



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Прикладная математика»

Программирование в Delphi: принятие решений

Методические указания к лабораторной работе № 3
по курсам «Информатика», «Алгоритмические языки
и программирование»

Автор
Е.Н. Ладоса, Д.С. Цымбалов, О.В. Яценко,
Е.О. Власова

Ростов-на-Дону, 2018



Аннотация

Описаны способы и особенности реализации условных операторов Object Pascal. Целью работы ставится усвоение студентами программирования сложных алгоритмов с ветвлением и необязательным выполнением фрагментов. Предназначены для студентов всех специальностей факультета «Информатика и вычислительная техника».

Автор

Доцент, к.т.н.
Ладоша Е.Н.

Старший преподаватель кафедры
«Электроника и электротехника»
Цымбалов Д.С.

Доцент, к.ф.-м.н.
Яценко О.В.

Студент ДГТУ
Власова Е.О.



Цель работы

Цель работы сводится к освоению так называемых **структур принятия решений**, которые применяются для сравнения двух или большего количества выражений. Рассматриваются операторы сравнения, булевы операторы, и изучаются два вида структур: 1) условный оператор `if` 2) и оператор `case`.

Сравнение

Чтобы понять, как устроены структуры принятия решений, нужно сначала ознакомиться с различными операторами, входящими в их состав. Это **операторы сравнения** и **булевы** (или **логические**) **операторы**.

Операторы сравнения

Для любых двух выражений A и B одного и того же типа всегда справедливо одно из трех следующих условий: $A < B$, $A = B$ или $A > B$. Формально выражение определяется как любая правильная последовательность операторов и операндов, результатом которой является строка, переменная или объект. Для сравнения выражений используются операторы сравнения. Как и в математике, в программировании операторы $<$, $=$ и $>$ можно объединять в операторы $<$, $>$ и \neq . В таблице 1 перечислены операторы сравнения Object Pascal и их математические аналоги. Эти операторы сравнения используются в большинстве языков высокого уровня

Результатом операции сравнения является **булево выражение**, которое может принимать значение `True` или `False` (истина (1) или ложь (0)). Результат булева выражения, содержащего числовые выражения, очевиден. Например, булево выражение $5 < 2$ имеет значение `False`, а выражение $5 > 2$ – `True`, при сравнении строковых выражений фактически сравниваются числовые значения ASCII их символов. Например, значение ASCII символа `A` равно 65, а символа `a` – 97, поэтому значение булева выражения `'America' < 'america'` равно `True`.

Еще один пример: значение булева выражения `'America' < 'AMERICA'` равно `False`. Можете ли вы сказать, почему?

Таблица 1

Операторы сравнения Object Pascal

| Операция сравнения | Оператор сравнения | Математический аналог |
|--------------------|--------------------|-----------------------|
| Меньше или равно | <code><=</code> | \leq |
| Больше | <code>></code> | $>$ |

| | | |
|------------------|--------|--------|
| Больше или равно | \geq | \geq |
| Равно | $=$ | $=$ |
| Не равно | \neq | \neq |
| Меньше | $<$ | $<$ |

Булевы операторы

Операндами булевых операторов являются булевы (логические) выражения. Три первичных булевых оператора – `and`, `or` и `not`. Все другие булевы операторы выводятся из первичных, т.е. могут быть заменены комбинацией первичных операторов. Результаты булевых операций принято отображать в **таблицах истинности** (см. табл. 2).

Таблица 2

Таблица истинности булевых операторов `and`, `or` и `not`

| A | B | A and B | A or B | not A | not B |
|-------|-------|---------|--------|-------|-------|
| False | False | False | False | True | True |
| False | True | False | True | True | False |
| True | False | False | True | False | True |
| True | True | True | True | False | False |

Как видно из таблицы 2, оператор `and` выполняет **логическую конъюнкцию** двух выражений. Это значит, что результат имеет значение `True`, только если оба операнда равны `True`. Оператор `or` выполняет **логическую дизъюнкцию** двух выражений: результат равен `True`, если хотя бы один из операндов равен `True`. Оператор `not` выполняет **логическое отрицание**: значение результата операции противоположно значению операнда.

Компилятор Delphi поддерживает два режима вычисления булевых выражений: режим **полного** и режим **неполного вычисления**. Эти режимы относятся к операторам `and` и `or`. В режиме полного вычисления компилятор продолжает вычисление каждой конъюнкции и дизъюнкции, даже если результат всего выражения уже известен. Пусть, например, результирующее выражение равно `A or B`, где `A` и `B` – некоторые выражения. Допустим, значение выражения `A` равно `True`. Тогда выражение `B` можно не вычислять, так как уже известно, что результирующее выражение истинно. Тем не менее в режиме полного вычисления процессор обрабатывает оба выражения – как `A`, так и `B`. В режиме неполного вычисления компилятор конструирует процесс таким образом, что вычисление прекращается, как только становится известным результат всего выражения. Следовательно, в этом примере, если `A` истинно, то значение `B` не вычисляется. Применение режима полного вычисления обязательно, когда один из операндов – функция, дополнительно выполняющая какие-либо другие действия. Обычно режим неполного вычисления является предпочтительным, так как в этом случае



выполняемый код работает быстрее и занимает в памяти меньше места. Кроме того, в режиме неполного вычисления можно создавать конструкции, которые в режиме полного вычисления вызвали бы во время выполнения ошибку.

По умолчанию компилятор установлен в режим неполного вычисления. Директива этого режима имеет вид `{ $B- }` или `{ $BOOLEVAL off }`. Для локального включения режима полного вычисления можно использовать директиву `{ $B+ }` или `{ $BOOLEVAL ON }`. В среде Delphi компилятор можно переключить в режим полного вычисления для всего проекта. Для этого нужно выбрать команду `Projects → Options...` и в разделе `Syntax options` (Параметры синтаксиса) вкладки `Compiler` (Компилятор) установить флажок `Complete Boolean eval` (Полное булево вычисление).

Следующий пример выполните сначала в режиме неполного вычисления, а затем в режиме полного вычисления. Прокомментируйте, что произошло и в чем отличие первого запуска программы от второго.

```
program Project2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var
  byDemon, byNumber : byte;
  bValue : boolean;
begin
  byDemon:=0;
  byNumber:=1;
  bValue:=(byDemon<>0) and ( (byNumber/byDemon)>1) ;
  writeln(value);
  readln;
end.
```

Условные операторы

В Object Pascal есть два условных оператора: `if` и `case`.

Операторы `if`

Условный оператор `if` (если) очень похож на сослагательное наклонение в естественном языке. Рассмотрим следующее предложение с ключевым словом "если":

В Object Pascal, как и в большинстве других языков высокого уровня, оператор `if` составляет основу структур принятия решений. Синтаксис простой формы оператора `if`, называемой `if-then`, имеет вид

```
if выражение then begin
  [операторы];
```

end;

Все операторы между ключевыми словами `begin` и `end` выполняются, только если значение выражения условие равно `True`. *Выражение* условие должно иметь булев тип. На рис. 1 показана блок-схема оператора `if-then`.

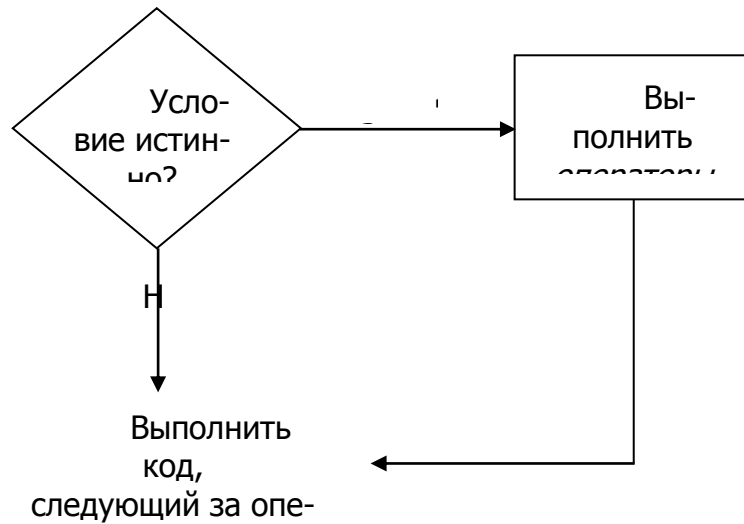


Рис. 2. Блок-схема оператора `if-then`

Оператор `if-then`, не содержащий операторов, выполняющихся при истинности условия, называется пустым оператором `if`. Такие операторы бесполезны и должны быть удалены из кода.

Если оператор `if-then` содержит только один выполняемый оператор, то заключать его в блок с помощью ключевых слов `begin` и `end` не обязательно. Так, предыдущий пример можно записать следующим образом:

```
if (difference >= 10) then handicap := 5;
```

Однако заключение единственного оператора в блок считается хорошим стилем программирования, так как это делает код более понятным. Кроме того, это уменьшает вероятность синтаксических ошибок, когда в будущем программист решит добавить несколько выполняемых операторов.

В некоторых случаях необходима более сложная структура принятия решения `if-then-else`, синтаксис которого имеет вид.

```
if выражение then begin
    [операторы1;]
end
else begin
    [операторы2;]
end;
```

На рис. 2 показана блок-схема простой формы оператора `if-then-else`. Группа операторов *операторы1* выполняется, только когда *выражение* условие имеет значение `True`. Операторы *операторы2* выполняются, только если *вы-*

ражение условие имеет значение False.

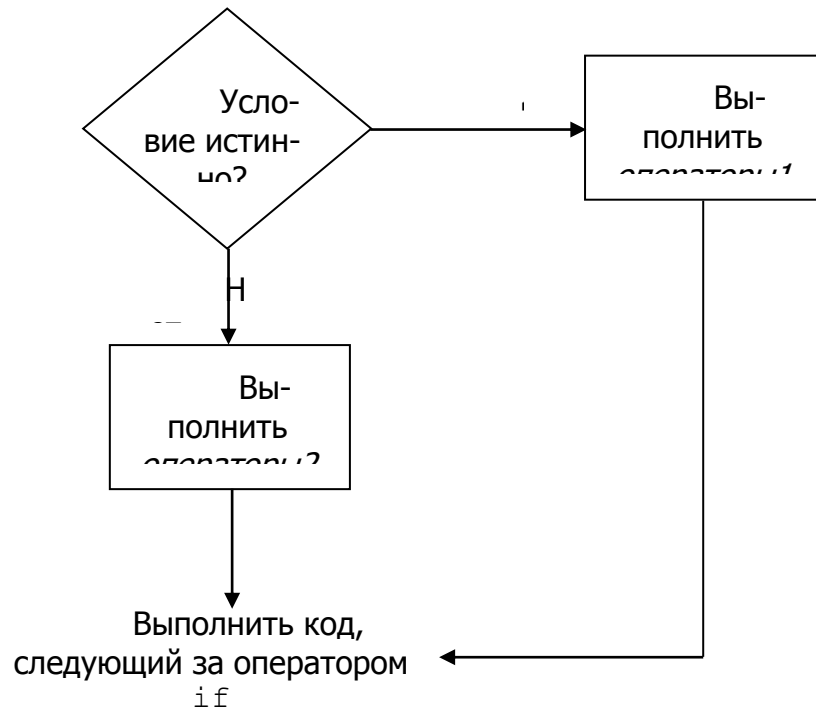


Рис. 2. Блок-схема простой формы оператора if-then-else

Размещение в операторе if перед ключевым словом else точки с запятой (;) – распространенная синтаксическая ошибка. В операторе if между предложением then и ключевым словом if точки с запятой не должно быть никогда. Точка с запятой ставится после всего оператора if, чтобы отделить его от следующего оператора, а между предложениями then и else ставятся только пробелы или символ перехода на следующую строку. Внутри составного оператора, ограниченного ключевыми словами begin и end и входящего в состав оператора if, точки с запятой ставятся, естественно, после каждого оператора.

Операторы if, содержащие предложения else-if, могут быть записаны как **последовательные операторы** вида if-then. В этом случае последовательность условий не менее важна и тоже должна быть записана логически правильно. Применение конструкций проиллюстрировано следующими примерами. Программа, которая по введенному значению аргумента вычисляет значение функции, заданной в виде:

$$y = \begin{cases} 0, & x < -2 \\ -x - 2, & -2 \leq x < -1 \\ x, & -1 \leq x < 1 \\ -x + 2, & 1 \leq x < 2 \\ 0, & x \geq 2 \end{cases}.$$

Конструкция else-if:



```
program calc_function_1;
{$APPTYPE CONSOLE}
uses SysUtils;
var x, y : real;
begin
  writeln(' Введите значение аргумента'); readln(x);
  if x < -2 then y := 0 else
    if x < -1 then y := -x - 2 else
      if x < 1 then y := x else
        if x < 2 then y := -x + 2 else
          y := 0;
  writeln(' Для x= ', x:6:2, ' значение функции y = ', y:6:2);
end.
```

Конструкция if-then:

```
program calc_function_2;
{$APPTYPE CONSOLE}
uses SysUtils;
var x, y : real;
begin
  writeln(' Введите значение аргумента'); readln(x);
  if x < -2 then y := 0;
  if (x >= -2) and (x < -1) then y := -x - 2;
  if (x >= -1) and (x < 1) then y := x;
  if (x >= 1) and (x < 2) then y := -x + 2;
  if x >= 2 then y := 0;
  writeln(' Для x= ', x:6:2, ' значение функции y = ', y:6:2);
end.
```

В программировании часто используются **вложенные операторы if**, т.е. расположенные внутри других операторов if, например:

```
if выражение1 then begin
  if выражение2 then begin
    [операторы1;]
  end
  else begin
    [операторы2;]
  end;
end;
```

При этом else ассоциируется с ближайшим доступным ключевым словом if. Для того чтобы заставить компилятор прочитать наш пример другим способом, нужно "закрыть" для предложения else второй оператор if:

```
if выражение1 then begin
  if выражение2 then begin
```



```
        [операторы1; ]  
    end;  
end  
else begin  
    [операторы2; ]  
end;
```

Использование вложенных операторов `if` определяется главным образом **стилем программирования**. Используйте метод, который кажется вам более простым и удобным.

Операторы `case`

Другая структура принятия решений в Object Pascal – оператор `case`. Каждый оператор `case` можно заменить эквивалентным ему оператором `if`, однако обратное неверно – не всякий оператор `if` можно заменить эквивалентным `case`. Тем не менее оператор `case` используется довольно часто и поддерживается почти во всех языках высокого уровня. Общий синтаксис оператора `case` имеет вид

```
case выражение of  
    список_значений_1: begin  
                        операторы1;  
                        end;  
    список_значений_2: begin  
                        операторы2;  
                        end;  
    .  
    .  
    .  
    список_значений_N: begin  
                        операторыN;  
                        end;  
else begin  
    операторыX;  
end;  
end;
```

В этом синтаксисе *выражение* сравнивается с выражениями каждого списка. Оно должно быть *выражением порядкового типа*, т.е. типа `Integer`, `Char`, `Boolean` или других подобных типов. Кроме того, каждое выражение в списках должно быть порядковым и вычисляемым во время компиляции.

Например, в списках допустимы выражения 12, True, 4 - 9 * 5 или Integer('Z')

Переменные и вызовы большинства функций в списках значений недопустимы. *Список_значений* может также содержать **поддиапазон**, имеющий форму *первое_значение..последнее_значение*, оба из которых должны быть порядковыми, причем *первое_значение* \leq *последнее_значение*. И наконец, *список_значений* может быть представлен в форме *значение1, значение2,..., значениеN* в которой каждое *значение* является порядковым значением или поддиапазоном порядковых значений.

Оператор case может содержать любое количество *списков_значений* и только одно предложение else. Выполнение оператора case аналогично выполнению структуры if-then-else. Если значение управляющего *выражения* совпадает с любым значением из какого-либо списка, то выполняются операторы этого списка. Затем управление передается на оператор, следующий за оператором case (происходит выход из оператора case). Если управляющее *выражение* совпадает с значением, присутствующими в нескольких списках, то выполняются операторы самого верхнего из этих списков. Если управляющее *выражение* не совпадает ни с одним из значений в списках, то выполняются операторы предложения else. Включение предложения else в оператор case не обязательно, однако оно гарантирует, что код обработает любое непредвиденное значение управляющего выражения. Если управляющее *выражение* не совпадает ни с одним из значений в списках и предложение else отсутствует, то не выполняется ни один оператор и происходит выход из оператора case.

Операторы case можно вкладывать друг в друга аналогично вложенным операторам if. Каждый вложенный оператор case должен иметь ассоциированное с ним ключевое слово end.

Рассмотрим использования оператора case на примере приведенном ниже.

```
program prog_case;
{$APPTYPE CONSOLE}
uses SysUtils;
var chChar : Char;
begin
  writeln("Enter a symbol ");
  readln(chChar);
  case chChar of
    'A'..'Z': begin
      writeln(chChar, ' uppercase letter.')
    end;
```

```
'a'..'z': begin
    writeln(chChar, ' lowercase letter.')
end;
'0'..'9': begin
    writeln(chChar, ' number.')
end;
' ': begin
    writeln(chChar, ' space.')
end;
else begin
    writeln(chChar, ' other symbol.')
end;
end;
readln;
end.
```

Контрольные задания

1. Создайте программу, приглашающую пользователя ввести две строки и выводящую строку, символы которой имеют более высокое числовое значение ASCII из двух соответствующих символов, введенных пользователем. В строках используются только символы английского алфавита.
2. Создайте программу, приглашающую пользователя ввести число и определяющую, является ли это число положительным, отрицательным или равным нулю.
3. Используемый нами григорианский календарь был введен в 1582 году. Разработайте программу, определяющую день недели каждого задаваемого дня после 1582 года. Программа должна выполнить следующие действия.
 - а) Пригласить пользователя ввести месяц и год.
 - б) Определить количество дней в месяце и пригласить пользователя ввести номер дня. Проверить, допустим ли введенный пользователем номер дня в месяце. Все годы, делящиеся на 4, являются високосными, кроме делящихся на 100, но не на 400. Например, годы 1600 и 2000 високосные, а 1700, 1800 и 1900 – невисокосные. Если вам удастся, то вы сможете проверить это условие в одном операторе if.
 - в) Определить день недели с помощью следующего алгоритма.
 - Предполагается, что январь и февраль – это тринадцатый и четырнадцатый месяцы предыдущего года. Например, 1/10/1998 (в американской системе обозначений месяц/день/год) заменяется на 13/10/1997, а 2/10/1998 – на 14/10/1997.
 - Пусть m , d и y обозначают месяц, день и год. Необходимо вычис-

лить величину w как

$$w := d + 2 * m + \text{Int}((3/5) * (m + 1)) + y + \text{Int}(y/4) - \text{Int}(y/100) + \text{Int}(y/400) + 2;$$

- Остаток деления w на 7 равен номеру дня в неделе. Предполагается, что 0 – это суббота, 1 — воскресенье, 2 — понедельник и т.д.

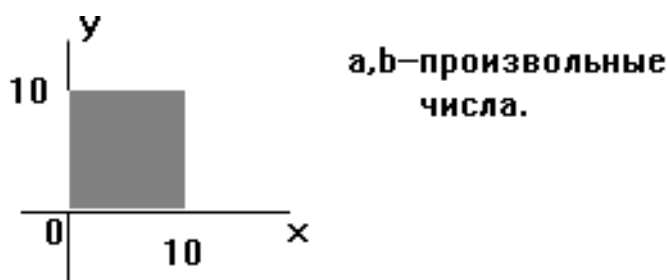
Контрольные вопросы

1. Назовите три оператора сравнения и приведите примеры их использования.
2. Назовите три первичных булевых оператора.
3. Чем отличаются режимы полного и неполного вычисления булевых выражений?
4. Что такое приоритеты операций и почему они так важны?
5. Приведите синтаксис и опишите действия, выполняемые операторами `if` и `case` в самом общем виде.
6. Можно ли заменить любой оператор `if` эквивалентным ему оператором `case` наоборот?
7. Что такое вложенные структуры принятия решений?

Задачи для самостоятельного выполнения

1. Разработайте программу, определяющую количество монет каждого типа в сдаче, величина которой вводится пользователем. Например, если пользователь вводит \$5.88, то программа должна вывести 3 монеты по 25 центов, 1 монеты по 10 центов, 3 монеты по 1 центов (не беспокойтесь о количестве долларов и падеже существительных). Программа должна учитывать только монеты по 1, 5, 10 и 25 центов. Результат должен состоять из минимально возможного количества монет.
2. Определить, имеется ли среди заданных целых чисел A, B, C хотя бы одно чётное.
3. Даны три числа. Вывести на экран те из них, которые принадлежат заданному отрезку $[e, f]$.
4. Определить число, полученное выписыванием в обратном порядке цифр заданного целого трёхзначного числа.
5. Определить, есть ли среди цифр заданного целого трёхзначного числа одинаковые.
6. Выбрать наибольшее из трёх заданных чисел.
7. Определить номер квадранта, в котором находится точка с заданными координатами (x, y) .
8. Написать программу, которая запрашивает у пользователя номер месяца и

- выводит соответствующее название времени года. В случае ввода недопустимого числа должно выдаваться сообщение «Ошибка ввода!».
- Написать программу, которая запрашивает у пользователя номер дня недели и выводит одно из сообщений: «Увы – рабочий день!», «Ура! Суббота!», «Ура! Воскресенье!».
 - Написать программу, которая после ввода с клавиатуры числа (в диапазоне от 1 до 999), обозначает денежную единицу, дописывая слово «рубль» в правильной форме. Например, 12 рублей, 21 рубль и т. д.
 - Составьте программу, которая определяла бы вид треугольника (равносторонний, равнобедренный, разносторонний, прямоугольный, тупоугольный, остроугольный), если по данным трем отрезкам его можно построить.
 - Определить, лежит ли точка $A(a;b)$ внутри квадрата



- Напишите программу, в результате выполнения которой выводится значение true, если $\pi > e$. Возможны следующие варианты:
 - числа π и e описать как константы с точностью 10^{-5}
 - числа π и e представить с машинной точностью.
- Вычислить функцию

$$W = \begin{cases} a_1 x^2 - a_2 x + a_3 & , \text{ если } 10 < x < 17 , \\ \frac{\sqrt{\ln|a_3| + 6.5}}{1.5a_1 + a_2^2} & \text{ в остальных случаях.} \end{cases}$$
- Дано число x . Напечатать в порядке возрастания числа $\sin x$, $1 + |x|$ и $(1 + x^2)^x$.
- Написать программу, вычисляющую стоимость междугородного разговора в соответствии с таблицей:

| Город | Код | Цена руб/мин |
|-------------|-----|--------------|
| Владивосток | 432 | 9,20 |
| Москва | 095 | 4,10 |
| Краснодар | 861 | 2,05 |
| Волгоград | 844 | 2,50 |

Входными данными должны являться код города и число минут.

На выходе мы должны иметь сообщение о стоимости минуты и сумме за разговор.

17. Пусть даны координаты трех вершин прямоугольника. Определить координаты четвертой вершины.

Список использованной литературы

1. *Фаронов В.В.* Delphi 3. Учебный курс. М.: «Нолидж», 1998. 400 с.
2. *Галисеев Г.В.* Программирование в среде Delphi 8 for .NET. М.: Издательский дом «Вильямс», 2004. 304 с.
3. *Павловска Т.А.* Паскаль. Программирование на языке высокого уровня. СПб.: Питер, 2003. 393 с.
4. *Абрамов С.А. и др.* Задачи по программированию. М.: Наука, 1988. 224 с.