Программирование на языке C/AL

NAVISION®
Academy

Программирование на языке C/AL

Настоящий материал предназначен только для ознакомительных целей. Navision a/s не берет на себя никаких обязательств в случае использования его для иных целей. Navision a/s не несет ответственности за любые факты специального, прямого, косвенного или последующего ущерба, вне зависимости от того, порожден ли он выполнением контрактных обязательств, является следствием халатности, либо вытекает из использования или представления настоящего материала. Настоящий материал может подвергаться изменениям без какого-либо уведомления.

В соответствии с законодательством Дании об авторском праве, воспроизведение любой части данного материала в любой форме и любыми средствами допустимо только с разрешения Navision a/s.

Представленное в материале программное обеспечение поставляется на условиях лицензирования и должно использоваться и копироваться в строгом соответствии с положениями лицензионных соглашений.

COPYRIGHT

Copyright © 2002 Navision a/s, Frydenlunds Allé 6, 2950 Vedbaek, Denmark. All rights reserved.

ТОВАРНЫЕ ЗНАКИ

Товарные знаки, используемые в данном материале, и обозначаемые как ™ или ® являются соответственно товарными знаками или зарегистрированными товарными знаками Navision a/s или Navision Development a/s. Товарные знаки Microsoft, Windows, Windows NT, SQL Server и BackOffice являются зарегистрированными товарными знаками и товарными знаками корпорации Microsoft, действующими на территории США и других стран мира.

Товарные знаки Navision a/s и Navision Development представлены на сайте http://trademarks.navision.com

Использована гарнитура Arial.

Опубликовано Microsoft Business Solutions CIS.

DocID: AT-360-SST-003-v.01.00-RURU

Оглавление

Глава 1.

Введение в программирование на языке C/AL **1-1**

1.1 Модуль программирования на языке C/AL 1-2

1.2 Определение языка C/AL 1-3

1.3 Доступ к C/AL 1-5

1.4 Контрольные вопросы 1-9

Глава 2.

Простые типы данных **2-1**

2.1 Простые типы данных 2-2

2.2 Числовые типы данных 2-3

2.3 Строковые типы данных 2-5

2.4 Булев тип, дата и время 2-7

2.5 Контрольные вопросы 2-9

Глава 3.

Идентификаторы и переменные **3-1**

3.1 Идентификаторы и переменные 3-2

3.2 Синтаксис идентификаторов 3-4

3.3 Инициализация и область видимости переменных 3-5

3.4 Упражнение: создание нового программного модуля 3-6

3.5 Отображение переменных на экране 3-8

3.6 Контрольные вопросы 3-16

Глава 4.

Оператор присваивания **4-1**

4.1 Операторы присваивания 4-2
4.2 Присваивание значения переменной 4-3
4.3 Автоматическое преобразование типов 4-4
4.4 Разделитель операторов 4-6
4.5 Использование операторов присваивания и меню Symbol 4-7
4.6 Контрольные вопросы 4-13

Глава 5.

Выражения **5-1**

5.1 Выражения, операнды и операции 5-2
5.2 Синтаксис выражений 5-4
5.3 Операция работы со строками 5-5
5.4 Вызовы функций в выражениях 5-7
5.5 Контрольные вопросы 5-10

Глава 6.

Числовые выражения **6-1**

6.1 Числовые выражения и приоритет операций 6-2
6.2 Виды арифметических операций 6-3
6.3 Особенности приоритета операций 6-9
6.4 Контрольные вопросы 6-12

Глава 7.

Логические и реляционные выражения 7-1

7.1 Логические и реляционные операции и выражения 7-2
7.2 Реляционные выражения для сравнения 7-4
7.3 Реляционные выражения для работы со множествами 7-6
7.4 Использование логических выражений 7-7
7.5 Добавление в форму логических и реляционных выражений 7-10
7.6 Контрольные вопросы 7-16

Глава 8.

Операторы IF И EXIT 8-1

8.1 Условные операторы и булевы выражения 8-2
8.2 Оператор IF 8-3
8.3 Оператор EXIT 8-6
8.4 Ведение вычислений с помощью операторов IF и EXIT 8-7
8.5 Контрольные вопросы 8-14

Глава 9.

Комментарии и составные операторы 9-1

9.1 Комментарии и составные операторы 9-2
9.2 Синтаксис составных операторов с использованием BEGIN и END 9-3
9.3 Составные операторы, использующие вложенные операторы IF 9-5
9.4 Добавление комментариев 9-8
9.5 Кодирование с помощью составных операторов и комментариев 9-10
9.6 Контрольные вопросы 9-11

Глава 10.

Массивы **10-1**

10.1 Определение массива 10-2
10.2 Синтаксис 10-4
10.3 Потенциал массивов 10-6
10.4 Строки как массивы символов 10-8
10.5 Использование массивов 10-9
10.6 Контрольные вопросы 10-14

Глава 11.

Операторы цикла **11-1**

11.1 Операторы цикла 11-2
11.2 Оператор FOR 11-3
11.3 Оператор WHILE…DO 11-6
11.4 Оператор REPEAT…UNTIL 11-7
11.5 Кодирование с использованием операторов цикла 11-9
11.6 Контрольные вопросы 11-14

Глава 12.

Другие операторы **12-1**

12.1 Оператор WITH 12-2
12.2 Оператор CASE 12-4

Глава 13.

Вызов встроенных функций **13-1**

13.1 Функции и параметры 13-2
13.2 Использование встроенных функций 13-4
13.3 Контрольные вопросы 13-8

Глава 14.

Создание ваших собственных функций 14-1

14.1 Формальные и фактические параметры 14-2
14.2 Локальные функции и переменные 14-4
14.3 Зачем надо создавать функции 14-5
14.4 Оператор EXIT 14-7
14.5 Создание функций 14-8
14.6 Контрольные вопросы 14-13

Глава 15.

Функции языка C/AL 15-1

15.1 Функции взаимодействия с пользователем 15-2
15.2 Функции работы со строками 15-5
15.3 Системные функции 15-8
15.4 Функции работы с датой 15-9
15.5 Функции работы с числами 15-12
15.6 Функции работы с массивами 15-14
15.7 Другие важные функции 15-15

Приложение A.

Контрольный тест A-I

Приложение B.

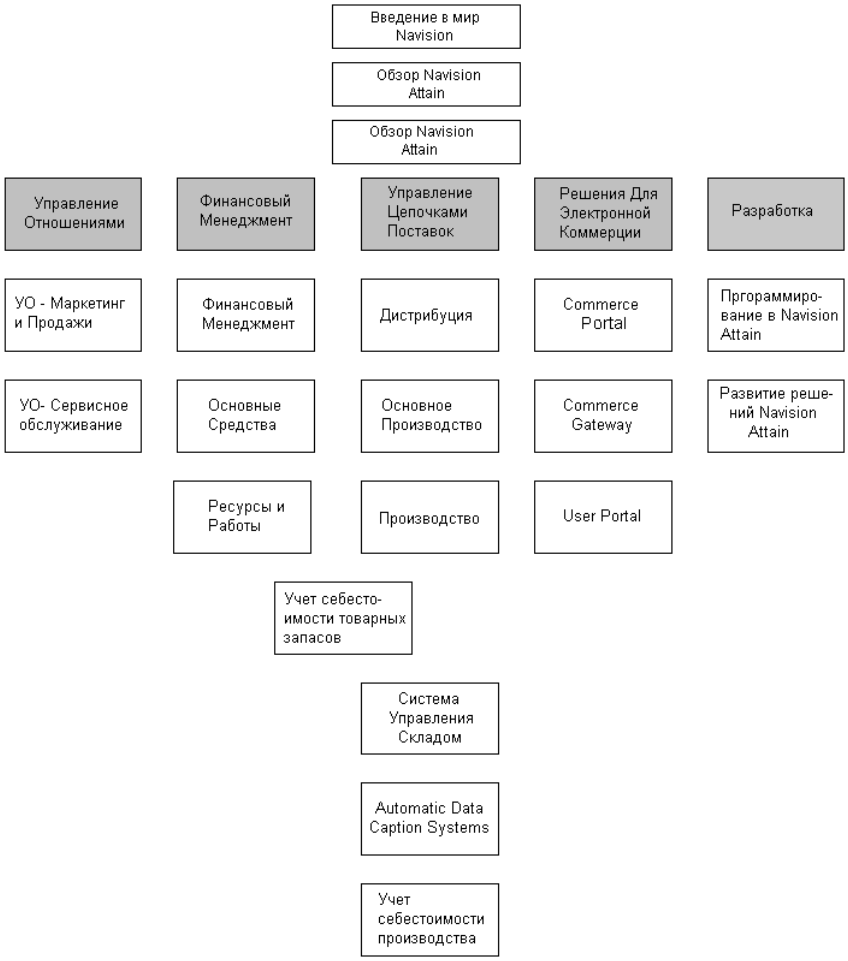
Ответы на контрольный тест B-I

Предисловие

Настоящий курс является одним из четырех учебных курсов в рамках программы обучения “Программирование в Navision” (Navision Programming), которая, в свою очередь, входит в состав учебных материалов “Разработка решений Navision” (Navision Solution Development). Данный курс ориентирован на применение в качестве материала для самостоятельного обучения и может использоваться как отдельно, так и совместно с любым из трех других курсов в рамках данной учебной программы.

Когда вы завершите все четыре курса по программе “Программирование в Navision”, вы сможете пройти тестирование на соответствие требованиям сертификации в качестве консультанта в области программирования для Navision (Navision Attain Consultant in Programming).

На следующей диаграмме представлены все курсы в рамках программы обучения Navision (серым цветом выделены предметные области).



Глава 1

Введение в программирование на языке C/AL

Эта глава является введением в язык клиентских приложений (Client Application Language) для Navision.

Глава содержит следующие разделы:

- Модуль программирования на языке C/AL
- Определение языка C/AL
- Работа с языком C/AL
- Контрольные вопросы

1.1 МОДУЛЬ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C/AL

В рамках данного модуля вы познакомитесь с языком программирования для Navision, а также с нашей инструментальной средой. Вы узнаете, как создавать переменные с использованием Паскаль-ориентированного языка программирования и как работать со встроенными функциями для Navision, способными исполнять простой код. Глава служит хорошей основой для подготовки к изучению программирования для Navision. В настоящем курсе представлены следующие главы:

- Глава 1. Введение
- Глава 2. Простые типы данных
- Глава 3. Идентификаторы и переменные
- Глава 4. Оператор присваивания
- Глава 5. Выражения
- Глава 6. Числовые выражения
- Глава 7. Логические выражения и выражения отношений
- Глава 8. Операторы IF и Exit
- Глава 9. Составные операторы и комментарии
- Глава 10. Массивы
- Глава 11. Операторы цикла
- Глава 12. Другие операторы
- Глава 13. Вызов встроенных функций
- Глава 14. Создание собственных функций
- Глава 15. Функции языка C/AL

Данный курс является составной частью программы самообучения. Полностью прочитайте материал и обстоятельно выполняйте каждое упражнение. Если вам требуется возвращаться назад и повторно перечитывать главу, не жалейте на это время. Полученные знания будут подвергаться проверке.

1.2 Определение языка C/AL

Язык C/AL (Client Application Language, язык клиентских приложений) – это язык программирования, используемый в интегрированной инструментальной клиент-серверной среде (Client / Server Integrated Development Environment, C/SIDE), включенной в Navision.

Для чего он используется?

Существует много целей, для которых может использоваться язык программирования. В то же время, многие из них достигаются с помощью стандартных C/SIDE-объектов. Например:

- представление данных обеспечивается объектами форм (form objects) и объектами отчетов (report objects);
- сбор данных преимущественно обеспечивается объектами форм и объектами портов данных (dataport objects);
- хранение и организация данных обеспечивается объектами таблиц (table objects) совместно со встроенной СУБД (Database Management System, DBMS).

В среде C/SIDE основная цель языка программирования состоит в манипулировании данными. В рамках C/AL вы можете создавать бизнес-правила, чтобы гарантировать целостность хранящихся в таблицах данных и то, что эти данные согласуются с тем, как ваш клиент ведет свой бизнес. Вы можете добавлять новые данные или передавать данные из одной таблицы в другую (например, из журнала в гроссбух). Если есть необходимость соединять в одном отчете или одной экранной форме данные из нескольких таблиц, то, возможно, потребуется прибегать к программированию.

Другой целью языка C/AL служит управление выполнением различных C/SIDE-объектов. С помощью C/AL вы сможете управлять объектами таким образом, чтобы соответствовать требованиям бизнеса ваших клиентов.

Где он используется?

Программирование на C/AL можно найти в любом прикладном объекте системы Navision. В действительности прикладной объект программного модуля используется только для программирования. Если вы находитесь в Дизайнере объектов, щелкните мышью Codeunit, затем выберите программный модуль и щелкните Design. Вы тут же увидите Редактор C/AL и операторы языка программирования

(которые также называются “операторами языка C/AL ”, или просто “кодом”).

Другие объекты тоже могут иметь (хотя и не всегда) операторы C/AL. Такой код можно обнаружить в “триггерах” внутри объекта. Откройте среду Navision и следуйте нашим пояснениям.

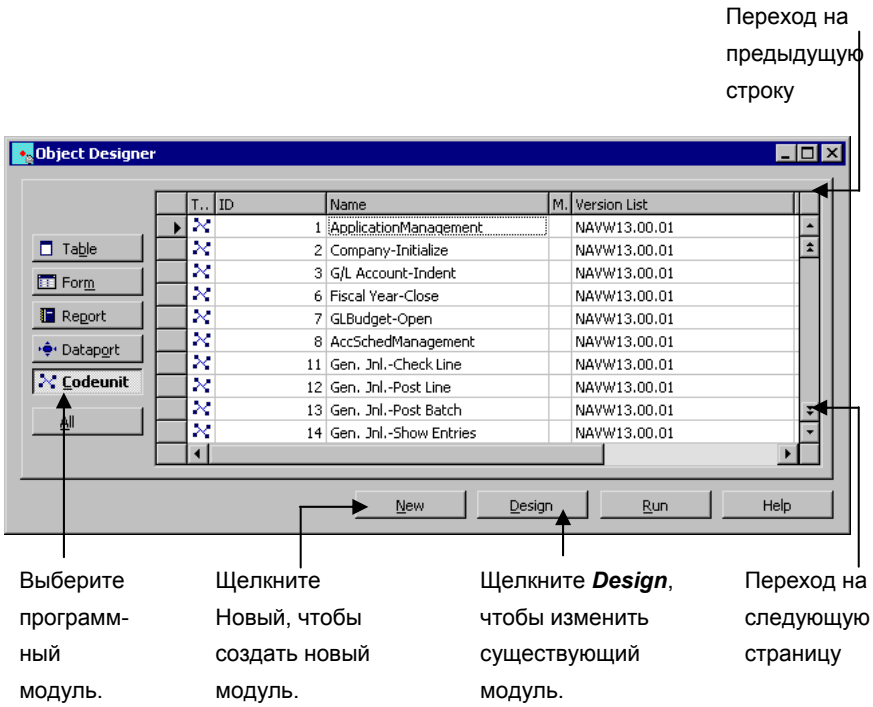
1.3 Доступ к C/AL

Откройте Дизайнер объектов

Чтобы запустить ***Дизайнер объектов***, выберите из планки меню вариант ***Tools***, а затем опцию ***Дизайнер объектов***.

Просмотрите программные объекты

Чтобы получить доступ к программным объектам, щелкните мышью ***Codeunit***, находясь в окне ***Дизайнер объектов.***



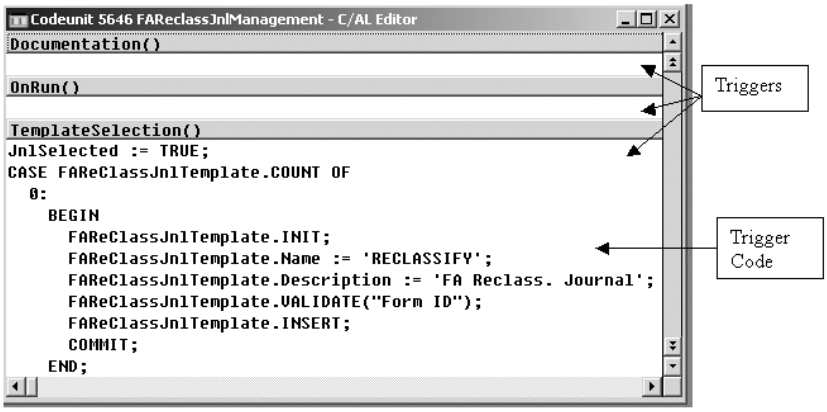
Выберите объект

Используя снимок экрана в качестве руководства, пролистайте список программных модулей и выберите в нем модуль с номером 5646 (FAReclassJnlManagement). Щелкните на нем мышью. Либо, если вы предпочитаете работать с клавиатурой, выберите тот же модуль с помощью клавиш “вверх” и “вниз”.

Посмотрите C/AL-код

Используя снимок экрана в качестве руководства, выберите программный модуль с номером 5646 и щелкните мышью кнопку

Design. Появится окно **Редактора C/AL**:



Текст на языке C/AL, который вы видите на планках серого цвета, называется "триггер". Если нет C/AL-кода между данным и следующим за ним триггером, то такой триггер (в нашем случае OnRun) называется пустым. В частности, между OnRun и UpdUnitCost нет триггерного кода.

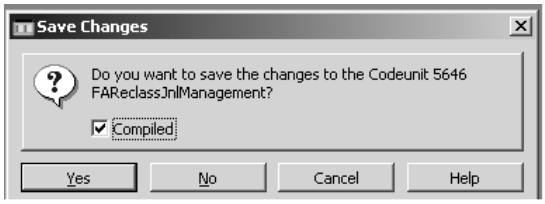
Выделяют три вида триггеров, которые здесь и представлены. Первый вид – документный (Documentation). Он не является настоящим триггером: никакой код в нем не исполняется. Просто вы можете использовать документный триггер, чтобы создавать документацию любого вида для заданного объекта. Многие используют такой триггер, чтобы документировать свои изменения по отношению к стандартным объектам. Каждый объект имеет документный триггер.

Второй вид триггеров – событийный (Event). Имя этого триггера всегда начинается с префикса `On`. C/AL-код в событийном триггере исполняется, когда возникает соответствующее событие. Например, код в событийном триггере OnRun выполняется всякий раз, когда запускается объект программного модуля. В данном случае раз нет никакого триггерного кода, то ничего и не выполняется. Каждый объект обладает собственным набором предопределенных событийных триггеров, которые можно запрограммировать.

Третий вид триггеров из тех, что представлены здесь, называется функциональным (Function). Такие триггеры формируются каждый раз, когда вы создаете в объекте функцию. C/AL-код в функциональном триггере исполняется при каждом вызове функции. В следующем разделе вы узнаете подробнее о создании и вызове функций. Кроме того, в настоящем курсе вы сможете узнать дополнительные сведения и о событийных тригеррах.

Закройте объект

После того как вы посмотрели модуль ItemCostManagement, закройте окно **C/AL Editor.** Это можно сделать, щелкнув **Close Box** или нажав клавишу ESC. Если вы не внесли никаких изменений, объект будет закрыт и вы сможете продолжить работу. Если же были сделаны какие-то правки, появится следующее окно:



Теперь, чтобы отменить внесенные изменения при выходе из редактора **C/AL Editor** и закрытии объекта, щелкните кнопку **No** (либо нажмите комбинацию клавиш ALT+N).

Посмотрите C/AL-код в объекте таблиц

1 Находясь в окне **Object Designer,** щелкните мышью **Table**. Пролистайте данные в окне **Object Designer** и выберите **Table 18** (в таблице **Customer**). Щелкните **Design,** чтобы открыть окно **Table Designer**. Обратите внимание, что это не **Редактор C/AL**. Это список полей, которые образуют описание таблицы. С ними вы познакомитесь позже.

2 Чтобы получить доступ к C/AL-коду, вы должны открыть Редактор C/AL. Этого можно добиться, щелкнув мышью значок **C/AL Editor** на строке инструментов:

C/AL Editor

Другой способ – нажать клавишу F9. Раз вы находитесь в редакторе C/AL Editor, обратите внимание, что в данном Table Object имеется множество тригерров. Большинство из них пустые, но некоторые содержат C/AL-код.

Много кода находится в триггерах, которые называются именем поля ввода с предшествующим этому имени префиксом OnValidate. Это событийные триггеры, и код “коммутируется” завершением

пользователем ввода данных в конкретном поле. Об этом и других тригеррах вы сможете узнать в рамках курса “Программирование для Navision” (Navision Attain Programming course). Теперь просто посмотрите еще раз на экран и закройте окно **Редактора C/AL**. Как только это произойдет, вы сможете закрыть Дизайнер таблиц (Table Designer), опять-таки без сохранения случайно сделанных вами правок.

1.4 Контрольные вопросы

Вопросы

Терминология

1 Как называется язык программирования для интегрированной инструментальной клиент-серверной среды?

2 Приведите примеры использования программного кода:

3 Где могут находиться операторы языка программирования?

4 Что нужно использовать для изменения исходного текста в объекте?

5 Перечислите три основные типа триггеров:

6 Какую клавишу требуется нажимать для изменения программного кода в объекте (отличном от объекта программного модуля)?

Практикум

Используя для просмотра Дизайнер объектов, изучите код в объекте форм под номером 39.

Используя для просмотра Дизайнер отчетов, изучите код в объекте отчетов под номером 324. Обратите внимание, что код будет различаться в зависимости от того, какая строка (также называемая "***DataItem***") выбрана в Дизайнере отчетов. Посмотрите, что произойдет, когда вы выберете строку, которая идет за последней строкой в Дизайнере отчетов, перед тем как откроется окно кода.

Ответы

Терминология

1 Как называется язык программирования для интегрированной инструментальной клиент-серверной среды?

C/AL (Client Application Language, язык клиентских приложений*)*

2 Приведите примеры использования программного кода.

Манипулирование данными
Представление данных
Сбор данных
Для управления работой объектов языка C/AL
Для реализации бизнес-правил и т.п.

3 Где могут находиться операторы языка программирования?

В прикладных объектах (формах, таблицах, отчетах, программных модулях и портах данных). Кроме того, в триггерах внутри этих прикладных объектов.

4 Что нужно использовать для изменения исходного текста в объекте?

Редактор C/AL

5 Перечислите три основные типа триггеров.

Триггеры документации (Documentation Triggers), триггеры событий (Event Triggers), триггеры функций (Function Triggers).

6 Какую клавишу требуется нажимать для изменения программного кода в объекте (отличном от объекта программного модуля)?

F9

Практикум

Объекты форм – обратите внимание, что форма сама имеет триггеры, и что отдельные органы управления в форме также обладают триггерами.

Объекты отчетов – каждый элемент данных имеет собственные триггеры. Триггеры отчетов можно обнаружить, если выбрать пустую строку, идущую за последним элементом данных в Дизайнере отчетов (Report Designer).

Глава 2

Простые типы данных

Данная глава знакомит с концепцией типов данных, а также поясняет, каковы различные типы данных, когда их использовать и как с ними работать.

В главе представлены следующие разделы:

- Простые типы данных
- Числовые типы данных
- Строковые типы данных
- Булев тип, дата и время
- Контрольные вопросы

2.1 Простые типы данных

Определения

Данные

Данные представляют собой известные факты, известные фрагменты информации. В рамках курса мы всегда будем использовать термин “данные” для обозначения информации, которая доступна нам для обработки в Navision с использованием C/SIDE.

Типы данных

Типы данных представляют различные виды информации, которые могут появляться в среде C/SIDE. Разные типы данных имеют разные значения, разный смысл для этих значений и разные способы их обработки. Например, если у нас имеются два различных значения – “25” и “37” – и мы их складываем, то получится разный результат в зависимости от того, какой тип данных они представляют. Если это числа, то результат будет “62”. Если же это текстовые строки, то результатом будет “2537”.

Константы

Константы – это значения, которые записываются непосредственно в операторах языка программирования. Они называются константами, поскольку их значения никогда не изменяются при работе пользователя с приложением. Константы могут изменяться только путем изменения C/AL-кода.

Простые типы данных

Простые типы данных – это такие типы, которые имеют только одно значение и которые не могут быть разбиты на другие значения иных типов.

Байт

Байт – это единица хранения данных, используемая в компьютерах. Один символ, хранящийся в компьютере, занимает один байт памяти. Производными терминами являются килобайт (Кбайт), равный 1024 байтов, мегабайт (Мбайт), равный 1024 Кбайт или 1048576 байтов, а также гигабайт (Гбайт), равный 1024 Мбайт или 1073741824 байтов.

2.2 Числовые типы данных

Числовые типы данных представляют собой все виды чисел или подсчетов. Существует множество автоматических методов, используемых средой C/SIDE для скрытого преобразования одного числового типа в другой, и во многих случаях они могут применяться взаимообразно. Однако, в некоторых ситуациях, различия могут быть весьма важными. Подчас это приводит к ошибкам и иногда к более коварным ситуациям.

Целый (Integer)

Целый тип (Integer) определяет целые числа, лежащие в диапазоне от -2 147 483 647 до +2 147 483 647. Он занимает 4 байта. По умолчанию целочисленная переменная равна нулю. Характерными константами целого типа в C/AL являются:

- 12
- 1000 (обратите внимание, что точки/запятые в числовых константах недопустимы)
- -100
- 0

Десятичный (Decimal)

Десятичный тип (Decimal) определяет целые и дробные числа в диапазоне от $-1x10^{63}$ (1 с 63 нулями) до $+1x10^{63}$, где дробная часть определена до $1x10^{-63}$ (62 нуля, идущие после запятой, предшествуют 1). Он обеспечивает точность до 18 цифр в двоично-десятичном формате (Binary Coded Decimal; BCD) и занимает 12 байтов. По умолчанию переменная десятичного типа равна нулю. Характерными константами десятичного типа в C/AL являются:

- 12.50
- 52000000000 (точки/запятые для разделения тысяч не используются)
- -2.0
- 0.008
- -127.9533

Вариантный (Option)

Вариантный тип (Option) представляет собой особый вид целого типа.

Он позволяет программисту задавать слова для каждого своего значения. Например, если мы создали переменную с именем Spectrum, то можем сделать ее вариантного типа с помощью следующей вариантной строки:

```
Red,Orange,Yellow,Green,Blue,Indigo,Violet
```

По умолчанию значение переменной вариантного типа равно нулю, при этом она является разновидностью целого типа, соответствует первому элементу с именем “Red”. Следовательно, “Green” соответствует целочисленному значению 3. Заметим, что между элементами не может быть никаких пробелов, так как пробел служит частью имени элемента.

Символьный (Char)

Символьный тип (Char) представляет одиночный символ. Он занимает 1 байт памяти. С точки зрения синтаксиса он рассматривается числовым типом и может принимать значения от 0 до 255. В выражениях он может использоваться наряду с другими числовыми типами. Характерными константами символьного типа в C/AL являются:

- 'b' (обратите внимание на одинарные кавычки вокруг символа)
- 'C'
- '3'
- '?'

2.3 СТРОКОВЫЕ ТИПЫ ДАННЫХ

Строковые данные – это данные, состоящие из строк символов. Данные, размещаемые в текстовых процессорах, -- строковые. В электронных таблицах, где большая часть данных считается числовыми, иногда вводятся и строковые данные с использованием специального отличающего их префикса. В константах языка C/AL для отделения строковых данных используется знак одинарных кавычек, известный также как апостроф ('). Все строковые константы заключаются в одинарные кавычки.

Текстовый (Text)

Текстовый тип – это строка длиной от 0 до 250 символов. Длина значения текстового типа соответствует количеству содержащихся в нем символов. Чтобы подсчитать объем занимаемой памяти, добавьте к длине значение 1 и округлите до ближайшего числа, кратного 4. Таким образом, текстовое значение длиной 8 занимает 12 байтов (1 + 8 с округлением до ближайшего числа, кратного 4). Характерными константами текстового типа в C/AL являются:

- 'Hello'
- '127.50' (хотя это и выглядит подобно числу, но будучи заключенным в кавычки считается текстовым значением)
- '' (это пустое значение, длина его равна 0)
- ' spaces before ... and after '
- 'Here''s how to use an apostrophe' (если внутри текстовой константы нужно использовать апостроф, он удваивается)

Кодовый (Code)

Кодовый тип – это специальный вид текста. Все буквы записываются в верхнем регистре, а все предшествующие и замыкающие пробелы отсекаются. Кроме того, когда значение типа отображается на пользовательском экране, то если все символы являются цифрами, кодовое значение автоматически выравнивается вправо. Чтобы подсчитать объем занимаемой памяти, надо к длине добавить 2 и округлить до ближайшего числа, кратного 4. Таким образом, кодовое значение длиной 11 будет занимать 16 байтов (2 + 11 с округлением до ближайшего числа, кратного 4). Те же самые текстовые константы, которые приводились выше, будучи преобразованы в кодовые значения выглядят так:

- 'HELLO'
- '127.50'
- ''
- 'SPACES BEFORE ... AND AFTER'
- 'HERE''S HOW TO USE AN APOSTROPHE'

Когда производится сравнение двух кодовых значений, либо когда они подвергаются сортировке, то исходят из упомянутого выше выравнивания вправо. Таким образом, для кодовых значений '10' больше '9', но '10A' меньше '9A'. Для текстовых значений '10' меньше '9'.

2.4 Булев тип, дата и время

Булев (Boolean)

Булевы данные, также известные как логические данные, представляют простейшую форму данных. Они занимают в памяти всего 1 байт, хотя когда хранятся в БД, используют 4 байта. Константами булева типа в C/AL являются только

- TRUE
- FALSE

Заметим, что при сравнении этих переменных FALSE меньше TRUE, поскольку FALSE хранится как 0, а TRUE хранится как 1. Однако, целочисленное значение не может быть взаимозаменяемым с константами TRUE и FALSE. В программном коде булева переменная должна иметь значение “TRUE” или “FALSE”, а не 0 или 1.

Дата (Date)

Тип дата (календарная дата) определяет значения, варьирующиеся в диапазоне от 1/1/0000 до 12/31/9999. Он занимает 4 байта памяти. Кроме того, значение этого типа может представлять обычную дату (*Normal Date)* или дату закрытия (*Closing Date)*. Дата закрытия представляет собой последнюю миллисекунду последней минуты последнего часа дня, поэтому она больше обычной даты с тем же календарным значением. Характерными константами типа Date в C/AL являются:

- 123197D (December 31, 1997)
- 030595D
- 08171953D (August 17, 1953)
- 0D (неопределенная дата с наименьшим допустимым значением)
- 063012D (June 30, 2012)

Все константы дат представляют собой обычные даты. Константы для дат закрытия в языке C/AL не существует.

Принятый синтакис: mmddyyD или mmddyyyyD. Для обозначения года могут использоваться две цифры. Они вопринимаются по-разному в зависимости от той версии Navision, которую вы используете. Для

версий 2.6 и выше, если год лежит в диапазоне от 30 до 99, то он считается относящимся к годам XX века (1900-е годы). Если же он находится в диапазоне от 00 to 29, то считается относящимся к XXI веку (2000-е годы). Для версий от 2.01 до 2.5 значение 19 будет обозначать 2019 год, а 20 – 1920 год.

Обратите внимание, что дата выделяется наличием буквы D в конце своего значения. Если буква D там отсутствует, то C/AL полагает, что это целое число. Соответственно, будет генерироваться ошибка о невозможности присваивания переменной типа Data целочисленного значения. Не используйте наклонную черту в качестве разделителя для месяца, дня и года. Она подразумевает операцию деления, и, как следствие, при ее использовании будет генерироваться ошибка о невозможности присваивания целого или десятичного значения переменной типа Data.

Время (Time)

Тип время представляет собой время суток (а не интервал времени), которое может изменяться в диапазоне от 00:00:00 до 23:59:59.999. Значения этого типа занимают 4 байта. Характерными константами типа Time в C/AL являются:

- 103000T (10:30am)
- 154530T (3:45:30pm)
- 0T (неопределенное время, меньше всех остальных значений)
- 030005.100T (3:00:05.1am)
- 225930.135T (10:59:30.135pm)

Обычно используется следующий синтаксис hhmmss[.xxx]T, где доли секунды (.xxx) носят необязательный характер. Аналогично типу дата тип время должен иметь в конце значения букву ‘T’, чтобы можно было отличить его от целого типа.

2.5 Контрольные вопросы

Вопросы

1 Какой тип данных должен использоваться для хранения дня рождения сотрудника?

2 Какой тип данных должен использоваться для хранения имени сотрудника?

3 Какой тип данных должен использоваться для записи еженедельных выплат по зарплате (необходимо записывать с точностью до центов)?

4 Какой тип данных должен использоваться для записи того факта, освобожден ли сотрудник от налоговых выплат?

5 Укажите тип данных следующей константы: 'You must enter a positive value'.

6 Укажите тип данных следующей константы: 123197

7 Укажите тип данных следующей константы: 327.01

8 Укажите тип данных следующей константы: 3,498

Ответы

1 Какой тип данных должен использоваться для хранения дня рождения сотрудника?

Date

2 Какой тип данных должен использоваться для хранения имени сотрудника?

Text

3 Какой тип данных должен использоваться для записи еженедельных выплат по зарплате (необходимо записывать с точностью до центов)?

Decimal

4 Какой тип данных должен использоваться для записи того факта, освобожден ли сотрудник от налоговых выплат?

Boolean

5 Укажите тип данных следующей константы: 'You must enter a positive value.'

Text

6 Укажите тип данных следующей константы: 123197

Integer

7 Укажите тип данных следующей константы: 327.01

Decimal

8 Укажите тип данных следующей константы: 3,498

Это недопустимая константа, поскольку в ней используется запятая, при этом число не заключено в кавычки.

Глава 3

Идентификаторы и переменные

Данная глава посвящена идентификаторам и переменным. Вы познакомитесь с синтаксисом идентификаторов, узнаете, как инициализировать переменные и как определять их область видимости. Она также объясняет, что такое системные переменные.

Глава содержит следующие разделы:

- Идентификаторы и переменные
- Синтаксис идентификаторов
- Инициализация и область видимости переменных
- Упражнение: создание нового программного модуля
- Отображение переменных на экране
- Контрольные вопросы

3.1 Идентификаторы и переменные

Определения терминов

Идентификатор

Идентификатор (identifier) – это имя какого-то элемента, способ его идентифицировать. Объекты, именные константы, переменные, поля и функции – все они имеют идентификаторы. Эти идентификаторы нужно использовать при обращении к указанным сущностям. Не все элементы программы имеют идентификаторы. Идентификаторов нет у обычных констант, операций и определенных зарезервированных слов. К ним обращение идет напрямую. Один из способов понять разницу состоит в том, что те элементы программирования, которые на ссылаются на нечто, хранящееся в памяти, требуют для доступа к ним идентификатор. В то время как другим, которые расположены в самом программном коде и ни на что находящееся за пределами кода не ссылаются, идентификатор не нужен.

Именная константа

Именная константа (named constant) представляет собой ссылку на значение, которое присваивается, когда приложение создается, и не изменяется в ходе его выполнения. Именная константа имеет идентификатор и потому хранится в памяти во время выполнения. Приложение может ссылаться на ее значение, но само значение меняться не может и потому остается константой. В системе Navision единственным примером именной константы является текстовая константа. Текстовая константа (text constant) создается таким же образом, что и переменная, за исключением двух моментов: 1) ее значение определяется в момент создания и 2) ее тип данных не может меняться; это всегда Text.

Переменная

Переменная (variable) представляет собой ссылку на значение. Которое может изменяться в ходе выполнения приложения. Переменная ссылается на фактическую ячейку памяти, где хранится значение. Переменная имеет имя, называемое также идентификатором, которое программист использует в программе вместо фактического адреса памяти. Переменная имеет тип, который описывает вид данных, способных храниться в данной ячейке памяти. Наконец, переменная обладает значением, которое представляет факктические данные, хранящиеся в этой ячейке памяти.

Синтаксис

Синтаксис представляет собой набор грамматических правил, который и определяет язык программирования. Строки программного кода, которые следуют этим правилам, считаются соответствующими корректному синтаксису. Компьютер не понимает программных строк, которые не соответствуют синтаксису; они не могут быть откомпилированы и исполнены. В главе 2 мы уже определяли корректный синтаксис для констант. Другие правила синтаксиса будут раскрыты в этой главе.

3.2 СИНТАКСИС ИДЕНТИФИКАТОРОВ

Существет два способа построения в программе допустимых идентификаторов. Первый – следование синтаксису языка Паскаль . Это означает, что первый символ в идентификаторе должен быть либо знаком подчеркивания (_), либо буквой (в верхнем или нижнем регистре). Вслед за первым символом может идти до 29 других символов, каждый из которых – это либо буква (в верхнем или нижнем регистре), либо знак подчеркивания, либо цифра (от 0 до 9).

Второй способ применяется, если вы не хотите следовать обычному синтаксису Паскаля. В этом случае при использовании в языке C/AL идентификатор должен быть заключен в кавычки ("). Внутри них вы можете использовать любые символы, за исключением управляющих (литеры, чей ASCII-код лежит в диапазоне 0 – 31, а также литера с кодом 255), и сам знак кавычек ("). Вы можете использовать даже пробелы.

Заметим, что если вы следуете второму методу, то количество символов в идентификаторе по-прежнему не может превышать 30, причем в их число не входят знаки кавычек. Среда C/SIDE в отношении идентификаторов не делает различий между буквами верхнего и нижнего регистров. Таким образом, если имеются два идентификатора, первый из которых "Account Number", а второй "Account number", то в среде C/SIDE они считаются идентичными.

Здесь есть тонкий момент. Внутри ондного объекта все идентификаторы должны быть уникальными. Идентификатор не может совпадать с зарезервированными словами (вроде BEGIN и END) и операциями (вроде DIV и MOD). Если это все же происходит, возникает синтаксическая ошибка. Кроме того, два идентификатора не должны иметь одинаковое имя внутри объекта, если не существует некоторого иного способа их отличить друг от друга. Если ссылка “неясная” (т.е. если C/SIDE не может понять, на какой конкретно элемент вы ссылаетесь), то в результате будет возникать ошибка.

3.3 Инициализация и область видимости переменных

Глобальные и локальные переменные

Все переменные имеют определенную область видимости (scope) , т.е. определенный набор мест, где к ним может быть осуществлен доступ. Говорят, что переменная имеет “глобальную” область видимости, если к ней может быть доступ из любого места в объекте. Говорят, что переменная имеет “локальную” область видимости, если она доступна только в единственном триггере в объекте. Переменные не могут быть доступны за пределами того объекта, где они определены.

Текстовые константы

Текстовые константы, подобно переменным, также имеют определенную область видимости (глобальную или локальную). В этом смысле доступ к ним подчиняется тем же правилам, что и для переменных.

Системные переменные

Существуют конкретные переменные, которые автоматически определяются и управляются самой системой. Примером может служить переменная Rec в табличных объектах. Каждый объект обладает собственным набором системных переменных (system-defined variables). Программист может использовать эти переменные везде, где это необходимо, не заботясь об их создании и инициализации.

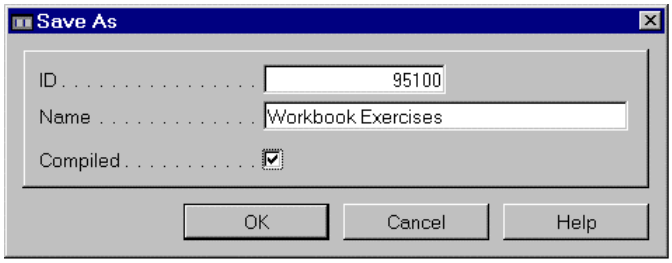
Инициализация переменных

До того как любой C/AL-код будет выполнен, все определенные программистом переменные и текстовые константы инициализируются соответствующими значениями.

- Для всех переменных числового типа начальное значение 0.
- Строковые переменные инициализируются пустой строкой ('').
- В случае текстовых констант начальное значение – то, что определено программистом.
- Для булевых констант начальное значение FALSE.
- Для переменных типов Date и Time начальное значение – 0D (неопределенная дата) и 0T (неопределенное время) соответственно.

3.4 Упражнение: создание нового программного модуля

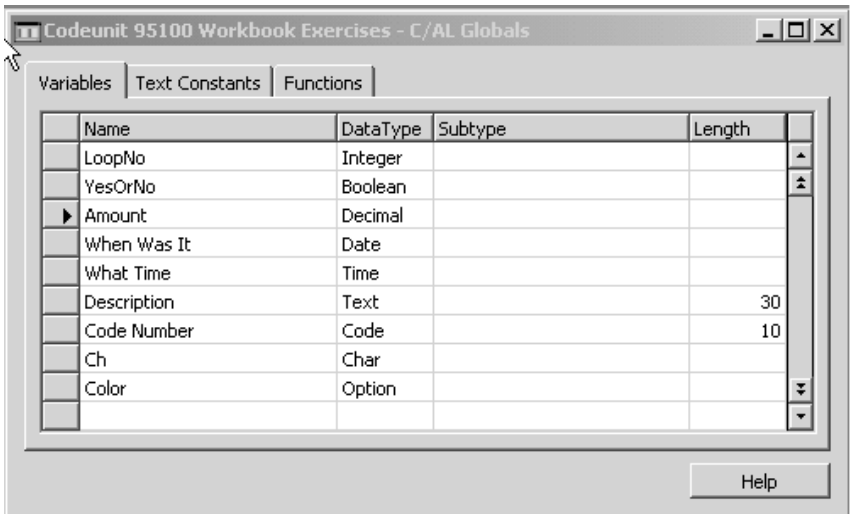
Откройте Object Designer, выбрав из строки меню пункты Tools, Object Designer. Щелкните Codeunit, чтобы увидеть список всех программных модулей. Теперь в нижней части окна **Object Designer** щелкните New. Появится пустое окно редактора **C/AL Editor**. Теперь закройте окно **Редактора C/AL**. Появится окнос вопросом "Do you want to save the changes to the codeunit"? (“Хотите ли вы сохранить изменения программного модуля”). Щелкните кнопку Yes. Теперь появится окно **Save**. Заполните в нем поля следующим образом:



Щелкните OK. Теперь сделайте прокрутку окна Object Designer до его нижней границы и найдите свой программный модуль. Выберите его и нажмите кнопку Design.

Описание переменной

Чтобы описать глобальные переменные в объекте, выберите в строке меню пункты View, C/AL Globals. Появится следующее окно:



Name	DataType	Subtype	Length
LoopNo	Integer		
YesOrNo	Boolean		
Amount	Decimal		
When Was It	Date		
What Time	Time		
Description	Text		30
Code Number	Code		10
Ch	Char		
Color	Option		

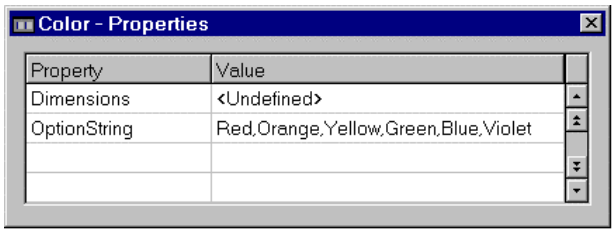
Заполните его так, как показано выше. Если вы нажмете клавишу F6,

находясь на колонке DataType, то вы увидите список типов данных, которые подлежат выбору. Заметим, что эти простые типы данных образуют только часть тех типов, которые могут быть созданы.

Как только вы заполните все поля, выберите переменную Color. О том, что переменная Color выбрана, можно судить по появлению черного треугольника слева от ее имени.

Теперь на строке инструментов щелкните кнопку Properties:

Появится окно **Properties**. Заполните свойство OptionString в точности так, как показано на следующем окне:



Обратите внимание, что слова отделяются запятыми, причем пробелов нет как между словами, так и перед запятыми. Теперь закройте сначала окно **Properties**, а затем и окно **C/AL Globals**. Наконец, закройте окно **Object Designer**. Если появится вопрос о необходимости сохранения, щелкните Yes.

3.5 Отображение переменных на экране

Щелкните Design и вернитесь обратно в программный модуль Workbook Exercises. Щелкните на первой строке Click on the first line относительно триггера OnRun. Теперь наберите следующую строку C/AL-кода:

```
MESSAGE('The value of %1 is %2','LoopNo',LoopNo);
```

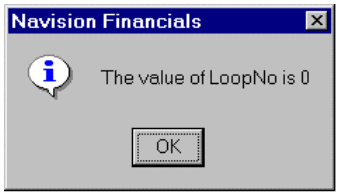
Функция MESSAGE позволяет вам отображать простое окно с находящимся в нем простым сообщением, а также опкой **OK**, которую пользователь должен нажать, прочитав сообщение.

Обратите внимание, что следом за идентификатором функции MESSAGE вы найдете открывающую скобку, текстовую константу, запятую, другую текстовую константу, еще одну запятую, переменную LoopNo, закрывающую скобку и точку с запятой. Когда функция, подобная этой, вызывается, то за ней идут параметры, заключенные в скобки. Здесь используются три параметра, отделенные друг от друга запятыми.

Первый параметр – текст, который содержит то сообщение, что должно быть выведено на экран. Этот текст имеет специальные строки параметризации (substitution strings). Строка параметризации допускается в определенных функциях, включая данную, и состоит из знака процента, за котором идет цифра (от 1 до 9). Когда эта функция исполняется на этапе выполнения, первая строка параметризации (%1) заменяется первым идущим за ней параметром, а вторая (%2) – вторым. Если параметр не является текстовым (например, упомянутый выше параметр LoopNo), он автоматически преобразуется или форматируется в текст до выполнения подстановки.

Закройте и сохраните программный модуль, а затем щелкните Run.

Появится следующее окно:



Обратите внимание, что первая строка параметризации была заменена вторым параметром (первым параметром, идущим за

сообщением), т.е. текстовой константой 'LoopNo'. Вторая строка параметризации была заменена третьим параметром (вторым параметром, идущим за сообщением), т.е. значением переменной LoopNo.

Заметим, что значение целочисленной переменной было автоматически инициализировано в 0 и что это значение было подставлено в сообщение. Щелкните OK, чтобы очистить окно сообщений.

3.6 Контрольные вопросы

Вопросы

1 Модифицируйте строку кода в программном модуле Workbook Exercises, чтобы отобразить на экране имя вашей второй переменной (YesOrNo) и ее значение. Запустите программный модуль и напишите ниже получившееся сообщение.

2 Выполните то же самое в отношении Amount.

3 Сделайте то же самое с When Was It. Не забудьте при использовании этой переменной в качестве третьего параметра заключить ее имя в кавычки.

4 Выполните аналогичные действия в отношении What Time.

5 Выполните аналогичные действия в отношении Description.

6 Выполните аналогичные действия в отношении Code Number.

7 Выполните аналогичные действия в отношении Ch.

8 Выполните аналогичные действия в отношении Color.

Ответы вы можете увидеть на следующей странице.

Ответы

1 Модифицируйте строку кода в программном модуле Workbook Exercises, чтобы отобразить на экране имя вашей второй переменной (YesOrNo) и ее значение. Запустите программный модуль и напишите ниже получившееся сообщение.

The value of YesOrNo is No

Заметим, что при преобразовании в текст значение TRUE превращается в 'Yes', а значение FALSE в 'No'.

2 Выполните то же самое в отношении Amount.

The value of Amount is 0.

3 Сделайте то же самое с When Was It. Не забудьте при использовании этой переменной в качестве третьего параметра заключить ее имя в кавычки.

The value of When Was It is

Заметим, что поскольку дата является неопределенной, то при преобразовании в текст онав заменяется пустой строкой. Строка кода должна быть такой:

```
MESSAGE('The value of %1 is %2','When Was It',"When
Was It");
```

4 Выполните аналогичные действия в отношении What Time.

The value of What Time is

5 Выполните аналогичные действия в отношении Description.

The value of Description is

6 Выполните аналогичные действия в отношении Code Number.

The value of Code Number is

7 Выполните аналогичные действия в отношении Ch.

The value of Ch is

Заметим, что когда Ch преобразуется в текст, ее нулевое

значение преобразуется в пустой текст.

8 Выполните аналогичные действия в отношении Color.

The value of Color is Red

Заметим, что первое слово, написанное в свойстве OptionString, соотвествует нулевому значению для Option.

Глава 4

Оператор присваивания

Данная глава разъясняет, как с использованием корректного синтаксиса присваивать значения переменным. Вы увидите, как преобразовывать данные из одного типа в другой и познакомитесь с синтаксисом разделителей операторов. Будет описано меню Symbol, которое используется для помощи при написании кода.

Глава содержит следующие разделы:

- Операторы присваивания
- Присваивания значения переменной
- Автоматическое преобразование типов
- Разделитель операторов
- Использование операторов присваивания и меню Symbol
- Контрольные вопросы

4.1 Операторы присваивания

Большинство разработчиков знакомы с терминами “присваивание” (assignment) и “оператор” (statement), но для уточнения мы здесь еще раз дадим им определения.

Присваивание

Присваивание (assignment) означает задание переменной некоторого значения. Поскольку переменная имеет значение, она хранит его до тех пор, пока ей не будет задано другое значение, либо пока не завершится текущий набор кода и когда компьютер больше не будет отслеживать эту переменную. Язык C/AL имеет несколько способов присваивания, одним из которых является простой оператор присваивания.

Оператор

Программный оператор (programming statement) – это простая законченная инструкция, записанная в коде. Вы можете рассматривать ее как строку кода, так как обычно мы пишем один оператор на строке. В действительности же, один оператор может занимать несколько строк в Редакторе C/AL Editor, и аналогично несколько операторов могут быть размещены на одной строке .

Оператор присваивания

Оператор присваивания (assignment statement) – это определенный тип присваивания. Это оператор, особым образом присваивающий значение переменной.

4.2 Присваивание значения переменной

Способность присваивать переменным различные значения – краеугольный камень программирования. Разные языки программирования для выполнения этого действия могут иметь разный синтаксис. В следующем подразделе описан подход языка C/AL.

Синтаксис присваивания

Мы уже видели, что определенные вызовы функций, подобно функции `MESSAGE`, которая была представлена в уроке 3, сами могут быть операторами. Мы называем их операторами вызова функций ("Function Call Statements"). Синтаксис оператора вызова функции прост:

```
<функция вызов>
```

Результат вызова функции будет сильно варьироваться в зависимости от того, какая функция вызывается. Более того, сам синтаксис вызова может меняться.

Синтаксис оператора присваивания почти столь же прост, как и оператор вызова функции:

```
<переменная> := <выражение>
```

Двоеточие со знаком равенства (:=) обозначает оператор присваивания. Этот оператор вычисляет выражение и присваивает переменной результирующее значение. Подробнее о выражениях мы поговорим в будущих уроках, а пока же будем использовать либо константу, либо другую переменную в правой части оператора присваивания. Константа и переменная могут выступать в качестве простейшего выражения.

4.3 АВТОМАТИЧЕСКОЕ ПРЕОБРАЗОВАНИЕ ТИПОВ

Для того чтобы переменной было присвоено некое значение, тип значения должен соответствовать типу переменной. В то же время, определенные типы (с учетом ограничений) могту автоматически преобразовываться во время присваивания.

Строковые типы данных

Оба строковых типа данных (Code и Text) могут автоматически преобразовываться друг в друга. Например, если переменная Description имела тип Text, а переменная "Code Number" была типа Code, то допустим следующий оператор:

```
"Code Number" := Description
```

Текстовое значение в Description будет преобразовано в кодовое перед присваиванием переменной "Code Number". Это означает, что все буквы из нижнего регистра будут приведены к верхнему регистру и что все предшествующие и замыкающие пробелы будут удалены. Таким образом, значение, присвоенное кодовой переменной, будет кодового типа. Заметим, что это процесс преобразования не затрагивает самого значения, хранящегося в переменной Description. Переменные, находящиеся по правую сторону операции присваивания, не затрагиваются при присваивании, а модифицируется лишь одна переменная, находящая с левой стороны знака операции присваивания.

Примечание

При выполнении автоматического преобразования существуют определенные ограничения. Например, предположим, что значение текстовой переменной Description имеет больше символов, чем то, которое может вместить кодовая переменная "Code Number". В этом случае при выполнении оператора присваивания на этапе выполнения возникнет ошибка, известная как ошибка этапа выполнения (Run-Time Error).

Числовые типы данных

Все числовые типы (целый, десятичный, вариантный и символьный) могут также автоматически преобразовываться друг в друга. Здесь имеются такие ограничения:

- Чтобы быть преобразованным к любому типу данных десятичное значение должно быть целым числом (не иметь дробной части).

- Если значение выходит за предел допустимого диапазона, предусмотренного типом переменной, то преобразование осуществляться не будет. Это справедливо для всех типов данных. Например, десятичное значение не может быть преобразовано в целое или вариантное, если его значение не будет лежать в диапазоне от -2 147 483 647 до +2 147 483 647 (допустимого диапазона для целого типа).

- Десятичное, целое и вариантное значения не могут быть преобразованы в символьное, пока они не будут лежать в диапазоне от 0 до 255 (допустимого диапазона для символьного типа). Если предпринимается попытка любого из подобных преобразований с выходом за допустимый диапазон представления, то будет генерировать ошибка этапа выполнения.

Другие типы данных

Хотя строковые типы данных могут автоматически преобразовываться друг в друга (что справедливо и для числовых тиипов), никакие иные типы переменных не могут быть автоматически преобразованы друг в друга с использованием присваивания.

4.4 Разделитель операторов

Как уже было сказано ранее, один оператор может занимать несколько строк кода, и одна строка кода может содержать несколько операторов. Если подобная ситуация возникает, то как компилятор C/AL определяет, где один оператор завершается, а другой начинается? Он распознает разделитель операторов в качестве такого индикатора. Разделителем операторов служит точка с запятой (;). Когда компилятор достигает точки с запятой (или конца триггера), он знает, что достиг конца оператора. Ниже приведен синтаксис триггера:

```
[ <оператор> { ; <оператор> } ]
```

Квадратные скобки используются для указания того, что заключенный в них фрагмент носит необязательный характер; он может встречаться 0 или 1 раз. Все, что заключено в фигурные скобки, также носит необязательный характер и может встречаться 0 или более раз. Другими словами, в триггере может появляться 0 или более операторов, при этом они отделяются друг от друга точкой с запятой.

Заметим, что за точкой с запятой обязательно должен идти оператор. Но как быть в конце триггера, где вы часто после последнего оператора ставите точку с запятой? В этой ситуации после последней точки с запятой и перед концом триггера автоматически вставляется пустой оператор (Null Statement). Он ничего не обозначает и ничего не делает, что и отражает его название. Хотя это и спорное суждение, важно рассматривать точку с запятой не как конец оператора (это не признак завершения оператора), а как признак появления нового оператора (это разделитель операторов). Подобное различие будет критично в понимании синтаксиса других видов операторов.

4.5 Использование операторов присваивания и меню Symbol

Следующие упражнения дадут вам навыки использования операторов присваивания и познакомят вас с весьма полезным инструментарием, доступным через меню Symbol.

Упражнение 1 **Простой оператор присваивания**

1 Перейдите в Object Designer, выберите Codeunit 95100 и щелкните Design.

2 В триггере OnRun введите следующие два оператора, удаляя любой код, который до этого там был введен.

```
LoopNo := 25;
MESSAGE('The value of %1 is %2','LoopNo',LoopNo);
```

3 Теперь осуществите выход, сохраните и запустите программный модуль.

Упражнение 2 **Множественные сообщения**

1 Снова вернитесь и отредактируйте программный модуль 95100. После введенного ранее кода добавьте следующие строки:

```
LoopNo := -30;
MESSAGE('The value of %1 is %2','LoopNo',LoopNo);
Amount := 27.50;
MESSAGE('The value of %1 is %2','Amount',Amount);
"When Was It" := 093097D;
MESSAGE('The value of %1 is %2','When Was It","When
Was It");
"Code Number" := ' abc 123 x';
MESSAGE('The value of %1 is %2','Code Number","Code
Number");
```

2 Теперь осуществите выход, сохраните и запустите программный модуль.

Заметим, что сообщения будут появляться последовательно, одно за другим. По мере того, как вы будете нажимать кнопку OK для каждого сообщения, будет появляться следующее. Кроме того, если вы посмотрите на последнее сообщение (оно отображает Code Number), то увидите, что здесь происходит автоматическое преобразование типов. Когда значение преобразовывалось от текстового типа к кодовому типу, предшествующие и замыкающие пробелы были удалены, а буквы переведены в верхний регистр.

Еще один момент не является очевидным: сообщения на экране не отображаются до тех пор, пока не завершится обработка. Другими словами, по мере того, как функция MESSAGE неоднократно вызывается, соответствующие сообщения добавляются в очередь. Очередь здесь подобна очереди в кассу: люди встают в один конец очереди и выходят из нее с другого конца по мере своего прохождения.

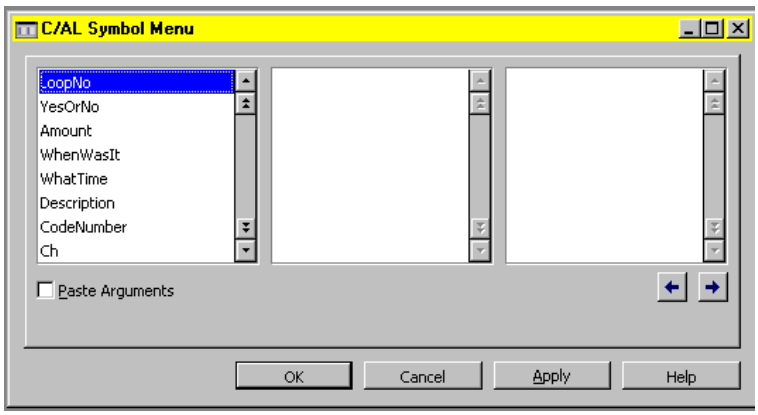
Как только программный модуль завершит свое выполнение, сообщения будут считаны из очереди и отображены на экране в порядке поступления в очередь. Поскольку сообщения сразу не отображаются, они не прерывают процесс обработки. Однако, это также означает, что они не могут использоваться для того, чтобы дать пользователю канал обратной связи в ходе обработки. Они могут применяться только для передачи пользователю результатов обработки.

Упражнение 3 Использование меню Symbol

Мы собираемся добавить дополнительные строки в программный модуль 95100, используя несколько иной метод. Перейдите к концу существующего кода, как будто бы вы вели ввод текста на другой строке. Вместо того, чтобы выбирать из строки меню пункты View, C/AL Symbol Menu , нажмите клавишу F5, либо щелкните на строке инструментов значок C/AL Symbol Menu.

C/AL Symbol Menu

Появится окно **C/AL Symbol Menu**:



В левой створке среди всего прочего вы увидите список всех идентификаторов, заданных программистов. Щелкните на строке, где указано "YesOrNo", затем щелкните кнопку OK. Меню Symbol исчезнет и в вашем коде появится слово `YesOrNo`.

После `YesOrNo` наберите:

```
:= TRUE;
```

теперь вся строка выглядит так:

```
YesOrNo := TRUE;
```

Затем нажмите клавишу ENTER и продолжите набор на следующей строке:

```
MESSAGE('The value of %1 is %2','
```

В ситуации, когда курсор находится сразу за последней введенной вами одинарной кавычкой, еще раз нажмите клавишу F5, выберите в меню Symbol пункт YesOrNo (тот, что был выбран в последний раз) и щелкните кнопку OK. Введите затем одиночную кавычку и запятую. Еще раз нажмите клавишу F5 и в то же время нажмите клавишу ENTER (она наживает кнопку OK в меню Symbol). Завершите ввод, набрав закрывающую скобку и точку с запятой. В результате получится следующее:

```
YesOrNo := TRUE;

MESSAGE('The value of %1 is %2','YesOrNo',YesOrNo);
```

Попробуйте еще раз использовать этот метод для переменной Description, но установите значение (после оператора присваивания) в

'Now is the time.' В результате получится следующее:

```
Description := 'Now is the time. ';

MESSAGE('The value of %1 is %2',
'Description',Description);
```

Вновь используем меню Symbol для ввода двух других строк, отвечающих за задание и отображение "What Time". Установите значение этой переменной, равное 153000T. В результате получится следующее:

```
"What Time" := 153000T;

MESSAGE('The value of %1 is %2','"What Time"',"What
Time");
```

Заметим, что при использовании меню Symbol везде, где необходимо, вставляются двойные кавычки. В данном случае, однако, двойные кавычки внутри одинарных не требуются, поэтому вам нужно вернуться и их удалить. В результате получится:

```
"What Time" := 153000T;

MESSAGE('The value of %1 is %2','What Time',"What
Time");
```

Теперь надо выйти, сохранить программный модуль и снова его запустить.

Обратите внимание, что отображение Time и Date зависит от тех установок, которые были сделаны в вашей системной Панели управления (Региональные стандарты). На большинстве американских систем, например, время будет отображаться как 3:30:00 PM. Но внеся простые изменения в Панели управления, вы сможете добиться отображения 15:30:00. Заметим, что это не требует никакого изменения кода и даже перекомпиляции в рамках среды C/SIDE. Достаточно изменений в самой Панели управления..

Упражнение 4 Символьные константы

В силу автоматического преобразования типов символьные переменные могут присваивать значения в виде числа или односимвольного текста.

Введите в своем программном модуле следующие строки:

```
Ch := 65;
MESSAGE('The value of %1 is %2','Ch',Ch);
Ch := 'A';
MESSAGE('The value of %1 is %2','Ch',Ch);
```

Обратите внимание, что оба значения 65 и 'A' приводят к появлению на экране одного и того же символьного значения. Это происходит по той причине, что 65 – это ASCII-код для буквы A, находящейся в верхнем регистре. Вы можете поэкспериментировать с с другими числами, чтобы посмотреть на аналогичные результаты. Эти ASCII-коды используются для представления символов в среде C/SIDE в той же степени, как и в других программах и компьютерных системах.

Упражнение 5 Константы вариантов

Ранее мы применяли целые числа в качестве вариантных констант. В среде C/SIDE для записи таких констант можно использовать и иной синтаксис. Введите в своем программном модуле следующие строки:

```
Color := 2;
MESSAGE('The value of %1 is %2','Color',Color);
Color := Color::Yellow;
MESSAGE('The value of %1 is %2','Color',Color);
```

Как только вы сохраните и запустите новый код, то увидите, что результаты для обоих сообщений одинаковые. Первый вариант всегда нумеруется значением 0, поэтому третий, а в нашем случае Yellow, нумеруется значением 2.

Обратите внимание, что синтаксис, используемый для вариантных констант выглядит так: после идентификатора переменной следуют два двоеточия, а за ними идет идентификатор варианта.

Упражнение 6 **Ошибки этапа выполнения**

Как мы уже говорили ранее, не всякое значение может быть автоматически приведено к другому типу.

Попробуйте ввести следующие строки в свое программный модуль, откомпилировать и запустить его. Посмотрите, какие виды ошибок вы получите:

```
Description := 'Now is the time. Here is the place.';
LoopNo := 27.5;
YesOrNo := 1;
Amount := 27.5; LoopNo := Amount;
```

Обратите внимание, что только первая и последняя строка будут откомпилированы, так как компилятор достаточно интеллектуальный, чтобы определить, что некоторые вещи никогда работать не будут. В первом случае и константа и переменная принадлежат текстовому типу. Ошибка переполнения здесь возникнет на этапе выполнения программного модуля, так как переменная Description может иметь длину максимум в 30 символов, а ей присваивают текстовую константу длиной 36 символов.

В последнем случае переменная **Amount** содержит значение, которое при выполнении программы не может быть преобразовано к целому типу. Такие виды ошибок называются ошибками этапа выполнения (Run-Time Errors). Заметим, что когда возникает такая ошибка, она приводит к остановке всей дальнейшей обработки и порождает вывод сообщения об ошибке.

4.6 Контрольные вопросы

Вопросы

Чтобы ответить на следующие вопросы, вы можете воспользоваться своим тестовым программным модулем. В нем были определены все переменные, на которые мы здесь ссылаемся.

1 Что появится на экране, если вы отобразите значение переменной Amount после того, как присвоили ей значение 15.00?

2 Каким будет значение, если вы вслед за этим присвоили значение Amount переменной LoopNo и затем вывели на экран значение LoopNo?

3 Что появится на экране, если вы задали переменной Color значение 4 и затем вывели значение Color на экран?

4 Что появится на экране, если вы задали переменной Color значение 8 и затем вывели значение Color на экран?

5 Присвойте переменной Description строку 'the time.', затем присвойте значение Description переменной Code Number и отобразите Code Number на экране. Что вы увидите?

6 Присвойте переменной Description строку 'Now is the time.', затем присвойте значение Description переменной Code Number и отобразите Code Number на экране. Что вы увидите?

7 Что появится на экране, если вы присвоите переменной Color значение Green, потом присвоите значение Color переменной LoopNo и отобразите значение переменной LoopNo?

Ответы

1 Что появится на экране, если вы отобразите значение переменной Amount после того, как присвоили ей значение 15.00?

15

2 Каким будет значение, если вы вслед за этим присвоили значение Amount переменной LoopNo и затем вывели на экран значение LoopNo?

15

3 Что появится на экране, если вы задали переменной Color значение 4 и затем вывели значение Color на экран?

Blue

4 Что появится на экране, если вы задали переменной Color значение 8 и затем вывели значение Color на экран?

8

5 Присвойте переменной Description строку 'the time.', затем присвойте значение Description переменной Code Number и отобразите Code Number на экране. Что вы увидите?

THE TIME.

6 Присвойте переменной Description строку 'Now is the time.', затем присвойте значение Description переменной Code Number и отобразите Code Number на экране. Что вы увидите?

A run-time error (text overflow).

7 Что появится на экране, если вы присвоите переменной Color значение Green, потом присвоите значение Color переменной LoopNo и отобразите значение переменной LoopNo?

3

Глава 5

Выражения

Данная глава посвящена выражениям, операциям, операндам и вычислению значений. В ней поясняется, как работают строковые операции и как использовать функции в выражениях. Кроме того, раскрывается нюансы приоритетов операций для числовых выражений и действия бинарных операций для различных типов данных.

Глава содержит следующие разделы:

- Выражения, операнды и операции
- Синтаксис выражений
- Операция работы со строками
- Вызовы функций в выражениях
- Контрольные вопросы

5.1 Выражения, операнды и операции

Мы начнем главу с описания терминов, а в последующих разделах покажем, как они используются.

Выражение

Выражение (*expression)* подобно уравнению, помещенному в программу. Это формула, сообщающая компьютеру, как в точности генерировать требуемое значение. Подобно переменной и константе, выражение имеет тип и значение. Однако в отличие от константы выражение, чтобы узнать его значение, должно вычисляться во время выполнения. Хотя значение константы известно всегда, а значение переменной определяется во время выполнения, система должна обратиться к ячейке памяти. Ниже приводятся примеры выражений:

- FunctionX + 7
- Quantity * UnitCost

Вычисление значений

Вычислять (*evaluate)* значение выражения означает фактически следовать инструкциям, задаваемым формулой, и определять в это время тип и значение выражения. В зависимости от значений переменных, входящих в выражение, само значение выражения может меняться, хотя тип его остается неизменным.

Операнд

Операнд, или терм (*term),* представляет собой часть выражения, которая участвует в вычислении значения выражения. В роли операнда может выступать переменная, константа, вызов функции (если функция возвращает значение). Операнд, в свою очередь, может быть другим выражением, заключенным в круглые скобки. Такая разновидность операнда известна как подвыражение (subexpression). Приведенные выше выражения имеют следующие операнды:

- FunctionX
- 7
- Quantity
- UnitCost

Операция

Операция (*operator)* является частью выражения, которая воздействует либо прямо на идущей за ней операнд (так называемая унарная операция, unary operator), либо на операнды по обе стороны знака операции (так называемая бинарная операция, binary operator). Операции записываются знаками (например, +, >, /, =) или же зарезервированными словами (например, DIV, MOD). Они уже определены в языке C/AL и программистом не могут быть ни переопределены, ни вновь добавлены. Во время вычисления выражения, когда операция выполняется над операндами, возникает новое значение, которое может быть значением выражения либо же операндом, используемым другой операцией. Некоторые примеры операций и их использования:

- 5 + (-8) ‘+’ является бинарной арифметической операцией, а ‘-‘ является унарной арифметической операцией.
- TotalCost/Quantity ’/’ является бинарной операцией.
- ‘cat’ + ‘ and dog’ ‘+’ здесь используется в качестве бинарной операции работы со строками
- (Quantity > 5) OR (Cost <= 200) ‘OR’ является бинарной логической операцией; ‘>’ и ‘<=’ являются бинарными реляционными операциями.

5.2 Синтаксис выражений

Описание синтаксиса выражений требует дополнительных синтаксических конкструкций по сравнению с теми, что мы уже рассмотрели. Поэтому синтаксис выражений достаточно сложен.

Используемый ниже знак ::= говорит о том, что элемент по левую сторону от этого знака определяется синтаксисом, указанным с правой стороны от знака. Заметим, что вертикальная черта (|) имеет значение "ИЛИ". Любые элементы, разделенные вертикальными чертами, считаются корректным синтаксисом.

- <унарная операция> ::= + | - | NOT
- <операция работы со строками> ::= +
- <арифметическая операция> ::= + | - | * | / | DIV | MOD
- <реляционная операция> ::= < | > | = | <= | >= | <> | IN
- <логическая операция> ::= NOT | OR | AND | XOR
- <операция> ::= <операция работы со строками> | <арифметическая операция> | <реляционная операция> | <логическая операция>
- <простой операнд> ::= <константа> | <переменная> | <вызов функции> | (<выражение>)
- <операнд> ::= <простой операнд> | <унарная операция> <простой операнд>
- <выражение> ::= <операнд> { <операция><операнд> }

Помните, что окружающие элемент фигурные скобки означают ноль или более повторений данного элемента. В этой и последующих главах мы коснемся как различных видов операций, так и различных типов выражений.

5.3 ОПЕРАЦИЯ РАБОТЫ СО СТРОКАМИ

Существует только одна операция работы со строками. Она обозначается знаком “плюс” (+) и отвечает за конкатенацию. Конкатенацией (concatenation) называется действие, которое “склеивает” две или более строк в одно целое. Операция конкатенации является бинарной операцией. Это означает, что она работает с левым и правым операндами. Оба операнда должны быть строками, т.е. относиться либо к типу Code, либо к типу Text. Если оба операнда типа Code, то результатом конкатенации будте тип Code. В противном случае результатом будет тип Text.

Пример выражения

Ниже приведен пример выражения и того, как оно вычисляется. Сначала идет набор присваиваний, а затем само выражение:

```
CodeA := 'HELLO THERE';

TextA := 'How Are You? ';

CodeB := CodeA + '. ' + TextA;
```

Выражение вычисляется следующим образом.

Сначала берется значение переменной `CodeA`.

Затем значение текстовой константы прикрепляется к точке с пробелом. Поскольку `CodeA` имеет тип Code, а текстовая константа – тип Text (обратите внимание на замыкающий пробел), то результатом будет значение типа Text, равное `'HELLO THERE. '`. Это значение становится первым операндом для следующей операции конкатенации.

Теперь берется значение переменной `TextA`. Затем это значение прикрепляется в конец ранее сгенерированного текста. Так как оба значения имеют тип Text, то и результирующим типом будет Text, а само значение станет равным `'HELLO THERE. How Are You? '`. Поскольку достигнут конец выражения, этот результат становится результатом всего выражения.

Как только значение выражения вычислено, новое значение присваивается переменной `CodeB` с помощью операции присваивания. Так как тип выражения равен Text, а тип переменной `CodeB` равен Code, то результат выражения должен быть

преобразован в Code путем автоматического преобразования типов. Результат такого преобразования выразится в том, что строка `'HELLO THERE. HOW ARE YOU?'` будет присвоена переменной CodeB.

Упражнение

Перейдите в Object Designer, выделите программный модуль 95100 и щелкните кнопку Design. В текущий список переменных добавьте следующие глобальные переменные, выбрав на строке меню пункты View, C/AL Globals:

```
CodeA Code  30

CodeB Code  50

TextA Text  50

TextB Text  80
```

В триггере OnRun введите следующие четыре оператора, удалив предварительно весь код, ранее введенный в ходе предыдущих упражнений.

```
CodeA := 'HELLO THERE';

TextA := 'How Are You? ';

CodeB := CodeA + '. ' + TextA;

MESSAGE('The value of %1 is %2','CodeB',CodeB);
```

Теперь осуществите выход, сохраните и запустите программный модуль. Вы должны получить тот же самый результат, который был описан выше.

5.4 Вызовы функций в выражениях

Помните, что если вы попытаетесь присвоить строковой переменной слишком много символов, получится ошибка этапа выполнения. Как же этого избежать, ведь ошибка не появится, пока программа не будет запущена? Один из подходов – спроектировать свою программу таким образом, чтобы эта ошибка никогда не возникала. Так, как это сделано в большей части Financials. В некоторых случаях это сделать невозможно. К счастью, есть другой подход.

Функция MAXSTRLEN

Имеется функция, которая может сообщить нам (во время выполнения) максимальную длину строки, которую может вместить переменная. Эта функция называется MAXSTRLEN. Она имеет один параметр – саму переменную, для которой определяется максимальная длина. Возвращаемое значение имеет тип Integer.

Введите в программный модуль 95100 после имеющихся там строк следующие две строки кода:

```
LoopNo := MAXSTRLEN(Description);

MESSAGE('The value of %1 is %2','LoopNo',LoopNo);
```

В результате значение переменной LoopNo станет равным 30. Это длина, которая была использована при создании переменной.

Заметим, что мы знаем, сколь длинная строка может быть помещена в переменную, но как мы сможем записать туда допустимый максимум символов? Сначала посмотрим на получающийся результат при условии, что мы ничего сами не предпринимаем. После имеющегося кода напишите следующие строки:

```
Description := 'The message is: ' + CodeB;

MESSAGE('The value of %1 is
%2','Description',Description);
```

Как только вы сохраните и запустите этот код, вы получите ожидавшуюся ошибку этапа выполнения. Проблема в том, что оператор присваивания передает все значение выражения, находящегося по правую сторону от знака присваивания, а в той переменной, что расположена по левую сторону от этого знака, нет для такой строки достаточного места. Нам нужно извлечь часть

текста.

Такое извлечение достигается с помощью функции COPYSTR.

Эта функция имеет три параметра.

- Первый параметр – это строка, из которой должна копироваться информация.
- Второй параметр – позиция первого символа в исходной строке, с которого должно начинаться копирование. Если мы хотим начать копировать с пятого символа, то второй параметр равен 5. Если же хотим начать с первого символа, то второй параметр равен 1.
- Третий параметр – это количество сиволов, подлежащих копированию. Этот параметр является необязательным и если он отсутствует, то копируются все символы строки, начиная от символа, указанного во втором параметре, и до конца строки. Если вы попытаетесь скопировать больше символов, чем возможно, то не произойдет ничего страшного. Максимум будет скопировано количество символов, содержащихся в исходной строке. Значение функции COPYSTR, как только оно вычислено, является копией первого параметра с учетом правила извлечения, описываемого вторым и третьим параметрами. Если первый параметр имеет тип Text, то типом результата функции COPYSTR будет тоже Text. Если первый параметр имеет тип Code, то типом результата функции COPYSTR также будет Code.

Теперь найдите в нашем программном модуле строку, содержащую такую запись:

```
Description := 'The message is: ' + CodeB;
```

Измените ее на следующее:

```
Description := COPYSTR('The message is: ' +
CodeB,1,MAXSTRLEN(Description));
```

Когда вы сохраните изменения и запустите модуль, то больше ошибки этапа выполнения вы не получите. Теперь переменная Description будет содержать столько символов, сколько может разместить.

Как это работает?

Давайте вычислим значение выражения вручную и посмотрим, как это

работает.

Выражение – это то, что находится в правой части оператора присвивания. Ниже приводится исходное выражение:

```
COPYSTR('The message is: ' +
CodeB,1,MAXSTRLEN(Description))
```

Сначала, чтобы вычислить значение функции COPYSTR, нам нужно вычислить каждый из ее параметров. Первый шаг – вычисление первого параметра, т.е. получение значения переменной CodeB и конкатенация его (обратите внимание на знак плюса) со строковой константой. В результате получим:

```
COPYSTR('The message is: HELLO THERE. HOW ARE
YOU?',1,MAXSTRLEN(Description))
```

Следующий наш шаг – вычисление второго параметра. Раз это константа, то значение уже есть. Далее мы должны вычислить третий параметр. В роли этого параметра выступает функция, которая возвращает заданную длину своего параметра (тот уже должен быть переменной, а никак не выражением). В результате получим:

```
COPYSTR('The message is: HELLO THERE. HOW ARE
YOU?',1,30)
```

Наконец, вычисляется сама функция COPYSTR. Она копирует символы из текста, заданного в первом параметре, начиная с символа, указанного во втором параметре, и длиной до 30 символов (что фиксирует третий параметр). Результат будет таким:

```
'The message is: HELLO THERE. '
```

Заметим, что как только выражение вычислено, может осуществляться и само присваивание:

```
Description := 'The message is: HELLO THERE. ';
```

Переменная Description теперь имеет новое значение.

5.5 Контрольные вопросы

Вопросы

1 Подчеркните выражение в этом операторе присваивания:

TextA := TextB;

2 В математике операции работают с тем, что называется операндами. Как эти же вещи называются в выражениях языка программирования?

3 Что выполняет операция, обозначаемая плюсом (+), по отношению к текстовым переменным и константам?

4 Подчеркните выражение в этом операторе присваивания:

```
TextA := 'The ' + TextB;
```

5 Подчеркните операции в следующем выражении:

```
TextA := 'The ' + TextB;
```

6 Подчеркните операнды в следующем выражении:

```
TextA := 'The ' + TextB;
```

Ответы вы можете увидеть на следующей странице.

Ответы

1 Подчеркните выражение в этом операторе присваивания:

TextA := TextB;

2 В математике операции работают с тем, что называется операндами. Как эти же вещи называются в выражениях языка программирования?

Термы

3 Что выполняет операция, обозначаемая плюсом (+), по отношению к текстовым переменным и константам?

Она осуществляет их конкатенацию.

4 Подчеркните выражение в этом операторе присваивания:

TextA := 'The ' + TextB;

5 Подчеркните операции в следующем выражении:

TextA := 'The ' + TextB;

6 Подчеркните операнды в следующем выражении:

TextA := 'The ' + TextB;

Глава 6

Числовые выражения

Данная глава посвящена арифметическим операциям языка C/AL и особенностям приоритета операций для числовых выражений.

Глава содержит следующие разделы:

- Числовые выражения и приоритет операций
- Виды арифметических операций
- Особенности приоритета операций
- Контрольные вопросы

6.1 ЧИСЛОВЫЕ ВЫРАЖЕНИЯ И ПРИОРИТЕТ ОПЕРАЦИЙ

В данной главе часто используются термины “числовые выражения” (numeric expressions) и “приоритет операций” (operator precedence). Ниже даются их определения.

Арифметические операции

Арифметические операции (arithmetic operators) – это операции, которые используются в числовых выражениях для работы с числовыми и нечисловыми операндами (terms). Арифметические операции обозначаются знаками сложения, вычитания, умножения и деления.

Числовое выражение

Числовое выражение (numeric expression) – это такой вид выражения, который использует как минимум одну арифметическую операцию и дает результат числового типа. Когда вычисляется числовое выражение, получается результат типа Decimal, Integer, Option или Char. И хотя отдельные операнды числового выражения могут и не быть числовыми, результат все же будет числовым.

Приоритет операций

Приоритет операций (operator precedence) задает порядок, в котором выполняются операции в данном выражении. Операции с более высоким приоритетом выполняются до операций с более низким приоритетом. Например, поскольку операция умножения (*) имеет более высокий приоритет, чем операция сложения (+), то выражение 5 + 2 * 3 будет иметь результат 11, а не 21, который получился, если бы операции выполнялись просто слева направо.

6.2 Виды арифметических операций

В языке C/AL существуют шесть разных арифметических операций.

Операция сложения (+)

Операция сложения (plus operator) используется для разных целей. Как уже было показано ранее, если операнды по обе стороны операции принадлежат строковым типам, то результатом тоже будет строка. В то же время плюс может использоваться как для унарной, так и для бинарной арифметической операции.

Когда плюс используется для унарной операции, то его функция состоит в том, что он не меняет знака идущего за ним операнда. По сути он ничего не обозначает и используется редко. Если же используется, то его задача – показать, что значение положительное. Ниже приведен пример выражения, использующего унарную операцию сложения, и того же выражения, но без этой операции:

- IntVariable * +11
- IntVariable * 11

При использовании плюса в качестве бинарной операции его назначение состоит в том, чтобы добавить идущий за ним операнд к тому операнду, который идет перед ним. Оба операнда могут быть числовыми, либо один может быть типа Date или Time, а другой – типа Integer. Если один из операндов имеет десятичное значение (Decimal), то и результат будет десятичным. Если оба операнда принадлежат типу Char и их сумма меньше 256, то результат будет типа Char; в противном случае результат будет типа Integer. Если оба операнда принадлежат типу Option или Integer, а их сумма дает допустимое для Integer значение, то результат будет типа Integer. В противном случае результат будет типа Decimal.

Если один операнд имеет тип Date, а другой – Integer, то результатом будет дата, которая отстоит от даты операнда типа Date на количество дней, указанных в операнде типа Integer (последующая дата). Таким образом, значение выражения `03202001D + 7` будет `03272001D`. Если в итоге получается недопустимая дата, то будет генерироваться ошибка этапа выполнения.

Аналогичным образом, если один операнд имеет тип Time, а другой – Integer, то результатом будет момент времени, который отстоит от момента времени операнда типа Time на количество миллисекунд,

указанных в операнде типа Integer (последующее время). Соответственно, значением выражения `115815T + 350000` будет `120405T`. Если в итоге получается недопустимое время, то будет генерироваться ошибка этапа выполнения.

Использование операции сложения по отношению к операндам типа Date или Time не является примером арифметической операции, поскольку результат получается нечисловой. Однако мы приводим данные о работе с Date и Time для удобства. В представленной ниже таблице вы найдете соответствующую информацию.

Заметим, что левая колонка отвечает за тип левого операнда данной операции, тогда как верхняя строка – за тип правого операнда.

Сложение (+)	**Char**	**Option**	**Integer**	**Decimal**	**Date**	**Time**
Char	Char	Integer	Integer	Decimal	N/A	N/A
Option	Integer	Integer	Integer	Decimal	N/A	N/A
Integer	Integer	Integer	Integer	Decimal	N/A	N/A
Decimal	Decimal	Decimal	Decimal	Decimal	N/A	N/A
Date	Date	Date	Date	N/A	N/A	N/A
Time	Time	Time	Time	N/A	N/A	N/A

Помните, что если результат должен быть типа Char, но если получается недопустимое для Char значение, то результат будет типа Integer. Если результат должен быть типа Integer, но получается недопустимое для Integer значение, то результат будет типа Decimal.

Операция вычитания (-)

Подобно операции сложения операция вычитания (minus operator) может использоваться как в унарном, так и в бинарном варианте. Когда знак минус используется для обозначения унарной операции, то ее функция состоит в смене знака правого операнда.

Когда минус используется для обозначения бинарной операции, то ее функция заключается в вычитании правого операнда из левого. Оба операнда могут быть числового типа, либо оба типа Date или типа Time, либо же левый операнд типа Date или Time, а правый – типа

Integer.

Если первый операнд типа Date, а второй – Integer, то результатом будет дата, которая отстоит от даты операнда типа Date на количество дней, указанных в операнде типа Integer (предшествующая дата). Таким образом, значение выражения `02252001D - 7` будет `02182001D`. Если в итоге получается недопустимая дата, то будет генерироваться ошибка этапа выполнения.

Если один операнд имеет тип Time, а другой – Integer, то результатом будет момент времени, который отстоит от момента времени операнда типа Time на количество миллисекунд, указанных в операнде типа Integer (предшествующее время). Соответственно, значение выражения `115815T - 350000` будет `115225T`. Если в итоге получается недопустимое время, то будет генерироваться ошибка этапа выполнения.

Если одна дата вычитается из другой, то результатом будет целое число, равное количеству дней между двумя датами. Если один момент времени вычитается из другого, то результатом будет целое число, равное количеству миллисекунд между двумя моментами времени.

В следующей таблице систематизирована информация о результирующем типе операции вычитания в случае использования операндов разных типов. Заметим, что левая колонка отвечает за тип левого операнда данной операции, тогда как верхняя строка – за тип правого операнда.

Вычитание (-)	**Char**	**Option**	**Integer**	**Decimal**	**Date**	**Time**
Char	Char	Integer	Integer	Decimal	N/A	N/A
Option	Integer	Integer	Integer	Decimal	N/A	N/A
Integer	Integer	Integer	Integer	Decimal	N/A	N/A
Decimal	Decimal	Decimal	Decimal	Decimal	N/A	N/A
Date	Date	Date	Date	N/A	Intege	N/A

					r	
Time	Time	Time	Time	N/A	N/A	Integ er

Помните, что если результат должен быть типа Char, но получается недопустимое для Char значение, то результат будет типа Integer. Если результат должен быть типа Integer, но получается недопустимое для Integer значение, то результат будет типа Decimal.

Операция умножения (*)

Операция умножения (times operator), или мультипликативная операция (multiplication operator), используется только как бинарная операция. Ее назначение – умножение левого операнда на правый.

В следующей таблице систематизирована информация о результирующем типе операции умножения в случае использования операндов разных типов.

Умножение (*)	**Char**	**Option**	**Integer**	**Decimal**
Char	Char	Integer	Integer	Decimal
Option	Integer	Integer	Integer	Decimal
Integer	Integer	Integer	Integer	Decimal
Decimal	Decimal	Decimal	Decimal	Decimal

Здесь применяются обычные правила автоматического преобразования от Char к Integer и затем к Decimal.

Операция деления (/)

Операция деления (divide operator) используется только в качестве бинарной операции. Ее назначение состоит в том, чтобы разделить левый операнд на правый. Результирующим типом данной операции деления всегда будет Decimal. Если правый операнд равен нулю (0), то результатом будет ошибка этапа выполнения.

Операция целочисленного деления (DIV)

Операция целочисленного деления (integer divide operator) используется только в качестве бинарной операции. Ее назначение

состоит в том, чтобы разделить левый операнд на правый. Результирующим типом данной операции деления всегда будет Integer. Если правый операнд равен нулю (0), то результатом будет ошибка этапа выполнения. Любые десятичные значения, которые получаются в ходе обычного деления, усекаются, а не округляются. Т.е. результатом выражения `17 DIV 8` будет `2`, тогда как результатом выражения `17 DIV 9` будет `1`.

Операция взятия остатка от деления (MOD)

Операция взятия остатка от деления (modulus operator) требует двух операндов. Первый операнд подвергается преобразованиям с помощью функции MOD и второго операнда. Второй операнд отвечает за основание системы счисления (number system). По определению, результат меняется от 0 до значения второго операнда, уменьшенного на 1. Например, если второй операнд равен 10, то результат операции MOD будет меняться от 0 до 9. Таким образом, операция MOD отвечает за то, что из первого операнда будет итеративно вычитаться основание вашей системы счисления (указанной во втором операнде) до тех пор, пока можно будет оставаться в области неотрицательных целых чисел.

Ниже приведены некоторые примеры:

- 15 MOD 10 равно 5 (поскольку 9 – это последнее доступное число, а 10 соответствует 0, 11 соответствует 1, 12 соответствует 2 и т.д.)
- 6 MOD 10 равно 6
- 10 MOD 10 равно 0
- 127 MOD 10 равно 7

Обратите внимание, что вы получите тот же самый результат, если разделите первый операнд нацело на второй и найдете остаток от этого деления.

Операция MOD (modulus operator, remainder operator) используется тоько в качестве бинарной операции. Ее назначение в том, чтобы разделить первый операнд на второй методом целочисленного деления, а затем получить остаток от этого деления. Результирующий тип операции всегда будет Integer. Если второй операнд равен (0), то будет генерироваться ошибка этапа выполнения .

- 17 MOD 8 = 1

- 17 MOD 9 = 8.

Упражнение 1 Арифметическая операция

1 Перейдите в Object Designer, выберите программный модуль 95100 и щелкните Design.

2 Добавьте в текущий список переменных следующие глобальные переменные, выбрав в строке меню пункты View, C/AL Globals:

```
Int1  Integer
Int2  Integer
Amt1  Decimal
Amt2  Decimal
```

3 В триггере OnRun, удалив из него предварительно все, что было введено в ходе предыдущих уроков, добавьте следующие операторы.

```
Int1 := 25 DIV 3;
Int2 := 25 MOD 3;
LoopNo := Int1 * 3 + Int2;
MESSAGE('The value of %1 is %2','LoopNo',LoopNo);
Amt1 := 25 / 3;
Amt2 := 0.00000000000000001;
Amount := (Amt1 - Int1) * 3 + Amt2;
MESSAGE('The value of %1 is %2','Amount',Amount);
```

4 Теперь осуществите выход, сохраните исправления и запустите программный модуль. В результате вы получите, что LoopNo станет равным 25, а Amount получит значение 1.

6.3 Особенности приоритета операций

Вам необходимо разбираться в приоритетах операций, чтобы потом не удивляться непредсказуемым результатам.

Уровни приоритета операций

Для арифметических операций различают три уровня приоритета операций.

- Наивысший уровень – это уровень унарной операции, включая плюс (+) и минус (-).
- Средний уровень – это мультипликативный уровень, который включает умножение (*), обе разновидности деления (/, DIV) и операцию взятия остатка от деления (MOD).
- Наименьший уровень – это аддитивный уровень приоритета, куда входят бинарные сложение (+) и вычитание (-).

Обычно вычисление выражения производится слева направо. Однако, если некая операция имеет более высокий приоритет, нежели та, что встречается раньше, то именно она и и выполняется до более низкоприоритетной. Чтобы избежать этих нюансов, программист может создавать подвыражения, заключая части выражения в круглые скобки. Подвыражения всегда вычисляются в первую очередь.

Цель этого упражнения – продемонстрировать различные функции деления, включая обычное деление, целочисленное деление и взятие остатка от деления. Пожалуйста, внимательно проанализируйте код и результаты, чтобы разобраться в том, как это работает. Если необходимо, вставьте другие сообщения для просмотра промежуточных результатов.

Упражнение 2 Приоритет операций

1 Вернитесь в программный модуль 95100 и добавьте после имеющихся строк следующие строки кода:

```
Int1 := 5 + 3 * 6 - 2 DIV -2;
MESSAGE('The value of %1 is %2','Int1',Int1);
```

2 Сохраните изменения, запустите программный модуль и убедитесь в том, что результат даст значение 24.

Следуя правилам приоритета операций, вычисления проводились таким образом.

- Выполнялась операция умножения, при которой левый операнд (3) умножался на правый операнд (6); в результате получился новый операнд (18), а все выражение приняло вид:
  `Int1 := 5 + 18 - 2 DIV –2`
- Выполнялась операция целочисленного деления (DIV), при которой левый операнд (2) делился на правый операнд (-2), в результате чего получился новый операнд (–1), а исходное выражение приняло вид:
  `Int1 := 5 + 18 – (-1)`
- Выполнялась операция сложения (+), при которой правый операнд (18) добавлялся к левому операнду (5), в результате чего получился новый операнд (23):
  `Int1 := 23 – (-1)`
- Наконец, выполнялась бинарная операция вычитания (-), при которой правый операнд (-1) вычитался из левого операнда (23), в результате чего в итоге получилось значение 24 (23 минус отрицательная 1 дает 24).

3 Теперь создадим подвыражения, добавив круглые скобки:

```
Int1 := (5 + 3) * (6 - 2) DIV -2;
```

4 Сохраним изменения и вновь запустим программный модуль. Вы увидите, что в результате получится отрицательное значение -16.

Исходя из того, что подвыражения вычисляются в первую очередь, проследим за тем, как велись вычисления в нашем примере.

- Вычислялось первое подвыражение. При выполнении операции сложения (+) второй операнд (3) добавлялся к первому (5), в результате значение первого подвыражения составило 8. Это значение стало

операндом в общем выражении.
```
Int1 := 8 * (6 - 2) DIV -2;
```

- Вычислялось второе подвыражение. При выполнении бинарной операции вычитания (-) второй ее операнд (2) вычитался из первого (6), в результате получилось значение 4. Опять-таки это значение стало операндом в общем выражении.
```
Int1 := 8 * 4 DIV -2;
```
- Выполнялась унарная операция вычитания (-) для операнда (2), в результате значение операции стало равно минус 2 (-2).
```
Int1 := 8 * 4 DIV -2;
```
- Выполнялась операция умножения (*) , при которой ее первый операнд (8) умножался на второй (4), что в результате привело к новому операнду 32.
```
Int1 := 32 DIV -2;
```
- Наконец, выполнялась операция целочисленного деления (DIV), при которой первый операнд (32) делился нацело на второй операнд (-2), что в итоге дало значение, равное минус 16 (-16).
```
Int1 := -16;
```

6.4 Контрольные вопросы

Для каждого из приведенных выражений напишите ниже тип результата вычисления выражения и получающееся значение:

1 57 * 10

2 57 / 10

3 57 DIV 10

4 57 MOD 10

5 2000000 * 3000

6 9 / 4 - 9 DIV 4

7 (3 - 10) * - 5 - 10 + 2.5 * 4

8 02201996D + 14

9 02101996D – 14

10 01201996D - 02101996D

Ответы

1	57 * 10	Integer	570
2	57 / 10	Decimal	5.7
3	57 DIV 10	Integer	5
4	57 MOD 10	Integer	7
5	2000000 * 3000	Decimal	6000000000
6	9 / 4 - 9 DIV 4	Decimal	0.25
7	(3 - 10) * - 5 - 10 + 2.5 * 4	Decimal	35
8	02201996D + 14	Date	03051996D
9	02101996D - 14	Date	01271996D
10	01201996D - 02101996D	Integer	-21

Глава 7

Логические и реляционные выражения

В данной главе представлены логические и реляционные выражения, доступные в языке C/AL операции, а также их приоритет выполнения.

Глава содержит следующие разделы:

- Логические и реляционные операции и выражения
- Реляционные выражения для сравнения
- Реляционные выражения для работы со множествами
- Использование логических выражений
- Добавление в форму логических и реляционных выражений
- Контрольные вопросы

7.1 Логические и реляционные операции и выражения

Логические (logical expressions) и реляционные (relational expressions) выражения отличаются от арифметических тем, что их результатом всегда будет булево значение (TRUE или FALSE). Чтобы вычислить свое значение, эти выражения используют логические или реляционные операции. Однако между ними существуют определенные различия, поэтому важно хорошо понимать их концепции.

Реляционная операция

Реляционная операция (relational operator) используется в реляционном выражении для проверки отношения между левым и правым операндами выражения, что в итоге дает результат типа Boolean. Реляционными операциями являются:

- = (равно)
- < (меньше)
- > (больше)
- <= (меньше или равно)
- >= (больше или равно)
- <> (не равно)
- IN (принадлежность множеству)

Реляционное выражение

Реляционное выражение (relational expression) – это выражение, которое с помощью реляционной операции (операции сравнения) сравнивает значения и формирует результат типа Boolean (TRUE или FALSE). Операнды в этом выражении обычно не булевы, но их типы должны быть совместимы. Таким образом, Integer может сравниваться с Decimal, поскольку оба типа числовые. В то же время Integer нельзя сравнивать с Text, так как первый тип числовой, а второй – строковый. Ниже приводятся некоторые примеры:

- 5 <= IntVar
- DecVar <> IntVar

Логическая операция

Логическая операция (logical operator) использует в логическом выражении один или два булевых операнда. К логическим операциям

относятся AND (“И”), OR (“ИЛИ”) и XOR (“исключающее ИЛИ”). Единственная логическая унарная операция – это NOT (“НЕ”).

Логическое выражение

Логическое выражение (logical expression) использует как минимум одну логическую операцию, которая дает булево значение. Оно схоже с реляционным выражением в том смысле, что имеет операнды любого типа, но реляционное выражение должно иметь два операнда и одну операцию. Логическое выражение может иметь множество операндов, но при этом все они должны быть типа Boolean. Ниже представлены примеры логических выражений:

- TRUE AND FALSE (результат: FALSE)
- FALSE OR NOT FALSE (результат: TRUE)
- (Quantity> 5) OR (Quantity <= 10) OR (Price < 100) (три операнда)

7.2 РЕЛЯЦИОННЫЕ ВЫРАЖЕНИЯ ДЛЯ СРАВНЕНИЯ

Все реляционные операции, за исключением IN, сравнивают два значения. Эти значения должны быть одинакового или совместимых типов. Все числовые типы (Integer и Decimal) совместимы между собой. Совместимы и оба строковых типа (Text и Code).

Сравнение чисел

Если сравнение производится между двумя числами, то применяются обычные правила. См. следующие примеры:

- 57 = 57 дает TRUE
- 57 = 58 дает FALSE
- 57 < 58 дает TRUE
- 57 <= 58 дает TRUE
- 57 > 58 дает FALSE
- 57 >= 57 дает TRUE
- 57 <> 58 дает TRUE

Сравнение строк

При сравнении строк используется модифицированный алфавитный порядок языка C/AL, а не просто порядок ASCII-кодировки, которую применяют другие языки программирования. Одно из основных отличий – это то, что специальные символы разных естественных языков (подобно букве é в датском языке и ñ в испанском) размещаются на своих нормальных позициях, а не в конце кодовой ASCII-таблицы. Другое важное отличие состоит в том, что цифры в алфавитном порядке идут после букв, тогда как в ASCII они идут перед буквами. Наконец, в C/AL считается, что буквы нижнего регистра идут перед буквами верхнего (в ASCII – наоборот). Ниже приводятся примеры:

- 'X' = 'X' дает TRUE
- 'X' = 'x' дает FALSE
- 'ark' > 'arc' дает TRUE
- 'arC' > 'arc' дает TRUE
- '10' > '2' дает FALSE
- '00' <= 'OO' дает FALSE
- 'é' > 'f' дает FALSE

- 'abc' < 'ab' дает FALSE
- ' a' <> 'a' дает TRUE

Помните, что когда в сравнении участвует переменная типа Code, то используются специальные правила: удаляются все предшествующие и замыкающие пробелы, а буквы переводятся в верхний регистр. Кроме того, кодовые значения, состоящие только из цифр, перед проведением сравнения выравниваются вправо. Т.е. в приведенном выше пятом примере ('10' > '2') значение выражения будет TRUE, если эти значения перед сравнением были загружены в переменные типа Code.

Сравнение дат и времени

Значения типа Date (дата) и Time (время) сравниваются так, как это и следовало ожидать. Дата (время) будущего больше, чем дата (время) прошлого. Дата закрытия (которая представляет собой последнюю секунду последней минуты указанной даты) больше соответствующей обычной даты (указанного дня) и меньше обычной даты следующего дня.

Сравнение булевых значений

С помощью реляционных операций булевы значения обычно не сравниваются, но такое возможно. Когда это происходит, то TRUE считается больше, чем FALSE.

Приоритет реляционной операции

Реляционные операции имеют самый низкий приоритет среди всех операций. Сравнение осуществляется после того, как вычислены выражения по обе стороны знака реляционной операции.

Соответственно, в следующем примере выражение примет значение TRUE:

```
5 * 7 < 6 * 6
```

Сначала вычисляется выражение, находящееся слева от знака операции сравнения (5 * 7); получается 35. Затем вычисляется выражение справа от знака операции (6 * 6) , что дает 36. И только потом сравниваются полученные значения 35 и 36.

7.3 РЕЛЯЦИОННЫЕ ВЫРАЖЕНИЯ ДЛЯ РАБОТЫ СО МНОЖЕСТВАМИ

В качестве реляционной операции принадлежности множеству используется операция IN. Она производит проверку того, принадлежит ли первый операнд данному множеству. Таким образом, данная операция помимо самого проверяемого элемента требует и наличия списка значений (множества). Этот список является частью выражения и называется константным множеством (set constant).

Константные множества

В языке C/AL нет переменных типа множество (set), но есть константы этого типа. Константное множество состоит из открывающей квадратной скобки ([), за которой идут список допустимых значений, которые разделены запятыми, и закрывающая квадратная скобка (]). Например, множество всех четных чисел от 1 до 10 выглядит так:

```
[2,4,6,8,10]
```

Отдельные значения в константном множестве могут быть представлены и диапазонами (range). Множество всех чисел от 1 до 20, которые не делятся на 10, записывается так:

```
[1..9,11..19]
```

Кроме того, в качестве отдельных значений (в том числе и в границах диапазона) могут использоваться выражения. Список чисел от 10 до 20, не включающих переменную “n” (если она лежит в диапазоне от 10 до 20), будет выглядеть так:

```
[10..n-1,n+1..20]
```

Операция IN

После того как вы построили константное множество, само действие операции IN достаточно простое. Она проверяет, принадлежит ли ее левый операнд множеству, указанному в правом операнде. Пример:

- 5 IN [2,4,6,8,10] равно FALSE
- 5 IN [2,4..6,8,10] равно TRUE
- 10 IN [1..9,11..19] равно FALSE
- 'M' IN ['A'..'Z'] равно TRUE

7.4 Использование логических выражений

Как уже указывалось ранее, логическая операция работает с булевыми операндами и возвращает булево значение. Нередко эти булевы операнды являются результатом реляционного выражения.

Результаты логической операции

Вот как работают логические операции.

- Унарная булева операция NOT (“НЕ”) является отрицанием идущего за ней операнда, изменяя значение TRUE на FALSE, а FALSE на TRUE.
- Операция AND (“И”) дает значение TRUE, если оба операнда имеют значение TRUE, в противном случае получается FALSE.
- Операция OR (“ИЛИ”) дает значение FALSE, если оба операнда имеют значение FALSE, в противном случае получается TRUE.
- Операция XOR (исключающее “ИЛИ”) дает значение TRUE, если только один из операндов (но не оба) имеет значение TRUE, но если оба операнда имеют значение TRUE или оба операнда имеют значение FALSE, то в результате будет FALSE.

Данные правила занесем в таблицы истинности:

NOT	**True**	**False**
Результат:	False	True

OR	**True**	**False**
True	True	True
False	True	False

AND	**True**	**False**
True	True	False
False	False	False

OR	**True**	**False**
True	False	True
False	True	False

Приоритет логический операций

Операция NOT имеет такой же приоритет, как и все другие унарные операции (+) и (-), т.е. наивысший приоритет. Это означает, что она выполняется в выражении раньше любой другой операции. Операция AND имеет такой же приоритет, как и мультипликативные операции, тогда как XOR и OR имеют приоритет такой же, как у аддитивных операций. Мы уже ранее указывали, что реляционные операции имеют самый низкий приоритет.

В следующей таблице представлена приоритетность всех уже известных нам операций:

Вид операции	Операция	Пояснение
подвыражение или операнды	( ) [ ] . ::	Подвыражение в круглых скобках вычисляется в первую очередь. Другие операции используются с операндами.
унарная	+ - NOT	самый высокий приоритет в выражении
мультипликативная	* / DIV MOD AND	
аддитивная	+ - OR XOR	
реляционная	< <= = >= > <> IN	самый низкий приоритет в выражении
диапазон	..	используется в константных множествах

Когда вы имеете дело с логическими выражениями, особенно важно помнить, что реляционные операции имеют более низкий приоритет, нежели логические операции. Таким образом, если вы хотите гарантировать, что N находится в диапазоне от 10 до 20, то вы не можете записать это в виде следующего выражения:

```
N >= 10 AND N <= 20
```

Если вы запишете так, то первой будет выполняться операция AND с целочисленными, а не булевыми, операндами (10) и (N). Это повлечет

за собой ошибку. Для корректности вам надо явным образом использовать для группирования операций круглые скобки:

```
(N >= 10) AND (N <= 20)
```

Теперь уже сначала вычисляются реляционные выражения, что дает в результате два булевых операнда. Ну а уже затем к ним применяется логическая операция AND.

7.5 Добавление в форму логических и реляционных выражений

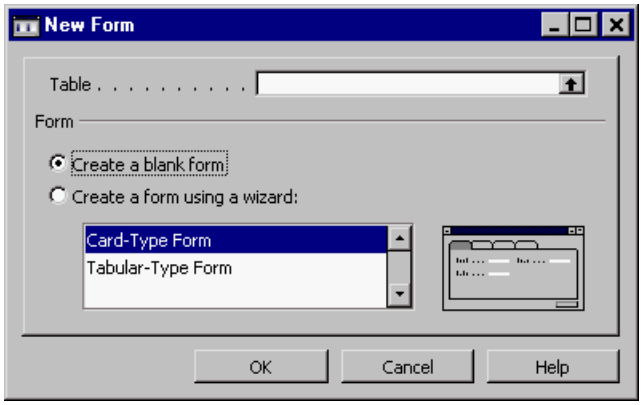
Ранее для вычисления выражений мы уже работали с простым программным модулем и в нем использовали функцию `MESSAGE`, позволяющую выводить результаты. Теперь, чтобы можно было протестировать побольше данных без перекомпиляции модуля, создадим для работы с ним соответствующую форму.

Создание новой формы

1 Откройте Object Designer путем выбора из строки меню пунктов Tools, Object Designer.

2 Чтобы просмотреть все формы, щелкните Form.

3 Щелкните New в нижней части окна Object Designer. Появится приведенное ниже окно **New Form**:



4 Щелкните OK. Появится окно с незаполненной формой. Теперь мы сохраним ее, чтобы она появлялась в окне Object Designer в качестве формы.

5 Закройте форму. Появится окно с вопросом "Do you want to save the changes to the Form"? (“Хотите ли вы сохранить внесенные в форму изменения?”) Щелкните кнопку Yes. Появится окно **Save As**.

6 Сохраните окно со следующими значениями свойств:

ID	Name
95100	Workbook Test Form

7 Чтобы сохранить форму, щелкните кнопку OK.

8 Прокрутите текст в окне Object Designer и найдите свою новую форму.

9 Выберите ее и щелкните Design. Появится окно с незаполненной формой.

Добавление в форму переменных

1 Щелкните View, C/AL Globals.

2 В окне **C/AL Globals** создайте две переменные типа Integer и назовите их Value1 и Value2.

3 Создайте переменную типа Boolean и дайте ей имя Result. Закройте окно C/AL Globals.

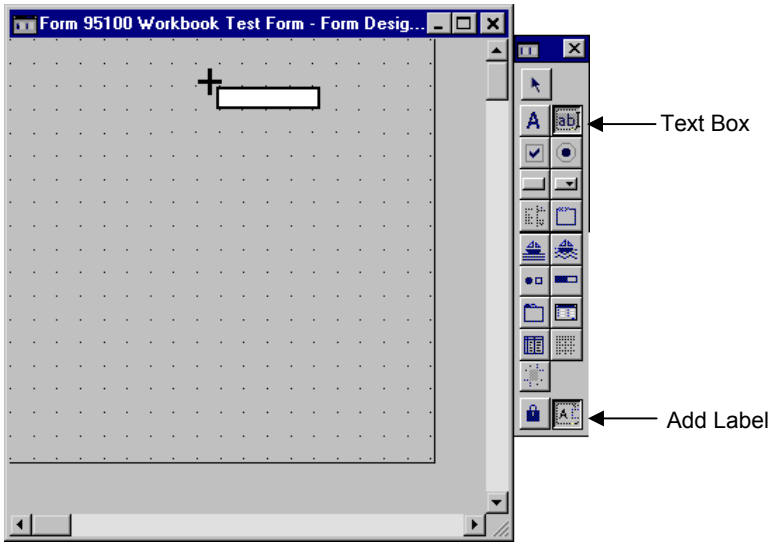
4 Чтобы открыть окно **Toolbox,** щелкните в строке инструментов кнопку Toolbox:

Toolbox

5 В правом нижнем углу выберите инструмент Add Label.

6 Выделите в правом столбце инструмент Text Box.

7 Перемещайте курсор, пока он не попадет на незаполненную форму (см. картинку):

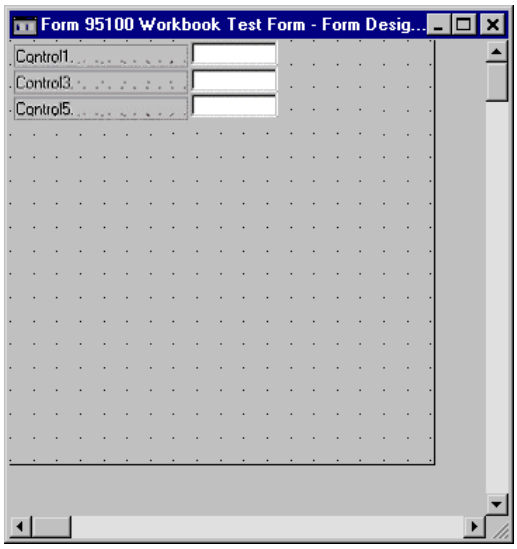


8 Курсор сменится на курсор позиционирования. Установите курсор так, чтобы перекрестье находилось по центру в верхней части незаполненной формы. Щелкните мышью, чтобы вставить текстовую зону. Появится текстовая зона с именем "Control" (белый прямоугольник), а слева от нее будет соответствующая надпись "Control" (серый прямоугольник).

9 Еще раз щелкните по инструменту Text Box, переместите курсор позиционирования в точку прямо под существующим элементом управления Text Box и еще раз щелкните мышью.

10 Наконец, вновь щелкните на инструмент Text Box, переместите курсор позиционирования немного ниже двух других текстовых зон и щелкните мышью.

В итоге вы должны увидеть форму такого вида:



Если элементы управления расположены неровно, вы можете поправить это, перемещая их с помощью мыши путем перетаскивания. Заметим, что когда вы перетаскиваете текстовую зону, вместе с ней перемещается и соответствующая надпись (серый прямоугольник).

11 Щелкните мышью в первой текстовой зоне, а затем в строке инструментов щелкните кнопку Properties.

Кнопка Properties расположена справа от кнопки Toolbox. Появится окно **Properties**, которое вы уже видели, но на этот раз в нем будет больше свойств. В данном примере нам нужны только два из них.

12 Найдите свойство Caption и измените его значение на *First Value*.

13 Найдите свойство SourceExpr и измените его значение на *Value1*. Теперь закройте окно **Properties**.

14 Щелкните во второй текстовой зоне, а затем щелкните кнопку Properties.

15 Измените значение свойства Caption на *Second Value*, а свойства SourceExpr на *Value2*. Наконец, щелкните на последней текстовой зоне. Появится окно **Properties**. Измените значение свойства Caption на *Result*, а значение свойства SourceExpr на *Result*.

16 Закройте форму, сохраните и запустите ее.

Заметим, что в работающей форме вы можете задать любое из трех значений. Первые два могут быть типа Integer, а вот третье ограничено значениями Yes или No (т.е. типа Boolean). Когда булево значение отображается в текстовом виде, то TRUE выводится как "Yes", а FALSE выводится как "No".

Добавление командной кнопки и C/AL-кода

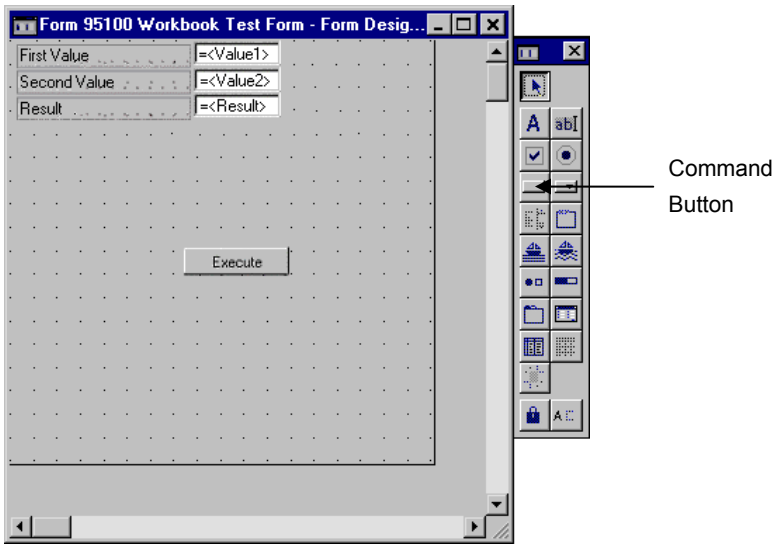
1 Откройте еще раз форму, щелкнув Design. Появится форма и Toolbox.

2 Щелкните по инструменту Add Label, чтобы снять с него выделение.

3 Теперь щелкните по инструменту Command Button. Переместите курсор позиционирования в середину формы и щелкните кнопку мыши.

4 Откройте окно **Properties** и измените значение свойства Caption на *Execute*.

После закрытия окна **Properties** экран будет выглядеть примерно так:



5 Щелкните по инструменту Command Button и нажмите на строке инструментов кнопку C/AL Code, либо нажмите клавишу F9.

6 Найдите триггер OnPush. Его код будет выполняться всякий раз, когда будет нажиматься командная кнопка. В качестве примера того, что вы можете сделать, введите в секцию кода триггера OnPush следующий оператор:

```
Result := Value1 > Value2;
```

7 Теперь закройте форму, сохраните и запустите ее.

8 Попробуйте ввести одно значение в поле First Value, а другое в поле Second Value, затем нажмите кнопку Execute. Поле Result автоматически обновится и в нем будет показан результат вычисления того выражения, которое фигурирует в операторе присваивания триггерного кода для OnPush. Попробуйте ввести еще несколько пар значений, в том числе и используя отрицательные числа.

Как только вы сможете уверенно предсказывать результат до нажатия кнопки Execute, разместите в триггере OnPush вместо ранее введенного следующий код:

```
Result := (Value1 >= Value2) AND (Value1 <= 2 *
Value2);
```

Попробуйте и другие формулы. Посмотрите, что еще вы можете с помощью этого примера узнать нового о том, как вычисляются выражения.

7.6 Контрольные вопросы

При подготовке ответов на представленные ниже вопросы можете смело использовать только что созданную тестовую форму. Помните, что каждое свойство SourceExpr определяет, какая переменная должна быть выведена на экран и что надо обновлять в рамках данного элемента управления. Около каждого выражения напишите его результат:

1 5 * 7 > 35

2 5 * -7 > -36

3 (3 > 5 - 1) OR (7 < 5 * 2)

4 (27 MOD 5 = 2) AND (27 DIV 5 = 5)

5 (5 > 3) XOR (7 = 7) AND (9 >= 10)

6 (10 > 2) AND ('10' > '2')

7 NOT (11 + 7 < 15) OR ('Great' > 'Greater') AND ('Less' < 'Lesser')

8 TRUE OR FALSE = TRUE

9 Пусть переменная X имеет тип Boolean:

NOT (X AND TRUE OR FALSE) = (NOT X OR FALSE) AND TRUE

Ответы вы можете увидеть на следующей странице.

Ответы

1 FALSE: 5 * 7 > 35

2 TRUE: 5 * -7 > -36

3 TRUE: (3 > 5 - 1) OR (7 < 5 * 2)

4 TRUE: (27 MOD 5 = 2) AND (27 DIV 5 = 5)

5 TRUE: (5 > 3) XOR (7 = 7) AND (9 >= 10)

6 FALSE: (10 > 2) AND ('10' > '2')

7 TRUE: NOT (11 + 7 < 15) OR ('Great' > 'Greater') AND ('Less' < 'Lesser')

8 TRUE: TRUE OR FALSE = TRUE

9 TRUE: NOT (X AND TRUE OR FALSE) = (NOT X OR FALSE) AND TRUE

Объяснение. Раз переменная X имеет тип Boolean, давайте рассмотрим все варианты.

Пусть X имеет значение TRUE:

NOT (TRUE AND TRUE OR FALSE) = (NOT TRUE OR FALSE) AND TRUE

NOT (TRUE OR FALSE) = (FALSE OR FALSE) AND TRUE

NOT TRUE = FALSE AND TRUE

FALSE = FALSE Да, это TRUE

Пусть X имеет значение FALSE:

NOT (FALSE AND TRUE OR FALSE) = (NOT FALSE OR FALSE) AND TRUE

NOT (FALSE OR FALSE) = (TRUE OR FALSE) AND TRUE

NOT FALSE = TRUE AND TRUE

TRUE = TRUE Да, это тоже TRUE

Глава 8

Операторы IF и EXIT

В данной главе мы рассмотрим два часто используемых оператора: IF и EXIT.

Глава содержит следующие разделы:

- Условные операторы и булевы выражения
- Оператор IF
- Оператор EXIT
- Ведение вычислений с помощью операторов IF и EXIT
- Контрольные вопросы

8.1 Условные операторы и булевы выражения

Чтобы все слушатели курса единоообразно понимали термины, дадим соответствующие определения.

Условный оператор

Условный оператор (conditional statement) осуществляет проверку условия и выполняет (в зависимости от его значения) один или несколько операторов. Оператор IF является наиболее активно используемым условным оператором, который для своего условия предусматривает только два значения True и False.

Булево выражение

Результатом булева выражения (Boolean expression) является булево значение: True или False. Это может быть булева переменная или константа, реляционное выражение (отношение) или логическое выражение. Заметим, что термины “булево выражение” и “логическое выражение” в большинстве случаев являются синонимами. Однако мы используем термин “булево выражение”, когда хотим подчеркнуть, что может быть использовано реляционное или другое логическое выражение.

8.2 Оператор IF

Оператор IF используется, когда вы хотите, чтобы конкретный оператор выполнялся только в том случае, если значение истинно (True). Он может быть использован и для выполнения другого оператора, который должен соответствовать ложному значению (False). Существуют два способа написания оператора IF. Применение того или иного способа зависит от ситуации.

Конструкция IF - THEN

Если только условие истинно, то для выполнения оператора применяется следующий синтаксис:

```
IF <булево выражение> THEN <оператор>
```

При вычислении булева выражения устанавливается условие True или False. Если значение выражения равно True, то выполняется оператор, идущий после зарезервированного слова THEN. Если же булево значение равно False, то оператор, идущий после зарезервированного слова THEN, не выполняется.

Например, если вы хотите проверить целочисленное значение, чтобы узнать, отрицательно оно или нет, то можете использовать следующий оператор IF:

```
IF Amount < 0 THEN
  Amount := -Amount;
```

Вычисляется реляционное выражение `Amount < 0`. Если оно истинно, то выполняется оператор присваивания, и Amount принимает положительное значение. Если же значение выражения равно False, то оператор присваивания не выполняется, и Amount сохраняет неотрицательное значение.

Обратите внимание, как был написан приведенный выше оператор IF. Тот оператор, который должен выполняться в случае истинности значения реляционного выражения, размещается строчкой ниже конструкции `IF – THEN` и с отступом вправо на два символа. Ваша программа будет работать и без этого отступа или даже с размещением на той же строке, что и IF. Следуя соглашению, мы запишем оператор IF на двух строках и с использованием отступа. Применение такого приема сделает ваш код более наглядным.

Конструкция IF-THEN-ELSE

Как мы уже говорили, существует два способа написания оператора

IF. Если требуется обеспечить выполнение одного оператора при истинности условия, а второго – при его ложности, то используется следующий синтаксис:

```
IF <булево выражение> THEN <оператор 1> ELSE <оператор
2>
```

Условие формируется булевым выражением. Если булево выражение равно True, то выполняется оператор, идущий после зарезервированного слова THEN. Если же булево выражение принимает значение False, то выполняется оператор, следующий за ELSE. В такой разновидности оператора IF будет выполняться только один из упомянутых операторов, но не оба сразу.

В частности, если вы хотите найти цену за единицу товара (UnitPrice), то можете разделить общую сумму (TotalPrice) на количество однородного товара (Quantity). Но если количество равно 0, то при делении возникнет ошибка. Чтобы избежать этого, надо сначала проверить значение переменной Quantity. И если количество будет равно нулю, вы сможете установить корректную цену за единицу товара. Таким образом, получаем следующий код:

```
IF Quantity <> 0 THEN
  UnitPrice := TotalPrice / Quantity
ELSE
  UnitPrice := 0;
```

Сначала вычисляется реляционное выражение Quantity <> 0. Если его значение равно True (количество товара не равно нулю), то общая сумма делится на количество и полученное значение присваивается переменной UnitPrice. Если же значение равно False (количество равно 0), то цена за единицу товара устанавливается равной нулю.

Еще раз обратите внимание на то, как написан оператор IF. Два оператора, которые могут быть выполнены, выделяются отступом в два пробела. При этом ключевое слово ELSE выравнивается по вертикали со словом IF. Несмотря на то, что программа будет работать так же без пробелов и новых строк, данный прием полезен, поскольку позволяет визуально определить, какое же условие приводит к выполнению ELSE-блока (т.е. отрицание условия при соответствующем ключевом слове IF).

Еще один момент, на который стоит обратить внимание, – после первого оператора присваивания нет точки с запятой. Это связано с тем, что он находится в середине оператора IF. Если бы мы поставили здесь точку с запятой (разделитель операторов), то система посчитала бы, что далее идет новый оператор. Однако за

присваиванием идет ключевое слово ELSE, и поскольку такого оператора не бывает (это неотъемлемая часть оператора IF), в результате возникнет ошибка компиляции.

8.3 Оператор EXIT

Как правило, код в триггере выполняется сверху вниз, а затем управление возвращается в тот объект, который и вызвал данный триггер, либо непосредственно пользователю. Если по каким-то причинам вы не хотите выполнять оставшуюся часть триггерного кода, то можете использовать оператор EXIT. Когда оператор EXIT выполняется, остаток триггерного кода игнорируется.

Оператор EXIT часто используется совместно с оператором IF, чтобы в случае определенных условий прерывать выполнение триггерного кода. Будем полагать, что в разобранной выше ситуации мы решили пропустить остаток кода, если Quantity равно нулю. Это можно записать в виде:

```
IF Quantity = 0 THEN
  EXIT;

UnitPrice := TotalPrice / Quantity;
```

Сначала вычисляется реляционное выражение Quantity = 0. Если оно истинно, то выполняется оператор EXIT, который пропускает оставшуюся часть триггерного кода. Если значение ложно, то оператор EXIT не выполняется, и потому следующим исполняемым оператором будет оператор присваивания, идущий за EXIT. Заметим, что этот оператор присваивания не выделен отступом, поскольку не является частью оператора IF.

8.4 Ведение вычислений с помощью операторов IF и EXIT

Мы собираемся расширить ту форму, которую создали в рамках последнего урока. Посмотрите еще раз предыдущую главу, если у вас остались какие-то вопросы по тому, как создавать элементы управления, перемещать их и задавать их свойства.

Добавление дополнительных переменных

Для данного примера нам потребуется добавить несколько переменных.

Войдите в Object Designer, выберите форму, которую вы создали в главе 7 (Form 95100) и щелкните Design.

В список текущих переменных добавьте следующие глобальные переменные. Это можно сделать, выбрав в строке меню пункты View, C/AL Globals:

Имя	Тип данных
Quantity	Integer
UnitPrice	Decimal
TotalSales	Decimal
TotalCredits	Decimal
GrandTotal	Decimal

Кроме того, измените тип данных для переменной Result с Boolean на Decimal.

Изменение существующих элементов управления

Имеющиеся элементы управления потребуют определенной модификации, так что они будут показывать нам значения двух новых переменных.

Закройте окно **C/AL Globals** и выберите первую текстовую зону. Она имеет налпись "First Value".

Откройте окно **Properties** и измените следующие свойства:

Свойство	Значение

Caption	Quantity
SourceExpr	Quantity

Щелкните (или “выберите”) вторую текстовую зону, около которой есть надпись "Second Value". Измените свойства как показано ниже:

Свойство	Значение
Caption	Unit Price
MinValue	0
SourceExpr	UnitPrice

Добавление новых элементов управления

Чтобы получить доступ к значениям остальных переменных и к двум командным кнопкам для проведения вычислений, нам потребуется три новых текстовых зоны.

Откройте окно **Toolbox**, нажав на строке инструментов кнопку Toolbox .

Выберите инструмент Add Label и добавьте в форму три дополнительные текстовые зоны с соответствующими надписями.

На каждой из трех новых текстовых зон установите следующие свойства.

Первая текстовая зона:

Свойство	Значение
Editable	No
Focusable	No
Caption	Total Sales
SourceExpr	TotalSales

Вторая текстовая зона:

Свойство	Значение

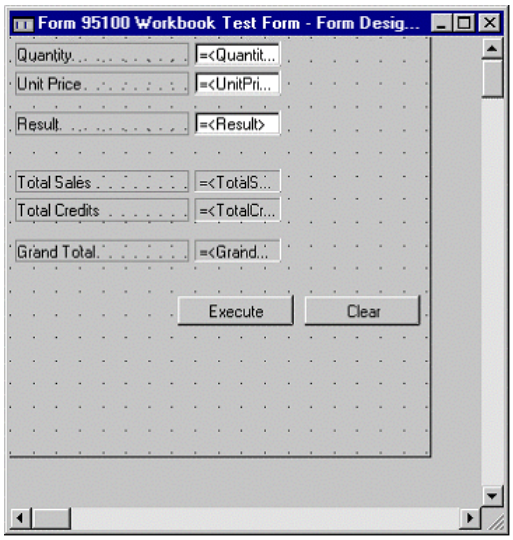
Editable	No
Focusable	No
Caption	Total Credits
SourceExpr	TotalCredits

Третья текстовая зона:

Свойство	Значение
Editable	No
Focusable	No
Caption	Grand Total
SourceExpr	GrandTotal

Снимите выделение с инструмента Add Label, выделите инструмент Command Button и добавьте в форму одну дополнительную командную кнопку. Установите значение ее свойства Caption в *Clear*.

Теперь, используя приемы, обсуждавшиеся в предыдущей главе, переместите текстовую зону командные кнопки так, чтобы форма выглядела примерно следующим образом:



Добавление в командные кнопки нового триггерного кода

Чтобы решить нашу задачу, нужно запрограммировать командные

кнопки.

В окне **Form Designer** выберите командную кнопку Execute и нажмите в строке инструментов кнопку C/AL Code.

Удалите из триггера OnPush имеющийся там код и замените его на следующий. Чтобы не набирать имена переменных, а выбирать их из списка, можно воспользоваться меню Symbol, для этого нужно нажать клавишу F5 (для более подробной информации см. главу 4):

```
IF Quantity = 0 THEN
  EXIT;

Result := Quantity * UnitPrice;

IF Result < 0 THEN
  TotalCredits := TotalCredits + Result
ELSE
  TotalSales := TotalSales + Result;

GrandTotal := GrandTotal + Result;
```

Три суммирующих переменных (TotalSales, TotalCredits и GrandTotal) обновляются с помощью стандартного приема программирования, который называется инкрементирование (*incrementing)*. Новое значение суммирующей переменной устанавливается равным текущему значению с увеличением на определенное число, называемое приращением (*increment)*. В рассмотренном выше коде новое значение переменной GrandTotal устанавливается равным текущему значению GrandTotal плюс значение переменной Result.

В ходе своего выполнения данный триггерный код делает следующее. Сначала он проверяет значение переменной Quantity. Если Quantity равно нулю, то оставшаяся часть триггера пропускается. В противном случае значение Result устанавливается равным произведению UnitPrice на Quantity. Потом проверяется значение Result. Если оно меньше нуля, то инкрементируется переменная TotalCredits. Если же значение Result больше или равно нулю, то инкрементируется переменная TotalSales. Переменная GrandTotal инкрементируется вне зависимости от значения переменной Result.

Выберите командную кнопку Clear и вновь откройте Редактор C/AL. Добавьте в триггер OnPush следующий код:

```
Quantity := 0;
UnitPrice := 0;
Result := 0;
TotalSales := 0;
```

```
TotalCredits := 0;
GrandTotal := 0;
```

Когда данный код выполнился, все переменные сбрасываются в ноль.

Настройка данных

Теперь нам надо проверить этот код.

Выйдите из Form Designer, сохраните изменения и запустите новую форму.

Заполните поля **Unit Price** и **Quantity**, а затем щелкните кнопку Execute. Теперь установите в поле **Quantity** другое значение и вновь щелкните Execute. Что произошло?

Введите в поле **Quantity** отрицательное значение (например, -4) и щелкните кнопку Execute. Что произошло на этот раз?

Что произошло, когда вы ввели отрицательное значение в поле **Unit Price**?

Почему это произошло?

Что произошло, когда вы нажали кнопку Clear?

Выполнение триггерного кода вручную

Если вы не понимаете, что происходит, когда вы нажали кнопку Execute, попробуйте исполнить код вручную. Чтобы сделать это, выпишите на листочке бумаги начальные значения переменных и для каждой строки кода выясните, когда она выполняется. После этого запишите ниже результирующие значения переменных.

Здесь приведен пример того, как вручную исполнить код. Заметим, что для данного примера код кнопки Execute уже неоднократно запускался, о чем свидетельствуют отличные от нуля значения переменных TotalSales, TotalCredits и GrandTotal.

```
IF Quantity = 0 THEN
  EXIT;
Result := Quantity * UnitPrice;
IF Result < 0 THEN
```

```
  TotalCredits := TotalCredits + Result
ELSE
  TotalSales := TotalSales + Result;
GrandTotal := GrandTotal + Result;
```

Шаг 1: Существующие значения до начала выполнения:

Quantity	UnitPrice	Result	TotalSales	TotalCredits	GrandTotal
-5	2.50	120.50	357.50	-7.00	350.50

Шаг 2: Проверка Quantity:

```
IF Quantity = 0 THEN
```

Quantity	UnitPrice	Result	TotalSales	TotalCredits	GrandTotal
-5	2.50	120.50	357.50	-7.00	350.50

Заметим, что ни одна переменная не изменялась, велась только проверка. В результате оператор EXIT не выполнялся.

Шаг 3: Вычисление Result:

```
Result := Quantity * UnitPrice;
```

Quantity	UnitPrice	Result	TotalSales	TotalCredits	GrandTotal
5	2.50	-62.50	357.50	-7.00	350.50

Новое значение Result отрицательное и равно -62.50.

Шаг 4: Проверка Result:

```
IF Result < 0 THEN
```

Quantity	UnitPrice	Result	TotalSales	TotalCredits	GrandTotal
-5	2.50	-62.50	357.50	-7.00	350.50

Опять-таки, в это время не менялась ни одна переменная. Проверялось значение переменной Result и выяснялось, какой следующий оператор надо выполнять. Поскольку Result меньше 0, то будет увеличено значение TotalCredits, тогда как инкрементирование переменной TotalSales выполняться не будет.

Шаг 5: Увеличение TotalCredits:

```
TotalCredits := TotalCredits + Result
```

Quantity	UnitPrice	Result	TotalSales	TotalCredits	GrandTotal
-5	2.50	-62.50	357.50	-69.50	350.50

Новое значение переменной TotalCredits равно ее текущему значению, увеличенному на значение Result.

Шаг 6: Увеличение GrandTotal:

```
GrandTotal := GrandTotal + Result;
```

Quantity	UnitPrice	Result	TotalSales	TotalCredits	GrandTotal
-5	2.50	-62.50	357.50	-69.50	288.00

Новое значение переменной GrandTotal равно ее текущему значению, увеличенному на значение Result.

Крестиком отмечены те строки, которые исполнялись в этом примере.

```
x     IF Quantity = 0 THEN
        EXIT;
x     Result := Quantity * UnitPrice;
x     IF Result < 0 THEN
x       TotalCredits := TotalCredits + Result
      ELSE
        TotalSales := TotalSales + Result;
x     GrandTotal := GrandTotal + Result;
```

8.5 Контрольные вопросы

Вопросы

Используя форму Workbook Test Form, которую вы создали в ходе этой и предыдущей глав, добавьте способность накапливать сумму по проданному объему товару, сумму по кредитуемому объему и общую сумму.

Опять-таки, с использованием данной формы добавьте два счетчика, которые увеличиваются на 1 всякий раз, когда вы щелкаете кнопку Execute, и сбрасываются в ноль, когда вы нажимаете кнопку Clear.

Один счетчик должен использоваться для подсчета объема продаж (Quantity > 0), а другой – для подсчета количества кредитов (Quantity < 0).

Кроме того, когда нажата кнопка Execute, используйте эти счетчики для вычисления средних значений проданного объема и кредитуемого объема.

Ответы вы можете увидеть на следующей странице.

Ответы

1 Добавьте способность накапливать сумму по проданному объему товару, сумму по кредитуемому объему и общую сумму.

Добавьте соответствующие переменные в список глобальных переменных (убедитесь, что все они, как и Quantity, принадлежат типу Integer). Добавьте в форму элементы управления для отображения значений этих переменных. Добавьте для кнопки Clear в конец существующего кода триггера OnPush следующий код:

```
TotalQtySold := 0;
TotalQtyCredited := 0;
GrandTotalQty := 0;
```

Используя имена переменных, которые были созданы, добавьте для кнопки Execute в конец существующего кода триггера OnPush следующий код:

```
IF Quantity < 0 THEN
  TotalQtyCredited := TotalQtyCredited + Quantity
ELSE
  TotalQtySold := TotalQtySold + Quantity;
GrandTotalQty := GrandTotalQty + Quantity;
```

2 Добавьте два счетчика, которые увеличиваются на 1 всякий раз, когда вы щелкаете кнопку Execute, и сбрасываются в ноль, когда вы нажимаете кнопку Clear. Один счетчик должен использоваться для подсчета объема продаж (Quantity > 0), а другой – для подсчета количества кредитов (Quantity < 0). Кроме того, когда нажата кнопка Execute, используйте эти счетчики для вычисления средних значений проданного объема и кредитуемого объема.

Создайте две дополнительные глобальные переменные (AvgQtySold, AvgQtyCredited). Добавьте в форму элементы управления для отображения значений этих переменных. Добавьте для кнопки Clear в конец существующего кода триггера OnPush следующий код:

```
SalesCounter := 0;
CreditCounter := 0;
AvgQtySold := 0;
AvgQtyCredited := 0;
```

Добавьте для кнопки Execute в конец существующего кода триггера OnPush следующий код:

```
IF Quantity < 0 THEN
  CreditCounter := CreditCounter + 1
ELSE
  SalesCounter := SalesCounter + 1;
IF SalesCounter <> 0 THEN
  AvgQtySold := TotalQtySold DIV SalesCounter;
IF CreditCounter <> 0 THEN
  AvgQtyCredited := TotalQtyCredited DIV
CreditCounter;
```

Заметим, что здесь используется операция DIV. Если бы использовалось обычное деление (/) и результат получил бы тип Decimal, то он не мог бы быть присвоен целочисленной переменной.

Глава 9

Комментарии и составные операторы

Данная глава посвящена использованию составных операторов для выполнения множества присваиваний. В ней поясняется, как использовать ключевые слова BEGIN и END, а также вложенные операторы IF. Использование приема отступа помогает сделать код более наглядным. В главе также разбираются вопросы добавления комментария в C/AL-код.

Глава содержит следующие разделы:

- Комментарии и составные операторы
- Синтаксис составных операторов с использованием BEGIN и END
- Составные операторы, использующие вложенные операторы IF
- Добавление комментариев
- Кодирование с помощью составных операторов и комментариев
- Контрольные вопросы

9.1 Комментарии и составные операторы

Ниже даются определения двух терминов, которые используются в этой главе.

Составной оператор

Составной оператор (compound statement) позволяет расширить возможности условного оператора. Помните, что условный оператор может выполнять только один оператор, если соответствующее условие истинно (True). Если имеется ELSE-блок, то в случае ложности условия (False) будет запускаться оператор из этого блока. Для выполнения более чем одного оператора в каждом из условных блоков нам понадобится использовать составной оператор.

Составной оператор состоит из нескольких операторов. Например, в случае оператора IF, если мы хотим выполнить два или более присваивания, когда условие равно True, то после ключевого слова THEN нам нужно разместить составной оператор. Как правило, после THEN разрешается использовать только один оператор. Но если этот оператор составной, то в нем могут идти сразу несколько операторов присваивания.

Комментарий

Комментарий (comment) – это описание того, что вы размещаете в своем коде, возможно, самого кода, либо указание того, кто и зачем вносил в код изменения. В то же время, поскольку это описание, а не машинная инструкция, мы не хотим, чтобы компилятор обрабатывал комментарий. Используя специальный синтаксис, мы даем знать компилятору, что в данном месте находится комментарий, который компилятору не надо пытаться транслировать, а нужно просто игнорировать.

9.2 Синтаксис составных операторов с использованием BEGIN и END

Чтобы использовать составные операторы, необходимо понимать их синтаксис.

Синтаксис составного оператора

Синтаксис составного оператора выглядит следующим образом:

```
BEGIN <оператор> {; <оператор>} END
```

Фигурные скобки ({ }) указывают на то, что заключенные в них синтаксические конструкции могут повторяться 0 или более раз. Другими словами, второго оператора может не быть вовсе, за первым могут идти два, пять – сколько требуется – операторов.

Составной оператор IF-THEN

С точки зрения синтаксиса, составной оператор может использоваться там же, где используется простой оператор. Например, если вы хотите сначала вычислить цену за единицу товара (unit price) исходя из суммарной цены (extended price), то добавьте суммарную цену к итоговой цене (total price), но только в том случае, если количество единиц товара не равно нулю. В этом случае оператор IF примет вид:

```
IF Quantity <> 0 THEN BEGIN
  UnitPrice := ExtendedPrice / Quantity;
  TotalPrice := TotalPrice + ExtendedPrice;
END;
```

Обратите внимание, что в составном операторе используется отступ. Операторы в составном операторе всегда имеют отступ в два пробела от позиций ключевых слов BEGIN и END, которые располагают на одной линии по вертикали. Если четко следовать этому правилу наряду с правилами отступа для операторов IF, то в результате получится:

```
IF Quantity <> 0 THEN
  BEGIN
    UnitPrice := ExtendedPrice / Quantity;
    TotalPrice := TotalPrice + ExtendedPrice;
  END;
```

Чтобы предотвратить двойной отступ (для IF и для BEGIN) и сохранить наглядность программного кода, применяют специальное правило для операторов IF. Ключевое слово BEGIN в операторе IF

располагается на той же самой линии, что и THEN, а ключевое слово END выравнивается по вертикали на начало строки, содержащей BEGIN.

Составной оператор IF-THEN-ELSE

Давайте добавим блок ELSE. Будем считать, что если Quantity равно нулю, то вы не только хотите установить цену за единицу товара в 0, но и также игнорировать выполнение оставшейся части триггера. С использованием "нормальных" правил отступа мы получили бы:

```
IF Quantity <> 0 THEN
  BEGIN
    UnitPrice := ExtendedPrice / Quantity;
    TotalPrice := TotalPrice + ExtendedPrice;
  END
ELSE
  BEGIN
    UnitPrice := 0;
    EXIT;
  END;
```

Однако для ELSE-блока есть специальные правила отступа. Чтобы сократить количество строк кода (видеть их на экране как можно больше за один раз) и в то же время сохранять наглядность кода, допускаются два изменения к правилам. Во-первых, ключевое слово ELSE появляется на той же строке, где и END от соответствующего IF. Во-вторых, ключевое слово BEGIN располагается на той же строке, где и ключевое слово ELSE. Таким образом, стандарт отступа, принятый в Navision для оператора IF, приводит к следующему расположению кода:

```
IF Quantity <> 0 THEN BEGIN
  UnitPrice := ExtendedPrice / Quantity;
  TotalPrice := TotalPrice + ExtendedPrice;
END ELSE BEGIN
  UnitPrice := 0;
  EXIT;
END;
```

9.3 Составные операторы, использующие вложенные операторы IF

В блоках THEN и ELSE оператора IF помимо присваивания может размещаться другой оператор IF. В таком случае второй оператор IF называется вложенным (nested IF statement). Это может затруднять понимание кода, особенно когда мы пытаемся разобраться, к какому из них относится ELSE-блок.

Конфликты при вложенных операторах IF

Правило состоит в том, что ELSE относится к ближайшему IF, который еще не имеет ELSE-блока. Здесь реально помочь в понимании кода могут правила отступа. Взгляните, например, на следующий фрагмент кода, в котором не используется отступ:

```
IF Amount <> 0 THEN
IF Amount > 0 THEN
Sales := Sales + Amount
ELSE
IF Reason = Reason::Return THEN
IF ReasonForReturn = ReasonForReturn::Defective THEN
Refund := Refund + Amount
ELSE
Credits := Credits + Amount
ELSE
Sales := Sales - Amount;
```

Без анализа кода трудно сказать, при каких условиях переменная Sales будет уменьшать свое значение на величину Amount. Даже хуже, предположим, что в приведенном фрагменте отступы были сделаны некорректно. Это может вводить в заблуждение:

```
IF Amount <> 0 THEN
  IF Amount > 0 THEN
    Sales := Sales + Amount
  ELSE
    IF Reason = Reason::Return THEN
      IF ReasonForReturn = ReasonForReturn::Defective
THEN
        Refund := Refund + Amount
    ELSE
      Credits := Credits + Amount
ELSE
  Sales := Sales - Amount;
```

Использование отступа для вложенного оператора IF

Применим, однако, то правило, которое гласит, что ELSE относится к

ближайшему IF, который еще не имеет ELSE-блока, а также воспользуемся стандартными правилами отступа, которые были приведены в этой главе. Тогда можно поправить код следующим образом:

```
IF Amount <> 0 THEN
  IF Amount > 0 THEN
    Sales := Sales + Amount
  ELSE
    IF Reason = Reason::Return THEN
      IF ReasonForReturn = ReasonForReturn::Defective
THEN
        Refund := Refund + Amount
      ELSE
        Credits := Credits + Amount
    ELSE
      Sales := Sales - Amount;
```

Заметим, что во всех этих примерах выполняемый код один и тот же, поскольку компилятор игнорирует лишние пробелы и пустые строки. Однако в последнем примере программисту будет гораздо легче разобраться в том, при каких условиях предпринимаются те или иные действия.

Кроме того, подчеркнем важность выбора значимых по смыслу имен для переменных. Допустим, что тот же самый код был корректно трансформирован с использованием отступов, но имена переменных были выбраны неудачно:

```
IF Amt1 <> 0 THEN
  IF Amt1 > 0 THEN
    Amt2 := Amt2 + Amt1
  ELSE
    IF OptA = 1 THEN
      IF OptB = 3 THEN
        Amt3 := Amt3 + Amt1
      ELSE
        Amt4 := Amt4 + Amt1
    ELSE
      Amt2 := Amt2 - Amt1;
```

Отличие между плохим и хорошим программированием нередко состоит в том, насколько хорошо ваш код самодокументирован. Используя правила отступа, наполненные смыслом имена переменных, оперируя булевыми переменными для обозначения выбора “да-нет” и вариантного типа для случаев более чем с двумя вариантами выбора, вы можете добиться того, что ваш код будет самодокументированным. Просто читая находящийся перед вами код, вы легко поймете, что он означает. Иногда все же полезно в код добавлять пояснения. Мы теперь вплотную подошли к предмету

изучения следующего раздела.

9.4 Добавление комментариев

Для комментариев существуют два различных синтаксических правила. Любая из этих разновидностей комментария может применяться в любом месте кода. В то же время в зависимости от ситуации у каждой из разновидностей есть свои преимущества перед другой.

Строчный комментарий

Первая разновидность – строчный комментарий (single line comment), иными словами, комментарий, который представлен в одной строке кода. Термин “закомментировать” (*to* comment out) означает, что весь фрагмент кода превращается в комментарий, а потому компилятором будет проигнорирован. Это достигается путем размещения на строке двух наклонных черт (//). Все, что идет после них на данной строке, считается комментарием и компилятором игнорируется. Ниже приведены примеры:

```
// Calculate the Unit Price
IF Quantity <> 0 THEN  // Do not allow division by zero
  UnitPrice := TotalPrice / Quantity
ELSE  // that is, if Quantity = 0
  UnitPrice := 0;
```

Первый комментарий – это пример такой строки программы, которая полностью отдана под комментарий. Следующие два показывают, что вы можете иметь на строке наряду с кодом еще и комментарий. В каждом из этих случае компилятор игнорирует все, что располагается на данной строке вслед за двумя наклонными чертами.

Блочный комментарий

Вторая синтаксическая разновидность комментария – блочный комментарий (block of comments). Он начинается с размещенной в коде открывающей фигурной скобки ({). Компилятор будет игнорировать все, что встретит вплоть до соответствующей закрывающей фигурной скобки (}). Он будет пропускать в этом случае даже пустые строки. Ниже приведен пример блочного комментария:

```
{Следующий код используется для вычисления цены за
единицу товара (Unit Price).  Заметим, что мы
сравниваем Quantity с нулем, чтобы избежать деления на
ноль. В такой ситуации мы присваиваем UnitPrice
значение 0.}
```

```
IF Quantity <> 0 THEN
  UnitPrice := TotalPrice / Quantity
ELSE
  UnitPrice := 0;
```

Вложенные комментарии

Один из типичных случаев использования блочного комментария – локализация проблемных фрагментов кода. Открывающая фигурная скобка вставляется в начале такого фрагмента, а закрывающая – в его конце. В результате весь фрагмент будет закомментирован. Этот пример позволит вам сосредоточиться на остальной части кода, либо просто изъять часть кода, как раз и приводящую к анализируемой проблеме.

Но как быть в той ситуации, когда во фрагменте, который нужно закомментировать, уже имеются фигурные скобки, говорящие о наличии комментария? Это не проблема.

Блочные комментарии могут быть вложенными. Когда компилятор встречает открывающую фигурную скобку, он все остальное будет трактовать как комментарий, пока не обнаружит соответствующую закрывающую скобку. Таким образом, в вашем коде для триггеров каждой открывающей скобке должна соответствовать закрывающая скобка. Ниже приведен пример вложенного комментария:

```
{Следующий код временно удален 8/15/01...

{Следующий код используется для вычисления цены за
единицу товара (Unit Price).  Заметим, что мы
сравниваем Quantity с нулем, чтобы избежать деления на
ноль. В такой ситуации мы присваиваем UnitPrice
значение 0.}

IF Quantity <> 0 THEN
  UnitPrice := TotalPrice / Quantity
ELSE
}
  UnitPrice := 0;
```

Заметим, что когда начнется выполнение данного фрагмента кода, исполнится только та строка, где UnitPrice присваивается значение 0. Все остальное будет проигнорировано.

9.5 Кодирование с помощью составных операторов и комментариев

После выполнения упражнений главы 8 и теста для самопроверки в той же главе триггер OnPush для кнопки Execute уже содержит определенную часть кода. Однако теперь его можно трансформировать, используя составные операторы.

Войдем в ту форму, которую мы создали в главах 7 и 8 (Form 95100), и подкорректируем код таким образом, чтобы максимально использовать возможности составных операторов. В результате получится что-то вроде следующего:

```
IF Quantity = 0 THEN
  EXIT;
Result := Quantity * UnitPrice;
IF Quantity < 0 THEN BEGIN
  TotalCredits := TotalCredits + Result;
  TotalQtyCredited := TotalQtyCredited + Quantity;
  CreditCounter := CreditCounter + 1;
  AvgQtyCredited := TotalQtyCredited DIV CreditCounter;
END ELSE BEGIN
  TotalSales := TotalSales + Result;
  TotalQtySold := TotalQtySold + Quantity;
  SalesCounter := SalesCounter + 1;
  AvgQtySold := TotalQtySold DIV SalesCounter;
END;
GrandTotal := GrandTotal + Result;
GrandTotalQty := GrandTotalQty + Quantity;
```

Поскольку этот код лаконичен и прост для понимания, программисту может и не потребоваться добавлять к нему комментарии. Однако, исключительно в качестве упражнения попробуйте вставить комментарии, как строчные, так и блочные. Откомпилируйте код, чтобы удостовериться в том, что вы все исправили корректно.

9.6 Контрольные вопросы

1 В следующей серии операторов соедините линиями ключевые слова ELSE с соответствующими им ключевыми словами IF:

```
IF (X1 > X2) OR B3 THEN BEGIN IF X7 < X2 THEN
A1 := (X1 + X7) / 2 ELSE IF X6 < X2 THEN
A1 := (X1 + X6) / 2;
X7 := X6 + 1; END ELSE IF (X1 < X2) AND B5 THEN
IF X6 > X7 THEN BEGIN
IF B2 THEN A2 := X1 / 2 + X7 / 2 ELSE
A2 := X1 * 2 + X7 * 2; END ELSE A1 := X1 ELSE
A2 := X2;
IF B1 THEN EXIT;
```

2 Каково будет значение переменной A5 после выполнения следующих операторов?

```
{Initialize Variables}
A5 := 7; // Initialize answer
B1 := {TRUE;} FALSE; B2 := TRUE; // FALSE;
A1 := { 5 // be sure to set this one correctly
A2 := 3 * A5;
A3 := 10 } 11; // either one is OK
A2 := 2 * A5;
// IF B2 THEN A5 := A5 + 1
IF (A1 < A5) OR {B2 AND} B1 THEN
A5 := 3 * A5 // ELSE A5 := A2 / A5;
ELSE A5 := A1 + A5;
```

Ответы

1 Здесь приведен тот же самый набор операторов, но уже с корректным отступом и линиями, показывающими связь между IF и ELSE.

```
IF (X1 > X2) OR B3 THEN BEGIN
  IF X7 < X2 THEN
    A1 := (X1 + X7) / 2
  ELSE
    IF X6 < X2 THEN
      A1 := (X1 + X6) / 2;
  X7 := X6 + 1;
END ELSE
  IF (X1 < X2) AND B5 THEN
  IF X6 > X7 THEN BEGIN
    IF B2 THEN
     A2 := X1 / 2 + X7 / 2
    ELSE
     A2 := X1 * 2 + X7 * 2;
  END ELSE
    A1 := X1
  ELSE
    A2 := X2;
IF B1 THEN
  EXIT;
```

2 Ниже представлены те же операторы, что фигурировали в вопросе, но с удаленными комментариями и корректным отступом.

```
A5 := 7;
B1 := FALSE;
B2 := TRUE;
A1 := 11;
A2 := 2 * A5;
IF (A1 < A5) OR B1 THEN
  A5 := 3 * A5
ELSE
  A5 := A1 + A5;
```

```
A5 = 18
```

Глава 10

Массивы

В данной главе рассматриваются массивы: что это такое, где и как их использовать в C/AL-коде.

Глава содержит следующие разделы:

- Определение массива
- Синтаксис
- Потенциал массивов
- Строки как массивы символов
- Использование массивов
- Контрольные вопросы

10.1 Определение массива

Массивы представляют собой специальные переменные, которые имеют большую функциональность, чем те переменные, что мы до этого обсуждали.

Простые переменные и типы данных

Простые типы данных (simple data types) рассматриваются в главе 2. Они определяются как типы данных, которые имеют простые (скалярные) значения. Простая переменная (simple variable) – это переменная, принадлежащая простому типу данных.

Составные переменные и типы данных

Составной тип данных (complex data type) – это тип с составными значениями. Составная переменная (complex variable) – это переменная, принадлежащая составному типу, т.е. имеющая составные значения.

Массив

Массив (array) – это составная переменная, которая фактически хранит в себе группу переменных. Эта группа в совокупности определяется одним идентификатором и одним типом данных. Например, мы можем создать переменную-массив со следующими идентификатором и типом данных:

- Идентификатор: QuantityArray
- Тип данных: Integer

Элемент

Элемент (element) представляет собой одну переменную в массиве. Массив состоит из одного или более элементов. И хотя допускается создавать массив, состоящий из одного элемента, это делается редко, так как в этом случае не будет преимуществ по сравнению с обычной переменной. Все элементы массива имеют один и тот же тип данных, вне зависимости от того, как массив был задан. При определении массива должно быть указано количество его элементов.

- Элементов: 5

Индекс

Индекс (index) используется для ссылки на отдельный элемент массива. Чтобы получить доступ к элементу массива, необходимо использовать имя переменной-массива и номер элемента (индекс). Чтобы получить доступ к четвертому элементу массива в приведенном примере, нужно написать:

- QuantityArray[4]

Размерность

Массив может обладать одним или несколькими измерениями (dimensions). Простейшим является одномерным массив (размерность по умолчанию). Такой массив содержит элементы только одного измерения. Его можно сравнить с элементами, отложенными по оси абсцисс (ось X).

В рассматриваемом примере у нас всего 5 элементов (одна ось, или измерение, содержащая 5 элементов).

Если мы определим массив с двумя измерениями, то он будет содержать 25 элементов (5 элементов в одном измерении и 5 элементов в другом). Будем представлять такой массив, как двумерный график, использующий оси абсцисс (x-axis) и ординат (y-axis). Каждая ось имеет 5 элементов (от 1 до 5), но при этом дает комбинацию 25 различных точек плоскости (с использованием только целых чисел).

Математическая формула, определяющая общее количество элементов массива, проста: это произведение количества элементов в каждом измерении. В языке C/AL массивы могут иметь до 10 измерений.

Чтобы использовать конкретный элемент в многомерном массиве, необходимо указать индекс для каждого измерения. Следовательно, количество индексов должно совпадать с количеством измерений. Помните, что эти значения должны быть целочисленными.

10.2 Синтаксис

Необходимо иметь в виду, что массив – это прежде всего переменная. Единственное отличие обращения к простой переменной от обращения к сложной состоит в добавлении индекса элемента.

Синтаксис переменной

Когда вы хотите ссылаться на любую переменную или массив как на одно целое, вы используете соответствующий идентификатор. Например, существует встроенная функция с именем CLEAR, которая имеет следующий синтаксис :

```
CLEAR(<переменная>)
```

Функция CLEAR очищает переменную, которую вы передали в качестве параметра. Если эта переменная числового типа, то функция CLEAR сбрасывает ее значение в 0. Если переменная строкового типа, то она принимает значение пустой строки. Если переменная представляет собой массив, то обнуляется каждый элемент массива. Таким образом, если у вас есть одномерный массив с именем SaleAmount, элементы которого имеют тип Decimal, то для сброса в 0 всех элементов SaleAmount используется следующий вызов функции.

```
CLEAR(SaleAmount);
```

Синтаксис элемента массива

Когда вы хотите сослаться на один элемент массива, вы должны использовать идентификатор массива и индекс элемента, соблюдая следующий синтаксис:

```
<идентификатор>[<индексное выражение>{,<индексное
выражение>}]
```

Квадратные скобки ([ ]) фигурируют в финальной форме записи элемента, тогда как фигурные ({ }) выполняют служебную роль и указывают на то, что заключенная внутри них последовательность может появляться ноль или более раз. Одномерный массив требует одного индексного выражения, а двумерный – уже двух, разделенных запятой. Каждое индексное выражение должно при его вычислении давать результат типа Integer. Если вы хотите пятому элементу массива SaleAmount присвоить значение 0, то надо использовать следующий оператор присваивания:

```
SaleAmount[5] := 0;
```

Как представлять себе массив

Самой простой способ представить себе массив – это рассматривать его как строку ячеек:

5	27	8	17	25	3	7	12
Box[1	Box[2	Box[3	Box[4	Box[5	Box[6	Box[7	Box[8

Здесь записан одномерный массив с именем Box, насчитывающий 8 элементов. Каждый элемент массива – это ячейка. Четвертый элемент массива содержит значение 17. Если мы хотим сослаться на четвертый элемент массива, то должны записать Box[4].

Столь же несложно представить себе и устройство двумерного массива, если рассматривать его подобием шахматной доски или таблицы умножения.

Box[1,c]	1	2	3	4	5	6
Box[2,c]	2	4	6	8	10	12
Box[3,c]	3	6	9	12	15	18
Box[4,c]	4	8	12	16	20	24
Box[5,c]	5	10	15	20	25	30
	Box[r,1]	Box[r,2]	Box[r,3]	Box[r,4]	Box[r,5]	Box[r,6]

Здесь записан двумерный массив с идентификатором Box, содержащий в общей сложности 30 элементов, которые разбиты на 5 строк по 6 элементов в каждой. Первое измерение может рассматриваться как номер строки, а второе – как номер столбца. Чтобы идентифицировать конкретный элемент, вы должны указать его строку и столбец. Например, если мы хотим обратиться к элементу, находящему на 4-й строке в 6-м столбце, то должны записать Box[4,6] (там содержится значение 24).

Как только число измерений возрастет до 3 и более, станет сложно визуально представлять такую структуру. К счастью, имеется всего несколько причин для использования в Navision массивов с размерностью более 1 (хотя допустимо иметь до 10 измерений). В остальной части главы мы ограничимся рассмотрением одномерных массивов.

10.3 Потенциал массивов

Если бы мы могли только обращаться к конкретному элементу массива …

```
SaleAmount[5]
```

….то пользы в этом было бы немного. Да, сформировав массив с 10 элементами, мы сэкономим на создании 10 разных переменных. Но, разумеется, с точки зрения разработчика, существует немало других полезных свойств массива.

Использование выражений в качестве индексов

Истинный потенциал массивов заключается в возможности использовать выражение в качестве индекса:

```
SaleAmount[Counter]
```

Это дает возможность определять индекс элемента в процессе выполнения программы. В приведенном примере элемент определяется значением Counter. Это весьма полезно как в контексте упражнения этой главы, так и в отношении следующей главы, где речь пойдет об операторах цикла.

В то же время это таит в себе определенную проблему. Как во время выполнения мы узнаем, допустимым ли при вычислении выражения будет индекс элемента? Если массив имеет 8 элементов, а мы будем обращаться к элементу с индексом 12, то возникнет ошибка этапа выполнения.

Чтобы ее избежать, часто используют оператор IF. Это позволяет проверять значение переменной до того, как использовать ее в роли индекса. Например:

```
IF Counter <= 8 THEN
  SaleAmount[Counter] := Answer;
```

Функция ARRAYLEN

Изложенный выше прием не идеален и грозит новой проблемой. Если мы впоследствии изменим размер массива с 8 на 7, то пока не возникнет ошибка этапа выполнения, не сможем узнать, что строка написана некорректно. Чтобы избежать этой неприятности, в языке C/AL предусмотрена встроенная функция ARRAYLEN, имеющая

следующий синтаксис:

```
Result := ARRAYLEN(<массив>)
```

Наличие левой части оператора присваивания "Result :=" сигнализирует о том, что функция ARRAYLEN возвращает значение (в данном случае типа Integer). Результатом будет количество элементов того массива, который передается функции в качестве параметра. Например, если у нас есть одномерный массив с 8 элементами, то функция будет возвращать целочисленное значение 8:

```
ARRAYLEN(SaleAmount)
```

Примечание. Функция ARRAYLEN может использоваться и с многомерными массивами. Если вы хотите узнать подробности этого, посмотрите интерактивную справку для данной функции в среде C/SIDE.

Преимущество использования функции ARRAYLEN в том, что если вы впоследствии переопределите размер массива, то не придется модифицировать программный код. Таким образом, в нашем первом примере мы могли бы проверять корректность индекса с помощью следующего кода:

```
IF Counter <= ARRAYLEN(SaleAmount) THEN
  SaleAmount[Counter] := Answer;
```

Поскольку здесь число 8 не используется, то ничто и не говорит о том, что в массиве ровно 8 элементов. Если размер массива будет изменен с 8 на 7, этот код по-прежнему будет корректным.

10.4 Строки как массивы символов

В некоторых случаях строковую переменную (типа Text или Code) удобно рассматривать как массив символов. Язык C/AL позволяет получать доступ к каждому символу как к элементу массива.

Символы как элементы

Каждый элемент рассматривается как переменная типа Char. В частности, если вы исполняете следующий код…

```
Str := 'Walk in the park';
Str[13] := 'd';
MESSAGE(Str);
```

…то в результате получится сообщение "Walk in the dark".

Отметим, что поскольку Char является числовым типом, вы можете при работе со строками подобным образом использовать ASCII-коды символов.

Например, если вы импортировали текстовую строку, то проверить ее на наличие символа табуляции можно так:

```
IF Str[idx] = 9 THEN
  MESSAGE('There is a TAB character at position %1 in
the text.',idx);
```

Заметим, что длина строки определяется особым образом. Вне зависимости от того, считываете ли вы конкретный элемент строки, либо его изменяете, длина строки остается прежней. Элементы за границами длины строки считаются неопределенными.

Если бы мы изменили в нашем первом примере 25-й элемент (а не 13-й) на `d`, то сообщение осталось бы прежним: "Walk in the park". Это следствие того, что 25-й символ расположен за границей длины строки.

Аналогичным образом, если бы индекс `idx` был бы больше длины строки `Str`, то мы могли бы найти там символ табуляции, но он бы не являлся частью строки `Str`.

Если вам требуется знать количество элементов строки, используемой как массив символов, то вы должны использовать функцию MAXSTRLEN. Для этой цели функция ARRAYLEN не подходит.

10.5 Использование массивов

Важно понимать, как создавать и использовать массивы в среде C/SIDE.

Создание переменной-массива

1 Откройте Object Designer, выберите форму, которую вы создали в предыдущих главах (Form 95100), и щелкните Design.

2 В текущий список переменных добавьте следующие глобальные переменные, выбрав на строке меню пункты View, C/AL Globals:

Имя	Тип данных
Counter	Integer
Amount	Decimal

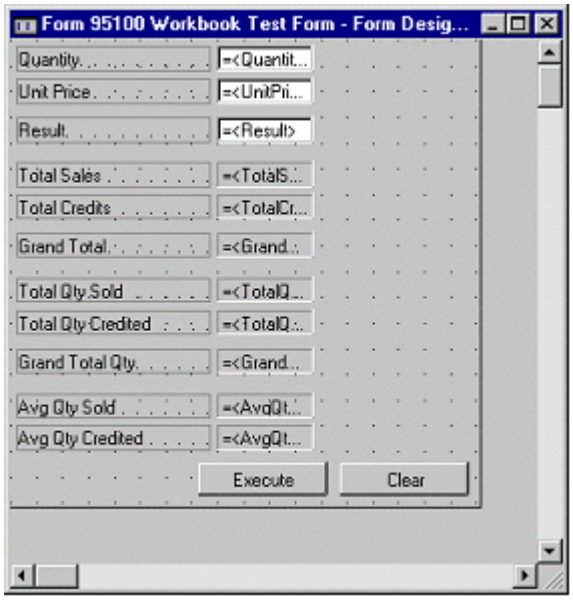
3 Установив курсор на той строке, где находится переменная Amount, откройте окно **Properties**, нажав на строке инструментов кнопку Properties.

4 Измените значение свойства Dimensions на *10*. Это означает, что Amount будет одномерным массивом (поскольку введено одно число) типа Decimal, максимальное значение индекса которого по первому (и единственному) измерению равно 10. Другими словами, массив будет содержать 10 элементов.

Примечание. Здесь мы этого не делаем, но если вы хотите создать двумерный массив, введите в качестве значения свойства Dimensions два числа, отделив их друг от друга точкой с запятой (;). Например, чтобы создать рассмотренный ранее массив Box, надо ввести в свойство Dimensions значения 5;7. Это приведет к созданию двумерного массива, максимальное значение индекса которого по первому измерению равно 5, а по второму 7, что в совокупности означает 35 элементов.

Добавление элементов управления для просмотра массива

1 Закройте окно **C/AL Globals**. Теперь ваша форма будет выглядеть приблизительно так:



2 Откройте окно **Toolbox**, нажав на строке инструментов кнопку Toolbox.

3 Проверьте, что инструмент Add Label не выделен и добавьте в форму текстовую зону (без надписи) справа от Result.

4 Задайте в новой текстовой зоне следующие значения свойств:

Свойство	Значение
Editable	No
Focusable	No
BlankZero	Yes
SourceExpr	Amount[1]

5 Убедитесь, что новый элемент управления по-прежнему выделен (если это не так, щелкните по нему) и щелкните на строке инструментов кнопку Copy:

6 На строке инструментов 9 раз щелкните кнопку Paste. Каждый новый элемент управления должен появляться под предыдущим. Всего в итоге будет 10 элементов управления.

7 Выделяйте по очереди каждый элемент управления, переходите в его свойства и изменяйте SourceExpr так, чтобы каждый элемент управления ссылался на соответствующий элемент массива Amount. Например, второй элемент управления должен относиться к Amount[2], третий – к Amount[3] и т.д., а десятый – к Amount[10].

К этому моменту ваша форма должна принять примерно такой вид:

Добавление кода для кнопки Clear

1 Выделите кнопку Clear и перейдите к ее триггерному коду OnPush. Давайте обнулим все ее переменные.

2 Вместо того чтобы обнулять каждый элемент отдельно, мы применим к всему массиву функцию CLEAR. Прибегайте к такому

приему не только для ради экономии программного кода (представьте, сколько бы мы выиграли места, если бы массив Amount содержал 1000 элементов). Этот прием удобен и тем, что позволяет нам не знать точного количества элементов массива.

3 Добавьте следующие две строки, чтобы обнулить переменные:

```
Counter := 0;
CLEAR(Amount);
```

Добавление кода для кнопки Execute

1 Выделите кнопку Execute и перейдите к коду ее триггера OnPush. В самом его конце добавьте следующие строки кода (с комментарием):

```
// Display results in Amount column on form
Counter := Counter + 1;
Amount[Counter] := Result;
```

Выполняются следующие действия.

- Вы первый раз щелкаете кнопку Execute, при этом переменная Counter увеличивается на 1, а элемент Amount[1] присваивает результат. Значение переменной Amount[1] отображается на экране в первом из добавленных нами элементов управления.

- Вы во второй раз щелкаете кнопку Execute, при этом переменная Counter увеличивается и принимает значение 2; результат присваивается элементу Amount[2]. Значение переменной Amount[2] отображается на экране во втором из добавленных нами элементов управления.

2 Сохраните и запустите форму.

3 В поля **Unit Price** и **Quantity** введите значения и после каждого ввода щелкните кнопку Execute. Что произошло?

4 Щелкните кнопку Clear. К чему это привело?

5 В поля **Unit Price** и **Quantity** введите новые значение и продолжайте нажимать кнопку Execute до тех пор, пока не заполнится столбец **Amount**. Затем еще один раз нажмите Execute. Теперь какой получился результат? Почему?

6 Вернитесь назад и исправьте возникшую проблему. Войдите в

код триггера OnPush кнопки Execute и поправьте последние строки так, чтобы они выглядели следующим образом:

```
// Display results in Amount column on form
Counter := Counter + 1;
IF Counter <= ARRAYLEN(Amount) THEN
  Amount[Counter] := Result;
```

7 Сохраните и запустите эту форму. Что происходит, когда вы нажимаете кнопку Execute более 10 раз?

Теперь мы устранили эту проблему. При использовании массивов нам нужно о ней не забывать.

10.6 Контрольные вопросы

Вопросы

Для каждого из перечисленных ниже примеров поясните, можно ли подобным образом решить задачу.

1 У нас есть список студентов, перенумерованных от 1 до 99. Каждому студенту дается тест. Мы хотим для каждого студента заносить в массив его оценку (от 0 до 100) так, что оценка студента с номером 1 будет находиться в элементе с номером 1, оценка студента с номером 2 будет находиться в элементе с номером 2 и т.д.

2 У нас есть список студентов, перенумерованных от 1 до 99. Каждому студенту дается тест. Мы хотим создать двумерный массив, содержащий две строки и 99 столбцов. В первую строку мы занесем соответствующую фамилию студента, т.е. фамилия первого студента будет находиться в элементе [1,1], фамилия второго студента – в элементе [1,2] и т.д. Во вторую строку мы будем заносить оценку студента (от 0 до 100) так, что оценка студента с номером 1 будет находиться в элементе [2,1], оценка студента с номером 2 будет находиться в элементе [2,2] и т.д.

3 Мы создаем массив, в котором содержится количество домов, приписанных каждому почтовому индексу (ZIP code). Для каждого 5-значного почтового кода выделяется один элемент массива, где и записано соответствующее число домов. Чтобы узнать количество домов, которые приписаны почтовому индексу 30071, мы обратимся к элементу 30071 и т.д.

4 Мы создаем массив, содержащий количество домов для каждого штата. Для каждого 2-символьного индекса, обозначающего штат США, выделяется отдельный элемент массива, и каждый элемент массива будет хранить число домов в этом штате. Чтобы узнать число домов в штате Джорджия (Georgia) нужно обратиться к элементу 'GA' и т.д.

Ответы

1 У нас есть список студентов, перенумерованных от 1 до 99. Каждому студенту дается тест. Мы хотим для каждого студента заносить в массив его оценку (от 0 до 100) так, что оценка студента с номером 1 будет находиться в элементе с номером 1, оценка студента с номером 2 будет находиться в элементе с номером 2 и т.д.

Да, можно.

2 У нас есть список студентов, перенумерованных от 1 до 99. Каждому студенту дается тест. Мы хотим создать двумерный массив, содержащий две строки и 99 столбцов. В первую строку мы занесем соответствующую фамилию студента, т.е. фамилия первого студента будет находиться в элементе [1,1], фамилия второго студента – в элементе [1,2] и т.д. Во вторую строку мы будем заносить оценку студента (от 0 до 100) так, что оценка студента с номером 1 будет находиться в элементе [2,1], оценка студента с номером 2 будет находиться в элементе [2,2] и т.д.

Нет, нельзя. Все элементы массива имеют один и тот же тип. В данном случае фамилия студента имеет тип Text, а его оценка – тип Integer. Для решения этой задачи надо использовать два отдельных массива.

3 Мы создаем массив, в котором содержится количество домов, приписанных каждому почтовому индексу (ZIP code). Для каждого 5-значного почтового кода выделяется один элемент массива, где и записано соответствующее число домов. Чтобы узнать количество домов, которые приписаны почтовому индексу 30071, мы обратимся к элементу 30071 и т.д.

Да, можно. Максимальное целое число превышает 2 миллиарда, тогда как максимальный почтовый индекс меньше 100 тысяч. Кроме того, максимальное крличество элементов в одном массиве составляет 1 миллион. Мы попадаем в эти пределы.

4 Мы создаем массив, содержащий количество домов для каждого штата. Для каждого 2-символьного индекса, обозначающего штат США, выделяется отдельный элемент массива, и каждый элемент массива будет хранить число домов в этом штате. Чтобы узнать число домов в штате Джорджия (Georgia) нужно обратиться к элементу 'GA' и т.д.

Нет, нельзя. Хотя элементы массива могут быть произвольного типа, индексы его должны быть целочисленными. В данном случае в роли индекса планировалось использовать тип Text или Code.

Глава 11

Операторы цикла

В данной главе представлены операторы цикла FOR, WHILE…DO и REPEAT…UNTIL.

Глава содержит следующие разделы:

- Операторы цикла
- Оператор FOR
- Оператор WHILE…DO
- Оператор REPEAT…UNTIL
- Кодирование с использованием операторов цикла
- Контрольные вопросы

11.1 Операторы цикла

Оператор цикла (repetitive statement) – это оператор, который позволяет выполнять много раз один или несколько других операторов. Существует несколько видов операторов цикла, и все они отличаются способом определения числа повторных выполнений заключенных внутри них операторов. Оператор цикла часто называют просто циклом (loop), поскольку как только выполнится последний вложенный оператор в этой серии, управление по циклу вернется на первый оператор серии.

В данной главе представлены три различных вида операторов цикла, а также их сходства и различия.

11.2 Оператор FOR

Оператор FOR используется в тех случаях, когда вложенные в него операторы должны быть выполнены фиксированное количество раз. Существует две разновидности оператора FOR.

Оператор FOR…TO

В большинстве случаев оператор вида FOR…TO имеет синтаксис:

```
FOR <переменная цикла> := <начальное значение> TO
<конечное значение> DO <оператор>
```

Переменная цикла должна быть переменной типа Boolean, Date, Time или любого числового типа. Начальное и конечное значения должны быть выражениями либо переменными того же типа данных, что и переменная цикла. Например:

```
FOR idx := 4 TO 8 DO
  Total := Total + 2.5;
```

В данном случае переменная цикла (`idx`), начальное значение (`4`) и конечное значение (`8`) принадлежат типу Integer. В приведенной ниже таблице описывается, что происходит.

1.	Вычисляется начальное значение, и оно присваивается переменной цикла.	idx := 4
2.	Вычисляется конечное значение (после ключевого слова TO).	Конечное значение равно 8
3.	Переменная цикла сравнивается с конечным значением. Если она превышает это значение, то оператор FOR завершается.	ЕСЛИ idx > 8, TO оператор FOR завершается.
4.	Выполняется оператор, следующий за ключевым словом DO.	Total := Total + 2.5
5.	Переменная цикла увеличивается на 1.	idx := idx + 1 (5)
6.	Происходит возврат к шагу 3.	

Совокупный результат здесь состоит в том, что переменная Total увеличивается на 2.5 в общей сложности 5 раз, т.е. ее приращение составляет 12.5. Заметим, что как начальное, так и конечное значение

вычисляются только один раз – в начале работы оператора FOR.

Обратите внимание, что переменная цикла не должна изменяться внутри самого цикла FOR, иначе это может привести к непредсказуемым результатам. Кроме того, значение переменной цикла за его пределами (после завершения цикла FOR) считается неопределенным.

Использование BEGIN и END

Как же быть, если нам необходимо внутри цикла запускать несколько операторов? Тогда мы будем использовать составной оператор, добавив ключевые слова BEGIN и END. Пример:

```
FOR idx := 4 TO 8 DO BEGIN
  Total := Total + 2.5;
  GrandTotal := GrandTotal + Total;
END;
```

Оператор FOR…DOWNTO

Другая разновидность оператора FOR (оператор FOR … DOWNTO) имеет следующий синтаксис:

```
FOR <переменная цикла> := <начальное значение> DOWNTO
<конечное значение> DO <оператор>
```

Правила для переменной цикла, начального и конечного значений те же самые, что и для первой разновидности цикла FOR. Единственное отличие состоит в том, что в операторе вида FOR TO переменная цикла последовательно увеличивает свое значение, тогда как в операторе вида FOR DOWNTO она его уменьшает. Ниже приведены пример и пояснение использования массива и составного оператора:

```
FOR idx := 9 DOWNTO 1 DO BEGIN
  TotalSales := TotalSales + Sales[idx];
  NumberSales := NumberSales + 1;
END;
```

1.	Вычисляется начальное значение, и оно присваивается переменной цикла.	idx := 9
2.	Вычисляется конечное значение (после ключевого слова TO).	Конечное значение равно 1
3.	Переменная цикла сравнивается с конечным значением. Если она меньше этого значения, то оператор FOR	ЕСЛИ idx < 1, TO оператор FOR

	завершается.	завершается.
4.	Выполняется оператор, следующий за ключевым словом DO.	TotalSales := TotalSales + Sales[9]; NumberSales := NumberSales + 1;
5.	Переменная цикла уменьшается на 1.	idx := idx - 1 (8)
6.	Происходит возврат к шагу 3.	

При каждом выполнении первого оператора, входящего в составной оператор, доступны разные элементы массива Sales. Сначала это элемент с индексом 9, затем с индексом 8 и т.д.

11.3 Оператор WHILE…DO

Цикл WHILE используется в тех случаях, когда определенное условие на протяжении всего цикла сохраняет значение TRUE. Ниже приведен синтаксис оператора WHILE:

```
WHILE <булево выражение> DO <оператор>
```

Данный цикл проще оператора FOR. Пока булево выражение имеет значение TRUE, в цикле выполняются операторы, заключенные между зарезервированными словами BEGIN и END. Как только булево выражение принимает значение FALSE, происходит выход из оператора WHILE и передача управления на первый идущий за ним оператор.

Ниже представлен пример, демонстрирующий применение составного оператора:

```
WHILE Sales[idx + 1] <> 0 DO BEGIN
  idx := idx + 1;
  TotalSales := TotalSales + Sales[idx];
END;
```

Вот как он выполняется (см. таблицу).

1.	Вычисляется булево выражение. Если оно истинно, переходим к шагу 2. Если нет, то оператор WHILE завершается.	Sales[idx + 1] <> 0?
2.	Выполняется оператор, идущий после зарезервированного слова DO.	idx := idx + 1; TotalSales := TotalSales + Sales[idx];
3.	Возврат к шагу 1.	

Заметим, что булево выражение проверяется каждый раз до начала выполнения очередного цикла. Если оно ложно с самого начала, то цикл никогда не выполняется.

Заметим также, что элемент Sales[idx], который добавляется к значению Total Sales внутри цикла WHILE, представляет собой то же самое значение, что проверяется в начале каждого цикла. Производится упреждающее сравнение следующего элемента (idx + 1). Как только цикл WHILE завершен, индекс idx по-прежнему указывает на последний ненулевой элемент массива Sales.

11.4 Оператор REPEAT…UNTIL

Оператор REPEAT используется, когда один или более операторов должны выполняться до тех пор, пока некоторое условие не станет истинно. Далее приведен синтаксис оператора REPEAT:

```
REPEAT <оператор> { ; <оператор> } UNTIL <булево
выражение>
```

Между операторами REPEAT и WHILE существуют следующие отличия.

- Во-первых, между ключевыми словами REPEAT и UNTIL может быть заключено несколько операторов, при этом не используются ключевые слова BEGIN и END.
- Во-вторых, булево выражение вычисляется в конце каждого цикла, т.е. операторы внутри REPEAT будут выполнены хотя бы один раз.
- В-третьих, цикл REPEAT завершается, когда булево выражение становится равным TRUE, а не FALSE (как в случае WHILE).

Ниже дан пример использования цикла REPEAT, который почти идентичен приведенному выше циклу WHILE:

```
REPEAT
  idx := idx + 1;
  TotalSales := TotalSales + Sales[idx];
UNTIL Sales[idx] = 0;
```

Вот как ведется выполнение.

1.	Выполняются операторы, заключенные между ключевыми словами REPEAT и UNTIL.	idx := idx + 1; TotalSales := TotalSales + Sales[idx];
2.	Вычисляется булево выражение. Если оно истинно, то цикл REPEAT завершается. В противном случае возврат к шагу 1.	Sales[idx] = 0?

Заметим, что поскольку булево выражение не вычисляется до конца цикла, то увеличение индекса и приращение TotalSales происходит даже в том случае, если значение Sales с самого начала равно 0. Следовательно, в конце этого цикла переменная idx будет ссылаться на первый элемент массива Sales, который равен 0, а не на последний

ненулевой элемент массива Sales (как в случае WHILE).

Кроме того, обратите внимание, что булево выражение должно быть переписано, поскольку для продолжения цикла REPEAT требуется, чтобы оно принимало значение FALSE, а не TRUE (как в случае WHILE).

11.5 Кодирование с использованием операторов цикла

В нашей тестовой форме, созданной в предыдущей главе, показывались первые 10 элементов списка. Мы изменим ее так, чтобы всегда показывались последние 10 элементов, которые добавлены в список.

Добавление цикла FOR

1 Войдите в Object Designer, выберите форму 95100 и щелкните Design.

2 Добавьте к текущему списку переменных следующую глобальную переменную, выбрав в строке меню пункты View, C/AL Globals:

```
idx            Integer
```

3 Выделите кнопку Execute и получите доступ к коду ее триггера OnPush. Найдите строку, на которой стоит комментарий "Display results in Amount column on form". Внесите исправления так, чтобы получился следующий код:

```
// Display results in Amount column on form
Counter := Counter + 1;
IF Counter > ARRAYLEN(Amount) THEN BEGIN
  Counter := ARRAYLEN(Amount);
  FOR idx := 2 TO Counter DO
    Amount[idx-1] := Amount[idx];
END;
Amount[Counter] := Result;
```

Рассмотрим, что здесь делается. Как только переменная Counter становится больше длины массива Amount, мы присваиваем ей длину массива. Затем с помощью цикла FOR мы смещаем элементы массива на один индекс к его началу. Элемент 2 становится элементом 1, элемент 3 становится элементом 2 и т.д. Наконец, элемент 10 смещается на индекс 9, и цикл FOR завершается. Новый результат записывается в последний элемент массива.

4 Теперь сохраните и запустите форму.

5 Введите некоторые значения и нажмите кнопку Execute. Посмотрите, что происходит. Пока список не будет заполнен, ничего не изменится, но посмотрите, что произойдет, если вы нажмете кнопку Execute более 10 раз.

Сортировка. Этап 1

1 Вновь откройте Form Designer и добавьте новую командную кнопку, установив ее свойству Caption значение *Sort*. Если вам нужна помощь в том, как это сделать, обратитесь к упражнению из главы 7.

2 Чтобы найти триггер OnPush и добавить следующий код, выделите кнопку и нажмите клавишу F9:

```
FOR idx := ARRAYLEN(Amount) DOWNTO 2 DO
  IF Amount[idx] < Amount[idx-1] THEN BEGIN
    TempAmount := Amount[idx];
    Amount[idx] := Amount[idx-1];
    Amount[idx-1] := TempAmount;
  END;
```

3 Перед компиляцией необходимо добавить переменную TempAmount типа Decimal.

4 Выйдите, откомпилируйте, сохраните и запустите эту форму.

5 Заполните поля и щелкните кнопку Execute, вводя различные количества всякий раз, пока столбец не заполнится.

6 Щелкните кнопку Sort. Что произошло?

7 Еще раз щелкните кнопку Sort. Что произошло на этот раз?

8 Продолжайте нажимать кнопку Sort до тех пор, пока перестанут происходить изменения. Заметим, что к тому моменту все записи будут отсортированы. Теперь взгляните на приведенный выше код и ответьте на следующие вопросы. Если необходимо, попробуйте его выполнить вручную.

Почему в операторе FOR мы используем конечное значение 2, а не 1?

Почему после DO в операторе FOR нам не нужно использовать ключевое слово BEGIN?

Что выполняется на трех строках составного оператора (между BEGIN и END)?

Почему для этого используется переменная TempAmount?

Если записи в столбце размещены в неверном порядке (отсортированы в порядке убывания), то сколько раз потребуется нажать кнопку Sort, чтобы получить нужный порядок (по возрастанию)?

Сортировка. Этап 2

Очевидно, что было бы лучше, если бы пользователь мог щелкнуть кнопку Sort всего один раз, после чего список стал полностью отсортированным. Заметим, однако, что иногда для этого достаточно одного нажатия, а в других случаях требуется нажимать кнопку Sort неоднократно. Цикл FOR для этой цели не подходит, поскольку мы в точности не знаем, сколько раз надо выполнять подобный цикл.

Проверим, насколько для нашей задачи подходит цикл REPEAT. Добавьте новую переменную IsSorted типа Boolean.

Затем внесите изменения в триггерный код OnPush для кнопки Sort:

```
REPEAT
  IsSorted := TRUE;
  FOR idx := ARRAYLEN(Amount) DOWNTO 2 DO
    IF Amount[idx] < Amount[idx-1] THEN BEGIN
      TempAmount := Amount[idx];
      Amount[idx] := Amount[idx-1];
      Amount[idx-1] := TempAmount;
      IsSorted := FALSE;
    END;
UNTIL IsSorted;
```

Заметим, что переменная IsSorted используется в качестве признака завершения процесса сортировки. В начале каждого цикла REPEAT мы присваиваем ей значение и затем производим обмен соседних элементов массива, присваивая ей значение FALSE.

В конце цикла REPEAT мы проверяем, равна ли эта переменная TRUE. Если да, то твердо знаем, что все элементы упорядочены. И поскольку нет необходимости в дальнейшей сортировке, мы выходим из цикла. Если же эта переменная равна FALSE, цикл повторяется

снова. Когда булева переменная используется в качестве признака завершения, программисты называют ее флагом (flag).

Обратите внимание, что мы не можем сказать, отсортирован ли список, пока хотя бы раз не выполним цикл. Вот почему мы использовали цикл REPEAT, а не WHILE.

Сохраните и запустите эту форму. Проведите эксперимент, подобный тому, что мы делали ранее, заполняя таблицу и нажимая кнопку Sort. Обратите внимание, что список будет отсортирован за одно нажатие кнопки.

Посмотрите еще раз на оператор IF. Для сравнения элементов мы использовали операцию “меньше” (<).

Что произойдет, если вы измените ее на операцию “больше” (>)?

Что произойдет, если вы измените ее на операцию “меньше или равно” (<=)? Если не уверены в ответе, проведите эксперимент с программным кодом.

Проверьте вариант, когда имеются два элемента с одинаковыми значениями. Для следующего упражнения измените операцию на “меньше”.

Сортировка. Этап 3

Теперь у нас есть достаточно хорошая процедура сортировки. Для небольших массивов она работает нормально. Мы написали ее так, что размер массива может быть любым. Что будет в случае, если массив будет содержать 100 элементов? Ну а если 1000 элементов или даже 10000 элементов? Когда мы поместили один цикл (FOR) внутрь другого цикла REPEAT, то создали структуру со вложенными циклами. Когда используются вложенные циклы, есть опасность снижения эффективности, особенно при увеличении числа элементов. Вопрос в том, сколько раз будет выполняться внутренний цикл.

Всегда, когда цикл FOR выполняется один раз, оператор IF выполняется 9 раз, поскольку длина массива равна 10, а мы не собираемся перемещать за границы массива его первый элемент. Если цикл REPEAT выполнится 9 раз, то и цикл FOR выполнится 9 раз, а оператор IF выполнится 9 * 9 = 81 раз. Ну а если длина массива равна 100, то оператор IF выполнится 99 * FOR раз, а цикл FOR будет выполняться в цикле REPEAT до 99 раз. В такой ситуации оператор IF может выполниться до 99 * 99 = 9801 раз!

Мы можем значительно сократить количество циклов, если обратим внимание на то, что произошло, когда впервые сами написали цикл FOR. Всякий раз, когда мы его запускали, самый младший элемент поднимался вверх. Если вы вдумаетесь, то обнаружите, что первый элемент после первого запуска FOR никогда больше проверяться не будет. Как только цикл FOR выполнится второй раз, вы уже никогда не будете проверять первые два элемента. Каждый последующий цикл сокращает количество элементов, подлежащих сравнению. И это в самом худшем случае. Но ситуация может сложиться и так, что один раз нажав кнопку Sort, вы сможете упорядочить первые три элемента, и нам их уже не понадобится сравнивать!

Ниже показано, как мы можем использовать преимущества подобного приема. Но сначала создайте целочисленную переменную LowestSwitch.

Теперь перепишем процедуру сортировки так:

```
LowestSwitch := 2;
REPEAT
  IsSorted := TRUE;
  FOR idx := ARRAYLEN(Amount) DOWNTO LowestSwitch DO
    IF Amount[idx] < Amount[idx-1] THEN BEGIN
      TempAmount := Amount[idx];
      Amount[idx] := Amount[idx-1];
      Amount[idx-1] := TempAmount;
      IsSorted := FALSE;
      LowestSwitch := idx + 1;
    END;
UNTIL IsSorted;
```

Вместо того, чтобы каждый раз вести сортировку до индекса 2 (сравните с 1), теперь цикл FOR будет вести сортировку до индекса LowestSwitch (сравните с LowestSwitch - 1). Переменная LowestSwitch начинает со значения 2, но как только происходит обмен местами соседних элементов массива (в THEN-блоке оператора IF), она увеличивает свое значение на 1. Поскольку цикл FOR перебирает элементы снизу вверх (от старших индексов к младшим), то переменная LowestSwitch будет в следующий раз самым верхним из просматриваемых элементов. Даже в самом худшем случае она будет на 1 больше, чем предыдущее значение.

Например, если массив имеет 10 элементов, то если раньше в самом худшем случае вложенный оператор IF мы выполняли бы 9 * 9 = 81 раз, то теперь он будет выполняться максимум 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 45 раз. Если в массиве 100 элементов, то в худшем случае он будет выполняться 4950 раз вместо 9801.

Другими словами, такая простая модификация приводит к двукратному ускорению сортировки.

11.6 Контрольные вопросы

Вопросы

1 Какой оператор цикла надо использовать, если вы до его выполнения знаете, сколько раз он должен выполниться?

2 Какой оператор цикла не требует ключевых слов BEGIN и END для заключения внутри себя составного оператора?

3 Какие операторы проверяют условие в начале каждого цикла?

4 Замените следующий оператор WHILE на эквивалентный оператор REPEAT. Опишите, в чем различия при их выполнении.

```
WHILE X > 0 DO BEGIN
  Y := A * X * X + B * X + C;
  X := X - 1;
END;
```

Ответы см. на следующей странице.

Ответы

1 Какой оператор цикла надо использовать, если вы до его выполнения знаете, сколько раз он должен выполниться?

Оператор FOR

2 Какой оператор цикла не требует ключевых слов BEGIN и END для заключения внутри себя составного оператора?

Оператор REPEAT

3 Какие операторы проверяют условие в начале каждого цикла?

Операторы FOR и WHILE

4 Замените следующий оператор WHILE на эквивалентный оператор REPEAT. Опишите, в чем различия при их выполнении.

```
WHILE X > 0 DO BEGIN
  Y := A * X * X + B * X + C;
  X := X - 1;
END;

REPEAT
  Y := A * X * X + B * X + C;
  X := X - 1;
UNTIL X <= 0;
```

Отличие состоит в том, что если переменная X с самого начала меньше или равна 0, то в случае оператора WHILE не будет выполнен ни один вложенный в него оператор, тогда как в случае оператора REPEAT цикл выполнится ровно один раз.

Если же X с самого начала больше 0, то между WHILE и REPEAT не будет никаких различий.

Глава 12

Другие операторы

В данной главе мы знакомим вас с двумя операторами, WITH и CASE. В соответствующих разделах главы будут представлены примеры, иллюстрирующие эти операторы. Контрольных вопросов в данной главе не будет.

- Оператор WITH
- Оператор CASE

12.1 Оператор WITH

Оператор WITH используется для упрощения работы с переменными типа запись. Поэтому прежде чем перейти к рассмотрению особенностей оператора WITH, нам необходимо разобраться в том, что же такое запись.

Записи

Запись (record) – это составной тип данных (см. главу о массивах). Подобно массиву (array), запись содержит несколько значений. В случае массива все значения обладают одним и тем же именем и типом, а различаются лишь номером элемента, или индексом.

В случае записи мы имеем различные значения, также называемые полями (fields), каждое из которых имеет собственные тип и имя. Чтобы различать эти значения в качестве разделителя имени записи и имени поля используется точка (.). Таким образом, если запись, Customer имеет поле Name, то в программном коде доступ к имени покупателя выглядит как Customer.Name.

Записи представляют одну строку, или запись, в таблице базы данных. Поля, образующие запись, задаются с использованием Table Object. То, как это делается, раскрыто в курсе программирования Navision.

Синтаксис оператора WITH

Предположим, что у вас имеются в массиве некоторые данные, подлежащие загрузке в переменную типа запись. Реализующий это код может выглядеть примерно так:

```
Customer.Name := Txt[1];
Customer.Address := Txt[2];
Customer.City := Txt[3];
Customer.Contact := Txt[4];
Customer."Phone No." := Txt[5];
```

Чтобы упростить кодирование и сделать данный код более наглядным, используется оператор WITH. Синтаксис оператора WITH:

```
WITH <запись> DO <оператор>
```

Внутри оператора (который может быть и составным) имя переменной типа запись не требуется. Вместо этого могут напрямую использоваться поля этой записи, которые можно рассматривать как

самостоятельные переменные.

Таким образом, приведенный выше пример можно переписать так:

```
WITH Customer DO BEGIN
  Name := Txt[1];
  Address := Txt[2];
  City := Txt[3];
  Contact := Txt[4];
  "Phone No." := Txt[5];
END;
```

Обратите внимание, что имя записи больше не требуется. После компиляции компьютер будет знать, что эти поля относятся к записи Customer. Однако, если имя переменной будет совпадать с именем поля, то как C/SIDE сможет разобраться, на что ссылается идентификатор? В этом случае он будет отдавать предпочтение полю, а не переменной, хотя подобных коллизий желательно избегать.

Неявные операторы WITH

Во многих важных местах, где используется оператор WITH, вы даже не видите этот оператор. Он выступает в неявной форме. Неявные операторы WITH можно встретить в разных объектах.

Например, в Table Object существует неявный оператор WITH, охватывающий весь объект. Каждая ссылка на поле, описанное в этой таблице, имеет неявную переменную типа запись (называемую "Rec") а точка автоматически и невидимо для глаз размещается перед именем поля.

В рамках курса программирования для Navision вы узнаете о других ситуациях, где встречаются неявные операторы WITH.

12.2 Оператор CASE

Подобно оператору IF оператор CASE является условным оператором. Оператор IF используется, когда условие имеет только два значения True (“истина”) или False (“ложь”). Если условие истинно (True), то выполняется оператор, идущий за ключевым словом THEN. Если условие ложно (False), то выполняется оператор, следующий за ключевым словом ELSE.

Синтаксис оператора Case

Оператор CASE используется в тех ситуациях, когда для условия имеется более двух возможных значений. Синтаксис его выглядит так:

```
CASE <выражение> OF
  <набор значений 1> : <оператор 1>;
  <набор значений 2> : <оператор 2>;
  ...
  <набор значений n> : <оператор n>;
  [ELSE <оператор n+1>]
END
```

Набор значений (value set) подобен набору констант (constant set), участвующему в операции IN, за исключением того, что не стоит в квадратных скобках. Таким образом, он может содержать одно значение, несколько значений, разделенных запятыми, диапазоны, несколько диапазонов, разделенных запятыми, и т.п. Все значения, фигурирующие в наборе значений, должны иметь тип, совместимый с типом CASE-выражения.

Заметим, что блок ELSE для оператора CASE носит необязательный характер, а потому в синтаксисе заключен в квадратные скобки.

Когда оператор CASE выполняется, сначала вычисляется CASE-выражение. Это выражение иногда называют селектором (selector). Затем по очереди вычисляется каждое значение в каждом наборе значений.

Если одно из значений в первом наборе соответствует значению CASE-выражения, то выполняется первый оператор. Если одно из значений во втором наборе соответствует значению CASE-выражения, то выполняется второй оператор. Так продолжается до тех пор, пока не выполнится один из операторов, представленных в CASE.

Если проанализирован последний набор значений и ни один из

операторов не был выполнен, то ищется ELSE-блок. Если он не пуст, то выполняется находящийся в нем оператор. Если же ELSE-блок отсутствует, то для данного оператора CASE не выполняется ни одного оператора.

Обратите внимание, что выполняться будет только один из операторов в вариантах CASE. Если одно и то же значение соответствует разным наборам значений в CASE, то выполняется оператор только для первого из них (кто идет раньше).

Упражнение 1 **Простое использование оператора CASE**

1 Перейдите в Object Designer, выберите программный модуль 95100 и щелкните Design.

2 В триггере OnRun введите следующие операторы, удалив весь ранее введенный туда код.

```
Int2 := 3;

FOR Int1 := 0 TO 8 DO BEGIN
  Color := Int1;

  CASE Color OF
    Color::Orange:
      Int2 := Int1 + Int2;
    Color::Red,Color::Green:
      Int2 := Int2 * 2;
    Color::Yellow:
      BEGIN
        Amt1 := Int1 * 5;
        Int2 := Amt1 DIV 2;
      END;
    Color::Blue,Color::Violet:
      BEGIN
        Amt1 := Int1 * 2 + Amt1;
        Int2 := Amt1 DIV 3;
      END;
    ELSE
      BEGIN
        MESSAGE('Invalid Color');
        Int2 := Int2 + 3;
      END;
  END;

  MESSAGE('The value for %1 is %2',Color,Int2);
END;
```

3 Перед тем, как сохранить этот код, попробуйте выполнить его вручную и предсказать, какие появятся различные выражения.

Если вы забыли, какие использовались цвета, то выберите View, C/AL Globals для отображения списка переменных, потом выберите вариантную переменную Color и нажмите клавишу F5, чтобы открыть окно **Properties**. На отдельном листочке бумаги выпишите прогнозируемые вами сообщения.

4 Теперь осуществите выход, сохраните изменения и запустите программный модуль.

Верны ли были ваши ожидания? Если нет, еще раз просмотрите код и, проанализировав его, выясните, где вы ошиблись.

Упражнение 2 Сложное использование оператора CASE

1 В Object Designer выберите Form 95100 и щелкните Design.

2 Выберите кнопку Execute и перейдите к коду его триггера OnPush. Модифицируйте код в самом начале (до комментария) так, чтобы он имел следующий вид:

```
Result := Quantity * UnitPrice;

CASE TRUE OF
  Quantity = 0:
    EXIT;
  Quantity < 0:
    BEGIN
      TotalCredits := TotalCredits + Result;
      TotalQtyCredited := TotalQtyCredited + Quantity;
      CreditCounter := CreditCounter + 1;
                  AvgQtyCredited := TotalQtyCredited
                  DIV
        CreditCounter;
    END;
  Quantity > 0:
    BEGIN
      TotalSales := TotalSales + Result;
      TotalQtySold := TotalQtySold + Quantity;
```

```
      SalesCounter := SalesCounter + 1;
      AvgQtySold := TotalQtySold DIV SalesCounter;
    END;
END;

GrandTotal := GrandTotal + Result;
GrandTotalQty := GrandTotalQty + Quantity;
```

Заметьте, что мы заменили несколько операторов IF на один оператор CASE.

Давайте аккуратно разберемся в том, как мы строили этот оператор. Обычно в операторе CASE в качестве селектора используется CASE-выражение, и каждый набор значений представляет собой список из одной или нескольких констант. В данном случае мы использовали для селектора константу (TRUE), а каждый набор значений представляет теперь обычное выражение с использованием переменных. Правила для выполнения такого оператора CASE остаются теми же самыми. Сначала вычисляется значение селектора, а затем – каждое значение в каждом наборе. Первый набор значений, где совпадают оба значения, и будет выполнен.

3 Выполните выход, сохраните и запустите форму.

4 Удостоверьтесь в том, что кнопка Execute работает так же, как и раньше.

Заметим, что имеется одно различие. Какое же?

Как нужно поправить код, что выполнение было идентично предыдущему коду?

Глава 13

Вызов встроенных функций

В этой главе рассматриваются функции и их параметры. В ней также представлены некоторые встроенные функции, предоставляемые языком C/AL.

Глава содержит следующие разделы:

- Функции и параметры
- Использование встроенных функций
- Контрольные вопросы

13.1 Функции и параметры

Очень важно до конца понимать, что такое функции и параметры, а также то, как они используются. Поэтому сначала мы дадим определения этим и связанным с ними терминам.

Функции

Функция (function) – это обладающий именем фрагмент программы (иногда называемый подпрограммой, "sub-program" или "subroutine").

Когда используется имя функции (также называемое идентификатором), текущая программа приостанавливается на время выполнения триггерного кода для данной функции. Использование подобным образом идентификатора реализует вызов функции. Когда триггерный код в вызванной функции завершен, то функция возвращает управление в ту точку, откуда она была вызвана. Что произойдет при возврате из функции, напрямую зависит от способа вызова функции.

Функция может быть вызвана в качестве выражения. Например, в следующем выражении используется функция с именем CalculatePrice:

```
TotalCost := Quantity * CalculatePrice;
```

Функция CalculatePrice должна вернуть значение, которое и будет использовано в выражении. Это возвращаемое значение затем умножается на значение переменной Quantity и результат присваивается переменной TotalCost.

Функция может также быть вызвана с помощью оператора вызова (function call statement). Данный оператор просто вызывает функцию и не получает результирующего значения. Ниже приведен пример вызова подобным образом функции с именем RunFunction:

```
IF Quantity > 5 THEN
  RunFunction;
```

Функция RunFunction не возвращает в вызывающую функцию никаких данных.

Встроенные функции

Встроенная функция (built-in function) – это функция, априори

доступная в среде C/SIDE (без необходимости ее кодировать). Вы не можете увидеть ее триггерный код, поскольку она встроена в среду C/SIDE.

Параметр

Параметр (parameter) – это одна или несколько переменных либо выражений, которые передаются функции во время ее вызова. Параметр предоставляет функции информацию и, если требуется, функция может изменять эту информацию. Большинство функций имеет параметры; в этом случае за идентификатором функции следуют круглые скобки. Внутри скобок заключены один или несколько параметров. Если параметров несколько, то они отделяются друг от друга запятыми.

Передача по значению

Иногда передаваемый в функцию параметр только предоставляет ей информацию. В этом случае говорят, что параметр передается по значению (by value). Параметр знает только значение переменной или выражения, которые и были использованы при передаче. Поскольку это просто значение, то изменения, которые будет осуществлять функция по отношению к этому параметру, никоим образом не затронут ни одну переменную в вызывающем триггерном коде.

Передача по ссылке

В других случаях параметр передается функции, и она его модифицирует. Тогда говорят, что параметр передается по ссылке (by reference), или по имени (by name). Параметр фактически знает ту ячейку переменной в памяти, где хранится данная переменная. Параметр передает в новую функцию адрес ячейки памяти. Любые изменения, которые функция делает в отношении параметров такого вида, имеют долговременный характер и воздействуют на переменные в вызывающем триггерном коде.

Более детально эти механизмы раскрываются в следующей главе. Здесь же важно обратить внимание вот на какой момент. Если идет передача по значению, то в качестве параметра вы можете использовать любое выражение. Но если параметр передается по ссылке, то вы должны обязательно использовать в качестве параметра переменную, поскольку ее значение может измениться. Переменная имеет конкретный адрес в памяти, тогда как выражение и константа их не имеют.

13.2 Использование встроенных функций

В предыдущих главах мы уже затрагивали некоторые встроенные функции. Потратьте немного времени, чтобы еще раз взглянуть на них:

Функция	Описание	Доп. информация
MESSAGE	Отображает на экране сообщение	Глава 3
MAXSTRLEN	Возвращает заданную длину текстовой переменной	Глава 5
COPYSTR	Возвращает часть строки	Глава 5
CLEAR	Очищает переменную	Глава 10
ARRAYLEN	Возращает количество элементов массива	Глава 10

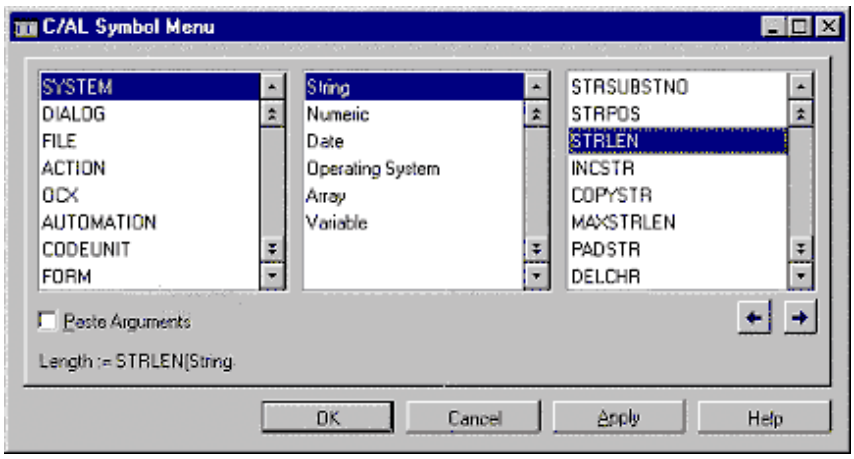
Заметим, что некоторые из них, вроде MESSAGE и CLEAR, не возвращают значений, поэтому они могут использоваться только в операторах вызова. Другие же, COPYSTR и MAXSTRLEN, возвращают значения, а потому могут быть использованы внутри выражений. Кроме того, заметим, что функция CLEAR изменяет переданный параметр и служит примером передачи параметра по ссылке, тогда как другие функции применяют способ передачи параметров по значению.

Меню Symbol в C/AL

1 В Object Designer вновь откройте программный модуль 95100.

2 В строке меню выберите пункты View, C/AL Symbol Menu, либо нажмите клавишу F5. Вы может также нажать кнопку Symbol на строке инструментов.

Появится следующее окно:



Если этого сразу не произойдет, осуществите прокрутку списка в левой панели, пока функции там не сгруппируются так, как показано на картинке.

3 Выделите вариант DIALOG, чтобы увидеть на экране некоторые встроенные функции.

4 В правой панели выделите функцию MESSAGE. Заметим, что внизу фрейма (над командными кнопками) появится строка, подобная следующей:

```
MESSAGE(String [, Value1] …)
```

Она описывает синтаксис функции. Обратите внимание, что функция не возвращает значения и имеет один обязательный параметр (String), а также несколько необязательных параметров. Квадратные скобки говорят о необязательности параметра, а многоточие – о многократном повторении этого фрагмента.

5 Теперь выделите функцию ERROR (прямо над функцией MESSAGE). Как можно видеть, ее синтаксис таков:

```
ERROR(String [, Value1] …)
```

Синтаксис функции `ERROR` идентичен синтаксису MESSAGE (за исключением идентификатора функции и ее действия). Когда функция ERROR вызвается в качестве отдельного оператора, обработка останавливается с указанием причины ошибки и выводом на экран сообщения (способом, аналогичным MESSAGE).

6 В левой панели выделите SYSTEM. Заметим, что в средней панели представлен список различных групп функций, таких как функции работы со строками, числовые (математические) функции и т.п. По мере выделения в средней панели каждого из этих названий происходят изменения в правой панели. Например, выделите в средней панели Date, а затем в правой панели выделите DATE2DMY. Синтаксис этой функции появится внизу окна:

```
Number := DATE2DMY(Date, What)
```

Сочетание "Number :=", предшествующее идентификатору функции, указывает на то, что данная функция возвращает значение и что это значение является числом. Первый параметр (Date) является выражением типа Date. Но что же обозначает параметр What (второй параметр)? Чтобы найти ответ на этот вопрос, вам нужно при выделенной функции DATE2DMY нажать клавишу F1.

На экране появится интерактивная справка для данной функции, где вы может прочитать полное описание того, что она делает и что возвращает. Здесь же вы найдете, что What является целочисленным выражением, которое имеет одно из трех значений. Если What равно 1, то функция возвращает день месяца. Если What равно 2, то функция возвращает номер месяца (от 1 до 12). Если же What равно 3, то функция возвращает год (4 цифры).

Использование функции DATE2DMY

Ниже представлен пример использования данной функции. В нем из переменной типа Date извлекается месяц и отображается на экране в понятном для пользователя виде. Чтобы посмотреть, как это работает, вы можете воспользоваться программным модулем Workbook Exercises.

```
"When Was It" := TODAY;
CASE DATE2DMY("When Was It",2) OF
  1:Description := 'January';
  2:Description := 'February';
  3:Description := 'March';
  4:Description := 'April';
  5:Description := 'May';
  6:Description := 'June';
  7:Description := 'July';
  8:Description := 'August';
  9:Description := 'September';
  10:Description := 'October';
  11:Description := 'November';
```

```
  12:Description := 'December';
END;
MESSAGE('%1 is in %2',"When Was It",Description);
```

Заметим, что первая строка этого кода использует другую функцию, носящую имя TODAY. Эта функция не имеет параметров. Она всегда возвращает текущую дату в вашей операционной системе, иначе называемую еще системной датой.

13.3 Контрольные вопросы

Вопросы

Вопросы в этом разделе относятся к конкретному коду. Чтобы ответить на них, вы должны вручную прогнать его, а также воспользоваться меню Symbol и интерактивной справкой, дабы разобраться в работе отдельных функций. Заметим, что все эти функции могут быть найдены в подразделе String раздела SYSTEM. Не запускайте этот код в тестовом программном модуле, пока не дойдете до вопроса 8.

```
// UserInput is a Text variable of length 100. The
other variables are Integers.
UserInput := 'The colors are red, orange, yellow,
green, blue and violet.';
Count := 0;

REPEAT
  Comma := STRPOS(UserInput,','); //Q1
  IF Comma > 0 THEN BEGIN //Q2
    Count := Count + 1;
    UserInput := DELSTR(UserInput,Comma,1); //Q3
    UserInput := COPYSTR(INSSTR(UserInput,
      ' and',Comma),1,MAXSTRLEN(UserInput));
  END;
UNTIL Comma <= 0;

// Display Results
MESSAGE('The sentence is "%1". Number of commas is
%2.',UserInput,Count);
```

1 Каково значение переменной Comma после первого выполнения оператора, выделенного меткой Q1?

2 Может ли когда-нибудь в этом коде значение переменной Comma быть меньше нуля (см. оператор с меткой Q2)?

3 Каково значение переменной UserInput после первого выполнения оператора, выделенного меткой Q3?

4 Каково значение переменной Count в момент, когда в первый раз достигается строка комментария?

5 Предположим, что переменная UserInput была переопределена таким образом, что она имеет тип Text и длину 60. Каким будет ее значение, когда выполнение достигает функции MESSAGE после строки комментария?

Заметим, что при определенных обстоятельствах может быть потерян конец предложения. Предположим, что вы модифицировали условие цикла при UNTIL для оператора REPEAT так, что оно выглядит следующим образом:

```
UNTIL (Comma <= 0) OR (STRLEN(UserInput) + 3 >
MAXSTRLEN(UserInput));
```

6 Каким теперь будет значение переменной UserInput, когда будет достигнута функция MESSAGE?

7 Хотя стало несколько лучше, но это далеко от желаемого результата. Почему такое решение не работает?

8 Вернитесь назад и запустите данный код своего тестового программного модуля. Не забудьте, что вы должны определить некоторые новые переменные. Теперь внесите как можно более простое изменение, чтобы устранить проблему, затронутую в вопросе 7. Напишите свои изменения:

Ответы

1 Каково значение переменной Comma после первого выполнения оператора, выделенного меткой Q1?

19

2 Может ли когда-нибудь в этом коде значение переменной Comma быть меньше нуля (см. оператор с меткой Q2)?

Нет

3 Каково значение переменной UserInput после первого выполнения оператора, выделенного меткой Q3?

The colors are red, orange, yellow, green, blue and violet.

4 Каково значение переменной Count в момент, когда в первый раз достигается строка комментария?

4

5 Предположим, что переменная UserInput была переопределена таким образом, что она имеет тип Text и длину 60. Каким будет ее значение, когда выполнение достигает функции MESSAGE после строки комментария?

The colors are red and orange and yellow and green and blue

6 Каким теперь будет значение переменной UserInput, когда будет достигнута функция MESSAGE?

The colors are red and orange, yellow, green, blue and violet

7 Хотя стало несколько лучше, но это далеко от желаемого результата. Почему такое решение не работает?

Поскольку цикл REPEAT всегда выполняется как минимум один раз, а проверка делалась после того, как переменная уже была изменена.

8 Вернитесь назад и запустите данный код своего тестового программного модуля. Не забудьте, что вы должны определить некоторые новые переменные. Теперь внесите как можно более простое изменение, чтобы устранить проблему, затронутую в вопросе 7. Напишите свои изменения:

Здесь приведен один из возможных ответов:

```
UserInput := 'The colors are red, orange, yellow,
              green, blue and violet.';
Count := 0;

WHILE (STRPOS(UserInput,',') <> 0) AND
      (STRLEN(UserInput) + 3 <=
MAXSTRLEN(UserInput))
            DO BEGIN
  Comma := STRPOS(UserInput,',');
  Count := Count + 1;
  UserInput := DELSTR(UserInput,Comma,1);
  UserInput := INSSTR(UserInput,' and',Comma);
END;

MESSAGE('The sentence is "%1". Number of commas is
%2.',UserInput,Count);
```

Обратите внимание, что цикл REPEAT заменен на цикл WHILE. При этом условие цикла должно быть заменено на свое логическое отрицание, так как цикл REPEAT выполняется до тех пока условие цикла имеет значение FALSE, а цикл WHILE выполняется, пока условие цикла равно TRUE. Чтобы обеспечить логическое отрицание любого логического выражения, нужно перед ним поставить знак `NOT`, а все выражение заключить в круглые скобки. Либо осуществить следующее преобразование: изменить каждую реляционную операцию на противоположную (“=" на "<>" и наоборот, ">" на "<=" и наоборот, а также "<" на ">=" и наоборот) и изменить каждую операцию `AND` на `OR`, а каждую операцию `OR` на `AND`.

Глава 14

Создание ваших собственных функций

Данная глава объясняет, зачем и как создавать собственные функции. Рассматривается также оператор EXIT.

Глава содержит следующие разделы:

- Формальные и фактические параметры
- Локальные функции и переменные
- Зачем надо создавать функции
- Оператор EXIT
- Создание функций
- Контрольные вопросы

14.1 Формальные и фактические параметры

Формальный параметр

Формальный параметр (formal parameter) – это параметр, фигурирующий в описании функции. В упражнении, приведенном в предыдущей главе, мы использовали функцию DELSTR. Когда вы посмотрите на описание функции DELSTR в меню Symbol, вы увидите, что ее синтаксис выглядит так:

```
NewString := DELSTR(String, Position [, Length])
```

Слова, которые появляются в круглых скобках, являются формальными параметрами. Если бы вы могли посмотреть триггерный код для встроенной функции, то увидели бы, что эти формальные параметры могли использоваться в этом триггерном коде в качестве переменных.

Фактический параметр

Фактический параметр (actual parameter) – это такой параметр, который подставляется при вызове функции. Когда мы вызывали в упомянутом упражнении функцию DELSTR, то использовали следующую строку:

```
UserInput := DELSTR(UserInput,Comma,1);
```

Константы и переменные, которые появляются в круглых скобках, являются фактическими параметрами.

Заметим, что между формальными и фактическими параметрами существует взаимнооднозначное соответствие. Фактический параметр UserInput становится формальным параметром String, фактический параметр Comma становится формальным параметром Position, а фактический параметр 1 становится формальным параметром Length.

Поскольку для всех трех параметров мы используем передачу по значению, то по сути мы передаем в функцию значения всех трех фактических параметров. Таким образом, фактические параметры не изменяются при изменении внутри функции самих формальных параметров.

Если бы мы использовали передачу параметров по ссылке, то нам надо было бы передавать в функцию ссылку на переменную (ее адрес в памяти). Затем если бы формальные параметры изменялись внутри

функции, то изменялись бы и соответствующие фактические параметры. Поскольку константы не могут изменяться, то мы не могли бы использовать 1 в качестве фактического параметра (если бы параметр Length передавался по ссылке). Это привело бы к генерированию синтаксической ошибки.

14.2 Локальные функции и переменные

Локальная функция

Локальная функция (local function) – это функция, которая может вызываться только внутри того объекта, в котором она определена. Любая функция, которая не была определена как локальная, может вызываться из других объектов, находящихся за пределами данного объекта (где она была определена).

Локальная переменная

Локальная переменная (local variable) – это переменная, область видимости которой ограничена данной функцией. Это означает, что в триггерном коде функции локальная переменная может использоваться подобно любой другой переменной. Однако в остальной части объекта данная переменная недоступна. Если на имя локальной переменной ссылаются за пределами функции, где она определена, то возникает синтаксическая ошибка. Формальные параметры функции тоже трактуются как локальные переменные этой функции.

14.3 ЗАЧЕМ НАДО СОЗДАВАТЬ ФУНКЦИИ

Мы можем писать весьма полезные программы и без применения функций. Так зачем же мы должны их создавать? Верно, писать программы можно и без функций. Более того, почти для всего, что вам может потребоваться в среде C/SIDE, нет необходимости создавать функции с целью расширения имеющейся функциональности. В то же время существуют довольно серьезные причины, зачем программисту создавать собственные функции. Вот они:

- Чтобы сконструировать свою программу. По отношению к вашему коду функция выполняет роль заголовка, что позволяет получить хорошо структурированный документ. Функции упрощают анализ программы, причем не только для вас, но и для других программистов, которым впоследствии может понадобиться вносить в код коррективы.

- Чтобы упростить свои задачи. Когда вы проектируете свою программу, то можете разбить сложную проблему на несколько более мелких задач. Каждая из этих задач может стать функцией в вашей программе, а вся программа может состоять из этих мелких задач. Если функция станет чересчур сложной, вы можете вновь ее разбить на более мелкие задачи и создать для каждой из них свою функцию.

- Чтобы облегчить свою работу по повторному использованию кода. Если вы обнаружите, что в двух разных частях программы нужно реализовать в точности такую же или очень схожую задачу, то вы может создать для этого функцию. Затем вместо того чтобы многократно в разных местах программы писать такой же или подобный код, вы можете разместить его в функции и вызывать ее из этих мест.

- Чтобы уменьшить вероятность появления ошибок. Функция может быть вами оттестирована с помощью специальных тестовых программ, анализирующих все ее возможности. Таким образом, когда вы будете использовать ее в другом месте, то при поиске ошибок он может быть упрощен. Соответственно, если ошибка обнаружена в функции, то она может быть исправлена в одном месте, что автоматически устранит ее в тех местах, откуда функция вызывалась. Если же вы должны править ошибку в каждом месте, то вы можете что-то упустить или даже внести новую ошибку.

- Чтобы упростить внесение изменений. Если вам требуется определенным образом поправить работу своей программы, и

в разных местах у вас используется похожий код, то вам придется вносить изменения в каждое из этих мест. Если вы разместите общий для них код в одной функции, то изменения придется вносить только в одном месте. Это сокращает объем работ и уменьшает риск появления новых ошибок.

- Чтобы локализовать данные. Когда функция выполняет некую задачу, она может иметь собственные локальные данные, которые не могут оказывать воздействие на другие функции в том же объекте. Если у вас имеются глобальные переменные, используемые во всем тексте программы, то это таит в себе угрозу непреднамеренного разрушения этих данных.

- Чтобы сократить размер своих объектов. Хотя в первом приближении экономия незначительная, она может быть полезна, особенно когда вы пытаетесь разместить что-то в своем коде.

Существует немало серьезных причин для создания функций. Действительно, когда вы проектируете код для объекта, то сначала должны сформулировать основные задачи и создать (определить) для каждой из них свою функцию. Затем, когда вы увидите, что в различных местах программы решаются схожие задачи, то всегда имеет смысл подумать над введением для них новой функции.

Другая причина, подталкивающая к созданию функций, специфична для среды C/SIDE. Дело в том, что функции являются одним из основных способов взаимодействия между объектами. Помните, что переменные не могут разделяться между объектами, а функции могут. Более подробно это будет рассмотрено в курсе программирования для Navision.

14.4 Оператор EXIT

Как мы уже обсуждали, оператор EXIT используется для остановки исполнения триггера. Он может применяться подобным образом и для функции Triggers. Однако в функциях оператор EXIT приобретает дополнительные особенности.

Если вызов функции происходит внутри выражения, то требуется, чтобы она возвращала значение. Когда вы пишете собственную функцию, которая возвращает значение, то можете выполнять это с помощью оператора EXIT. Ниже представлен его синтаксис:

```
EXIT(<выражение>);
```

Давайте, например, создадим функцию с именем Square, которая используется для вычисления квадрата числа. Следующее выражение…

```
Answer := 4 + Square(5);
```

...приведет к присваиванию переменной Answer значения 29. Триггерный код функции Square с формальным параметром Param может быть таким

```
EXIT(Param * Param);
```

Параметром оператора EXIT служит выражение, которое вычисляет квадрат числа и возвращает его через имя функции в точку вызова.

14.5 Создание функций

1 В Object Designer выберите Form 95100 и щелкните Design.

2 Переместите в правый верхний угол формы путем перетаскивания командную кнопку Sort.

3 Выберите командную кнопку Execute. Нажмите на строке инструментов кнопку Copy (либо на клавиатуре комбинацию клавиш CTRL+C). Щелкните в любом месте на форме, которая не имеет у себя элементов управления. Теперь на строке инструментов нажмите кнопку Paste (либо нажмите на клавиатуре комбинацию клавиш CTRL+V). Копия кнопки Execute будет помещена в левом верхнем углу формы. Теперь перетащите эту копию на позицию за исходной кнопкой Execute.

4 Откройте для левой кнопки окно **Properties** и измените значение свойства Caption на *Sale*. Выберите исходную кнопку Execute и измените значение свойства Caption на *Credit*.

В этот момент ваша форма имеет вид:

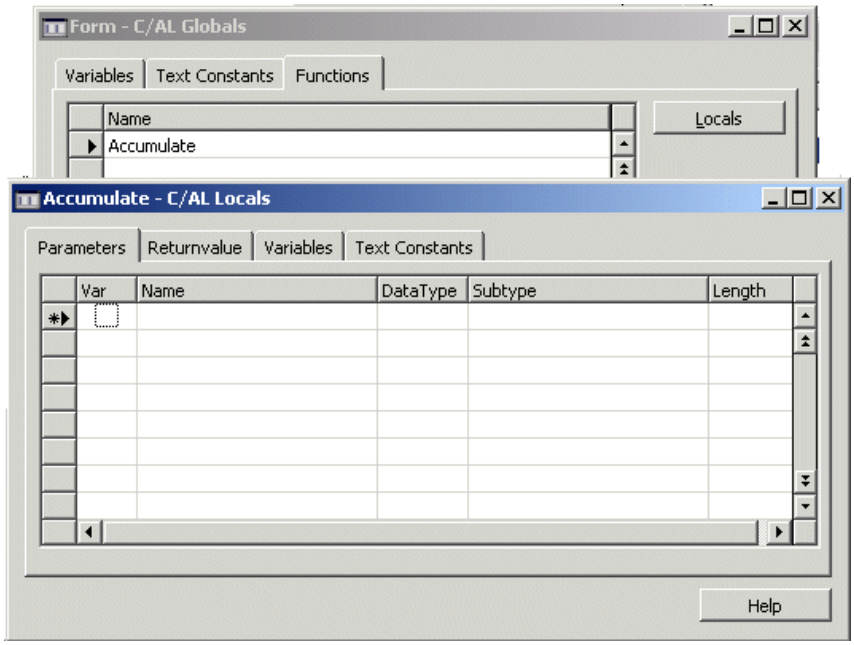
Определение функции

1 Чтобы открыть окно **C/AL Globals,** щелкните View, C/AL Globals. Заметим, что на этой форме имеются три закладки, а до сего момента мы использовали только первую из них. Теперь щелкните третью закладку – **Functions**. Сюда мы зайдем для того, чтобы определить функцию.

2 На первой строке в столбце **Name** введите имя нашей первой функции – *Accumulate*.

3 Как только строка воспримет ввод и отобразит Accumulate, откройте окно **Properties**. Заметим, что одно из свойств называется Local. Так как мы не хотим, чтобы эта функция вызывалась из объектов за пределами данной формы, установите значение свойства в *Yes*. Это позволит сделать нашу функцию локальной. Закройте окно **Properties**.

4 В окне **C/AL Globals** щелкните Locals. Появится окно **C/AL Locals**. Эта форма используется для завершения определения нашей новой функции Accumulate. Она должна отобразить в своем заголовке название Accumulate:



В этом окне вы можете установить для функции формальные параметры, возвращаемое значение, локальные переменные и текстовые константы, которые могут понадобиться.

5 Для этой функции нам необходим только один параметр, поэтому введите *Qty* в качестве имени и *Integer* в качестве типа данных.

Столбец с именем Var используется тогда, когда вы хотите, чтобы данный параметр передавался по ссылке. Параметр, переданный по ссылке, должен быть переменной (не выражением). Вот почему он назван Var, что является сокращением от Variable. В данном случае мы не будет трогать

этот столбец.

6 Закройте эту форму, чтобы мы смогли вернуться в окно **C/AL Globals**, которое все еще находится на закладке Functions. Мы закончили формировать определение функции Accumulate.

Определение дополнительных функций

1 Создайте новую функцию с именем *Extend*.

2 Установите значение ее свойства Local в *Yes* (как описано выше).

3 Щелкните Locals и создайте два параметра. Первый должен быть целочисленным с именем *Qty,* а второй должен иметь тип Decimal и называться *Unit*.

4 В окне **C/AL Locals** выберите вторую закладку – Return Value. В поле **Return Type** щелкните на кнопку AssistButton, чтобы увидеть список возможных типов возвращаемых значений. Выберите *Decimal* в качестве типа возвращаемого значения для этой функции.

5 Закройте окно **C/AL Locals**. При этом будет завершено формирование определения функции Extend.

6 Создайте новую функцию *AddToArray*.

7 Установите ее свойство Local в значение *Yes*, а затем щелкните Locals. Добавьте параметр типа Integer с именем *CurLen*. Поставьте флажок в поле **Var**. Это сделает параметр CurLen Var-параметром, что означает передачу его по ссылке, а не по значению.

8 Создайте второй параметр с именем *NumToAdd* и типом данных Decimal. Этот параметр не будет Var-параметром; он будет передаваться по значению.

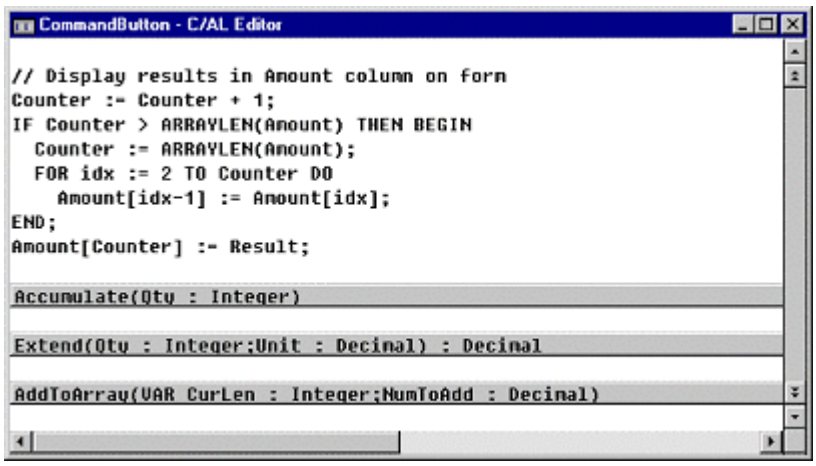
9 В окне **C/AL Locals** выберите третью закладку – Variables. Добавьте переменную типа Integer с именем *idx*.

10 Закройте окна **C/AL Locals** и **C/AL Globals**.

Добавление кода в вашу функцию

1 Чтобы открыть окно **Редактора C/AL**, щелкните командную

кнопку Sale, а затем щелкните кнопку C/AL Code (либо нажмите клавишу F9). Прокрутите код вниз и вы увидите три новые функции:



2 Если эти три строки триггерного кода функции выглядят не так, вернитесь в окно **C/AL Globals** и скорректируйте определение функции, чтобы было полное соответствие.

3 Для каждой из этих трех функций введите триггерный код (приведен ниже). Заметим, что весь этот код похож на код триггера OnPush для кнопки Sale. Если вы чувствуете, что это вас не запутает, можете скопировать код оттуда и поместить его в эти функции. Затем внесите в него правки, чтобы он соответствовал следующему:

```
Accumulate:

Result := Extend(Qty,UnitPrice);
IF Qty < 0 THEN BEGIN
  TotalCredits := TotalCredits + Result;
  TotalQtyCredited := TotalQtyCredited + Qty;
  CreditCounter := CreditCounter + 1;
  AvgQtyCredited := TotalQtyCredited DIV CreditCounter;
END ELSE BEGIN
  TotalSales := TotalSales + Result;
  TotalQtySold := TotalQtySold + Qty;
  SalesCounter := SalesCounter + 1;
  AvgQtySold := TotalQtySold DIV SalesCounter;
END;
GrandTotal := GrandTotal + Result;
GrandTotalQty := GrandTotalQty + Qty;
```

```
Extend:

EXIT(Qty * Unit);




AddToArray:

CurLen := CurLen + 1;
IF CurLen > ARRAYLEN(Amount) THEN BEGIN
  CurLen := ARRAYLEN(Amount);
  FOR idx := 2 TO CurLen DO
    Amount[idx-1] := Amount[idx];
END;
Amount[CurLen] := NumToAdd;
```

Вызов вашей функции

1 Прокрутите код триггера OnPush для командной кнопки Sale и поправьте его так, чтобы он соответствовал следующему:

```
IF Quantity = 0 THEN
 EXIT;
Accumulate(Quantity);
AddToArray(Counter,Extend(Quantity,UnitPrice));
```

2 Закройте форму с кодом, а затем щелкните командную кнопку Credit. Откройте вновь форму с кодом и поправьте код триггера OnPush для командной кнопки Credit, чтобы он выглядел так:

```
IF Quantity = 0 THEN
  EXIT;
Accumulate(-Quantity);
AddToArray(Counter,Extend(-Quantity,UnitPrice));
```

Заметим, что обе подпрограммы маленькие и очень похожи друг на друга. Единственное различие состоит в том, что кнопка Credit перед вызовом функций возвращает отрицательное значение Quantity. Это означает, что пользователь больше не будет вводить отрицательное количество. Вместо этого он будет просто вводить количество, а затем (в зависимости от требуемого вида транзакции) нажимать кнопку Sale, либо кнопку Credit.

3 Закройте и сохраните этот Form Object. Затем запустите его и посмотрите, что получилось.

14.6 Контрольные вопросы

Зайдите в Object Designer и найдите программный модуль с номером 1, ApplicationManagement. Откройте его и, просмотрев объект, ответьте на приведенные ниже вопросы. Помните, если вопрос касается определения функции, то лучше взглянуть на окно **C/AL Globals** (закладка Functions) и на окно **C/AL Locals**. Если вопрос касается кода, то лучше взглянуть на триггерный код.

Вопросы

1 Сколько параметров имеет функция LoginStart?

2 Каков тип возвращаемого значения (если таковое имеется) функции ReadRounding?

3 Каким образом в функции ReadSymbol передается первый параметр (Token): по ссылке или по значению?

4 Каким образом в функции ReadSymbol передается первый параметр (Text): по ссылке или по значению?

5 Сколько локальных переменных определено в функции AutoFormatTranslate?

6 Взгляните на код функции AutoFormatTranslate function. Сколько строк триггерного кода может возвращать значение этой функции?

7 Взгляните на код и определение функции ApplicationLanguage. Есть ли что-нибудь в этой функции, что воздействует на данные за ее пределами (не считая возвращаемого значения)? Если да, то что?

8 Взгляните на код и определение функции ReadCharacter. Есть ли что-нибудь в этой функции, что воздействует на данные за ее пределами (не считая возвращаемого значения)? Если да, то что?

Ответы

1 Сколько параметров имеет функция LoginStart?

Ни одного (0)

2 Каков тип возвращаемого значения (если таковое имеется) функции ReadRounding?

Decimal

3 Каким образом в функции ReadSymbol передается первый параметр (Token): по ссылке или по значению?

По значению

4 Каким образом в функции ReadSymbol передается первый параметр (Text): по ссылке или по значению?

По ссылке (флажок в столбце Var)

5 Сколько локальных переменных определено в функции AutoFormatTranslate?

4 (Currency, GLSetup, OK, GLSetupRead)

6 Взгляните на код функции AutoFormatTranslate function. Сколько строк триггерного кода может возвращать значение этой функции?

8 (8 различных строк содержат оператор EXIT)

7 Взгляните на код и определение функции ApplicationLanguage. Есть ли что-нибудь в этой функции, что воздействует на данные за ее пределами (не считая возвращаемого значения)? Если да, то что?

Нет. Все переменные здесь либо локальные, либо являются формальными параметрами, которые передаются по значению.

8 Взгляните на код и определение функции ReadCharacter. Есть ли что-нибудь в этой функции, что воздействует на данные за ее пределами (не считая возвращаемого значения)? Если да, то что?

Да. Обе переменные Text и Position передаются по ссылке. Это означает, что на них ссылаются за пределами функции.

Глава 15

Функции языка C/AL

В данной главе приводится обзор функций, которые язык C/AL предоставляет для работы со строками, числами, датами и др. Для каждой функции представлен ее синтаксис, а для наиболее часто используемых функций приведены примеры.

Заметим, что в данной главе не используются текстовые константы, как это требуется в системе Navision. Для дополнительной информации о текстовых константах см. руководство *Application Developer's Guide*.

Глава содержит следующие разделы:

- Функции взаимодействия с пользователем
- Функции работы со строками
- Системные функции
- Функции работы с датой
- Функции работы с числами
- Функции работы с массивами
- Другие важные функции

15.1 Функции взаимодействия с пользователем

Следующие функции отвечают за обратную связь с пользователем и за ввод информации в программу.

MESSAGE

```
MESSAGE (String [, Value1, ...])
```

Функция MESSAGE запускается немодально. Это означает, что сначала выполняется оставшийся после MESSAGE код. Функция выводит на экран сообщение для пользователя (String), но только после того, как весь процесс обработки будет завершен. Окно остается открытым, пока пользователь не щелкнет кнопку OK (или не нажмет клавишу ENTER).

Данная функция часто используется для отладки, в частности, для отображения значений переменных (Value1). Однако если возникнет ошибка, то поскольку MESSAGE работает немодально, процесс будет остановлен и сообщение не появится.

При работе со строками знак плюса (“+”) используется для конкатенации текста, а обратная наклонная черта (“\”) является признаком перехода на новую строчку. Символы “%” и “#” могут использоваться в качестве знакомест для заполнителей. Знак процента (“%”) используется для свободного формата, а знак фунта (“#”) – для фиксированного формата.

Пример:

```
Value1 := 12345.678;
Value2 := 987.65
MESSAGE( 'The Format of Value1 is %1 \' + ' The Fixed
Format of Value2 is #2#########', Value1, Value2);
```

В окне сообщений появится:

The Free Format of Value1 is 12,345.678

The Fixed Format of Value2 is 987.65

CONFIRM

```
CONFIRM (String [, Default] [, Value1, ...])
```

Данная функция аналогична MESSAGE, но дает возможность

пользователю отвечать на вопрос, щелкая кнопки Yes и No. Кроме того, в зависимости от выбора пользователя она возвращает булево значение (True или False). Поскольку эта функция запускается модально, то система будет ожидать ответа пользователя.

CONFIRM часто используется в тех случаях, когда надо получить от пользователя подтверждение на продолжение данного процесса. Navision использует CONFIRM перед обновлением записей БД. Пользователю дается возможность прервать или продолжить процесс обновления данных.

Значением по умолчанию является False, как и для обычного булевого типа, следовательно, кнопка No служит кнопкой по умолчанию и является активной. Это означает, что если пользователь просто нажмет клавишу Enter, то функция вернет значение False. Если же нужно, чтобы пользователь чаще выбирал кнопку Yes, то установив параметр Default в значение TRUE, вы сделаете кнопку Yes кнопкой по умолчанию.

Пример:

```
IF NOT CONFIRM('Do You  want to post the Journal
Lines?') THEN
  EXIT()
// Otherwise run the posting routine.
```

Функция CONFIRM ограничена использованием допустимых вариантных значений (Yes, No). Если же нужны другие значения, то можно использовать другую функцию – STRMENU.

STRMENU

```
STRMENU (OptionString [, DefaultNumber])
```

Данная функция используется для создания и отображения формы с вариантной группой, возвращая при этом значение (типа Integer) выбора пользователя. Если значение по умолчанию (DefaultNumber) не указывается, то таковым является первый элемент в вариантной строке (OptionString) со значением 1. Значение 0 возвращается, если нажимается клавиша ESC, что говорит о том, что пользователь отказался от выбора.

Пример:

```
MESSAGE('Your selection returns the value of %1',
STRMENU('Yes, No, N/A'));
```

ERROR

```
ERROR(String [, Value1, ...])
```

Данная функция генерирует код ошибки и выходит из текущего процесса, отменяя весь процесс (а не только одну функцию). Поэтому если программный код выполняется в рамках транзакции, то транзакция останавливается и происходит откат всех необновленных данных.

Функция возвращает пользователю сообщение об ошибке (String), информируя о причине прерывания процесса выполнения.

Пример:

```
IF Number <= 0 THEN BEGIN

  ERROR('Number must be positive.  Current value: %1.',
Number);

  MESSAGE('This Message will never be displayed');

END;
```

15.2 Функции работы со строками

Эти функции дают возможность пользователю оперировать строками. Для более подробной информации см. интерактивную справку *C/SIDE Reference Guide*.

STRPOS

```
Position := STRPOS(String, SubString)
```

Функция STRPOS используется для поиска первого вхождения подстроки (SubString) в строку (String). При этом возвращается позиция в строке первого символа подстроки. Если внутри строки данная подстрока не найдена, то функция возвращает 0. Как и все функции работы со строками, STRPOS чувствительна к регистру букв.

COPYSTR

```
NewString := COPYSTR(String, Position [, Length])
```

Функция COPYSTR применяется для копирования подстроки указанной длины (Length) из указанной позиции (Position) в строке (String) в новую строку. Если длина не указана, то результат будет возвращать все символы, начиная от указанной позиции до конца строки.

PADSTR

```
NewString := PADSTR(String, Length [, FillCharacter])
```

Данная функция используется для изменения длины строки на новую длину. Если параметр Length меньше реальной длины строки String, то строка усекается. В противном случае она расширяется до нового размера символом FillCharacter. Если параметр FillCharacter не указан, то строка будет расширяться пробелами.

Существуют и две другие функции, схожие с PADSTR. Функция DELSTR используется для удаления подстроки из данной строки. А функция INSSTR используется для вставки подстроки в конкретную позицию данной строки. Синтаксис этих функций выглядит так:

```
NewString := DELSTR(String, Position [, Length])

NewString := INSSTR(String, SubString, Position)
```

STRLEN

```
Length := STRLEN(String)
```

Данная функция возвращает целочисленное значение, которое является длиной строки String.

MAXSTRLEN

```
MaxLength := MAXSTLEN(String)
```

Эта функция аналогична STRLEN, также возвращает целочисленное значение, но оно представляет собой максимально возможную длину строковой переменной.

LOWERCASE и UPPERCASE

```
NewString := LOWERCASE(String),

NewString := UPPERCASE(String)
```

Эти функции отвечают за преобразование строки к буквам нижнего (LOWERCASE) и верхнего (UPPERCASE) регистра соответственно.

CONVERTSTR

```
NewString := CONVERTSTR(String, FromCharacters,
ToCharacters)
```

Данная функция преобразует символы в строке, опираясь на строки FromCharacter и ToCharacters, которые служат таблицей преобразования.

DELCHR

```
NewString := DELCHR(String [, Where] [, Which])
```

Данная функция используется для удаления одного или более символов строки. Она может удалять либо символы из конца строки, либо все их вхождения в строку. Заметим, что она может быть использована для вырезания пробелов:

```
DELCHR(StringVar, '<>', ' ')
```

INCSTR

```
NewString := INCRSTR(String)
```

Используйте эту функцию для увеличения в строке положительного числа или уменьшения отрицательного числа на 1. Если в строке встречается более одного числа, то изменяется только первое.

SELECTSTR

```
NewString := SELECTSTR(Number, CommaString)
```

Данная функция извлекает из строки (CommaString), содержащей разделенные запятыми подстроки, нужную подстроку с указанным номером (Number). Нумерация подстрок идет от 1.

STRCHECKSUM

```
CheckNumber := STRCHECKSUM(String [,WeightString] [,
Modulus])
```

Эта функция используется для различных приложений, в частности, для работы со штрих-кодом. Функция вычисляет контрольную сумму для строки, содержащей число.

15.3 Системные функции

Данные функции не требуют никаких параметров, поскольку они возвращают информацию, хранящуюся в системе.

USERID

```
Name := USERID
```

Эта функция возвращает идентификатор данного пользователя.

COMPANYNAME

```
Name := COMPANYNAME
```

Эта функция возвращает текущую группу БД (company), которая используется в программе.

TODAY и TIME

```
DateVar := TODAY, TimeVar := TIME
```

Эти функции возвращают системные дату и время.

WORKDATE

```
[WorkDate] := WORKDATE([NewDate])
```

Эта функция возвращает текущую рабочую дату (workdate), либо может сбрасывать эту дату. И хотя разработчик может править рабочую дату, надо помнить, что в стандартных приложениях Navision только пользователь может ее изменять. Это не вносит корректив в программный код.

15.4 Функции работы с датой

Следующие функции предоставляют специфическую информацию, связанную с датой.

DATE2DMY

```
IntegerVar := DATE2DMY(Date, Integer)
```

Эта функция возвращает информацию о параметре Date. Если параметр Integer имеет значение 1, то функция вернет день месяца (1-31). Если Integer имеет значение 2, то будет возвращен номер месяца (1-12); если же значение 3, то номер года (0000-9999).

DATE2DWY

```
IntegerVar := DATE2DWY(Date, Integer)
```

Эта функция также возвращает информацию о параметре Date. Если параметр Integer имеет значение 1, то функция вернет день недели (1-7; где 1 обозначает понедельник). Если значение 2, то это будет номер недели с начала года (1-53); если же 3, то номер года (0000-9999).

CALCDATE

```
CALCDATE(DateExpression [, Date])
```

CALCDATE – это весьма полезная функция. Она вычисляет новую дату исходя из указанной даты (Date) и соответствующего хронологического выражения (DateExpression). Вычисление дат на основе дат, указанных в документах, часто требуется при использовании Navision.

Ниже приведены примеры хронологических выражений:

‘CQ + 1M – 10D’:	последний день текущего квартала + 1 месяц – 10 дней
‘-WD2’:	второй день последней недели (последний вторник)
‘CM + 30D’:	последний день текущего месяца + 30 дней

Calcdate(‘CM +15D’, 030502D) возвращает значение 04/15/02.

Функция CALCDATE ведет здесь отсчет от 5 марта 2002 г. и переходит к последнему дню текущего месяца (CM). В итоге это даст 03/31/02. Затем добавляется 15 дней и получается 15 апреля 2002 г.

Хронологическое выражение может быть любой длины; оно должно при этом иметь хотя бы одно подвыражение. Система обрабатывает строку слева направо, оперируя подвыражениями. Подвыражение (SubExpression) – это положительное или отрицательное значение (Term).

Значение Term может быть комбинацией число-масштаб (Number–Unit), масштаб-число (Unit–Number) или префикс-масштаб (Prefix–Unit). Число (Number) – это положительное целое число. Масштаб (Unit) может быть одним из следующих:

D	(Date, дата)
WD	(WeekDay, день недели)
W	(Week, неделя)
M	(Month, месяц)
Q	(Quarter, квартал)
P	(Period, период)
Y	(Year, год)

Существует один префикс – C. Он устанавливается для даты закрытия (Closing date) данного периода. Дата закрытия для квартала равна 12:59 PM в последний день последнего месяца квартала.

Каждая дата, начиная от 01/01/0000 до 12/31/9999, имеет соответствующую обычную дату и дату закрытия (последняя – для записи в журнал). С точки зрения системы дата закрытия равна 12:59 PM текущей даты. Дата закрытия при сортировке располагается сразу после соответствующей обычной даты, но перед следующей обычной датой. Буква D говорит о том, что дата обычная; буква C характеризует дату закрытия. Следующие функции определяют, какой вид даты используется:

NORMALDATE

```
ReturnDate := NORMALDATE(Date)
```

Чаще всего встретить эту функцию можно в процедурах записи в гроссбух. Для дат записи в гроссбух (posting dates) и для дат счет-фактур (invoice dates) должны использоваться обычные даты. Следующий код ведет проверку того, является ли дата обычной.

Пример:

```
IF "Posting Date" <> NORMALDATE("Posting Date") THEN
  ERROR('Posting Date cannot be a closing date');
```

CLOSINGDATE

```
ReturnDate := CLOSINGDATE(Date)
```

Эта функция противоположна функции NORMALDATE. Она возвращает соответствующую дату закрытия.

15.5 Функции работы с числами

Следующие функции используются довольно редко, за исключением функции ROUND. Последняя применяется при работе с валютой и налогами.

ABS

```
NewNumber := ABS(Number)
```

Данная функция возвращает абсолютной значение числа (по модулю).

POWER

```
NewNumber := POWER(Number, Power)
```

Возводит параметр Number в степень Power. Эта функция может также вычислять корни. Например, чтобы вычислить квадратный корень числа 16, надо записать POWER(16, .5) , что в результате вернет 4.

ROUND

```
NewNumber := ROUND(Number [,Precision] [, Direction])
```

Эта функция ведет округление числа. Параметр Precision отвечает за точность округления. Для US-версии значением по умолчанию является .01, однако, установки в G/L могут воздействовать на это значение. Параметр Direction определяет направление округления:

=	округлять до ближайшего значения (по умолчанию)
>	округлять вверх
<	округлять вниз.

RANDOMIZE

```
RANDOMIZE ([Seed])
```

Эта функция генерирует серию случайных чисел. Seed – это необязательный целочисленный параметр, который используется для создания уникальной серии чисел. При использовании одного и того же значения Seed будет формироваться одинаковая серия чисел. Если параметр опущен, то вместо него используется текущее системное время (общее количество миллисекунд от полуночи).

RANDOM

```
Number := RANDOM(MaxNumber)
```

Используя серию случайных чисел, формируемую функцией RANDOMIZE, эта функция возвращает псевдослучайное число, лежащее в диапазоне от 1 до MaxNumber. Если RANDOMIZE не будет вызвана еще раз, то RANDOM будет возвращать значения из той же построенной серии случайных чисел.

15.6 Функции работы с массивами

Массивы используются в Navision редко, однако, они частенько встречаются в отчетах для информации об адресах и квитанциях.

ARRAYLEN

```
Length := ARRAYLEN(Array [, Dimension])
```

Возвращает общее количество элементов массива (Array) или количество элементов для указанного измерения (Dimension). Для трехмерного массива допустимы измерения с номерами 1, 2 и 3. Если измерение не указывается, то функция возвращает количество всех элементов массива.

COMPRESSARRAY

```
[Count :=] COMPRESSARRAY(StringArray)
```

Данная функция полезна только для текстовых или кодовых массивов. Она перемещает все непустые строки в массиве в его начало. Количество элементов в массиве остается прежним, но пустые строки и строки, содержащие одни пробелы, переносятся в конец массива.

Если возвращаемое значение присваивается какой-нибудь переменной, то в нем передается количество непустых строк, “сжатых” системой.

Хорошим примером использования данной функции может служить ситуация, когда печатаются имена и адреса. Она будет полезна, в частности, для удаления пустых строк в многострочных адресах.

COPYARRAY

```
COPYARRAY(NewArray, Array, Position [, Length])
```

Данная функция копирует один или несколько элементов одного массива (Array) в другой (NewArray). Параметр Position отвечает за позицию первого элемента, подлежащего копированию. Параметр Length обозначает количество копируемых элементов массива. Если параметр Length опущен, то копируются все элементы, начиная с позиции Position и до конца массива. Эта функция используется только для одномерных массивов. Повторно вызывайте эту функцию для копирования нескольких измерений массивов.

15.7 Другие важные функции

В Navision существуют и другие важные функции.

EXIT

```
EXIT (Value)
```

Эта команда обеспечивает непосредственный выход из текущей функции или триггера. Если имеется родительская функция, которая вызвала текущую функцию или триггер, то ей возвращается значение Value. Это не приводит ни в возникновению ошибке, ни к откату транзакций.

Пример:

```
Function AddTen (Number : Integer) : Integer
BEGIN
  Exit(Number + 10);
  Message ('This line is never read.')
END;
CLEAR
CLEAR(Variable)
```

Функция CLEAR очищает значение переменной (или всех элементов массива) в соответствии со следующими правилами:

Тип переменной	Результат CLEAR
Number	0 (ноль)
String	пустая строка
Date	0D (неопределенная дата)
Time	0T (неопределенное время)
Boolean	FALSE

CLEARALL

```
CLEARALL
```

Эта функция без параметров очищает все внутренние переменные в текущем объекте и во всех вызванных объектах, таких как отчеты, программные модули и другие, содержащие C/AL-код. Она работает путем вызова функции CLEAR непосредственно для каждой переменной.

Когда объект вызывается циклично в одном и том же процессе, то система в промежутке между вызовами сохраняет в памяти все значения переменных и фильтров. Функция CLEARALL позволит “обнулять” эти значения.

EVALUATE

```
[Ok := ] EVALUATE (Variable, String)
```

Данная функция используется для преобразования строкового выражения в другой подходящий тип данных. Результат помещается в переменную Variable. Данная функция может преобразовывать строковые выражения в типы Decimal, Integer, Date, Time, Boolean, Option, Text и Code.

FORMAT

```
String := FORMAT(Value [, Length] [, FormatNumber |
FormatString])
```

Это очень полезная функция, но потребуется немало времени, чтобы овладеть всеми ее тонкостями. В самом первом приближении она может преобразовывать любой тип переменной к строковой переменной. Следующий оператор меняет десятичную переменную на строковую:

```
TextVar := Format(DecimalVar);
```

Параметр Length (типа Integer) гарантирует, что если значение превышает максимальную длину, допустимую для String, то ошибки этапа выполнения не будет.

- Если Length = 0, то система будет возвращать все значение целиком (это значение по умолчанию).
- Если Length > 0, то String будет содержать в точности Length символов. Если при этом строковое представление значения Value содержит менее Length символов, то система в зависимости от выбранного вами формата вставит предшествующие или замыкающие пробелы. Если строковое представление значения Value содержит более Length символов, то система соответственно усечет String.
- Если Length < 0, то String будет содержать максимум Length символов. Если строковое представление значения Value содержит менее Length символов, то длина строки String будет равна длине Value. Если Value содержит более Length

символов, система соответственно усечет String.

Единовременно используется только один из следующих двух параметров.

FormatNumber

FormatNumber определяет формат представления, который будет использовать система. Основными вариантами являются:

0	Standard Display Format (по умолчанию для всех типов данных)
1	Standard Display Format 2 (редактирование)
2	C/AL Code Constant Format

Другие варианты доступны в зависимости от типа данных. Для более подробной информации см. интерактивную справку.

FormatString

FormatString – это более мощный параметр. Это специальная строка, которая определяет шаблон представления (определенный в свойстве Format). Это свойство описывает подробно различные предопределенные форматы и то, как создавать настраиваемые форматы.

Примеры

```
MESSAGE('The formatted value:  %1', FORMAT(-123456.78,
15, 0));
```

В окне сообщений появится – The formatted value: 123,456.78

```
MESSAGE('The formatted value:  %1', FORMAT(-123456.78,
5, 3));           // Changed length to 5
```

В окне сообщений появится – The formatted value: 123,4

```
MESSAGE('Today is %1', FORMAT(TODAY,0,'<Month Text>
<Day>.');
```

В окне сообщений появится – Today is April 15.

Часто возникает необходимость показывать отрицательные числа как строки в круглых скобках вместо знака минуса. Если они входят в вертикальный список чисел, то возникает потребность добавлять в

конце положительных чисел замыкающий пробел (в том смысле, что отрицательные показатели за счет скобки выравниваются с положительными показателями).

Ниже приведен пример использования функции FORMAT:

```
IF DecimalVar < 0 THEN
   StringVar := FORMAT(DecimalVar, 0, '(<Integer
Thousand><decimals>)')
ELSE
   StringVar := FORMAT (DecimalVar, 0, '<Integer
Thousand><decimals> ')
```

Заметим, что знаки ‘<’ и ‘>’ являются частью параметра FormatString. Для отрицательных чисел круглые скобки заключаются внутрь одинарных кавычек, а для положительных с целью выравнивания перед закрывающей кавычкой вставляется один пробел.

Приложение А

Контрольный тест

Данное приложение дает слушателю курса возможность оценить полученные знания. В приложении представлены вопросы из каждой главы курса. Ответы вы можете найти в приложении В.

Вопросы

1) Что из нижеперечисленного не используется в программном коде?
   a) манипулирование данными;
   b) реализация бизнес-правил;
   c) получение данных (Data Acquisition);
   d) создание ключей в базе данных.

2) Какие объекты в среде C/SIDE имеют C/AL-код?
   a) таблицы
   b) программные модули
   c) отчеты
   d) все объекты содержат C/AL-код

3) Какой тип данных должен использоваться для хранения информации о дне рождения сотрудника?
   a) Date
   b) Integer
   c) Text
   d) Code

4) Какой тип данных должен использоваться для хранения информации об имени рождения сотрудника?
   a) Date
   b) Integer
   c) Text
   d) Code

5) К какому типу данных принадлежит эта константа: 123197 ?
   a) Date
   b) Integer
   c) Text
   d) Code
   e) Это недопустимая константа

6) К какому типу данных принадлежит эта константа: 123197D ?
   a) Date
   b) Integer
   c) Text
   d) Code
   e) Это недопустимая константа

7) Как в программном коде нужно адресоваться к переменной, носящей имя Customer Name?
   a) ‘Customer Name’
   b) Customer_Name
   c) “Customer Name
   d) “Customer Name”
   e) Customer.Name

8) Что появится в результате выполнения следующего оператора?
   `MESSAGE(‘%1’,TRUE);`
   a) Ошибка: Boolean-переменные не отображаются на экране
   b) Окно сообщений с текстом “TRUE”
   c) Окно сообщений с текстом “FALSE”
   d) Окно сообщений с текстом “Yes”

9) Какое значение имеет пустая переменная типа Text?
   a) “”
   b) ‘’
   c) Null
   d) Nil
   e) Ни одно из упомянутых.

10) Какое значение имеет пустая переменная типа Integer?
   a) 0
   b) 1
   c) Null
   d) Nil
   e) Ни одно из упомянутых.

11) Что произойдет, если вы используете оператор A :=15.00, тогда как A является переменной типа Integer?

   a) Возникнет ошибка при попытке откомпилировать данный код.

   b) Возникнет ошибка периода выполнения, поскольку вы пытаетесь присвоить целочисленной переменной значение типа Decimal.

   c) Переменной A будет присвоено значение 15.

   d) Переменная A будет нового типа: Decimal.

12) Что произойдет, если вы используете оператор A :='Word', при том что A является переменной типа Code?

   a) Возникнет ошибка при попытке откомпилировать данный код.

   b) Возникнет ошибка периода выполнения, поскольку вы пытаетесь присвоитьь переменной типа Code значение типа Text.

   c) Переменной A будет присвоено значение 'WORD'.

   d) Переменная A будет нового типа: Text.

13) Что произойдет, если вы используете оператор A :='Word is too long', при том что A является переменной типа Text длиной 4?

   a) Возникнет ошибка при попытке откомпилировать данный код.

   b) Возникнет ошибка периода выполнения, поскольку вы пытаетесь присвоить переменной A слишком много символов.

   c) Переменной A будет присвоено значение 'Word'.

   d) Переменная A увеличит свой размер.

14) Верно ли следующее утверждение: вы можете переменной типа Option присвоить значение –1 ?

   a) да

   b) нет

15) Какая часть оператора C := A + B является выражением?

   a) C := A + B

   b) A + B

   c) C

   d) C :=

16) Что делает операция (+) по отношению к текстовым переменным или константам?

    a) Конкатенирует значения

    b) Не делает ничего, для конкатенации используется операция (&)

    c) В языке C/AL нет возможности конкатенировать строки

17) Каким будет результирующий тип данных для следующего выражения: 57/10

    a) Integer

    b) Decimal

    c) Date

    d) Code

18) Каким будет результирующий тип данных для следующего выражения: 2000000 * 3000

    a) Integer

    b) Decimal

    c) Date

    d) Code

19) Каким будет результирующий тип данных для следующего выражения: 123100D – 010100D

    a) Integer

    b) Decimal

    c) Date

    d) Code

20) Каким будет результирующий тип данных для следующего выражения: 123100D + 21

    a) Integer

    b) Decimal

    c) Date

    d) Code

21) Что из нижеследующего является реляционной операцией (Relational Operator)?

a) =

b) :=

c) DIV

d) NOT

22) Что из нижеследующего является логической операцией?

a) =

b) :=

c) DIV

d) NOT

23) Что из нижеследующего является операцией присваивания?

a) =

b) :=

c) DIV

d) NOT

24) Каким является результирующий тип для реляционной операции?

a) Integer

b) Boolean

c) Зависит от сопоставляемых типов данных

d) Не возвращает никакого типа данных

25) Каково будет значение выражения TRUE XOR FALSE?

a) TRUE

b) FALSE

c) Вы не можете выполнять операцию XOR по отношению к булевым значениям

d) Ни одно из вышеназванных значений

26) Какое из следующих утверждений корректно по отношению к оператору IF THEN ELSE?

   a) После ключевого слова THEN всегда ставится точка с запятой

   b) Перед ключевым словом ELSE всегда ставится точка с запятой

   c) После ключевого слова ELSE всегда ставится точка с запятой

   d) В конце оператора IF THEN ELSE всегда ставится точка с запятой

   e) Ни одно из вышеупомянутых утверждений не является корректным.

27) Какие ключевые слова необходимо использовать, если требуется внутрь оператора IF THEN вставить несколько других операторов?

   a) Не требуется никаких специальных ключевых слов

   b) BEGIN END

   c) START END

   d) Нужны фигурные скобки {} , а не ключевые слова

28) Что из нижеперечисленного является допустимым комментарием?

   a) /* Comment */

   b) ‘Comment

   c) { Comment

   d) //Comment

29) Как получить доступ к 5-ому элементу массива с именем A?

   a) A[4]

   b) A[5]

   c) A(4)

   d) A(5)

30) Верно ли следующее утверждение: массивы всегда создаются динамически?

   a) да

   b) нет

31) Какой оператор цикла вы будете использовать, если вы априори можете определить, сколько раз ваши операторы должны выполняться?

a) оператор FOR

b) оператор WHILE

c) оператор REPEAT

d) этого нельзя сделать

32) Какой оператор цикла не требует наличия BEGIN и END при выполнении в цикле более одного оператора?

a) оператор FOR

b) оператор WHILE

c) оператор REPEAT

d) все они этого требуют

33) Какой оператор цикла проверяет условие в начале каждого цикла?

a) оператор FOR

b) оператор WHILE

c) оператор REPEAT

d) варианты A и B одновременно

e) все упомянутые

34) Какие типы данных могут использоваться с оператором WITH?

a) целые и десятичные

b) переменные типа запись

c) все простые типы данных

d) все составные типы данных

e) ни один из упомянутых вариантов

35) Справедливо ли следующее утверждение: оператор CASE в языке C/AL может использоваться только с тпами данных Integer и Option?

a) верно

b) неверно

36) Что такое локальная переменная?

a) переменная, область видимости которой ограничена рамками одного объекта

b) переменная, область видимости которой ограничена рамками одной функции

c) переменная, область видимости которой ограничена рамками одного программного модуля

d) ни один из упомянутых вариантов

37) Существует два способа передачи параметров функции; какие из них поддерживаются в языке C/AL?

a) передача по целочисленному значению

b) передача по ссылке

c) передача через переменную-запись

d) передача по значению

e) варианты B и C одновременно

38) Где можно задать новую функцию в языке C/AL?

a) пункты меню View, Define Variables

b) пункты меню View, C/AL Globals

c) пункты меню View, C/AL Functions

d) ни один из упомянутых вариантов

39) Что означает аббревиатура C/AL?

a) Client/Appliance Leverage

b) Client/Analyst Language

c) Client/Analyst Library

d) ни один из упомянутых вариантов

40) Что из следующих операторов не является собственно оператором EXIT?

a) EXIT;

b) EXIT(‘Now’);

c) EXIT(Round(123.255, 0.01));

d) IF EXIT THEN X := 1;

Приложение B

Ответы на контрольный тест

Данное приложение дает слушателю курса ответы на вопросы, поставленные в приложении A.

Вопросы

1) Что из нижеперечисленного не используется в программном коде?
   a) манипулирование данными;
   b) реализация бизнес-правил;
   c) получение данных (Data Acquisition);
   d) создание ключей в базе данных.

2) Какие объекты в среде C/SIDE имеют C/AL-код?
   a) таблицы
   b) программные модули
   c) отчеты
   d) все объекты содержат C/AL-код

3) Какой тип данных должен использоваться для хранения информации о дне рождения сотрудника?
   a) Date
   b) Integer
   c) Text
   d) Code

4) Какой тип данных должен использоваться для хранения информации об имени рождения сотрудника?
   a) Date
   b) Integer
   c) Text
   d) Code

5) К какому типу данных принадлежит эта константа: 123197 ?
   a) Date
   b) Integer
   c) Text
   d) Code
   e) это недопустимая константа

6) К какому типу данных принадлежит эта константа: 123197D ?

   a) Date

   b) Integer

   c) Text

   d) Code

   e) это недопустимая константа

7) Как в программном коде нужно адресоваться к переменной, носящей имя Customer Name?

   a) ‘Customer Name’

   b) Customer_Name

   c) “Customer Name

   d) “Customer Name”

   e) Customer.Name

8) Что появится в результате выполнения следующего оператора?
   `MESSAGE(‘%1’,TRUE);`

   a) Ошибка: Boolean-переменные не отображаются на экране

   b) Окно сообщений с текстом “TRUE”

   c) Окно сообщений с текстом “FALSE”

   d) Окно сообщений с текстом “Yes”

9) Какое значение имеет пустая переменная типа Text?

   a) “”

   b) ‘’

   c) Null

   d) Nil

   e) Ни одно из упомянутых.

10) Какое значение имеет пустая переменная типа Integer?

   a) 0

   b) 1

   c) Null

   d) Nil

   e) ни одно из упомянутых

11) Что произойдет, если вы используете оператор A :=15.00, тогда как A является переменной типа Integer?

a) Возникнет ошибка при попытке откомпилировать данный код.

b) Возникнет ошибка периода выполнения, поскольку вы пытаетесь присвоить целочисленной переменной значение типа Decimal.

c) Переменной A будет присвоено значение 15.

d) Переменная A будет нового типа: Decimal.

12) Что произойдет, если вы используете оператор A :='Word', при том что A является переменной типа Code?

a) Возникнет ошибка при попытке откомпилировать данный код.

b) Возникнет ошибка периода выполнения, поскольку вы пытаетесь присвоитьь переменной типа Code значение типа Text.

c) Переменной A будет присвоено значение 'WORD'.

d) Переменная A будет нового типа: Text.

13) Что произойдет, если вы используете оператор A :='Word is too long', при том что A является переменной типа Text длиной 4?

a) Возникнет ошибка при попытке откомпилировать данный код.

b) Возникнет ошибка периода выполнения, поскольку вы пытаетесь присвоить переменной A слишком много символов.

c) Переменной A будет присвоено значение 'Word'.

d) Переменная A увеличит свой размер.

14) Верно ли следующее утверждение: вы можете переменной типа Option присвоить значение –1 ?

a) да

b) нет

15) Какая часть оператора C := A + B является выражением?

a) C := A + B

b) A + B

c) C

d) C :=

16) Что делает операция (+) по отношению к текстовым переменным или константам?

    a) Конкатенирует значения

    b) Не делает ничего, для конкатенации используется операция (&)

    c) В языке C/AL нет возможности конкатенировать строки

17) Каким будет результирующий тип данных для следующего выражения: 57/10

    a) Integer

    b) Decimal

    c) Date

    d) Code

18) Каким будет результирующий тип данных для следующего выражения: 2000000 * 3000

    a) Integer

    b) Decimal

    c) Date

    d) Code

19) Каким будет результирующий тип данных для следующего выражения: 123100D – 010100D

    a) Integer

    b) Decimal

    c) Date

    d) Code

20) Каким будет результирующий тип данных для следующего выражения: 123100D + 21

    a) Integer

    b) Decimal

    c) Date

    d) Code

21) Что из нижеследующего является реляционной операцией (Relational Operator)?

a) =

b) :=

c) DIV

d) NOT

22) Что из нижеследующего является логической операцией?

a) =

b) :=

c) DIV

d) NOT

23) Что из нижеследующего является операцией присваивания?

a) =

b) :=

c) DIV

d) NOT

24) Каким является результирующий тип для реляционной операции?

a) Integer

b) Boolean

c) Зависит от сопоставляемых типов данных

d) Не возвращает никакого типа данных

25) Каково будет значение выражения TRUE XOR FALSE?

a) TRUE

b) FALSE

c) Вы не можете выполнять операцию XOR по отношению к булевым значениям

d) Ни одно из вышеназванных значений

26) Какое из следующих утверждений корректно по отношению к оператору IF THEN ELSE?

a) После ключевого слова THEN всегда ставится точка с запятой

b) Перед ключевым словом ELSE всегда ставится точка с запятой

c) После ключевого слова ELSE всегда ставится точка с запятой

d) В конце оператора IF THEN ELSE всегда ставится точка с запятой

e) Ни одно из вышеупомянутых утверждений не является корректным.

27) Какие ключевые слова необходимо использовать, если требуется внутрь оператора IF THEN вставить несколько других операторов?

a) Не требуется никаких специальных ключевых слов

b) BEGIN END

c) START END

d) Нужны фигурные скобки {} , а не ключевые слова

28) Что из нижеперечисленного является допустимым комментарием?

a) /* Comment */

b) ‘Comment

c) { Comment

d) //Comment

29) Как получить доступ к 5-ому элементу массива с именем A?

a) A[4]

b) A[5]

c) A(4)

d) A(5)

30) Верно ли следующее утверждение: массивы всегда создаются динамически?

a) да

b) нет

31) Какой оператор цикла вы будете использовать, если вы априори можете определить, сколько раз ваши операторы должны выполняться?

a) оператор FOR

b) оператор WHILE

c) оператор REPEAT

d) этого нельзя сделать

32) Какой оператор цикла не требует наличия BEGIN и END при выполнении в цикле более одного оператора?

a) оператор FOR

b) оператор WHILE

c) оператор REPEAT

d) все они этого требуют

33) Какой оператор цикла проверяет условие в начале каждого цикла?

a) оператор FOR

b) оператор WHILE

c) оператор REPEAT

d) варианты A и B одновременно

e) все упомянутые

34) Какие типы данных могут использоваться с оператором WITH?

a) целые и десятичные

b) переменные типа запись

c) все простые типы данных

d) все составные типы данных

e) ни один из упомянутых вариантов

35) Справедливо ли следующее утверждение: оператор CASE в языке C/AL может использоваться только с тпами данных Integer и Option?

a) верно

b) неверно

36) Что такое локальная переменная?

a) переменная, область видимости которой ограничена рамками одного объекта

b) переменная, область видимости которой ограничена рамками одной функции

c) переменная, область видимости которой ограничена рамками одного программного модуля

d) ни один из упомянутых вариантов

37) Существует два способа передачи параметров функции; какие из них поддерживаются в языке C/AL?

a) передача по целочисленному значению

b) передача по ссылке

c) передача через переменную-запись

d) передача по значению

e) варианты B и C одновременно

38) Где можно задать новую функцию в языке C/AL?

a) пункты меню View, Define Variables

b) пункты меню View, C/AL Globals

c) пункты меню View, C/AL Functions

d) ни один из упомянутых вариантов

39) Что означает аббревиатура C/AL?

a) Client/Appliance Leverage

b) Client/Analyst Language

c) Client/Analyst Library

d) ни один из упомянутых вариантов

40) Что из следующих операторов не является собственно оператором EXIT?

a) EXIT;

b) EXIT(‘Now’);

c) EXIT(Round(123.255, 0.01));

d) IF EXIT THEN X := 1;