, или (2) получить строку длиннее определенной разработчиком максимальной длины строки;
- аргументы типа данных STRING или CHAR и аргументы типа данных WSTRING или WCHAR смешаны в одной функции.

Таблица 34 — Строковые функции

Номер	Описание	Графическая форма	Пример
1	Длина строки		Длина строки A:= LEN('ASTRING'); ... эквивалентно A:= 7;

Окончание таблицы 34

Номер	Описание	Графическая форма	Пример
2	Левая подстрока		L самых левых символов из IN A:= LEFT(IN:='ASTR', L:=3); эквивалентно A:= 'AST';
3	Правая подстрока		L самых правых символов из IN A:= LEFT(IN:='ASTR', L:=3); эквивалентно A:= 'STR';
4	Средняя подстрока		L символов из IN, начиная с P-й позиции символа A:= MID(IN:='ASTR', L:=2, P:=2); эквивалентно A:= 'ST';
5	Расширяемая конкатенация		Расширяемая конкатенация A:= CONCAT('AB','CD','E'); эквивалентно A:= 'ABCDE';
6	Вставить		Вставить строку IN2 в строку IN1 после P-й позиции символа A:= INSERT(IN1:='ABC', IN2:='XY', P=2); эквивалентно A:= 'ABXYC';
7	Удалить		Удалить L символов из строки IN, начиная с P-й позиции символа A:= DELETE(IN:='ABXYC', L:=2, P:=3); эквивалентно A:= 'ABC';
8	Заменить		Заменить L символов строки IN1 строкой IN2, начиная в P-й позиции символа A:= REPLACE(IN1:='ABCDE', IN2:='X', L:=2, P:=3); эквивалентно A:= 'ABXE';
9	Найти		Найти позицию символа в начала первого вхождения строки IN2 в строку IN1. Если вхождения строки IN2 не обнаружены, то OUT:= 0. A:= FIND(IN1:='ABCBC', IN2:='BC'); ... эквивалентно A:= 2;

Примечание 1 — Примеры в этой таблице даются на языке ST.

Примечание 2 — Все входные переменные функции CONCAT имеют тип ANY_CHARS, то есть могут быть также типа CHAR или типа WCHAR.

Примечание 3 — Входная переменная IN2 функций INSERT, REPLACE, FIND имеет тип ANY_CHARS, то есть может также иметь тип CHAR или WCHAR.

6.6.2.5.11 Функции даты и продолжительности времени

В функциях сравнения и выбора разрешено также использование входных и выходных переменных, имеющих типы данных времени и продолжительности времени, показанные в таблице 35.

Возникает ошибка, если результат вычисления одной из этих функций превышает определенный разработчиком диапазон значений выходного типа данных.

Таблица 35 — Числовые функции типов данных времени и продолжительности времени

Номер	Описание (имя функции)	Символ	IN1	IN2	OUT
1a	ADD	+	TIME, LTIME	TIME, LTIME	TIME, LTIME
1b	ADD_TIME	+	TIME	TIME	TIME
1c	ADD_LTIME	+	LTIME	LTIME	LTIME
2a	ADD	+	TOD, LTOD	LTIME	TOD, LTOD
2b	ADD_TOD_TIME	+	TOD	TIME	TOD
2c	ADD_LTOD_LTIME	+	LTOD	LTIME	LTOD
3a	ADD	+	DT, LDT	TIME, LTIME	DT, LDT
3b	ADD_DT_TIME	+	DT	TIME	DT
3c	ADD_LDT_LTIME	+	LDT	LTIME	LDT
4a	SUB	–	TIME, LTIME	TIME, LTIME	TIME, LTIME
4b	SUB_TIME	–	TIME	TIME	TIME
4c	SUB_LTIME	–	LTIME	LTIME	LTIME
5a	SUB	–	DATE	DATE	TIME
5b	SUB_DATE_DATE	–	DATE	DATE	TIME
5c	SUB_LDATE_LDATE	–	LDATE	LDATE	LTIME
6a	SUB	–	TOD, LTOD	TIME, LTIME	TOD, LTOD
6b	SUB_TOD_TIME	–	TOD	TIME	TOD
6c	SUB_LTOD_LTIME	–	LTOD	LTIME	LTOD
7a	SUB	–	TOD, LTOD	TOD, LTOD	TIME, LTIME
7b	SUB_TOD_TOD	–	TOD	TOD	TIME
7c	SUB_TOD_TOD	–	LTOD	LTOD	LTIME
8a	SUB	–	DT, LDT	TIME, LTIME	DT, LDT
8b	SUB_DT_TIME	–	DT	TIME	DT
8c	SUB_LDT_LTIME	–	LDT	LTIME	LDT
9a	SUB	–	DT, LDT	DT, LDT	TIME, LTIME
9b	SUB_DT_DT	–	DT	DT	TIME
9c	SUB_LDT_LDT	–	LDT	LDT	LTIME
10a	MUL	*	TIME, LTIME	ANY_NUM	TIME, LTIME
10b	MUL_TIME	*	TIME	ANY_NUM	TIME
10c	MUL_LTIME	*	LTIME	ANY_NUM	LTIME
11a	DIV	/	TIME, LTIME	ANY_NUM	TIME, LTIME

Окончание таблицы 35

Номер	Описание (имя функции)	Символ	IN1	IN2	OUT
11b	DIV_TIME	/	TIME	ANY_NUM	TIME
11c	DIV_LTIME	/	LTIME	ANY_NUM	LTIME
Примечание — Данные стандартные функции поддерживают перегрузку, но только в пределах обоих наборов типов данных (TIME, DT, DATE, TOD) и (LTIME, LDT, DATE, LTOD).					

Пример —

Операторы языка ST

X:= DT#1986-04-28-08:40:00;

Y:= DT_TO_TOD(X);

W:= DT_TO_DATE(X);

имеют такой же результат, как и операторы с «извлеченными» данными.

X:= DT#1986-04-28-08:40:00;

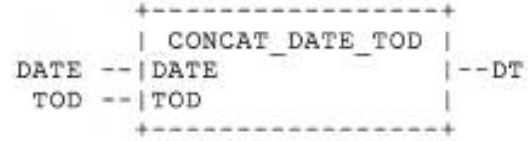
Y:= TIME_OF_DAY#08:40:00;

W:= DATE#1986-04-28;

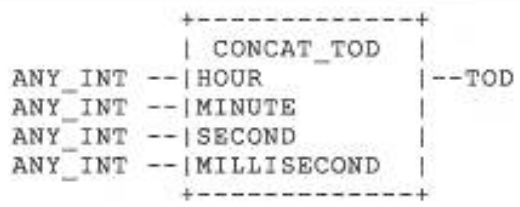
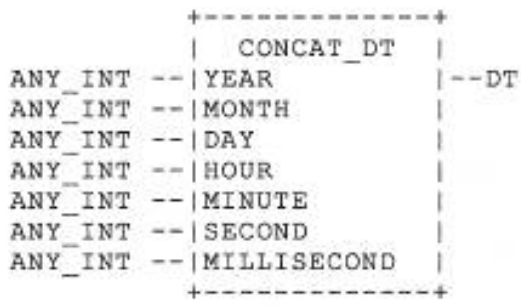
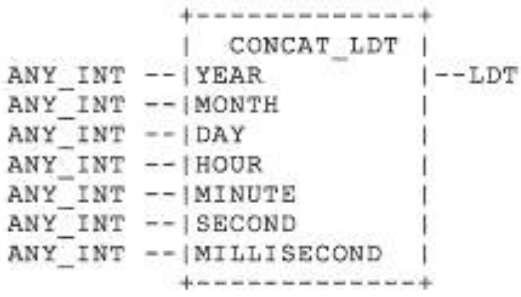
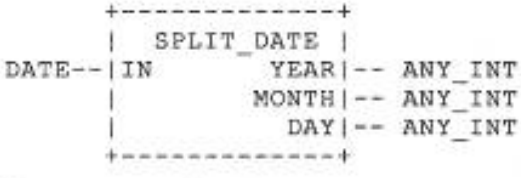
Функции конкатенации и расщепления данных, показанные в таблице 36, определены и для обработки данных даты и времени. Дополнительно определена функция получения дня недели.

Возникает ошибка, если результат вычисления одной из этих функций превышает определенный разработчиком диапазон значений выходного типа данных.

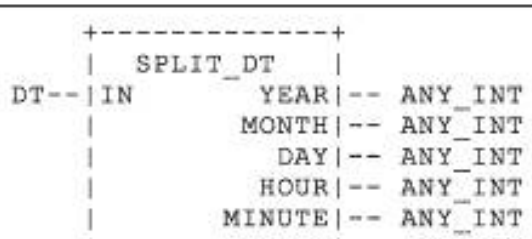
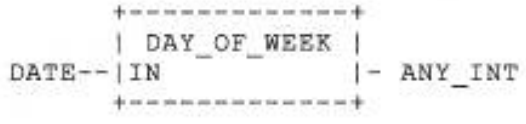
Таблица 36 — Дополнительные CONCAT и SPLIT функции для типов данных даты и времени

Номер	Описание	Графическая форма	Пример
	Конкатенация типов данных даты и времени		
1a	CONCAT_DATE _TOD		Соединить дату и время VAR myD: DATE; END_VAR myD:= CONCAT_DATE_TOD (D#2010-03-12, TOD#12:30:00);
1b	CONCAT_DATE _LTOD		Соединить дату и время суток VAR myD: DATE; END_VAR myD:= CONCAT_DATE_LTOD (D#2010-03-12, TOD#12:30:12.1223452);
2	CONCAT_DATE		Соединить дату и время суток VAR myD: DATE; END_VAR myD:= CONCAT_DATE (2010,3,12);

Продолжение таблицы 36

Номер	Описание	Графическая форма	Пример
3a	CONCAT_TOD		Соединить время суток VAR myTOD: TOD; END_VAR myTD:=CONCAT_TOD(16,33,12,0);
3b	CONCAT_LTOD		Соединить время суток VAR myTOD: LTOD; END_VAR myTD:=CONCAT_TOD(16,33,12,0);
4a	CONCAT_DT		Соединить время суток VAR myDT: DT; Day: USINT; END_VAR Day := 17; myDT:= CONCAT_DT (2010,3,Day,12,33,12,0);
4b	CONCAT_LDT		Соединить время суток VAR myDT: LDT; Day: USINT; END_VAR Day := 17; myDT:= CONCAT_LDT (2010,3,Day,12,33,12,0);
	Расщепление типов даты и времени		
5	SPLIT_DATE	 См. примечание 2	Расщепить дату VAR myD: DATE:= DATE#2010-03-10; myYear: UINT; myMonth, myDay: USINT; END_VAR SPLIT_DATE (myD, myYear, myMonth, myDay);

Продолжение таблицы 36

Номер	Описание	Графическая форма	Пример
6a	SPLIT_TOD	 См. примечание 2	Расщепить время суток VAR myTOD: TOD:= TOD#14:12:03; myHour, myMin, mySec: USINT; myMilliSec: UINT; END_VAR SPLIT_TOD(myTOD, myHour, myMin, mySec, myMilliSec);
6b	SPLIT_LTOD	 См. примечание 2	Расщепить время суток VAR myTOD: LTOD:= TOD#14:12:03; myHour, myMin, mySec: USINT; myMilliSec: UINT; END_VAR SPLIT_TOD(myTOD, myHour, myMin, mySec, myMilliSec);
7a	SPLIT_DT	 См. примечание 2	Расщепить дату VAR myDT: DT := DT#2010-03-10-14:12:03:00; myYear, myMilliSec: UINT; myMonth, myDay, myHour, myMin, mySec: USINT; END_VAR SPLIT_DT(myDT, myYear, myMonth, myDay, myHour, myMin, mySec, myMilliSec);
7b	SPLIT_LDT	 См. примечание 2	Расщепить дату VAR myDT: LDT := DT#2010-03-10-14:12:03:00; myYear, myMilliSec: UINT; UINT; myMonth, myDay, myHour, myMin, mySec: USINT; END_VAR SPLIT_DT(myDT, myYear, myMonth, myDay, myHour, myMin, mySec, myMilliSec);
	Получить день недели		
8	DAY_OF_WEEK	 См. примечание 2	Получить день недели: VAR myD: DATE:= DATE#2010-03-10; myDoW: USINT; END_VAR myDoW:= DAY_OF_WEEK(myD);
	Функция DAY_OF_WEEK возвращает 0 для воскресенья, 1 для понедельника, …, 6 для субботы		

Окончание таблицы 36

Примечание 1 — Тип данных входной переменной YEAR должен быть, по меньшей мере, 16-битовым типом для поддержки допустимого значения года.

Примечание 2 — Типы данных для типов данных выходных переменных ANY_INT определяет разработчик.

Примечание 3 — Разработчик может определять дополнительные входные и выходные переменные в соответствии с поддерживаемой точностью, например, микросекунды и наносекунды.

6.6.2.5.12 Функции преобразования порядка следования байтов

Функции преобразования порядка следования байтов преобразуют этот порядок при обмене информацией с определенным разработчиком PLC.

Порядок следования байтов определяет упорядочение байтов в длинных типах данных и переменных.

Значения данных в порядке big endian (от старшего к младшему) помещаются в байтах памяти, начиная с левого байта и оканчивая правым.

Значения данных в порядке little endian (от младшего к старшему) помещаются в байтах памяти, начиная с правого байта и оканчивая левым.

Независимо от порядка следования байтов, битовое смещение 0 адресует самый правый бит типа данных.

Использование частичного доступа с маленькими числам возвращает нижнюю часть значения независимо от указанного порядка следования байтов.

Пример 1 — Порядок следования байтов

TYPE D: DWORD:= 16#1234_5678; END_TYPE;

Расположение в памяти

для порядка big endian: 16#12, 16#34, 16#56, 16#78

для порядка little endian: 16#78, 16#56, 16#34, 16#12.

Пример 2 — Порядок следования байтов

TYPE L: ULINT:= 16#1234_5678_9ABC_DEF0; END_TYPE;

Расположение в памяти

для порядка big endian: 16#12, 16#34,16#56, 16#78, 16#9A, 16#BC, 16#DE, 16#F0

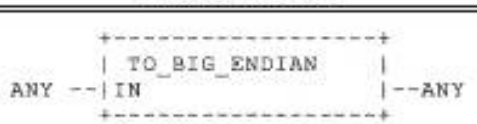
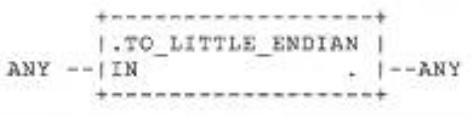
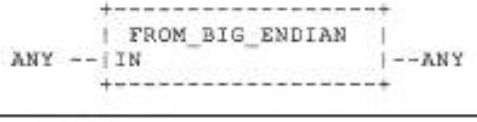
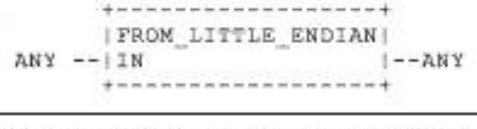
для порядка little endian: 16#F0, 16#DE, 16#BC, 16#9A, 16#78, 16#56, 16#34, 16#12

В качестве входных и выходных переменных функций преобразования порядка следования байтов поддерживаются следующие типы данных:

- ANY_INT с размером больше или равным 16 бит.
- ANY_BIT с размером больше или равным 16 бит.
- ANY_REAL;
- WCHAR;
- TIME;
- массивы этих типов данных;
- структуры, содержащие компоненты этих типов данных.

Другие типы не преобразуются, но могут содержаться в структурах, подлежащих преобразованию. Функции преобразования порядка следования байтов показаны в таблице 37.

Таблица 37 — Функции преобразования порядка следования байтов

Номер	Описание	Графическая форма	Текстовая форма
1	TO_BIG_ENDIAN		Преобразование в формат данных big endian A:= TO_BIG_ENDIAN(B);
2	TO_LITTLE_ENDIAN		Преобразование в формат данных little endian B:= TO_LITTLE_ENDIAN(A);
3	BIG_ENDIAN_TO		Преобразование из формата данных big endian A:= FROM_BIG_ENDIAN(B);
4	LITTLE_ENDIAN_TO		Преобразование из формата данных little endian A:= FROM_LITTLE_ENDIAN(B);
Типы данных на входной и выходной стороне должны иметь одинаковый тип данных. Примечание — В случае, если переменная уже находится в требуемом формате, функция не изменяет представления данных.			

6.6.2.5.13 Функции перечислимых типов данных

Функции выбора и сравнения, перечисленные в таблице 38 могут применяться к входным переменным, имеющим перечислимый тип данных.

Таблица 38 — Функции перечислимых типов данных

Номер	Описание/ имя функции	Символ	Номер свойства «x» в таблице «y»
1	SEL		Свойство 2, таблица 32
2	MUX		Свойство 6, таблица 32
3[a)]	EQ	=	Свойство 3, таблица 33
4[a)]	NE	<>	Свойство 6, таблица 33
Примечание — К данной таблице применяются положения примечаний 1 и 2 таблицы 33.			
Примечание — К данной таблице применяются положения подстрочных примечаний a) и b) таблицы 33.			

6.6.2.5.14 Функции подтверждения

Функции подтверждения проверяют, содержит ли заданный входной параметр допустимое значение.

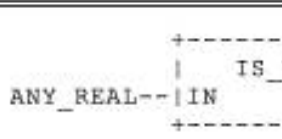
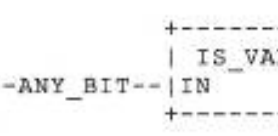
Для типов данных REAL и LREAL определена перегруженная функция IS_VALID. Функции подтверждения возвращает результат FALSE, если действительное число не является числом (NaN) или равно бесконечности (+Inf, -Inf).

Разработчик может поддерживать дополнительные типы данных посредством функции подтверждения IS_VALID. Результат таких расширений определяется разработчиком.

Перегруженная функция IS_VALID_BCD определена для типов данных BYTE, WORD, DWORD и LWORD. Функции подтверждения возвращает результат FALSE, если значение не удовлетворяет определению BCD.

Перечень свойств функций подтверждения приведен в таблице 39.

Таблица 39 — Функции подтверждения

Номер	Функция	Графическая форма	Пример
1	IS_VALID		Подтверждение значения типа REAL VAR R: REAL; END_VAR IF IS_VALID(R) THEN ...
2	IS_VALID_ BCD		Тест подтверждения слова BCD VAR W: WORD; END_VAR IF IS_VALID_BCD(W) THEN ...

6.6.3 Функциональные блоки

6.6.3.1 Общие положения

Функциональный блок — это программный компонент, который представляет хорошо определенную часть программы для обеспечения модульности и структуризации.

Концепция функционального блока реализуется типом функционального блока и экземпляром функционального блока:

- тип функционального блока состоит из следующих частей:
  - определение структуры данных, разделенной на входные, выходные и внутренние переменные;
  - набор операций, выполняемых с элементами структуры данных при вызове экземпляра типа функционального блока;
- экземпляр функционального блока:
  - это многократное, именованное применение (экземпляры) типа функционального блока;
  - каждый экземпляр имеет связанный идентификатор (имя экземпляра), и структуру данных, содержащую статические входные, выходные и внутренние переменные.

Статические переменные сохраняют свое значение от одного выполнения экземпляра функционального блока до следующего. Поэтому, вызов экземпляра функционального блока с одинаковыми входными параметрами не всегда выдает одинаковые выходные значения.

Если не оговорено иное, к функциональным блокам применяются общие свойства программных компонентов;

- объектно-ориентированный функциональный блок.

Функциональный блок может быть расширен набором объектно-ориентированных свойств.

Объектно-ориентированный функциональный блок является также расширенным множеством классов.

6.6.3.2 Объявление типа функционального блока

Тип функционального блока объявляется таким же образом, как и функции.

Свойства объявления типа функционального блока определены в таблице 40:

1) ключевое слово FUNCTION_BLOCK, за которым следует имя объявляемого функционального блока;

2) множество операций, составляющее тело функционального блока;

3) завершающее ключевое слово END_FUNCTION_BLOCK после тела функционального блока;

4) конструкции VAR_INPUT, VAR_OUTPUT и VAR_IN_OUT, при необходимости, определяющие имена и типы переменных;

5) значения переменных, которые объявляются через конструкцию VAR_EXTERNAL, могут изменяться из функционального блока;

6) значения констант, которые объявляются через конструкцию VAR_EXTERNAL CONSTANT и не могут изменяться из функционального блока;

7) массивы переменной длины могут использоваться как VAR_IN_OUT;

8) выходные и статические переменные могут инициализироваться;

9) переменные EN и ENO объявляются так же, как и входные и выходные переменные. Имеются специфические свойства функциональных блоков (отличные от свойств функций);

10) конструкция VAR...END_VAR и также конструкция VAR_TEMP...END_VAR, при необходимости, определяющие имена и типы внутренних переменных функциональных блоков. В отличие от функций, переменные, объявленные в секции VAR, являются статическими;

11) переменные секции VAR(статические) могут быть объявлены как PUBLIC или PRIVATE. По умолчанию используется спецификатор доступ PRIVATE. Переменные PUBLIC могут использоваться вне функционального блока, используя такой же синтаксис, как при доступе к выходным переменным функционального блока;

12) для входных, выходных и внутренних переменных функционального блока могут использоваться квалификаторы RETAIN или NON_RETAIN, как показано в таблице 40;

13) в текстовых объявлениях квалификаторы R_EDGE и F_EDGE используются для обозначения функции детектирования фронта сигнала логических входных переменных. Это приводит к неявному объявлению в данном функциональном блоке функционального блока типа R_TRIG или F_TRIG, соответственно, для выполнения обнаружения требуемого фронта. Пример такой конструкции приведен в таблице 40;

14) в графических объявлениях для детектирования задних и передних фронтов сигнала применяется конструкция, показанная в таблице. При использовании набора символов в графических объявлениях, символы «>» и «<» показываются на границе функционального блока;

15) в объявлении внутренних переменных функционального блока может использоваться символ «*», как определено в таблице 16;

16) если в объявлениях типов стандартных входных и выходных переменных функционального блока используются родовые типы данных, то правила определения фактических типов выходных параметров таких типов функциональных блоков являются частью определения типа функционального блока;

17) экземпляры других функциональных блоков, классов, объектно-ориентированных функциональных блоков могут объявляться во всех секциях переменных, за исключением секции VAR_TEMP;

18) экземпляр функционального блока, объявленный внутри типа функционального блока, не должен иметь, во избежание неопределенностей, такое же имя, как функция из той же области имен.

Таблица 40 — Объявление типа функционального блока

Номер	Описание	Пример
1	Объявление типа функционального блока FUNCTION_BLOCK ... END_FUNCTION_BLOCK	FUNCTION_BLOCK myFB ... END_FUNCTION_BLOCK
2a	Объявление входных переменных VAR_INPUT ...END_VAR	VAR_INPUT IN: BOOL; T1: TIME; END_VAR
2b	Объявление входных переменных VAR_OUTPUT ... END_VAR	VAR_OUTPUT OUT: BOOL; ET_OFF: TIME; END_VAR
2c	Объявление входных-выходных переменных VAR_IN_OUT ... END_VAR	VAR_IN_OUT A: INT; END_VAR
2d	Объявление временных переменных VAR_TEMP ... END_VAR	VAR_TEMP I: INT; END_VAR2e
	Объявление статических переменных VAR ... END_VAR	VAR B: REAL; END_VAR
2f	Объявление внешних переменных VAR_EXTERNAL ... END_VAR	VAR_EXTERNAL B: REAL; END_VAR Соответствует следующему: VAR_GLOBAL B: REAL

Продолжение таблицы 40

Номер	Описание	Пример
2g	Объявление внешних переменных VAR_EXTERNAL CONSTANT ... END_VAR	VAR_EXTERNAL CONSTANT B: REAL; END_VAR Соответствует следующему: VAR_GLOBAL B: REAL
3a	Инициализация входных параметров	VAR_INPUT MN: INT:= 0;
3b	Инициализация выходных параметров	VAR_OUTPUT RES: INT:= 1;
3c	Инициализация статических переменных	VAR B: REAL:= 12.1;
3d	Инициализация временных переменных	VAR_TEMP I: INT:= 1;
—	Входной параметр EN и выходной параметр ENO	Определено в таблице 18
4a	Объявление квалификатора RETAIN для входных переменных	VAR_INPUT RETAIN X: REAL; END_VAR
4b	Объявление квалификатора RETAIN для выходных переменных	REAL; END_VAR
4c	Объявление квалификатора RETAIN для выходных переменных	VAR_INPUT NON_RETAIN X: REAL; END_VAR
4d	Объявление квалификатора NON_RETAIN для выходных переменных	VAR_OUTPUT NON_RETAIN X: REAL; END_VAR
4e	Объявление квалификатора NON_RETAIN для статических переменных	VAR RETAIN X: REAL; END_VAR
4f	Объявление квалификатора NON_RETAIN для статических переменных	VAR NON_RETAIN X: REAL; END_VAR
5a	Объявление квалификатора RETAIN для локальных экземпляров функционального блока	VAR RETAIN TMR1: TON; END_VAR
5b	Объявление квалификатора NON_RETAIN для локальных экземпляров функционального блока	VAR NON_RETAIN TMR1: TON; END_VAR
6a	Текстовое объявление: - входных переменных переднего фронта	FUNCTION_BLOCK AND_EDGE VAR_INPUT X: BOOL R_EDGE; Y: BOOL F_EDGE; END_VAR VAR_OUTPUT Z: BOOL; END_VAR Z:= X AND Y; (*пример на языке ST *) END_FUNCTION_BLOCK
6b	- входных переменных заднего фронта (текстовое)	См. выше

Окончание таблицы 40

Номер	Описание	Пример
7a	Графическое объявление: - входных переменных переднего фронта (>)	FUNCTION_BLOCK (* Внешний интерфейс *)  (* тело функционального блока *)  END_FUNCTION_BLOCK
7b	Графическое объявление: - входных переменных заднего фронта (<)	См. выше
Примечание — Свойства 1—3 этой таблицы эквивалентны функциям, см. таблицу 19.		

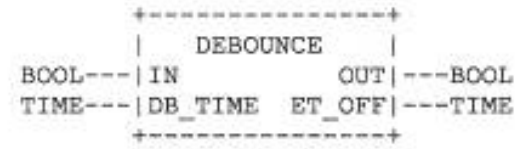
Ниже приведены примеры объявления типа FB.

Пример 1 — Объявление типа функционального блока

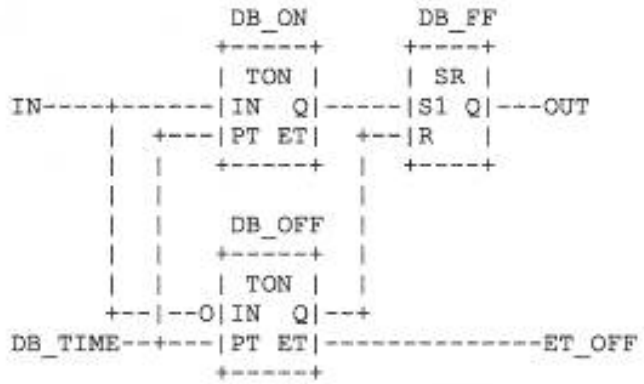
```
FUNCTION_BLOCK DEBOUNCE
(*** Внешний интерфейс ***)
VAR_INPUT
    IN: BOOL;    (* Неявно = 0 *)
    DB_TIME: TIME:= t#10ms;     (* Неявно = t#10ms *)
END_VAR

VAR_OUTPUT
    OUT: BOOL; (* Неявно = 0 *)
    ET_OFF: TIME;        (* Неявно = t#0s *)
END_VAR

VAR DB_ON: TON;        (** Внутренние переменные **)
    DB_OFF:     TON;    (**и экземпляры FB **)
    DB_FF:      SR;
END_VAR

(*** Тело функционального блока ***)
    DB_ON (IN:= IN, PT:= DB_TIME);
    DB_OFF(IN:= NOT IN, PT:= DB_TIME);
    DB_FF (S1:= DB_ON.Q, R:= DB_OFF.Q);
    OUT:= DB_FF.Q1;
    ET_OFF:= DB_OFF.ET;
END_FUNCTION_BLOCK
```

a) Текстовое объявление (язык ST)

```
FUNCTION_BLOCK
    (* Интерфейс внешних параметров *)
```




```
    (* Тело типа функционального блока *)
```

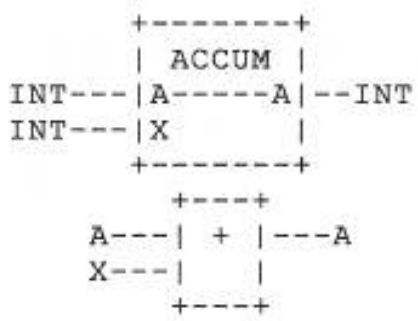



```
END_FUNCTION_BLOCK
```

b) Графическое объявление (язык FBD)

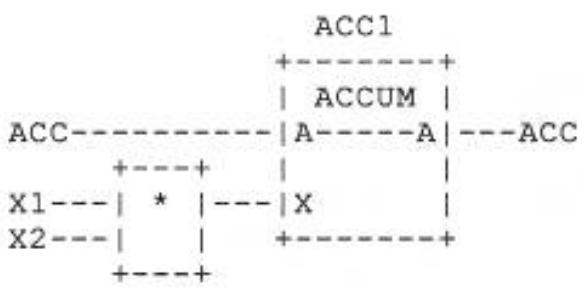
Пример ниже показывает объявление и графическое использование входных переменных в функциональном блоке, как задано в таблице 40.

Пример 2 –




```
FUNCTION_BLOCK ACCUM
VAR_IN_OUT A: INT; END_VAR
VAR_INPUT X: INT; END_VAR
A:= A+X;
END_FUNCTION_BLOCK
```

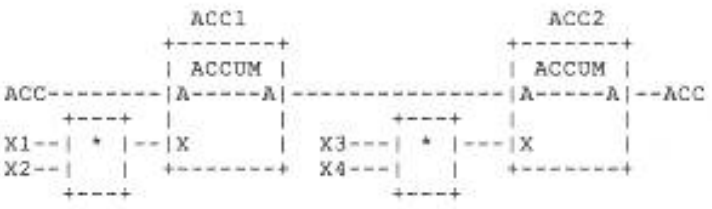
a) Графическое и текстовое объявление типа функционального блока и функции




```
VAR
    ACC: INT;
    X1: INT;
    X2: INT;
END_VAR
```

Это объявление предположительное:
эффект выполнения:
*ACC:= ACC+X1*X2;*

b) Допустимое использование экземпляра функционального блока и функции



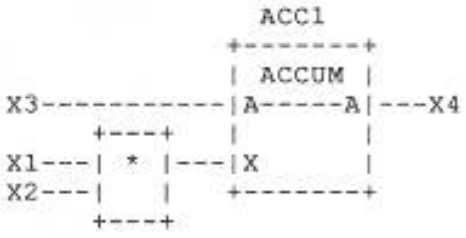
Объявления как в примере b) предполагаются для

ACC, X1, X2, X3 и X4;

эффект выполнения следующий:

*ACC:= ACC+X1*X2+X3*X4;*

c) Допустимое использование экземпляра функционального блока



VAR

X1: INT;

X2: INT;

X3: INT;

X4: INT;

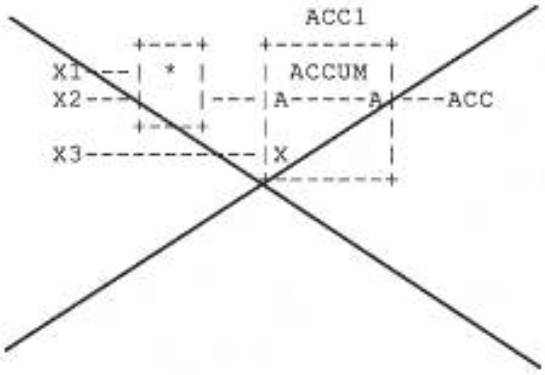
END_VAR

Объявление предположительное: эффект выполнения:

*X3:= X3+X1*X2;*

X4:= X3;

d) Допустимое использование экземпляра функционального блока



НЕДОПУСТИМО!
Соединение к входной-выходной переменной A не является переменной или именем функционального блока (см. предыдущий текст)

e) Неразрешенное использование экземпляра FB

Следующий пример показывает функциональный блок AND_EDGE, используемый в таблице 40.

Пример 3 — Объявление типа функционального блока AND_EDGE

Объявление функционального блока AND_EDGE на примере выше в таблице 40 эквивалентно следующему объявлению:

```
FUNCTION_BLOCK AND_EDGE
  VAR_INPUT
    X: BOOL;
    Y: BOOL;
  END_VAR
  VAR
    X_TRIG: R_TRIG;
    Y_TRIG: F_TRIG;
  END_VAR VAR_
  OUTPUT
    Z: BOOL;
  END_VAR
```

```
  X_TRIG(CLK:= X);
  Y_TRIG(CLK:= Y);
  Z:= X_TRIG.Q AND Y_TRIG.Q;
END_FUNCTION_BLOCK
```

Определение функциональных блоков обнаружения фронта R_TRIG и F_TRIG см. в таблице 44.

6.6.3.3 Объявление экземпляра функционального блока

Экземпляр функционального блока объявляется таким же образом, как и описанные структурные переменные.

Когда объявляется экземпляр функционального блока, начальные значения входных, выходных и общих переменных могут объявляться в перечне, заключенном в скобки, с последующим оператором присваивания, который следует за идентификатором типа функционального блока, как показано в таблице 41.

Элементы, для которых начальные значения не перечислены в описанном выше перечне инициализации, получают неявное начальное значение, объявленное для этих элементов в объявлении типа функционального блока.

Таблица 41 — Объявление экземпляра функционального блока

Номер	Описание	Пример
1	Объявление экземпляра функционального блока	VAR FB_instance_1, FB_instance_2: my FB_Type; T1, T2, T3: TON; END_VAR
2	Объявление экземпляра FB с инициализацией его переменных	VAR TempLoop: PID:=(PropBand:= 2.5, Integral:= T#5s); END_VAR Распределяет начальные значения входным и выходным переменным экземпляра функционального блока

6.6.3.4 Вызов функционального блока

6.6.3.4.1 Общие положения

Вызов экземпляра функционального блока может быть представлен в текстовой или графической форме.

Свойства вызова функционального блока (включая формальный и неформальный вызовы) похожи на свойства вызова функций со следующими расширениями:

1) текстуальный вызов функционального блока состоит из имени экземпляра с последующим перечнем параметров;

2) в графическом представлении имя экземпляра функционального блока располагается над блоком;

3) входные переменные и выходные переменные экземпляра функционального блока сохраняются и могут быть представлены как элементы структурированных типов данных. В связи с этим, присваивание входных переменных и доступ к выходным переменным могут осуществляться разными способами:

a) немедленно во время вызова функционального блока (типовой способ); или

b) отдельно от вызова. Такие отдельные присваивания становятся эффективными во время следующего вызова функционального блока;

c) неприсвоенные или несоединенные входные переменные функционального блока сохраняют свои инициализированные значения от последнего предыдущего вызова при наличии такового.

Возможна ситуация, когда не указано фактического параметра для входной-выходной переменной экземпляра функционального блока, используемой в качестве входной переменной другого экземпляра другого функционального блока. Однако экземпляру функционального блока будет предоставлено допустимое значение. Это может быть: значение, полученное инициализацией; сохраненное значение

предшествующего вызова; значение, ранее использовавшееся в функциональном блоке; значение, полученное методом. Если допустимое значение не будет получено, возникает ошибка времени выполнения.

К вызову функционального блока применяются следующие правила:

4) Если экземпляр функционального блока вызывается с входным параметром EN=0, разработчик определяет установлены ли в экземпляре входные и входные-выходные переменные.

5) Имя экземпляра функционального блока может использоваться в качестве входного параметра экземпляра функционального блока, если оно объявлено как входная переменная в секции VAR_INPUT или как входная-выходная переменная экземпляра функционального блока в секции VAR_IN_OUT.

6) Выходные значения экземпляра другого функционального блока, чье имя передается в функциональный блок через конструкцию VAR_INPUT, VAR_IN_OUT или VAR_EXTERNAL могут использоваться для доступа, но не могут изменяться из функционального блока.

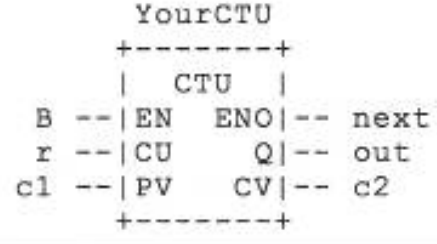
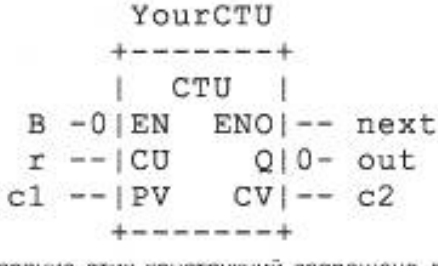
7) Функциональный блок, имя экземпляра которого передается в функциональный блок через конструкцию VAR_IN_OUT или VAR_EXTERNAL, может вызываться из функционального блока.

8) Через конструкцию VAR_IN_OUT в функциональный блок могут передаваться только переменные или имена экземпляров функциональных блоков.

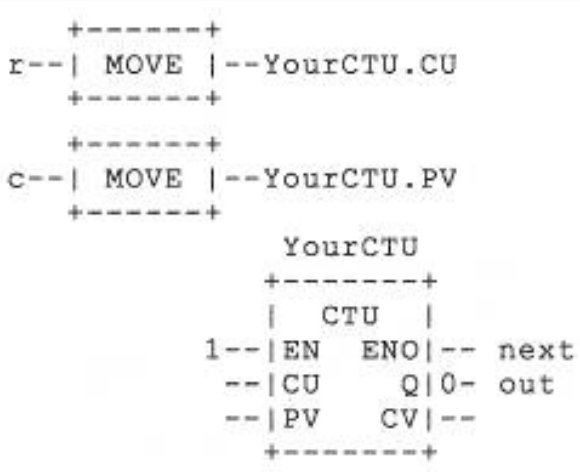
Это делается для предотвращения непреднамеренных изменений таких выходных переменных. Тем не менее, «каскадное» использование конструкций VAR_IN_OUT разрешено.

Свойства вызова функционального блока приведены в следующей таблице 42.

Таблица 42 — Вызов функционального блока

Номер	Описание	Пример
1	Полный формальный вызов (только текстовый) Используется, если указание параметров EN и ENO в вызове является обязательным	YourCTU(EN:= not B, CU:= r, PV:= c1, ENO=> next, Q => out, CV => c2);
2	Неполный формальный вызов (только текстовый)	YourCTU (Q => out, CV => c2); Переменные EN, CU, PV будут иметь значение последнего вызов или начальное значение, если FB не вызывался раньше
3	Графический вызов	
4	Графический вызов с отрицаниями логических входных и выходных переменных	 Использование этих конструкций запрещено для входных-выходных переменных

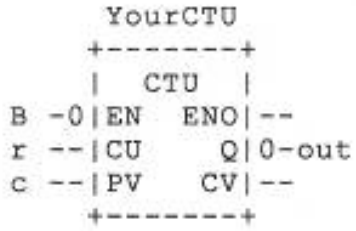
Продолжение таблицы 42

Номер	Описание	Пример
5a	Графический вызов с использованием конструкции VAR_IN_OUT	
5b	Графический вызов с присваиванием переменной из конструкции VAR_IN_OUT	
6a	Текстовый вызов с отдельным присваиванием входной переменной FB_Instance.Input:= x;	YourTon.IN:= r; YourTon.PT:= t; YourTon(not Q => out);
6b	Графический вызов с отдельным присваиванием входной переменной	
7	Чтение выходной переменной после вызова FB (текстовая форма) x:= FB_Instance.Output;	—
8a	Выходная переменная, присвоенная в вызове FB (текстовая форма)	—
8b	Выходная переменная, присвоенная в вызове FB с отрицанием (текстовая форма)	—
9a	Текстовый вызов с именем экземпляра функционального блока как входной переменной	VAR_INPUT I_TMR: TON; END_VAR EXPIRED:= I_TMR.Q; В данном и следующих примерах предполагается, что переменные EXPIRED и A_VAR были объявлены с типом BOOL
9b	Графический вызов с именем экземпляра функционального блока как входной переменной	См. а)
10a	Текстовый вызов с именем экземпляра функционального блока как переменной из VAR_IN_OUT	VAR_IN_OUT IO_TMR: TOF; END_VAR IO_TMR (IN:=A_VAR, PT:= T#10S); EXPIRED:= IO_TMR.Q;
10b	Графический вызов с именем экземпляра функционального блока как переменной из VAR_IN_OUT	—
11a	Текстовый вызов с именем экземпляра функционального блока как внешней переменной	VAR_EXTERNAL EX_TMR: TOF; END_VAR EX_TMR(IN:= A_VAR, PT:= T#10S); EXPIRED:= EX_TMR.Q;

Окончание таблицы 42

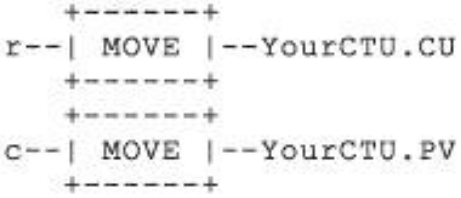
Номер	Описание	Пример
11b	Графический вызов с именем экземпляра функционального блока как внешней переменной	—

Пример — Вызов функционального блока с

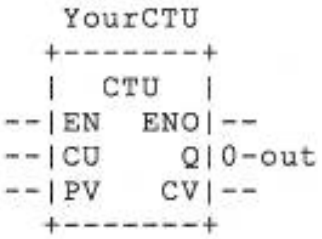



```
YourCTU (EN:= not b,
         CU:= r,
         PV:= c,
         not Q => out);
```

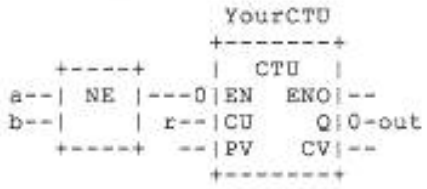
a) Вызов FB с немедленным присваиванием входных переменных (типичное использование)




```
YourCTU.CU:= r;
YourCTU.PV:= V;

YourCTU(not Q => out);
```



b) Вызов FB с отдельным присваиванием входной переменной



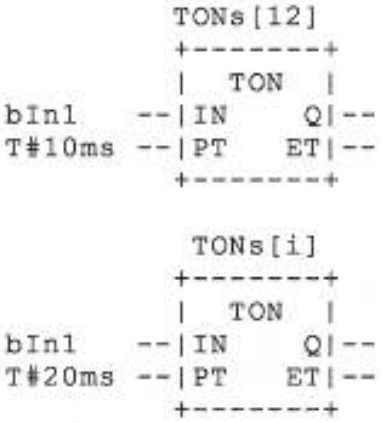

```
VAR a, b, r, out: BOOL;
    YourCTU: CTU; END_VAR
YourCTU (EN := NOT (a <> b),
         CU := r,
         NOT Q => out);
```

c) Вызов FB с немедленным доступом к выходной переменной (типовое использование)

В вызове также разрешено использование отрицания

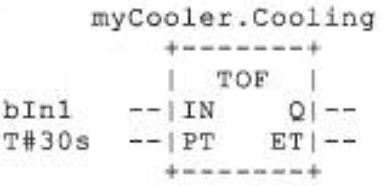

```
VAR FF75: SR; END_VAR (* Объявление *) FF75(S1:=
bIn1,       (* вызов *)
         R:= bIn2);
bOut3:= FF75.Q1;
```

d) Вызов FB с текстовым отдельным присваиванием выходной переменной (после вызова)




```
VAR
    TONs: array [0..100] OF TON;
    i: INT;
END_VAR

TON[12](IN:= bIn1, PT:= T#10ms);

TON[i](IN:= bIn1, PT:= T#20ms);
```

e) Вызов FB, используя массив экземпляров




```
TYPE
    Cooler: STRUCT
        Temp:Temp: INT;
        Cooling: TOF;
    END_STRUCT;
END_TYPE
VAR
    myCooler: Cooler;
END_VAR
    myCooler.Cooling(IN:= bIn1, PT:= T#30s);
```

f) Вызов FB с использованием экземпляра как элемента структуры

6.6.3.4.2 Использование входных и выходных параметров

На рисунках 13 и 14 приведена сводка правил использования входных и выходных параметров функционального блока в контексте вызова этого функционального блока. Присваивание входных и входных-выходных параметров становится эффективным при следующем вызове FB.

FUNCTION_BLOCK FB_TYPE;
VAR_INPUT In: REAL; END_VAR
VAR_OUTPUT Out: REAL; END_VAR
VAR_IN_OUT In_out: REAL; END_VAR
VAR M: REAL; END_VAR
END_FUNCTION-BLOCK
VAR FB_INST: FB_TYPE; A, B, C: REAL; END_VAR

Использование	a) Внутри функционального блока	b) Outside function block
1 Чтение входной переменной	M:= In;	~~A:= In~~ Не разрешено (см. примечания 1 и 2)
2 Присваивание входной переменной	~~In:= M;~~ Не разрешено (см. примечание 1)	// Вызов с немедленным присваиванием параметра FB_INST(In:= A); // Отдельное присваивание (см. примечание 4) FB_INST.In:= A;
3 Чтение выходной переменной	M:= Out;	// Вызов с немедленным присваиванием параметра FB_INST(Out => B); // Отдельное присваивание B:= FB_INST.Out;
4 Присваивание выходной переменной	Out:= M;	~~FB_INST.Out:= B;~~Не разрешено (см. примечание 1)
5 Чтение входной-выходной переменной	M:= In_out;	~~FB_INST(In_out=> C);~~ Не разрешено ~~C:= FB_INST.In_ou~~ Не разрешено
6 Присваивание входной-выходной переменной	In_out:= M; (см. примечание 3)	// Вызов с немедленным присваиванием параметра ~~FB_INST(In_out:= C); FB_INST.In_out:= C;~~ Не разрешено

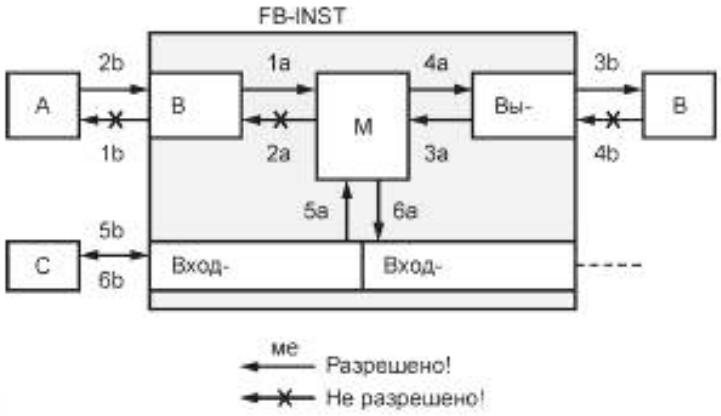
Примечание 1 — Использования, перечисленные в данной таблице с пометкой «Не разрешено», могут приводить к непредсказуемым побочным эффектам, определяемым разработчиком.

Примечание 2 — Чтение и запись (присваивание) входных и выходных параметров и внутренних переменных функционального блока могут выполняться «функцией взаимодействия», «функцией интерфейса оператора» или «функциями программирования, тестирования и мониторинга», определенными в МЭК 61131-1.

Примечание 3 — Изменение в функциональном блоке переменной, объявленной в секции VAR_IN_OUT, разрешено.

Рисунок 13 — Использование входных и выходных параметров функционального блока (правила)

Использование входных и выходных параметров, определенных правилами на рисунке 13, иллюстрируется на рисунке 14.



Метки 1a, 1b,... соответствуют правилам из рисунка 13.

Рисунок 14 — Использование входных и выходных параметров функционального блока (иллюстрация правил)

Следующие примеры демонстрируют графическое использование имен функциональных блоков в качестве параметров и внешних переменных.

Примеры — Графическое использование имен функциональных блоков в качестве параметров и внешних переменных.

FUNCTION_BLOCK

(* Внешний интерфейс *)

```
        +---------------+
        |   INSIDE_A    |
  TON---|I_TMR  EXPIRED|---BOOL
        +---------------+
```

(* Тело функционального блока *)

```
              +-------+
              | MOVE  |
  I_TMR.Q---  |       |---EXPIRED
              +-------+
```

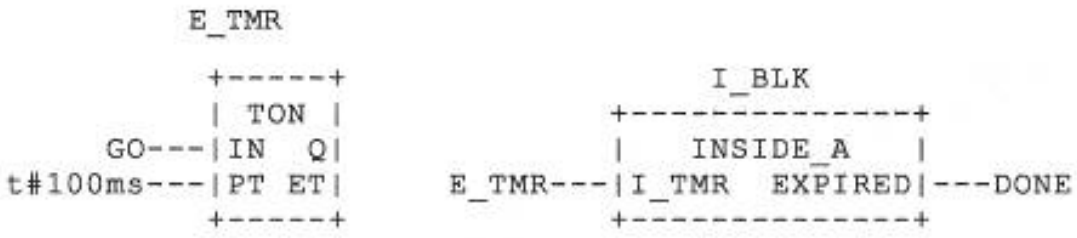
END_FUNCTION_BLOCK

FUNCTION_BLOCK

(* Внешний интерфейс *)

```
        +---------------+
        |   EXAMPLE_A   |
  BOOL---|GO          DONE|---BOOL
        +---------------+
```

(* Тело функционального блока *)

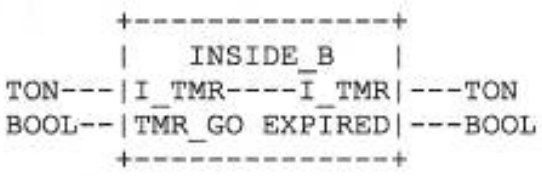


END_FUNCTION_BLOCK

a) Имя функционального блока как входная переменная (см. примечание)

FUNCTION_BLOCK

(Внешний интерфейс *)*

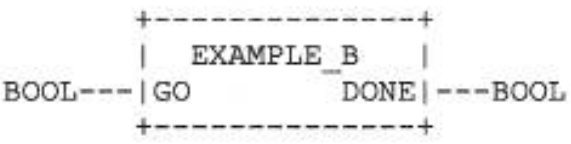


(Тело функционального блока *)*

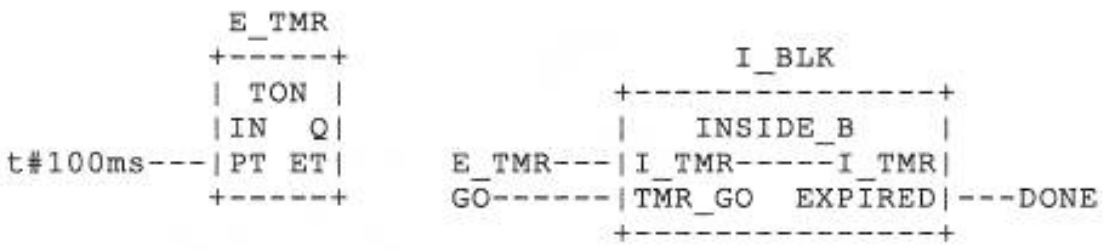
END_FUNCTION_BLOCK

FUNCTION_BLOCK

(Внешний интерфейс *)*



(Тело функционального блока *)*



END_FUNCTION_BLOCK

b) Имя функционального блока как входная-выходная переменная

```
FUNCTION_BLOCK
(* Внешний интерфейс *)

      +--------------+
      |   INSIDE_C   |
BOOL--|TMR_GO EXPIRED|---
      +--------------+

VAR_EXTERNAL X_TMR: TON; END_VAR
(* Тело функционального блока *)

             X_TMR
            +------+
            | TON  |
TMR_GO---   |IN   Q|---EXPIRED
            |PT  ET|
            +------+

END_FUNCTION_BLOCK
PROGRAM
(* Внешний интерфейс *)

        +--------------+
        |  EXAMPLE_C   |
BOOL---|GO          DONE|---BOOL
        +--------------+

VAR_GLOBAL X_TMR: TON; END_VAR
(* Тело программы *)

           I_BLK
      +---------------+
      |   INSIDE_C    |
GO---|TMR_GO  EXPIRED|---DONE
      +---------------+

END_PROыGRAM
```

c) Имя функционального блока как внешняя переменная

Примечание — I_TMR здесь не представлена графически, так как это будет предполагать вызов I_TMR внутри INSIDE_A, что запрещено правилами 3) и 4) на рисунке 13.

6.6.3.5 Стандартные функциональные блоки

6.6.3.5.1 Общие положения

Определения стандартных функциональных блоков, общие для всех языков программирования PLC, приведены ниже. Пользователь может предоставлять дополнительные стандартные функциональные блоки.

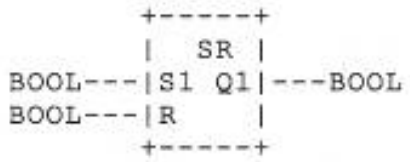
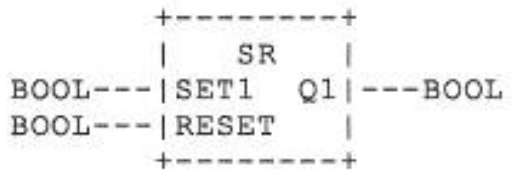
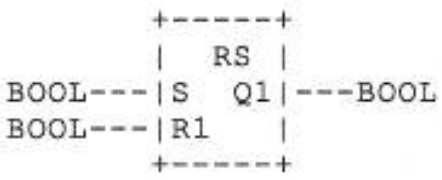
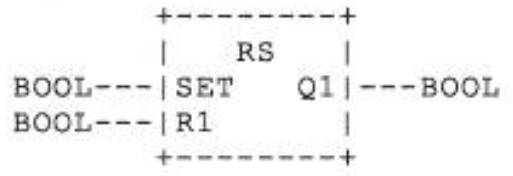
Там, где в данном разделе показываются стандартные функциональные блоки, могут быть также написаны эквивалентные текстовые объявления, как для примера в таблице 44.

Стандартные функциональные блоки могут быть перегружены и могут иметь расширяемые входные и выходные переменные. Определение таких типов функциональных блоков описывает все ограничения на число и типы данных таких входных и выходных переменных. Использование таких возможностей нестандартных функциональных блоков не входит в задачу данного стандарта

6.6.3.5.2 Бистабильные элементы

Графическая форма и тело функционального блока стандартных бистабильных элементов показаны в таблице 43.

Таблица 43 — Стандартные функциональные блоки с двумя устойчивыми состояниями[a)]

Номер	Описание/графическая форма	Тело функционального блока
1a	Бистабильный функциональный блок (доминанта включения): SR (S1,R,Q1)	
		
1b	Бистабильный функциональный блок (доминанта включения) с длинными именами входных параметров: SR (SET1, RESET, Q1)	
		
2a	Бистабильный функциональный блока (доминанта выключения): RS (S, R1, Q1)	
		
2b	Бистабильный функциональный блок (доминанта выключения) с длинными именами входных параметров: RS (SET, RESET1, Q1)	
		

[a)] Начальным состоянием выходной переменной Q1 является нормальное неявное значение 0 для логических переменных.

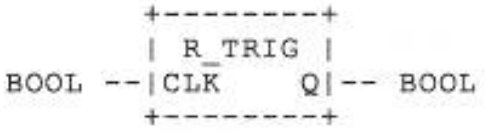
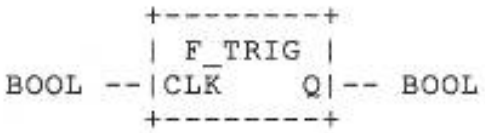
6.6.3.5.3 Определение фронта (R_TRIG и F_TRIG)

Графическое представление стандартного функционального блока обнаружения переднего и заднего фронта сигнала представлено в таблице 44. Поведение этих блоков эквивалентно определениям, данных в данной таблице. Данное поведение соответствует следующим правилам:

1 Выходная переменная Q функционального блока R_TRIG остается в значении BOOL#1 от одного вычисления вычислениями функционального блока до другого, отслеживая переход 0 к 1 входной переменной CLK, и возвращается в 0 при следующем выполнении.

Выходная переменная Q функционального блока F_TRIG остается в значении BOOL#1 от одного вычисления вычислениями функционального блока до другого, отслеживая переход 1 к 0 входной переменной CLK, и возвращается в 0 при следующем выполнении.

Таблица 44 — Стандартный функциональный блок обнаружения фронта

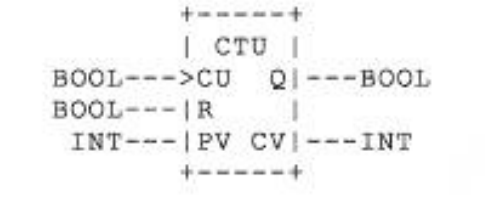
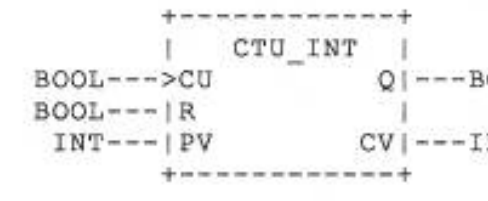
Номер	Описание/графическая форма	Определение (на языке ST)
1	Детектор переднего фронта: R_TRIG(CLK, Q)	
		FUNCTION_BLOCK R_TRIG VAR_INPUT CLK: BOOL; END_VAR VAR_OUTPUT Q: BOOL; END_VAR VAR M: BOOL; END_VAR Q:= CLK AND NOT M; M:= CLK; END_FUNCTION_BLOCK
2	Детектор заднего фронта: F_TRIG(CLK, Q)	
		FUNCTION_BLOCK F_TRIG VAR_INPUT CLK: BOOL; END_VAR VAR_OUTPUT Q: BOOL; END_VAR VAR M: BOOL; END_VAR Q:= NOT CLK AND NOT M; M:= NOT CLK; END_FUNCTION_BLOCK

Примечание — Когда входная переменная CLK экземпляра типа R_TRIG соединяется со значением BOOL#1, его выходная переменная Q сохраняет значение BOOL#1 после первого выполнения, следующего за «холодным рестартом». Выходная переменная Q сохраняет значение BOOL#0 после всех следующих выполнений. Это же применимо к экземпляру F_TRIG , входная переменная CLK которого отсоединяется или соединяется к значению FALSE.

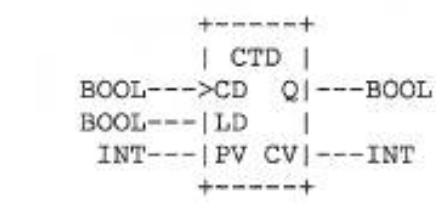
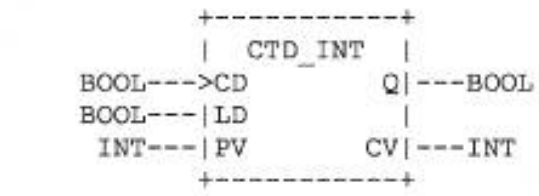
6.6.3.5.4 Счетчики

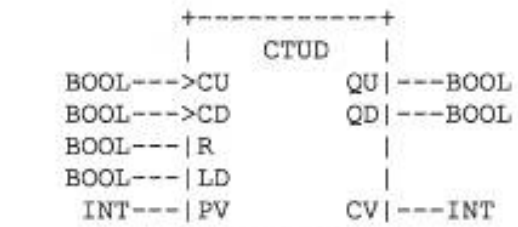
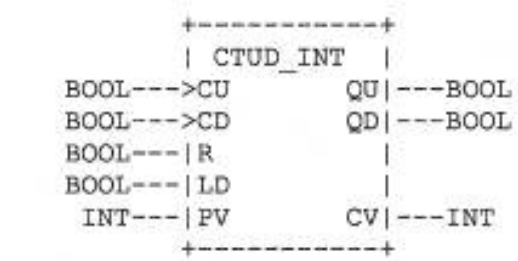
Графическое представление стандартных функциональных блоков счетчика с типами связанных входных и выходных переменных представлено в таблице 45. Функционирование этих функциональных блоков определяется в телах соответствующих функциональных блоков.

Таблица 45 — Стандартные функциональные блоки счетчиков

Номер	Описание/графическая форма	Тело функционального блока (язык ST)
	Возрастающий счетчик	
1a	CTU_INT(CU, R, PV, Q, CV) or CTU(..)	
	 а также: 	VAR_INPUT CU: BOOL R_EDGE; ... /* Фронт вычисляется внутри, используя тип данных R_EDGE */ IF R THEN CV:= 0; ELSIF CU AND (CV < PVmax) THEN CV:= CV+1; END_IF; Q:= (CV >= PV);

Продолжение таблицы 45

Номер	Описание/графическая форма	Тело функционального блока (язык ST)
1b	CTU_DINT PV, CV: DINT	см. 1a
1c	CTU_LINT PV, CV: LINT	см. 1a
1d	CTU_UDINT PV, CV: UDINT	см. 1a
1e	CTU_ULINT(CD, LD, PV, CV) PV, CV: ULINT	см. 1a
	Убывающие счетчики	
2a	CTD_INT(CD, LD, PV, Q, CV) or CTD	
	 а также: 	VAR_INPUT CU: BOOL R_EDGE; ... // Фронт вычисляется внутри типом данных R_EDGE IF LD THEN CV:= PV; ELSIF CD AND (CV > PVmin) THEN CV:= CV-1; END_IF; Q:= (CV <= 0);
2b	CTD_DINT PV, CV: DINT	См. 2a
2c	CTD_LINT PV, CV: LINT	
2d	CTD_UDINT PV, CV: UDINT	См. 2a
2e	CTD_ULINT PV, CV: UDINT	См. 2a
	Реверсивные счетчики	
3a	CTUD_INT(CD, LD, PV, Q, CV) or CTUD(..)	

	
	а также:
	 
		VAR_INPUT CU, CD: BOOL R_EDGE; ...
		// Фронт вычисляется внутри типом данных R_EDGE
		IF R
		THEN CV:= 0;
		ELSIF LD
		THEN CV:= PV;
		ELSE
		IF NOT (CU AND CD) THEN
		IF CU AND (CV < PVmax)
		THEN CV:= CV+1;
		ELSIF CD AND (CV > PVmin)
		THEN CV:= CV-1;
		END_IF;
		END_IF;
		END_IF;
		QU:= (CV >= PV);
		QD:= (CV <= 0);

Окончание таблицы 45

Номер	Описание/графическая форма	Тело функционального блока (язык ST)
3b	CTUD_DINT PV, CV: DINT	См. 3a
3c	CTUD_LINT PV, CV: LINT	См. 3a
3d	CTUD_UDINT PV, CV: UDINT	См. 3a
3e	CTUD_ULINT PV, CV: ULINT	См. 3a
Примечание — Числовые значения переменных предела PVmin и PVmax определяются разработчиком.		

6.6.3.5.5 Таймеры

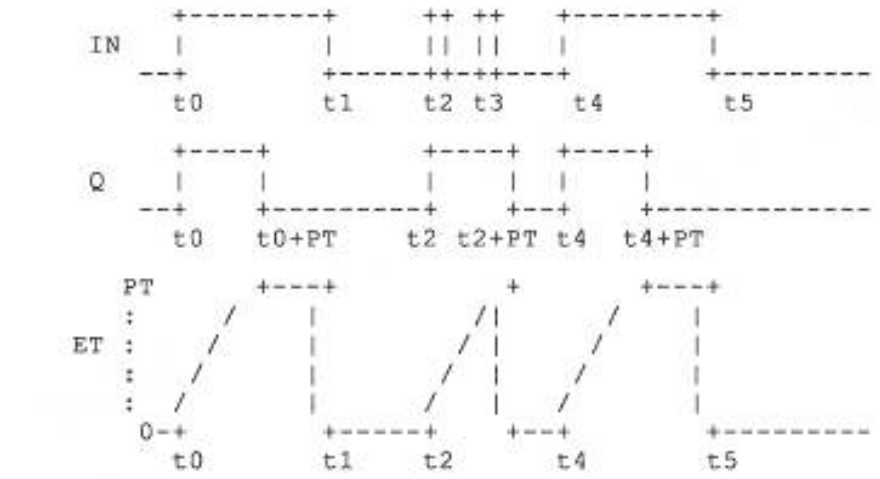
Графическая форма стандартных функциональных блоков таймера показана в таблице 46. Функционирование этих функциональных блоков определено на временной диаграмме, приведенной на рисунке 15.

Стандартные функциональные блоки счетчика могут быть перегружены типами данных TIME или LTIME, или базовый тип данных для стандартного таймера может быть определен как TIME или LTIME.

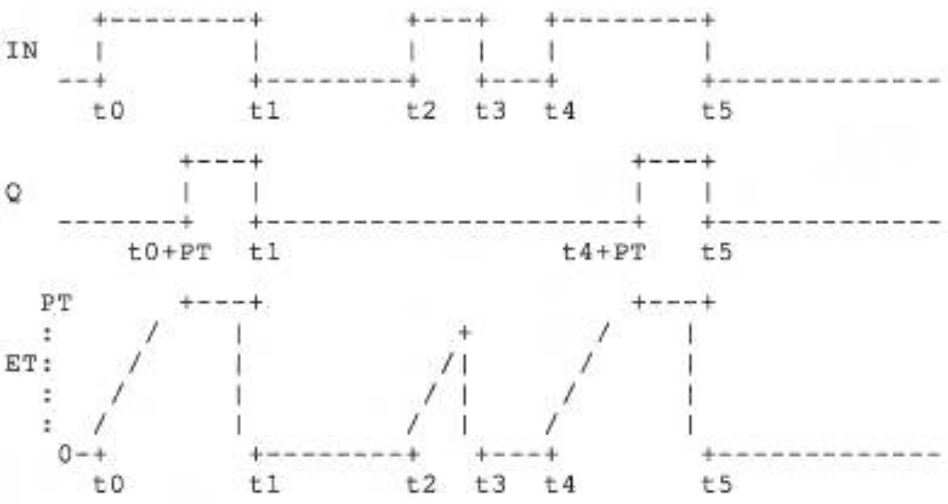
Таблица 46 — Стандартные функциональные блоки таймера

Номер	Описание		Символ	Графическая форма
1a	Импульсный таймер, перегруженный	TP	*** эквивалентно: TP	
1b	Импульсный таймер с типом данных TIME		TP_TIME	
1c	Импульсный таймер с типом данных LTIME		TP_LTIME	
2a	Таймер с задержкой включения, перегруженный	TON	TON	PT см. примечание
2b	Таймер с задержкой включения с типом данных TIME		TON_TIME	
2c	Таймер с задержкой включения с типом данных LTIME		TON_LTIME	IN: Input (Start)
2d[a)]	Таймер с задержкой включения, перегруженный	(графическая форма)	T---0	PT: Установленное время
3a	Таймер с задержкой отключения, перегруженный	TOF	TOF	Q: Выходная переменная
3b	Таймер с задержкой отключения с типом данных TIME		TOF_TIME	ET: Истекшее время
3c	Таймер с задержкой отключения с типом данных LTIME		TOF_LTIME	
3d[a)]	Таймер с задержкой отключения, перегруженный	(графическая форма)	0---T	
Примечание — Воздействие изменения значения входной переменной PT во время работы таймера, например, установка PT в t#0s при возобновлении функционирования экземпляра TP определяется параметром, задаваемым разработчиком.				
[a)] В текстовых языках свойства 2b и 3b не используются.				

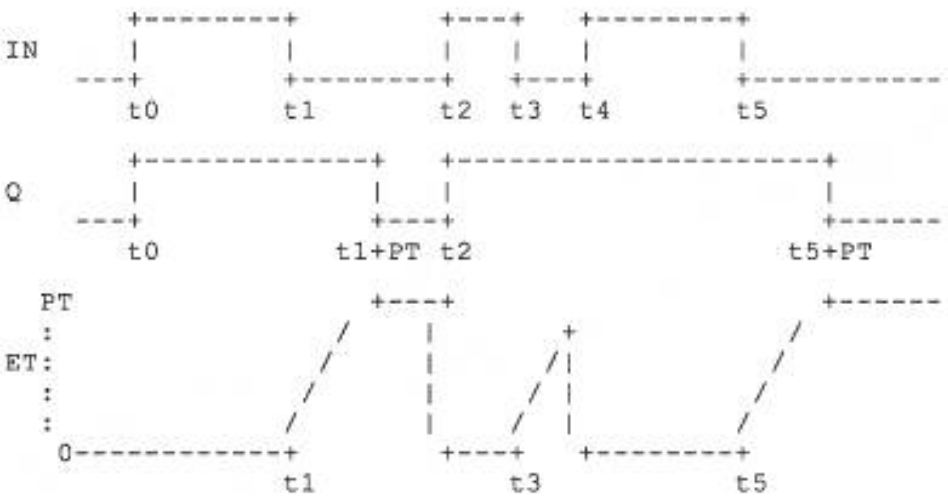
На рисунке 15 показана временная диаграмм стандартных функциональных блоков таймера.



a) Импульсный отсчет времени (PT)



b) Времени с задержкой включения (TON)



c) Расчет времени с задержкой отключения (TOF)

Рисунок 15 — Стандартные функциональные блока таймера — временные диаграммы (правила)

6.6.3.5.6 Функциональные блоки взаимодействия

Стандартные функциональные блоки взаимодействия для программируемых контроллеров определены в МЭК 61131-5. Данные функциональные блоки предоставляют функциональность взаимодействия, такую как средства проверки устройств, сбор данных опроса, запрограммированный сбор данных, управление параметрами, управление с взаимоблокировкой, запрограммированное аварийное оповещение, управление и защита соединений.

6.6.4 Программы

В МЭК 61131-1 программа определяется как «логический набор всех элементов и конструкций языка программирования, необходимый для запланированной обработки сигналов, требуемой для управления оборудованием или процессом системой PLC».

Объявление и использование программ идентично объявлению и использованию функциональных блоков с дополнительными свойствами, показанными в таблице 47, и со следующими отличиями:

1) ограничивающими ключевыми словами для программы являются PROGRAM...END_PROGRAM;

2) программа содержит конструкцию VAR_ACCESS...END_VAR, которые предоставляют средства определения именованных переменных, к которым может осуществляться доступ некоторыми службами связи, указанными в МЭК 61131-5. Путь доступа связывает каждую такую переменную с входными, выходными или внутренними переменными программы;

3) программы могут устанавливаться только в ресурсах, в то время как функциональные блоки могут устанавливаться в программах или других функциональных блоках;

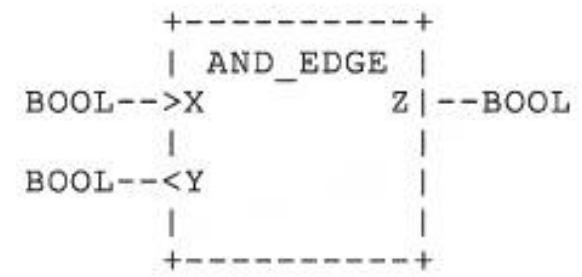
4) программа может содержать назначение своего расположения в объявлениях своих глобальных и внутренних переменных. Назначение расположения с частично определенным прямым представлением может использоваться только в объявлениях внутренних переменных программы;

5) объектно-ориентированные свойства программ не входят в задачу настоящего стандарта.

Таблица 47 — Объявление программы

Номер	Описание	Пример
1	Объявление программы PROGRAM ... END_PROGRAM	PROGRAM myPrg ... END_PROGRAM
2a	Объявление входных переменных VAR_INPUT ... END_VAR	VAR_INPUT IN: BOOL; T1: TIME; END_VAR
2b	Объявление выходных переменных VAR_OUTPUT ... END_VAR	VAR_OUTPUT OUT: BOOL; ET_OFF: TIME; END_VAR
2c	Объявление входных-выходных переменных VAR_IN_OUT ... END_VAR	VAR_IN_OUT A: INT; END_VAR
2d	Объявление временных переменных VAR_TEMP ... END_VAR	VAR_TEMP I: INT; END_VAR
2e	Определение статических переменных VAR ... END_VAR	VAR B: REAL; END_VAR
2f	Объявление внешних переменных END_VAR	VAR_EXTERNAL B: REAL; END_VAR Соответствует следующему: VAR_GLOBAL B: REAL
2g	Объявление внешних переменных VAR_EXTERNAL CONSTANT ... END_VAR	VAR_EXTERNAL CONSTANT B: REAL; END_VAR Соответствует следующему: VAR_GLOBAL B: REAL
3a	Инициализация входных параметров	VAR_INPUT MN: INT:= 0;
3b	Инициализация выходных параметров	VAR_OUTPUT RES: INT:= 1;
3c	Инициализация статических переменных	VAR B: REAL:= 12.1;
3d	Инициализация временных переменных	VAR_TEMP I: INT:= 1;

Продолжение таблицы 47

Номер	Описание	Пример
4a	Объявление квалификатора RETAIN для входных переменных	VAR_INPUT RETAIN X: REAL; END_VAR
4b	Объявление квалификатора RETAIN для выходных переменных	VAR_OUTPUT RETAIN X: REAL; END_VAR
4c	Объявление квалификатора NON_RETAIN для входных переменных	VAR_INPUT NON_RETAIN X: REAL; END_VAR
4d	Объявление квалификатора NON_RETAIN для выходных переменных	VAR_OUTPUT NON_RETAIN X: REAL; END_VAR
4e	Объявление квалификатора NON_RETAIN для статических переменных	REAL; END_VAR
4f	Объявление квалификатора NON_RETAIN для статических переменных	VAR NON_RETAIN X: REAL; END_VAR
5a	Объявление квалификатора RETAIN для локальных экземпляров функционального блока	VAR RETAIN TMR1: TON; END_VAR
5b	Объявление квалификатора NON_RETAIN для локальных экземпляров FB	VAR NON_RETAIN TMR1: TON; END_VAR
6a	Текстовое объявление - входных переменных переднего фронта	PROGRAM AND_EDGE VAR_INPUT X: BOOL R_EDGE; Y: BOOL F_EDGE; END_VAR VAR_OUTPUT Z: BOOL; END_VAR Z:= X AND Y; (* Пример на языке ST *) END_PROGRAM
6b	Текстовое объявление - входных переменных заднего фронта (текстовое)	См. выше
7a	Графическое объявление - входных переменных переднего фронта (>)	ПРОГРАММА (* Внешний интерфейс *)  (* тело функционального блока *)  END_PROGRAM

Окончание таблицы 47

Номер	Описание	Пример
7b	Графическое объявление - входных переменных заднего фронта (<)	См. выше
8a	Объявление VAR_GLOBAL...END_VAR в программе PROGRAM	VAR_GLOBAL z1: BYTE; END_VAR
8b	VAR_GLOBAL CONSTANT объявления в объявлениях типов программы ПРОГРАМ-МЫ	VAR_GLOBAL CONSTANT z2: BYTE; END_VAR
9	Объявление VAR_ACCESS...END_VAR в программе PROGRAM	VAR_ACCESS ABLE: STATION_1.%IX1.1: BOOL READ_ONLY; BAKER: STATION_1.P1.x2: UINT READ_WRITE; END_VAR
Примечание — Свойства от 2a до 7b эквивалентны соответствующим свойствам таблицы 40 для функциональных блоков.		

6.6.5 Классы

6.6.5.1 Общие положения

Элемент языка класс поддерживает объектно-ориентированную парадигму и характеризуется следующими принципами:

- определение структуры данных, разделенной на общие и внутренние переменные;
- выполняемые над элементами структуры данных;
- классы, состоящие из методов (алгоритмов) и структур данных;
- интерфейс с прототипами метода и реализация интерфейсов;
- наследование интерфейсов и классов.

Инстанцирование классов.

Примечание термины «класс» и «объект», используемые в языках C#, C++, Java, UML и т.д., соответствуют терминам «тип» и «экземпляр» языков программирования PLC из данного стандарта. Это показано ниже.

Языки программирования ИТ: C#, C++, Java, UML	**Языки PLC из стандарта**
Class (= тип класса)	Typ (тип функционального блока или класса)
Object (= экземпляр класса)	Instance (экземпляр функционального блока или класса)

Наследование интерфейса и классов с использованием механизмов реализации и расширения показано на рисунке 16. Это определено в 6.6.5.

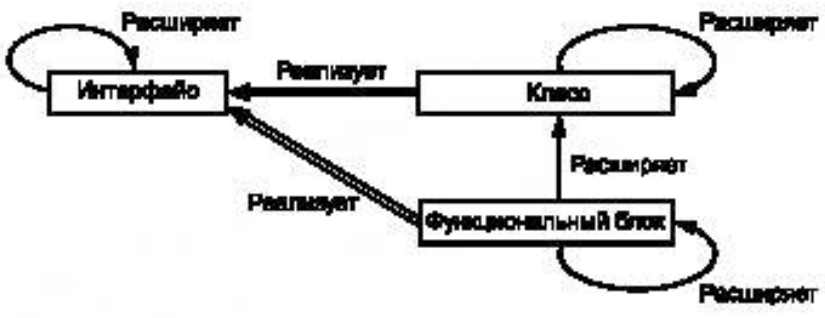


Рисунок 16 — Обзор наследования и реализации интерфейса

Класс — это программный компонент, разработанный для объектно-ориентированного программирования. По существу, класс содержит переменные и методы. Класс должен инстанцироваться до того, как его методы смогут вызываться и как можно осуществлять доступ к его переменным.

6.6.5.2 Объявление класса

Свойства объявления класса определены в таблице 48:

1) ключевое слово CLASS с последующим идентификатором, указывающим имя определяемого класса;

2) завершающее ключевое слово END_CLASS;

3) значения переменных, которые объявлены через конструкцию VAR_EXTERNAL, могут изменяться из класса;

4) Значения констант, которые объявлены через конструкцию VAR_EXTERNAL CONSTANT, не могут изменяться из класса;

5) конструкция VAR...END_VAR, при необходимости, указывающая имена и типы переменных класса;

6) переменные могут быть инициализированы;

7) переменные секции VAR (статические) могут быть объявлены как PUBLIC (общие). К общим переменным можно получать доступ извне класса, используя такой же синтаксис, как для доступа к выходным переменным функционального блока;

8) для внутренних переменных класса могут использоваться квалификаторы RETAIN и NON_RETAIN;

9) для объявления внутренних переменных класса может использоваться символ «*», как определено в таблице 16;

10) переменные могут быть общими PUBLIC, индивидуальными PRIVATE, внутренними INTERNAL или защищенными PROTECTED. По умолчанию используется спецификатор доступа PROTECTED;

11) класс может поддерживать наследование других классов для расширения базового класса;

12) класс может реализовывать один или более интерфейсов;

13) экземпляры других функциональных блоков, классы и блоки объектно-ориентированных функций могут быть объявлены в секциях переменных VAR и VAR_EXTERNAL;

14) экземпляр класса, объявленный внутри класса, не обязан использовать то же имя, как функция (той же области видимости) для предотвращения неопределенностей.

Класс имеет следующие различия от функционального блока:

- ключевые слова FUNCTION_BLOCK и END_FUNCTION_BLOCK заменены ключевыми словами CLASS и END_CLASS, соответственно;
- переменные объявляются только в секции VAR. Не разрешены секции VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT и VAR_TEMP. У класса нет тела;
- класс может определять только методы;
- вызов экземпляра класса невозможен. Могут вызываться только методы класса.

Реализация классов предоставляет по существу согласующееся подмножество свойств, определенных в таблице 48.

Таблица 48 — Класс

Номер	Описание Ключевое слово	Объяснение
1	CLASS ... END_CLASS	Определение класса
1a	Спецификатор FINAL	Класс не может использоваться в базовом классе
	Основанные на функциональном блоке	
2a	Определение переменных VAR ... END_VAR	VAR B: REAL; END_VAR
2b	Инициализация переменных	VAR B: REAL:= 12.1; END_VAR
3a	Квалификатор RETAIN для внутренних переменных	VAR RETAIN X: REAL; END_VAR

Окончание таблицы 48

Номер	Описание Ключевое слово	Объяснение
3b	Квалификатор RETAIN для внутренних переменных	VAR NON_RETAIN X: REAL; END_VAR
4a	Объявления VAR_EXTERNAL внутри объявлений типа класса	См. эквивалентный пример в таблице 40
4b	Объявления VAR_EXTERNAL CONSTANT в объявлениях типа класса	См. эквивалентный пример в таблице 40
	Методы и спецификаторы	
5	METHOD...END_METHOD	Определение метода
5a	Спецификатор PUBLIC	Метод может вызываться откуда угодно
5b	Спецификатор PRIVATE	Метод может вызываться только внутри определяющего программного компонента
5c	Спецификатор INTERNAL	Метод может вызываться из одного пространства имен
5d	Спецификатор PROTECTED	Метод может вызываться только из определяющего программного компонента и его наследников (неявно)
5e	Спецификатор FINAL	Метод не может быть перегружен
	Наследование	- данные свойства — такие же как в таблице 53 свойств наследования
6	EXTENDS	Класс является наследником класса Примечание — Наследование функциональных блоков отсутствует
7	OVERRIDE	Метод переопределяет базовый метод — см. динамическое связывание имен
8	ABSTRACT	Абстрактный класс — по меньшей мере, один метод является абстрактным Абстрактный метод — шаблон метода
	Ссылка на доступ	
9a	THIS	Ссылка на собственные методы
9b	SUPER	Ссылка на метод базового класса
	Спецификаторы доступа переменной	
10a	Спецификатор PUBLIC	Доступ к переменной возможен из любого места
10b	Спецификатор PRIVATE	Доступ к переменной осуществляется только внутри определяющего программного компонента
10c	Спецификатор INTERNAL	Доступ к переменной осуществляется только внутри одного пространства имен
10d	Спецификатор PROTECTED	Доступ к переменной осуществляется только из определяющего программного компонента и его наследников (неявно)
	Полиморфизм	
11a	с секцией переменных VAR_IN_OUT	Переменным из секции VAR_IN_OUT может быть присвоен экземпляр производного класса
11b	со ссылкой	Ссылке на (базовый) класс может быть присвоен адрес экземпляра производного класса

Пример ниже иллюстрирует свойства объявления класса и его использование.

Пример — Определение класса

```
Class CCounter
  VAR
    m_iCurrentValue: INT;        (* Default = 0 *)
    m_bCountUp: BOOL:=TRUE;
  END_VAR
  VAR PUBLIC
    m_iUpperLimit: INT:=+10000;
    m_iLowerLimit: INT:=-10000;
  END_VAR
METHOD Count (* Только тело *)
    IF (m_bCountUp AND m_iCurrentValue<m_iUpperLimit) THEN
      m_iCurrentValue:= m_iCurrentValue+1;
    END_IF;
    IF (NOT m_bCountUp AND m_iCurrentValue>m_iLowerLimit) THEN
      m_iCurrentValue:= m_iCurrentValue-1;
    END_IF;
END_METHOD

METHOD SetDirection
  VAR_INPUT
    bCountUp: BOOL;
  END_VAR
    m_bCountUp:=bCountUp;
END_METHOD
END_CLASS
```

6.6.5.3 Декларация экземпляра класса

Экземпляр класса объявляется подобно определению структурной переменной.

Когда объявляется экземпляр класса, начальные значения общих переменных могут присваиваться в перечне инициализации, заключенном в скобки, с последующим оператором присваивания, который следует за идентификатором класса, как показано в таблице 49.

Элементы, которым не присвоено значение в перечне инициализации, получают начальные значения из объявления класса.

Таблица 49 — Декларация экземпляра класса

Номер	Описание	Пример
1	Объявление экземпляра класса с неявной инициализацией	VAR MyCounter1: CCounter; END_VAR
2	Декларация экземпляра класса с инициализацией его общих переменных	VAR MyCounter2: CCounter:= (m_iUpperLimit:=20000; m_iLowerLimit:=-20000); END_VAR

6.6.5.4 Методы класса

6.6.5.4.1 Общие положения

Применительно к задачам языков программируемых контроллеров, концепция методов, хорошо известных в объектно-ориентированном программировании, принимается как набор факультативных элементов языка, используемых при определении класса.

Методы могут применяться для определения операций с данными экземпляров класса.

6.6.5.4.2 Сигнатура

В целях данного стандарта, термин сигнатура определен в разделе 3 как набор информации, однозначно определяющий идентичность параметров МЕТОДА.

Сигнатура включает:

- имя метода;
- тип результата;
- имена переменных, типы данных и порядок всех параметров,то есть входных, выходных и входных-выходных переменных.

Локальные переменные не являются частью сигнатуры. Переменные, объявленные в секции VAR_EXTERNAL и постоянные переменные не существенны для сигнатуры.

Спецификаторы доступа, такие как PUBLIC или PRIVATE не существенны для сигнатуры.

6.6.5.4.3 Объявление и выполнение метода

Класс может иметь набор методов.

Объявление метода должно подчиняться следующим правилам:

1 Методы объявляются в области действия класса.

2 Метод может объявляться на любом из языков, указанных в этом стандарте.

3 В текстовом объявлении методы перечисляются после объявления переменных класса.

4 Метод может объявлять свои собственные VAR_INPUT, внутренние временные переменные VAR и VAR_TEMP, VAR_OUTPUT, VAR_IN_OUT и результат метода.

Ключевые слова VAR_TEMP и VAR имеют то же самое значение и оба являются разрешенными для внутренних переменных.

(Ключевое слово VAR используется в функциях).

5 Объявление метода содержит один из следующих спецификаторов доступа: PUBLIC, PRIVATE, INTERNAL или PROTECTED. Если спецификатор доступа на задан, метод будет PROTECTED по умолчанию.

6 Объявление метода может содержать дополнительные ключевые слова OVERRIDE или ABSTRACT.

Примечание 1 — Перегрузка методов не входит в задачу настоящего стандарта.

Объявление метода должно подчиняться следующим правилам:

7 Во время выполнения метод может читать свои входные переменные и вычисляет выходные переменные и результат, используя временные переменные.

8 Результат метода присваивается его имени.

9 Все переменные метода и его результат являются временными (как переменные функции), то есть переменные не сохраняются от одного вычисления метода до другого.

Поэтому, вычисление выходных переменных метода возможно только в непосредственном контексте вызова метода.

10 Имена переменных каждого метода класса должны быть различными (уникальными). Имена локальных переменных различных методов могут быть одинаковыми.

11 Все методы класса имеют доступ для чтения/записи к статическим и внешним переменным, объявленным в классе.

12 Все переменные и результаты могут быть многозначными, например, массив или структура. Как это объявлено для функций, результат метода может использоваться как операнд в выражении.

13 Во время выполнения метод может использовать другие методы, определенные в этом классе. Методы экземпляра данного класса вызываются, используя ключевое слово THIS.

Следующий пример иллюстрирует упрощенное объявление класса с двумя методами и вызов метода.

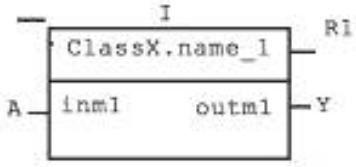
Пример 1

Примечание 2 — Алгоритм методов имеет доступ к их собственным данным и к данным класса.

(Временные параметры заключены в скобки)

Определение класса (типа) с методами:

```
CLASS name
  VAR vars; END_VAR
  VAR_EXTERNAL externals;
  END_VAR

METHOD name_1
  VAR_INPUT inputs;
END_VAR
  VAR_OUTPUT outputs;
END_VAR
END_METHOD

METHOD name_i
  VAR_INPUT inputs;
END_VAR
  VAR_OUTPUT outputs;
END_VAR
END_METHOD END_CLASS
```



Примечание 3 — Данное графическое представление метода служит только для иллюстрации

Вызов метода:

a) Использование результата (результат является необязательным)

```
R1:= I.method1(inm1:= A, outm1 => Y);
```

b) Использование вызова (результат не объявлен)

```
I.method1(inm1:= A, outm1 => Y);
```

Присваивание переменным метода за пределами метода

~~I.inm1 := A;~~ // Не разрешено;

Чтение выходных переменных метода за пределами метода

~~Y:= I.outm1;~~ // Не разрешено

Пример 2

Класс COUNTER с двумя методами для прямого счета метод UP5 показывает, как вызвать метод из собственного класса.

```
CLASS COUNTER
VAR
  CV: UINT;                          // Текущее значение счетчика
  Max: UINT:= 1000;
END_VAR

METHOD PUBLIC UP: UINT               / Метод прямого счета, используя inc
VAR_INPUT INC: UINT; END_VAR         // Приращение
VAR_OUTPUT QU: BOOL; END_VAR         // Обнаружение верхнего предела
  IF CV <= Max - INC
```

```
        THEN CV:= CV + INC;                    /// Увеличение текущего значения
             QU:= FALSE;
        ELSE QU:= TRUE;                        // Достигнут верхний предел
    END_IF
    UP:= CV;
  END_METHOD                                   // Результат метода

  METHOD PUBLIC UP5: UINT
  VAR_OUTPUT QU: BOOL; END_VAR
    UP5:= THIS.UP(INC:= 5, QU => QU);          // Count up by 5
  END_METHOD                                   // Достигнут верхний предел
END_CLASS                                      // Вызов внутреннего метода
```

6.6.5.4.4 Представление вызова метода

Методы могут вызываться в текстовых языках (таблица 50) и в графических языках.

В представлениях всех языков имеется два разных случая вызова метода:

a) внутренний вызов метода из экземпляра собственного класса Имя метода предваряется ключевым словом «THIS». Данный вызов может выдаваться другим методом;

b) внешний вызов метода экземпляра другого класса. Имени метода предшествует имя экземпляра и «.».

Этот вызов может выдаваться методом или телом функционального блока, где объявлен экземпляр класса.

Примечание — Используются следующие синтаксисы:

- синтаксис A() используется для вызова глобальной функции A;
- синтаксис THIS.A() используется для вызова метода из собственного экземпляра класса;
- синтаксис THIS.A() используется для вызова метода A из другого экземпляра класса.

6.6.5.4.5 Текстовое представление вызова

Метод с результатом вызывается как операнд выражения. Метод без результата не должен вызываться внутри выражения.

Метод может вызываться формально или неформально.

Внешний вызов метода дополнительно требует имени экземпляра внешнего класса.

Пример 1 — ... class_instance_name.method_name(parameters).

Внутренний вызов метода требует использования THIS вместо имени экземпляра.

Пример 2 — ... THIS.method_name (parameters).

Таблица 50 — Текстовый вызов методов — Формальный и неформальный перечень параметров

Номер	Описание	Пример
1a	Полный формальный вызов (только текстовый) Используется, если указание параметров EN и ENO в вызове является обязательным	A:= COUNTER.UP(EN:= TRUE, INC:= B, START:= 1, ENO=> %MX1, QU => C);
1b	Неполный формальный вызов (только текстовый) Используется, если указание параметров EN и ENO в вызове не является обязательным	A:= COUNTER.UP(INC:= B, QU => C); Переменная START будет иметь неявное значение 0 (ноль)
2	Неформальный вызов (только текстовый) (с фиксированным порядком параметров и полный)	A:= COUNTER.UP(B, 1, C); Данный т вызов эквивалентен вызову в примере 1a, но без параметров EN и ENO

6.6.5.4.6 Графическое представление

Графическое представление вызова метода подобно представлению функции или функционального блока. Это — прямоугольный блок с входами слева и выходами справа от блока.

Вызовы метода могут поддерживать параметры EN и ENO, как определено в таблице 18.

- внутренний вызов показывает имя класса и имя метода, разделенные точками внутри блока. Ключевое слово THIS размещают над блоком;
- внешний вызов показывает имя класса и имя метода, разделенные точкой внутри блока Над блоком размещают имя экземпляра класса.

6.6.5.4.7 Ошибка

Использование выхода метода независимо от вызова метода рассматривается как ошибка. См. пример ниже.

Пример — Внутренний и внешний вызов метода

```
VAR
    CT:     COUNTER;
    LIMIT:  BOOL;
    VALUE: UINT;
END_VAR
```

1) В структурированном тексте (язык ST).

a) Внутренний вызов метода:

```
VALUE:= THIS.UP (INC:= 5, QU => LIMIT);
```

b) Внешний вызов модуля:

```
VALUE:= CT.UP (INC:= 5, QU => LIMIT);
```

2) На функциональных блоковых диаграммах (язык FBD)

Вызван в классе другого метода

Ключевое слово THIS обязательно
Метод UP возвращает результат

Графическое представление служит только для иллюстрации

Переменая On разрешает вызов метода

CT — экземпляр класса, объявленный в другом классе или функциональном блоке

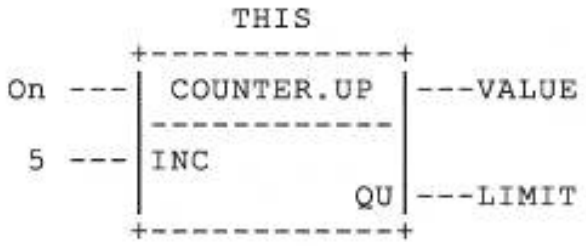
Вызывается методом или в теле функционального блока

Метод UP возвращает результат

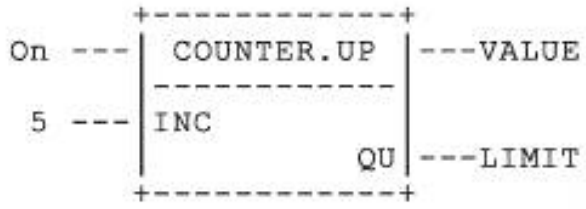
Графическое представление служит только для иллюстрации

Переменная On разрешает вызов метода;

a) Внутренний вызов метода:

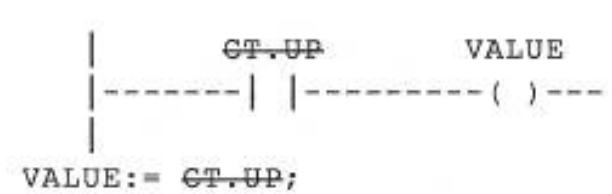


b) Внешний вызов модуля:



3) Ошибка Использование выхода метода без графического или текстового вызова

Данная оценка выхода метода не возможна, так как метод не сохраняет выходы от одного исполнения до другого.



6.6.5.5 Наследование класса (EXTENDS, SUPER, OVERRIDE, FINAL)

6.6.5.5.1 Общие положения

Применительно к языкам программирования PLC, концепция наследования, определенная в объектно-ориентированном программировании применяется как способ создания новых элементов.

Наследование классов показано на рисунке 17. На базе существующего класса может быть порожден один или более классов. Данный процесс может повторяться многократно.

Примечание — «Множественное наследование» не поддерживается.

Порожденный (дочерний) класс обычно расширяет базовый (родительский) класс дополнительными методами.

Термин «базовый» класс означает всех «предков», то есть родительский класс и его родительские классы и т. д.

Наследование классов с использованием EXTENDS

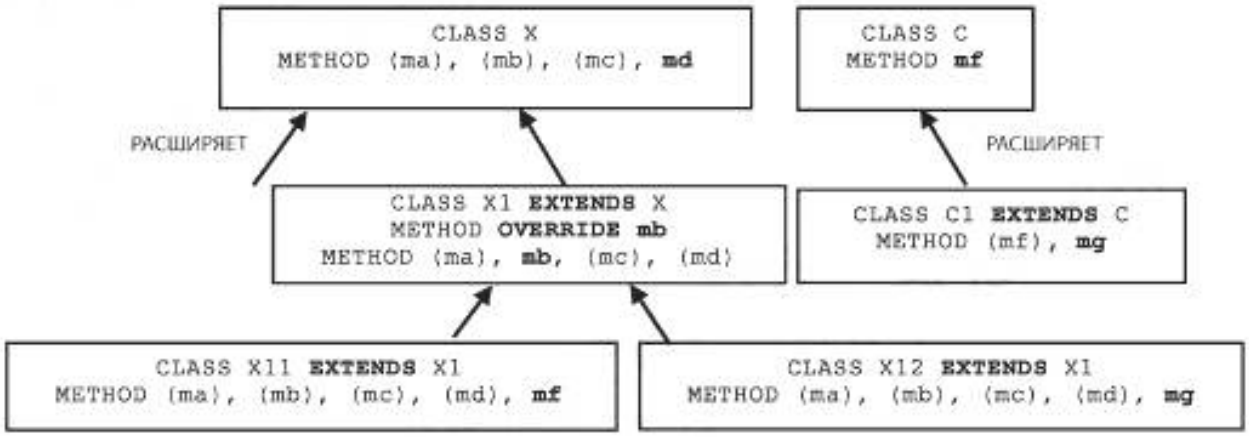


Рисунок 17 — Наследование классов (иллюстрация)

6.6.5.5.2 Ключевое слово EXTENDS для классов

Класс может порождаться из уже существующего класса (базового класса), используя ключевое слово EXTENDS.

Пример — CLASS X1 EXTENDS X;

Применяются следующие правила:

1 Порожденный класс без всяких объявлений наследует все классы (если таковые имеются) из базового класса со следующими исключениями:

- не наследуются методы PRIVATE;
- за пределами пространства имен не наследуются методы INTERNAL.

2 Порожденный класс наследует все переменные (если таковые имеются) из базового класса.

3 Порожденный класс наследует только из базового класса. Множественное наследование в данном стандарте не поддерживается.

Примечание — Класс может реализовывать (используя ключевое слово IMPLEMENTS) один или более интерфейсов.

4 Порожденный класс может расширять базовый класс, то есть может иметь собственные методы и переменные в дополнение к унаследованным методам и переменным базового класса, и таким образом создавать новую функциональность.

5 Класс, используемый в качестве базового класса, сам может быть порожденным классом. Тогда он переносит в порожденный класс также методы и переменные, которые он унаследовал.

Это может повторяться несколько раз.

6 Если определение базового класса изменяется, все порожденные классы (и их потомки) также соответственно изменяют свою функциональность.

6.6.5.5.3 OVERRIDE метод

Порожденный класс может переопределять (заменять) один или более унаследованных методов своими собственными реализациями метода (методов). При переопределении базовых методов применяются следующие правила:

1 Метод, переопределяющий унаследованный метод, имеет такую же сигнатуру (имя метода и переменные) в пределах области действия класса.

2 Метод, переопределяющий унаследованный класс, должен иметь следующие свойства:

- ключевое слово OVERRIDE следует за ключевым словом METHOD;
- порожденный класс имеет доступ к базовым методам, которые определены как PUBLIC or PROTECTED или NTERNAL в том же пространстве имен;
- новый метод будет иметь те же спецификаторы доступа. Но спецификатор FINAL может быть использован для переопределенного метода.

Пример — METHOD OVERRIDE mb;

6.6.5.5.4 FINAL для классов и методов

Метод со спецификатором FINAL не будет переопределяться. Класс со спецификатором FINAL не может быть базовым классом.

Пример 1 — METHOD FINAL mb;

Пример 2 — CLASS FINAL c1.

6.6.5.5.5 Ошибки при использовании ключевых слов (EXTENDS, SUPER, OVERRIDE, FINAL)

Следующие ситуации рассматриваются как ошибка:

1 Порожденный класс определяет переменную с именем переменной (определенной или унаследованной), уже содержащейся в базовом классе. Данное правило не применяется к переменным, объявленным как PRIVATE.

2 Порожденный класс определяет метод с именем, уже содержащемся в базовом классе.

3 Порожденный класс порождается из его собственного базового класса (прямо или косвенно), то есть рекурсия не разрешена.

Класс определяет метод с ключевым словом OVERRIDE, который не переопределяет метод базового класса.

Пример — Наследование и переопределение

Класс, расширяющий класс LIGHTROOM.

```
CLASS LIGHTROOM
VAR LIGHT: BOOL; END_VAR

METHOD PUBLIC DAYTIME
    LIGHT:= FALSE;
END_METHOD

METHOD PUBLIC NIGHTTIME
    LIGHT:= TRUE;
END_METHOD

END_CLASS
    CLASS LIGHT2ROOM EXTENDS LIGHTROOM
VAR LIGHT2: BOOL; END_VAR    // Second light

METHOD PUBLIC OVERRIDE DAYTIME
    LIGHT := FALSE;    // Доступ к переменным родительского класса
    LIGHT := FALSE;    // конкретная реализация END_METHOD

METHOD PUBLIC OVERRIDE NIGHTTIME
    LIGHT := TRUE;     // Доступ к переменным родительского класса
```

```
    LIGHT := TRUE;     // конкретная реализация
    END_METHOD

END_CLASS
```

6.6.5.6 Динамическое связывание имен (OVERRIDE)

Связывание имен — это ассоциация имени метода с именем реализации. Привязка имени (например, компилятором) до выполнения программы называется статической или «ранней» привязкой. Привязка, выполняемая во время выполнения программы, называется динамической или «поздней» привязкой.

В случае вызова внутреннего метода, свойство переопределения с ключевым словом OVERRIDE приводит к различию между статической и динамической формой связывания имен:

- статическое связывание: ассоциирует имя метода с реализацией метода класса с вызовом внутреннего метода, или содержит метод, выполняющий вызов внутреннего метода;
- динамическое связывание: ассоциирует имя метода с реализацией метода фактического типа экземпляра класса.

Пример 1 — Динамическое связывание имен

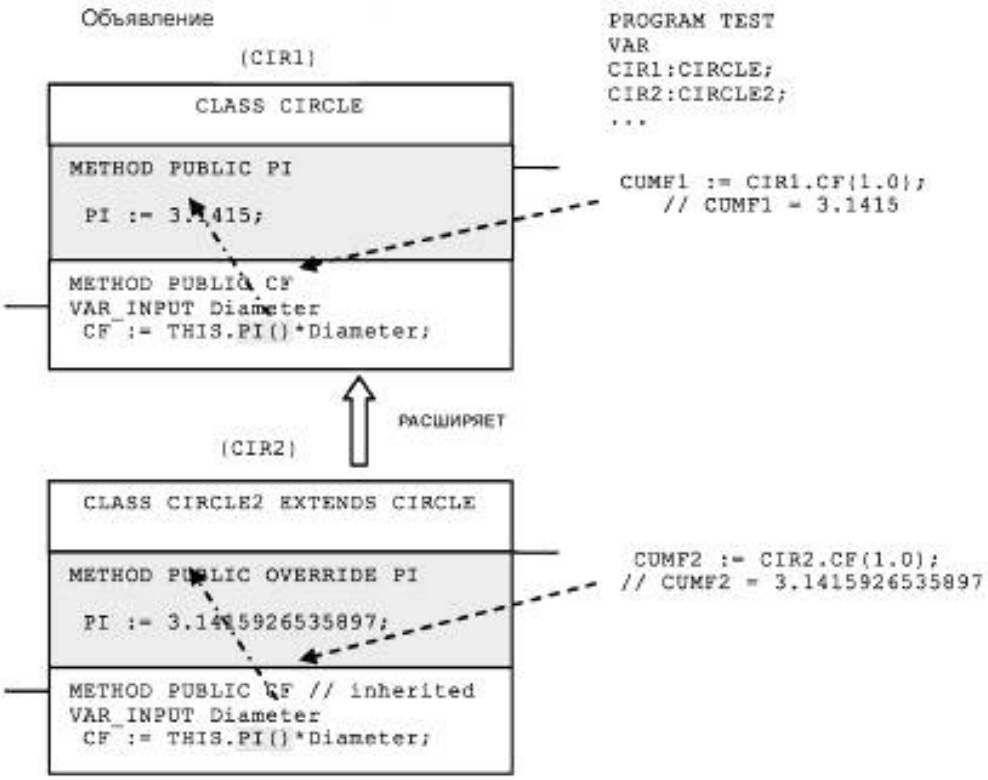
```
Переопределение с воздействием на связывание
// Объявление
CLASS CIRCLE

METHOD PUBLIC PI: LREAL       // Метод дает менее точное определение
    PI PI:= 3.1415;
END_METHOD

METHOD PUBLIC CF: LREAL       // Метод вычисляет длину окружности
    VAR_INPUT DIAMETER: LREAL; END_VAR
    CF:= THIS.PI() * DIAMETER;          // Внутренний вызов метода:
END_METHOD   // используя динамические связывания PI
END_CLASS

CLASS CIRCLE 2 EXTENDS CIRCLE                 // Класс с методом, переопределяющим PI

METHOD PUBLIC OVERRIDE PI: LREAL   // Метод выдает более точное значение PI
    PI:= 3.1415926535897;
END_METHOD
END_CLASS

PROGRAM TEST VAR
    CIR1:       CIRCLE;       // Экземпляр CIRCLE
    CIR2:       CIRCLE2;      // Экземпляр CIRCLE2
    CUMF1:      LREAL;
    CUMF2:      LREAL;
    DYNAMIC: BOOL;
END_VAR

    CUMF1:= CIR1.CF(1.0);             // Вызов метода CIR1
    CUMF2:= CIR2.CF(1.0);             // Вызов метода CIR2
    DYNAMIC:= CUMF1 <> CUMF2;         // Динамическое связывание приводит к значению True
END_PROGRAM
```

В данном примере класс CIRCLE содержит внутренний вызов своего метода PI с низкой точностью для вычисления длины окружности (CF).

Порожденный класс CIRCLE2 переопределяет этот метод более точным определением PI.

Вызов метода PI() ссылается либо на CIRCLE.PI, либо на CIRCLE2.PI, в соответствии с типом экземпляра, в котором выполнялся вызов метода CF. Здесь значение CUMF2 точнее значения CUMF1.

Пример 2 — Графическая иллюстрация приведенного выше текстового примера (упрощенная)



6.6.5.7 Вызов метода собственного или базового класса (THIS, SUPER)

6.6.5.7.1 Общие положения

Для доступа к методу, определенному внутри или вне собственного класса, используются ключевые слова THIS и SUPER.

6.6.5.7.2 THIS

THIS обеспечивает ссылку на экземпляр собственного класса.

С ключевым словом THIS метод экземпляра собственного класса может быть вызван любым другим методом экземпляра этого класса.

THIS может быть передан переменной типа INTERFACE.

Ключевое слово THIS не может использоваться с другим экземпляром, например, выражение myInstance. THIS не разрешено.

Пример — Использование ключевого слова THIS

Для удобства данные примеры копируются из приведенных выше примеров.

```
INTERFACE ROOM
    METHOD DAYTIME END_METHOD // Вызывается в дневное время METHOD NIGHTTIME END_METHOD
                              // Вызывается в ночное время
END_INTERFACE
FUNCTION_BLOCK ROOM_CTRL      //
  VAR_INPUT
    RM: ROOM;                 // Интерфейс ROOM как типа входной переменной
  END_VAR
```

```
    VAR_EXTERNAL
        Actual_TOD: TOD; // Глобальное определение времени
    END_VAR

    IF (RM = NULL)          // ВАЖНО: проверить законность ссылки!
    THEN RETURN;
    END_IF;

    IF Actual_TOD >= TOD#20:15 OR Actual_TOD <= TOD#6:00
    THEN RM.NIGHTTIME();          // вызов метода из RM
    ELSE RM.DAYTIME();
    END_IF;

END_FUNCTION_BLOCK

// Применяет ключевое слово THIS для назначения собственного экземпляра

CLASS DARKROOM IMPLEMENTS ROOM// См. объявление ROOM вышеVAR_EXTERNAL
        Ext_Room_Ctrl: ROOM_CTRL;       // См. объявление ROOM_CTRL выше END_VAR
METHOD PUBLIC DAYTIME;       END_METHOD METHOD PUBLIC NIGHTTIME;       END_METHOD

METHOD PUBLIC EXT_1
Ext_Room_Ctrl(RM:= THIS);       // Вызвать Ext_Room_Ctrl с собственным экземпляром
END_METHOD
END_CLASS
```

6.6.5.7.3 Ключевое слово SUPER

Ключевое слово SUPER обеспечивает доступ к методам реализации базового класса.

С ключевым словом SUPER можно вызвать метод, который действителен в экземпляре базового (родительского) класса. Таким образом, имеет место статическое связывание.

Ключевое слово SUPER не может использоваться с экземплярами других программных элементов, например, выражение my- Room.SUPER.DAYTIME() не разрешено.

Ключевое слово SUPER не может использоваться для доступа к старшим предкам порожденных методов, например, выражение SUPER.SUPER.aMethod не разрешено.

Пример — Использование ключевого слова SUPER и полиморфизм

LIGHT2ROOM с использованием SUPER как альтернативная реализация приведенного выше примера. Для удобства сюда скопированы некоторые предыдущие примеры.

```
INTERFACE ROOM
        METHOD DAYTIME          END_METHOD   // Вызывается в дневное время
        METHOD NIGHTTIME        END_METHOD   // Вызывается в ночное время
END_INTERFACE

CLASS LIGHTROOM IMPLEMENTS ROOM

VAR LIGHT: BOOL; END_VAR

METHOD PUBLIC DAYTIME
        LIGHT:= FALSE;
END_METHOD
METHOD PUBLIC NIGHTTIME
```

```
        LIGHT:= TRUE;
END_METHOD
END_CLASS

FUNCTION_BLOCK ROOM_CTRL
  VAR_INPUT
        RM: ROOM;                  // Интерфейс ROOM как тип переменной
  END_VAR

  VAR_EXTERNAL
        Actual_TOD: TOD;           // Глобальное определение времени
  END_VAR

  IF (RM = NULL)                   // ВАЖНО: проверить действительность ссылки!
  THEN RETURN;
  END_IF;
  IF Actual_TOD >= TOD#20:15 OR
        Actual_TOD <= TOD#06:00
  THEN RM.NIGHTTIME();             // Вызвать метод RM (динамическое связывание с
                                   //либо LIGHTROOM.NIGHTTIME
                                   // либо LIGHT2ROOM.NIGHTTIME)
  ELSE RM.DAYTIME();
  END_IF;
END_FUNCTION_BLOCK

// Применяет ключевое слово SUPER для вызова метода базового класса
CLASS LIGHT2ROOM EXTENDS LIGHTROOM      // см. выше
VAR LIGHT2: BOOL; END_VAR               // логическая переменная light

METHOD PUBLIC OVERRIDE DAYTIME
        SUPER.DAYTIME();                // Вызов метода в LIGHTROOM
        LIGHT2:= TRUE;
END_METHOD

METHOD PUBLIC OVERRIDE NIGHTTIME
        SUPER.NIGHTTIME()               // Вызов метода в LIGHTROOM
        LIGHT2:= FALSE;
END_METHOD
END_CLASS

// Использование полиморфизма и динамическое связывание
PROGRAM C VAR
        MyRoom1: LIGHTROOM;             // См. выше
        MyRoom2: LIGHT2ROOM;            // См. выше
        My_Room_Ctrl: ROOM_CTRL;        // См. выше
END_VAR

        My_Room_Ctrl(RM:= MyRoom1);     // Вызовы в My_Room_Ctrl вызывают методы LIGHTROOM
        My_Room_Ctrl(RM:= MyRoom2);     // Вызовы в My_Room_Ctrl вызывают методы LIGHT2ROOM
END_PROGRAM
```

6.6.5.8 Абстрактный класс и абстрактный метод

6.6.5.8.1 Общие положения

Модификатор ABSTRACT может использоваться с классами или отдельными методами. Разработчик определяет реализацию этих свойств в соответствии с таблицей 48.

6.6.5.8.2 Абстрактный класс

Использование модификатора ABSTRACT в объявлении класса указывает, что класс предназначается для использования в качестве базового типа для других классов.

Пример — CLASS ABSTRACT A1

Абстрактный класс имеет следующие свойства:

- абстрактный класс не может инстанцироваться;
- абстрактный класс содержит, по меньшей мере, один абстрактный метод.

Класс (неабстрактный), порожденный из абстрактного класса включает фактические реализации всех унаследованных абстрактных методов.

Абстрактный класс может использоваться как тип входных и входных-выходных параметров.

6.6.5.8.3 Абстрактный метод

Все методы абстрактного класса, отмеченные модификатором ABSTRACT, будут реализовываться классами, порожденными из абстрактного класса, если сам порожденный класс не отмечен как ABSTRACT.

Методы класса, унаследованные из интерфейса, получают ключевое слово ABSTRACT, если они еще не реализованы.

Ключевое слово ABSTRACT не используется в сочетании с ключевым словом OVERRIDE. Ключевое слово ABSTRACT может использоваться только с методами абстрактного класса.

Пример — METHOD PUBLIC ABSTRACT M1.

6.6.5.9 Спецификаторы доступа (PROTECTED, PUBLIC, PRIVATE, INTERNAL) к методу

Для каждого метода должно быть определено, откуда он может вызываться. Доступность метода определяется с использованием следующих спецификаторов доступа, следующие за ключевым словом METHOD.

- PROTECTED

Если реализовано наследование, применим спецификатор доступа PROTECTED. Он указывает, что метод доступен только внутри класса и из всех порожденных классов.

PROTECTED является умолчанием и может быть опущен.

Примечание — Если наследование не поддерживается, спецификатор доступа PROTECTED действует так же, как PRIVATE.

- PUBLIC

Спецификатор доступа PUBLIC указывает, что метод доступен из любого места, где может использоваться класс.

- PRIVATE

Спецификатор доступа PRIVATE указывает, что метод доступен только внутри самого класса.

INTERNAL

Если пространство имен реализовано, то может использоваться спецификатор доступа INTERNAL. Он указывает для методов, что они доступны только в пределах ПРОСТРАНСТВА ИМЕН, в котором класс объявлен.

Неявно доступ к прототипам методов всегда общий (PUBLIC), поэтому для прототипов методов не используется спецификатор доступа.

Все неправильные использования спецификаторов доступа считаются ошибкой.

Пример — Спецификаторы доступа для методов.

Иллюстрация доступности (вызова) методов, определенных в классе C:

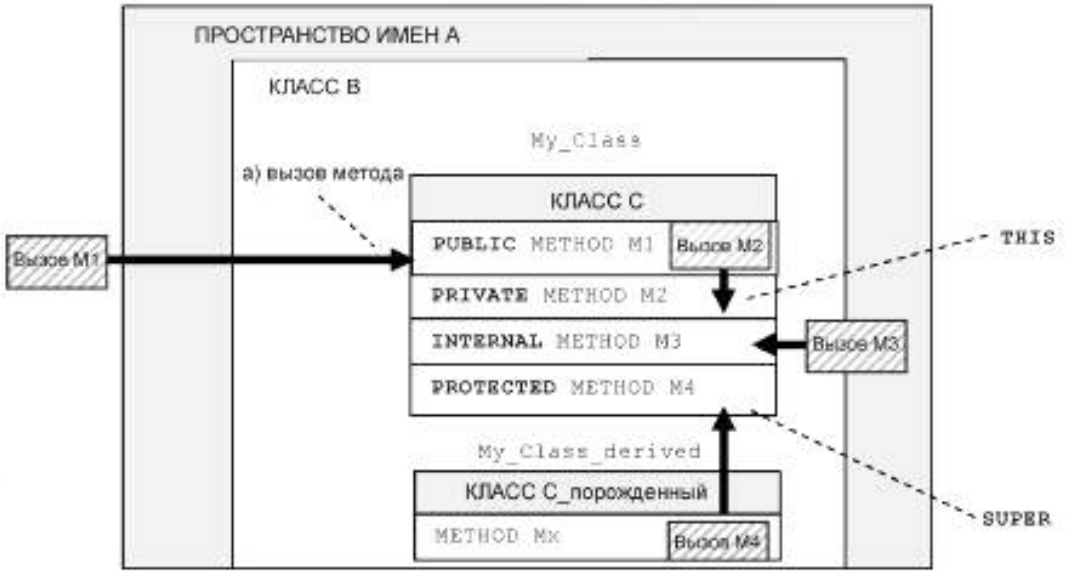
а) спецификаторы доступа: PUBLIC, PRIVATE, INTERNAL, PROTECTED

- PUBLIC Метод **M1** доступен посредством вызова **M1** из класса B (а также класса C);
- PRIVATE Метод **M2** доступен посредством вызова **M2** только из класса C;
- INTERNAL Метод **M3** доступен посредством вызова **M3** из ПРОСТРАНСТВА ИМЕН A (а также класса B и класса C);

- PROTECTED Метод **M4** доступен посредством вызова M4 из класса порожденный_C (а также класса C);

b) вызовы методов изнутри и извне:

- метод **M2** вызывается из класса C — с ключевым словом THIS;
- методы **M1, M3** и **M4** класса C вызываются из класса C — с ключевым словом SUPER для метода **M4.**



6.6.5.10 Спецификаторы доступа к переменной (PROTECTED, PUBLIC, PRIVATE, INTERNAL)

Для секции VAR спецификатор доступа определяет, откуда разрешен доступ к переменным этой секции. Доступность переменных определяется с использованием одного из следующих спецификаторов доступа, располагающихся вслед за ключевым словом VAR.

Примечание — Спецификаторы доступа могут комбинироваться с другими спецификаторами, такими как RETAIN или CONSTANT в любом порядке.

- PROTECTED

Если наследование реализовано, то спецификатор доступа PROTECTED является применимым. Для переменных он указывает, что они достижимы только изнутри класса и изнутри всех порожденных классов. Спецификатор доступа PROTECTED применяется по умолчанию, и может быть опущен.

Если наследование реализовано, но не используется, спецификатор PROTECTED имеет такой же эффект как спецификатор PRIVATE.

- PUBLIC

Спецификатор доступа PUBLIC для переменных указывает, что они доступны в любом месте, где может использоваться класс.

- PRIVATE

Спецификатор доступа PRIVATE для переменных указывает, что доступ к ним может осуществляться только из самого класса.

Если наследование не реализовано, спецификатор доступа PRIVATE используется по умолчанию и может быть опущен.

- INTERNAL

Если реализовано пространство имен, спецификатор доступа INTERNAL является применимым. Он указывает, что переменные доступны только из ПРОСТРАНСТВА ИМЕН, в котором объявлен класс.

Все неправильные использования спецификаторов доступа считаются ошибкой.

6.6.6 Интерфейс

6.6.6.1 Общие положения

В объектно-ориентированном программировании концепция интерфейса вводится для обеспечения отделения спецификации интерфейса от его реализации как класса. Это позволяет использовать различные реализации общей спецификации интерфейса.

Определение интерфейса начинается с ключевого слова INTERFACE с последующим именем интерфейса и оканчивается ключевым словом END_INTERFACE (см. таблицу 51).

Интерфейс может содержать набор (неявно общих) прототипов методов.

6.6.6.2 Использование интерфейса

Спецификация интерфейса может использоваться двумя способами:

a) в объявлении класса

Она определяет, какие методы реализует класс, например, для повторного использования спецификации интерфейса, как показано на рисунке 18;

b) как тип переменной

Переменные, тип которых — интерфейс, являются ссылками на экземпляры классов и им может присваиваться значение до использования. Интерфейсы не используются как входные-выходные переменные.

Таблица 51 — Интерфейс

Номер	Описание Ключевое слово	Объяснение
1	INTERFACE ... END_INTERFACE	Определение интерфейса
	Методы и спецификаторы	
2	METHOD...END_METHOD	Определение метода
	Наследование	
3	EXTENDS	Интерфейс наследует из интерфейса
	Использование интерфейса	
4a	IMPLEMENTS интерфейс	Реализует интерфейс как объявление класса
4b	IMPLEMENTS множественные интерфейсы	Реализует более одного интерфейса в объявлении класса
4c	Интерфейс как тип переменной	Ссылка на реализацию (экземпляр функционального блока) интерфейса

6.6.6.3 Прототип метода

Прототип метода — это сокращенное объявление метода для использования с интерфейсом. Он содержит имя метода, переменные VAR_INPUT, VAR_OUTPUT and VAR_IN_OUT и результат метода. Определение прототипа метода не содержит никакого алгоритма (кода) и временных переменных, то есть он еще не включает реализации.

Доступ к прототипам метода всегда PUBLIC; поэтому спецификатор доступа не используется в прототипе метода.

Ниже приведена иллюстрация интерфейса INTERFACE general_drive, включающая:

a) прототипы метода (без алгоритма);

b) класс drive_A и класс drive_B: IMPLEMENTS INTERFACE general_drive.

Данные классы имеют методы с разными алгоритмами.

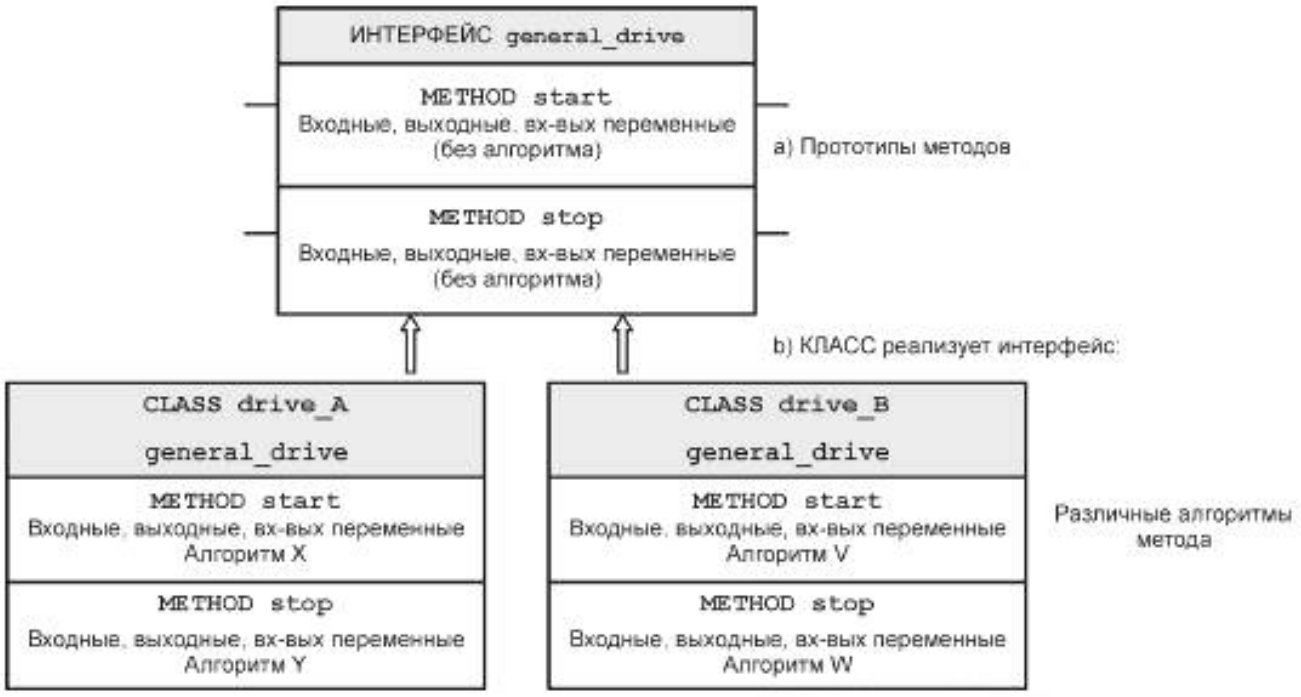


Рисунок 18 — Интерфейс с порожденными классами (иллюстрация)

6.6.6.4 Использование интерфейса в объявлении класса (IMPLEMENTS)

6.6.6.4.1 Общие положения

Класс может реализовывать один или более ИНТЕРФЕЙСОВ с использованием ключевоего слова IMPLEMENTS.

Пример — CLASS B IMPLEMENTS A1, A2;

Класс реализует алгоритмы всех методов, указанных прототипами метода, которые содержатся в спецификациях ИНТЕРФЕЙСА.

Класс, который не реализует все прототипы метода, будет отмечен как ABSTRACT и не может быть инстанцирован.

Примечание — Реализация прототипа метода может иметь дополнительные временные переменные в методе.

6.6.6.4.2 Ошибки

Следующие ситуации рассматриваются как ошибка:

1 Если класс не реализует все методы, определенные в базовом (родительском) интерфейсе, и класс инстанцирован.

2 Если класс реализует метод с таким же именем, которое определено в интерфейсе, но с другой сигнатурой.

3 Если класс реализует метод с таким же именем, которое определено в интерфейсе, но не со спецификатором доступа PUBLIC или INTERNAL.

6.6.6.4.3 Пример

Приведенный ниже пример иллюстрирует объявление интерфейса в классе и использование посредством внешнего вызова метода

Пример — Класс реализует интерфейс.

```
// Объявление

INTERFACE ROOM
    METHOD DAYTIME        END_METHOD   // Вызывается в дневное время
    METHOD NIGHTTIME      END_METHOD   // Вызывается в ночное время
END_INTERFACE

CLASS LIGHTROOM IMPLEMENTS ROOM
    VAR LIGHT: BOOL; END_VAR

METHOD PUBLIC DAYTIME
    LIGHT:= FALSE;
END_METHOD

METHOD PUBLIC NIGHTTIME
    LIGHT:= TRUE;
    END_METHOD
END_CLASS

    // Использование (посредством внешнего вызова метода)

PROGRAM A
    VAR       MyRoom: LIGHTROOM; END_VAR; // Инстанцирование класса
    VAR_EXTERNAL Actual_TOD: TOD; END_VAR; // определение глобального времени
    IF Actual_TOD >= TOD#20:15 OR Actual_TOD <= TOD#6:00
      THEN MyRoom.NIGHTTIME();
      ELSE MyRoom.DAYTIME();
    END_IF;
END_PROGRAM
```

6.6.6.5 Использование интерфейса как типа переменной

6.6.6.5.1 Общие положения

Интерфейс может использоваться как тип переменной. Затем данная переменная становится указателем на экземпляр класса, реализующего интерфейс. Переменной должна быть назначена ссылка на экземпляр класса до того, как она может использоваться. Данное правило применяется во всех случаях, где может использоваться переменная.

Переменной типа INTERFACE могут быть назначены следующие значения:

1) экземпляр класса, реализующего интерфейс;

2) экземпляр класса, порожденного (посредством EXTENDS) из класса, реализующего интерфейс;

3) другая переменная того же порожденного типа INTERFACE;

4) специальное значение NULL, указывающее на недопустимую ссылку. Данное значение также является начальным переменной, если она не инициализирована иным образом.

Переменная типа INTERFACE может сравниваться на равенство с другой переменной того же типа. Результат имеет значение TRUE, если переменные ссылаются на один и тот же экземпляр или если значения обоих переменных равны NULL.

6.6.6.5.2 Ошибка

Значение переменной типа интерфейс должно быть присвоено до ее использования, и должна быть проведена проверка, что оно указывает на действительный экземпляр класса. В противном случае возникает ошибка времени выполнения.

Примечание — Для предотвращения ошибки времени выполнения, инструментальные программные средства должны предоставить неявный «пустой» метод. Другой способ состоит в предварительной проверке того, что назначен действительный экземпляр класса.

6.6.6.5.3 Пример

В примерах 1 и 2 показаны объявление и использование интерфейсов как типа переменной.

Пример 1 — Тип функционального блока с вызовом методов интерфейса

```
// Объявление

INTERFACE ROOM
    METHOD DAYTIME END_METHOD          // вызывается в дневное время
    METHOD NIGHTTIME END_METHOD        // вызывается в ночное время
END_INTERFACE

CLASS LIGHTROOM IMPLEMENTS ROOM
    VAR LIGHT: BOOL; END_VAR

METHOD PUBLIC DAYTIME
    LIGHT:= FALSE;
END_METHOD

METHOD PUBLIC NIGHTTIME
    LIGHT:= TRUE;
  END_METHOD
END_CLASS

FUNCTION_BLOCK ROOM_CTRL
  VAR_INPUT RM: ROOM; END_VAR // Интерфейс ROOM как тип (входной) переменной
  VAR_EXTERNAL
  Actual_TOD: TOD; END_VAR    // Определение глобального времени

  IF (RM = NULL)                          // Важно: тест на действительную ссылку!
  THEN RETURN;
  END_IF;
  IF Actual_TOD >= TOD#20:15 OR
    Actual_TOD <= TOD#06:00
  THEN RM.NIGHTTIME();                // Вызов метода RM ELSE RM.
  DAYTIME();
  END_IF;
END_FUNCTION_BLOCK

// Использование

PROGRAM B
  VAR                                 // Инстанцирование
    My_Room:       LIGHTROOM;    // См. LIGHTROOM IMPLEMENTS ROOM
    My_Room_Ctrl:  ROOM_CTRL;    // См. ROOM_CTRL выше
  END_VAR

  My_Room_Ctrl(RM:= My_Room);
            // Вызов FB с передачей экземпляра класса в качестве
            входной переменной
  END_PROGRAM
```

В данном примере функциональный блок объявляет переменную типа интерфейс как параметр. Вызов экземпляра функционального блока передает экземпляр (указатель) класса, реализующего интерфейс, этой переменной. Затем метод, вызванный в классе, использует методы переданного экземпляра класса. Таким образом, можно передавать экземпляры различных классов, реализующих интерфейс.

Объявление:

Интерфейс ROOM с двумя методами и класс LIGHTROOM, реализующий интерфейс.

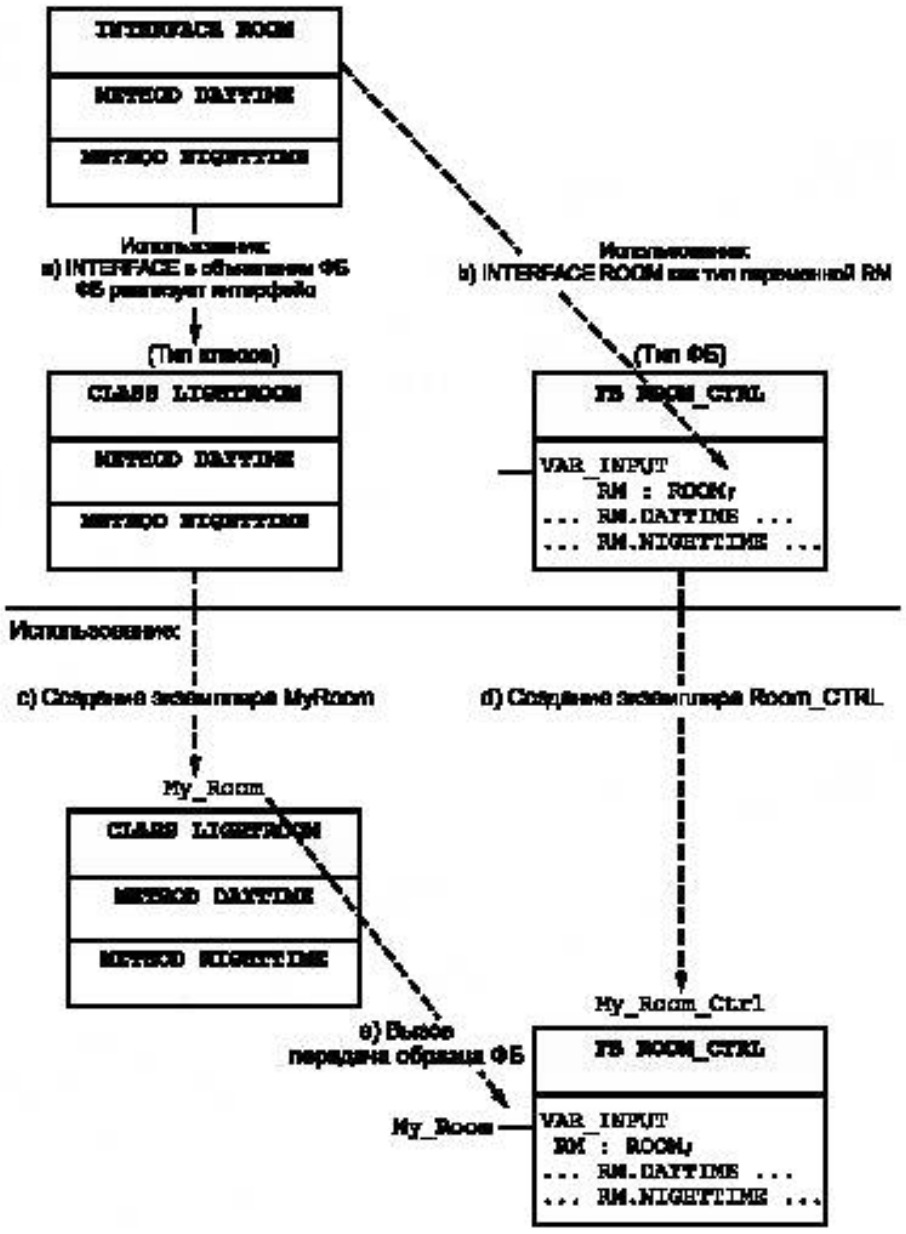
Функциональный блок ROOM_CTRL с входной переменной RM, которая имеет тип интерфейса ROOM. Функциональный блок ROOM_CTRL вызывает методы переданного класса, которые реализуют интерфейс.

Использование:

Программа B инстанцирует класс My_Room и функциональный блок My_Room_Ctrl и вызывает функциональный блок My_Room_Ctrl с передачей класса My_Room входной переменной RM типа интерфейс ROOM.

Пример 2 — Иллюстрация отношений из примера 1

Объявление:



Примечание — Функциональный блок не имеет реализованных методов, но вызывает методы переданного класса!

6.6.6.6 Наследование интерфейса (EXTENDS)

6.6.6.6.1 Общие положения

Применительно к языкам программирования PLC концепция наследования и реализации, определенная в объектно-ориентированном программировании применяется как способ создания новых элементов, как показано на рисунке 19 a), b), c) ниже.

a) Наследование интерфейса

Порожденный (дочерний) интерфейс расширяет (EXTENDS) базовый (родительский) интерфейс, который уже был определен, или

b) Реализация класса

Порожденный класс реализует (IMPLEMENTS) один или более интерфейсов, которые уже были определены, или

c) Наследование класса

Порожденный класс расширяет (EXTENDS) базовый класс, который уже был определен.

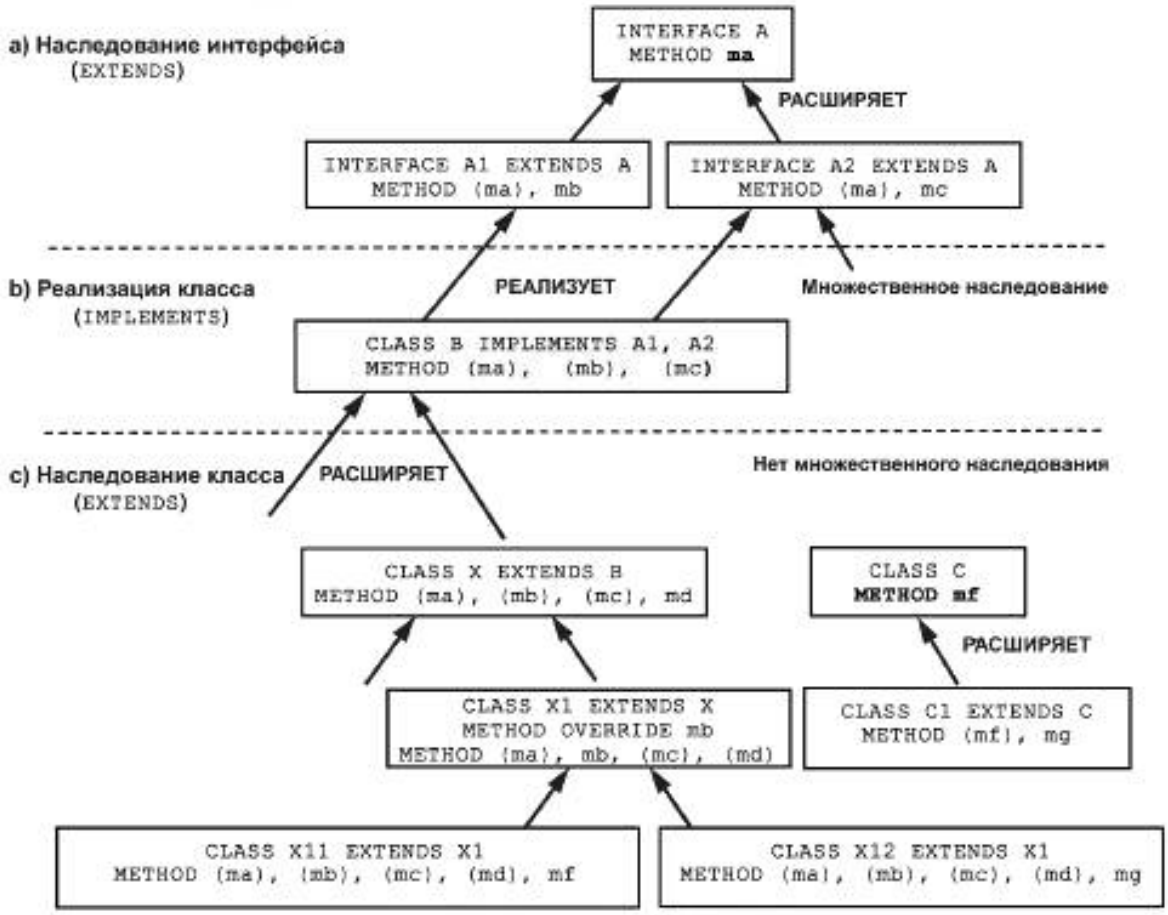


Иллюстрация иерархии наследования:

a) наследование интерфейса с использованием ключевого слова EXTENDS;

b) реализация интерфейса, используя ключевое слово IMPLEMENTS;

c) класса, используя ключевые слова EXTENDS и OVERRIDE.

Рисунок 19 — Наследование интерфейса и класса

Наследование интерфейса, как показано на рисунке 19 а) является первым из трех уровней наследование/реализация. На основе базового интерфейса можно породить один или более интерфейсов.

Интерфейс может быть порожден из одного или более существующих интерфейсов (базовых интерфейсов), используя ключевое слово EXTENDS.

Пример — Интерфейс A1 расширяет интерфейс A.

Применяются следующие правила:

1 Порожденный (дочерний) интерфейс наследует без дополнительных объявлений все прототипы методов из его базового (родительского) интерфейса.

2 Порожденный интерфейс может наследовать из произвольного числа базовых интерфейсов.

3 Порожденный интерфейс может расширять множество прототипов методов, то есть он может иметь прототипы метода дополнительные к прототипам метода своего базового интерфейса и, таким образом, создавать новую функциональность.

4 Интерфейс, используемый как базовый интерфейс, может сам являться порожденным интерфейсом. Когда он передается своим порожденным интерфейсам, наследуются также прототипы метода.

Данный процесс может повторяться многократно.

5 Если базовый интерфейс изменяет свое определение, все порожденные интерфейсы (и их потомки) также имеют эту измененную функциональность.

6.6.6.6.2 Ошибка

Следующие ситуации будут рассматриваться как ошибка:

1) интерфейс определяет дополнительный прототип метода (в соответствии с правилом 3) с таким же именем прототипа метода, как и один из его базовых интерфейсов;

2) интерфейс является своим собственным базовым интерфейсом, явно или неявно, то есть рекурсия не разрешена.

Пример — Свойство OVERRIDE, как определено в 6.6.5.5 для классов, не применимо для интерфейсов.

6.6.6.7 Попытка присваивания

6.6.6.7.1 Общие положения

Попытка присваивания используется для проверки того, реализует ли экземпляр данный интерфейс (см. таблицу 52). Это применимо для классов и функциональных блоков.

Если экземпляр, на который дана ссылка, принадлежит классу или типу функционального блока, реализующего интерфейс, то результат является действительной ссылкой на данный экземпляр. В противном случае, результатом является NULL.

Синтаксис попытки присваивания может также использоваться для безопасных преобразований ссылок интерфейсов в ссылки на классы (или типов функциональных блоков), или ссылки на базовый тип в ссылку на порожденный тип (нисходящее преобразование типа).

Результат попытки присваивания подтверждается отличием от значения NULL перед использованием.

6.6.6.6.7.2 Текстовое представление

В перечне инструкций (язык IL), оператор «ST» (Сохранить) используется как показано в следующем примере.

Пример 1

```
LD      interface2      // в языке IL
ST? interface1
```

В структурированном тексте (язык ST), оператор «?=» используется как показано в следующем примере.

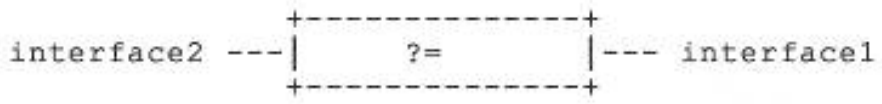
Пример 2

```
interface1 ?= interface2;    // в языке ST
```

6.6.6.6.7.3 Графическое представление

В графических языках используется следующая конструкция:

Пример 1



Пример 2 — Попытка присваивания с ссылками интерфейса

Успешная и неудачная попытка присваивания с ссылками интерфейса

```
// Объявление

CLASS C IMPLEMENTS ITF1, ITF2
END_CLASS

// Использование

PROGRAM A
  VAR
    inst: C;
    interf1: ITF1;
    interf2: ITF2;
    interf3: ITF3;
  END_VAR

interf1:= inst;          // теперь interf1 содержит действительную ссылку
interf2 ?= interf1;      // interf2 будет содержать действительную ссылку
                         // равную interf2:= inst;
interf3 ?= interf1;      // interf3 будет равно NULL
END_PROGRAM
```

Пример 3 — Попытка присваивания с указателями интерфейса

```
// Объявление

CLASS CIBase IMPLEMENTS ITF1, ITF2
END_CLASS

CLASS CIDerived EXTENDS CIBase
END_CLASS

// Использование

PROGRAM A
  VAR
    instbase: CIBase;
    instderived: CIDerived;
    rinstBase1, pinstBase2: REF_TO CIBase;
    rinstDerived1, rinstDerived2: REF_TO CIDerived;
    rinstDerived3, rinstDerived4: REF_TO CIDerived;
    interf1: ITF1;
    interf2: ITF2;
    interf3: ITF3;
  END_VAR
```

```
rinstBase1:= REF(instBase);      // rinstbase1 ссылается на базовый класс
rinstBase2:= REF(instDerived);   // rinstbase2 ссылается на порожденный класс
rinstDerived1 ?= rinstBase1;     // rinstDerived1 == NULL
rinstDerived2 ?= rinstBase2;     // rinstDerived2 будет содержать действительную
                                 // ссылку на instDerived
interf1:= instbase;          // interf1 является ссылкой на базовый класс
interf2:= instderived;       // interf2 является ссылкой на порожденный класс
rinstDerived3 ?= interf1;    // rinstDerived3 == NULL
rinstDerived4 ?= interf2;    // rinstDerived4 будет содержать действительную
                             // ссылку на instDerived
END_PROGRAM
```

Результат попытки присваивания подтверждается отличием от значения NULL перед использованием.

Таблица 52 — Попытка присваивания

Номер	Описание	Пример
1	Попытка присваивания интерфейсов, используя «?=»	См. выше
2	Попытка присваивания интерфейсов, используя «?=»	См. выше

6.6.7 Объектно-ориентированные свойства функциональных блоков

6.6.7.1 Общие положения

Концепция функциональных блоков МЭК 61131-3 расширена для поддержки объектно-ориентированной парадигмы в том объеме, как она определена для классов:

- в функциональных блоках дополнительно используются методы;
- функциональными блоками дополнительно реализуются интерфейсы;
- поддерживается наследование функциональных блоков.

В объектно-ориентированных функциональных блоках поддерживаются все свойства, определенные в таблице 40.

Кроме того, разработчик объектно-ориентированных функциональных блоков предоставляет внутренне согласованное подмножество свойств объектно-ориентированных функциональных блоков, определенное в таблице 53.

Таблица 53 — Объектно-ориентированный функциональный блок

Номер	Описание Ключевое слово	Объяснение
1	Объектно-ориентированный функциональный блок	Объектно-ориентированное расширение концепции функциональных блоков
1a	Спецификатор FINAL	Функциональный блок не может использоваться как базовый функциональный блок
	Методы и спецификаторы	
5	METHOD...END_METHOD	Определение метода
5a	Спецификатор PUBLIC	Метод может вызываться откуда угодно
5b	Спецификатор PRIVATE	Метод может вызываться только внутри определяющего программного компонента
5c	Спецификатор INTERNAL	Метод может вызываться из одного и того же пространства имен
5d	Спецификатор PROTECTED	Метод может вызываться только из определяющего программного компонента и его наследников (неявно)

Окончание таблицы 53

Номер	Описание Ключевое слово	Объяснение
5e	Спецификатор FINAL	Метод не может быть перегружен
	Использование интерфейса	
6a	IMPLEMENTS интерфейс	Реализует интерфейс в объявлении функционального блока
6b	IMPLEMENTS множественные интерфейсы	Реализует более одного интерфейса в объявлении функционального блока
6c	Интерфейс как тип переменной	Поддержка ссылок на реализации (экземпляры функциональных блоков) интерфейса
	Наследование	
7a	EXTENDS	Функциональный блок наследует из базового функционального блока
7b	EXTENDS	Функциональный блок наследует из базового функционального блока
8	OVERRIDE	Метод переопределяет базовый метод — см. динамическое связывание имен
9	ABSTRACT	Абстрактный функциональный блок — по меньшей мере, один метод является абстрактным Абстрактный метод — это шаблон метода
	Ссылка на доступ	
10a	THIS	Ссылка на собственные методы
10b	Ключевое слово SUPER	Ссылка доступа на метод в базовом функциональном блоке
10c	SUPER()	Ссылка доступа на тело в базовом функциональном блоке
	Спецификаторы доступа переменной	
11a	Спецификатор PUBLIC	Доступ к переменной возможен из любого места
11b	Спецификатор PRIVATE	Доступ к переменной осуществляется только внутри определяющего программного компонента
11c	Спецификатор INTERNAL	Доступ к переменной осуществляется только внутри одного и того же пространства имен
11d	Спецификатор PROTECTED	Доступ к переменной осуществляется только из определяющего программного компонента и его наследников (неявно)
	Полиморфизм	
12a	с секцией переменных VAR_IN_OUT с одинаковой сигнатурой	Секции VAR_IN_OUT базового типа FB может присваиваться экземпляр порожденного типа FB без дополнительных переменных VAR_IN_OUT, VAR_INPUT и VAR_OUTPUT
12b	с секцией переменных VAR_IN_OUT с совместимой сигнатурой	Секции VAR_IN_OUT базового типа FB может присваиваться экземпляр порожденного типа FB без дополнительных переменных VAR_IN_OUT
12c	со ссылкой с одинаковой сигнатурой	Ссылке (базового) типа FB может присваиваться адрес экземпляра порожденного типа FB без дополнительных переменных VAR_IN_OUT, VAR_INPUT и VAR_OUTPUT
12d	со ссылкой с совместимой сигнатурой	Ссылке (базового) типа FB может присваиваться адрес экземпляра порожденного типа FB без дополнительных переменных VAR_IN_OUT

6.6.7.2 Методы для функциональных блоков

6.6.7.2.1 Общие положения

Концепция методов принимается как набор факультативных элементов языка, используемых в определении типа функционального блока.

Методы могут применяться для определения операций с данными экземпляров функционального блока.

6.6.7.2.2 Варианты функциональных блоков

Функциональный блок может иметь тело функционального блока и дополнительно набор методов. Так как тело FB и/или методы могут быть опущены, существует три варианта функционального блока. Это показано в примере на рисунках 20 a), 20 b), 20 c).

a) Функциональный блок, имеющий только тело

Данный функциональный блок известен из МЭК 61131-3.

В данном случая у функционального блока нет реализованных методов. Элементы функционального блока (входные переменные, выходные переменные и т. п.) и вызовы функционального блока показаны на примере на рисунке 20 a).

b) Функциональный блок телом FB и методами

Методы поддерживают доступ к их собственным локально определенным переменным, а также к переменным, определенным в секциях VAR_INPUT, VAR_OUTPUT и VAR объявления функционального блока.

c) Функциональные блоки, имеющие только методы

В данном случае, функциональный блок имеет реализованное пустое тело функционального блока. Элементы функционального блока и вызов методов показан на рисунке 20 b).

В данном случае, функциональный блок может также быть объявлен как класс.

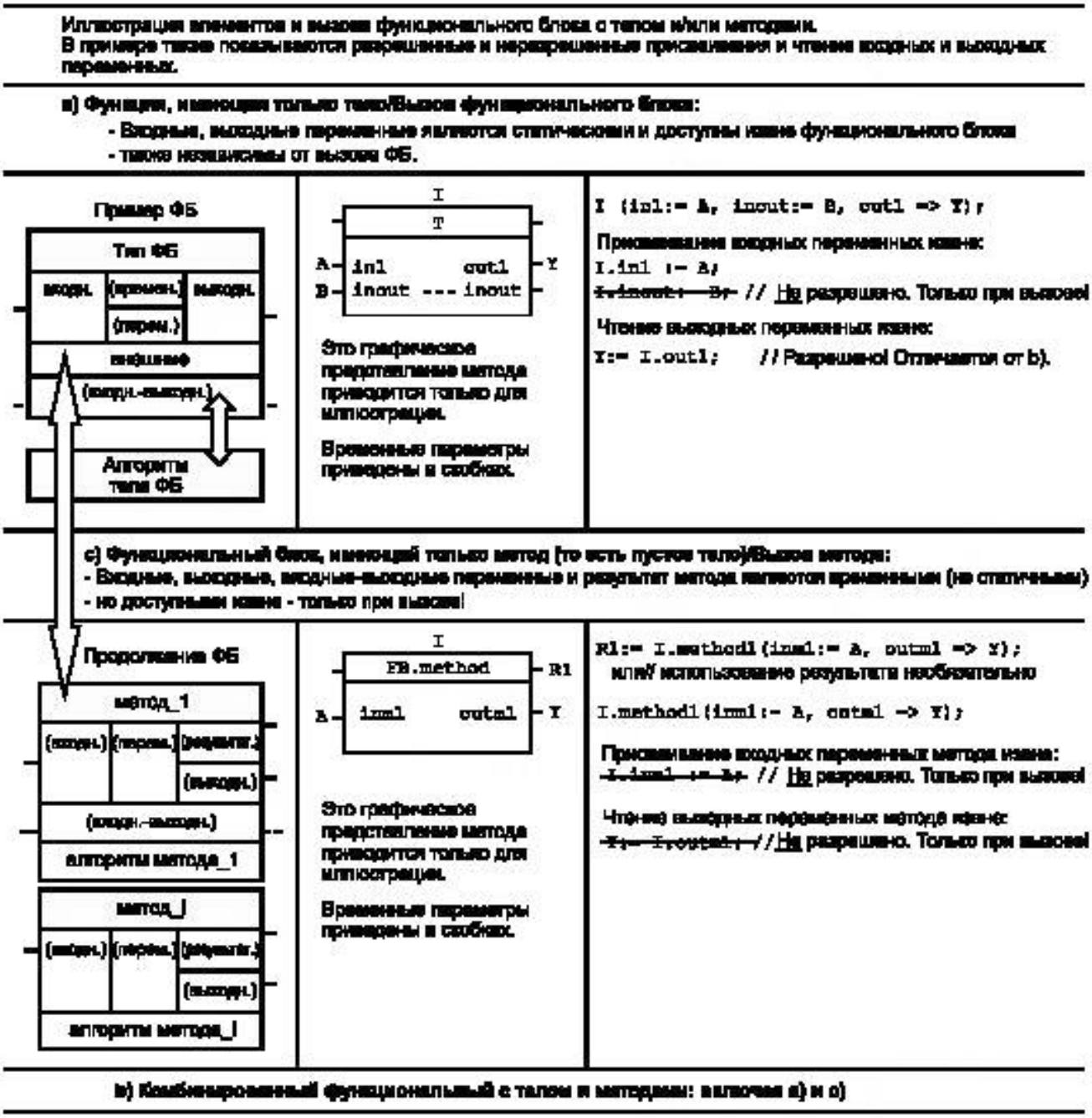


Рисунок 20 — Функциональный блок с необязательным телом и методами (иллюстрация)

6.6.7.2.3 Объявление и выполнение метода

Функциональный блок может иметь набор методов, приведенных на рисунке 20 c).

Объявление метода подчиняется правилам метода класса, а также дополнительно следующим правилам:

1 Методы объявляются в области действия типа функционального блока.

2 В текстовом объявлении методы перечисляются между частью объявлений функционального блока и телом функционального блока.

Выполненные методы подчиняются правилам для методов класса и дополнительно следующим правилам:

3 Все методы имеют доступ для чтения-записи к статическим переменным, объявленным в функциональном блоке: Входные переменные (кроме тех, которые имеют тип данных BOOL R_EDGE или BOOL F_EDGE), входные, статические и внешние переменные.

4 Метод не имеет доступа к временным переменным VAR_TEMP и входным-выходным переменным VAR_IN_OUT функционального блока.

5 Переменные метода недоступны в теле (алгоритме) функционального блока.

6.6.7.2.4 Представление вызова метода

Методы могут вызываться так же, как определено для классов в текстовых и графических языках.

6.6.7.2.5 Спецификаторы доступа (PROTECTED, PUBLIC, PRIVATE, INTERNAL) к методу

Для каждого метода должно быть определено, откуда он может вызываться.

6.6.7.2.6 Спецификаторы доступа к переменным (PROTECTED, PUBLIC, PRIVATE, INTERNAL)

Для секции VAR должно быть определено, откуда разрешен доступ к переменным этой секции.

Доступ к входным и выходным переменным неявно является общим (PUBLIC), поэтому в секциях входных и выходных переменных отсутствует спецификатор доступа. Входные-выходные переменные могут использоваться только в теле функционального блока и в операторе вызова. Доступ к переменным секции VAR_EXTERNAL всегда неявно является защищенным (PROTECTED); поэтому объявление этих переменных не использует спецификатора доступа.

6.6.7.2.7 Наследование функционального блока (EXTENDS, SUPER, OVERRIDE, FINAL)

6.6.7.2.8 Общие положения

Наследование функционального блока похоже на наследование классов. На основе существующего класса или типа функционального блока можно породить один или более функциональных блоков. Данный процесс может повторяться многократно.

6.6.7.2.9 SUPER() в теле порожденного функционального блока

Порожденные функциональные блоки и их базовый функциональный блок могут иметь тело функционального блока. Тело функционального блока не наследуется автоматически из базового функционального блока. По умолчанию, оно пустое. Затем его можно вызвать, используя функцию SUPER().

В этом случае, применяются приведенные выше правила для EXTENDS функционального блока и, дополнительно, следующие правила:

1 Тело (если имеется) порожденного функционального блока будет вычисляться при вызове функционального блока.

2 Для того чтобы дополнительно выполнить тело базового функционального блока (если оно имеется) в порожденном функциональном блоке, используется вызов SUPER(). Вызов SUPER() не имеет параметров.

Вызов SUPER() осуществляется только один раз в теле функционального блока и не используется в цикле.

3 Имена переменных в базовом и порожденном функциональных блоках должны быть уникальными.

4 Вызов функционального блока связывается динамически.

a) Тип порожденного функционального блока может использоваться везде, где может использоваться тип базового функционального блока.

b) Тип порожденного функционального блока может использоваться везде, где может использоваться тип базового функционального блока.

5 SUPER() может вызываться в теле функционального блока, но не в методе функционального блока.

На рисунке 21 показаны примеры использования SUPER():

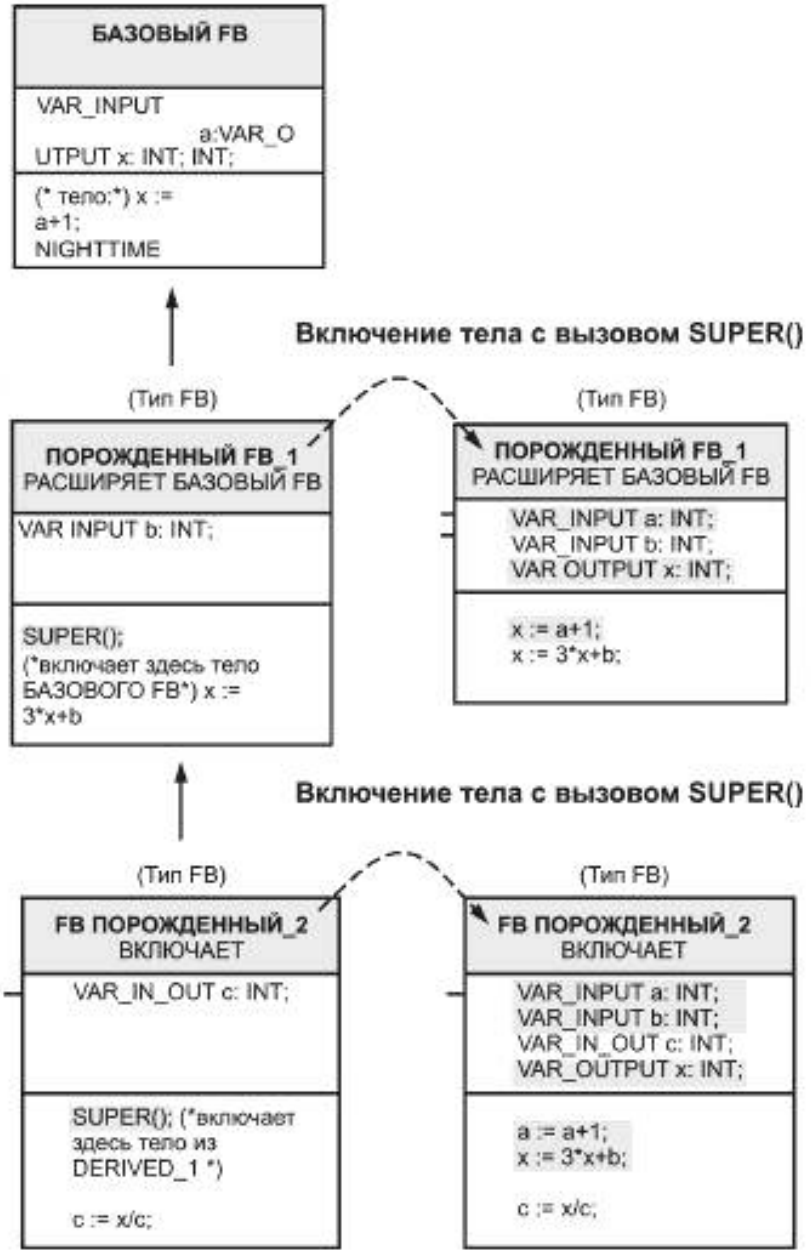


Рисунок 21 — Наследование тела функционального блока с SUPER() (пример)

6.6.7.2.10 OVERRIDE (переопределение метода)

Тип порожденного функционального блока может переопределять (заменять) один или более унаследованных методов собственной реализацией метода.

6.6.7.2.11 FINAL для функционального блока и методов

Метод со спецификатором FINAL не может быть переопределен.

Функциональный блок со спецификатором FINAL не может быть базовым функциональным блоком.

6.6.7.3 Динамическое связывание имен (OVERRIDE)

Связывание имен — это ассоциация имени метода или имени функционального блока с реализацией метода или функционального блока. Оно используется как определено в 6.6.5.6 для методов и функциональных блоков.

6.6.7.4 Вызов метода из собственного и базового FB (THIS, SUPER) и полиморфизм

Для доступа к методу, определенному внутри и снаружи функционального блока используются ключевые слова THIS и SUPER.

6.6.7.5 Абстрактный функциональный блок и абстрактный метод

Модификатор ABSTRACT может также использоваться с функциональными блоками. Реализация этих свойств определяется разработчиком.

6.6.7.6 Спецификаторы доступа (PROTECTED, PUBLIC, PRIVATE, INTERNAL) к методу

Для каждого метода определяется, откуда разрешен вызов метода, как это определено для классов.

6.6.7.7 Спецификаторы доступа к переменной (PROTECTED, PUBLIC, PRIVATE, INTERNAL)

Для секции VAR определяется, откуда разрешен доступ к переменным секции, как определено это для классов.

Доступ к входным и выходным переменным неявно определен как общий (PUBLIC), поэтому спецификатор доступа в секциях входных и выходных переменных не используется. Выходные переменные неявно доступны только для чтения. Входные-выходные переменные могут использоваться только в теле функционального блока и в операторе вызова. Доступ к переменным секции VAR_EXTERNAL всегда неявно является защищенным (PROTECTED); поэтому объявление данных переменных не использует спецификатора доступа.

6.6.8 Полиморфизм

6.6.8.1 Общие положения

Существует четыре случая, где проявляется полиморфизм, они показаны в 6.6.8.2, 6.6.8.3, 6.6.8.4 и 6.6.8.5.

6.6.8.2 Полиморфизм в интерфейсе

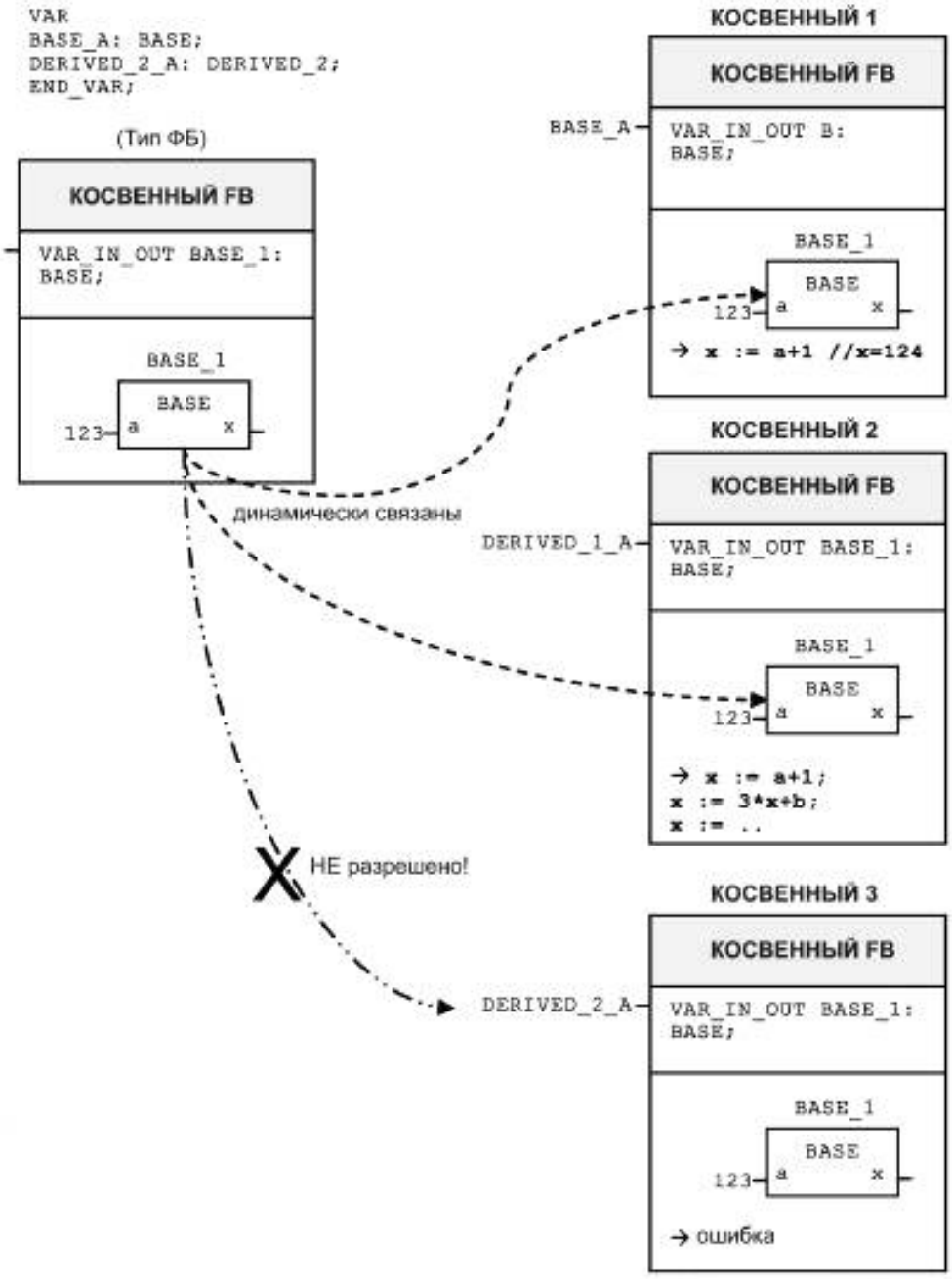
Так как интерфейс нельзя инстанцировать, только порожденные типы могут присваиваться ссылке на интерфейс. Таким образом, любой вызов метода через ссылку на интерфейс представляет собой динамическое связывание.

6.6.8.3 Полиморфизм в секции VAR_IN_OUT

Входным-выходным переменным типа может присваиваться тип порожденного функционального блока, если тип порожденного функционального блока не имеет дополнительных входных-выходных переменных. Разработчик определяет, может ли присваиваться экземпляр типа порожденного функционального блока с дополнительными входными-выходными переменными.

Таким образом, вызов функционального блока и вызов методов функционального блока через экземпляр секции VAR_IN_OUT является случаем динамического связывания.

Пример 1 — Динамическое связывание вызовов функционального блока



Если порожденный блок добавил входные-выходные переменные, то динамическое связывание вызова функционального блока должно приводить к INDIRECT_3 в вычислении неназначенной входной-выходной переменной c и вызывать ошибку периода выполнения. Следовательно, присваивание экземпляра порожденных функциональных блоков является ошибкой.

Пример 2

```
CLASS LIGHTROOM
  VAR LIGHT: BOOL; END_VAR
  METHOD PUBLIC SET_DAYTIME
  VAR_INPUT: DAYTIME: BOOL; END_VAR
    LIGHT:= NOT(DAYTIME);
  END_METHOD
END_CLASS
```

```
CLASS LIGHT2ROOM EXTENDS LIGHTROOM
  VAR  LIGHT2: BOOL; END_VAR                    // Вторая переменная light
  METHOD PUBLIC OVERRIDE SET_DAYTIME
  VAR_INPUT: DAYTIME: BOOL; END_VAR
     SUPER.SET_DAYTIME(DAYTIME);   // Вызов LIGHTROOM.SET_DAYTIME LIGHT2:= NOT(DAYTIME);
  END_METHOD
END_CLASS

FUNCTION_BLOCK ROOM_CTRL
  VAR_IN_OUT RM: LIGHTROOM; END_VAR
  VAR_EXTERNAL Actual_TOD: TOD; END_VAR   // Определение глобального времени
        // В этом случае функциональный блок для вызова динамически связан
        // RM может ссылаться на порожденный класс!
        RM.SET_DAYTIME(DAYTIME:= (Actual_TOD <= TOD#20:15) AND
(Actual_TOD >= TOD#6:00));
END_FUNCTION_BLOCK

// Использование полиморфизма и динамического связывания со ссылкой

PROGRAM D
VAR
     MyRoom1: LIGHTROOM;
     MyRoom2: LIGHT2ROOM;
     My_Room_Ctrl: ROOM_CTRL;
END_VAR

     My_Room_Ctrl(RM:= MyRoom1);
     My_Room_Ctrl(RM:= MyRoom2);
END_PROGRAM;
```

6.6.8.4 Полиморфизм со ссылкой

Пример производного типа может быть назначен для ссылки на базовый класс.

Переменная, имеющая тип, может быть назначена как ссылка на производный тип функционального блока, если производный тип функционального блока не имеет дополнительных входных-выходных переменных. Разработчик определяет, будет ли назначаться ссылка на производный тип функционального блока с дополнительными входными-выходными переменными.

Таким образом, вызов функционального блока и вызов методов функционального блока посредством разыменования ссылки представляют собой случаи динамического связывания.

Пример 1 — Альтернативная реализация примера lightroom

```
FUNCTION_BLOCK LIGHTROOM
VAR LIGHT: BOOL; END_VAR
VAR_INPUT: DAYTIME: BOOL; END_VAR
LIGHT:= NOT(DAYTIME);
END_FUNCTION_BLOCK

FUNCTION_BLOCK LIGHT2ROOM EXTENDS LIGHTROOM
VAR LIGHT2: BOOL; END_VAR    // Дополнительное освещение

SUPER();                                    // Вызов LIGHTROOM
LIGHT2:= NOT(DAYTIME);
END_FUNCTION_BLOCK
```

```
FUNCTION_BLOCK ROOM_CTRL
    VAR_INPUT RM: REF_TO LIGHTROOM; END_VAR
    VAR_EXTERNAL Actual_TOD: TOD; END_VAR // Определение глобального времени

// в этом случае, функциональный блок для вызова динамически связанного
// RM может относиться к производному типу функционального блока!

IF RM <> NULL THEN
    RM^.DAYTIME:= (Actual_TOD <= TOD#20:15) AND (Actual_TOD >= TOD#6:00));
END_IF
END_FUNCTION_BLOCK

// Использование полиморфизма и динамического связывания со ссылкой
PROGRAM D
VAR
    MyRoom1: LIGHTROOM;          // см. выше
    MyRoom2: LIGHT2ROOM;         // см. выше
    My_Room_Ctrl: ROOM_CTRL;     // см. выше
END_VAR

My_Room_Ctrl(RM:= REF(MyRoom1));
My_Room_Ctrl(RM:= REF(MyRoom2));
END_PROGRAM;
```

6.6.8.5 Полиморфизм с THIS

Во время выполнения программы THIS может содержать ссылку на текущий тип функционального блока или на все его производные типы функциональных блоков. Таким образом, любой вызов метода функционального блока с использованием THIS — это случай динамического связывания.

Примечание — При особых обстоятельствах, например, если тип или метод функционального блока объявлен как FINAL, или отсутствуют производные типы функциональных блоков, то ссылка или THIS могут быть полностью определены в период компиляции. В данном случае нет необходимости в динамическом связывании.

6.7 Элементы последовательной функциональной схемы (SFC)

6.7.1 Общие положения

Подраздел 6.7 определяет элементы последовательной функциональной схемы (SFC) для использования в структурировании внутренней организации программного компонента программируемого контроллера, записанные в одном из языков, определенных в настоящем стандарте, для цели выполнения функций последовательного управления. Определения в подразделе 6.7 приведены из МЭК 60848 с изменениями, необходимыми для того, чтобы преобразовать представления из документального стандарта в набор элементов реализации управления для программного компонента программируемого контроллера.

Элементы SFC обеспечивают средства разбиения программного компонента программируемого контроллера на набор шагов и переходов, соединенных между собой направленными связями. С каждым шагом связан набор действий, а с каждым переходом связано условие перехода.

Поскольку элементы SFC нуждаются в сохранении информации о состоянии, программные компоненты, которые могут быть структурированы с использованием таких элементов, представляют собой функциональные блоки и программы.

Если какая-либо часть программного компонента разбивается на элементы SFC, то программный компонент в целом также подвергается разбиению. Если разбиение SFC для программного компонента не предусмотрено, то программный блок в целом рассматривается как одиночное действие, которое исполняется под управлением вызывающего объекта.

6.7.2 Шаги

Шаг представляет ситуацию, в которой поведение программного компонента относительно его входов и выходов следует набору правил, определяемых связанными действиями шага. Шаг может быть активным или неактивным. В любой заданный момент состояние программного компонента определяется набором активных шагов и значений их внутренних и внешних переменных.

Как показано в таблице 54 шаг графически представляется блоком, содержащим имя шага в форме идентификатора или текстуально с помощью конструкции STEP...END_STEP. Направленная в шаг связь (связи) графически представляется вертикальной линией, присоединенной к верху шага. Направленная из шага связь (связи) графически представляется вертикальной линией, присоединенной к низу шага. Как альтернатива, направленные связи представляются в текстовом виде с помощью конструкции TRANSITION... END_TRANSITION.

Флаг шага (активное или неактивное состояние шага) представляется логическим значением элемента логической структуры ***.X, где *** — имя шага, как показано в таблице 54. Эта логическая переменная имеет значение 1, когда соответствующий шаг активен, и значение 0, когда он неактивен. Состояние этой переменной доступно для графического соединения в правой части шага, как показано в таблице 54.

Аналогично, истекшее время ***. T с момента начала шага представляется структурным элементом типа TIME, как показано в таблице 54. Когда шаг деактивирован, значение истекшего времени шага остается на том значении, которое оно имело, когда шаг был деактивирован. Когда шаг активирован, значение истекшего времени шага сбрасывается в t#0s.

Область действия имен шага, флагов шага и времен шага является локальной для программного компонента, в котором появляются шаги.

Начальное состояние программного компонента представлено начальными значениями его внутренних и выходных переменных, и его набором начальных шагов, т. е. шагов, которые первоначально активны. Каждая сеть SFC, или ее текстовый эквивалент, имеет ровно один начальный шаг.

Начальный шаг графически изображается с двойными линиями для границ. Когда для графического представления используется набор символов, установленный в 6.1.1, начальный шаг должен быть изображен так, как показано в таблице 54.

Для инициализации системы начальное истекшее время по умолчанию для шагов — t#0s, а начальное состояние по умолчанию равно BOOL#0 для обычных шагов и BOOL#1 для начальных шагов. Однако когда экземпляр функционального блока или программы объявляется как сохраняемый для экземпляра, состояния и истекшие времена (если поддерживаются) всех шагов, содержащихся в программе или функциональном блоке, должны рассматриваться как сохраняемые для инициализации системы.

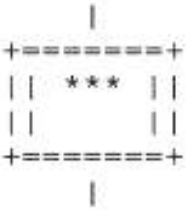
Максимальное число шагов на SFC и точность истекшего времени шага зависят от реализации.

Ошибка возникает, если:

1) сеть SFC содержит не единственный начальный шаг;

2) программа пользователя предпринимает попытки присвоить значение непосредственно состоянию шага или времени шага.

Таблица 54 — Шаг SFC

Номер	Описание	Представление
1a	Шаг — графическая форма с направленными связями	
1b	Начальный шаг — графическая форма с направленной связью	

Окончание таблицы 54

Номер	Описание	Представление
2a	Шаг — текстовая форма без направленных связей	STEP ***: (* Тело шага *) END_STEP
2b	Начальный шаг — текстовая форма без направленных связей	INITIAL_STEP ***: (* Тело шага *) END_STEP
3a[a)]	Флаг шага — общая форма ***.X = BOOL#1 когда *** активно, в противном случае BOOL#0	***.X
3b[a)]	Флаг шага — непосредственное присоединение логической переменной***.X к правой стороне шага	
4[a)]	Истекшее время шага — общая форма ***.T = переменная типа TIME	***.T

Примечание 1 — Верхняя направленная связь к начальному шагу не представлена, если у нее нет предшественников.

Примечание 2 — *** = имя шага.

[a)] Если свойство 3a, 3b или 4 поддерживается, то возникает ошибка, если программа пользователя пытается изменить связанную переменную. Например, если S4 — имя шага, тогда следующие утверждения будут ошибками в языке ST, определенном в подразделе 7.3:

S4.X:= 1; (* ошибка *)

S4.T:= t#100ms; (* ошибка *)

6.7.3 Переходы

Переход представляет условие, в соответствии с которым управление переходит от одного или большего числа шагов, предшествующих переходу, к одному или большему числу последующих шагов вдоль соответствующей направленной связи. Переход представляется горизонтальной линией поперек вертикальной направленной связи.

Направление эволюции, в соответствии в направленными связями, — от низа предшествующего шага (шагов) к верху последующего шага (шагов).

Каждый переход должен иметь связанное условие перехода, которое представляет собой результат оценки одиночного логического выражения. Условие перехода, которое всегда истинно, должно быть представлено символом 1 или ключевым словом TRUE.

Условие перехода может быть связано с переходом с помощью одного из следующих средств, как показано в таблице 55:

a) помещение соответствующего логического выражения на языке ST физически или логически рядом с вертикальной направленной связью;

b) посредством сети релейно-контактных схем на языке LD физически или логически рядом с вертикальной направленной связью;

c) посредством сети на языке FBD, определенном в 8.3, физически или логически рядом с вертикальной направленной связью;

d) посредством сети LD или FBD, выходы которой пересекают вертикальную направленную связь через соединитель;

e) за счет конструкции TRANSITION...END_TRANSITION с использованием языка ST. Конструкция должна включать:

- ключевые слова TRANSITION FROM с последующим именем предшествующего шага (или, если имеется более одного предшественника, с перечнем предшествующих шагов в скобках);
- ключевое слово TO, за которым следует имя следующего шага (или, если имеется более одного преемника, перечень имен следующих шагов в скобках);

- оператор присваивания «:=», за которым следует логическое выражение в языке ST, определяющее условие перехода;
- ключевое слово завершения END_TRANSITION;

f) с помощью конструкции TRANSITION...END_TRANSITION с использованием языка IL. Она должна включать:

- ключевые слова TRANSITION FROM, за которыми следует имя предшествующего шага (или, если имеется более одного предшественника, перечень имен предшествующих шагов в скобках), за которым следует двоеточие «:»;
- ключевое слово TO, за которым следует имя шага последующего шага (или, если имеется более одного преемника, перечень последующих шагов в скобках);
- начиная с новой строки, перечень инструкций на языке IL, результат оценки которых определяет условие перехода;
- ключевое слово завершения END_TRANSITION на отдельной строке;

g) использованием имени перехода в форме идентификатора справа от направленной связи. Данный идентификатор должен относиться к конструкции TRANSITION...END_TRANSITION, определяющей один из следующих объектов, оценка которых приведет к присваиванию логического значения переменной, обозначенной именем перехода:

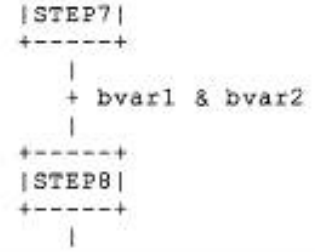
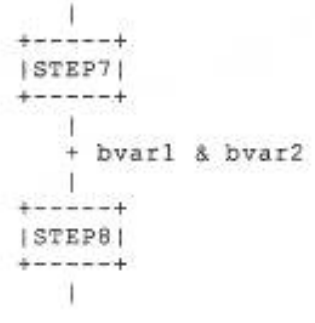
- на языке LD или FBD;
- перечень инструкций на языке IL;
- присваивание логического выражения на языке ST.

Область действия имени перехода должна быть локальной для программного модуля, в котором расположен переход.

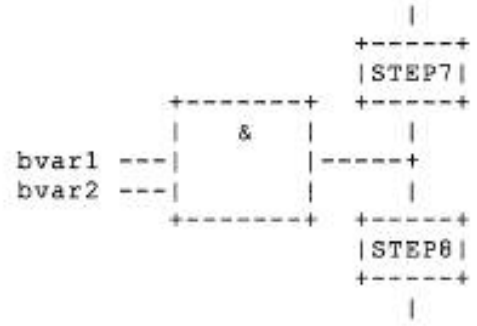
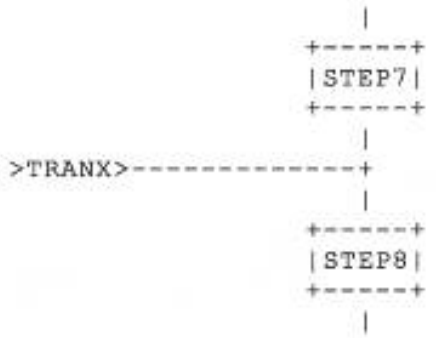
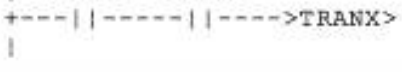
Ошибка возникает, если во время оценки условия перехода появляется какой-либо побочный эффект (например, назначение значения переменной, кроме имени перехода).

Максимальное число переходов на SFC и на шаг определяется разработчиком.

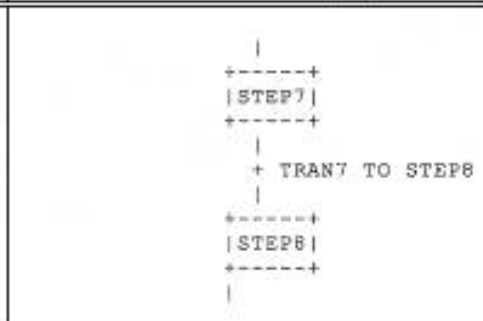
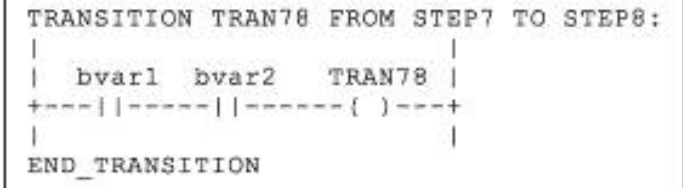
Таблица 55 — Переход SFC и условие перехода

Номер	Описание	Пример
1[a]	Условие перехода на языке ST, расположенное физически или логически рядом с переходом	
2[a]	Условие перехода на языке LD, расположенное физически или логически рядом с переходом	

Продолжение таблицы 55

Номер	Описание	Пример
3[a]	Условие перехода на языке FBD, расположенное физически или логически рядом с переходом	
4[a]	Использование соединителя	
5[a]	Условие перехода на языке LD	
6[a]	Условие перехода на языке FBD	
7[b]	Текстовый эквивалент свойства 1 на языке ST	STEP STEP7: END_STEP TRANSITION FROM STEP7 TO STEP8 := bvar1 & bvar2; END_TRANSITION STEP STEP8: END_STEP
8[b]	Текстовый эквивалент свойства 1 на языке IL	STEP STEP7: END_STEP TRANSITION FROM STEP7 TO STEP 8: LD bvar1 AND bvar2 END_TRANSITION STEP STEP8: END_STEP

Окончание таблицы 55

Номер	Описание	Пример
9[a)]	Использование имени перехода	
10[a)]	Условие перехода на языке LD	
11[a)]	Условие перехода на языке FBD	
12[b)]	Условие перехода на языке IL	TRANSITION TRAN78 FROM STEP7 TO STEP8: LD bvar1 AND bvar2 END_TRANSITION
13[b)]	Условие перехода на языке ST	TRANSITION TRAN78 FROM STEP7 TO STEP8 := bvar1 & bvar2; END_TRANSITION

[a)] Если свойство 1 таблицы 54 поддерживается, то поддерживается одно или более свойств 1, 2, 3, 4, 5, 6, 9, 10 или 11 настоящей таблицы.

[b)] Если свойство 2 таблицы 54 поддерживается, то поддерживается одно или более свойств 7, 8, 12 или 13 настоящей таблицы.

6.7.4 Действия

6.7.4.1 Общие положения

Действие может быть логической переменной, совокупностью команд на языке IL, совокупностью операторов на языке ST, совокупностью цепей на языке LD, совокупностью сетей на языке FBD или организованной последовательной функциональной схемой (SFC).

Действие описывается посредством одного или большего числа механизмов, определенных в 6.7.4.1 и связывается с шагами с помощью тел текстовых шагов или графических блоков действий. Управление действиями выражается классификаторами действий.

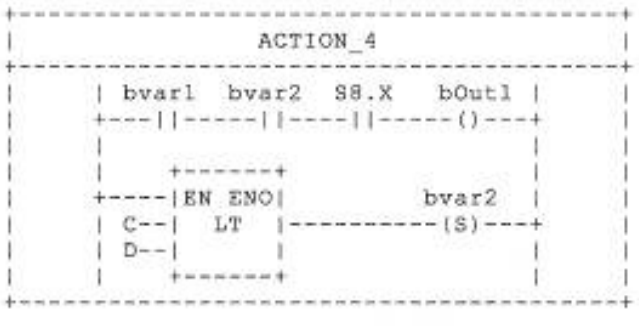
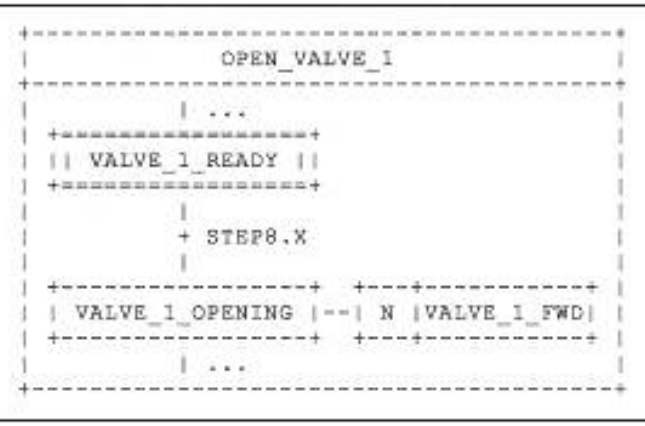
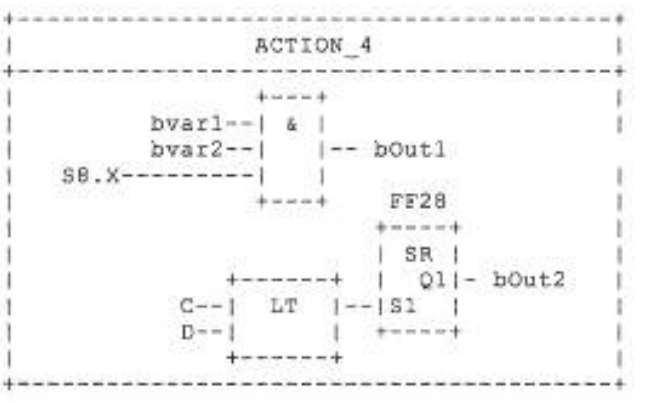
Ошибка возникает, если значение логической переменной, используемое как имя действия, изменяется любым способом, кроме как имя одного или более действий в одной и той же SFC.

Реализация программируемого контроллера, который поддерживает элементы SFC, должна обеспечивать один или более механизмов, определяемых в таблице 56, для объявления действий. Область видимости объявления действия является локальной для программного компонента, содержащего описание.

6.7.4.2 Объявление

С каждым шагом должно быть связано нулевое или большее число действий. Шаг, содержащий нуль связанных действий, должен рассматриваться как имеющий функцию WAIT, то есть ожидающий, когда последующее условие перехода станет истинным.

Таблица 56 — Объявление действий SFC

Номер	Описание[a), b)]	Пример
1	Любая логическая переменная, описанная в блоке VAR или VAR_OUTPUT или их графические эквиваленты могут быть действием	
2l	Графическое описание на языке LD	
2s	Включение элементов SFC в действие	
2f	Графическое описание на языке FBD	

3s	Текстовое описание на языке ST	ACTION ACTION_4:
		bOut1:= bvar1 & bvar2 & S8.X;
		FF28(S1:= (C<D));
		bOut2:= FF28.Q;
		END_ACTION

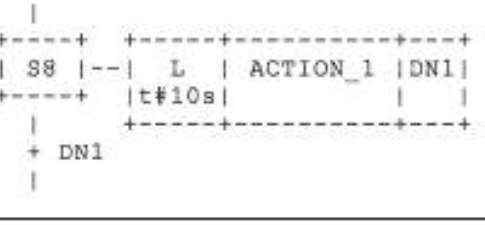
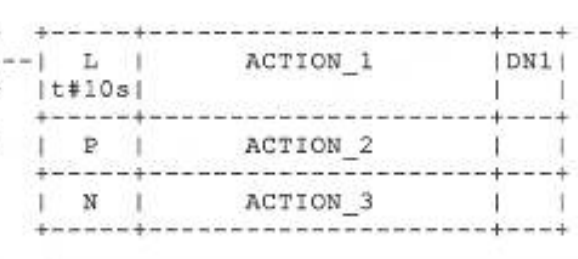
Окончание таблицы 56

Номер	Описание[a), b)]	Пример
3i	Текстовое описание на языке IL	ACTION ACTION_4: LD S8.X AND bvar1 AND bvar2 ST bOut1 LD C LT D S1 FF28 LD FF28.Q ST bOut2 END_ACTION
Примечание — Флаг шага S8.X использован в этих примерах для получения желаемого результата такого, как в случае, когда S8 деактивирован, bOut2:= 0.		
a) Если свойство 1 таблицы 54 поддерживается, то должно поддерживаться одно или более свойств в настоящей таблице, или свойство 4 таблицы 57. b) Если свойство 2 таблицы 54 поддерживается, то должно поддерживаться одно или более свойств 1, 3s или 3i из настоящей таблицы.		

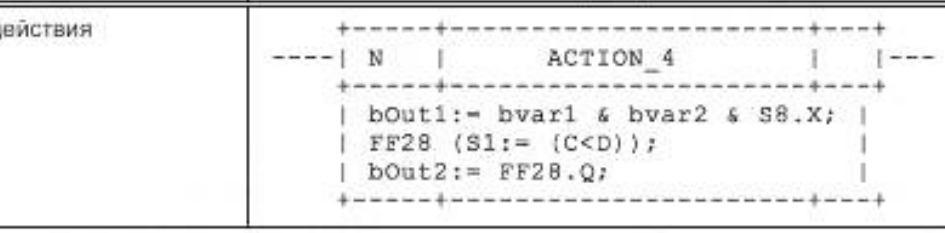
6.7.4.3 Связь с шагами

Реализация программируемого контроллера, который поддерживает элементы SFC, предоставляет один или более механизмов, определяемых в таблице 57, для связи действий с шагами. Максимальное число блоков действий на шаг определяется реализацией.

Таблица 57 — Связь шаг/действие

Номер	Описание	Пример
1	Блок действия, расположенный физически или логически рядом и с шагом	
2	Сцепленные блоки действия, расположенные физически или логически рядом с шагом	
3	Текстовое тело шага	STEP S8: ACTION_1(L,t#10s,DN1); ACTION_2(P); ACTION_3(N); END_STEP

Окончание таблицы 57

Номер	Описание	Пример
4[a)]	Поле «d» блока действия	

[a)] Когда используется свойство 4, то соответствующее имя действия не может быть использовано в любом другом блоке действия.

6.7.4.4 Блоки действий

Как показано в таблице 58, блок действия — это графический элемент для сочетания логической переменной с одним из классификаторов действий для получения разрешающего условия в соответствии с правилами для связанного действия.

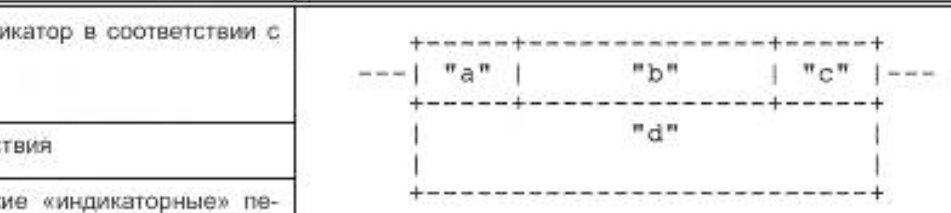
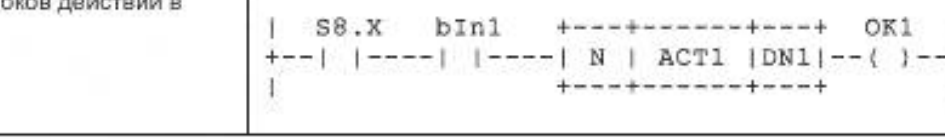
Блок действия предоставляет средства опционального задания логических «индикаторных» переменных, указанных полем «c» в таблице 58, которые могут быть установлены заданным действием для индикации его завершения, ожидания, условий ошибки и т. д. Если поле «c» отсутствует, а поле «b» определяет, что действие должно быть логической переменной, то эта переменная интерпретируется как переменная «c», при необходимости. Если поле «c» не определено, а поле «b» не определяет логическую переменную, то значение «индикаторной» переменной всегда считается равным FALSE.

Когда блоки действий сцеплены графически, как показано в таблице 57, такие конкатенации могут иметь несколько «индикаторных» переменных, но имеют только одну общую логическую входную переменную, которая одновременно действует на все сцепленные блоки.

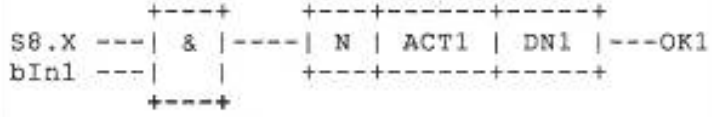
Использование «индикаторной» переменной не рекомендуется.

Помимо того, что блок действия связан с шагом, он может использоваться как графический элемент в языках LD или FBD.

Таблица 58 — Блок действия

Номер	Описание	Графическая форма/пример
1[a)]	«a»: Классификатор в соответствии с 6.7.4.5	
2	«b»: Имя действия	
3[b)]	«c»: Логические «индикаторные» переменные (не рекомендуется)	
	«d»: Действие использует:	—
4i	язык IL	
4s	язык ST	
4l	язык LD	
4f	язык FBD	
5l	Использование блоков действий в языке LD	

Окончание таблицы 58

Номер	Описание	Графическая форма/пример
5f	Использование блоков действий в языке FBD	

[a] Поле «a» может быть пропущено, когда классификатор равен «N».

[b] Поле «c» может быть пропущено, когда индикаторная переменная не используется.

6.7.4.5 Классификаторы действий

Классификатор действия связан с каждой связью шага/действия или с каждым событием блока действий. Значение этого классификатора должно быть одним из значений, перечисленных в таблице 59. Кроме того, классификаторы L, D, SD, DS и SL должны иметь связанную продолжительность времени типа TIME.

Таблица 59 — Классификаторы действий

Номер	Описание	Классификатор
1	Не сохраняется (нулевой классификатор)	Отсутствует
2	Не сохраненяется	N
3	Сброс переопределения	R
4	**Установка (Сохранено)**	S
5	Ограничено по времени	L
6	Отложено	D
7	Импульс	P
8	**Сохранено и отложено по времени**	SD
9	**Отложено и сохранено**	DS
10	**Сохранено и ограничено по времени**	SL
11	Импульс (передний фронт)	P1
12	Импульс (задний фронт)	P0

6.7.4.6 Управление действием

Управление действиями функционально эквивалентно применению следующих правил:

a) С каждым действием был связан функциональный эквивалент экземпляра функционального блока ACTION_CONTROL, определенного на рисунках 22 и 23. Если действие объявлено как логическая переменная, то выход Q этого блока представляет собой состояние этой логической переменной. Если действие объявлено как совокупность операторов или сетей, то эта совокупность должна выполняться непрерывно, пока выход A (активации) функционального блока ACTION_CONTROL поддерживается равным BOOL#1. В этом случае состояние выхода Q (называемое «флагом действия») доступно в пределах действия чтением доступной только для чтения логической переменной, которая имеет форму ссылки на выход Q экземпляра функционального блока, имя экземпляра которого совпадает с именем соответствующего действия, например, ACTION1.Q.

Разработчик может выбрать более простую реализацию, как показано на рисунке 23 b). В этом случае, если действие объявлено как совокупность операторов или сетей, то эта совокупность должна исполняться непрерывно, пока выход Q функционального блока ACTION_ CONTROL поддерживает значение BOOL#1. В любом случае разработчик определяет, какое из свойств таблицы 60 поддерживается.

Примечание 1 — Условие Q=FALSE обычно используется действием для определения того, что оно выполняется получением конечного результата во время его текущей активации.

Примечание 2 — Значение Q равно FALSE во время выполнения действий, вызванных классификаторами P0 и P1.

Примечание 3 — Значение A равно TRUE только для одного выполнения действия, вызванного классификатором P1 или P0. Для всех других классификаторов A должно быть истинным для одного дополнительного выполнения после заднего фронта Q.

Примечание 4 — Доступ к функциональному эквиваленту выходов Q или A функционального блока ACTION_CONTROL снаружи относительно связанного действия определяется разработчиком;

b) Логический вход в блок ACTION_CONTROL для действия должен быть заявлен как связанный с шагом или с блоком действия, если соответствующий классификатор эквивалентен имени входа (N, R, S, L, D, P, P0, P1, SD, DS или SL). Связь заявляется, как активная, если связанный шаг является активным, или если вход связанного блока действия имеет значение BOOL#1. Активные связи действия эквивалентны набору активных связей всех входов с его функциональным блоком ACTION_CONTROL.

Логический вход в блок ACTION_CONTROL должен иметь значение BOOL#1, если он имеет, по меньшей мере, одну активную связь, и значение BOOL#0 в противном случае;

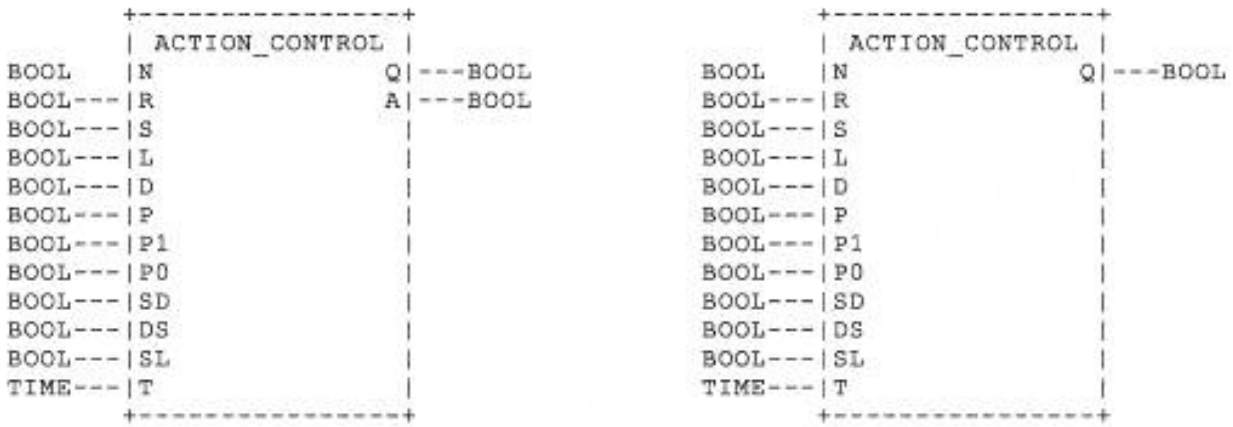
c) Значением входа T в блок ACTION_CONTROL является доля продолжительности связанного со временем классификатора (L, D, SD, DS или SL) активной связи. Если такая связь не существует, значением входа T должно быть t#0s;

d) Ошибка возникает, если выполняется одно или более из следующих условий:

- более одной активной связи действия имеет квалификатор времени (L, D, SD, DS или SL);
- вход SD в блок ACTION_CONTROL имеет значение BOOL#1, когда выход Q1 его блока SL_FF имеет значение BOOL#1;
- вход SL в блок ACTION_CONTROL имеет значение BOOL#1, когда выход Q1 его блока SD_FF имеет значение BOOL#1;

e) Не требуется реализации собственно блока ACTION_CONTROL, а требуется только, чтобы управление действиями было эквивалентно предшествующим правилам. Как показано на рисунке 24, необходимо реализовывать только те части управления действием, которые соответствуют конкретному действию. В частности, следует отметить, что простой функции MOVE (:=) и функции логического OR достаточно для управления действиями логической переменной, если связи последней имеют только классификаторы «N».

На рисунках 22 и 23 приведена сводка интерфейса параметров и тело функционального блока ACTION_CONTROL. На рисунке 24 приведен пример управления действием.

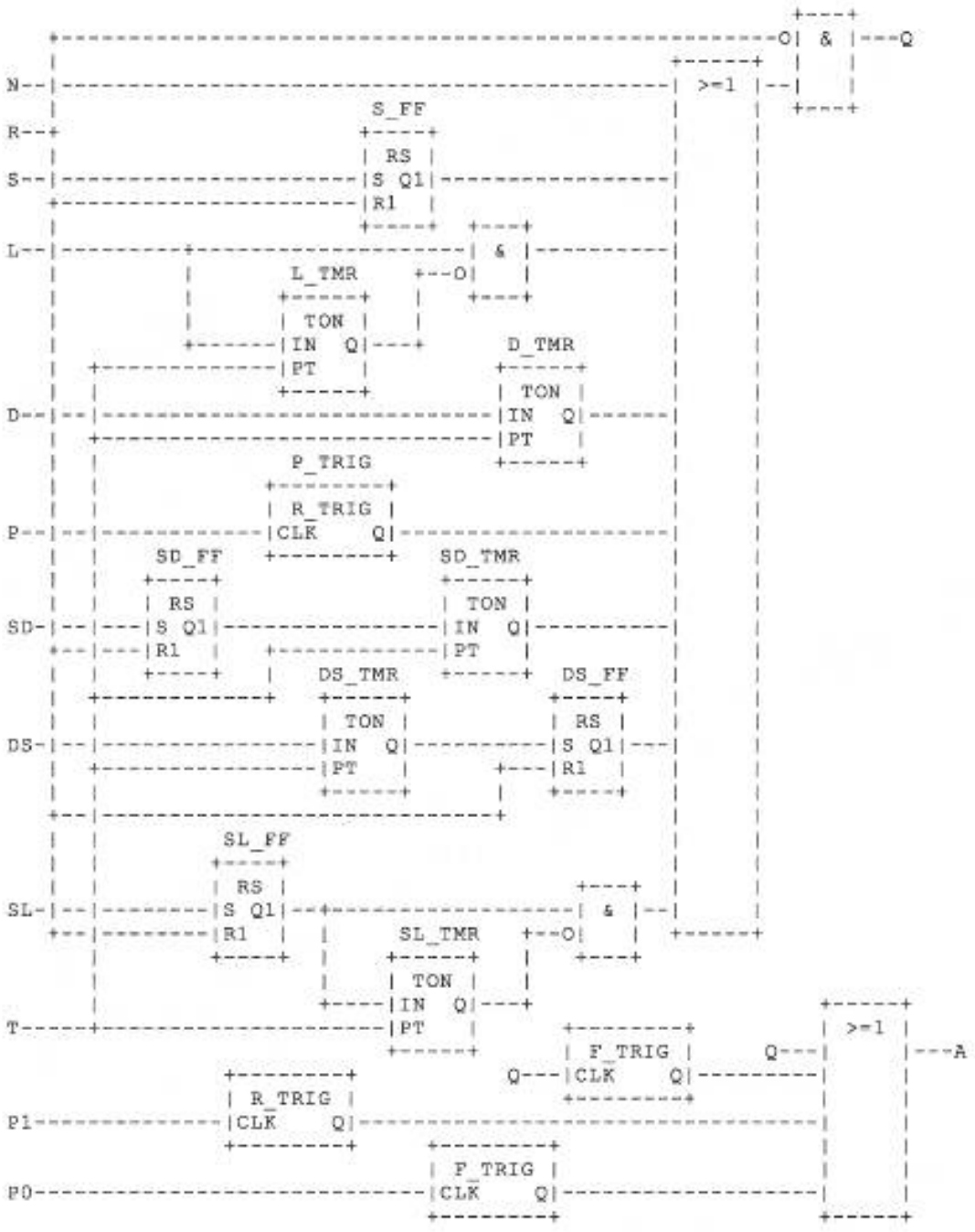


a) С логикой «завершающего сканирования»

b) Без логики «завершающего сканирования»

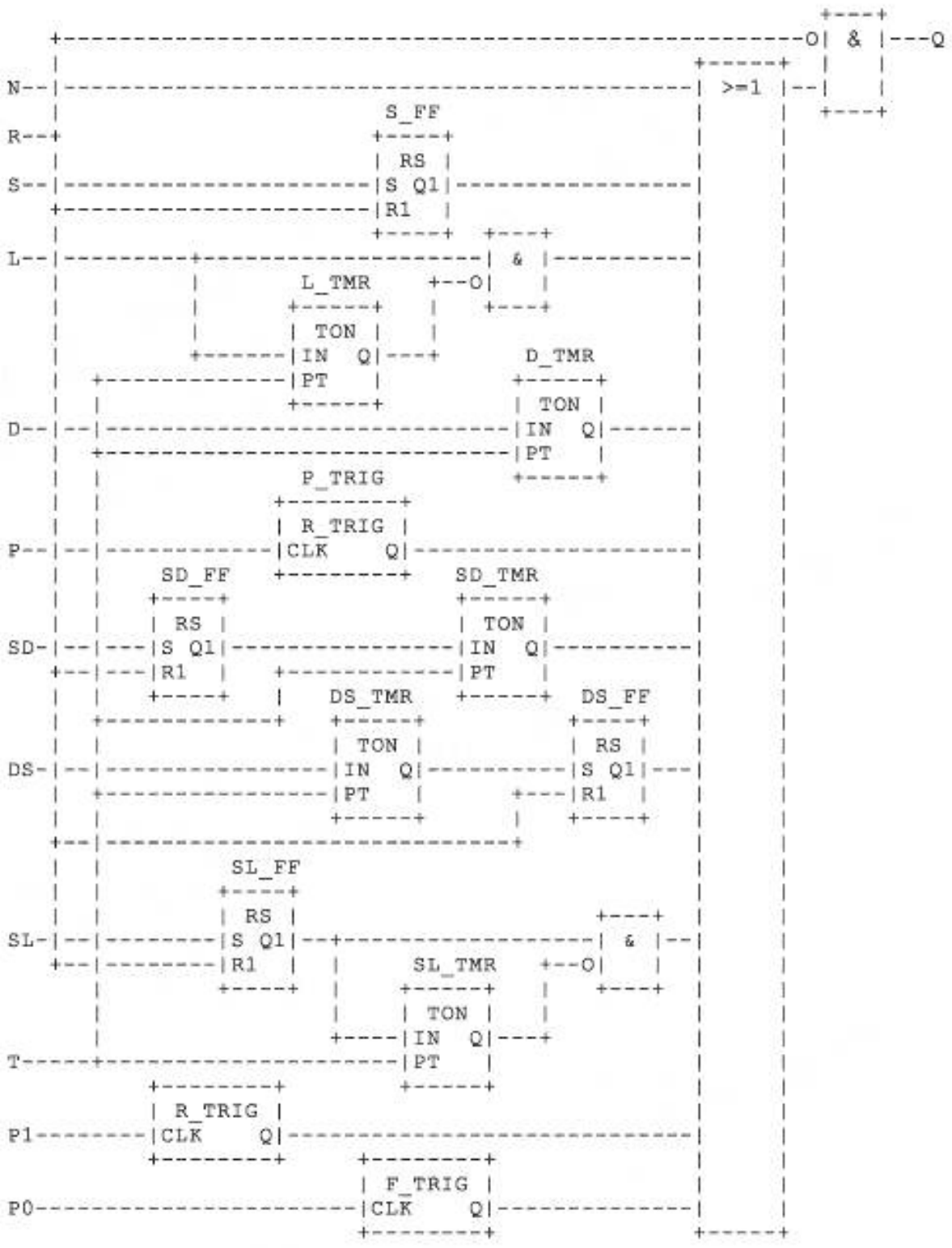
Примечание — Данные интерфейсы невидимы для пользователя.

Рисунок 22 — Функциональный блок ACTION_CONTROL — Внешний интерфейс (сводка)



а) Тело с логикой «завершающего сканирования»

Рисунок 23 — Тело функционального блока ACTION_CONTROL (обзор)

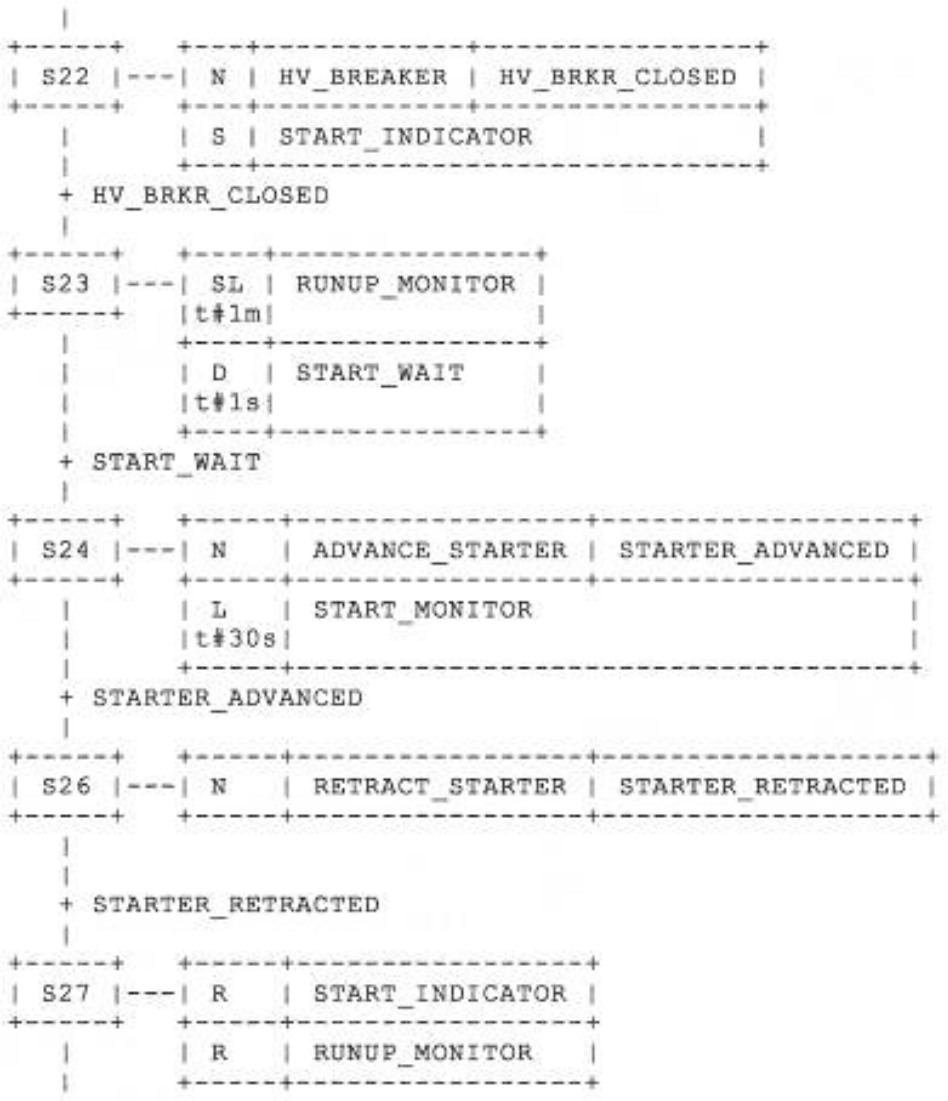


b) Тело без логики «завершающего сканирования»

Примечание 1 — Экземпляры таких типов функционального блока не видимы для пользователя.

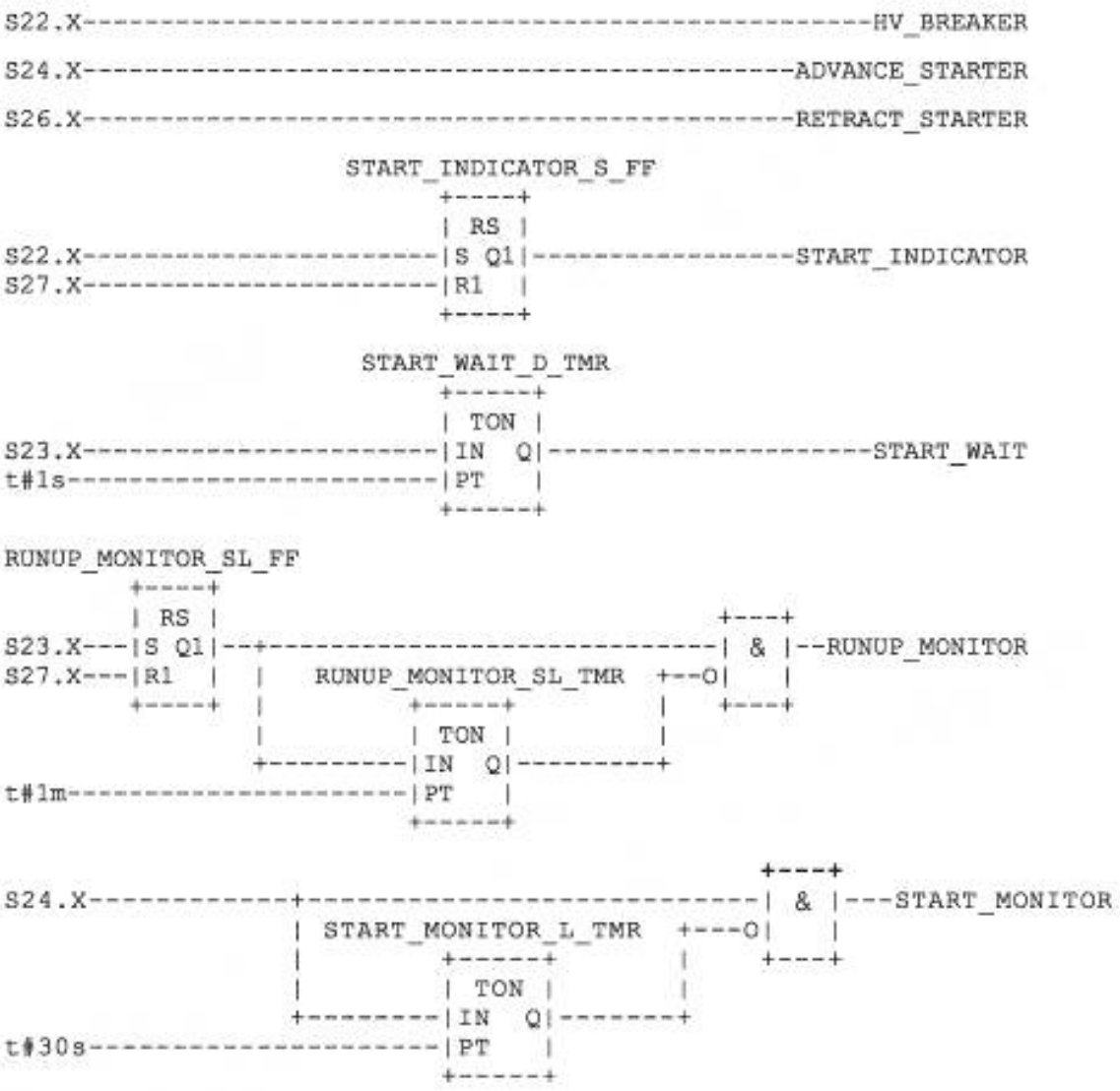
Примечание 2 — Внешние интерфейсы таких типов функционального блока приведены выше.

Рисунок 23



а) Представление SFC

Рисунок 24 — Управление действием (пример)



b) Функциональный эквивалент

Примечание — В данном примере не показана полная сеть SFC и ее соответствующие объявления.

Рисунок 24

Два возможных свойства управления действием приведены в таблице 60.

Таблица 60 — Свойства управлением действием

Номер	Описание	Ссылка
1	С завершающим сканированием	в соответствии с рисунком 22 a) и рисунком 23 a)
2	Без завершающего сканирования	в соответствии с рисунком 22 b) и рисунком 23 b)
Данные свойства являются взаимно исключающими, т. е. в заданной реализации SFC будет поддерживаться только одно из них.		

6.7.5 Правила эволюции

Начальная ситуация сети SFC характеризуется начальным шагом, который находится в активном состоянии при инициализации программы или функционального блока, содержащего сеть.

Эволюции активных состояний шагов должны происходить вдоль направленных связей, когда они вызваны очисткой одного или большего числа переходов.

Переход разрешен, когда все предшествующие шаги, присоединенные к соответствующему символу перехода направленными связями, являются активными. Пересечение перехода происходит, когда переход разрешен и когда соответствующее условие перехода является истинным.

Очистка перехода вызывает деактивацию (или «сброс») всех непосредственно предшествующих шагов, соединенных с соответствующим символом перехода направленными связями, с последующей активацией всех непосредственно последующих шагов.

Изменение шаг/переход и переход/шаг всегда должно поддерживаться в соединениях элементов SFC, то есть:

- два шага никогда не должны быть связаны непосредственно; они всегда должны разделяться переходом;
- два перехода никогда не должны быть связаны непосредственно; они всегда должны разделяться шагом.

Когда очистка перехода приводит к активации нескольких шагов одновременно, то последовательности, к которым принадлежат такие шаги, называются параллельными последовательностями. После их параллельной активации, эволюция таких последовательностей становится независимой. Чтобы подчеркнуть особый характер таких конструкций, дивергенция и конвергенция параллельных последовательностей обозначается двойной горизонтальной линией.

Ошибка возникнет, если имеется возможность того, что неприоритетные переходы в разветвлении выбора, как показано в свойстве 2a таблицы 61, одновременно являются истинными. Пользователь может предпринять меры предосторожности, чтобы избежать этой ошибки, как показано в свойствах 2b и 2c таблицы 61.

Синтаксис и семантика разрешенных комбинаций шагов и переходов определены в таблице 61.

Время очистки перехода теоретически может считаться пренебрежительно малым, но оно никогда не будет равно нулю. На практике время очистки будет обусловлено реализацией программируемого контроллера. По той же причине длительность активности шага никогда не может рассматриваться равной нулю.

Несколько переходов, которые могут быть очищены параллельно, должны очищаться параллельно, в пределах временных ограничений реализации конкретного программируемого контроллера и ограничений по приоритету, определенных в таблице 61.

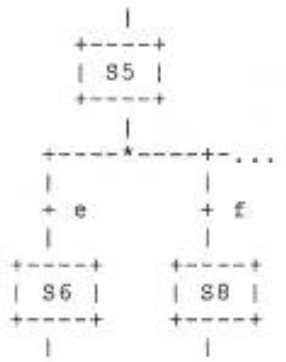
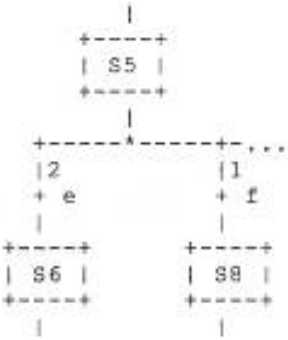
Испытание условия (условий) перехода преемника активного шага не выполняться до тех пор, пока влияния активации шага не распространятся по программному компоненту, в которой описывается шаг.

Рисунок 25 демонстрирует применение данных правил. На этом рисунке активное состояние шага указано присутствием звездочки «*» в соответствующем блоке. Это примечание используется только для иллюстрации, и не является обязательным свойством языка.

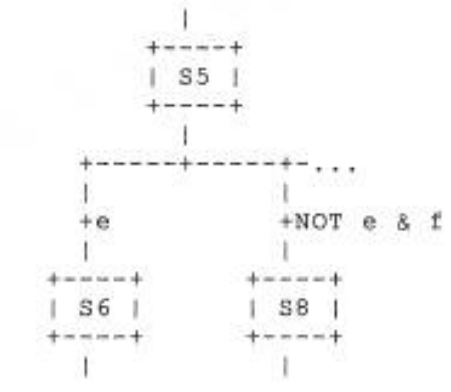
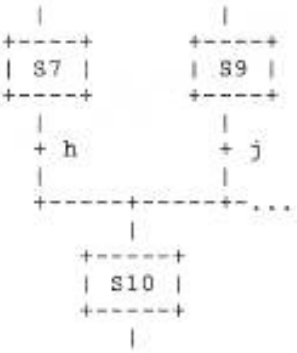
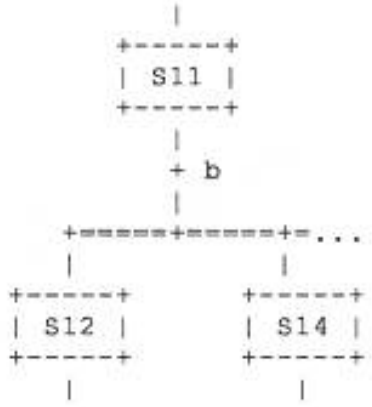
Применение правил, приведенных в этом подразделе, не может предотвратить формулировку «небезопасных» SFC, таких как приведенная на рисунке 26 a), которая может продемонстрировать неконтролируемое распространение маркеров. Аналогично, применение этих правил не может предотвратить формулировку «недосягаемых» SFCs, таких как приведенная на рисунке 26 b), которая может проявлять «замкнутое» поведение. Система программируемого контроллера рассматривает наличие таких условий как ошибку.

Максимально допустимая ширина конструкций «дивергенции» и «конвергенции» в таблице 61 определяется разработчиком.

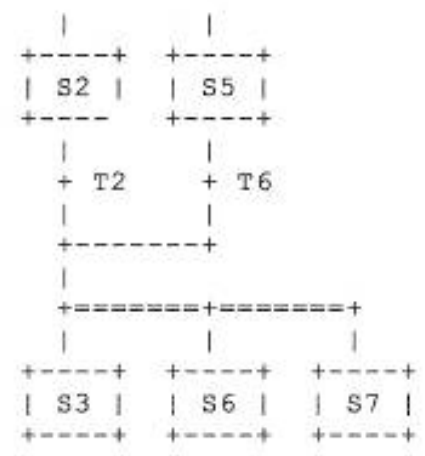
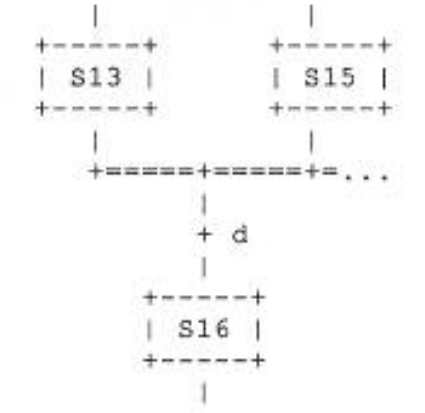
Таблица 61 — Эволюция последовательности (графическая форма)

Номер	Описание	Объяснение	Пример
1	Простая последовательность	Изменение шаг — переход повторяется последовательно	 Эволюция от шага S3 к шагу S4 происходит, если и только если шаг S3 находится в активном состоянии, а условие перехода с равно TRUE
2a	Дивергенция последовательности с приоритетом слева направо	Выбор между несколькими последовательностями представлен как несколько символов перехода под горизонтальной линией, поскольку имеются различные возможные эволюции. Звездочка обозначает приоритет оценок перехода слева направо	 Эволюция происходит от S5 к S6, если S5 активен, а условие перехода «e» равно TRUE (независимо от значения «f»), или от S5 до S8, только если S5 активен, и «f» равно TRUE, а «e» равно FALSE
2b	Дивергенция последовательности с нумерованными ветвями	Звездочка «*», за которой следуют нумерованные ветви, указывает на определяемый пользователем приоритет оценки перехода, причем ветвь с наименьшим номером имеет наивысший приоритет	 Эволюция происходит от S5 к S8, если S5 активен, а условие перехода «f» равно TRUE (независимо от значения «e»), или от S5 к S6, только если S5 активен и «e» равно TRUE, а «f» равно FALSE

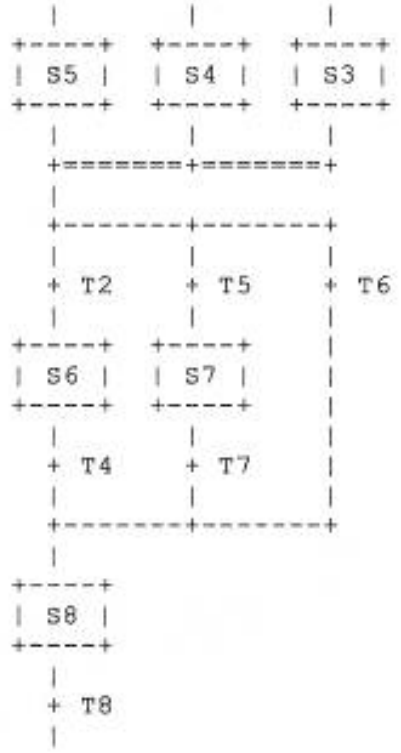
Продолжение таблицы 61

Номер	Описание	Объяснение	Пример
2c	Дивергенция последовательности со взаимным исключением	Присоединение («+») ветви указывает, что пользователь будет гарантировать, что условия перехода являются взаимно исключающими	 Эволюция происходит от S5 к S6, если S5 активен, а условие перехода — это TRUE, или от S5 к S8, только если S5 активен, и «e» равно FALSE, а «f» равно TRUE
3	Конвергенция последовательности	Завершение выбора последовательности представлено как несколько символов перехода над горизонтальной линией, поскольку здесь имеются пути выбора, которые должны быть завершены	 Эволюция происходит от S7 к S10, если S7 активен, а условие перехода «h» равно TRUE, или от S9 к S10, если S9 — активен, а «j» равно TRUE
4a	Параллельная дивергенция после одиночного перехода	Двойной горизонтальной линии синхронизации может предшествовать условие одиночного перехода	 Эволюция происходит от S11 к S12, S14, если S11 активен, а условие перехода «b», связанное с обычным переходом, равно TRUE После параллельной активации S12, S14 и т. д. эволюция каждой последовательности продолжается независимо

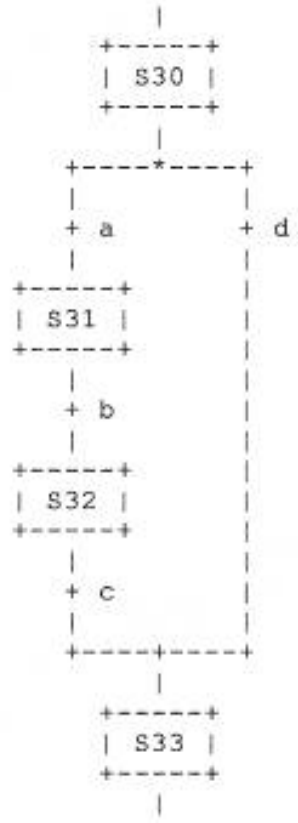
Продолжение таблицы 61

Номер	Описание	Объяснение	Пример
4b	Параллельная дивергенция после конверсии	Двойной горизонтальной линии синхронизации может предшествовать конвергенция выбора последовательности	 Эволюция происходит к шагам S3, S6 и S7, если S2 активен, а переход T2 равен TRUE или S5 активен, а переход T6 равен TRUE
4c	Параллельная конвергенция перед одним переходом	За двойными линиями параллельной конвергенции может следовать одиночный переход	 Эволюция происходит от S13, S15, ... к S16 только в случае, если все приведенные выше шаги, присоединенные к двойной горизонтальной линии, активны, а условие перехода «d», связанное с общим переходом равно TRUE

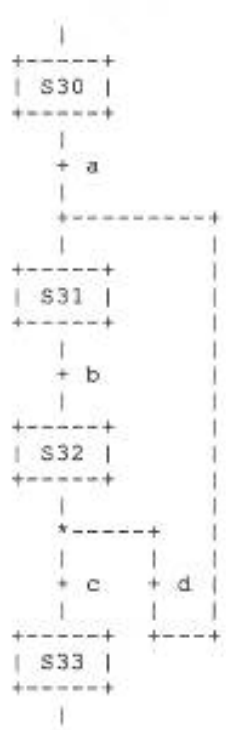
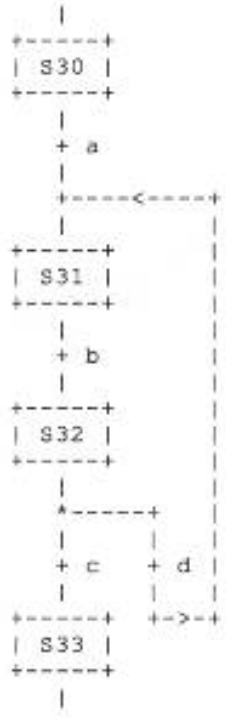
Продолжение таблицы 61

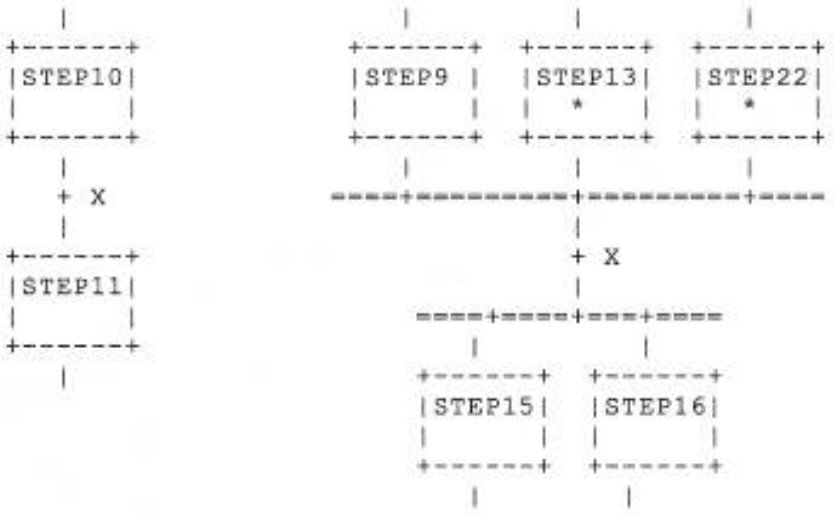
Номер	Описание	Объяснение	Пример
4d	Параллельная конвергенция перед выбором последовательности	За двойными линиями параллельной конвергенции может следовать дивергенция выбора последовательности	 Эволюция происходит от S5, S4 и S3 к одному из шагов S6, S7 или S8, только в случае, если все приведенные выше шаги, присоединенные к двойной горизонтальной линии, активны, а условие перехода T2, T5 или T6 равно TRUE, соответственно

Продолжение таблицы 61

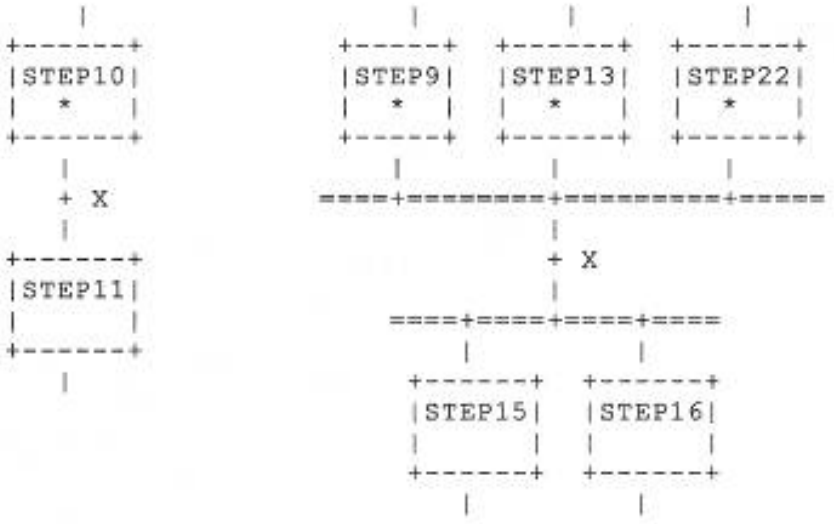
Номер	Описание	Объяснение	Пример
5a, b, c	Пропуск последовательности	«Пропуск последовательности» — это особый случай выбора последовательности, (свойство 2), в которой одна или более ветвей не содержат шагов. Свойства 5a, 5b и 5c соответствуют опциям, заданным в свойствах 2a, 2b представления заданным свойствами 2a, 2b и 2c, соответственно	 (показано свойство 5a) Эволюция происходит от S30 к S33, если «a» равно FALSE, а «d» — это TRUE, равно TRUE, то есть последовательность (S31, S32) будет пропущена

Окончание таблицы 61

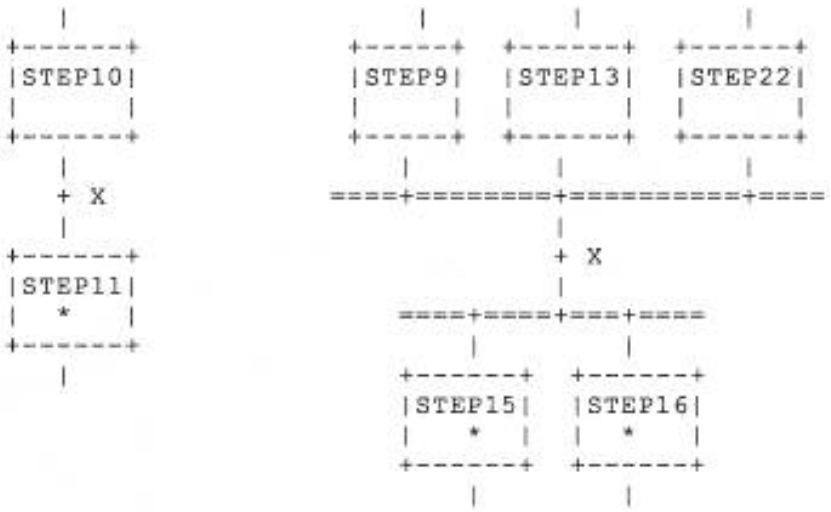
Номер	Описание	Объяснение	Пример
6a, b, c	Цикл последовательности	«Цикл последовательности» — это особый случай выбора последовательности (свойство 2), в которой одна или большее число ветвей возвращается к предшествующему шагу. Свойства 6a, 6b и 6c относятся к опциям представления, заданным в свойствах 2a, 2b и 2c соответственно	 (показано свойство 6a) Эволюция происходит от S32 к S31, если «c» равно FALSE, а «d» равно TRUE, т. е. последовательность (S31, S32) должна быть повторена
7	Стрелки направления	Если это необходимо для ясности, символ «<» набора символов, определенного в 6.1.1, может использоваться для того, чтобы указывать поток управления справа налево, а символ «>», чтобы представлять поток управления слева направо Если используется данное свойство, соответствующий символ должен быть расположен между двумя символами «-», то есть в последовательности символов «-<-» или «->-», как показано в соответствующем примере	



a) Переход не разрешен (см. примечание 2)



b) Переход разрешен, но не очищен (X = 0)



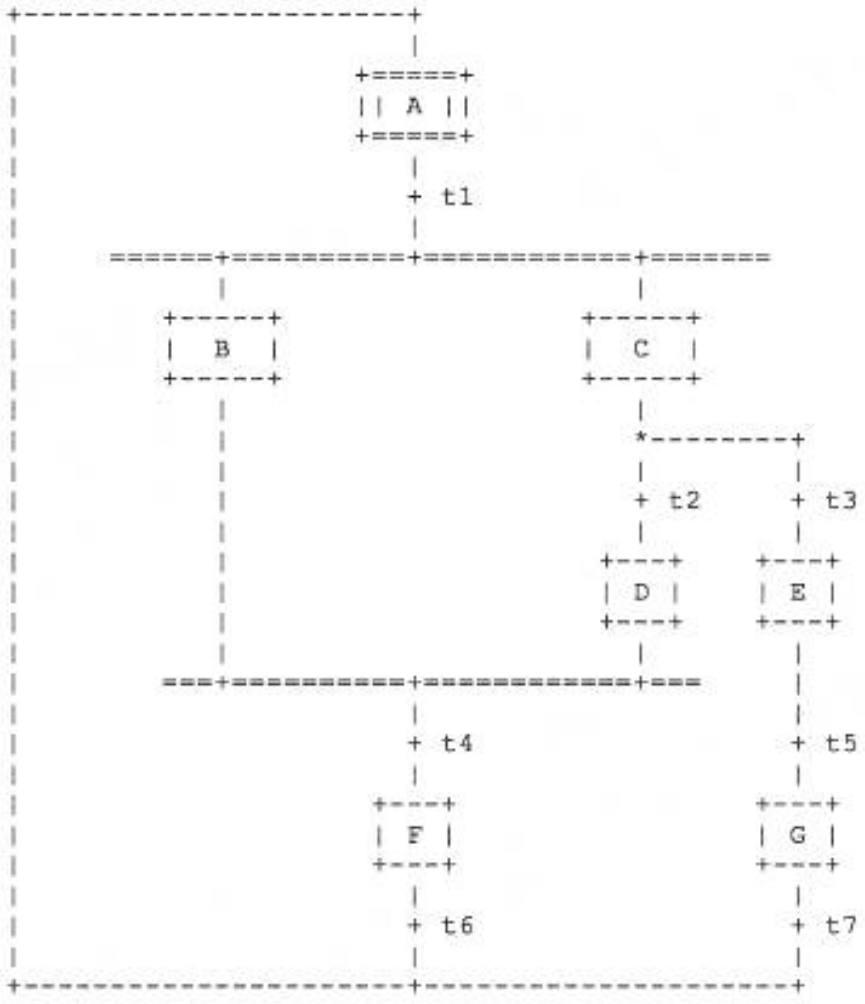
c) Переход очищен (X = 1)

Рисунок 25 — Эволюция SFC (правила), лист 1

Примечание 1 — На данном рисунке активное состояние шага обозначено звездочкой «*» в соответствующем блоке. Данное примечание используется только для иллюстрации и не является обязательным свойством языка.

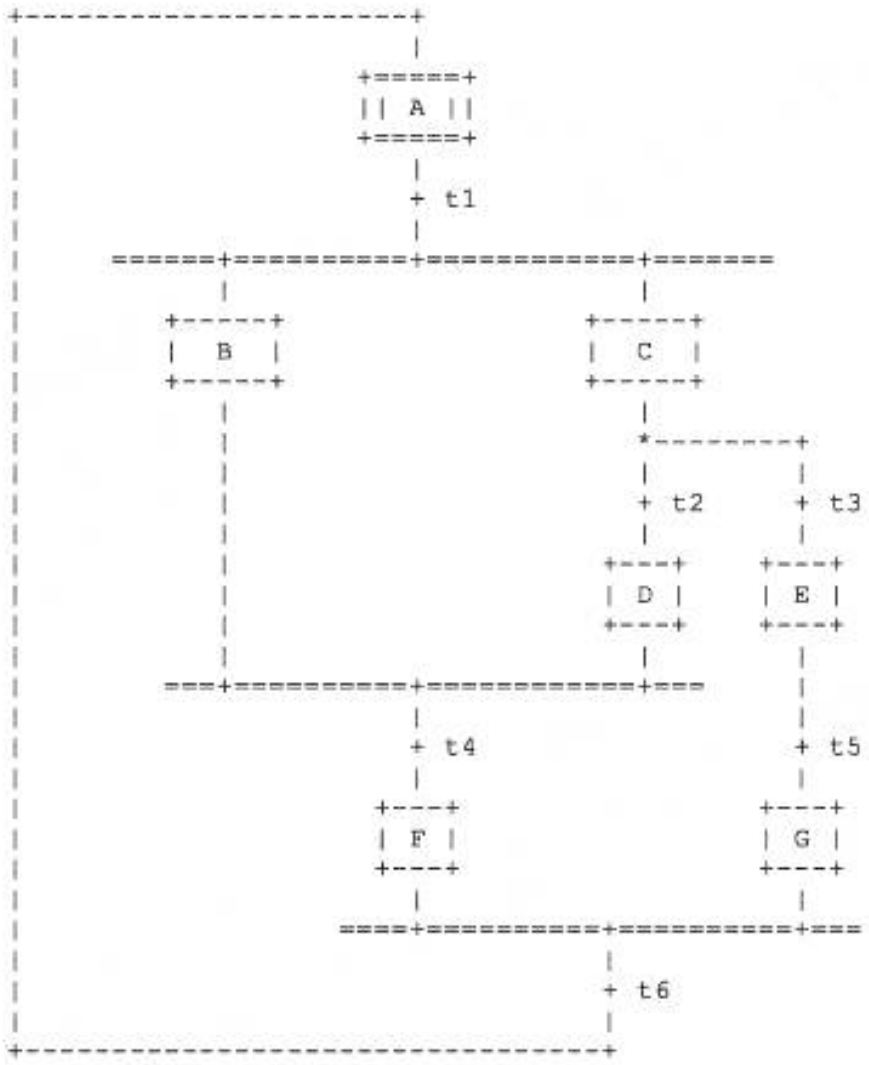
Примечание 2 — В случае а) значение логической переменной X может быть как TRUE, так и FALSE.

Рисунок 25



а) Ошибка SFC: «небезопасная» SFC

Рисунок 26 — Ошибки SFC (пример)



b) Ошибка SFC: «недостижимая SFC»

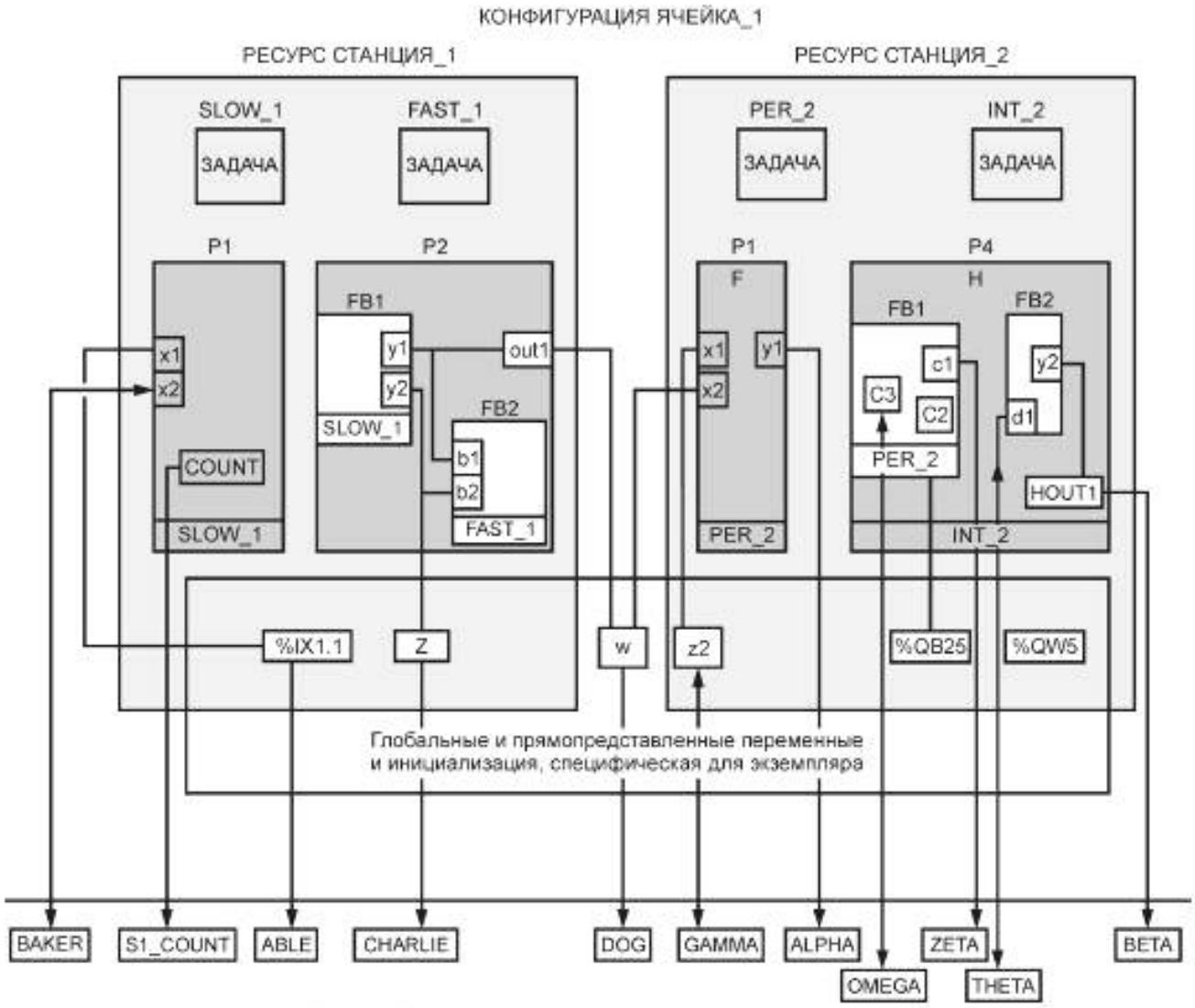
Рисунок 26

6.8 Элементы конфигурации

6.8.1 Общие положения

Конфигурация состоит из ресурсов, задач (которые определены внутри ресурсов), глобальных переменных, путей доступа и специфичных инициализаций экземпляра. Каждый из этих элементов подробно определен в подразделе 6.8.

Графический пример простой конфигурации приведен на рисунке 27. Скелетные описания для соответствующих функциональных блоков и программ приведены на рисунке 27 b). Объявление примера на рисунке 27 приведено на рисунке 28.



а) Графическое представление

```
FUNCTION_BLOCK A
    VAR_OUTPUT
      y1: UINT;
      y2: BYTE;
    END_VAR
END_FUNCTION_BLOCK

FUNCTION_BLOCK B
    VAR_INPUT
      b1: UINT;
      b2: BYTE;
    END_VAR
END_FUNCTION_BLOCK

FUNCTION_BLOCK C
    VAR_OUTPUT
      c1: BOOL;
    END_VAR
    VAR
      C2 AT %Q*: BYTE;
      C3: INT;
    END_VAR
END_FUNCTION_BLOCK

FUNCTION_BLOCK D
    VAR_INPUT
      d1: BOOL;
    END_VAR
    VAR_OUTPUT
      y2: INT;
    END_VAR
END_FUNCTION_BLOCK
```

Рисунок 27 — Конфигурация (пример), лист 1

```
PROGRAM F
  VAR_INPUT
      x1: BOOL;
      x2: UINT;
  END_VAR
  VAR_OUTPUT
      y1: BYTE;
  END_VAR
  VAR
      COUNT: INT;
      TIME1: TON;
  END_VAR
  END_PROGRAM
PROGRAM G
  VAR_OUTPUT
      out1: UINT;
  END_VAR
  VAR_EXTERNAL
      z1: BYTE;
  END_VAR
  VAR
      FB1: A;
      FB2: B;
  END_VAR
  FB1(...);
      out1:= FB1.y1;
      z1:= FB1.y2;
      FB2(b1:= FB1.y1, b2:= FB1.y2);
END_PROGRAM

PROGRAM H
  VAR_OUTPUT
      HOUT1: INT;
  END_VAR
  VAR
      FB1: C;
      FB2: D;
  END_VAR
  FB1(...);
  FB2(...);
  HOUT1:= FB2.y2;
END_PROGRAM
```

b) Объявления скелетного функционального блока и программы

Рисунок 27, лист 2

В таблице 62 перечислены свойства языка для объявления конфигураций, ресурсов, глобальных переменных, путей доступа и специфических инициализаций экземпляра.

- Задачи

Рисунок 27 представляет примеры свойств TASK, соответствующих конфигурации экземпляра, приведенной на рисунке 27 a) и поддерживающих описания, приведенные на рисунке 27 b).

- Ресурсы

Классификатор ON в конструкции RESOURCE...ON...END_RESOURCE используется, чтобы указать тип «функции обработки информации» и ее функций «интерфейса человек-машина» и «интерфейса датчика и привода», на основе которых реализуется ресурс и его связанные программы и задачи. Разработчик обеспечивает библиотеку ресурсов таких элементов, как показанные на рисунке 3. С каждым элементом этой библиотеки связан идентификатор (имя типа ресурса) для использования в объявлении ресурса.

Примечание 1 — Конструкция RESOURCE...ON...END_RESOURCE в конфигурации с одним ресурсом не требуется.

- Глобальные переменные

Область действия секции VAR_GLOBAL ограничивается конфигурацией или ресурсом, в котором она описана, за исключением того, что путь доступа может быть объявлен для глобальной переменной в ресурсе с использованием свойства 10d из таблицы 62.

- Пути доступа

Конструкция VAR_ACCESS...END_VAR предоставляет средства задания имен переменных, которые можно использовать для дистанционного доступа некоторыми сервисами связи, определяемыми в МЭК 61131-5. Путь доступа связывает каждое такое имя переменной с глобальной переменной, прямо представленной переменной или любой входной, выходной или внутренней переменной программы или функционального блока.

Связь должна сопровождаться определением имени переменной с полной иерархической конкатенацией имен экземпляра, начиная с имени ресурса (если имеется), за которым следует имя экземпляра программы (если имеется), за которым следует имя (имена) экземпляра (экземпляров) функционального блока (если имеется). Имя переменной связывается в конце цепочки. Все имена в конкатенации должны быть разделены точками. Если такая переменная — это многоэлементная переменная (структура или массив), то путь доступа также может быть задан для элемента переменной.

Не должно быть возможно определить пути доступа к переменным, объявленным в секциях VAR_TEMP, VAR_EXTERNAL или VAR_IN_OUT.

Направление пути доступа задается как READ_WRITE или READ_ONLY, указывая, что сервисы связи могут как считывать, так и изменять значение переменной в первом случае, или только считывать, но не изменять значение во втором случае. Если направление не задано, то направление по умолчанию — READ_ONLY.

Доступ к переменным, объявленным как CONSTANT, и к входам функционального блока, которые внешне соединены с другими переменными, — READ_ONLY.

Примечание 2 — Эффект использования доступа READ_WRITE для выходных переменных функционального блока определяется разработчиком.

- Конфигурации

Конструкция VAR_CONFIG...END_VAR предоставляет средства для присваивания конкретных расположений, зависящих от экземпляра, символически представленным переменным, которые назначены для соответствующей цели, с использованием отметки звездочки «*», или чтобы присваивания начальных значений, специфических для экземпляра, символически представленным переменным, или и для того и для другого.

Присваивание должно сопровождаться определением имени объекта для расположения или инициализации с полной иерархической конкатенацией имен экземпляров, начиная с имени ресурса (если имеется), за которым следует имя экземпляра программы (если имеется), за которым следует имя (имена) экземпляра (экземпляров) функционального блока (если имеется). Имя переменной для расположения или инициализации присоединяется в конце цепочки, за которой следует имя компонента структуры (если переменная структурирована). Все имена в конкатенации должны быть разделены точками. Присваивание расположения или присваивание начального значения следуют синтаксису и семантике.

Специфические для экземпляра начальные значения, предоставляемые конструкцией VAR_CONFIG...END_VAR, всегда имеют приоритет над специфическими для типа начальными значениями. Нельзя определять специфические для экземпляра инициализации для переменных, которые определены в объявлениях VAR_TEMP, VAR_EXTERNAL, VAR CONSTANT или VAR_IN_OUT.

Таблица 62 — Конфигурация и объявление ресурса

Номер	Описание
1	CONFIGURATION...END_CONFIGURATION
2	VAR_GLOBAL...END_VAR внутри CONFIGURATION
3	RESOURCE...ON...END_RESOURCE
4	VAR_GLOBAL...END_VAR внутри RESOURCE
5a	Периодическая TASK
5b	Непериодическая TASK
6a	WITH для PROGRAM со связанными задачами TASK
6b	WITH для FUNCTION_BLOCK со связанными задачами TASK
6c	PROGRAM без связанных задач TASK
7	Прямо представленные переменные в VAR_GLOBAL
8a	Соединение прямо представленных переменных со входами PROGRAM
8b	Соединение переменных GLOBAL со входами PROGRAM
9a	Соединение выходов PROGRAM с прямо представленными переменными
9b	Соединение выходов PROGRAM с переменными GLOBAL
10a	VAR_ACCESS...END_VAR
10b	Пути доступа к прямо представленным переменным
10c	Пути доступа к входам PROGRAM
10d	Пути доступа к переменным GLOBAL в RESOURCEs
10e	Пути доступа к переменным GLOBAL в CONFIGURATIONs
10f	Пути доступа к выходам PROGRAM
10g	Пути доступа к внутренним переменным PROGRAM
10h	Пути доступа к входам функционального блока
10i	Пути доступа к выходам функционального блока
11a	VAR_CONFIG...END_VAR к переменным Данное свойство поддерживается, если поддерживается свойство «частичное определение» с символом «*» в таблице 16
11b	VAR_CONFIG...END_VAR для компонент структур
12a	VAR_GLOBAL CONSTANT в RESOURCE
12b	VAR_GLOBAL CONSTANT в CONFIGURATION
13a	VAR_EXTERNAL в RESOURCE
13b	VAR_EXTERNAL CONSTANT в RESOURCE

На следующем рисунке приведено объявление примера на рисунке 27.

Код программы	использует свойство таблицы 62
CONFIGURATION CELL_1	1
VAR_GLOBAL w: UINT; END_VAR	2
RESOURCE STATION_1 ON PROCESSOR_TYPE_1	3
VAR_GLOBAL z1: BYTE; END_VAR 4	4
TASK SLOW_1(INTERVAL:= t#20ms, PRIORITY:= 2);	5a
TASK FAST_1(INTERVAL:= t#10ms, PRIORITY:= 1);	5a
PROGRAM P1 WITH SLOW_1:	6a
F(x1:= %IX1.1);	8a
PROGRAM P2: G(OUT1 => w,	9b
FB1 WITH SLOW_1,	6b
FB2 WITH FAST_1);	6b
END_RESOURCE	3
RESOURCE STATION_2 ON PROCESSOR_TYPE_2	3
VAR_GLOBAL z2 : BOOL;	4
AT %QW5: INT ;	7
END_VAR	4
TASK PER_2(INTERVAL:= t#50ms, PRIORITY:= 2);	5a
TASK INT_2(SINGLE:= z2, PRIORITY:= 1);	5b
PROGRAM P1 WITH PER_2:	6a
F(x1:= z2, x2:= w);	8b
PROGRAM P4 WITH INT_2:	6a
H(HOUT1 => %QW5,	9a
FB1 WITH PER_2);	6b
END_RESOURCE	3

Рисунок 28 — Описание CONFIGURATION и RESOURCE (пример), лист 1

VAR_ACCESS	10a
ABLE : STATION_1.%IX1.1 : BOOL READ_ONLY;	10b
BAKER : STATION_1.P1.x2 : UINT READ_WRITE;	10c
CHARLIE : STATION_1.z1 : BYTE;	10d
DOG : w : UINT READ_ONLY;	10e
ALPHA : STATION_2.P1.y1 : BYTE READ_ONLY;	10f
BETA : STATION_2.P4.HOUT1 : INT READ_ONLY;	10f
GAMMA : STATION_2.z2 : BOOL READ_WRITE;	10d
S1_COUNT : STATION_1.P1.COUNT : INT;	10g
THETA : STATION_2.P4.FB2.d1 : BOOL READ_WRITE;	10h
ZETA : STATION_2.P4.FB1.c1 : BOOL READ_ONLY;	10i
OMEGA : STATION_2.P4.FB1.C3 : INT READ_WRITE;	10k
END_VAR	10a
VAR_CONFIG	11
STATION_1.P1.COUNT: INT:= 1;	
STATION_2.P1.COUNT: INT:= 100;	
STATION_1.P1.TIME1: TON:= (PT:= T#2.5s);	
STATION_2.P1.TIME1: TON:= (PT:= T#4.5s);	
STATION_2.P4.FB1.C2 AT %QB25: BYTE;	
END_VAR	
END_CONFIGURATION	1

Примечание 1 — Графическое и полуграфическое представление таких свойств допускается, но не входит в задачу настоящего стандарта.

Примечание 2 — Ошибка, если тип данных, объявленный в операторе VAR_ACCESS отличается от типа данных, объявленного для переменной в другом месте, например, если переменная BAKER объявлена как WORD в приведенных выше примерах.

Рисунок 28, лист 2

6.8.2 Задачи

Для целей настоящего стандарта задача определяется как элемент управления выполнением, который способен вызывать, как на периодической основе, так и при появлении переднего фронта заданной логической переменной, выполнение набора программных компонентов, которые могут включать программы и функциональные блоки, экземпляры которых заданы в объявлении программ.

Максимальное число задач на ресурс и допустимый интервал между задачами определяются разработчиком.

Задачи и их связь с программными компонентами может быть представлена графически или текстуально с использованием конструкции WITH как показано в таблице 63, в виде части ресурсов внутри конфигураций. Задача неявно разрешается или блокируется связанным с ней ресурсом в соответствии с механизмами. Управление программными компонентами при разрешенных задачах подчиняется следующим правилам:

a) Связанные программные компоненты должны быть спланированы для выполнения при каждом переднем фронте на входного параметра SINGLE задачи;

b) Если входной параметр INTERVAL — ненулевой, то связанные программные компоненты назначаются для периодического выполнения через заданный интервал времени, пока входной параметр SINGLE остается нулевым (0). Если входной параметр INTERVAL равен нулю (значение по умолчанию), периодическое выполнение связанных программных компонентов происходить не будет;

c) Входной параметр PRIORITY задачи устанавливает приоритет планирования связанных программных модулей, где нуль (0) имеет наивысший приоритет, а более низкие приоритеты имеют последовательно большие цифровые значения. Как показано в таблице 63, приоритет программного компонента (т. е. приоритет связанной с ним задачи) может использоваться для планирования с приоритетами или без приоритетов:

- в планировании без приоритетов вычислительные возможности становятся доступными на ресурсе, когда завершается выполнение программного компонента или функции операционной системы. Когда вычислительные возможности доступны, программный компонент с наивысшим плановым приоритетом начинает выполнение. Если в ожидании имеется более одного программного компонента с наивысшим плановым приоритетом, то будет выполняться программный компонент с наибольшим временем ожидания и наивысшим плановым приоритетом;
- в планировании с приоритетом, когда программный компонент назначен, он прерывает выполнение программного компонента с более низким приоритетом на том же ресурсе, то есть выполнение компонента с более низким приоритетом может быть задержано до завершения выполнения компонента с более высоким приоритетом. Программный компонент не прерывает выполнение другого компонента с таким же или более высоким приоритетом. В зависимости от плановых приоритетов, программный компонент может не начать выполнение в спланированный момент. Однако в примерах, приведенных в таблице 63, все программные компоненты завершают работу в срок, то есть они заканчивают выполнение до того, как будут спланированы для повторного выполнения. Разработчик предоставляет информацию, позволяющую пользователю определить, должны ли выдерживаться все сроки выполнения в предлагаемой конфигурации;

d) Программа без связанной задачи будет иметь самый низкий приоритет в системе. Любая такая программа должна быть спланирована для выполнения после «пуска» ее ресурса и должна быть перепланирована для выполнения, как только ее выполнение заканчивается;

e) Когда экземпляр функционального блока связан с задачей, его выполнение должно происходить под исключительным управлением задачи, независимо от правил оценки программного компонента, в котором объявлен связанный с задачей функциональный блок;

f) Экземпляр функционального блока, который не прямо связан с задачей, будет следовать обычным правилам для порядка оценки элементов языка для программного компонента (который сам может находиться под управлением задачи), в котором объявлен экземпляр функционального блока.

Примечание 1 — Экземпляры класса не могут иметь связанной задачи.

Примечание 2 — Методы функционального блока или класса выполняются в программном компоненте, который они вызывают;

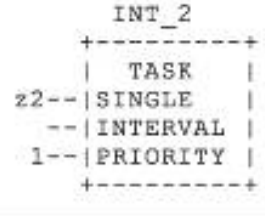
g) Выполнение функциональных блоков внутри программы должно быть синхронизировано, чтобы обеспечить достижение параллельности доступа к данным в соответствии со следующими правилами:

- если функциональный блок получает более одного входного параметра от другого функционального блока, то когда первый FB выполняется, все входные параметра последнего должны представлять результаты той же оценки;
- если один или более функциональных блоков получает входные параметры от одного и того же функционального блока, и если все «целевые» блоки явно или неявно связаны с одной и той же задачей, тогда все входы на все такие «целевые» блоки во время их оценки будут представлять результаты одной и той же оценки «исходного» блока.

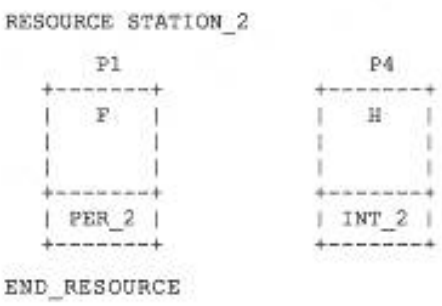
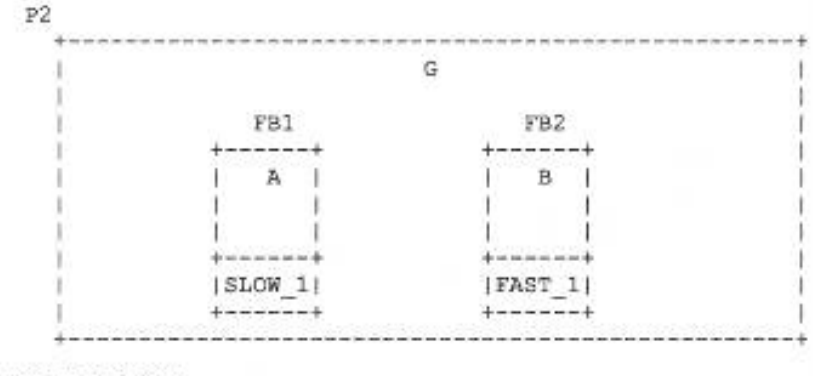
Необходимо обеспечить меры для сохранения выходных параметров функций или функциональных блоков, которые явно связаны с задачей, или которые используются как входные параметры в программные компоненты, имеющие явные связи с задачей, как необходимые для удовлетворения приведенных выше правил.

Ошибка возникает, если задача не может быть спланирована или удовлетворить заданному сроку ее выполнения вследствие чрезмерных требований к ресурсу или других конфликтов планирования задачи.

Таблица 63 — Задача

Номер	Описание	Примеры
1a	Текстовое объявление периодической задачи TASK	(свойство 5a таблицы 62)
1b	Текстовое объявление непериодической задачи TASK	(свойство 5b таблицы 62)
	Графическое представление TASK (общая форма)	
2a	Графическое представление периодической TASK (с INTERVAL)	
2b	Графическое представление непериодической TASK (с SINGLE)	
3a	Текстовая связь с PROGRAMs	(свойство 6a таблицы 62)
3b	Текстовая связь с функциональными блоками	(свойство 6b таблицы 62)

Окончание таблицы 63

Номер	Описание	Примеры
4a	Графическое представление с PROGRAM	
4b	Графическая связь с функциональными блоками внутри PROGRAMs	
5a	Планирование без приоритетов	См. рисунок 28
5b	Планирование с приоритетами	См. рисунок 28

Примечание 1 — Подробности объявлений RESOURCE и PROGRAM не показаны.

Примечание 2 — Обозначение X@Y указывает, что программный компонент X спланирован или выполняется с приоритетом Y.

Следующие примеры показывают планирование без приоритетов и с приоритетами, определяемое в таблице 63 свойствами 5a и 5b.

Пример 1 — Планирование без приоритетов и с приоритетами

1 Планирование без приоритетов

- Ресурс STATION_1 как сконфигурирован на рисунке 28
- Времена исполнения: P1 = 2 мс; P2 = 8 мс
- P2.FB1 = P2.FB2 = 2 мс (см. примечание 1)
- STATION_1 запускается при t = 0

Планирование (повторяется каждые 40 мс)		
t(мс)	**Выполнение**	**Ожидание**
0	P2.FB2@1	P1@2, P2.FB1@2, P2
2	P1@2	P2.FB1@2, P2
4	P2.FB1@2	P2
6	P2	—
10	P2	P2.FB2@1
14	P2.FB2@1	P2
16	P2	(перезапуск P2)
20	P2	P2.FB2@1, P1@2, P2.FB1@2
24	P2.FB2@1	P1@2, P2.FB1@2, P2
26	P1@2	P2.FB1@2, P2
28	P2.FB1@2	P2
30	P2.FB2@1	P2
32	P2	—
40	P2.FB2@1	P1@2, P2.FB1@2, P2
- Ресурс STATION_2 как сконфигурирован на рисунке 28 - Времена исполнения: P1 = 30 мс, P4 = 5 мс, P4.FB1 = 10 мс - INT_2 срабатывает при t = 25, 50, 90,... мс - STATION_2 запускается при t= 0		
План		
t(мс)	**Исполнение**	**Ожидание**
0	P1@2	P4.FB1@2
25	P1@2	P4.FB1@2, P4@1
30	P4@1	P4.FB1@2
35	P4.FB1@2	—
50	P4@1	P1@2, P4.FB1@2
55	P1@2	P4.FB1@2
85	P4.FB1@2	—
90	P4.FB1@2	P4@1
95	P4@1	—
100	P1@2	P4.FB1@2

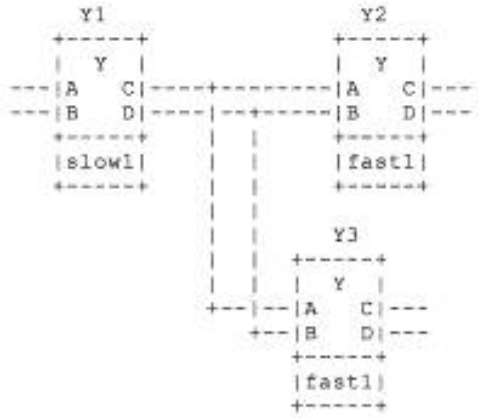
2 Планирование с приоритетами	См. таблицу 63, 5b
- Ресурс STATION_1 как сконфигурирован на рисунке 28 - Времена исполнения: P1 = 2 мс; P2 = 8 мс; P2.FB1 = P2.FB2 = 2 мс - STATION_2 запускается при t= 0	

План		
t(мс)	Исполнение	Ожидание
0	P2.FB2@1	P1@2, P2.FB1@2, P2
2	P1@2	P2.FB1@2, P2
4	P2.FB1@2	P2
6	P2	—
10	P2.FB2@1	P2
12	P2	—
16	P2	(перезапуск P2)
20	P2.FB2@1	P1@2, P2.FB1@2, P2
- Ресурс STATION_2 как сконфигурирован на рисунке 28 - Времена исполнения: P1 = 30 мс, P4 = 5 мс, P4.FB1 = 10 мс - INT_2 срабатывает при t = 25, 50, 90,... мс - STATION_2 запускается при t = 0		
План		
t(мс)	Исполнение	Ожидание
0	P1@2	P4.FB1@2
25	P4@1	P1@2, P4.FB1@2
30	P1@2	P4.FB1@2
35	P4.FB1@2	—
50	P4@1	P1@2, P4.FB1@2
55	P1@2	P4.FB1@2
85	P4.FB1@2	—
90	P4@1	P4.FB1@2
95	P4.FB1@2	—
100	P1@2	P4.FB1@2
Примечание 1 — Времена выполнения P2.FB1 и P2.FB2 не включены во время выполнения P2. Примечание 2 — Время выполнения P4.FB1 не включено во время выполнения P4.		

Пример 2 — Связи задачи с экземплярами функционального блока

RESOURCE R1

PROGRAM X

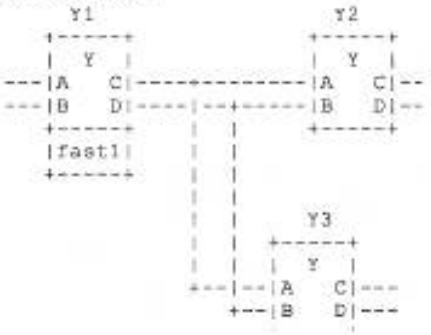


END_PROGRAM

a) Функциональные блоки с явными связями задачи

P1

PROGRAM X



END_PROGRAM

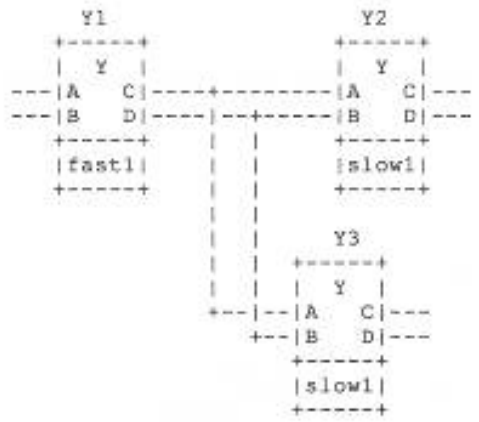
slow1

b) Функциональные блоки с неявными связями задачи

RESOURCE R1

P1

PROGRAM X



END_PROGRAM

c) Явные связи задачи эквивалентны b)

Примечание 3 — Графические представления в этих примерах являются только иллюстративными, но не нормативными.

6.9 Пространства имен

6.9.1 Общие положения

Для целей функционирования языков программирования программируемого контроллера пространство имен — это элемент языка, объединяющий другие элементы языка в общий объект.

Одно и то же имя элемента языка, объявленное внутри пространства имен, может также использоваться внутри других пространств имен.

Пространства имен и типы, не имеющие охватывающего пространства имен, являются членами глобального пространства имен. Глобальное пространство имен включает имена, описанные в глобальной области видимости. Все стандартные функции и функциональные блоки являются элементами глобального пространства имен.

Пространства имен могут быть вложенными.

Пространства имен и типы, объявленные внутри пространства имен, являются членами этого пространства имен. Члены пространства имен находятся в локальной области видимости пространства имен.

С пространствами имен может быть реализована концепция библиотеки наряду с концепцией модулей. Пространства имен можно использовать, чтобы избежать неоднозначностей идентификатора. Типовое приложение пространства имен находится в контексте средств объектно-ориентированного программирования.

6.9.2 Объявление

Описание пространства имен начинается с ключевого слова NAMESPACE, за которым опционально следует спецификатор доступа INTERNAL, имя пространства имен и окончания с ключевым словом END_NAMESPACE. Пространство имен содержит набор элементов языка, за каждым из которых опционально следует следующий спецификатор доступа:

- INTERNAL для доступа только внутри собственно пространства имен.

Спецификатор доступа может быть применен к описанию следующих элементов языка:

- определяемых пользователем типов данных — с использованием ключевого слова TYPE;
- функций;
- программ;
- типов функциональных блоков и их переменных и методов;
- классов и их переменных и методов;
- интерфейсов;
- пространств имен.

Если спецификатор доступа не задан, элементы языка пространства имен доступны извне пространства имен, т.е. пространство имен является общедоступным по умолчанию.

Примеры 1 и 2 показывают объявление пространства имен и объявление вложенного пространства имен.

Пример 1 — Объявление пространства имен

```
NAMESPACE Timers
    FUNCTION INTERNAL TimeTicks DWORD
    // ...объявление и операции здесь
    END_FUNCTION
    // другие элементы пространства имен без спецификатора являются PUBLIC по умолчанию
    TYPE по умолчанию
      LOCAL_TIME:   STRUCT
        TIMEZONE:    STRING [40];
        DST:  BOOL;  // Декретное время
        TOD:  TOD;
        END_STRUCT;
    END_TYPE;
    ...
FUNCTION_BLOCK TON
// ... объявление и операции
END_FUNCTION_BLOCK
...
FUNCTION_BLOCK TOF
// ... объявление и операции
END_FUNCTION_BLOCK
END_NAMESPACE (*Timers*)
```

Пример 2 — Объявление вложенного пространства имен

```
NAMESPACE Standard   // Пространство имен = PUBLIC по умолчанию
    NAMESPACE Timers          // Пространство имен= PUBLIC по умолчанию
      FUNCTION INTERNAL TimeTicds DWORD
      // ...объявление и операции здесь
      END_FUNCTION

    // другие элементы пространства имен без спецификатора являются PUBLIC за счет
    TYPE по умолчанию
    LOCAL_TIME:
    STRUCT TIMEZONE: STRING [40];
    DST:        BOOL;  // Декретное время
    TOD:        TOD;
    END_STRUCT;
    END_TYPE;
    ...
    FUNCTION_BLOCK TON    // определяет реализацию TON с новым именем
    // ... объявление и операции здесь
    END_FUNCTION_BLOCK
    ...
    FUNCTION_BLOCK TOF    // определяет реализацию TOF с новым именем
    // ... объявление и операции здесь END_FUNCTION_BLOCK

    CLASS A
    ETHOD INTERNAL M1
    ...
```

```
    END_METHOD
    METHOD PUBLIC M2 // PUBLIC задано здесь для замены PROTECTED по умолчанию
    ...
    END_METHOD
    END_CLASS
    CLASS INTERNAL B
    METHOD INTERNAL M1
    ...
    END_METHOD
    METHOD PUBLIC M2
    ...
    END_METHOD
    END_CLASS

END_NAMESPACE (*Таймеры*) NAMESPACE счетчики
    FUNCTION_BLOCK CUP
    // ... объявление и операции здесь
    END_FUNCTION_BLOCK
    ...
    FUNCTION_BLOCK CDOWN
    // ... объявление и операции здесь
    END_FUNCTION_BLOCK
    END_NAMESPACE (*Timers*)
END_NAMESPACE (*Стандарт*)
```

Доступность элементов пространства имен, методов и переменных функциональных блоков внутри и извне пространства имен зависит от спецификаторов доступа переменной или метода вместе со спецификатором пространства имен при объявлении пространства имен и элементов языка.

Правила доступности суммированы на рисунке 29.

Спецификатор пространства имен	Общедоступно (по умолчанию, спецификатор отсутствует)		INTERNAL		
Спецификатор доступа R элементу языка, переменной или методу	Доступ извне пространства имен	Доступ изнутри пространства имен, но извне программного компонента	Доступ извне пространства имен		Доступ изнутри пространства имен, но извне программного компонента
			Все пространства имен, кроме предка	Предок пространства имен	
PRIVATE	Нет	Нет	Нет	Нет	Нет
PROTECTED	Нет	Нет	Нет	Нет	Нет
INTERNAL	Нет	Да	Нет	Нет	Да
PUBLIC	Да	Да	Нет	Да	Да

Рисунок 29 — Доступность с использованием пространств имен (правила)

В случае иерархических пространств имен, внешнее пространство имен может дополнительно ограничивать доступ; оно может не разрешать дополнительный доступ к объектам, которые уже являются внутренними для внутреннего пространства имен.

Пример 3 — Вложенные пространства имен и спецификаторы доступа

```
NAMESPACE pN1
  NAMESPACE pN11
    FUNCTION pF1 ... END_FUNCTION                  // доступно отовсюду
    FUNCTION INTERNAL iF2 ... END_FUNCTION         // доступно в pN11
    FUNCTION_BLOCK pFB1                            // доступно отовсюду
    VAR PUBLIC pVar1: REAL: ... END_VAR            // доступно отовсюду
      VAR INTERNAL iVar2: REAL ... END_VAR         // доступно в pN11
      ...
    END_FUNCTION_BLOCK
    FUNCTION_BLOCK INTERNAL iFB2          // доступно в pN11
    VAR PUBLIC pVar3: REAL: ... END_VAR   // доступно в pN11
    VAR INTERNAL iVar4: REAL ... END_VAR  // доступно в pN11
      ...
    END_FUNCTION_BLOCK
    CLASS pC1
    VAR PUBLIC pVar5: REAL: ... END_VAR            // доступно отовсюду
    VAR INTERNAL iVar6: REAL ... END_VAR           // доступно в pN11
    METHOD pM1 ... END_METHOD                      // доступно отовсюду
    METHOD INTERNAL iM2 ... END_METHOD             // доступно в pN11
    END_CLASS
    CLASS INTERNAL iC2
    VAR PUBLIC pVar7: REAL: ... END_VAR            // доступно в pN11
    VAR INTERNAL iVar8: REAL ... END_VAR           // доступно в pN11
    ETHOD pM3 ... END_METHOD                       // доступно в pN11
    METHOD INTERNAL iM4 ... END_METHOD             // доступно в pN11
    END_CLASS
END_NAMESPACE
NAMESPACE INTERNAL iN12
    FUNCTION pF1 ... END_FUNCTION                  // доступно в pN11
    FUNCTION INTERNAL iF2 ... END_FUNCTION         // доступно в iN12
    FUNCTION_BLOCK pFB1                            // доступно в pN1
      VAR PUBLIC pVar1: REAL: ... END_VAR          // доступно в pN1
      VAR INTERNAL iVar2: REAL ... END_VAR         // доступно в iN12
      ...
    END_FUNCTION_BLOCK

    FUNCTION_BLOCK INTERNAL iFB2                   // доступно в iN12
      VAR PUBLIC pVar3: REAL: ... END_VAR          // доступно в iN12
      VAR INTERNAL iVar4: REAL ... END_VAR         // доступно в iN12
      ...
    END_FUNCTION_BLOCK
    CLASS pC1
      VAR PUBLIC pVar5: REAL: ... END_VAR          // доступно в pN1
      VAR INTERNAL iVar6: REAL ... END_VAR         // доступно в iN12
      METHOD pM1 ... END_METHOD                    // доступно в pN1
      METHOD INTERNAL iM2 ... END_METHOD           // доступно в iN12
    END_CLASS
    CLASS INTERNAL iC2
```

```
        VAR PUBLIC pVar7: REAL: ... END_VAR          // доступно в in iN12
        VAR INTERNAL iVar8: REAL ... END_VAR         // доступно в in iN12
        METHOD pM3 ... END_METHOD                    // доступно в iN12
        METHOD INTERNAL iM4 ... END_METHOD           // доступно в iN12
    END_CLASS
  END_NAMESPACE
END_NAMESPACE
```

В таблице 64 показаны свойства, определенные для пространства имен.

Таблица 64 — Пространство имен

Номер	Описание	Пример
1a	Общее пространство имен (без спецификатора доступа)	NAMESPACE name declaration(s) declaration(s) END_NAMESPACE Все содержащиеся элементы доступны в соответствии со своими спецификаторами доступа
1b	Внутреннее пространство имен (со спецификатором INTERNAL)	NAMESPACE INTERNAL name declaration(s) declaration(s) END_NAMESPACE Все содержащиеся элементы без какого-либо спецификатора или со спецификатором доступа PUBLIC доступны в пространстве имен на уровень выше
2	Вложенные пространства имен	См. пример 2
3	Спецификатор доступа к переменной INTERNAL	CLASS C1 VAR INTERNAL myInternalVar: INT; END_VAR VAR PUBLIC myPublicVar: INT; END_VAR END_CLASS
4	Спецификатор доступа к методу INTERNAL	CLASS C2 METHOD INTERNAL myInternalMethod: INT; ... END_METHOD METHOD PUBLIC myPublicMethod: INT; ... END_METHOD END_CLASS
5	Элемент языка со спецификатором доступа INTERNAL: Типы данных, определяемые пользователем - с использованием ключевого слова TYPE Функции Типы функциональных блоков Классы Интерфейсы	CLASS INTERNAL METHOD INTERNAL myInternalMethod: INT; ... END_METHOD METHOD PUBLIC myPublicMethod: INT; ... END_METHOD END_CLASS CLASS METHOD INTERNAL myInternalMethod: INT; ... END_METHOD METHOD PUBLIC myPublicMethod: INT; ... END_METHOD END_CLASS

Именем пространства имен может быть простой идентификатор или полностью уточненное имя, состоящее из последовательности идентификаторов пространства имен, разделенных точками («.»). Последняя форма допускает объявление вложенного пространства имен без объявлений лексически

вложенных нескольких пространств имен. Она также поддерживает расширение существующего пространства имен с дополнительными элементами языка за счет дополнительного объявления.

Лексически вложенные пространства имен, описываемые несколькими объявлениями пространств имен с ключевым словом NAMESPACE, текстуально вложены, как показано в первом из трех свойств в таблице 65. Все три свойства вносят элементы языка в одно и то же пространство имен Standard.Timers.HighResolution. Второе свойство показывает расширение того же пространства имен, объявленного полностью уточненным именем. Третье свойство смешивает объявление пространства имен с полностью уточненным именем и лексически вложенными ключевыми словами NAMESPACE для добавления дополнительного программного компонента к пространству имен.

В таблице 65 показаны свойства, определенные для опций объявления вложенного пространства имен.

Таблица 65 — Варианты объявления вложенного пространства имен

Номер	Описание	Пример
1	Объявление лексически вложенного пространства имен Эквивалентно свойству 2 в таблице 64	NAMESPACE Standard NAMESPACE Timers AMESPACE HighResolution FUNCTION PUBLIC TimeTick: DWORD // ...объявление и операции END_FUNCTION END_NAMESPACE (*HighResolution*) END_NAMESPACE (*Timers*) END_NAMESPACE (*Standard*)
2	Объявление пространства имен полностью уточненным именем	NAMESPACE Standard.Timers.HighResolution FUNCTION PUBLIC TimeResolution: DWORD // ...объявление и операции END_FUNCTION END_NAMESPACE (*Standard.Timers.HighResolution*)
3	Смешанные лексически вложенное пространство имен и пространство имен, вложенное использованием полностью уточненного имени	NAMESPACE Standard.Timers NAMESPACE HighResolution FUNCTION PUBLIC TimeLimit: DWORD // ...объявление и операции END_FUNCTION END_NAMESPACE (*HighResolution*) END_NAMESPACE (*Standard.Timers*)

Примечание — Несколько объявлений пространства имен с одним и тем же полностью уточненным именем осуществляет вложение в одно и то же пространство имен. В примерах этой таблицы функции TimeTick, TimeResolution и TimeLimit являются членами одного и того же пространства имен Standard.Timers.HighResolution даже если они определены в отдельных объявлениях пространства имен; например, в различных файлах программы Structured Text.

6.9.3 Использование

Элементы пространства имен могут быть доступны извне относительно пространства имен использованием предшествующего имени пространства имен и последующей точки «.». В этом нет необходимости изнутри пространства имен, но допустимо.

К элементам языка, объявляемым со спецификатором доступа INTERNAL, не может быть доступа извне относительно пространства имен, за исключением собственного пространства имен.

Доступ к элементам во вложенных пространствах имен возможен с использованием наименования всех родительских пространств имен, как показано в примере.

Пример — Использование Timer TON из пространства имен Standard.Timers

```
FUNCTION_BLOCK Uses_Timer
VAR
```

```
    Ton1: Standard.Timers.TON;
    (* запускает таймер передним фронтом, сбрасывает таймер задним фронтом*)
    Ton2: PUBLIC.TON; (* использует стандартный таймер *)
bTest: BOOL;
END_VAR
    Ton1(In:= bTest, PT:= t#5s);
END_FUNCTION_BLOCK
```

6.9.4 Директива USING пространства имен

Директива USING может задаваться вслед за именем пространства имен, программным компонентом, именем и объявлением результата функции или метода. Если директива USING используется внутри функционального блока, класса или структуры, она следует непосредственно за именем типа.

Если директива USING используется внутри функции или метода, она непосредственно следует за объявлением типа результата функции или метода.

Директива USING начинается с ключевого слова USING, за которым следует одно или несколько полностью уточненных имен пространств имен, как показано в таблице 64, свойство 2. Это разрешает использование элементов языка, содержащихся в заданных пространствах имен, непосредственно в окружающем пространстве имен соответствующего программного компонента. Окружающее пространство имен также может являться глобальным пространством имен.

В пределах объявления членов в пространстве имен, которое содержит директиву пространства имен USING, на типы, содержащиеся в заданном пространстве имен, можно ссылаться прямо. В приведенном ниже примере в пределах объявления членов пространства имен Infeed, члены типа Standard. Timers прямо доступны, и, поэтому, функциональный блок Uses_Timer может объявлять переменную экземпляра функционального блока TON без квалификации.

В примерах 1 и 2 ниже показано использование директивы пространства имен USING.

Пример 1 — Директива пространства имен USING

```
NAMESPACE Counters
    FUNCTION_BLOCK CUP
    // ... объявление и операции
    END_FUNCTION_BLOCK
END_NAMESPACE (*Standard.Counters*)

NAMESPACE Standard.Timers
    FUNCTION_BLOCK TON
    // ... объявление и операции
    END_FUNCTION_BLOCK
END_NAMESPACE (*Standard.Timers*)

NAMESPACE Infeed
FUNCTION_BLOCK Uses_Std
    USING Standard.Timers;

VAR
    Ton1: TON;
    (* запускает таймер с передним фронтом, сбрасывает таймер с задним фронтом*)
    Cnt1: Counters.CUP;
    bTest: BOOL;
END_VAR
    Ton1(In:= bTest, PT:= t#5s);
END_FUNCTION_BLOCK
END_NAMESPACE
```

Директива пространства имен USING делает доступными типы, содержащиеся в заданном пространстве имен, но специально не делает доступными типы, содержащиеся во вложенных пространствах имен. Директива пространства имен USING делает доступными типы, содержащиеся в Standard, но не типы пространств имен, вложенные в Standard. Таким образом, ссылка на Timers.TON в объявлении Uses_Timer приводит к ошибке компиляции, поскольку в области видимости отсутствуют члены с именем Standard.

Пример 2 — Недопустимый импорт вложенных пространств имен

```
NAMESPACE Standard.Timers
    FUNCTION_BLOCK TON
    // ... объявление и операции
    END_FUNCTION_BLOCK
END_NAMESPACE (*Standard.Timers*)
NAMESPACE Infeed
    USING Standard;
    USING Standard.Counters;
    FUNCTION_BLOCK Uses_Timer
    VAR
    ~~Ton1: Timers.TON; // ERROR.~~ Вложенные пространства имен не импортируются.
    (* запускает таймер с передним фронтом, сбрасывает таймер с задним фронтом*)
    bTest: BOOL;
    END_VAR
    Ton1(In:= bTest, PT:= t#5s);
    END_FUNCTION_BLOCK
END_NAMESPACE (*Standard.Timers.HighResolution*)
```

Для доступа к элементам языка пространства имен в глобальном пространстве имен должны использоваться ключевое слово USING и идентификаторы пространства имен.

В таблице 66 показаны свойства, определяемые для директивы пространства имен USING.

Таблица 66 — Директива пространства имен USING

Номер	Описание	Пример
1	USING в глобальном пространстве имен	USING Standard.Timers; FUNCTION PUBLIC TimeTick: DWORD VAR Ton1: TON; END_VAR // ...объявление и операции END_FUNCTION
2	USING в другом пространстве имен	NAMESPACE Standard.Timers.HighResolution USING Counters; FUNCTION PUBLIC TimeResolution: DWORD // ...объявление и операции END_FUNCTION END_NAMESPACE (*Standard.Timers.HighResolution*)

Окончание таблицы 66

Номер	Описание	Пример
3	USING в программных компонентах: - функции - типы функционального блока - классы - методы - интерфейсы	FUNCTION_BLOCK Uses_Std USING Standard.Timers, Counters; VAR Ton1: TON; (* запускает таймер с передним фронтом, сбрасывает таймер с задним фронтом*) Cnt1: CUP; bTest: BOOL; END_VAR Ton1(In:= bTest, PT:= t#5s); END_FUNCTION_BLOCK FUNCTION myFun: INT USING Lib1, Lib2; USING Lib3; VAR .... END_FUNCTION

7 Текстовые языки

7.1 Общие элементы

Текстовые языки, определяемые в настоящем стандарте, это IL (перечень инструкций) и ST (структурированный текст). Последовательная функциональная схема (SFC) может использоваться совместно с любым из этих языков.

В подразделе 7.2 определяется семантика языка IL, синтаксис которого приведен в приложении А. В подразделе 7.3 определяется семантика языка ST, синтаксис которого задан.

Текстовые элементы, указанные в разделе 6 являются общими для текстовых языков (IL и ST), определяемых в разделе 7. В частности, приведенные ниже элементы структурирования программы на рисунке 30 являются общими для текстовых языков:

```
TYPE              ...END_TYPE
VAR               ...END_VAR
VAR_INPUT         ...END_VAR
VAR_OUTPUT        ...END_VAR
VAR_IN_OUT        ...END_VAR
VAR_EXTERNAL      ...END_VAR
VAR_TEMP          ...END_VAR
VAR_ACCESS        ...END_VAR
VAR_GLOBAL        ...END_VAR
VAR_CONFIG        ...END_VAR
FUNCTION          ...END_FUNCTION
FUNCTION_BLOCK...END_FUNCTION BLOCK
PROGRAM           ...END_PROGRAM
METHOD            ...END_METHOD
STEP              ...END_STEP
TRANSITION        ...END_TRANSITION
ACTION            ...END_ACTION
NAMESPACE         ...END_NAMESPACE
```

Рисунок 30 — Общие текстовые элементы (обзор)

7.2 Перечень инструкций (IL)

7.2.1 Общие положения

Этот язык устарел как язык типа ассемблера. Поэтому он не рекомендуется и не будет использоваться в следующей редакции настоящего стандарта.

7.2.2 Инструкции

Перечень инструкций состоит из последовательности инструкций. Каждая инструкция начинается на новой строке и содержит оператор с необязательными модификаторами, и, если необходимо для конкретной операции, один или большее число операндов, разделенных запятыми. Операнды могут быть любыми представлениями данных литералов, перечислимыми значениями и переменными.

Инструкции может предшествовать метка идентификации, за которой следует двоеточие «:». Между командами могут вставляться пустые строки.

Пример — Поля списка инструкций

МЕТКА	*ОПЕРАТОР*	*ОПЕРАНД*	*КОММЕНТАРИЙ*
START:	*LD*	*%IX1*	*(* Кнопка *)*
	ANDN	*%MX5*	*(* Не запрещено*)*
	ST	*%QX2*	*(* Вентилятор включен*)*

7.2.3 Операторы, модификаторы и операнды

7.2.3.1 Общие положения

Стандартные операторы с их разрешенными модификаторами и операндами должны быть такими, как показано в таблице 68.

7.2.3.2 «Текущий результат»

Если иное не оговорено в таблице 68, семантика операторов должна быть следующей:

result:= result OP operand.

То есть значение вычисляемого выражения заменяется его текущим значением, действующим на оператор в соответствии с операндом.

Пример 1 — Инструкция AND %IX1 интерпретируется как result:= result AND %IX1.

Операторы сравнения интерпретируются с текущим результатом слева от сравнения и операндом справа с логическим результатом.

Пример 2 — Инструкция GT %IW10 имеет логический результат 1, если текущее значение больше, чем значение Input Word 10, и логический результат в противном случае.

7.2.3.3 Модификатор

Модификатор «N» указывает поразрядное логическое отрицание (дополнение до единицы) операнда.

Пример 1 — Инструкция ANDN %IX2 интерпретируется как result:= result AND NOT %IX2.

Ошибка возникает, если текущий результат и операнд имеют разный тип данных или если результат числовой операции превышает область значений для его типа данных.

Левый скобочный модификатор «(» указывает, что вычисление оператора должно быть отложено до появления правого скобочного оператора «)». В таблице 67 показаны две эквивалентные формы последовательности инструкций со скобками. Оба свойства в таблице 67 интерпретируются как

result:= result AND (%IX1 OR %IX2)

Операнд должен быть литералом, как определено в 6.3, перечислимым значением или переменной.

Функция REF() и оператор разыменования «^» используются в определении операндов, а в таблице 67 показано выражение в скобках.

Таблица 67 — Выражение в скобках для языка IL

Номер	Описание	Пример
1	Выражение в скобках начинается с оператора в явном виде:	AND(LD %IX1 (NOTE) OR %IX2)
2	Выражение в скобках (краткая форма)	AND(%IX1 OR% IX2)
Примечание — В свойстве 1 оператор LD может быть изменен или операция LD может быть заменена вызовом другой операции или функции, соответственно.		

Модификатор «С» указывает, что связанная инструкция выполняется, только если значение текущего вычисленного результата равно логической 1 (или логическому 0, если оператор объединен с модификатором «N»). В таблице 68 показаны операторы списка инструкций.

Таблица 68 — Операторы языка IL

Номер	Описание оператора[a)]	Модификатор (см. примечание)	Объяснение
1	LD	N	Установить текущий результат, равный операнду
2	ST	N	Сохранить текущий результат по адресу операнда
3	S[e)], R[e)]		Установить операнд в 1, если текущий результат равен логической 1 Сбросить операнд в 0, если текущий результат равен логической 1
4	AND	N, (	Логическое И
5	&	N, (	Логическое И
6	OR	N, (	Логическое ИЛИ
7	XOR	N, (	Логическое исключающее ИЛИ
8	NOT[d)]		Логическое отрицание (дополнение до единицы)
9	ADD	(	Сложение
10	SUB	(	Вычитание
11	MUL	(	Умножение
12	DIV	(	Деление
13	MOD	(	Деление по модулю
14	GT	(	Сравнение: >
15	GE	(	Сравнение: >=
16	EQ	(	Сравнение: =
17	NE	(	Сравнение: <>
18	LE	(	Сравнение: <=
19	LT	(	Сравнение: <
20	JMP[b)]	C, N	Переход к метке
21	CAL[c)]	C, N	Вызов функционального блока (см. таблицу 69)

Окончание таблицы 68

Номер	Описание оператора[a)]	Модификатор (см. примечание)	Объяснение
22	RET[f)]	C, N	Возврат из вызванной функции, функционального блока или программы
23	)		Вычислить отложенную операцию
24	ST?		Попытка присваивания. Сохранить с проверкой

Для объяснения модификаторов и оценки выражений см. предшествующий текст.

[a)] Если иное не указано, эти операторы должны быть перегружены или типизированы.

[b)] Операнд инструкции JMP должен быть меткой инструкции, к которой должно переходить выполнение. Когда инструкция JMP содержится в конструкции ACTION... END_ACTION, операнд должен быть меткой внутри той же самой конструкции.

[c)] Операнд этой инструкции должен быть именем экземпляра вызываемого функционального блока.

[d)] Результатом этой операции должно быть побитовое логическое отрицание (дополнение до единицы) текущего результата.

[e)] Типом операнда этой инструкции должен быть BOOL.

[f)] Эта инструкция не имеет операнда.

7.2.4 Функции и функциональные блоки

7.2.4.1 Общие положения

Общие правила и свойства вызова функции и вызова функционального блока также применимы и в IL. Свойства для вызова функциональных блоков и функций определены в таблице 69.

7.2.4.2 Функция

Функции вызываются путем помещения имени функции в поле оператора. Параметры задаются вместе в одном поле операнда или же каждый параметр — в поле операнда строка за строкой.

В случае неформального вызова первый параметр функции не обязательно должен содержаться в параметре, однако текущий результат используется как первый параметр функции. Дополнительные параметры (начиная со второго), при необходимости, задаются в поле операнда, разделенные запятыми, в порядке их объявления.

Функции могут иметь результат. Как показано в свойствах 3 таблицы 69, при успешном выполнении инструкции RET или после достижения конца программного компонента, программный компонент предоставляет результат как «текущий результат».

Если вызвана функция, которая не имеет результата, то «текущий результат» является неопределенным.

7.2.4.3 Функциональный блок

Функциональный блок вызывается размещением ключевого слова CAL в поле оператора, а имени экземпляра функционального блока — в поле операнда. Параметры задаются вместе или же каждый параметр помещается в поле операнда.

Функциональные блоки вызываются при определенных условиях или безусловно оператором EN.

Все назначения параметров, определяемые в перечне параметров вызова условного функционального блока, выполняются только вместе с вызовом, если условие является истинным.

Если вызван экземпляр функционального блока, то «текущий результат» является неопределенным.

7.2.4.4 Методы

Методы вызываются помещением имени экземпляра функционального блока, за которым следует одиночный период «.» и имя метода, в поле оператора. Параметры задаются вместе в одном поле операнда или же каждый параметр — в поле операнда строка за строкой.

В случае неформального вызова первый параметр метода не обязательно должен содержаться в параметре, однако текущий результат используется как первый параметр функции.

Дополнительные параметры (начиная со второго), при необходимости, задаются в поле операнда, разделенные запятыми, в порядке их объявления.

Методы могут иметь результат. Как показано в свойствах 4 таблицы 69, при успешном исполнении инструкции RET или при достижении конца программного компонента, программный компонент предоставляет результат как «текущий результат».

Если вызван метод, который не имеет результата, то «текущий результат» является неопределенным. В таблице 69 приведены альтернативные вызовы языка IL.

Таблица 69 — Вызовы для языка IL

Номер	Описание	Пример (см. примечание)
1a	Вызов функционального блока с перечнем неформальных параметров	CAL C10(%IX10, FALSE, A, OUT, B) CAL CMD_TMR(%IX5, T#300ms, OUT, ELAPSED)
1b	Вызов функционального блока с перечнем формальных параметров	CAL C10(// FB имя экземпляра CU := %IX10, R := FALSE, PV := A, Q => OUT, CV => B) CAL CMD_TMR(IN := %IX5, PT := T#300ms, Q => OUT, ET => ELAPSED, ENO => ERR)
2	Вызов функционального блока с загрузкой/сохранением стандартных входных параметров	LD A ADD D5 ST C10.PV LD %IX10 ST C10.CU CAL C10 // FB имя экземпляра LD C10.CV // текущий результат

3a	Вызов функции с перечнем формальных параметров	LIMIT(// Имя функции
		EN := COND,
		IN := B,
		MN := 1,
		MX := 5,
		ENO => TEMPL
		)
		ST A // Новый текущий результат

3b	Вызов функции с перечнем неформальных параметров	LD 1 //установить текущий результат LIMIT B, 5 // и использовать его как IN ST A // Новый текущий результат

4a	Вызов метода с перечнем формальных параметров	FB_INST.M1(// Имя метода
		EN := COND,
		IN := B,
		MN := 1,
		MX := 5,
		ENO => TEMPL
		)
		ST A // Новый текущий результат

Окончание таблицы 69

Номер	Описание	Пример (см. примечание)
4b	Вызов метода с перечнем неформальных параметров	LD 1 //установить текущий результат FB_INST.M1 B, 5 // и использовать его как IN ST A // новый текущий результат

Примечание — В приведенных выше примерах предполагается объявление

```
VAR
    C10       : CTU;
    CMD_TMR: TON;
    A, B      : INT;
    ELAPSED: TIME;
    OUT, ERR, TEMPL, COND: BOOL;
END_VAR
```

Стандартные входные операторы стандартных функциональных блоков, определенные в таблице 70, могут использоваться в сочетании со свойством 2 (загрузка/сохранение) в таблице 69. Данный вызов эквивалентен CAL с перечнем параметров, который содержит только одну переменную с именем входного оператора.

Параметры, которые не предоставляются, берутся из последнего присваивания или, если они не предоставлены, из инициализации. Это свойство поддерживает проблемную ситуацию, где события являются предсказуемыми, и поэтому только одна переменная может быть изменена от одного вызова к следующему.

Пример 1

Вместе с объявлением

VAR C10: CTU; END_VAR

в последовательности команд

```
LD      15
PV      C10
```

дает такой же результат, что и

```
CAL     C10(PV:=15)
```

Пропущенные входы R и CU имеют значения, присвоенные им раньше. Поскольку вход CU детектирует передний фронт, данным вызовом будет установлено только значение входа PV; отсчет не может произойти, поскольку непредоставленный параметр не может измениться. В отличие от этого результаты последовательности

```
LD      %IX10
CU      C10
```

в отсчете при максимуме в каждом втором вызове зависят от скорости изменения входа %IX10. Каждый вызов использует ранее установленные значения для PV и R.

Пример 2

С бистабильными функциональными блоками, с получением декларации

VAR FORWARD: SR; END_VAR

это приводит к неявно условному поведению. Последовательность

```
LD      FALSE
S1      FORWARD
```

не изменяет состояние бистабильного FORWARD. Следующая последовательность

```
LD      TRUE
R       FORWARD
```

сбрасывает бистабильное состояние.

Таблица 70 — Стандартные операторы функционального блока для языка IL

Номер	Функциональный блок	Входной оператор	Выходной оператор
1	SR	S1, R	Q
2	RS	S, R1	Q
3	F/R_TRIG	CLK	Q
4	CTU	CU, R, PV	CV, Q, также RESET
5	CTD	CD, PV	CV, Q
6	CTUD	CU, CD, R, PV	CV, QU, QD, также RESET
7	TP	IN, PT	CV, Q
8	TON	IN, PT	CV, Q
9	TOF	IN, PT	CV, Q
Примечание — LD (загрузка) не является необходимой как входной оператор стандартного функционального блока, поскольку функциональные возможности LD включены в PV.			

Параметры, которые не предоставляются, берутся из последнего присваивания или, если они не предоставлены, то из инициализации. Данное свойство поддерживает проблемную ситуацию, где события являются предсказуемыми, и поэтому только одна переменная может изменяться от одного вызова к следующему.

7.3 Структурированный текст (ST)

7.3.1 Общие положения

Текстовый язык программирования "Structured Text, ST" предоставляется из языка программирования Паскаль для использования в настоящем стандарте.

7.3.2 Выражения

В языке ST конец текстовой строки должен интерпретироваться так же, как символ пробела (SP).

Выражение — это конструкция, которая при вычислении дает значение, соответствующее одному из типов данных. Максимально допустимая длина выражений определяется разработчиком.

Выражения состоят из операторов и операндов. Операнд должен быть литералом, перечислимым значением, переменной, вызовом функции с результатом, вызовом метода с результатом, вызовом экземпляра функционального блока с результатом или другим выражением.

Операторы языка ST обобщены в таблице 71.

Разработчик определяет явные и неявные преобразования типа.

При вычислении выражения применяются следующие правила:

1 Операторы применяют операнды в последовательности, определяемой приоритетом операторов, приведенным в таблице 71. Оператор с наивысшим приоритетом в выражении применяется первым, за ним следует оператор со следующим более низким приоритетом и т. д. до завершения вычисления.

Пример 1

Если A, B, C и D типа INT со значениями 1, 2, 3 и 4, соответственно, тогда

A+B-C*ABS(D)

вычисляется до -9, а

(A+B-C)*ABS(D)

вычисляется до 0.

2 Операторы равного приоритета должны применяться как записанные в выражении слева направо.

Пример 2

A+B+C вычисляется как (A+B)+C.

3 Когда оператор имеет два операнда, первым вычисляется крайний слева операнд.

Пример 3

В выражении

SIN(A)*COS(B) выражение SIN(A) вычисляется вначале, за ним следует COS(B), затем следует вычисление произведения.

4 Логические выражения вычисляются только до степени, необходимой для определения результирующего значения, включая возможные побочные эффекты. Степень, до которой оценивается логическое выражение определяется разработчиком.

Пример 4

Для выражения(A>B)&(C<D) достаточно, если

A<=B, чтобы оценить только (A>B), чтобы решить, что значение выражения равно FALSE.

5 Функции и методы вызываются как элементы выражения, включающие имя функции или метода, за которыми следует перечень параметров в скобках.

6 Когда оператор в выражении представлен как одна из перегруженных функций, преобразование операндов и результаты следуют правилу и приведенным ниже примерам.

Приведенные ниже условия при выполнении операторов рассматриваются как ошибки:

a) сделана попытка деления на нуль;

b) операнды не относятся к корректному типу данных для операции;

c) результат числовой операции превышает диапазон значений для ее типа данных.

Таблица 71 — Операторы языка ST

Номер	Описание Операция[a)]	Символ	Пример	Приоритет
1	Скобки	(выражение)	(A+B/C), (A+B)/C, A/(B+C)	11 (Самый высокий)
2	Вычисление результата функции и метода - если результат объявлен	Идентификатор (перечень параметров)	LN(A), MAX(X,Y), myclass.my_method(x)	10
3	Разыменование	^	R^	9
4	Отрицание	–	-A, - A	8
5	Унарный плюс	+	+B, + B	8
5	Дополнение	NOT	NOT C	8
7	Возведение в степень[b)]	**	A**B, B ** B	7
8	Умножить	*	A*B, A * B	6
9	Разделить	/	A/B, A / B / D	6
10	Модуль	MOD	A MOD B	6
11	Добавить	+	A+B, A + B + C	5
12	Вычесть	–	A-B, A – B – C	5
13	Сравнение	< , > , <= , >=	A<B A < B < C	4
14	Равенство	=	A=B, A=B & B=C	4
15	Неравенство	<>	A<>B, A <> B	4
16a	Логическое И	&	A&B, A & B, A & B & C	3
16b	Логическое И	AND	A AND B	3
17	Логическое исключающее ИЛИ	XOR	A XOR B	2
18	Логическое ИЛИ	OR	A OR B	1 (Низший)

Окончание таблицы 71

[a] Те же правила применимы к операндам этих операторов как к входам соответствующих стандартных функций. [b] Результат вычисления выражения A**B должен быть таким же, как результат вычисления функции EXPT(A, B).

7.3.3 Операторы

7.3.3.1 Общие положения

Операторы языка ST обобщены в таблице 72. Максимально допустимая длина операторов устанавливается разработчиком.

Таблица 72 — Операторы языка ST

Номер	Описание	Примеры
1	**Присваивание** переменная := выражение;	
1a	Переменная и выражение простого типа данных	A:= B; CV:= CV+1; C:= SIN(X);
1b	Переменные и выражение простого типа данных с неявным преобразованием типа в соответствии с рисунком 11	A_Real:= B_Int;
1c	Переменная и выражение типа данных, определяемого пользователем	A_Struct1:= B_Struct1; C_Array1 := D_Array1;
1d	Экземпляры типа функционального блока	A_Instance1:= B_Instance1;
	Вызов функции	
2a[b]		FCT(17);
2b[b] 2c[b]	Вызов функционального блока и использование выходной переменной функционального блока	CMD_TMR(IN:= bIn1, PT:= T#300ms); A:= CMD_TMR.Q; FB_INST.M1(17);
3	ВОЗВРАТ	RETURN;
	Выбор	
4	IF ... THEN ... ELSIF ... THEN ... ELSE ...END_IF	D:= B*B — 4.0*A*C; IF D < 0.0 THEN NROOTS:= 0; ELSIF D = 0.0 THEN NROOTS:= 1; X1:= - B/(2.0*A); ELSE NROOTS:= 2; X1:= (- B + SQRT(D))/(2.0*A); X2:= (- B - SQRT(D))/(2.0*A); END_IF;

Окончание таблицы 72

Номер	Описание	Примеры
5	CASE ... OF ... ELSE ... END_CASE	TW:= WORD_BCD_TO_INT(THUMBWHEEL); TW_ERROR:= 0; CASE TW OF 1,5: DISPLAY:= OVEN_TEMP; 2: DISPLAY:= MOTOR_SPEED; 3: DISPLAY:= GROSS - TARE; 4,6..10: DISPLAY:= STATUS(TW - 4); ELSE DISPLAY := 0; TW_ERROR:= 1; END_CASE; QW100:= INT_TO_BCD(DISPLAY);
	Итерация	

6	FOR ... TO ... BY ... DO
	...
	END_FOR
		J:= 101;
		FOR I:= 1 TO 100 BY 2 DO
		IF WORDS[I] = 'KEY' THEN
		J:= I;
		EXIT;
		END_IF;
		END_FOR;

7	WHILE ... DO ... END_WHILE	J:= 1; WHILE J <= 100 & WORDS[J] <> 'KEY' DO J:= J+2; END_WHILE;
8	REPEAT ... UNTIL ... END_REPEAT	J:= -1; REPEAT J:= J+2; UNTIL J = 101 OR WORDS[J] = 'KEY' END_REPEAT;
9[a)]		J:= 1; WHILE (J <= 100 AND WORDS[J] <> 'KEY') DO ..IF (J MOD 3 = 0) THEN CONTINUE; END_IF; (* если j =1,2,4,5,7,8, ... тогда этот оператор*); ... END_WHILE;
10[a)]	Выход из итерации	EXIT; (см. также в свойстве 6)
11	Пустой оператор	;

[a)] Если оператор EXIT или CONTINUE (свойство 9 или 11) поддерживается, то он должен поддерживаться для всех операторов итерации (FOR, WHILE, REPEAT), которые поддерживаются в реализации.

[b)] Если функция, тип функционального блока или метод дает результат, а вызов отсутствует в выражении присваивания, то результат отменяется.

7.3.3.2 Присваивание (Сравнение, результат, вызов)

7.3.3.2.1 Общие положения

Оператор присваивания заменяет текущее значение единственной или многоэлементной переменной на результат оценки выражения. Оператор присваивания включает ссылку на переменную на левой стороне, за которой следует оператор присваивания «:=», за которым следует выражение, которое должно быть вычислено.

Например, оператор

A := B;

используется для замены одиночного значения данных переменной A на текущее значение переменной B, если оба типа INT или переменная B может быть неявно преобразована в тип INT.

Если A и B — многоэлементные переменные, типы данных A и B должны быть одинаковыми. В этом случае элементы переменной A получают значения элементов переменной B.

Например, если A и B имеют типы ANALOG_CHANNEL_CONFIGURATION, то значения всех элементов структурированной переменной A должны быть заменены текущими значениями соответствующих элементов переменной B.

7.3.3.2.2 Сравнение

Сравнение возвращает его результат как логическое значение. Сравнение должно включать ссылку на переменную на левой стороне, за которой следует оператор присваивания, за которым следует ссылка на переменную на правой стороне. Переменные могут быть одноэлементными или многоэлементными переменными.

Сравнение

A = B

должно использоваться для сравнения значения данных переменной A с значением переменной B, если обе относятся к одному типу данных или одна из переменных может быть неявно преобразована в тип данных другой переменной.

Если A и B — многоэлементные переменные, то типы данных A и B должны быть одинаковыми. В этом случае элементы переменной A сравниваются со значениями элементов переменной B.

7.3.3.2.3 Результат

Присваивание также используется, чтобы присвоить результат функции типу функционального блока или методу. Если результат определен для этого программного компонента, выполняется, по крайней мере, одно присваивание к имени этого программного компонента. Возвращаемое значение должно быть результатом самого последнего выполнения такого присваивания. Ошибкой является возврат вычисления с переменной ENO значения TRUE, если только не было выполнено, по крайней мере, одно такое присваивание.

7.3.3.2.4

Функция, метод и операторы управления функционального блока включают механизмы для вызова этого программного компонента и для возврата управления вызывающему объекту до физического завершения программного компонента.

- FUNCTION

Функция вызывается оператором, включающим имя функции, за которым следует перечень параметров в скобках, как показано в таблице 72.

Для вызовов функции применяются правила и свойства, определенные в 6.6.1.7 FUNCTION_BLOCK.

Функциональные блоки вызываются оператором, включающим имя экземпляра функционального блока, которым следует перечень параметров в скобках, как показано в таблице 72.

- METHOD

Методы вызываются оператором, включающим имя экземпляра, за которым следует «.» и имя метода, и перечень параметров в скобках.

- RETURN

Оператор RETURN обеспечивает ранний выход из функции, функционального блока или программы (например, как результат оценки оператора IF).

7.3.3.3 Операторы выбора (IF, CASE)

7.3.3.3.1 Общие положения

Операторы выбора включают операторы IF и CASE. Оператор выбора выбирает один (или группу) составляющих его операторов для выполнения на основе указанного условия. Примеры операторов выбора приведены в таблице 72.

7.3.3.3.2 IF

Оператор IF указывает, что группа операторов выполняется, только если связанное логическое выражение при вычислении принимает значение 1 (TRUE). Если условие является ложным, то или оператор не выполняется, или выполняется группа операторов следующая за ключевым словом ELSE (или ключевым словом ELSIF, если связанное логическое условие является истинным).

7.3.3.3.3 CASE

Оператор CASE включает выражение, которое вычисляет переменную простого типа данных («селектор»), и перечень групп операторов, причем каждая группа маркируется одним или большим числом литералов, перечислимых значений или поддиапазонов, в зависимости от того, что применимо. Типы данных таких маркеров должны соответствовать типу данных переменной селектора, т.е. переменная селектора должна быть сравнимой с маркерами.

Это указывает на то, что выполняется первая группа операторов, один из диапазонов которых содержит вычисленное значение селектора. Если значение селектора находится вне диапазона для любого из случаев, то выполняется последовательность операторов, следующая за ключевым словом ELSE (если она имеется в операторе CASE). В противном случае ни одна из последовательностей операторов не выполняется.

Максимально допустимое число выборов в операторах CASE определяется разработчиком.

7.3.3.4 Операторы итерации (WHILE, REPEAT, EXIT, CONTINUE, FOR)

7.3.3.4.1 Общие положения

Операторы итерации указывают, что группа связанных операторов должна выполняться повторно.

Операторы WHILE и REPEAT не должны использоваться для достижения межпроцессной синхронизации, например, как «цикл ожидания» с внешне определяемым условием завершения. Для этой цели должны использоваться элементы SFC.

Ошибка возникает, если оператор WHILE или REPEAT используется в алгоритме, для которого удовлетворение условия завершения цикла или выполнение оператора EXIT не может быть гарантировано.

Оператор FOR используется, когда число итераций может быть определено заранее; в противном случае используются конструкции WHILE или REPEAT.

7.3.3.4.2 FOR

Оператор FOR указывает, что последовательность операторов выполняется повторно, до ключевого слова END_FOR, в то время как последовательность значений присваивается переменной управления циклом FOR. Переменная управления, начальное значение и конечное значение должны быть выражениями одного и того же целого типа (например, SINT, INT или DINT) и не должны изменяться ни в каком из повторяющихся операторов.

Оператор FOR приращивает переменную управления вверх или вниз от начального до конечного значения в приращениях, определяемых значением выражения. Если конструкция BY пропускается, то значение приращения по умолчанию приравнивается к 1.

Пример —

Цикл FOR, задаваемый выражением

FOR I:= 3 TO 1 STEP -1 DO ...;

завершается, когда значение переменной достигает 0.

Тест на условие завершения выполняется в начале каждой итерации, так что последовательность операторов не выполняется, если значение переменной управления превышает конечное значение, то есть значение переменной управления больше, или, соответственно, меньше конечного значения, если значение инкремента положительное, или, соответственно, отрицательное. Значение переменной управления после завершения цикла FOR определяется разработчиком.

Итерация завершается, когда значение переменной управления находится вне диапазона, заданного конструкцией TO.

Следующий пример использования оператора FOR приведен в свойстве 6 таблицы 72. В этом примере цикл FOR используется, чтобы определить индекс J первого появления (если это имеет место) строки «KEY» в нечетных элементах массива строк WORDS с диапазоном значений индексов (1..100). Если появление не обнаружено, J будет иметь значение 101.

7.3.3.4.3 WHILE

Оператор WHILE вызывает выполнение последовательности операторов до ключевого слова END_WHILE. Операторы выполняются повторно до тех пор, пока связанное логическое выражение

станет ложным. Если выражение изначально ложное, то группа операторов вообще не выполняется.

Например, пример FOR...END_FOR может быть переписан с использованием конструкции WHILE...END_WHILE, приведенной в таблице 72.

7.3.3.4.4 REPEAT

Оператор REPEAT вызывает последовательность операторов до ключевого слова UNTIL для выполнения повторно (и, по крайней мере, однократно), пока связанное логическое условие является истинным.

Например, пример WHILE...END_WHILE может быть переписан с использованием конструкции WHILE...END_WHILE, приведенной в таблице 72.

7.3.3.4.5 CONTINUE

Оператор CONTINUE используется для перехода через оставшиеся операторы цикла итерации, в котором CONTINUE размещается после последнего оператора цикла непосредственно перед терминатором цикла (END_FOR, END_WHILE или END_REPEAT).

Пример —

После выполнения операторов, значение переменной, если значение логической переменной FLAG=0 и SUM=9, если FLAG=1.

```
SUM:= 0;
FOR I:= 1 TO 3 DO
  FOR J:= 1 TO 2 DO
    SUM:= SUM + 1;
    IF FLAG THEN
      CONTINUE;
    END_IF;
    SUM:= SUM + 1;
    END_FOR;
  SUM:= SUM + 1;
END_FOR;
```

7.3.3.4.6 EXIT

Оператор EXIT используется для завершения итераций до удовлетворения условия завершения.

Когда оператор EXIT размещается внутри вложенных итеративных конструкций, выход происходит от внутреннего цикла, в котором размещен EXIT, то есть управление переходит к следующему оператору после терминатора первого цикла (END_FOR, END_WHILE или END_REPEAT), за которым следует оператор EXIT.

Пример —

После выполнения операторов, значение переменной SUM=15, если значение логической переменной FLAG= 0 и SUM=6, если FLAG=1.

```
SUM:= 0;
FOR I:= 1 TO 3 DO
  FOR J:= 1 TO 2 DO
    SUM:= SUM + 1;
    IF FLAG THEN
      EXIT;
    END_IF;
    SUM:= SUM + 1;
  END_FOR;
  SUM:= SUM + 1;
END_FOR
```

8 Графические языки

8.1 Общие элементы

8.1.1 Общие положения

Графические языки, определенные в настоящем стандарте, это LD (релейно-контактные схемы) и FBD (функциональные блоковые диаграммы). Элементы последовательной функциональной схемы (SFC) могут использоваться совместно с любым из этих языков.

Элементы применяются к обоим графическим языкам настоящего стандарта, то есть к LD и FB и к графическому представлению элементов последовательной функциональной схемы (SFC).

8.1.2 Представление переменных и экземпляров

В графических языках все поддерживаемые типы данных должны быть доступны как в операндах или так и в параметрах.

В графических языках должны поддерживаться все объявления экземпляров.

Использование выражения как параметров или как индекса массива не входит в задачу настоящего стандарта.

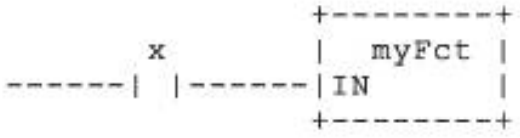
Пример —

```
TYPE
    SType: STRUCT                          Объявления типа
        x: BOOL;
        a: INT;
        t: TON; END_STRUCT;
END_TYPE;

VAR                                        Объявления переменной
        x: BOOL;
        i: INT;
        Xs: ARRAY[1..10] OF BOOL;
        S:  SType;
        Ss: ARRAY [0..3]   OF SType;
        t: TON;
        Ts: ARRAY [0..20] OF TON;
END_VAR
```

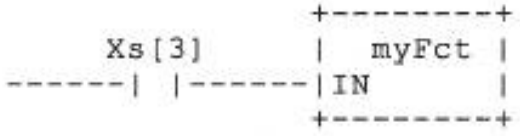
а) Объявления переменной и типа

Использует операнд:

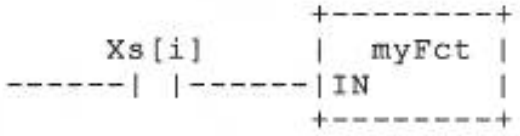
как элементарную переменную



как элемент массива с постоянным индексом

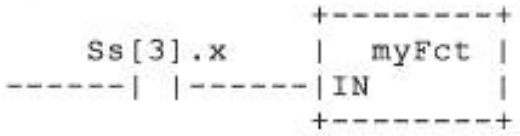


как элемент массива с переменным индексом



как элемент структуры

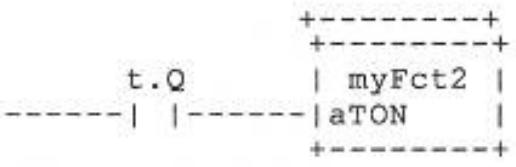
как элемент структурированного массива



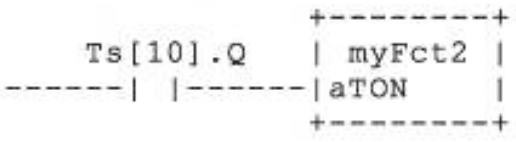
b) Представление операндов

Экземпляр, используемый как параметр:

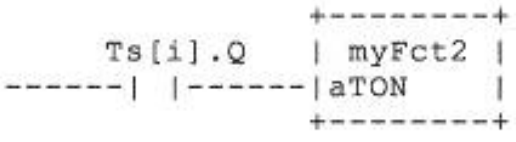
как нормальный экземпляр



как элемент массива с постоянным индексом

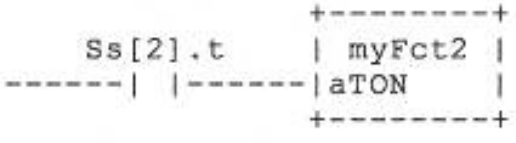


как элемент массива с переменным индексом

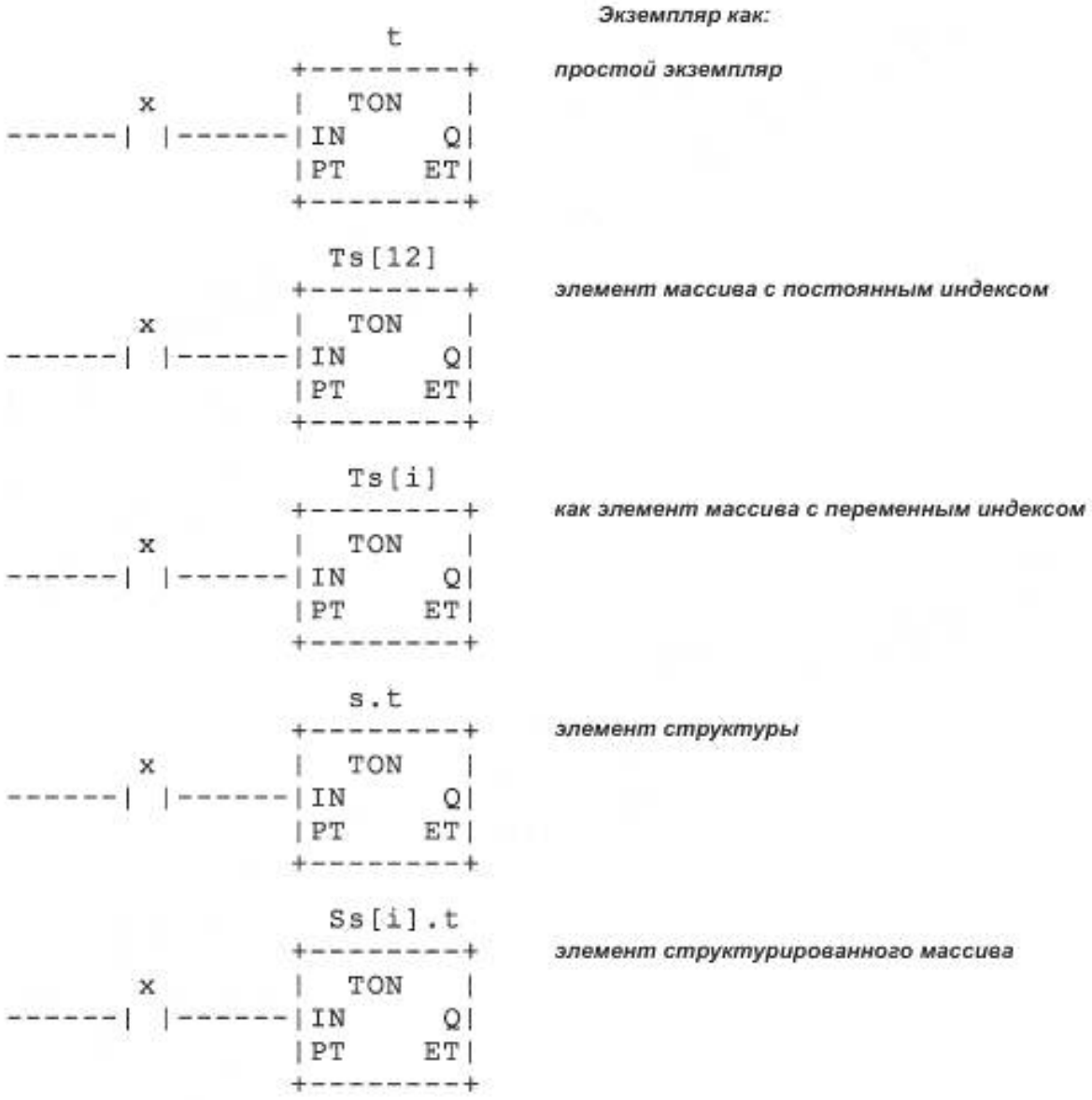


как элемент структуры

как элемент структурированного массива



c) Представление экземпляра как параметра



d) Представление вызова экземпляра

8.1.3 Представление линий и блоков

Использование букв, полуграфических и графических, для представления графических элементов определяется разработчиком и не является нормативным требованием.

Графические элементы языка, определенные в настоящем разделе 8, изображаются с элементами строки с использованием символов из набора символов. Примеры приведены ниже.

Линии можно расширить за счет использования соединителя. Сохранение данных или связь с элементами данных не должны быть связаны с использованием соединителей; однако, чтобы избежать неоднозначности, ошибка возникает, если идентификатор, используемый как метка соединителя, совпадает с именем другого именованного элемента внутри одного и того же программного компонента.

Любые ограничения на топологию сети в конкретной реализации должны быть выражены как определяемые разработчиком.

Пример — Графические элементы

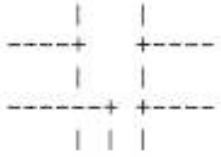
Горизонтальные линии

Вертикальные линии

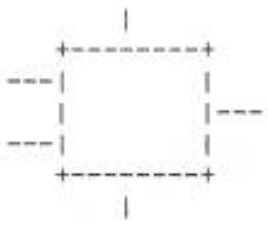
Горизонтальное/вертикальное соединение (узел)

Пересечение линий без соединения (узла нет)

Соединенные и несоединенные углы (узлы)



Блоки с соединительными линиями



Соединители и продолжение

8.1.4 Направление потока в сетях

Сеть определяется как максимальный набор взаимосвязанных графических элементов, исключая левые и правые шины в случае сетей в языке LD. Должны быть приняты меры, чтобы связать с каждой сетью или группой сетей в графическом языке сетевые метки, ограниченные справа двоеточием «:». Данная метка должна иметь форму идентификатора или десятичного целого без знака. Область видимости сети и ее метка должны быть локальными для программного компонента, в котором расположена сеть.

Графические языки используются для представления потока концептуальной величины через одну или большее число сетей, представляющих план управления, то есть:

- «Поток энергии»,

аналогичный потоку электрической энергии в электромеханической релейной системе обычно используется в релейно-контактных схемах.

Поток энергии в языке LD должен проходить слева направо.

- «Поток сигналов»,

аналогичен потоку сигналов между элементами системы обработки сигналов, типично используемому в функциональных блоковых диаграммах.

Поток сигналов в языке FBD должен проходить с выходной (правой) стороны функции или функционального блока ко входной (левой) стороне функции или функционального блока (блоков), соединенных таким образом.

- «Поток деятельности»,

аналогичен потоку управления между элементами организации или между шагами электромеханического секвенсора, типично используемого в последовательных функциональных схемах.

Поток деятельности между элементами SCF должен проходить от низа шага через соответствующий переход к верху соответствующего последующего шага (шагов).

8.1.5 Вычисление сетей

8.1.5.1 Общие положения

Порядок, в котором вычисляются сети и их элементы, не обязательно такой же, что и порядок, в котором они помечаются или выводятся на экран. Аналогично, нет необходимости, чтобы все сети вычислялись до того, как может быть повторено вычисление заданной сети.

Однако, когда тело программного компонента состоит из одной или нескольких сетей, результаты вычисления сети внутри указанного тела должны быть функционально эквивалентны соблюдению следующих правил:

a) Ни один элемент сети не вычисляется, пока не вычислены состояния всех его входов.

b) Вычисление элемента сети не является окончательным, пока не вычислены состояния всех его выходов.

c) Вычисление сети не завершено, пока не вычислены состояния выходов всех ее элементов, даже если сеть содержит один из элементов управления выполнением.

d) Порядок вычисления сети должен соответствовать положениям для языка LD и для языка FBD.

8.1.5.2 Обратная связь

Считается, что в сети имеется обратная связь, если выход функции или функционального блока используется как вход в функцию или функциональный блок, который предшествует ему в сети; а связанная переменная называется переменной обратной связи.

Например в приведенном ниже примере логическая переменная RUN является переменной обратной связи. Переменная обратной связи может также являться выходным элементом структуры данных функционального блока.

Обратные связи можно использовать в описываемых графических языках, в соответствии со следующими правилами:

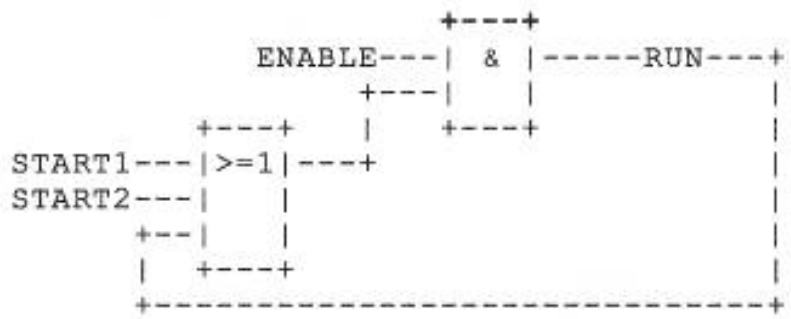
a) Заданные в явном виде циклы, такие как приведенный в примере ниже a), могут появляться только в языке FBD.

b) Пользователь должен иметь возможность использовать определяемые разработчиком средства для определения порядка выполнения элементов в явном виде, например, путем выбора переменных обратной связи для формирования заданного в неявном виде цикла, как показано в приведенном ниже примере b).

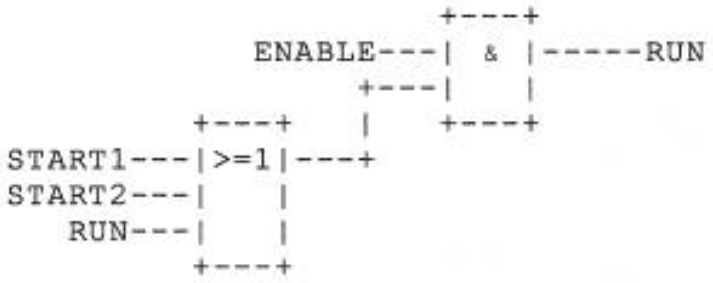
c) Переменные обратной связи должны инициализироватся одним из механизмов. Начальное значение используется во время первого вычисления сети. Ошибка возникает, если переменная обратной связи не инициализирована.

d) После того, как элемент с переменной обратной связи вычислен как выход, новое значение переменной обратной связи используется до следующего вычисления элемента.

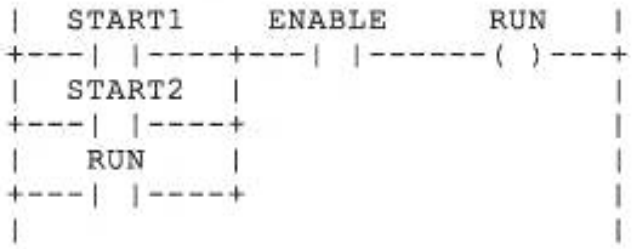
Пример — Обратная связь



a) Цикл, заданный в явном виде



b) Цикл, заданный в явном виде



c) Эквивалент в языке LD

8.1.6 Элементы управления выполнением

Передача управления программой в языках LD и FBD представляется графическими элементами, приведенными в таблице 73.

Переходы показываются логической сигнальной линией, завершающейся двойной пунктирной линией со стрелкой. Сигнальная линия для условия перехода должна начинаться у логической переменной, у логического выхода функции или функционального блока, или на линии потока энергии релейно-контактных схем. Передача управления программой назначенной сетевой происходит, когда логическое значение сигнальной линии равно (TRUE); поэтому безусловный переход — это особый случай условного перехода.

Целью перехода должна быть сетевая метка внутри тела программного компонента или тела метода, внутри которого происходит переход. Если переход происходит внутри конструкции ACTION... END_ACTION, то цель перехода должна находиться внутри той же конструкции.

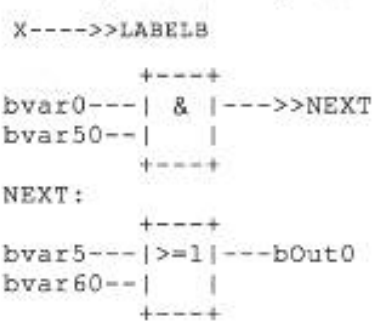
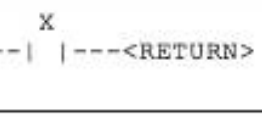
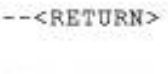
Условные возвраты от функций и функциональных блоков реализуются с использованием конструкции RETURN, как показано в таблице 73. Выполнение программы передается назад к вызывающему объекту, когда логический вход равен 1 (TRUE), и продолжается обычным способом, когда логический вход равен 0 (FALSE). Безусловные возвраты обеспечиваются физическим окончанием функции или функционального блока, или элементом RETURN, соединенным с левой шиной в языке LD, как показано в таблице 73.

Таблица 73 — Элементы управления графического выполнения

Номер	Описание	Объяснение
	Безусловный переход	
1a	язык FBD	`1---->>LABELA`

1b	язык LD	`|`
		`+---->>LABELA`
		`|`

Окончание таблицы 73

Номер	Описание	Объяснение
	Условный переход	
2a	язык FBD	Пример: Условие перехода, цель перехода 
2b	язык LD	Пример: Условие перехода, цель перехода 
	Условный возврат	
3a	язык LD	
3b	язык FBD	
	Безусловный возврат	
4	язык LD	

8.2 Релейно-контактные схемы (язык LD)

8.2.1 Общие положения

Подраздел 8.2 определяет язык LD для программирования релейно-контактных схем программируемых контроллеров.

Программа LD разрешает программируемому контроллеру тестировать и изменять данные с помощью стандартизированных графических символов. Данные символы размещаются в сетях способом, соответствующим «звену» релейно-контактных логических схем. Сети языка LD связаны слева и справа с помощью шин питания.

Использование букв, полуграфических и графических, для представления графических элементов определяется разработчиком и не является нормативным требованием.

8.2.2 Шины питания

Как показано в таблице 74, сеть языка LD ограничивается слева вертикальной линией, известной как левая шина питания, а справа — вертикальной линией известной как правая шина питания. Правая шина питания может задаваться в явном виде или подразумеваться.

8.2.3 Элементы и состояния связей

Как показано в таблице 74, элементы каналов могут быть горизонтальными или вертикальными. Состояние элемента связи обозначается «ON» или «OFF», в соответствии с литеральными логическими значениями 1 или 0, соответственно. Термин состояние связи является синонимом термина поток энергии.

Состояние левой шины считается равным ON во все моменты времени. Для правой шины состояние не определено.

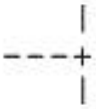
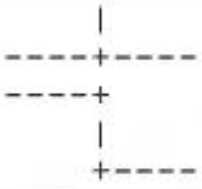
Элемент горизонтальной связи указывается горизонтальной линией. Элемент горизонтальной связи передает состояние элемента непосредственно слева от него элементу непосредственно справа от него.

Элемент вертикальной связи включает вертикальную линию, пересекающуюся с одним или более элементами горизонтальной связи на каждой из сторон. Состояние вертикальной связи представляет состояния включающего OR состояний ON горизонтальных связей на своей левой стороне, то есть со-стояние вертикальной связи равно:

- OFF, если состояние всех присоединенных к ней слева горизонтальных связей равно;
- ON, если состояние одной или более присоединенных к ней слева горизонтальных связей равно.

Состояние вертикальной связи копируется на все присоединенные к ней справа горизонтальные связи. Состояние вертикальной связи не копируется на какие-либо присоединенные к ней слева горизонтальные связи.

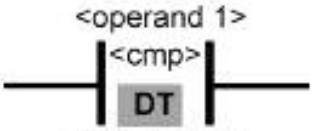
Таблица 74 — Шины питания и элементы связи

Номер	Описание	Символ
1	Левая шина питания (с присоединенной горизонтальной связью)	
2	Правая шина питания (с присоединенной горизонтальной связью)	
3	Горизонтальный канал	
4	Вертикальный канал (с присоединенными горизонтальными связями)	

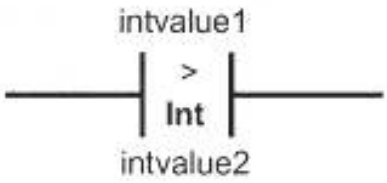
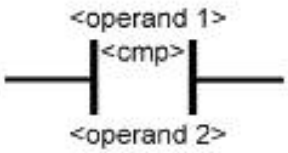
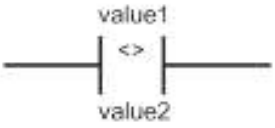
8.2.4 Контакты

Контакт — это элемент, который передает состояние на горизонтальную связь справа, что эквивалентно логическому AND состояния горизонтальной связи слева с соответствующей функцией связанного логического входа, выхода или переменной памяти. Контакт не изменяет значение связанной логической переменной. Стандартные символы контактов приведены в таблице 75.

Таблица 75 — Контакты

Номер	Описание	Объяснение, символ
	Статические контакты	
1	Нормально разомкнутый контакт	*** --( )- Состояние левой связи копируется в правую связь, если состояние связанной логической переменной (отмеченной «***») равно ON. В противном случае состояние правой связи равно OFF
2	Нормально замкнутый контакт	*** --(/)- Состояние левой связи копируется на правую связь, если состояние связанной логической переменной равно OFF. В противном случае состояние правой связи равно OFF
	Контакты, чувствительные к переходу	
3	Контакт, чувствительный к положительному переходу	*** --(P)- Состояние правой связи — это от одной оценки этого элемента до следующей, когда переход связанной переменной от OFF к ON распознает в то же время, что состояние левой связи равно ON. Состояние правой связи равно OFF во все другие моменты времени
4	Контакт, чувствительный к отрицательному переходу	*** --(N)- Состояние правой связи равно ON от одного вычисления этого элемента до следующего, когда переход связанной переменной из OFF в ON распознает в то же время, что состояние левой связи равно ON. Состояние правой связи равно OFF во все другие моменты времени
5а	Контакт сравнения (типизированный)	 Состояние правой связи равно ON от одного вычисления этого элемента до другого, когда левая связь равна ON, а результат <cmp> операндов 1 и 2 — это истинно Состояние правой связи должно быть OFF в противном случае <cmp> может быть заменено одной из функций сравнения, которая действительна для заданного типа данных DT — это тип данных обоих заданных операндов

Окончание таблицы 75

Номер	Описание	Объяснение, символ
		Пример —  Если левая связь равна ON и (intvalue1 > intvalue2), правая связь переключается в ON. Как intvalue1, так и intvalue2 относятся к типу данных INT
5b	Контакт сравнения (перегруженный)	 Состояние правой связи равно ON от одного вычисления этого элемента к следующего, когда левая связь равна ON, а результат <cmp> операндов 1 и 2 равен TRUE Состояние правой связи должно быть OFF в противном случае <cmp> может быть заменено одной из функций сравнения, которая действительна для типа данных операндов. Применяются правила, определенные в подразделе 6.6.1.7 ***Пример –***  Если левая связь ON и (value1 <> value2), то правая связь переключается ON

8.2.5 Катушки

Катушка копирует состояние связи слева от нее на связь справа от нее без измененеия, и сохраняет соответствующую функцию состояния или перехода левой связи в соответствующей логической переменной. Стандартные символы катушек приведены в таблице 76.

Пример — В цепи, приведенной ниже, значение логического выхода всегда TRUE, в то время как значение выходов c, d и e при завершении вычисления цепи равно значению входа b.

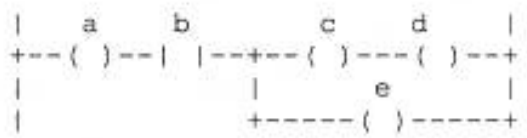


Таблица 76 — Катушки

Номер	Описание	Объяснение, символ
	Катушки без фиксации	
1	Катушка	*** --( )- Состояние левой связи копируется в связанную логическую переменную и на правую связь
2	Обратная обмотка	*** --(/)- Состояние левой связи копируется на правую связь. Обратная величина состояния левой связи копируется в связанную логическую переменную, то есть, если состояние левой связи равно OFF, то состояние связанной переменной равно ON, и наоборот
	Катушки с фиксацией	
3	Устанавливающая катушка (с фиксацией включения)	*** --(S)- Связанная логическая переменная устанавливается в состояние ON, когда левая связь находится в состоянии ON, и остается установленной до сброса катушкой RESET
4	Сбрасывающая катушка (с фиксацией выключения)	*** --(R)-- Связанная логическая переменная сбрасывается в состояние OFF, когда левая связь находится в состоянии ON и остается сброшенной до установки за счет катушкой SET
	Катушки, чувствительные к переходу	
8	Катушка, чувствительная к положительному переходу	*** --(P)- Состояние связанной логической переменной равно ON от одного вычисления этого элемента до другого, когда распознается переход левой связи из OFF в ON. Состояние левой связи всегда копируется в правую связь
9	Катушка, чувствительная к отрицательному переходу	*** --(N)- Состояние связанной логической переменной равно ON от одного вычисления этого элемента до другого, когда распознается переход левой связи из ON в OFF. Состояние левой связи всегда копируется в правую связь

8.2.6 Функции и функциональные блоки

Представление функций, методов и функциональных блоков в языке LD осуществляется со следующими исключениями:

a) фактические соединения переменной могут альтернативно показываться записью соответствующих данных или переменной вне блока рядом с формальным именем переменной внутри;

b) по крайней мере, один логический вход и один логический выход показывается на каждом блоке, чтобы разрешить поток энергии через блок.

8.2.7 Порядок оценки сети

Внутри программного компонента, записанного на LD, сети должны быть оценены в порядке сверху вниз по мере их появления в релейно-контактных схемах, исключая случай, когда этот порядок модифицируется элементами управления выполнением.

8.3 Функциональные блоковые диаграммы (FBD)

8.3.1 Общие положения

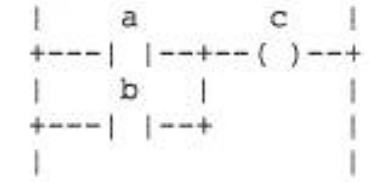
В подразделе 8.3 определяется графический язык FBD программируемых контроллеров, который соответствует, насколько это возможно, МЭК 60617-12. В случае, если существует конфликт между настоящим стандартом и МЭК 60617-12, положения настоящего стандарта будут применяться для программирования программируемых контроллеров в языке FBD.

8.3.2 Соединение элементов

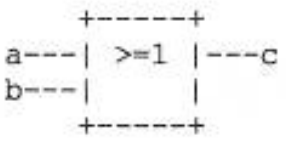
Элементы языка FBD связаны линиями прохождения сигнала, в соответствии с соглашениями 8.1.4.

Выходы функциональных блоков не соединяются вместе. В частности, конструкция «монтажного ИЛИ» языка не разрешена в языке FBD; вместо этого требуется заданный в явном виде блок логического «ИЛИ», как показано в примере, приведенном ниже.

Пример — Логическое ИЛИ/



a) «Монтажное ИЛИ» в языке LD



b) Функция в языке FBD

8.3.3 Порядок оценки сети

Когда программный компонент, записанный на языке FBD, содержит более одной сети, разработчик должен предоставить средства, с помощью которых пользователь может определить порядок вычисления сетей.

Приложение А
(обязательное)

Формальная спецификация элементов языка

Синтаксис текстовых языков определен в варианте «расширенной БНФ» (Бэкус-Науровой формы).

Синтаксис этого варианта РБНФ следующий:

Для целей настоящего приложения А, терминальные текстовые символы состоят из соответствующей строки символов, заключенной в парные одиночные кавычки. Например, терминальный символ, представляемый строкой символов ABC, представлен посредством 'ABC'.

Нетерминальные текстовые символы должны быть представлены строками букв нижнего регистра, числами и символом подчеркивания «_», начиная с буквы верхнего регистра.

Продукционные правила

Правила вывода для текстовых языков имеют форму
non_terminal_symbol : extended_structure;
Данное правило можно прочитать как: «non_terminal_symbol может включать extended_structure».
Расширенные структуры можно конструировать в соответствии со следующими правилами:
любой терминальный символ — расширенная структура;
любой нетерминальный символ — расширенная структура.

Если S — расширенная структура, то следующие выражения также являются расширенными структурами:

(S) — означает собственно S;

(S)* — замыкание, означающее нуль или большее число конкатенаций S;

(S)+ — замыкание, означающее одно или большее число сцеплений S;

(S)? — вариант, означающий нуль или одно появление S.

Если S1 и S2 — это расширенные структуры, тогда следующие выражения являются расширенными структурами:

S1 | S2 — изменение, означающее выбор S1 или S2;

S1 S2 — сцепление, означающее S1, за которым следует S2;

Сцепление предшествует изменению, то есть

S1 | S2 S3 — эквивалентно S1 | (S2 S3), S1 S2 | S3 — эквивалентно (S1 S2) | S3.

Если S — это расширенная структура, которая обозначает одиночный символ или изменение одиночных символов, тогда следующее также является расширенной структурой:

−(S) — отрицание, означающее любой одиночный символ, который не находится в S. Отрицание предшествует замыканию или опции, то есть

−(S)* — эквивалентно (−(S))*.

Следующие символы используются для обозначения определенных символов или классов символов:

. — любой одиночный символ;

\' — символ одиночной кавычки;

\n — новая строка;

\r — возврат каретки;

\t — табулятор.

Комментарии в грамматике начинаются двойной косой чертой и заканчиваются в конце строки:
// Это комментарий

// Таблица 1 — Наборы символов
// Таблица 2 — Идентификаторы

```
Буква       : 'A'..'Z' | '_';
Цифра       : '0'..'9';
Бит         : '0'..'1';
```

```
Octal_Digit          : '0'..'7';
Hex_Digit            : '0'..'9' | 'A'..'F';
Идентификатор        : Буква ( Буква | Цифра)*;

// Таблица 3 — Комментарии
Комментарий          : '//' ~( '\n' | '\r' )* '\r' ? '\n' {$channel=HIDDEN;}
                     | '(*' ( options{greedy=false;}: . )* '*)' {$channel=HIDDEN;}
                     | '/*' ( options{greedy=false;}: . )* '*/' {$channel=HIDDEN;};
WS                   : ( ' ' | '\t' | '\r' | '\n' ) {$channel=HIDDEN;}; // пробел
EOL                  : '\n';

// Таблица 4 — Прагма
Прагма               : '{' ( options{greedy=false;}: . )* '}' {$channel=HIDDEN;};

// Таблица 5 — Числовые литералы
Константа            : Numeric_Literal | Char_Literal | Time_Literal | Bit_Str_Literal | Bool_Literal;
Numeric_Literal      : Int_Literal | Real_Literal;
Int_Literal          : ( Int_Type_Name '#' )? ( Signed_Int | Binary_Int | Octal_Int | Hex_Int );
Unsigned_Int         : Digit ( '_' ? Digit )*;
Signed_Int           : ( '+' | '-' )? Unsigned_Int;
Binary_Int           : '2#' ( '_' ? Bit )+;
Octal_Int            : '8#' ( '_' ? Octal_Digit )+;
Hex_Int              : '16#' ( '_' ? Hex_Digit )+;
Real_Literal         : ( Real_Type_Name '#' )? Signed_Int '.' Unsigned_Int ( 'E' Signed_Int )?;
Bit_Str_Literal      : ( Multibits_Type_Name '#' )? ( Unsigned_Int | Binary_Int | Octal_Int | Hex_Int );
Bool_Literal         : ( Bool_Type_Name '#' )? ( '0' | '1' | 'FALSE' | 'TRUE' );

// Таблица 6 — Символьно-строковые литералы
// Таблица 7 — Двухсимвольные комбинации в символьных строках
Char_Literal         : ( 'STRING#' )? Char_Str;
Char_Str             : S_Byte_Char_Str | D_Byte_Char_Str;
S_Byte_Char_Str      : '\'' S_Byte_Char_Value + '\'';
D_Byte_Char_Str      : '"' D_Byte_Char_Value + '"';
S_Byte_Char_Value    : Common_Char_Value | '$\'' | '"' | '$' Hex_Digit Hex_Digit;
D_Byte_Char_Value    : Common_Char_Value | '\'' | '$"' | '$' Hex_Digit Hex_Digit Hex_Digit Hex_Digit;
Common_Char_Valu     : ' ' | '!' | '#' | '%' | '&' | '('..'/' | '0'..'9' | ':'..'@' | 'A'..'Z' | '['..'`' | 'a'..'z' | '{'..'~'
                     | '$$' | '$L' | '$N' | '$P' | '$R' | '$T';
                                   //любой печатаемый символ, за исключением $, 'and'

// Таблица 8 — Литералы длительности
// Таблица 9 — Литералы даты и времени суток
Time_Literal         : Duration | Time_Of_Day | Date | Date_And_Time;
Длительность         : ( Time_Type_Name | 'T' | 'LT' ) '#' ( '+' | '-' )? Интервал;
Fix_Point            : Unsigned_Int ( '.' Unsigned_Int )?;
Интервал             : Сутки | Часы | Минуты | Секунды | Миллисекунды | Микросекунды
                     | Наносекунды;
Сутки                : ( Fix_Point 'd' ) | ( Unsigned_Int 'd' '_' ? )? Часы?;
```

```
Часы                : ( Fix_Point 'h' ) | ( Unsigned_Int 'h' '_' ? )? Минуты?;
Минуты              : ( Fix_Point 'm' ) | ( Unsigned_Int 'm' '_' ? )? Секунды ?;
Секунды             : ( Fix_Point 's' ) | ( Unsigned_Int 's' '_' ? )? Миллисекунды?;
Миллисекунды        : ( Fix_Point 'ms' ) | ( Unsigned_Int 'ms' '_' ? )? Микросекунды?;
Микросекунды        : ( Fix_Point 'us' ) | ( Unsigned_Int 'us' '_' ? )? Наносекунды?;
Наносекунды         : Fix_Point 'ns';
Time_Of_Day         : ( Tod_Type_Name | 'LTIME_OF_DAY' ) '#' Daytime;
Daytime             : Day_Hour ':' Day_Minute ':' Day_Second;
Day_Hour            : Unsigned_Int;
Day_Minute          : Unsigned_Int;
Day_Second          : Fix_Point;
Дата                : ( Date_Type_Name | 'D' | 'LD' ) '#' Date_Literal;
Date_Literal        : Год '-' Месяц '-' День;
Год                 : Unsigned_Int;
Месяц               : Unsigned_Int;
День                : Unsigned_Int;
Date_And_Time       : ( DT_Type_Name | 'LDATE_AND_TIME' ) '#' Date_Literal '-' Daytime;
```

// Таблица 10 — Элементарные типы данных

```
Data_Type_Access    : Elem_Type_Name | Derived_Type_Access;
Elem_Type_Name      : Numeric_Type_Name | Bit_Str_Type_Name
                    | String_Type_Name | Date_Type_Name | Time_Type_Name;
Numeric_Type_Name   : Int_Type_Name | Real_Type_Name;
Int_Type_Name       : Sign_Int_Type_Name | Unsign_Int_Type_Name;
Sign_Int_Type_Name  : 'SINT' | 'INT' | 'DINT' | 'LINT';
Unsign_Int_Type_Name: 'USINT' | 'UINT' | 'UDINT' | 'ULINT';
Real_Type_Name      : 'REAL' | 'LREAL';
String_Type_Name    : 'STRING' ( '[' Unsigned_Int ']' )? | 'WSTRING' ( '[' Unsigned_Int ']' )? | 'CHAR' | 'WCHAR';
Time_Type_Name      : 'TIME' | 'LTIME';
Date_Type_Name      : 'DATE' | 'LDATE';
Tod_Type_Name       : 'TIME_OF_DAY' | 'TOD' | 'LTOD';
DT_Type_Name        : 'DATE_AND_TIME' | 'DT' | 'LDT';
Bit_Str_Type_Name   : Bool_Type_Name | Multibits_Type_Name;
Bool_Type_Name      : 'BOOL';
Multibits_Type_Name : 'BYTE' | 'WORD' | 'DWORD' | 'LWORD';
```

// Таблица 11 — Объявление определяемых пользователем типов данных и инициализации

```
Derived_Type_Access     : Single_Elem_Type_Access | Array_Type_Access | Struct_Type_Access
                        | String_Type_Access | Class_Type_Access | Ref_Type_Access
                        | Interface_Type_Access;
String_Type_Access      : ( Namespace_Name '.' )* String_Type_Name;
Single_Elem_Type_Access : Simple_Type_Access | Subrange_Type_Access
                        | Enum_Type_Access;
Simple_Type_Access      : ( Namespace_Name '.' )* Simple_Type_Name;
Subrange_Type_Access    : ( Namespace_Name '.' )* Subrange_Type_Name;
Enum_Type_Access        : ( Namespace_Name '.' )* Enum_Type_Name;
```

```
Array_Type_Access       : ( Namespace_Name '.' )* Array_Type_Name;
Struct_Type_Access      : ( Namespace_Name '.' )* Struct_Type_Name;
Simple_Type_Name        : Идентификатор;
Subrange_Type_Name      : Идентификатор;
Enum_Type_Name          : Идентификатор;
Array_Type_Name         : Идентификатор;
Struct_Type_Name        : Идентификатор;
Data_Type_Decl          : 'TYPE' ( Type_Decl ';' )+ 'END_TYPE';
Type_Decl               : Simple_Type_Decl | Subrange_Type_Decl | Enum_Type_Decl
                        | Array_Type_Decl | Struct_Type_Decl
                        | Str_Type_Decl | Ref_Type_Decl;
Simple_Type_Decl        : Simple_Type_Name ':' Simple_Spec_Init;
Simple_Spec_Init        : Simple_Spec ( ':=' Constant_Expr )?;
Simple_Spec             : Elem_Type_Name | Simple_Type_Access;
Subrange_Type_Decl      : Subrange_Type_Name ':' Subrange_Spec_Init;
Subrange_Spec_Init      : Subrange_Spec '(' ':=' Signed_Int )?;
Subrange_Spec           : Int_Type_Name '(' Subrange ')' | Subrange_Type_Access;
Subrange                : Constant_Expr '..' Constant_Expr;
Enum_Type_Decl          : Enum_Type_Name ':' ( ( Elem_Type_Name ? Named_Spec_Init )
                        | Enum_Spec_Init );
Named_Spec_Init         : '(' Enum_Value_Spec ( ',' Enum_Value_Spec )* ')'
                        ( ':=' Enum_Value )?;
Enum_Spec_Init          : ( ( '(' Identifier ( ',' Identifier )* ')' ) | Enum_Type_Access )
                        ( ':=' Enum_Value )?;
Enum_Value_Spec         : Identifier ( ':=' ( Int_Literal | Constant_Expr ) )?;
Enum_Value              : ( Enum_Type_Name '#' )? Identifier;
Array_Type_Decl         : Array_Type_Name ':' Array_Spec_Init;
Array_Spec_Init         : Array_Spec ( ':=' Array_Init )?;
Array_Spec              : Array_Type_Access | 'ARRAY' '[' Subrange ( ',' Subrange )* ']' 'OF' Data_Type_
Access;
Array_Init              : '[' Array_Elem_Init ( ',' Array_Elem_Init )* ']';
Array_Elem_Init         : Array_Elem_Init_Value | Unsigned_Int '(' Array_Elem_Init_Value ? ')';
Array_Elem_Init_Value   : Constant_Expr | Enum_Value | Struct_Init | Array_Init;
Struct_Type_Decl        : Struct_Type_Name ':' Struct_Spec;
Struct_Spec             : Struct_Decl | Struct_Spec_Init;
Struct_Spec_Init        : Struct_Type_Access ( ':=' Struct_Init )?;
Struct_Decl             : 'STRUCT' 'OVERLAP' ? ( Struct_Elem_Decl ';' )+ 'END_STRUCT';
Struct_Elem_Decl        : Struct_Elem_Name ( Located_At Multibit_Part_Access ? )? ':'
                        ( Simple_Spec_Init | Subrange_Spec_Init | Enum_Spec_Init
                        | Array_Spec_Init | Struct_Spec_Init );
Struct_Elem_Name        : Identifier;
Struct_Init             : '(' Struct_Elem_Init ( ',' Struct_Elem_Init )* ')';
Struct_Elem_Init        : Struct_Elem_Name ':=' ( Constant_Expr | Enum_Value | Array_Init
                        | Struct_Init | Ref_Value );
Str_Type_Decl           : String_Type_Name ':' String_Type_Name ( ':=' Char_Str )?;
```

```
// Таблица 16 — Прямо представленные переменные
Direct_Variable      : '%' ( 'I' | 'Q' | 'M' ) ( 'X' | 'B' | 'W' | 'D' | 'L' )? Unsigned_Int ( '.' Unsigned_Int )*;

// Таблица 12 — Операции со ссылками
Ref_Type_Decl        : Ref_Type_Name ':' Ref_Spec_Init;
Ref_Spec_Init        : Ref_Spec ( ':=' Ref_Value )?;
Ref_Spec             : 'REF_TO' + Data_Type_Access;
Ref_Type_Name        : Идентификатор;
Ref_Type_Access      : ( Namespace_Name '.' )* Ref_Type_Name;
Ref_Name             : Идентификатор;
Ref_Value            : Ref_Addr | 'NULL';
Ref_Addr             : 'REF' '(' ( Symbolic_Variable | FB_Instance_Name
                     | Class_Instance_Name ) ')'; Ref_Assign
Ref_Name '           : =' ( Ref_Name | Ref_Deref | Ref_Value );
Ref_Deref            : Ref_Name '^' +;

// Таблица 13 — Объявление переменных/Таблица 14 — Инициализация переменных
Переменная           : Direct_Variable | Symbolic_Variable;
Symbolic_Variable    : ( ( 'THIS' '.' ) | ( Namespace_Name '.' )+ )? ( Var_Access | Multi_Elem_Var );
Var_Access           : Variable_Name | Ref_Deref;
Variable_Name        : Идентификатор;
Multi_Elem_Var       : Var_Access ( Subscript_List | Struct_Variable )+;
Subscript_List       : '[' Subscript ( ',' Subscript )* ']';
Индекс               : Выражение;
Struct_Variable      : '.' Struct_Elem_Select;
Struct_Elem_Select   : Var_Access;
Input_Decls          : 'VAR_INPUT' ( 'RETAIN' | 'NON_RETAIN' )? ( Input_Decl ';' )* 'END_VAR';
Input_Decl           : Var_Decl_Init | Edge_Decl | Array_Conform_Decl;
Edge_Decl            : Variable_List ':' 'BOOL' ( 'R_EDGE' | 'F_EDGE' );
Var_Decl_Init        : Variable_List ':' ( Simple_Spec_Init | Str_Var_Decl | Ref_Spec_Init )
                     | Array_Var_Decl_Init | Struct_Var_Decl_Init
                     | FB_Decl_Init | Interface_Spec_Init;
Ref_Var_Decl         : Variable_List ':' Ref_Spec;
Interface_Var_Decl   : Variable_List ':' Interface_Type_Access;
Variable_List        : Variable_Name ( ',' Variable_Name )*;
Array_Var_Decl_Init  : Variable_List ':' Array_Spec_Init;
Array_Conformand     : 'ARRAY' '[' '*' ( ',' '*' :)* ']' 'OF' Data_Type_Access;
Array_Conform_Decl   : Variable_List ':' Array_Conformand;
Struct_Var_Decl_Init : Variable_List ':' Struct_Spec_Init;
FB_Decl_No_Ini       : FB_Name ( ',' FB_Name )* ':' FB_Type_Access;
FB_Decl_Init         : FB_Decl_No_Init ( ':=' Struct_Init )?;
FB_Name              : Идентификатор;
FB_Instance_Name     : ( Namespace_Name '.' )* FB_Name '^' *;
Output_Decls         : 'VAR_OUTPUT' ( 'RETAIN' | 'NON_RETAIN' )? ( Output_Decl ';' )* 'END_VAR';
Output_Decl          : Var_Decl_Init | Array_Conform_Decl;
In_Out_Decls         : 'VAR_IN_OUT' ( In_Out_Var_Decl ';' )* 'END_VAR';
```

```
In_Out_Var_Decl      : Var_Decl | Array_Conform_Decl | FB_Decl_No_Init;
Var_Decl             : Variable_List ':' ( Simple_Spec | Str_Var_Decl | Array_Var_Decl | Struct_Var_Decl );
Array_Var_Decl       : Variable_List ':' Array_Spec;
Struct_Var_Decl      : Variable_List ':' Struct_Type_Access;
Var_Decls            : 'VAR' 'CONSTANT' ? Access_Spec ? ( Var_Decl_Init ';' )* 'END_VAR';
Retain_Var_Decls     : 'VAR' 'RETAIN' Access_Spec ? ( Var_Decl_Init ';' )* 'END_VAR';
Loc_Var_Decls        : 'VAR' ( 'CONSTANT' | 'RETAIN' | 'NON_RETAIN' )?
                       ( Loc_Var_Decl ';' )* 'END_VAR';
Loc_Var_Decl         : Variable_Name ? Located_At ':' Loc_Var_Spec_Init;
Temp_Var_Decls       : 'VAR_TEMP' ( ( Var_Decl | Ref_Var_Decl | Interface_Var_Decl ) ';' )* 'END_VAR';
External_Var_Decls   : 'VAR_EXTERNAL' 'CONSTANT' ? ( External_Decl ';' )* 'END_VAR';
External_Decl        : Global_Var_Name ':'
                       ( Simple_Spec | Array_Spec | Struct_Type_Access | FB_Type_Access | Ref_Type_
                       Access );
Global_Var_Name      : Идентификатор;
Global_Var_Decls     : VAR_GLOBAL' ( 'CONSTANT' | 'RETAIN' )? ( Global_Var_Decl ';' )* 'END_VAR';
Global_Var_Decl      : Global_Var_Spec ':' ( Loc_Var_Spec_Init | FB_Type_Access );
Global_Var_Spec      : ( Global_Var_Name ( ',' Global_Var_Name )* ) | ( Global_Var_Name Located_At );
Loc_Var_Spec_Init    : Simple_Spec_Init | Array_Spec_Init | Struct_Spec_Init
                       | S_Byte_Str_Spec | D_Byte_Str_Spec;
Located_At           : AT' Direct_Variable;
Str_Var_Decl         : S_Byte_Str_Var_Decl | D_Byte_Str_Var_Decl;
S_Byte_Str_Var_Decl  : Variable_List ':' S_Byte_Str_Spec;
S_Byte_Str_Spec      : 'STRING' ( '[' Unsigned_Int ']' )? ( ': =' S_Byte_Char_Str )?;
D_Byte_Str_Var_Decl  : Variable_List ':' D_Byte_Str_Spec;
D_Byte_Str_Spec      : 'WSTRING' ( '[' Unsigned_Int ']' )? ( ':=' D_Byte_Char_Str )?;
Loc_Partly_Var_Decl  : 'VAR' ( 'RETAIN' | 'NON_RETAIN' )? Loc_Partly_Var * 'END_VAR';
Loc_Partly_Var       : Variable_Name 'AT' '%' ( 'I' | 'Q' | 'M' ) '*' ':' Var_Spec ';';
Var_Spec             : Simple_Spec | Array_Spec | Struct_Type_Access
                       | ( 'STRING' | 'WSTRING' ) ( '[' Unsigned_Int ']' )?;
```

// Таблица 19 — Объявление функции

```
Func_Name            : Std_Func_Name | Derived_Func_Name;
Func_Access          : ( Namespace_Name '.' )* Func_Name;
Std_Func_Name        : 'TRUNC' | 'ABS' | 'SQRT' | 'LN' | 'LOG' | 'EXP'
                       | 'SIN' | 'COS' | 'TAN' | 'ASIN' | 'ACOS' | 'ATAN' | 'ATAN2
                       | 'ADD' | 'SUB' | 'MUL' | 'DIV' | 'MOD' | 'EXPT' | 'MOVE'
                       | 'SHL' | 'SHR' | 'ROL' | 'ROR'
                       | 'AND' | 'OR' | 'XOR' | 'NOT'
                       | 'SEL' | 'MAX' | 'MIN' | 'LIMIT' | 'MUX'
                       | 'GT' | 'GE' | 'EQ' | 'LE' | 'LT' | 'NE'
                       | 'LEN' | 'LEFT' | 'RIGHT' | 'MID' | 'CONCAT' | 'INSERT'
                       | 'DELETE' | 'REPLACE' | 'FIND';
                                                  // неполный перечень
Derived_Func_Name    : Идентификатор;
```

```
Func_Decl          : 'FUNCTION' Derived_Func_Name ( ':' Data_Type_Access )? Using_Directive *
                     ( IO_Var_Decls | Func_Var_Decls | Temp_Var_Decls )* Func_Body 'END_FUNCTION';
IO_Var_Decls       : Input_Decls | Output_Decls | In_Out_Decls;
Func_Var_Decls     : External_Var_Decls | Var_Decls;
Func_Body          : Ladder_Diagram | FB_Diagram | Instruction_List | Stmt_List
                     | Other_Languages;
```

//Таблица 40 — Объявление типа функционального блока
//Таблица 41 — Объявление экземпляра функционального блока

```
FB_Type_Name       : Std_FB_Name | Derived_FB_Name;
FB_Type_Access     : ( Namespace_Name '.' )* FB_Type_Name;
Std_FB_Name        : 'SR' | 'RS' | 'R_TRIG' | 'F_TRIG' | 'CTU' | 'CTD' | 'CTUD' | 'TP' | 'TON' | 'TOF';
                                    // неполный перечень
Derived_FB_Name    : Идентификатор;
FB_Decl            : 'FUNCTION_BLOCK' ( 'FINAL' | 'ABSTRACT' )?
                     Derived_FB_Name Using_Directive *
                     ( 'EXTENDS' ( FB_Type_Access | Class_Type_Access ) )?
                     ( 'IMPLEMENTS' Interface_Name_List )?
                     ( FB_IO_Var_Decls | Func_Var_Decls | Temp_Var_Decls | Other_Var_Decls )*
                     ( Method_Decl )* FB_Body ? 'END_FUNCTION_BLOCK';
FB_IO_Var_Decls    : FB_Input_Decls | FB_Output_Decls | In_Out_Decls;
FB_Input_Decls     : 'VAR_INPUT' ( 'RETAIN' | 'NON_RETAIN' )? ( FB_Input_Decl ';' )* 'END_VAR';
FB_Input_Decl      : Var_Decl_Init | Edge_Decl | Array_Conform_Decl;
FB_Output_Decls    : 'VAR_OUTPUT' ( 'RETAIN' | 'NON_RETAIN' )? ( FB_Output_Decl ';' )* 'END_VAR';
FB_Output_Decl     : Var_Decl_Init | Array_Conform_Decl;
Other_Var_Decls    : Retain_Var_Decls | No_Retain_Var_Decls | Loc_Partly_Var_Decl;
No_Retain_Var_Decls : 'VAR' 'NON_RETAIN' Access_Spec ? ( Var_Decl_Init ';' )* 'END_VAR';
FB_Body            : SFC | Ladder_Diagram | FB_Diagram | Instruction_List | Stmt_List | Other_Languages;
Method_Decl        : 'METHOD' Access_Spec ( 'FINAL' | 'ABSTRACT' )? 'OVERRIDE' ?
                     Method_Name ( ':' Data_Type_Access )?
                     ( IO_Var_Decls | Func_Var_Decls | Temp_Var_Decls )* Func_Body 'END_METHOD';
Method_Name        : Идентификатор;
```

//Таблица 48 — Класс
//Таблица 50 — Текстовый вызов методов — Формальный и неформальный перечень параметров

```
Class_Decl         : 'CLASS' ( 'FINAL' | 'ABSTRACT' )? Class_Type_Name Using_Directive *
                     ( 'EXTENDS' Class_Type_Access )? ( 'IMPLEMENTS' Interface_Name_List )?
                     ( Func_Var_Decls | Other_Var_Decls )* ( Method_Decl )* 'END_CLASS';
Class_Type_Name    : Идентификатор;
Class_Type_Access  : ( Namespace_Name '.' )* Class_Type_Name;
Class_Name         : Идентификатор;
Class_Instance_Name : ( Namespace_Name '.' )* Class_Name '^' *;
Interface_Decl     : 'INTERFACE' Interface_Type_Name Using_Directive *
                     ( 'EXTENDS' Interface_Name_List )? Method_Prototype * 'END_INTERFACE';
Method_Prototype   : 'METHOD' Method_Name ( ':' Data_Type_Access )? IO_Var_Decls * 'END_METHOD';
Interface_Spec_Init : Variable_List ( ':=' Interface_Value )?;
Interface_Value    : Symbolic_Variable | FB_Instance_Name | Class_Instance_Name | 'NULL';
```

```
Interface_Name_List     : Interface_Type_Access ( ',' Interface_Type_Access )*;
Interface_Type_Name     : Идентификатор;
Interface_Type_Access   : ( Namespace_Name '.' )* Interface_Type_Name;
Interface_Name          :Идентификатор;
Access_Spec             : 'PUBLIC' | 'PROTECTED' | 'PRIVATE' | 'INTERNAL';
```

// Таблица 47 — Объявление программы

```
Prog_Decl           : 'PROGRAM' Prog_Type_Name
                      ( IO_Var_Decls | Func_Var_Decls | Temp_Var_Decls | Other_Var_Decls
                      | Loc_Var_Decls | Prog_Access_Decls )* FB_Body 'END_PROGRAM';
Prog_Type_Name      : Идентификатор;
Prog_Type_Access    : ( Namespace_Name '.' )* Prog_Type_Name;
Prog_Access_Decls   : 'VAR_ACCESS' ( Prog_Access_Decl ';' )* 'END_VAR';
Prog_Access_Decl    : Access_Name ':' Symbolic_Variable Multibit_Part_Access ?
                      ':' Data_Type_Access Access_Direction ?;
```

// Таблица 54—61 Последовательная функциональная схема (SFC)

```
SFC                 : Sfc_Network +;
Sfc_Network         : Initial_Step ( Step | Transition | Action )*;
Initial_Step        : 'INITIAL_STEP' Step_Name ':' ( Action_Association ';' )* 'END_STEP';
Step                : 'STEP' Step_Name ':' ( Action_Association ';' )* 'END_STEP';
Step_Name           : Идентификатор;
Action_Qualifier    : 'N' | 'R' | 'S' | 'P' | ( ( 'L' | 'D' | 'SD' | 'DS' | 'SL' ) ',' Action_Time );
Action_Time         : Duration | Variable_Name;
Indicator_Name      : Variable_Name;
Переход             : 'TRANSITION' Transition_Name ? ( '(' 'PRIORITY' ':=' Unsigned_Int ')' )?
                      'FROM' Steps 'TO' Steps ':' Transition_Cond 'END_TRANSITION';
Transition_Name     : Идентификатор;
Шаги                : Step_Name | '(' Step_Name ( ',' Step_Name )+ ')';
Transition_Cond     : ':=' Expression ';' | ':' ( FBD_Network | LD_Rung ) | ':=' IL_Simple_Inst;
Action              : 'ACTION' Action_Name ':' FB_Body 'END_ACTION';
```

// Таблица 62 — Конфигурация и определение ресурса

```
Config_Name             : Идентификатор;
Resource_Type_Name      : Идентификатор;
Config_Decl             : 'CONFIGURATION' Config_Name Global_Var_Decls ?
                          ( Single_Resource_Decl | Resource_Decl + ) Access_Decls ? Config_Init ?
                          : 'END_CONFIGURATION';
Resource_Decl           : 'RESOURCE' Resource_Name 'ON' Resource_Type_Name
                          Global_Var_Decls ? Single_Resource_Decl
                          'END_RESOURCE';
Single_Resource_Decl    : ( Task_Config ';' )* ( Prog_Config ';' )+;
Resource_Name           : Идентификатор;
Access_Decls            : 'VAR_ACCESS' ( Access_Decl ';' )* 'END_VAR';
Access_Decl             : Access_Name ':' Access_Path ':' Data_Type_Access Access_Direction ?;
```

```
Access_Path          : ( Resource_Name '.' )? Direct_Variable
                     | ( Resource_Name '.' )? ( Prog_Name '.' )?
                     ( ( FB_Instance_Name | Class_Instance_Name ) '.' )* Symbolic_Variable;
Global_Var_Access    : ( Resource_Name '.' )? Global_Var_Name ( '.' Struct_Elem_Name )?;
Access_Name          : Identifier;
Prog_Output_Access   : Prog_Name '.' Symbolic_Variable;
Prog_Name            : Identifier;
Access_Direction     : 'READ_WRITE' | 'READ_ONLY';
Task_Config          : 'TASK' Task_Name Task_Init;
Task_Name            : Identifier;
Task_Init            : '(' ( 'SINGLE' ':=' Data_Source ',' )?
                     ( 'INTERVAL':'=' Data_Source ',' )?
                     'PRIORITY' ':=' Unsigned_Int ')';
Data_Source          : Constant | Global_Var_Access | Prog_Output_Access | Direct_Variable;
Prog_Config          : 'PROGRAM' ( 'RETAIN' | 'NON_RETAIN' )? Prog_Name ( 'WITH' Task_Name )? ':'
                     Prog_Type_Access ( '(' Prog_Conf_Elems ')' )?;
Prog_Conf_Elems      : Prog_Conf_Elem ( ',' Prog_Conf_Elem )*;
Prog_Conf_Elem       : FB_Task | Prog_Cnxn;
FB_Task              : FB_Instance_Name 'WITH' Task_Name;
Prog_Cnxn            : Symbolic_Variable ':=' Prog_Data_Source | Symbolic_Variable '=>' Data_Sink;
Prog_Data_Source     : Constant | Enum_Value | Global_Var_Access | Direct_Variable;
Data_Sink            : Global_Var_Access | Direct_Variable;
Config_Init          : 'VAR_CONFIG' ( Config_Inst_Init ';' )* 'END_VAR';
Config_Inst_Init     : Resource_Name '.' Prog_Name '.' ( ( FB_Instance_Name
                     | Class_Instance_Name ) '.' )*
                     ( Variable_Name Located_At ? ':' Loc_Var_Spec_Init
                     | ( ( FB_Instance_Name ':' FB_Type_Access )
                     | ( Class_Instance_Name ':' Class_Type_Access ) ) ':=' Struct_Init );
```

// Таблица 64 — Пространство имен

```
Namespace_Decl       : 'NAMESPACE' 'INTERNAL' ? Namespace_H_Name Using_Directive *
                     Namespace_Elements
                     'END_NAMESPACE';
Namespace_Elements   : ( Data_Type_Decl | Func_Decl | FB_Decl
                     | Class_Decl | Interface_Decl | Namespace_Decl )+;
Namespace_H_Name     : Namespace_Name ( '.' Namespace_Name )*;
Namespace_Name       :Идентификатор;
Using_Directive      : 'USING' Namespace_H_Name ( ',' Namespace_H_Name )* ';';
POU_Decl             : Using_Directive *
                     ( Global_Var_Decls | Data_Type_Decl | Access_Decls
                     | Func_Decl | FB_Decl | Class_Decl | Interface_Decl
                     | Namespace_Decl )+;
```

// Таблица 67—70 Перечень инструкций (IL)

```
Instruction_List        : IL_Instruction +;
IL_Instruction          : ( IL_Label ':' )? ( IL_Simple_Operation | IL_Expr | IL_Jump_Operation
                        | IL_Invocation | IL_Formal_Func_Call
                        | IL_Return_Operator )? EOL +;
IL_Simple_Inst          IL_Simple_Operation | IL_Expr | IL_Formal_Func_Call;
IL_Label                : Идентификатор;
IL_Simple_Operation     : IL_Simple_Operator IL_Operand ? | Func_Access IL_Operand_List ?;
IL_Expr                 : IL_Expr_Operator '(' IL_Operand ? EOL + IL_Simple_Inst_List ? ')';
IL_Jump_Operation       : IL_Jump_Operator IL_Label;
IL_Invocation           : IL_Call_Operator ((( FB_Instance_Name | Func_Name
                        | Method_Name | 'THIS'
                        | ( ( 'THIS' '.' ( ( FB_Instance_Name | Class_Instance_Name ) '.' )* ) Method_Name ) )
                        ( '(' ( ( EOL + IL_Param_List ? ) | IL_Operand_List ? ) ')' )? ) | 'SUPER' '(' ')' );
IL_Formal_Func_Call     : Func_Access '(' EOL + IL_Param_List ? ')';
IL_Operand              : Constant | Enum_Value | Variable_Access;
IL_Operand_List         : IL_Operand ( ',' IL_Operand )*;
IL_Simple_Inst_List     : IL_Simple_Instruction +;
IL_Simple_Instruction   : ( IL_Simple_Operation | IL_Expr | IL_Formal_Func_Call ) EOL +;
IL_Param_List           : IL_Param_Inst * IL_Param_Last_Inst;
IL_Param_Inst           : ( IL_Param_Assign | IL_Param_Out_Assign ) ',' EOL +;
IL_Param_Last_Inst      : ( IL_Param_Assign | IL_Param_Out_Assign ) EOL +;
IL_Param_Assign         : IL_Assignment ( IL_Operand | ( '(' EOL + IL_Simple_Inst_List ')' ) );
IL_Param_Out_Assign     : IL_Assign_Out_Operator Variable_Access;
IL_Simple_Operator      : 'LD' | 'LDN' | 'ST' | 'STN' | 'ST?' | 'NOT' | 'S' | 'R'
                        | 'S1' | 'R1' | 'CLK' | 'CU' | 'CD' | 'PV'
                        | 'IN' | 'PT' | IL_Expr_Operator;
IL_Expr_Operator        : 'AND' | '&' | 'OR' | 'XOR' | 'ANDN' | '&N' | 'ORN'
                        | 'XORN' | 'ADD' | 'SUB' | 'MUL' | 'DIV'
                        | 'MOD' | 'GT' | 'GE' | 'EQ' | 'LT' | 'LE' | 'NE';
IL_Assignment           : Variable_Name ':=';
IL_Assign_Out_          : 'NOT' ? Variable_Name '=>';
Operator
IL_Call_Operator        : 'CAL' | 'CALC' | 'CALCN';
IL_Return_Operator      : 'RT' | 'RETC' | 'RETCN';
IL_Jump_Operator        : 'JMP' | 'JMPC' | 'JMPCN';
```

//Таблица 71—72 Язык структурированного текста

```
Expression              : Xor_Expr ( 'OR' Xor_Expr )*;
Constant_Expr           : Выражение;
                        //константное выражение за время компиляции должно оценивать до постоянного
                        значения
Xor_Expr                : And_Expr ( 'XOR' And_Expr )*;
And_Expr                : Compare_Expr ( ( '&' | 'AND' ) Compare_Expr )*;
Compare_Expr            : ( Equ_Expr ( ( '=' | '<>' ) Equ_Expr )* );
Equ_Expr                : Add_Expr ( ( '<' | '>' | '<=' | '>=' ) Add_Expr )*;
Add_Expr                : Term ( ( '+' | '-' ) Term )*;
```

```
Term                    : Power_Expr ( '*' | '/' | 'MOD' Power_Expr )*;
Power_Expr              : Unary_Expr ( '**' Unary_Expr )*;
Unary_Expr              : '-' | '+' | 'NOT' ? Primary_Expr;
Primary_Expr            : Constant | Enum_Value | Variable_Access | Func_Call | Ref_Value| '(' Expression ')';
Variable_Access         : Variable Multibit_Part_Access ?;
Multibit_Part_Access    : '.' ( Unsigned_Int | '%' ( 'X' | 'B' | 'W' | 'D' | 'L' ) ? Unsigned_Int );
Func_Call               : Func_Access '(' ( Param_Assign ( ',' Param_Assign )* )? ')';
Stmt_List               : ( Stmt ? ';' )*;
Stmt                    : Assign_Stmt | Subprog_Ctrl_Stmt | Selection_Stmt | Iteration_Stmt;
Assign_Stmt             : ( Variable ':=' Expression ) | Ref_Assign | Assignment_Attempt;
Assignment_Attempt      : ( Ref_Name | Ref_Deref ) '?=' ( Ref_Name | Ref_Deref | Ref_Value );
Invocation              : ( FB_Instance_Name | Method_Name | 'THIS'
                        | ( ( 'THIS' '.' )? ( ( ( FB_Instance_Name | Class_Instance_Name ) '.' )+ ) Method_Name ) )
                        '(' ( Param_Assign ( ',' Param_Assign )* )? ')';
Subprog_Ctrl_Stmt       : Func_Call | Invocation | 'SUPER' '(' ')' | 'RETURN';
Param_Assign            : ( ( Variable_Name ':=' )? Expression ) | Ref_Assign
                        | ( 'NOT' ? Variable_Name '=>' Variable );
Selection_Stmt          : IF_Stmt | Case_Stmt;
IF_Stmt                 : 'IF' Expression 'THEN' Stmt_List ( 'ELSIF' Expression 'THEN'
                        Stmt_List )* ( 'ELSE' Stmt_List )?
                        'END_IF';
Case_Stmt               : 'CASE' Expression 'OF' Case_Selection + ( 'ELSE' Stmt_List )? 'END_CASE';
Case_Selection          : Case_List ':' Stmt_List;
Case_List               :Case_List_Elem ( ',' Case_List_Elem )*;
Case_List_Elem          : Subrange | Constant_Expr;
Iteration_Stmt          : For_Stmt | While_Stmt | Repeat_Stmt | 'EXIT' | 'CONTINUE';
For_Stmt                : 'FOR' Control_Variable ':=' For_List 'DO' Stmt_List 'END_FOR';
Control_Variable        : Идентификатор;
For_List                : Expression 'TO' Expression ( 'BY' Expression )?;
While_Stmt              : 'WHILE' Expression 'DO' Stmt_List 'END_WHILE';
Repeat_Stmt             : 'REPEAT' Stmt_List 'UNTIL' Expression 'END_REPEAT';
```

// Таблица 73—76 Элементы графических языков

```
Ladder_Diagram          : LD_Rung *;
LD_Rung                 : «синтаксис для графических языков здесь не показан»;
FB_Diagram              : FBD_Network *;
FBD_Network             : «синтаксис для графических языков здесь не показан»;

// Здесь не рассматривается
Other_Languages         : «синтаксис для других языков здесь не показан»;
```

Приложение В
(справочное)

Перечень основных изменений и расширений третьего издания

Настоящий стандарт полностью совместим с МЭК 61131-3. Следующий перечень показывает основные изменения и расширения:

Редакционные исправления: Структура, нумерация, порядок, формулировки, таблицы свойств, термины и определения, такие как класс, метод, ссылка, сигнатура.

Формат таблицы соответствия.

Новые основные свойства:

- типы данных с явно выраженным типом размещения с именованными значениями;
- элементарные типы данных;
- ссылка, функции и операции со ссылкой;
- проверка ограниченного доступа к ANY_BIT;
- ARRAY переменной длины;
- присваивание начального значения;
- правила преобразования типа: неявная — явная функция — правила вызова, без значения, возвращаемого функцией;
- функции преобразования типов численных данных, поразрядных данных и т. д.;
- функции, чтобы связать и разделить время и дату;
- класс, включая метод, интерфейс и т. д.;
- объектно-ориентированный FB, включая метод, интерфейс и т. д.;
- пространства имен;
- структурированный текст CONTINUE и т. д.;
- релейно-контактные схемы. Контакты для сравнения (типизированные и перегруженные);

приложение А — Формальная спецификация для элементов языка.

Удаления (информативных частей):

- приложение — Примеры;
- приложение — Совместимость с МЭК 61499.

Депрекации:

- восьмеричный литерал;
- использование прямо представленных переменных в теле программных компонентов и методов;
- перегруженное усечение TRUNC;
- перечень инструкций (IL);
- «индикаторная» переменная блока действий.

Приложение ДА
(справочное)

Сведения о соответствии ссылочных международных стандартов национальным стандартам Российской Федерации

Таблица ДА.1

Обозначение ссылочного международного стандарта	Степень соответствия	Обозначение и наименование соответствующего национального стандарта
МЭК 61131-1	IDT	ГОСТ Р МЭК 61131-1—2016 «Контроллеры программируемые. Часть 1. Общая информация»
МЭК 61131-5	—	*
ИСО/МЭК 10646:2012	—	*
ИСО/МЭК/IEEE 60559	—	*

*Соответствующий национальный стандарт отсутствует. До его утверждения рекомендуется использовать перевод на русский язык данного международного стандарта. Перевод данного международного стандарта находится в Федеральном информационном фонде технических регламентов и стандартов.

Примечание — В настоящей таблице использовано следующее условное обозначение степени соответствия стандартов:

- IDT — идентичные стандарты.

Библиография

IEC 60050 (все части). International Electrotechnical Vocabulary (доступен на сайте http://www.electropedia.org)

IEC 60848. GRAFCET specification language for sequential function charts

IEC 60617. Graphical symbols for diagrams (доступен на сайте http://std.iec.ch/iec60617)

IEC 61499 (все части). Function blocks

ISO/IEC 14977:1996. Information technology — Syntactic Metalanguage — Extended BNF

ISO/AFNOR:1989. Dictionary of computer science

УДК 681.58:681.3:006.354 ОКС 25.040.40 35.240.50 IDT

Ключевые слова: контроллеры программируемые, языки программирования, структурные модели, печатные символы, прагма, программные компоненты, текстовые языки, графические языки

Редактор *Л.А. Кудрявцева*
Технический редактор *В.Ю. Фотиева*
Корректор *М.В. Бучная*
Компьютерная верстка *Е.А. Кондрашовой*

Сдано в набор 17.05.2016. Подписано в печать 10.06.2016. Формат 60×84⅛.
Гарнитура Ариал. Усл. печ. л. 26,50. Уч.-изд. л. 24,40.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

Издано во ФГУП «СТАНДАРТИНФОРМ», 123995 Москва, Гранатный пер., 4.
www.gostinfo.ru info@gostinfo.ru