БИБЛИЯ Delphi

МИХАИЛ ФЛЕНОВ

- Программирование в Delphi от А до Я
- Программирование звука и графики с помощью OpenGL
- Динамические библиотеки
- Создание локальных, клиент-серверных и трехуровневых баз данных
- Практические рекомендации и примеры

bhv

УДК 681.3.068+800.92Delphi
ББК 32.973.26-018.01
Ф69

Фленов М. Е.

Ф69 Библия Delphi. — СПб.: БХВ-Петербург, 2004. — 880 с.: ил.

ISBN 5-94157-456-8

Цель книги — научить читателя строить логику программы и алгоритмы различных вычислений. Уметь программировать еще не достаточно, надо знать, как применять полученные знания на практике. Для этого подробно описывается логика выполнения каждого участка кода, чтобы читатель смог использовать эти знания при решении собственных задач. Книга содержит большое количество примеров практического программирования; некоторые из них вынесены в качестве дополнительной информации на прилагаемый компакт-диск. Электронная версия книги была размещена в Internet в 2003 году. Автор собрал все замечания и предложения по дополнению книги и написал совершенно новый вариант, который вы сейчас держите в руках. Таким образом, книга прошла массовое тестирование и теперь отражает потребности множества как начинающих, так и опытных программистов.

Для программистов

УДК 681.3.068+800.92Delphi
ББК 32.973.26-018.01

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Владимир Красильников*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.02.04.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 70,95.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в ФГУП ордена Трудового Красного Знамени "Техническая книга"
Министерства Российской Федерации по делам печати,
телерадиовещания и средств массовых коммуникаций.
190005, Санкт-Петербург, Измайловский пр., 29.

ISBN 5-94157-456-8

Содержание

Введение **1**

Структура книги 3

Глава 1. Основные принципы работы компьютера **7**

1.1. Основы работы персонального компьютера 7
1.2. Двоичная система работы процессора 8
1.3. Машинный язык 13
1.4. История языков программирования 14
1.5. Исполнение машинных инструкций 19

Глава 2. Машинная математика **23**

2.1. Основы машинной математики 23
2.2. Блок-схемы 25
2.3. Машинная логика и циклы 28
2.4. Программирование машинной логики 30

Глава 3. Начальные сведения о Delphi **33**

3.1. Установка Delphi 7 33
3.2. Замечание по установке в Windows 2000 43
3.3. Оболочка Delphi 6 45
3.4. Главное меню 47
3.5. Настройка Delphi 6 49
3.6. Настройка редактора кода 54
3.7. Настройка оболочки 55

Глава 4. Визуальная модель Delphi **57**

4.1. Процедурное программирование 57
4.2. Объектно-ориентированное программирование 62
4.3. Компонентная модель 67
4.4. Наследственность 68

Глава 5. Основы языка программирования Delphi **71**

5.1 "Hello World" или из чего состоит проект 71
5.2. Язык программирования Delphi 82
5.3. Типы данных в Delphi 89
5.3.1. Целочисленные типы данных 89
5.3.2. Вещественные типы данных 91

5.3.3. Символьные типы данных ... 91
5.3.4. Булевы типы ... 97
5.3.5. Массивы ... 99
5.3.6. Странный *PChar* ... 100
5.3.7. Константы ... 102
5.3.8. Всемогущий *Variant* ... 103
5.4. Процедуры и функции в Delphi ... 104
5.5. Рекурсивный вызов процедур ... 112
5.6. Встроенные процедуры ... 115
5.7. Возврат значений через параметры ... 116
5.8. Перегрузка ... 117
5.9. Методы объектов ... 119
5.10. Наследование объектов ... 120

Глава 6. Работа с компонентами ... 123

6.1. Основная форма и ее свойства ... 123
6.2. Событийная модель Windows ... 134
6.3. События главной формы ... 136
6.4. Палитра компонентов ... 137

Глава 7. Палитра компонентов *Standard* ... 139

7.1. Кнопка *(TButton)* ... 139
7.2. Изменение свойств кнопки (логические операции) ... 143
7.3. Надписи *(TLabel)* ... 148
7.4. Строки ввода *(TEdit)* ... 149
7.5. Многострочное поле ввода *(TMemo)* ... 151
7.6. Объект *TStrings* ... 156
7.6.1. Свойства объекта *TStrings* ... 156
7.6.2. Методы объекта *TStrings* ... 157
7.7. Компонент *CheckBox* ... 158
7.8. Панели *(TPanel)* ... 159
7.9. Кнопки выбора *(TRadioButton)* ... 161
7.10. Списки выбора *(TListBox)* ... 162
7.11. Выпадающие списки *(TComboBox)* ... 165
7.12. Полосы прокрутки *(TScrollBar)* ... 167
7.13. Группировка объектов *(TGroupBox)* ... 168
7.14. Группа компонентов *RadioButton (TRadioGroup)* ... 169
7.15. Ответы на вопросы ... 171

Глава 8. Учимся программировать ... 173

8.1. Циклы *for ... to ... do* ... 173
8.2. Циклы *while* ... 177
8.3. Циклы *repeat* ... 179
8.4. Управление циклами ... 180
8.5. Логические операторы ... 184

8.6. Работа со строками 188
8.6.1. Функция *Length* 188
8.6.2. Функция *Copy* 189
8.6.3. Функция *Delete* 190
8.6.4. Функция *Pos* 190
8.6.5. Функция *Insert* 191
8.7. Исключительные ситуации 191

Глава 9. Создание рабочих приложений 195

9.1. Создание главного меню программы 195
9.2. Создание дочерних окон 200
9.3. Модальные и немодальные окна 204
9.4. Обмен данными между формами 206
9.5. Многодокументные MDI-окна 208
9.6. Инициализация окон 212

Глава 10. Основные приемы программирования 219

10.1. Работа с массивами (динамические массивы) 219
10.2. Многомерные массивы 225
10.3. Работа с файлами 227
10.4. Работа с текстовыми файлами 231
10.5. Приведение типов 235
10.5.1. Преобразование целых чисел в строку и обратно 235
10.5.2. Преобразование даты в строку и обратно 237
10.5.3. Преобразование вещественных чисел 238
10.6. Преобразование совместимых типов (преобразование строк) 240
10.7. Указатели 240
10.8. Структуры, записи 243
10.9. Храним структуры в динамической памяти 247
10.10. Поиск файлов 249
10.11. Работа с системным реестром 252
10.12. Множества 259
10.13. Потоки 261

Глава 11. Обзор дополнительных компонентов Delphi 265

11.1. Дополнительные кнопки Delphi (*TSpeedButton* и *TBitBtn*) 265
11.2. Самостоятельная подготовка картинок для кнопок 271
11.3. Маскированная строка ввода *(TMaskEdit)* 272
11.4. Сеточки (*TStringGrid*, *TDrawGrid*) 273
11.5. Компоненты-украшения (*TImage*, *TShape*, *TBevel*) 281
11.6. Панель с полосами прокрутки *(TScrollBox)* 285
11.7. Маркированный список *(TCheckListBox)* 285
11.8. Полоса разделения *(TSplitter)* 287
11.9. Многострочный текст *(TStaticText)* 289
11.10. Редактор параметров *(TValueListEditor)* 289

11.11. Набор вкладок *(TTabControl)* 292
11.12. Набор страниц *(TPageControl)* 298
11.13. Набор картинок *(TImageList)* 300
11.14. Ползунки *(TTrackBar)* 300
11.15. Индикация состояния процесса *(TProgressBar)* 302
11.16. Простейшая анимация *(TAnimate)* 305
11.17. Выпадающий список выбора даты *(TDateTimePicker)* 307
11.18. Календарь *(TMonthCalendar)* 308
11.19. Дерево элементов *(TTreeView)* 308
11.20. Профессиональное использование компонента *TreeView* 314
11.21. Список элементов *(TListView)* 319
11.22. Простейший файловый менеджер 321
11.23. Улучшенный файловый менеджер (с возможностью запуска файлов) 333
11.24. Подсказки для чайников *(TStatusBar)* 335
11.25. Панель инструментов (*TToolBar* и *TControlBar*) 338
11.26. Перемещаемые панели и меню в стиле MS (Docking) 341
11.27. Меню и панели в стиле XP 345
11.28. Всплывающее меню в стиле XP 351

Глава 12. Графические возможности Delphi 353

12.1. Графическая система Windows 353
12.2. Первый пример работы с графикой 355
12.3. Свойства карандаша 357
12.4. Свойства кисти 361
12.5. Работа с текстом в графическом режиме 366
12.6. Вывод текста под углом 368
12.7. Работа с цветом 373
12.8. Методы объекта *TCanvas* 378
12.8.1. *Pixels* 378
12.8.2. *TextWidth* и *TextHeight* 379
12.8.3. *Arc* 379
12.8.4. *CopyRect* 379
12.8.5. *Draw* 380
12.8.6. *Ellipse* 381
12.8.7. *FillRect* 381
12.8.8. *FloodFill* 381
12.9. Компонент работы с графическими файлами *(TImage)* 381
12.10. Рисование на стандартных компонентах 386
12.11. Работа с экраном 391
12.12. Режимы рисования 393

Глава 13. Печать в Delphi 401

13.1. Объект *TPrinter* 401
13.2. Получение информации об установленном принтере 405
13.3. Текстовая печать 409

13.4. Печать содержимого формы 411
13.5. Вывод на печать изображения 416
13.6. Еще немного о печати 420

Глава 14. Delphi и базы данных 423

14.1. Теория реляционных баз данных 424
14.1.1. Локальные базы данных 425
14.1.2. Delphi и базы данных 427
14.2. Создание первой базы данных Access 428
14.3. Пример работы с базами данных 432
14.3.1. Свойства компонента *TADOTable* 437
14.3.2. Методы компонента *TADOTable* 439
14.4. Управление отображением данных 441
14.5. Поисковые поля 448
14.6. Улучшенный пример с поисковыми полями 456
14.7. Сортировка 459
14.8. Фильтрация данных 461
14.9. Язык запросов SQL 465
14.10. Связанные таблицы 473
14.11. Вычисляемые поля 479
14.12. Цветные сетки *DBGrid* 482
14.13. Подключение к базе данных во время выполнения программы 487
14.14. Расширения ADO 489
14.15. Обработка базы данных 496
14.16. Бинарные данные 499
14.17. События таблицы 504

Глава 15. Создание отчетности 507

15.1. Создание отчетности в Excel 508
15.2. Отчетность в Quick Reports 517
15.3. Печать таблиц с помощью Quick Reports 523
15.4. Печать связанных таблиц 525
15.5. Дополнительные возможности 527

Глава 16. Работа с DBF, Paradox, XML и клиент-серверными базами данных 529

16.1. Создание таблицы Paradox 529
16.2. Русификация таблиц Paradox и DBF 536
16.3. Быстрый поиск 537
16.4. Создание псевдонимов 539
16.5. Работа с XML-таблицами 542
16.6. Теория клиент-серверных баз данных 544
16.7. Пример работы с SQL Server 546
16.8. Многоуровневые приложения для баз данных 553
16.8.1. Реализация сервера бизнес-логики 555
16.8.2. Клиент для бизнес-логики 559

Глава 17. Потоки 565

17.1. Теория потоков 565
17.2. Простейший поток 567
17.3. Дополнительные возможности потоков 572
17.4. Подробней о синхронизации 574

Глава 18. Динамически компонуемые библиотеки 577

18.1. Что такое DLL 577
18.1.1. Решение № 1 577
18.1.2. Проблема № 1 578
18.1.3. Проблема № 2 578
18.1.4. Решение № 2 579
18.1.5. Из чего сделан Windows 581
18.1.6. Графические движки 581
18.2. Простой пример создания DLL 583
18.3. Замечания по использованию библиотек 587
18.4. Хранение формы в динамических библиотеках 588
18.5. Немодальные окна в динамических библиотеках 592
18.6. Явная загрузка библиотек 595
18.7. Точка входа 597
18.8. Вызов из библиотек процедур основной программы 599

Глава 19. Разработка собственных компонентов 603

19.1. Пакеты 604
19.2. Подготовка к созданию компонента 611
19.3. Создание первого компонента 615
19.4. Создание иконки компонента 626
19.5. События в компонентах 628

Глава 20. Мультимедиа 631

20.1. Простейшие способы проигрывания звука 631
20.2. Медиа-проигрыватель средствами Delphi 635
20.3. Звук без компонентов 640
20.4. Формат звукового файла WAV 647
20.5. Пример воспроизведения WAV-файла 648
20.6. Выбор устройства воспроизведения или записи 663
20.7. Функции записи звука 668
20.8. Преобразование форматов данных 672
20.9. Пример преобразования форматов данных 683

Глава 21. Графика OpenGL 691

21.1. Инициализация и отображения 2D-графики 691
21.2. Третье измерение и тест глубины 700
21.3. Реалистичное изображение (Туман) 705

21.4. Примитивная графика 708
21.5. Генерация собственных примитивов, масштабирование, перемещение объектов....716
21.6. Примитивы библиотеки GLU....722
21.7. Текстуры 726
21.8. Освещение....733
21.9. Заключение 737

Глава 22. OLE, COM, ActiveX....739

22.1. Теория OLE....739
22.2. OLE-контейнер....742
22.3. Создание собственного окна вставки OLE-объекта....747
22.4. Элементы управления ActiveX 751
22.5. Модель COM 758
22.6. Пример создания ActiveX-форм 761
22.7. Создание компонентов ActiveX 765

Глава 23. Буфер обмена 773

23.1. Буфер обмена и стандартные компоненты Delphi 773
23.2. Объект *Clipboard* 775
23.3 Картинки и буфер обмена....777
23.4. Создание собственного формата для работы с буфером 782

Глава 24. Дополнительная информация 791

24.1. Тестирование и отладка....791
24.2. Работа с редактором....799
24.2.1. Закладки 799
24.2.2. Копирование строк....800
24.2.3. Code Explorer....800
24.2.4. Редактор кода....802
24.3. Создание программ инсталляции 802
24.4. Как писать и распространять Shareware-программы....816

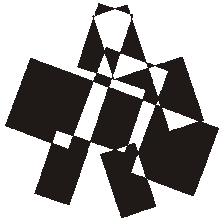
Глава 25. Сплошная практика 821

25.1. Создание Screen Saver....821
25.2. Компоненты в runtime....827
25.3. Тест на прочность 833
25.4. Сохранение и загрузка теста 848
25.5. Тестер....852

Приложение. Описание компакт-диска 860

Предметный указатель 861

Глава 2



Машинная математика

До начала перестройки в нашей стране довольно мало внимания уделялось подготовке программистов. Большинство их были выходцами с кафедр математики, на которых очень часто изучались определенные учебные дисциплины с уклоном в сторону информатики. На этих курсах учили писать *блок*-схемы, которые в свою очередь помогали понять логику работы программы.

С *блок*-схемами знакомятся на начальном этапе обучения программированию. Первое впечатление тех, кто столкнулся с *блок*-схемами, — абсолютно ненужная ерунда. Однако со временем становятся понятны их достоинства. Возможно, что и вам покажется это слишком просто, но все же желательно прочитать эту главу полностью. Здесь рассматривается теория всего процесса программирования, что позволяет понять логику выполнения команд в процессе реализации программы на процессоре. В дальнейшем останется только познакомитюся с практикой.

2.1. Основы машинной математики

В любом языке программирования можно выполнять математические операции любой сложности. Delphi в этом вопросе не исключение. Но пока мы не будем рассматривать все возможности этого языка в данной области, а остановимся только на математических операциях, чтобы сразу привыкать к математике Delphi. В табл. 2.1 представлены основные математические операции языка Delphi.

Таблица 2.1. *Основные математические операции Delphi*

Математическая операция	Описание	Математическая операция	Описание
`*`	Умножить	`+`	Сложение
`/`	Разделить	`-`	Вычитание
`Sqr`	Квадрат	`:=`	Присвоить значение
`Sqrt`	Квадратный корень		

Все эти операции выполняются в том же порядке, в котором перечислены. При этом соблюдается приоритет операций (приоритет умножения, деления и т. д. выше приоритета сложения и вычитания). Например, результатом вычисления выражения `2+2*2` будет 6, потому что сначала выполняется операция умножения, а потом сложения. Если вы хотите сначала выполнить сложение, а потом вычитание, то, как и в простой математике, нужно использовать скобки `(2 +2)*2=8`. В этом случае результат будет совершенно другим.

Для изучения компьютерной математики необходимо уяснить ряд понятий, и начнем мы с понятия *переменной*.

Переменная — это ячейка оперативной памяти, в которую можно записывать различные значения. Чаще всего этой ячейке памяти ставится в соответствие какое-нибудь имя. Например, можно определить переменную с именем `F`. Ей, в свою очередь, можно присваивать значения, например 5. Для этого достаточно записать выражение `F:=5`. Последовательность символов — знак двоеточия и равно — означает здесь операцию "присвоить".

Почему для присвоения значения используется именно `:=`, тогда как мы привыкли к простому знаку равенства? Это необходимо, чтобы отделить операцию присваивания от операции сравнения. Знак равенства в Delphi используется для сравнения чисел, а `:=` для присваивания значения переменным.

Значения переменных можно копировать. Допустим, имеется еще одна переменная `G`. Ей можно присвоить значение переменной `F` с помощью простого присваивания `G:=F`. После этого в переменной `G` тоже будет значение 5.

Переменной можно присваивать результаты каких-то вычислений, например: `F:=10/2`. Это достаточно простой пример. А вот уже целое выражение с использованием переменных:

```
F:=5;
G:=10;
F:=G/2.
```

Имя переменной может состоять как из одной, так и из нескольких букв. Например, переменная может иметь имя `Str` или `MyPeremen`. Единственное ограничение, которое следует здесь учитывать — это то, что имя должно состоять из английских букв и не должно содержать зарезервированные слова (о зарезервированных словах будет сказано немного позже). Вы также можете в имени переменной использовать числа (желательно в конце), например `Str1`, `Str2`, `Str3` и т. д.

Совет

Назначайте переменным осмысленные имена. Когда вы начнете писать большие программы, тяжело будет разобраться, что означает переменная, например, `i` или `b`.

Тип переменной — это тип значения, которое можно присвоить переменной. Очень часто используют термин *тип данных*, потому что это действительно тип данных, хранящихся в переменной. Он показывает, какого типа информация может быть присвоена переменной. В Delphi принято обязательно указывать типы переменных, чтобы сразу можно было определить, какую информацию можно туда записать, а какую нет.

Существует несколько основных типов переменных, которые на данный момент времени необходимо четко представлять.

В табл. 2.2 приведены только основные типы данных. Реально их намного больше. Когда мы перейдем к программированию, вы познакомитесь с большим количеством типов.

Таблица 2.2. *Основные типы данных в Delphi*

Название типа	Описание	Дополнительная информация
`Integer`	Целое число	Переменная этого типа может принимать в качестве значения любые целые числа, как положительные, так и отрицательные
`Real`	Вещественное число	Переменная этого типа может принимать в качестве значения целые и дробные числа со знаком и без
`String`	Строка	Переменная этого типа может принимать в качестве значения любые символы и наборы символов
`Boolean`	Булево значение	Переменная может принимать значение `true` или `false` (истина или ложь). Этот тип очень часто используется для организации логики

Строки — это любые символы или наборы символов. В языке Delphi они выделяются одинарными кавычками, например, `'Привет'`. Строки так же можно присваивать переменным, как и любое другое значение. Например:

```
Str — строковая переменная.
Str:='Привет!!!'
```

На этом основные сведения о машинной математике подошли к концу. Пора применить полученные знания на практике.

2.2. Блок-схемы

Давайте сразу зададим какой-нибудь простой пример, на котором попробуем определить логику его решения на процессоре. Допустим, нам надо получить произведение двух чисел.

В логике человека мы должны выполнить следующие операции:

Листинг 2.1. Произведение двух чисел

```
Старт.
Ввести число 1.
Ввести число 2.
Умножить число 1 на число 2.
Вывести результат.
```

Это простейшая и подробная логика, которой оперирует человек. Но машина так не может мыслить и по ее логике нужно рассуждать немного иначе. Для отображения машинной логики определение вычислительных шагов неудобно, потому что решение может быть извилистым, а не прямолинейным. Поэтому давайте знакомиться с блок-схемами на этой линейной задаче.

Блок-схемы принято чертить различными квадратами, овалами и прямоугольниками. Я особо не буду придерживаться стандартов, потому что это непринципиально в решении, но некоторых особенностей мы будем придерживаться. Основные типы блоков, которые используются для формирования блок-схемы, можно увидеть на рис. 2.1.

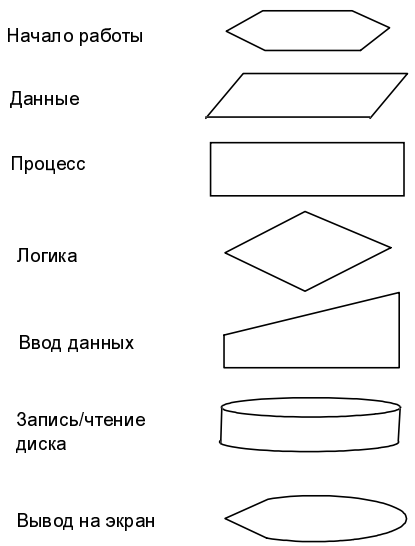


Рис. 2.1. Основные фигуры блок-схемы

Есть и другие, более изощренные блоки, но ими мы не будем пользоваться. Большинство блоков будут оформляться, как простой прямоугольник, потому что от формы блока суть особо не изменится. Самое главное (на мой взгляд) — выделить отдельным видом блока начало блок-схемы и ее логику. Все остальное можно оформить однообразно, наглядность от этого пострадает, но не сильно.

Итак, наша первая блок-схема умножения двух чисел будет выглядеть так, как это показано на рис. 2.2.

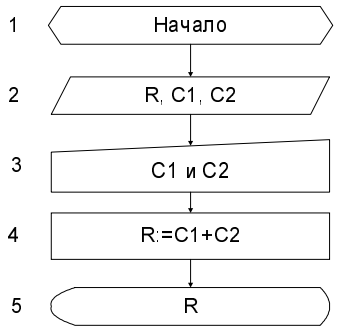


Рис. 2.2. Блок-схема умножения двух чисел

На первый взгляд все слишком сложно. Но это только первый взгляд. Реально здесь ничего сложного нет, просто очень громоздко и простейшая операция перемножения чисел превращается в несколько операций. Но все же надо объяснить происходящее подробнее, чтобы мы могли продвинуться дальше и разобраться с более сложными примерами. Для этого дадим характеристику каждого блока в отдельности.

- *Первый блок* — это начало. Если вы собираетесь строить блок-схемы, то обязательно указывай его, чтобы сразу можно было увидеть начало логики.
- *Второй блок* — перечисляет переменные, которые нам нужны для вычислений. В примере используются три переменные `R`, `C1` и `C2`. В переменную `R` будет помещен результат вычисления. Переменные `C1` и `C2` используются для хранения введенных данных. Здесь пока не указывается тип этих данных, но подразумевается, что это будут или целые числа, или вещественные.
- *Третий блок* — определяет ввод исходных значений переменных. Здесь показывается, что надо ввести значения переменных `C1` и `C2`.
- *Четвертый блок* — указывает на необходимость произвести умножение `C1` на `C2` и результат записать в переменную `R`.
- *Пятый блок* — вывод результата на экран.

Это простая блок-схема, в которой нет ничего особенного. Следующие примеры будут использовать уже логику, поэтому и блок-схемы будут более сложными. Однако на этих примерах вы сможете ощутить всю прелесть машинной математики и понять ее логику.

2.3. Машинная логика и циклы

В любом языке программирования есть множество операций сравнения, которые позволяют реализовать машинную логику. Что понимается под логическими операциями? Это операции сравнения на "равенство", "больше" или "меньше". Как показано в табл. 2.3, таких операций может быть достаточно много.

Таблица 2.3. *Логические операции*

Математическая операция	Описание	Математическая операция	Описание
=	Равно	>=	Больше либо равно
>	Больше	<=	Меньше либо равно
<	Меньше	<>	Не равно

Давайте рассмотрим простейшую логику компьютера на примере расчета значения факториала. Факториал — это произведение чисел от 1 до какого-то числа. Например, факториал 5 равен $1 \times 2 \times 3 \times 4 \times 5 = 120$. Факториал обозначается как знак восклицания "!".

Давайте напишем алгоритм в виде блок-схемы (рис. 2.3).

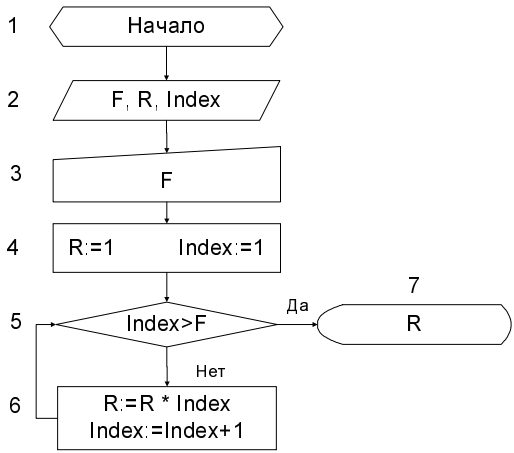


Рис. 2.3. Блок-схема расчета факториала

Представим, что мы не знаем числа, факториал которого мы должны вычислить. Пусть это число будет вводить пользователь. Поэтому расчетная формула выглядит, как число F факториал (F!). В этом случае надо перемножить все числа от 1 до F, т. е. $1 \times 2 \times 3 \times *4 \times ** \times$ F. Перемножение можно делать последовательно. Умножить 1×1. Затем проверить, не превысили ли мы число F. Если нет, то результат прошлого вычисления надо умножить на 2 и снова сделать проверку. Если не превысили F, то снова умножить результат прошлого вычисления на 3. И т. д.

Теперь постараемся объяснить назначение каждого блока в блок-схеме, чтобы было понятно, как она работает. В данном случае это очень важно. Работа блок-схемы очень похожа на то, как "мыслит" компьютер, выполняя заданную программу. Именно так вы должны будете размышлять, когда начнете писать свои первые программы.

Итак, давайте рассмотрим назначение каждого блока в отдельности:

1. Это начало.
2. Объявление трех переменных: `F`, `R` и `Index`. В переменной `F` будет храниться число, факториал которого надо вычислить. Переменная `R` используется для хранения результата вычислений. И наконец, в переменной `Index` — будет содержаться счетчик вычислений.
3. В этом блоке происходит ввод числа, факториал которого надо вычислить. Если пользователь ввел 5, то нам нужно вычислить 5!.
4. Здесь задаются начальные значения переменным. На этом этапе устанавливается значение счетчика и начальный результат вычислений равными 1. Почему именно 1? Да потому, что нам нужно перемножить все числа, начиная от 1, до числа, факториал которого вычисляется.
5. В этом блоке проверяется счетчик (`Index`). Проверка заключется в определении, превысило ли его значение число, которое ввел пользователь, или нет? Если "нет", надо перейти на блок 6. Допустим, пользователь ввел число 5. Значение счетчика пока равно 1. 1 меньше 5, значит, должен произойти переход на блок 6.
6. Здесь вычисляеся результат `R:=R*Index`. На этом этапе `Index = 1` и `R = 1`. Значит, `R:=1*1`. Результат будет 1, а значит, в `R` опять будет единица. Далее, мы увеличиваем `Index` на 1 (`Index:=Index+1`), после чего он становится равным 2 (`Index:=1+1`), и снова возвращаемся на блок 5. В нем опять происходит проверка `Index>F`. `Index` сейчас равен 2, а это меньше пяти. Значит, снова идем на блок 6. Здесь опять производится расчет `R:=R*Index` (`R:=1*2`), после чего `R` становится равным 2. Опять увеличиваем счетчик (`Index:=Index+1`), и `Index` становится равным 3. Снова переход на блок 5.

Я не стал дальше расписывать порядок решения данной задачи. Попробуйте сами пройтись по этой блок-схеме до самого конца. Очень важно понять, как она работает.

В описанной блок-схеме была задействована очень интересная и удобная форма организации вычислений, а именно организация цикла. *Цикл* — повторяющееся выполнение какого-либо блока. В данном случае несколько раз выполняется шестой блок. Если бы число факториала было известно заранее, можно было бы упростить блок-схему, написав формулу типа `R:=1*2*3*4*5`. Но мы не знаем число, которое введет пользователь. Поэтому приходится делать цикл и на каждом его этапе вычислять промежуточный результат.

Попробуйте сами написать блок-схему арифметической прогрессии. Ее формула достаточно проста. В этом случае можно модифицировать блок-схему, которая показана на рис. 2.3. Разница заключается только в том, что в арифметической прогрессии рассчитывается сумма, а не произведение чисел от 1 до определенного числа F.

Нарисуйте эту блок-схему на бумаге и попробуйте мысленно пройтись по ней, выполняя каждый шаг, как это делала бы машина. Если вы думаете, что блок-схемы слишком просты, попробуйте написать что-нибудь более сложное. Это необходимо сделать с целью научиться мыслить как компьютер. Только так вы сможете объяснить ему, что от него требуется, и что самое главное, он вас сможет понять.

2.4. Программирование машинной логики

Подошло время превратить нашу логику, описанную в блок-схеме на рис. 2.3, в настоящую программу. Пока эта программа будет существовать только на бумаге, но со временем ее можно превратить в настоящий исполняемый модуль.

Сначала напишем нашу программу на русском языке.

Листинг 2.2. Пример программы, описанной понятным человеку языком

```
Начало программы.

Переменные:
 F, R, Index - это целые числа;

Начало кода
 F:=5;
 R:=1;
 Index:=1;
```

```
От 1 до 5 выполнять
  Начало цикла
    R:=R*Index;
    Index:=Index+1;
  Конец цикла

  Вывести на экран переменную R.
Конец кода
```

Если не обращать внимания на все, что написано русским языком, можно считать, что первая программа уже написана. В принципе, программа на любом языке программирования выглядит приблизительно так, как это показано выше, только при программировании действуют жесткие правила описания команд. В данном случае наша программа состоит из следующих блоков:

- начало программы;
- описание переменных;
- начало кода (учтите, что описание переменных — это не код программы);
- заполнение переменных начальными значениями;
- запуск цикла от 1 до 5;
- выполнение в цикле расчета;
- вывод результата.

В принципе, ничего нового здесь нет. Просто блок-схема описана словами, похожими на программный язык. В дальнейшем, когда вы сами начнете писать свой собственный программный код, действовать будете точно так же. Сначала постройте алгоритм программы в виде блок-схемы или хотя бы мысленно представьте алгоритм работы будущей программы и только потом переносите все это в компьютер. Блок-схемы понадобятся только на начальном этапе, когда вы не можете сразу представить себе действия, которые нужно запрограммировать. Со временем блок-схемы отойдут сами собой, и уже через месяц практического программирования можно будет писать алгоритмы сразу же на Delphi. Все дело в практике.

При написании программы необходимо мыслить точно так же, как это делает машина. В противном случае получится аналогично тому, как если бы вы разговаривали с англичанином на английском языке, но терминами русского языка.

Совет

Мало знать английские слова, нужно еще и мыслить как англичанин. Я сразу вспоминаю случай, который приключился со мной в Испании. Как-то раз ко мне

подошла девушка и, узнав, что я из России, стала просить у меня деревянную ложку в качестве сувенира. У меня не было ложек, я просто не брал их с собой. Поэтому пришлось сказать ей "No", т. е. "Нет". Я это воспринимал, как ответ: "У меня НЕТ ложки". Она меня спрашивала, почему "НЕТ", понимая мой ответ так, как будто я отказываюсь дать ей эту ложку. На английском языке "NO" — это отрицание, а по нашему "Нет" — это не только отрицание, но и признак отсутствия (например, у меня нет ложки). Если бы я тогда мыслил, как англичанин, то смог бы ответить: "I don't have". Но этому я научился намного позже, тогда я еще очень слабо владел английским языком.

В большинстве случаев машинный язык схож с человеческим, потому что его создавали люди. Но иногда разница чувствительна, потому что машина не всегда может мыслить, как человек. В связи с этим необходимо научиться объяснять свои мысли понятным для машины языком.

Теперь мы готовы перейти к изучению программирования.