MITSUBISHI ELECTRIC
Changes for the Better

ПЛК

Основы программирования (язык структурированного текста)

В данном курсе описан порядок создания программ базового уровня, используемых для управления программируемым контроллером MELSEC.
В рамках данного курса для описания программ применяется язык структурированного текста (ST).

© Mitsubishi Electric Corporation, 2015. All Rights Reserved.

L(FOD)00543RUS

Введение Цель курса

В данном курсе описан порядок создания управляющих программ на языке структурированного текста (ST) для программируемых контроллеров MELSEC.

Предварительное прохождение указанного ниже курса или владение аналогичными знаниями является обязательным условием для изучения настоящего курса:

Programming Basics (Основы программирования)

Обладание знаниями или опытом программирования на языках C или BASIC может оказать помощь в усвоении содержания данного курса.

Введение Структура курса

Данный курс имеет следующее содержание.

Глава 1. Обзор языка структурированного текста

В этой главе описываются особенности и возможные сферы применения языка структурированного текста (ST).

Глава 2. Основные правила написания программ на языке ST

В этой главе описываются основные правила, применяемые при создании программ на языке ST.

Глава 3. Создание программ управления входами/выходами

В этой главе описывается порядок создания программ управления входами/выходами.

Глава 4. Арифметические операции

В этой главе описывается порядок создания программ для выполнения арифметических операций.

Глава 5. Условное ветвление программы

В этой главе описывается условное ветвление программы.

Глава 6. Хранение и обработка данных

В этой главе описываются методы написания компактных программ для хранения и обработки данных.

Глава 7. Обработка данных, представленных в виде строк

В этой главе описываются методы обработки данных, представленных в виде строк.

Заключительный тест

Проходной балл: 60% или выше

Введение Как использовать этот инструмент электронного обучения

Переход к следующей странице		Переход к следующей странице.
Возврат к предыдущей странице		Возврат к предыдущей странице.
Переход к требуемой странице		Появится экран «Содержание», на котором вы сможете перейти к требуемой странице.
Завершение обучения		Завершение обучения.

Введение Меры предосторожности при использовании

Меры безопасности

Если вы обучаетесь с использованием реальных изделий, внимательно изучите правила техники безопасности, приведенные в соответствующих руководствах.

Меры предосторожности относительно данного курса

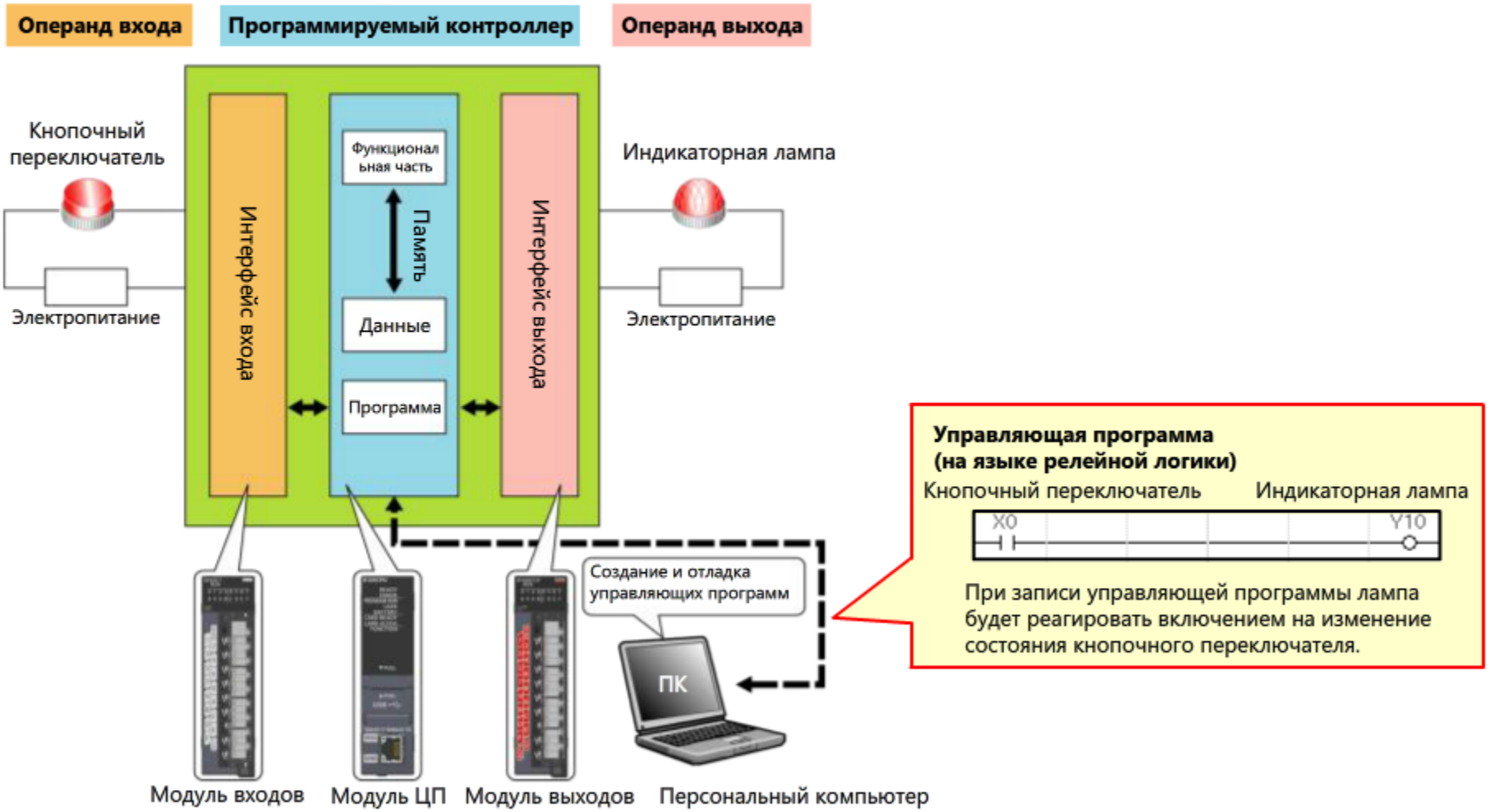
Отображаемые экраны используемого на практике инженерного программного обеспечения могут отличаться от представленных в данном курсе.
В данном курсе для создания программ применяются символы языка релейной логики MELSOFT GX Works3.

Глава 1 Обзор языка структурированного текста

В этой главе описываются особенности и возможные сферы применения языка структурированного текста (ST).

1.1 Управляющие программы

На представленном ниже рисунке показана конфигурация системы программируемого контроллера. Работа программируемых контроллеров осуществляется в соответствии с управляющими программами. Порядок функционирования программируемых контроллеров можно настроить в соответствии с собственными требованиями путем создания управляющих программ.



Языки программирования для программируемых контроллеров определяются международным стандартом, разработанным в организации International Electrotechnical Commission (IEC) (Международной электротехнической комиссии(МЭК)).

1.2 Особенности языка ST и его сравнение с другими языками программирования, отвечающими стандартам МЭК

Стандарт МЭК 61131 — это международный стандарт для систем программируемых контроллеров.
Языки программирования для программируемых контроллеров стандартизованы в соответствии с МЭК 61131-3.
Язык ST является одним из стандартных языков программирования.
Каждый язык обладает различными возможностями с точки зрения адаптации к прикладной области, в которой работает пользователь, и с учетом его навыков в программировании.

В таблице, представленной ниже, перечислены возможности языков программирования МЭК 61131-3.

Язык программирования	Возможности
Ladder **D**iagram (LD) (Диаграмма на языке релейной логики)	• Для создания программы, напоминающей по виду электрическую цепь, используются значки контактов и катушек. • Блок-схема программы проста для понимания и понятна даже для начинающих.
Structured **T**ext (ST) (Язык структурированного текста)	• Программы написаны с использованием текстов (символов). • Для тех пользователей, у кого есть опыт написания программ на языках C или BASIC, изучить язык ST будет просто. • Формулы для выполнения расчетов подобны математическим выражениям. Они просты для понимания. • Язык ST подходит для обработки данных.
Function **B**lock **D**iagram (FBD) (Язык диаграмм функциональных блоков)	• Программы записываются путем размещения блоков с различными функциями и указания отношений между блоками. • Использование FBD повышает читабельность программы, поскольку можно без труда рассмотреть операцию в целом.
Sequential **F**unction **C**hart (SFC) (Язык последовательных функциональных схем)	• Условия и технологические процессы записываются в виде блок-схем. • Блок-схема программы проста для понимания.
Instruction **L**ist (IL) (Язык списка команд)	• Язык IL подобен языку машинных команд. • На сегодняшний день язык IL применяется редко.

В этом курсе описывается порядок составления управляющих программ с использованием языка ST.

1.3 Сводная информация

Данная глава имеет следующее содержание:

- Взаимодействие между системами программируемых контроллеров и управляющими программами
- Международный стандарт для управляющих программ
- Особенности языка ST

Важные моменты для рассмотрения:

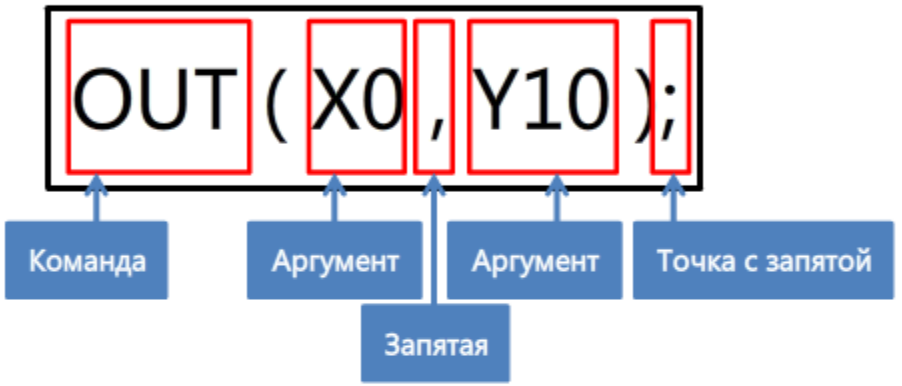
Взаимодействие между системами программируемых контроллеров и управляющими программами	• Работа программируемых контроллеров осуществляется в соответствии с управляющими программами. • Порядок функционирования программируемых контроллеров можно настроить в соответствии с собственными требованиями путем создания управляющих программ.
Международный стандарт для управляющих программ	• Язык ST является одним из языков программирования МЭК. • Среди других языков программирования МЭК можно отметить LD, FBD, SFC и IL, каждый из которых обладает различными возможностями для адаптации к прикладной области, в которой работает пользователь, и его навыкам в программировании.
Особенности языка ST	• Для тех пользователей, у кого есть опыт написания программ на языках C или BASIC, изучить язык ST будет просто. • Такие расчеты, как сложение и вычитание, можно записать в виде типичных, простых для понимания математических выражений. • Язык ST подходит для обработки данных.

Глава 2 Основные правила написания программ на языке ST

В этой главе описываются основные правила, применяемые при создании программ на языке ST.

2.1 Пример программы базового уровня (инструкция управления входом/выходом)

В данном разделе приводится иллюстрация в виде примера программы базового уровня на языке ST.
С помощью этой программы, используемой в качестве примера, выход Y10 переходит в состояние ВКЛ. после того, как в состояние ВКЛ. переходит вход X0, тогда как выход Y10 переходит в состояние ВЫКЛ. после перехода в состояние ВЫКЛ. входа X0.



Команда определяет операцию, которая будет выполнена.
Аргументы записываются в скобках после команды.
Они используются для описания переменных, арифметических выражений и значений констант.
В программируемых контроллерах MELSEC операнды модуля ЦП могут использоваться в качестве переменных.

Количество аргументов зависит от команды.
В случае использования нескольких аргументов они разделяются запятыми (,).

Одна приведенная выше строка представляет собой одну инструкцию. Каждая инструкция завершается точкой с запятой (;).
Программа записывается в виде сочетания инструкций.

2.2 Пример программы базового уровня (инструкция присваивания)

На следующем примере иллюстрируется программа, в которой используется инструкция присваивания.
В следующем операторе выполняется присваивание значения десятичной константы «5» переменной с именем «D10».

В этой инструкции присваивания используется оператор присваивания (:=). Следует заметить, что двоеточие (:) помещается слева от знака равенства (=).
С помощью оператора присваивания выполняется присваивание значения правой части — левой.

Добавление комментария в программу позволяет сделать операцию более понятной. Комментарий должен быть заключен между двумя звездочками (* *).

2.3 Система счисления

В примере, приведенном на предыдущей странице, переменной было присвоено десятичное значение.

В управлении с помощью последовательных программ иногда используются значения, отличные от десятичных, такие как двоичные и шестнадцатеричные.
В приведенной ниже таблице перечислены типы систем счисления, используемых в языке ST для программируемых контроллеров MELSEC.

Тип системы счисления	Система счисления	Пример
Двоичное представление	Должен добавляться префикс «2#».	**2#**11010
Восьмеричное представление	Должен добавляться префикс «8#».	**8#**32
Десятичное представление	Непосредственный ввод	26
	Должен добавляться префикс «K».	**K**26
Шестнадцатеричное представление	Должен добавляться префикс «16#».	**16#**1A
	Должен добавляться префикс «H»	**H**1A

Ниже представлены примеры программ, иллюстрирующих присваивание значений переменным.

```
D10 := 8#32;
D10 := K26;
D10 := H1A;
```

2.3.1 Битовое представление

Биты позволяют выразить состояния true/false (истина/ложь), такие как, например, состояния сигналов on/off (ВКЛ./ВЫКЛ.). С помощью битов также можно выразить реализацию/отсутствие реализации состояний.
В языке ST биты нельзя записать в виде «ON» и «OFF». Они могут быть выражены как «1» (ON) и «0» (OFF).
Биты также могут быть выражены как «TRUE» и «FALSE».

В приведенной ниже таблице перечислены различные представления данных.

Состояние	ON	OFF
	True	False
Числовое представление	1	0
Представление в виде true/false	TRUE	FALSE

Здесь представлены некоторые примеры присваивания значений переменным битового типа.

Числовое представление — Представление в виде true/false

```
X0 := 1;   =   X0 := TRUE;
```

Числовое представление — Представление в виде true/false

```
X0 := 0;   =   X0 := FALSE;
```

2.4 Последовательность выполнения программы

Инструкции языка ST выполняются в порядке сверху вниз.

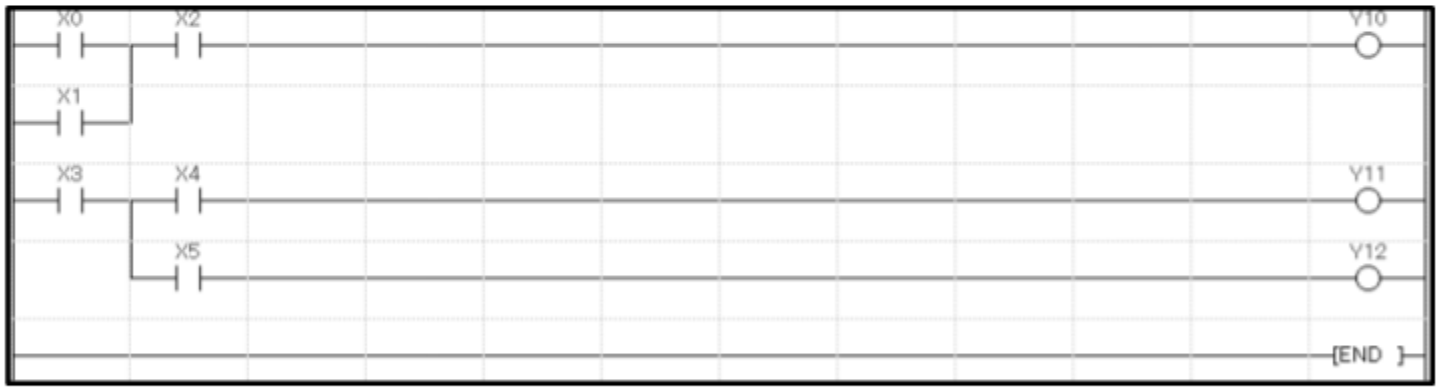
Пример программы на языке ST

```
Y10 := (X0 OR X1) AND X2;      (*Выполняется первой*)
Y11 := X3 AND X4;              (*Выполняется второй*)
Y12 := X3 AND X5;  (*Выполняется третьей. Инструкция END в конце не требуется.*)
```

Повторяемое выполнение

*Хотя инструкция END и требуется в конце программы на языке LD, в языке ST в ней нет необходимости.

В приведенной далее программе на языке релейной логики представлена та же операция, что и в примере программы на языке ST, приведенном выше.



Повторяемое выполнение

Как и в LD, команды на языке ST выполняются с повторением путем возврата к первой команде после достижения последней из них.

2.5 Сводная информация

Данная глава имеет следующее содержание:

- Программа базового уровня на языке ST
- Формат инструкции присваивания
- Система счисления
- Последовательность выполнения программы
- Комментарий

Важные моменты для рассмотрения:

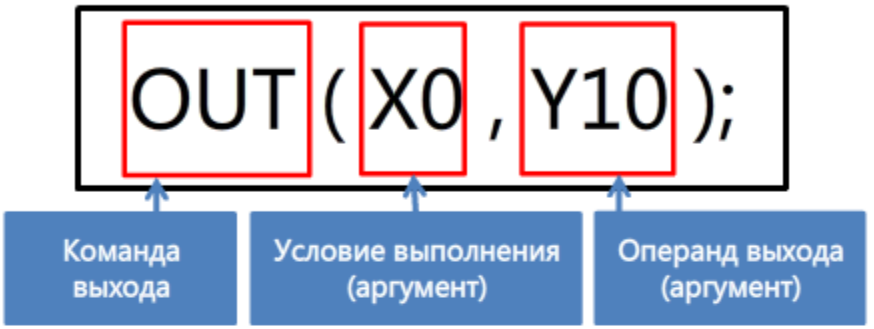
Программа базового уровня на языке ST	• Инструкция — это минимальный элемент программ на языке ST. • Каждая инструкция завершается точкой с запятой (;). • Программа записывается в виде сочетания инструкций.
Формат инструкции присваивания	• Для присваивания значений используется оператор присваивания (:=).
Система счисления	• Типы систем счисления в языке ST • Для битовых величин в языке ST используются значения «1» и «0» вместо представления в виде «ON» и «OFF». • Кроме того, битовые величины в ST могут задаваться в виде «TRUE» и «FALSE».
Последовательность выполнения программы	• Программы, составленные на языке ST, выполняются в порядке сверху вниз. • Как и в случае с программами на языке LD, процесс на языке ST выполняется с повторением, возвращаясь в начало программы после достижения конца процесса.
Комментарий	• Добавление комментария в программу позволяет сделать операцию более понятной. • Комментарии заключаются между двумя звездочками (* *).

Глава 3 Создание программ управления входами/выходами

В этой главе описывается порядок создания программ управления входами/выходами на языке ST.

3.1 Программы управления входами/выходами

Ниже приводится пример программы управления входами/выходами программируемого контроллера.



«OUT» — это команда выхода. Аргумент указывает на условие выполнения и операнд, куда направляется выход. Если условие выполнения X0 удовлетворяется, операнд Y10 принимает значение ВКЛ.

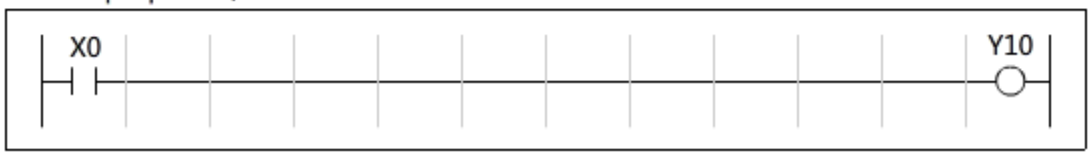
Нажмите на переключатель входа, показанный ниже. Переключатель входа X0 примет значение ВКЛ.

- Если переключатель входа X0 переходит в состояние ВКЛ., Y10 принимает значение ВКЛ.
- Если переключатель входа X0 переходит в состояние ВЫКЛ., Y10 принимает значение ВЫКЛ.

Пример программы управления входами/выходами, написанной на языке ST

```
OUT(X0,Y10);
```

Та же программа, написанная на языке LD



3.1 Программы управления входами/выходами

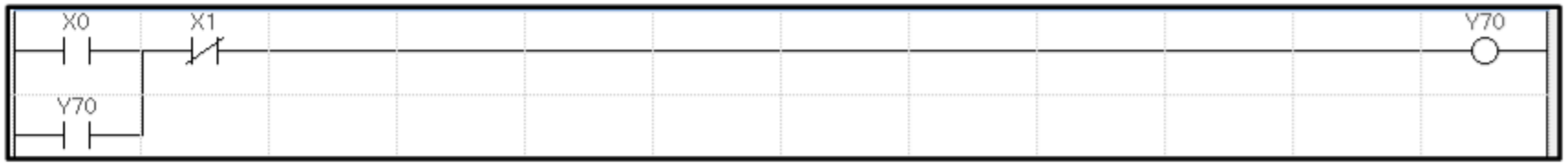
Как и в языке LD, имеется множество доступных команд помимо команды OUT — это команды управления входами/выходами и команды обработки данных.
См. руководство по программированию для вашего программируемого контроллера, чтобы получить более подробную информацию о доступных командах языка программирования ST.

Следует заметить, что запись операции «OUT(X0, Y10);» в виде «Y10 := X0;» приводит к тому же результату.

```
Y10 := X0; (*Та же операция, что и «OUT(X0, Y10);»*)
```

3.2 Сочетание нескольких условий

Приведенная ниже программа на языке релейной логики представляет собой самодостаточную цепь.



Эта же программа может быть записана на языке ST следующим образом.

```
Y70 := (X0 OR Y70) AND NOT X1;
```

Логический оператор

Как показано выше, логические операторы используются с целью записи на языке ST сочетаний нескольких условий.

В приведенной ниже таблице перечислены логические операторы.

Оператор	Значение
OR	Логическое ИЛИ
AND	Логическое И
NOT	Логическое отрицание
XOR	Исключающее ИЛИ

3.3 Определение значений переменных

При использовании с программируемыми контроллерами MELSEC языка ST как операнды, так и метки могут присваиваться в качестве альтернативных имен.
Пользователи могут применять метки в зависимости от приложений.

Если назначается метка, сопоставляемая с приложением, понимание операции упрощается.

```
Y10 := (X0 OR X1) AND X2; (*Записано с использованием имен операндов*)
```

```
Lamp := (Switch0 OR Switch1) AND Switch2;   (*Записано с использованием меток*)
```

Имена меток могут присваиваться с использованием инженерного программного обеспечения MELSOFT.

Последующие примеры программ, приведенные в данном курсе, представлены с использованием меток.

3.4 Сводная информация

Данная глава имеет следующее содержание:

Примеры программы управления входами/выходами

- Логические операторы используются с целью записи на языке ST сочетаний нескольких условий.
- Имена операндов и меток могут использоваться в качестве имен переменных.

Важные моменты для рассмотрения:

Сочетание нескольких условий	• Логические операторы используются с целью записи на языке ST сочетаний условий.
Определение значений переменных	• Если назначается метка, сопоставляемая с приложением, понимание операции упрощается.

Глава 4 Арифметические операции

В этой главе описывается порядок создания программ для выполнения арифметических операций.

- Описание арифметических операций
- Указание типов данных, соответствующих диапазонам чисел
- Присваивание имен переменным во избежание несоответствий в типах данных

4.1 Основные арифметические операции

В данном примере программы выполняется расчет суммарных объемов производства по двум производственным линиям. В правой части уравнения представлена арифметическая операция, в состав которой входят переменные и арифметические операторы.

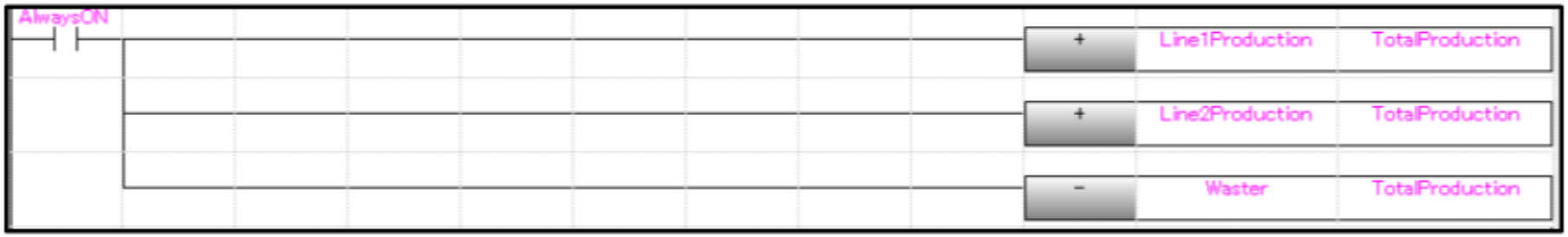
Пример арифметической программы, написанной на языке ST

Оператор сложения

Оператор вычитания

```
TotalProduction := Line1Production + Line2Production - Waster;
(*Суммируется объем производства по двум производственным линиям, из суммы вычитается количество бракованных изделий, а полученное значение присваивается соответствующей переменной. *)
```

Ниже представлена та же программа, написанная на языке LD.



Как показано выше, программа должна быть составлена в виде 3 строк на языке релейной логики, а с помощью ST ее можно записать в 1 строку.

В приведенной ниже таблице перечислены основные арифметические операторы.

Оператор	Значение
+	Сложение
–	Вычитание
*	Умножение
/	Деление

4.2 Типы данных переменных

Для каждой переменной должен быть указан тип данных, чтобы определить диапазон обрабатываемых значений. Типы данных для числовых величин, используемых в ST, таковы: бит, целое и действительное число.

Среди типов данных, используемых в ST, в приведенной ниже таблице отражены те типы данных, которые используются в данном курсе.

Тип данных		Диапазон данных
Битовый		Состояние ВКЛ./ВЫКЛ. для битовых операндов и состояния истина/ложь для результатов выполнения
Целое	Слово (без знака)	0—65 535
	Слово (со знаком)	– 32 768—32 767
	Двойное слово (без знака)	0—4 294 967 295
	Двойное слово (со знаком)	– 2 147 483 648—2 147 483 647

При использовании целого типа выберите тип слово или двойное слово в зависимости от диапазона используемых данных, а затем выберите, будет ли использоваться число со знаком или без знака в зависимости от потребностей в обработке отрицательных значений.
Укажите тип данных для переменной, если имя метки задается с помощью инженерного программного обеспечения MELSOFT.

4.3 Имена переменных, отражающие типы данных

Использование различных типов данных в правой и левой частях уравнения присваивания может привести к ошибке компиляции или непредвиденным результатам.
Внизу представлен пример такого случая.

```
ValueA := ValueB;    (* ValueA: слово, целое; ValueB: двойное слово, целое*)
```

Двойное слово, целое невозможно присвоить переменной типа слово, целое. При этом, в данном случае, тип данных определить нельзя.

К именам переменных можно добавлять префиксы, которые указывают тип данных, чтобы сделать его идентифицируемым визуально.
Такой тип присваивания имен известен как венгерская нотация.

Тип данных		Диапазон данных	Префикс	Расшифровка префикса
Битовый		Состояние ВКЛ./ВЫКЛ. для битовых операндов и состояния истина/ложь для результатов выполнения	b	**B**it (Битовый)
Целое	Слово (без знака)	0—65 535	u	**u**nsigned word (слово без знака)
	Слово (со знаком)	– 32 768—32 767	w	signed **w**ord (слово со знаком)
	Двойное слово (без знака)	0—4 294 967 295	ud	**u**nsigned **d**ouble-word (двойное слово без знака)
	Двойное слово (со знаком)	– 2 147 483 648—2 147 483 647	d	signed **d**ouble-word (двойное слово со знаком)

Пример программы, представленный в верхней части данной страницы, может быть записан следующим образом с применением венгерской нотации:

```
wValueA := dValueB; (*Переменная типа двойное слово не может быть присвоена переменной типа слово. *)
```

При использовании венгерской нотации несоответствие типа данных можно идентифицировать в процессе написания программы.

В оставшейся части курса имена переменных, используемые в примерах, записаны с использованием венгерской нотации.

4.4 Сводная информация

Данная глава имеет следующее содержание:

- Описание арифметических операций
- Указание типов данных, соответствующих диапазонам чисел
- Добавление имен переменных, отражающих типы данных

Важные моменты для рассмотрения:

Основные арифметические операции	• Операторы, которые являются общими для обычных языков программирования, могут использоваться в языке ST в случае быстрых вычислений.
Типы данных переменных	• Для каждой переменной должен быть указан тип данных, чтобы определить диапазон обрабатываемых значений.
Добавление имен переменных, отражающих типы данных	• Использование для описания имен венгерской нотации позволяет идентифицировать несоответствия типов данных переменных в процессе написания программ.

Глава 5 Условное ветвление программы

Управляющие программы также содержат участки кода, где фактическая обработка данных изменяется в зависимости от заданных условий.
В этой главе описывается условное ветвление программы.

5.1 Условное ветвление (IF)

Инструкции IF используются для условного ветвления программы. Инструкции IF описываются следующим образом.

```
IF conditional expression THEN
  Execution statement;      (*Осуществляется выполнение инструкции, если условное выражение удовлетворяется. *)
END_IF;                     (*END_IF; следует поместить в конце инструкций, относящихся к IF. *)
```

В программе, используемой в качестве примера, инструкция выполняется в том случае, если условное выражение удовлетворяется. Инструкция не выполняется, если условное выражение не удовлетворяется.

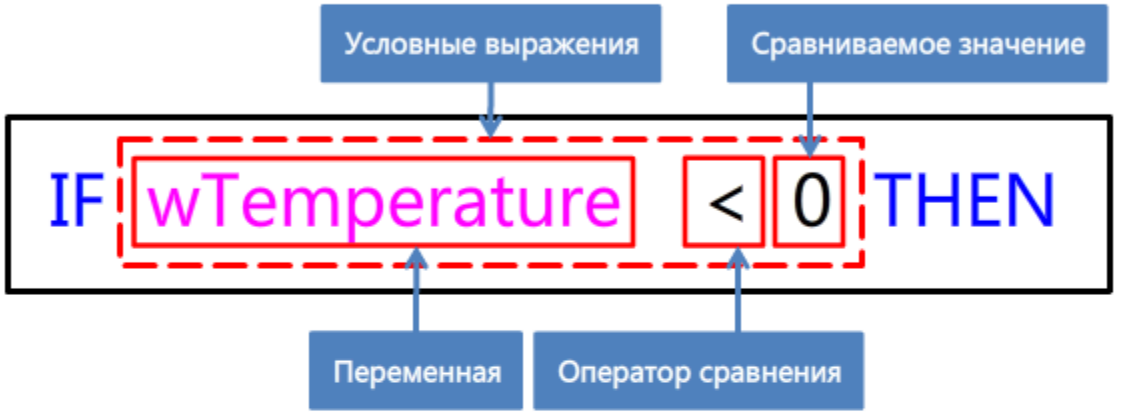
Приведенный ниже рисунок иллюстрирует блок-схему операции для данной программы, используемой в качестве примера.

Приведенный ниже рисунок иллюстрирует ветвление программы вследствие операции сравнения значений переменных.
В программе, используемой в качестве примера, нагреватель включается после того, как температура на панели управления падает до уровня ниже 0 градусов.

```
IF wTemperature < 0 THEN
  bHeater := 1; (*Нагреватель включается после того, как температура на панели управления падает до уровня ниже 0 градусов. *)
END_IF;
```

5.1.1 Написание условных выражений

На предыдущей странице было описано условное выражение «wTemperature < 0», которое означает «если значение переменной wTemperature меньше 0».
Как и в этом выражении, в условном выражении используются операторы сравнения, описывающие соотношение между значениями переменных и сравниваемыми с ними значениями.



В левой и в правой частях оператора сравнения сравниваемые значения записываются в виде переменных или констант.

Помимо сравнения переменных и констант, условные выражения могут записываться для сравнения переменных, а также для выполнения логических операций сравнения результатов или переменных битового типа.

Сравнение переменных

- uValue1 <= uValue2

Логическая операция для двух результатов сравнения

- (10 < uValue) AND (uValue <= 50)

Логическая операция с двумя переменными битового типа

- bSwitch0 OR bSwitch1

В приведенной ниже таблице перечислены типы операторов сравнения.

Оператор	Значение
>	Больше чем
<	Меньше чем
>=	Больше или равно
<=	Меньше или равно
=	Равно
<>	Не равно

5.1.2 Ветвление с использованием инструкции IF методом исключения (ELSE)

Простые инструкции IF (см. 5.1) используются в тех случаях, когда условное выражение удовлетворяется.
Чтобы выполнить другую инструкцию, если условное выражение не удовлетворяется, используется инструкция ELSE.

```
IF conditional expression THEN
  Execution statement 1;      (*Инструкция 1 выполняется, если условное выражение удовлетворяется. *)
  ELSE
  Execution statement 2;      (*Инструкция 2 выполняется, если условное выражение не удовлетворяется*)
END_IF;
```

Приведенный ниже рисунок иллюстрирует блок-схему операции для случая использования инструкции ELSE.

В приведенном ниже примере программы выполняются различные инструкции в зависимости от того, удовлетворяется или нет указанное условие.

Пример программы, приведенный в разделе 5.1, имеет недостаток, который состоит в том, что нагреватель продолжает увеличивать температуру после достижения 0 градусов. Тогда как приведенная ниже программа отключает нагреватель после того, как значение «wTemperature» превысит 0 градусов.

```
IF wTemperature < 0 THEN
  bHeater := 1; (*Включает нагреватель, если температура падает ниже 0 градусов. *)
  ELSE
  bHeater := 0; (*Отключает нагреватель после того, как значение температуры достигает
                  уровня 0 градусов или превышает его*)
END_IF;
```

5.1.3 Ветвление с использованием инструкции IF методом добавления (ELSEIF)

Инструкции ELSE используются для выполнения другой инструкции в тех случаях, когда условное выражение не удовлетворяется. С помощью инструкций ELSIF можно добавить еще одно условное ветвление. Оно означает, что если предыдущее условное выражение не удовлетворяется, тогда выполняется проверка следующего условного выражения.

```
IF Conditional expression 1 THEN
  Execution statement 1;     (*Инструкция 1 выполняется в том случае, если условное выражение 1 удовлетворяется. *)
  ELSIF Conditional expression 2 THEN
  Execution statement 2;     (*Инструкция 2 выполняется в том случае, если условное выражение 1 не удовлетворяется, а условное выражение 2 удовлетворяется. *)
  ELSE
  Execution statement 3;     (*Инструкция 3 выполняется в том случае, если условные выражения 1 и 2 не удовлетворяются. *)
END_IF;
```

Приведенный ниже рисунок иллюстрирует блок-схему операции для случая использования инструкции ELSEIF.

Инструкция ELSIF добавлена в пример программы, представленный в разделе 5.1.2, чтобы учесть случай, когда температура превышает 40 градусов.

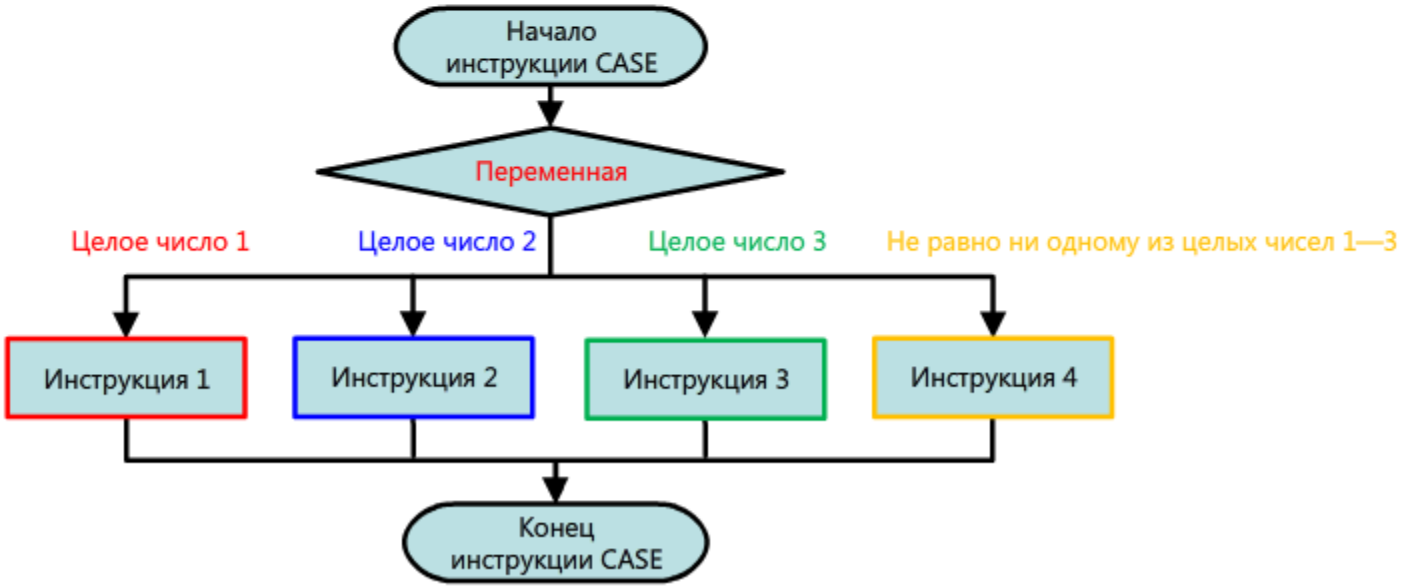
```
IF wTemperature < 0 THEN
   bHeater := 1; (*Включает нагреватель, если температура падает ниже 0 градусов. *)
   bCooler := 0; (*Отключает охладитель после того, как температура падает ниже 0 градусов. *)
   ELSIF 40 < wTemperature THEN
   bHeater := 0; (*Отключает нагреватель после того, как значение температуры превышает уровень 40 градусов. *)
   bCooler := 1; (*Включает охладитель после того, как значение температуры превышает уровень 40 градусов. *)
   ELSE
   bHeater := 0; (*Отключает нагреватель после того, как ни одно из вышеуказанных условий не удовлетворяется. *)
   bCooler := 0; (*Отключает охладитель после того, как ни одно из вышеуказанных условий не удовлетворяется. *)
END_IF;
```

5.2 Условное ветвление в зависимости от целочисленных значений (CASE)

Инструкции IF используются для ветвления с учетом того, удовлетворяется или нет условное выражение.
Инструкции CASE используются для ветвления в зависимости от значения целого числа.
На приведенном ниже рисунке записана инструкция CASE.

```
CASE Variable OF
    Integer value 1: Execution statement 1;  (*Инструкция 1 выполняется в случае, если значение переменной равно целому числу 1. *)
    Integer value 2: Execution statement 2;  (*Инструкция 2 выполняется в случае, если значение переменной равно целому числу 2. *)
    Integer value 3: Execution statement 3;  (*Инструкция 3 выполняется в случае, если значение переменной равно целому числу 3. *)
    ELSE Execution statement 4;  (*Инструкция 4 выполняется в случае, если значение переменной не равно никакому из целых чисел. *)
END_CASE;                        (*«END_CASE;» следует поместить в конце инструкции CASE. *)
```

Приведенный ниже рисунок иллюстрирует блок-схему операции для случая использования инструкции CASE.



5.2.1 Пример программы с инструкцией CASE

Выполнение инструкции CASE описывается с использованием примера программы.

Нажмите , чтобы перейти к следующей странице.
Для просмотра анимации снова нажмите кнопку «Play» (Воспроизведение).

Воспроизведение


```
CASE wWeight OF
  0..20:    uSize := 1;
  21..30:   uSize := 2;
  31..40:   uSize := 3;
  ELSE      uSize := 4;
END_CASE;
```

Вес	uSize	Условный размер
0—20 кг	1	M
21—30 кг	2	L
31—40 кг	3	XL
41 кг и более	4	Oversize

5.3 Сводная информация

Данная глава имеет следующее содержание:

- Условное ветвление с использованием инструкции IF
- Написание условных выражений
- Условное ветвление в зависимости от целочисленных значений (инструкция CASE)

Важные моменты для рассмотрения:

Инструкция IF	• С помощью инструкции IF происходит ветвление программы в тех случаях, когда условное выражение удовлетворяется. • Инструкция ELSE используется для ветвления в тех случаях, когда условное выражение не удовлетворяется. • Инструкция ELSIF используется для добавления еще одного ветвления в тех случаях, когда условное выражение в инструкции IF не удовлетворяется.
Условное выражение	• Условные выражения представляют собой отношения между переменными и значениями, сравниваемыми с ними с помощью операторов сравнения.
Инструкция CASE	• Инструкции CASE используются для ветвления в зависимости от значения целого числа.

Глава 6 Хранение и обработка данных

Также, как и в случае приложений, предназначенных для управления входами/выходами, современные программируемые контроллеры используются для обработки больших объемов данных, представляющих собой ядро производственных систем.
Чтобы иметь возможность обрабатывать большие объемы данных, такие данные должны быть сохранены, а затем считаны в случае необходимости.
В этой главе описываются методы написания компактных программ для хранения и обработки данных.

- С целью упорядочивания и организации значений переменных применяются массивы.
- Для организации связанных между собой переменных используются структуры данных.
- Программы, построенные на основе циклов, осуществляют эффективную обработку массивов с использованием инструкций FOR.

Имеется возможность создавать компактные программы для хранения и обработки данных с использованием массивов, структур и инструкций FOR.

6.1 Упорядочивание и хранение данных (массив)

При использовании массивов множество значений подвергается обработке с помощью одной переменной.
В представленном ниже примере данные по объему производства на предприятии автомобильной промышленности хранятся в зависимости от страны назначения.

Страна назначения	Страна A	Страна B	Страна C
Объем производства	35	75	65

Переменной присваиваются значения, соответствующие данным по объему производства в зависимости от страны.
Если не использовать массивы, то для каждой страны назначения потребуется создать свою переменную.
При использовании же массивов объемы производства по нескольким странам назначения можно задавать и хранить в одной переменной.

Если не использовать массив

uProductionA
uProductionB
uProductionC

Если использовать массив

uProduction

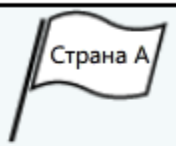
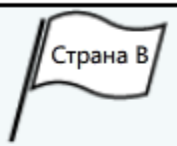
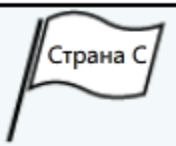
Отдельные переменные внутри массива задаются в соответствии с номером элемента. Номера элементов начинаются с [0].

В приведенном ниже примере программы переменной присваивается значение, соответствующее планируемому объему производства по стране A.

```
uShowProductionPlan := uProduction[0];
(*Задает номер элемента для страны A.*)
```

6.1.1 Двумерный массив (матрица)

В дальнейшем, помимо данных о стране назначения, используются данные о цвете окраски кузова.

Страна назначения									
Цвет окраски кузова									
Объем производства	10	5	20	15	40	20	25	30	10
	Итого 35			Итого 75			Итого 65		

Как показано в приведенной ниже таблице, данные можно разделить и хранить в соответствии с цветом окраски кузова (столбец) по каждой стране назначения (строка).

Цвет окраски кузова (столбец)

Страна назначения (строка)	Красный	Желтый	Синий
Страна A	[0,0] 10	[0,1] 5	[0,2] 20
Страна B	[1,0] 15	[1,1] 40	[1,2] 20
Страна C	[2,0] 25	[2,1] 30	[2,2] 10

Номер элемента, указывающий на страну назначения

Номер элемента, указывающий на цвет окраски кузова

Массивы, в которых данные организованы таким образом, в виде строк и столбцов, называются матрицами. Номера элементов, указывающие на номера строк и столбцов, отделяются друг от друга запятой.

Переменная-массив (матрица)

```
uProduction [1,1]
```

6.1.2 Присваивание значений элементам матрицы

С помощью матрицы в приведенном ниже примере программы выполняется присваивание значения, соответствующего количеству автомобилей, которое должно быть срочно выпущено в дополнение к запланированному объему производства автомобилей желтого цвета для страны назначения B.

```
uAdditionalProduction := 5;
uProduction[1,1] := uProduction[1,1] + uAdditionalProduction;
(*Добавляется объем производства (5 единиц) в дополнение к изначально запланированному. *)
```

Цвет окраски кузова (столбец)

Страна назначения (строка)	Красный	Желтый	Синий
Страна A	[0,0] 10	[0,1] 5	[0,2] 20
Страна B	[1,0] 15	[1,1] 40 -> 45	[1,2] 20
Страна C	[2,0] 25	[2,1] 30	[2,2] 10

6.1.3 Обработка информации, хранящейся в матрицах

В приведенном ниже примере программы с помощью матрицы выполняется расчет суммарного объема производства автомобилей всех цветов кузова для страны назначения С, а также присваивание значения переменной.

```
uProductionToday := uProduction[2,0] + uProduction[2,1] +uProduction[2,2];
(*Рассчитывается суммарный плановый объем на сегодняшний день всех цветов кузова для страны С и присваивается значение переменной «uProductionToday». *)
```

Страна назначения	Страна A			Страна B			Страна C		
Цвет окраски кузова									
Объем производства	10	5	20	15	45	20	25	30	10
	Итого 35			Итого 80			Итого 65		

Цвет окраски кузова (столбец)

Страна назначения (строка)	Красный	Желтый	Синий
Страна A	[0,0] 10	[0,1] 5	[0,2] 20
Страна B	[1,0] 15	[1,1] 45	[1,2] 20
Страна C	[2,0] 25	[2,1] 30	[2,2] 10

6.2 Использование циклов (FOR)

Ниже снова представлен пример программы (присваивается значение планового объема выпуска продукции на сегодняшний день), приведенный на предыдущей странице.

```
uProductionToday := uProduction[2,0] + uProduction[2,1] + uProduction[2,2];
```

При использовании данного примера программы, в случае если увеличивается количество цветов окраски кузова, добавляются новые переменные. Это приводит к тому, что выражение становится более длинным, и читать его становится труднее.

```
uProductionToday := uProduction[2,0] + uProduction[2,1] + uProduction[2,2]
                  + uProduction[2,3] + uProduction[2,4] + uProduction[2,5]...
```

В таком случае можно использовать инструкцию цикла, чтобы сделать программный код более понятным.

К инструкциям цикла относятся такие, как FOR, WHILE и REPEAT. В данном курсе рассматриваются инструкции FOR.

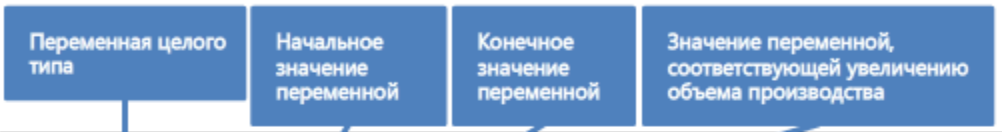
Инструкции FOR описываются следующим образом.

```
FOR variable := initial value TO final value BY increments DO
   Execution statement; (*Инструкция выполняется в цикле до тех пор, пока не будет достигнуто конечное значение. *)
END_FOR;                (*END_FOR следует поместить в конце инструкций, относящихся к FOR.  *)
```

Выполнение инструкции повторяется до тех пор, пока не будет достигнуто конечное значение переменной. После этого будет выполнен код, относящийся к «END_FOR;».

6.2 Использование циклов (FOR)

С помощью инструкции FOR в приведенном ниже примере программы рассчитывается запланированный объем производства автомобилей всех цветов кузова для страны C.




```
uProductionToday := 0;                    (*Инициализируется переменная. *)
FOR wColor := 0 TO 2 BY 1 DO
  uProductionToday := uProductionToday + uProduction[2,wColor];   (*Добавляется запланированный объем производства. *)
END_FOR;
```

При использовании инструкции FOR значение переменной «wColor» увеличивается на единицу, начиная с нулевого начального значения. Выполнение инструкции повторяется до тех пор, пока значение переменной не достигнет конечной величины, равной двум.
При выполнении инструкции переменная «wColor» выступает в качестве второго измерения для элемента массива «uProduction».
Значение переменной «wColor» увеличивается каждый раз при повторном выполнении инструкции. Для получения итоговой суммы каждый раз добавляется запланированный объем производства по каждому цвету окраски кузова.

В данном примере программы цикл выполняется три раза. (Первый раз: красный [0] => второй раз: желтый [1] => третий раз: синий [2])

Работа данной программы проиллюстрирована на следующей странице.

6.2 Использование циклов (FOR)

Выполнение инструкции FOR описывается на примере работы данной программы.

Массив значений прогнозируемых объемов производства

	Красный	Желтый	Синий
Страна A	[0,0] 10	[0,1] 5	[0,2] 20
Страна B	[1,0] 15	[1,1] 45	[1,2] 20
Страна C	[2,0] 25	[2,1] 30	[2,2] 10

Нажмите , чтобы перейти к следующей странице.
Для просмотра анимации снова нажмите кнопку «Play» (Воспроизведение).

Воспроизведение

```
uProductionToday := 0;

FOR wColor := 0 TO 2 BY 1 DO
        2
   uProductionToday := uProductionToday + uProduction[2,wColor];
                                          65
END_FOR;
```

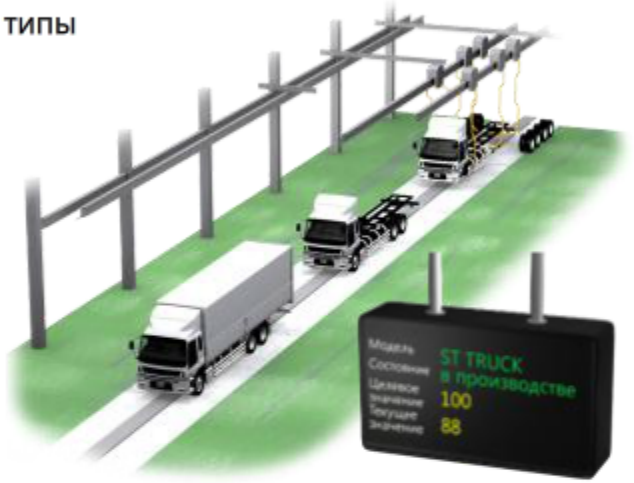
Number of repetition of the loop: 3

6.3 Хранение взаимосвязанных данных (структура)

Использование структуры позволяет одним именем переменной представить несколько связанных между собой других переменных.
В приведенном ниже примере на информационном табло отображается состояние линии по производству автомобилей.

В таблице, представленной ниже, перечислены имена переменных, значения и типы данных, соответствующие отображаемым позициям.

Наименование	Имя переменной	Значение	Тип данных переменной
Модель	sModel	'ST TRUCK'	Текстовая строка
Состояние	bStatus	'в производстве'	Битовый тип
Целевой объем производства на сегодняшний день	uPlanQty	'100'	Целое, тип слово (без знака)
Текущий объем производства	uActualQty	'88'	Целое, тип слово (без знака)



Если структура не применяется, то в случае наличия нескольких производственных линий имена переменных необходимо менять для каждой производственной линии.
Ниже приводятся примеры имен переменных по производственным линиям.

Первая производственная линия

s1stLineModel
b1stLineStatus
u1stLinePlanQty
u1stLineActualQty

Вторая производственная линия

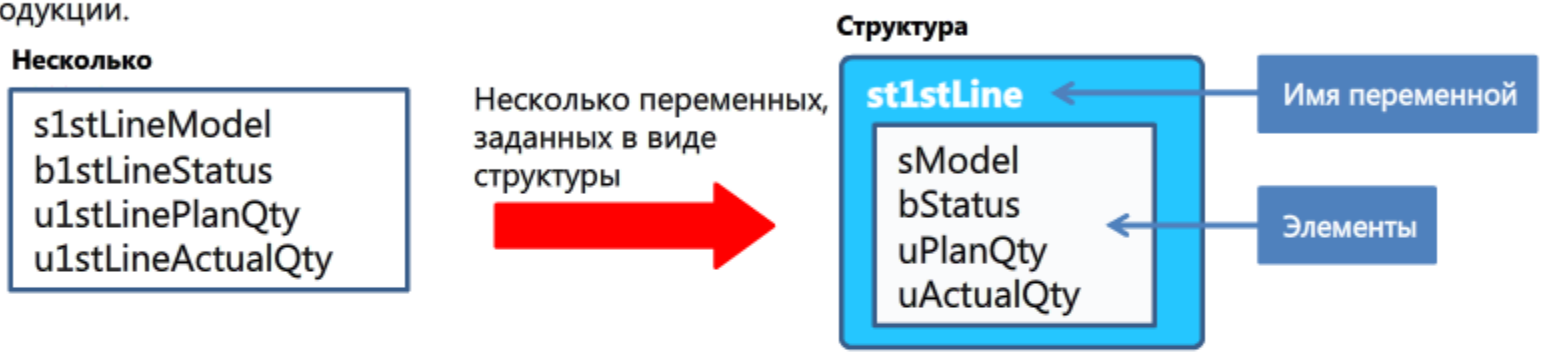
s2ndLineModel
b2ndLineStatus
u2ndLinePlanQty
u2ndLineActualQty

● ● ●

Если количество производственных линий увеличивается, число переменных также увеличится. Это приводит к тому, что программа становится более длинной, и читать ее становится труднее.

6.3 Хранение взаимосвязанных данных (структура)

Использование структуры позволяет одним именем переменной представить несколько связанных между собой других переменных, относящихся к одной производственной линии.
Подобные структуры используются для организации, хранения и обработки данных в виде пакета, куда входят состояния и характеристики физических объектов, таких как устройства, оборудование и материалы, используемые для изготовления продукции.



В имени **st**ructure variable (переменной-структуры) содержится префикс «st», указывающий, что речь идет о структуре. Отдельные переменные, определяемые в рамках структуры, называются ее элементами. Тип данных каждого отдельного элемента может отличаться от других.

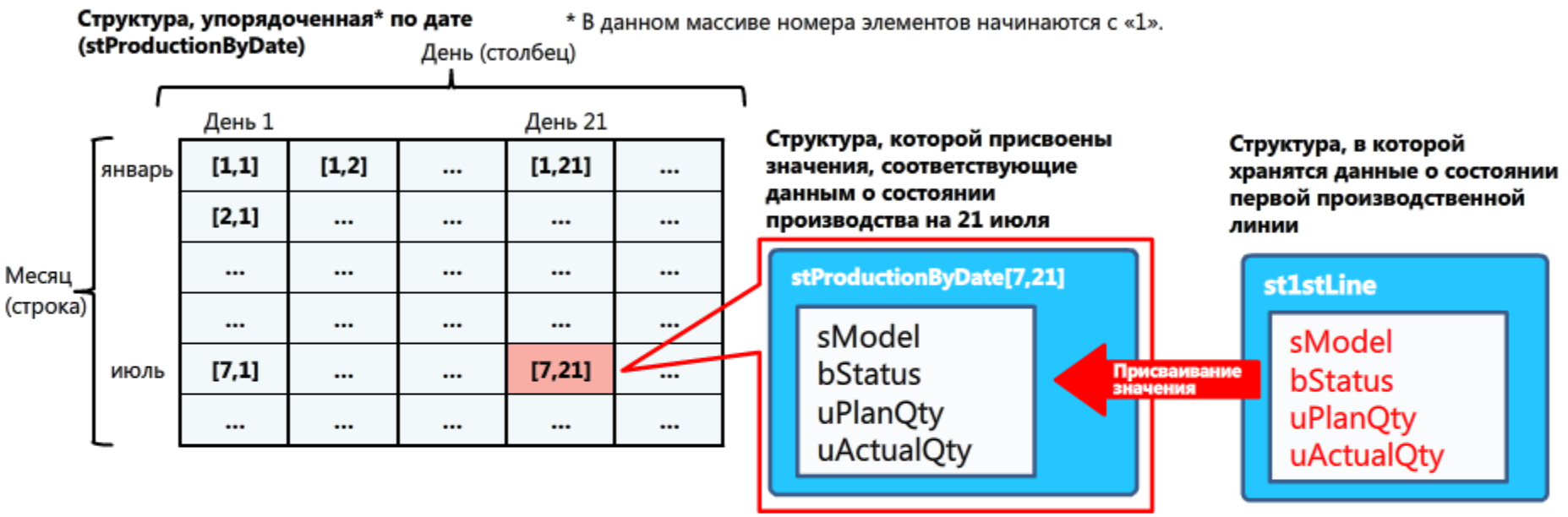
Каждый элемент массива, входящего в состав структуры, можно указать после номера элемента в массиве, используя для этого точку перед именем элемента.

В приведенном ниже примере программы элементу структуры, являющемуся переменной, относящейся к первой производственной линии, присваивается значение константы.

```
st1stLine.uPlanQty := 150;
 (*Для первой производственной линии устанавливается целевое
значение объема производства на сегодняшний день равным 150. *)
```

6.3.1 Хранение массивов, входящих в состав структуры

Структуры могут создаваться в виде массивов.
В приведенном ниже примере состояние производства хранится по дате.

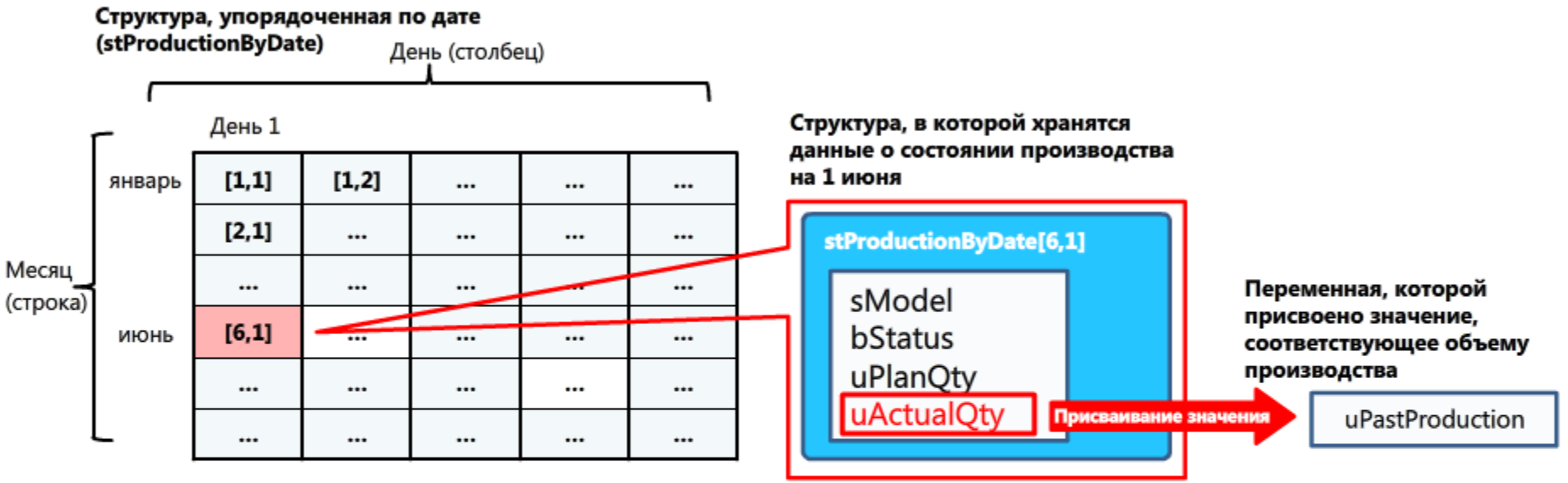



```
stProductionByDate[7,21] := st1stLine;
 (*Состояние производства на 21 июля хранится в структуре,
упорядоченной по дате (stProductionByDate). *)
```

Как в данном случае, элементы упорядоченной структуры нет необходимости определять по отдельности.

6.3.2 Считывание массивов, входящих в состав структуры

В приведенном ниже примере объем производства считывается из структуры, упорядоченной по дате, а затем выполняется присваивание значений переменной.




```
uPastProduction := stProductionByDate[6,1].uActualQty;
(*Переменной uPastProduction присваивается значение, соответствующее объему производства на 1 июня. *)
```

Каждый элемент массивов структуры можно задать путем добавления к номеру элемента массива точки (.) и имени элемента.

6.4 Сводная информация

Данная глава имеет следующее содержание:

- Обзор массивов и их использование
- Циклическая обработка с использованием инструкций FOR
- Обзор структур и их использование

Важные моменты для рассмотрения:

Массив	• Благодаря использованию массивов, имеется возможность обрабатывать несколько значений с помощью одной переменной. • Отдельные переменные в рамках массивов определяются номерами элементов, добавляемых после имени переменной.
Инструкция FOR	• Инструкции цикла используются в программах в тех случаях, когда следует выполнять повторяющиеся операции. • Инструкции FOR используются для организации повторного выполнения операции до тех пор, пока не будут удовлетворены условия для окончания операции цикла. Инструкции, расположенные до «END_FOR;», выполняются с повтором.
Структура	• Использование структур позволяет одним именем переменной представить несколько связанных между собой других переменных. В состав структур могут входить переменные, представляющие данные различных типов. • Имена отдельных переменных или элементов, определяемые в составе структур, отображаются путем добавления точки и имени элемента после имени переменной в составе структуры.

Глава 7 Обработка данных, представленных в виде строк

В некоторых случаях программируемые контроллеры для выдачи команд на подключенные устройства или для получения от них обратной связи используют данные строкового типа, как, например, со сканерами штрихкодов, регуляторами температуры или электронными весами. Для таких целей необходимо объединить или выделить данные строкового типа в соответствии с предъявляемыми требованиями.

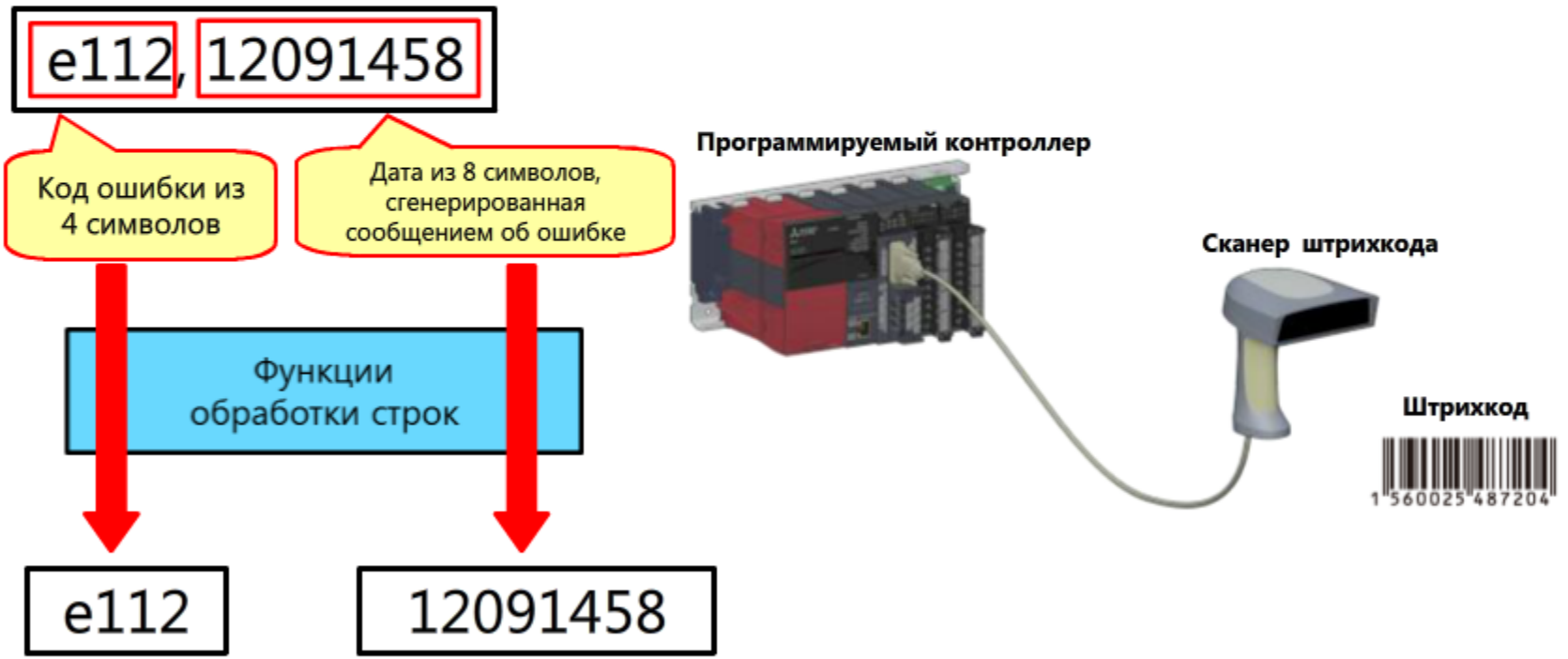
В этой главе описывается порядок обработки данных, представленных в виде строк.

7.1 Пример обработки данных, представленных в виде строк

В качестве примера обработки строки выбран сценарий, иллюстрирующий обработку данных, считанных со сканера штрихкодов.
Для обработки строк используются функции (отдельный тип команд).

Как показано на приведенной ниже иллюстрации, строки, считываемые сканером штрихкода, содержат код ошибки, имеющий фиксированную длину и состоящий из 4 символов, а также данные фиксированного размера, состоящие из 8 символов, представляющих собой месяц, дату и время, составленное из часов и минут. Пример программы обработки строк будет описан с использованием такой системы.

Пример данных, содержащихся в строке, считанной сканером штрихкода



Код ошибки выделен.
7.3 Выделение строк (LEFT)

Выделенное значение даты и времени возникновения ошибки (14:58, 9 декабря).
7.4 Выделение строк (MID)

7.2 Присваивание значения строке

Прежде чем перейти к описанию порядка выделения строк, в этом разделе будут описаны типы данных, используемые в строках.

Типы данных строкового типа, которые могут применяться в программируемых контроллерах, перечислены в таблице, приведенной ниже.

Тип данных	Тип символов, подлежащих обработке	Префиксы, определяемые в соответствии с венгерской нотацией	Расшифровка префикса
Строка	Строки, состоящие из алфавитно-цифровых символов и цифр (ASCII) или из японских иероглифов (кодировка Shift-JIS)	s	**s**tring (строка)
Строка [Юникод]	Строки различных языков и символы	ws	**w**ide **s**tring (расширенная строка)

Тип строки, использование которого зависит от устройства, подключаемого к программируемому контроллеру, или от соответствующего языка.
В данной главе описываются различные типы текстовых строк.

Если значение строкового типа присваивается строковой переменной, такая строка должна заключаться в одинарные кавычки (').

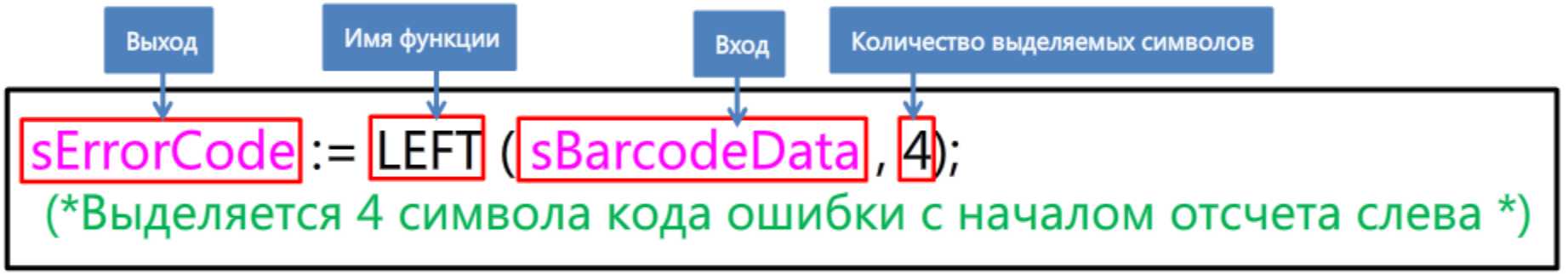
```
sDefault := 'e112,12091458'; (*Присваивание строкового значения*)
```

7.3 Выделение строк (LEFT)

Из строковой переменной «sBarcodeData», в которой содержится строка «e112,12091458», выделяется код ошибки «e112».

Имя переменной	Хранимая в памяти строка
sBarcodeData	e112, 12091458

Функция LEFT выделяет только заданное количество символов, начиная с левого края строки входа.
Ниже приводится иллюстрация программы, представленной в качестве примера.



Выделяется четыре символа с началом отсчета слева. Левой части присваивается значение «e112», которое представляет собой строку кода ошибки.

7.4 Выделение строк (MID)

Из строковой переменной «sBarcodeData», в которой содержится строка «e112,12091458», выделяется время возникновения ошибки «12091458».

Имя переменной	Хранимая в памяти строка
sBarcodeData	e112,12091458

Функция MID выделяет только заданное количество символов, начиная с указанной начальной позиции в строке входа. Ниже приводится иллюстрация программы, представленной в качестве примера.

В данном примере представлено выделение строки из 8 символов, начиная с шестого символа слева. Левой части присваивается значение «12091458», которое представляет собой строку времени возникновения ошибки.

7.5 Сводная информация

Данная глава имеет следующее содержание:

- Методы присваивания строковых значений переменным строкового типа
- Функции, выполняющие выделение строки (LEFT и MID)

Важные моменты для рассмотрения:

Присваивание значения строке	• Чтобы присвоить значение строкового типа строковой переменной, такая строка должна заключаться в одинарные кавычки ('). • Используйте либо строковый тип, либо строковый тип [Юникод] в зависимости от устройства, подключаемого к программируемому контроллеру, или соответствующего языка.
Функции для обработки строк	• Для обработки строк используются функции.

7.6 Краткие итоги курса

В этом курсе представлены основные методы, применяемые для создания программ на языке ST.
На этом данный курс электронного обучения заканчивается.

Программы на языке ST создаются с помощью инженерного программного обеспечения MELSOFT.
Подробные сведения о конкретных действиях, таких как ввод данных, редактирование, сохранение и компиляция программ с помощью инженерного программного обеспечения MELSOFT, см. в указанном ниже документе.

- Курс электронного обучения Mitsubishi FA «MELSOFT GX Works3 (Structured Text)»
  (MELSOFT GX Works3 (язык структурированного текста)) (будет выпущен в ближайшее время)
- Руководство по эксплуатации инженерного программного обеспечения MELSOFT

Для получения более подробной информации о языке ST см. указанный ниже документ.

- Пособие по программированию ПЛК

Для получения информации о командах и функциях, полезных для интересующей вас сферы применения, см. указанный ниже документ.

- Руководство по программированию ПЛК

Тест Заключительный тест

Теперь вы завершили все уроки курса Основы программирования (язык структурированного текста) и готовы к прохождению заключительного теста. Если вам неясны какие-либо из рассмотренных тем, воспользуйтесь возможностью еще раз просмотреть информацию по этим темам прямо сейчас.
Данный заключительный тест содержит всего 12 вопросов (20 пункта).
Вы можете проходить заключительный тест любое количество раз.

Порядок подсчета баллов за тест
После выбора ответа обязательно щелкните кнопку **Ответить**. Если вы продолжите, не нажав кнопку Ответить, ваш ответ будет потерян. (Будет считаться, что вы не ответили на вопрос.)

Результаты теста
Количество правильных ответов, количество вопросов, процент правильных ответов и результат (успешно ли пройден тест) будут отображаться на странице результатов.

Правильные ответы: 4

Всего вопросов: 4

Процент: 100%

Для успешного прохождения теста вы должны правильно ответить на **60%** вопросов.

Продолжить Просмотреть

- Щелкните кнопку **Продолжить**, чтобы завершить тест.
- Щелкните кнопку **Просмотреть**, чтобы просмотреть и проанализировать тест. (Правильные ответы будут отмечены)
- Щелкните кнопку **Повторить попытку**, чтобы пройти тест еще раз.

Тест Заключительный тест 1

Характеристики языка структурированного текста (ST)
Выберите неправильное описание языка ST.

- Для тех пользователей, у кого есть опыт написания программ на языках C или BASIC, изучить язык ST будет просто.
- Такие расчеты, как сложение и вычитание, можно записать в виде типичных математических выражений.
- Для создания программы, напоминающей по виду электрическую цепь, используются значки контактов и катушек.
- Язык ST подходит для обработки данных.

Ответить Назад

Тест Заключительный тест 2

Основополагающие принципы языка ST

Выберите правильную инструкцию, записанную на языке ST.

- uProduction = 15
- uProduction := 15:
- uProduction := 15;
- uProduction = 15;

Ответить Назад

Тест — Заключительный тест 3

Описание комментариев
Выберите правильный комментарий, записанный на языке ST.

- ' Присваивание переменной значения 1.
- (*Присваивание переменной значения 1. *)
- { Присваивание переменной значения 1. }
- <!-- Присваивание переменной значения 1. -->

Ответить | Назад

Тест Заключительный тест 4

Последовательность выполнения программы на языке ST
*Начальное значение «uTotalProduction» равно «100». Значение переменной «uTotalProduction» станет равным «101» после выполнения обработки с помощью приведенной ниже программы, используемой в качестве примера. Выберите правильное состояние «uTotalProduction» по прошествии нескольких секунд.

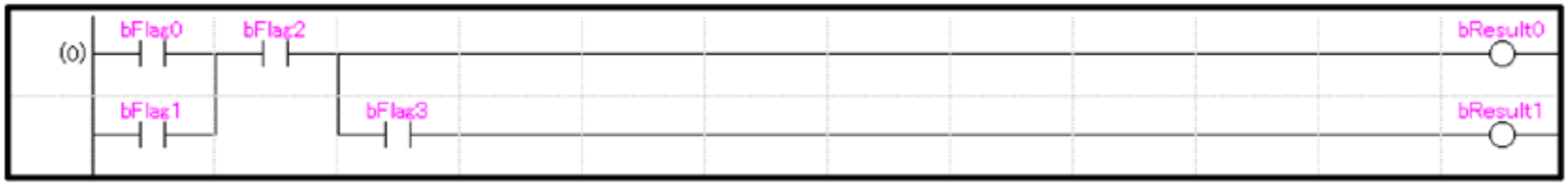
```
uTotalProduction := uTotalProduction + 1;
```

- ○ Сохраняется значение 101.
- ○ Значение продолжает изменяться.

Ответить Назад

Тест Заключительный тест 5

Сочетание нескольких условий
Выберите правильный пример программы на языке ST, в котором представлена та же операция, что и в приведенном ниже примере программы на языке LD.



- bResult0 := (bResult0 OR bFlag1) AND bFlag2;
  bResult1 := bResult0 AND bFlag3;

- bResult0 := (bFlag0 OR bFlag2) AND bFlag1;
  bResult1 := bResult0 AND bFlag3;

Ответить Назад

Тест Заключительный тест 6

Описание инструкций IF в языке ST

Приведенная ниже операция выполняется с использованием примера программы, представленного ниже.

- Если значение температуры падает ниже 5 градусов, нагреватель включается, а охладитель отключается.
- Если значение температуры превышает 50 градусов, нагреватель отключается, а охладитель включается.
- Если температура не соответствует условиям ни одной из приведенных выше инструкций, и нагреватель, и охладитель отключаются.

*Имена переменных: Температура (wTemperature), нагреватель (bHeater) и охладитель (bCooler)

Выберите правильный вариант для каждого пустого поля в примере программы.

```
IF wTemperature Q1 5 Q2
    bHeater := 1;
    bCooler := 0;
Q3 50 Q4 wTemperature Q2
    bHeater := 0;
    bCooler := 1;
Q5
    bHeater := 0;
    bCooler := 0;
END_IF;
```

Q1 --Select--

Q2 --Select--

Q3 --Select--

Q4 --Select--

Q5 --Select--

Ответить Назад

Тест Заключительный тест 7

Инструкции CASE

Выберите правильный вариант для каждого из приведенных ниже описаний (Q1—Q5) для инструкций CASE.

Инструкции CASE используются для ветвления в зависимости от значения (Q1).
В приведенном ниже примере программы, когда значение (Q2) равно 25, переменной (Q3) присваивается значение (Q4).
Если значение переменной (Q2) не равно 10, 25 или 8, переменной (Q3) присваивается значение (Q5).

```
CASE wCode OF
    10:     uLane := 1;
    25:     uLane := 2;
    8:      uLane := 3;
   ELSE     uLane := 4;
END_CASE;
```

Q1 --Select--

Q2 --Select--

Q3 --Select--

Q4 --Select--

Q5 --Select--

Ответить Назад

Тест Заключительный тест 8

Массивы и повторяющееся выполнение инструкций в языке ST
В представленном ниже примере программы вычисляется итоговое значение запланированного объема производства всех моделей, предназначенных для страны Y, после чего это значение присваивается переменной. Выберите элемент массива, который считывается после 3-го цикла выполнения инструкции FOR.

```
uProductionToday := 0;
FOR wCarModel := 0 TO 3 BY 1 DO
  uProductionToday := uProductionToday + uProduction[1,wCarModel];
END_FOR;
```

Массив использовался для хранения прогнозируемого количества изготовленных единиц
в зависимости от модели и страны назначения (uProduction)

- A
- B
- C
- D

Ответить Назад

Тест — Заключительный тест 9

Массивы и повторяющееся выполнение инструкций в языке ST
В приведенном ниже примере программы вычисляется суммарный объем производства в одни и те же дни недели. Используя массив, в котором хранятся объемы производства за отдельные дни, вычисляется итоговое значение за 4 недели. Выберите правильное значение для примера программы.

```
uTotalProduction := 0;
FOR wOnceAWeek := 1 TO ■ BY 7 DO
  uTotalProduction := uTotalProduction + uProductionByDate[2,wOnceAWeek];
END_FOR;
(*Извлекаются и суммируются значения объемов производства по одним и тем же дням за период 4 недели, начиная с 1 февраля. *)
```

Массив, в котором хранятся значения объемов производства за день (uProductionByDate)

- 22
- 21
- 4
- 28

Ответить | Назад

Тест Заключительный тест 10

Характеристики структур языка ST
Выберите неправильное описание структур.

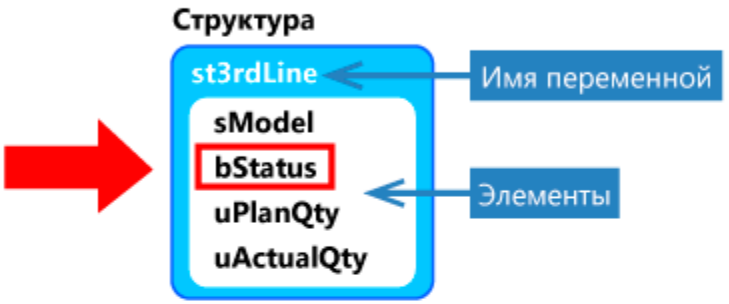
- Структуры используются для упорядочивания и хранения данных в операндах в соответствии с условиями, такими как состояния и характеристики.
- Благодаря использованию структур, программы, обрабатывающие большие объемы данных, могут быть составлены в компактной форме.
- Элементы, определяемые в рамках структуры, должны принадлежать к одному и тому же типу данных.
- Присваивание значений элементам одной и той же структуры может осуществляться без их индивидуального указания.

Ответить Назад

Тест Заключительный тест 11

Определение элементов структуры на языке ST
В представленной ниже структуре упорядочиваются переменные, относящиеся к производственной линии автомобилестроительного предприятия.
Выберите правильное описание для определения элемента «bStatus» в данной структуре.

Параметр	Имя переменной
Модель	sModel
Состояние	bStatus
Целевой объем производства за текущий день	uPlanQty
Текущий объем производства	uActualQty



- st3rdLine.bStatus
- st3rdLine->bStatus
- st3rdLine[bStatus]
- st3rdLine[1]

Ответить Назад

Тест Заключительный тест 12

Обработка строк в языке программирования ST
Приведенный ниже пример программы выполняет выделение заданной строки из исходной строки «e3211151602», которая хранится в переменной «sBarcodeData». Функция MID выделяет только заданное количество символов, начиная с указанной начальной позиции.
Выберите правильно выделенную строку.

- 1151
- 1602
- e321
- 1115

Ответить Назад

Тест **Результат теста**

Вы завершили заключительный тест. Ваша область результатов является следующей.

Правильные ответы: 12

Всего вопросов: 12

Процент: 100%

Продолжить Просмотреть

Поздравляем! Вы прошли тест.

Вы завершили курс Основы программирования (язык структурированного текста).

Благодарим за прохождение этого курса.

Надеемся, что вам понравились уроки, а информация, полученная в рамках этого курса, окажется полезной в будущем.

Вы можете проходить данный курс любое количество раз.

Просмотреть Закрыть