ИНФОРМАТИКА

Ремнев А. А., Федотова С. В.

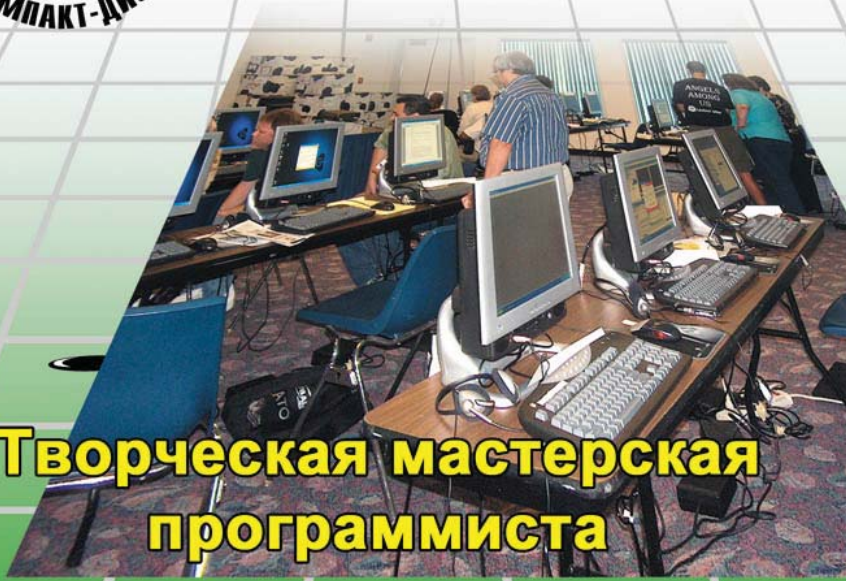
«СОЛОН»

Курс Delphi для начинающих

Полигон нестандартных задач

КОМПАКТ-ДИСК

Преподавателю и ученику



Творческая мастерская программиста

ISBN 5-98003-241-X

ДИСТАНЦИОННОЕ ОБУЧЕНИЕ

УДК 681.3
ББК 32.973-018
Р 37

А. А. Ремнев, С. В. Федотова

Курс Delphi для начинающих. Полигон нестандартных задач. — М.: СОЛОН-ПРЕСС, 2010. — 360 с.: ил. — (Серия «Дистанционное обучение»)

ISBN 5-98003-241-X

Книга написана на основе лекционных курсов раздела «Информатика и ИКТ. Алгоритмизация и визуальное программирование», проводимых в рамках проекта «Обучающие сетевые олимпиады» (ОСО-2006).

Проект ОСО-2005 является номинантом международного конкурса «ИТ-образование в Рунете» (http://ict.edu.ru/konkurs).

Курс программирования предполагает последовательное изучение материала от простого к сложному. Большая часть материалов книги посвящена разбору и решению практических задач. Все примеры программ, а также дополнительная информация методического плана представлены на компакт-диске пособия.

Книга будет полезна широкому кругу читателей — студентам вузов, учащимся лицеев, колледжей, школьникам. Особый интерес она вызовет у преподавателей информатики благодаря строго структурированному системному подходу в изложении материала, а также включению в рассмотрение методологии поисково-деятельностных образовательных технологий проблемного обучения, таких как: исследование, проект или проектно-исследовательская деятельность, решение олимпиадных задач. Поэтому книга может использоваться в качестве учебного пособия для самостоятельного изучения или как подробное методическое руководство.

К книге прилагается компакт-диск с обширной подборкой примеров и обучающих презентаций.

Книга входит в сборник публикаций проекта ОСО 2006.

По вопросам приобретения обращаться: **ООО «АЛЬЯНС-КНИГА КТК»**
Тел: (495) 258-91-94, 258-91-95, **www.abook.ru**
Сайт издательства «СОЛОН-ПРЕСС»: **www.solon-press.ru**
E-mail: **solon-avtor@coba.ru**

SBN 5-98003-241-X

Занятие № 1

Линейный алгоритм и простые вычисления

Если вы не знаете, что ваша программа должна делать, стоит ли ее начинать?
Э. Дейкстра

Переменные и выражения

Написание любой программы на любом языке программирования по своей сути является ничем иным, как автоматизированной обработкой данных. Для программной обработки в ЭВМ данные представляются в виде величин и их совокупностей.

Величина — это элемент данных с точки зрения их семантического (смыслового) содержания или обработки.

Смысловое разбиение данных производится во время постановки задачи и разработки алгоритма ее решения (входные, выходные и промежуточные).

Исходные (входные) — это данные, известные перед выполнением задачи из условия.

Выходные данные — результат решения задачи.

Данные, которые не являются ни аргументом, ни результатом алгоритма, а используются только для обозначения вычисляемого промежуточного значения, называются *промежуточными*.

С понятием величины связаны следующие характеристики (атрибуты):

- *имя* — это ее обозначение и место в памяти;
- *тип* — множество допустимых значений и множество применимых операций к ней;
- *значение* — динамическая характеристика, может меняться многократно в ходе исполнения алгоритма. Во время выполнения алгоритма в каждый конкретный момент величина имеет какое-то значение или не определена.

Постоянной называется величина, значение которой не изменяется в процессе исполнения алгоритма, а остается одним и тем же, указанным в тексте алгоритма.

Переменной называется величина, значение которой меняется в процессе исполнения алгоритма.

Данные представляются в программе в виде переменных (реже постоянных) величин, каждая из которых имеет имя, тип и значение.

Программирование основано на математике, и так же, как и в математике, вычисляемые в программе значения представляются посредством ***выражений***. Тип выражения определяется типами входящих в него величин, а также выполняемыми операциями.

В Бейсике нет необходимости заранее задавать тип величины — тип переменных (кроме символьных) определяет сам интерпретатор языка. В Паскале же, наоборот, указание типов величин является обязательным и первостепенным в прямом смысле слова этапом написания программы.

Различают переменные следующих простых типов:

целые	integer	[−32768; 32767]
вещественные	real	$(-2^{-38}; 2^{38})$
логические	boolean	0 (false), 1 (true)
символьный	char	любой допустимый символ
строковый	string	не более 255 символов

Простые переменные могут организовываться в наборы и структуры данных — массивы, множества, файлы. Работа со структурами данных будет рассмотрена нами позже.

Ввод и вывод данных

Обмен информацией с ЭВМ предполагает использование определенных средств ввода-вывода. В ЭВМ основным средством ввода является клавиатура, вывода — дисплей.

Процедура, которая в режиме диалога с клавиатуры присваивает значение для переменной величины, называется ***процедурой ввода***.

В языке Бейсик для этой цели служит оператор ***INPUT***:

```
INPUT "поясняющее сообщение"; список переменных
```

Например,

```
INPUT "Введите числа A и B"; A,B
```

В языке Паскаль аналогичную функцию выполняет оператор ***read (readln)***:

```
read(список переменных)
```

Например,

```
read(a,b)
```

Оба оператора ввода в Паскале идентичны по своему назначению, но отличие оператора `readln` заключается в том, что после своего завершения он переводит курсор на следующую экранную строку. Оператор `INPUT` выполняет эту операцию, называемую «перевод строки», автоматически.

Как только в программе встречается вызов процедуры ввода, ЭВМ приостанавливает выполнение программы и ждет, пока пользователь введет с клавиатуры соответствующие значения, которые по очереди будут присваиваться переменным, перечисленным в списке ввода.

(!) В Бейсике ввод нескольких значений следует разделять запятыми, в Паскале — пробелами.

Значения вводимых данных одновременно отображаются на экране дисплея. После нажатия клавиши **Enter**, когда все переменные примут свои значения из входного набора данных, определенного пользователем, выполнение программы продолжается с оператора, следующего за оператором ввода.

Процедура, которая выводит содержимое переменных на экран, называется *процедурой вывода* на экран.

В Бейсике используется оператор ***PRINT***:

```
PRINT «Значения A и B"; A,B
```

В Паскале используется оператор ***write (writeln)*** (различие аналогично оператору ввода `read` — есть или нет «перевода строки»):

```
writeln('Значения A и B ',a, ' ',b)
```

В списке вывода операторов может быть либо одно выражение, либо последовательность таких выражений. Обратите внимание, что в Бейсике несколько выражений разделяются между собой запятыми или точкой с запятой, а в Паскале — только запятыми. Причем в Паскале вывод идет слитно и для удобства восприятия приходится искусственно разделять выводимые величины пробелами. В Бейсике точка запятой при выводе «превращается» в один пробел (и кроме этого «запрещает» перевод строки, если стоит в качестве завершающего символа), а запятая «трансформируется» в зону, кратную восьми пробелам. Кроме того, поясняющие сообщения (величины строкового типа) на Бейсике заключаются в двойные кавычки, а в Паскале — в одинарные (апострофы).

Линейный алгоритм и вычисления

Линейный алгоритм является аналогом обычного последовательного решения какой-либо задачи, когда все действия записываются поочередно. В программировании реализация линейного алгоритма является наиболее простой конструкцией, так как подразумевает выполнение всего трех этапов:

1) ввод данных;
2) вычисления с помощью операторов присваивания;
3) вывод данных.

Оператор присваивания — один из самых простых и наиболее часто используемых операторов в любом языке программирования. Он предназначен для вычисления нового значения некоторой переменной, а также для определения значения, возвращаемого функцией. В общем виде оператор присваивания можно записать так:

переменная = выражение.

При записи алгебраических выражений используют арифметические операции (сложение, умножение, вычитание, деление), функции, круглые скобки.

Порядок действий (приоритет) при вычислении значения выражения:

1) вычисляются значения в скобках;
2) вычисляются значения функций;
3) выполняются унарные операции (смена знака и возведение в степень);
4) выполняются операции умножения и деления;
5) выполняются операции сложения и вычитания.

Рассмотрим пример программы линейного алгоритма.

Пример 1. *Длина отрезка задана в дюймах (1 дюйм = 2,54 см). Перевести значение длины в метрическую систему, т. е. выразить ее в сантиметрах.*

```
CLS                                  :' очистка экрана
INPUT "Длина в дюймах";D             :' ввод исходных данных
M=D*2.54                             :' вычисление
PRINT "Длина в сантиметрах";M    :'вывод результата
```

```
uses crt;                              {подключение модуля
                                        работы с экраном}
var d,m: real;                         {объявление переменных}
begin
  clrscr;                              {очистка экрана}
  write('Длина в дюймах:');            {вывод пояснения}
  readln(d);                           {ввод исходных
                                        данных}
  m:=d*2.54;                           {вычисление}
  writeln('Длина в сантиметрах ',m);      {вывод
                                           результата}
  readln;
end.
```

Различия в написании программ видны сразу. Программа на Паскале заметно больше и тем самым дает ощущение «мощи» языка. Однако при многих «плюсах» Паскаль заметно уступает Бейсику в вопросе оформления программ. Так, для соблюдения главного правила хорошего тона по отношению к пользователю программы — очистка экрана — в Бейсике нам было достаточно указать один оператор, а в Паскале пришлось еще и подключать дополнительный модуль, так как сам по себе Паскаль не имеет функций работы с экраном. Обязателен в Паскале ***и раздел описания переменных*** **var-**, в котором требуется указать имена и типы величин, используемых в программе. Неоспоримо «лучше» и оператор INPUT в плане вывода поясняющих сообщений — в Паскале подобная «фишка» реализуется связкой `write – readln`. Далее, для того чтобы увидеть результат на экране без возврата в среду программирования, в Паскале нам пришлось ставить дополнительный оператор ввода. В принципе это не является обязательным, но программирование — это не только умение логически мыслить и составлять программы, но и умение красиво и понятно оформить исходный текст самой программы и продумать интерфейс ее работы.

Можно сказать, что исторически сложился и стиль написания программ: для операторов и переменных на Бейсике используются прописные символы, в Паскале же, наоборот, строчные.

Обратите также внимание, что знак равенства, используемый в выражениях на языке Бейсик, в Паскале заменен на знак присваивания **:=**. И самое главное замечание: **каждая строка в программе на языке Паскаль должна обязательно завершаться точкой с запятой**.

Стандартные функции

При решении любых задач, а особенно задач, связанных с вычислением какого-либо математического выражения, возникает необходимость вычислений стандартных функций. Ниже предлагается систематизированная таблица наиболее часто используемых функций языков программирования. Различия между Бейсиком и Паскалем читателю предлагается найти самостоятельно.

Функция	Смысл	Бейсик	Паскаль	Значения
abs	$\|x\|$	ABS(X)	ABS(−1)=1 ABS(2)=2	
sqr	$\sqrt{x}$	SQR(X)	—	SQR(25)=5
	X^2	—	SQR(X)	SQR(4)=16
sqrt	$\sqrt{x}$	—	SQRT(X)	SQRT(36)=6
sin **cos**	sin x cos x	SIN(X) COS(X)		синус числа косинус числа
tan	tg x	TAN(X)	—	тангенс числа
mod	остаток от деления	a MOD b		5 mod 2 = 1 6 mod 3 = 0
int	округление	INT(a)	—	INT(4.4)=4 INT(6.8)=7
round		—	round(b)	round(4.4)=4 round (6.8)=7
fix	усечение до целого	FIX(E)	—	FIX(4.4)=4 FIX(6.8)=6
trunc		—	trunc(g)	trunc(4.4)=4 trunc(6.8)=6
\	деление нацело	A \ B	—	6 \ 4 = 1
div		—	a div b	12 div 5 = 2
exp	e^x	EXP(X)		экспонента числа
ln	ln x	LOG(X)	ln(x)	натуральный логарифм
степень	a^b	A^B	exp(b*ln(a))	степень числа

Этого «джентльменского» набора достаточно для решения большинства задач. Если же в задаче встречается «редкая» функция, то обычно ее выражают через стандартные. Например, котангенс числа записывают как cos(x)/sin(x).

При записи сложны выражений следует внимательно относиться к скобкам и запомнить простое правило: **число открывающих скобок всегда должно быть равно количеству закрывающих**.

Пример 2. *Треугольник задан величинами своих сторон. Найти его площадь.*

```
CLS
INPUT "Введите значения сторон";A,B,C
P=(A+B+C)/2
S=SQR(P*(P-A)*(P-B)*(P-C))                    :'формула Герона
PRINT "Площадь треугольника";S
```

```
var    a,b,c: integer;
    p,s: real;
begin
       write('A=');
       readln(a);
       write('B=');
       readln(b);
       write('C=');
       readln(c);
       p:=(a+b+c)/2;
       s:=sqrt(p*(p-a)*(p-b)*(p-c));
       writeln('S=',s:6:2);
end.
```

Управление выводом на экран

Написание программы предполагает создание диалога с пользователем и вывода результата на экран. Правилом хорошего тона в программировании, как уже было замечено выше, считается использование подсказок при вводе значений («Введите число»), расшифровка результата («Сумма чисел равна») и поясняющих сообщений при работе программы («Корней нет и не будет!»). Кроме того, в последнем примере на Паскале при выводе результата мы использовали так называемый ***формат (шаблон) вывода***. Первое число резервирует общее число знаков в выводимом числе, второе — количество знаков после десятичной точки. Это необхо-

димо для того, чтобы вместо малопонятной записи 1.34576848E+01 (что, кстати, означает $1.34576848*10^1$) на экране отобразилось вполне понятное нам число 13.46. В Бейсике для подобного «фокуса» используется расширенный оператор вывода ***PRINT USING***, в котором символы «решетки» задают требуемый формат вывода величины:

```
PRINT USING «####.###»; список переменных
```

В любом случае, последовательная структура программы подразумевает и последовательный вывод на экран. Тем не менее в Бейсике и Паскале есть операторы, позволяющие организовать вывод в любом месте экрана.

Экран в текстовом режиме содержит 25 строк по 80 символов. Соответственно, каждое знакоместо имеет свои координаты. Так, знакоместо 6-40 определяет символ, находящийся в 40-м столбце 6-й строки экрана. Буквой X обозначим номер строки, буквой Y — номер столбца.

В Бейсике выводом на экран управляет оператор ***LOCATE X,Y***, а в Паскале — функция ***gotoxy(x,y)***.

Следующим за ними оператор ввода или вывода начнет вывод сообщений, начиная с знакоместа X-У. Таким образом, если мы хотим вывести строку «HELLO, WORLD» в центре экрана, то мы можем сделать это так:

```
LOCATE 12, 34
PRINT "HELLO, WORLD"
```

Аналогично на Паскале:

```
gotoxy(12,34);
writeln('Hello world');
```

Задания для самостоятельного решения

1. Запишите указанные ниже выражения по правилам языка программирования.

$$A)\ a = \frac{\sqrt{|x-1|} - \sqrt{y}}{1 + x^2 + y^2} \qquad B)\ b = 1 + |y - x| + \frac{(y - x^2)}{2} + \frac{y - x}{3}$$

$$C)\ a = \frac{1 + \cos(y - 2)}{x^4 + \sin z} \qquad D)\ b = y + \frac{x}{y^2 + \left|\frac{x^2}{y + x^3}\right|}$$

$$E)\ a = \frac{1 + \sin^2(x + y)}{2 + |x - 2| + \sqrt{x^2 y^2 + 1}} \qquad F)\ b = x - \frac{x}{\sqrt{x}} + \frac{y}{\sqrt{y}}$$

2. Проанализируйте следующие задачи, определите входные и выходные данные.

3. Составьте список переменных, укажите тип каждой из них.

4. Составьте программы на языке Бейсик и Паскаль, проверьте правильность решения с помощью контрольных входных и выходных данных.

A. За год размер квартплаты повышался дважды. Первый раз на A%, а второй на B%. Год назад сумма оплаты составляла K рублей в месяц. Вычислить текущий размер квартплаты.

B. Известен объем продукции, выпускаемый N предприятиями отрасли. Вычислить средний объем продукции, выпускаемый одним предприятием.

C. Дана длина ребра куба. Найти объем куба и площадь его боковой поверхности.

D. На производство 1 кв. метра ткани затрачивают 15 минут. Определить количество ткани, выпущенное за T часов при ширине полотна L.

Практикум первый. Задачи… без алгоритмов

Строго говоря, каждая задача имеет алгоритм решения. Однако условимся на время, что задача имеет алгоритм решения только в том случае, если используются специальные алгоритмические конструкции. Иными словами, задачу линейного алгоритма будем называть задачей без алгоритма.

Действительно, многие сложные на первый вид задачи решаются без каких-либо алгоритмов, а лишь простыми вычислениями, основанными на простых логических рассуждениях или готовых математических формулах. Хорошим примером является следующая задача:

Владелец автомобиля приобрел новый карбюратор, который экономит 50% топлива, новую систему зажигания, которая экономит 30% топлива, и новые поршневые кольца, которые экономят 20%

топлива. Верно ли, что его автомобиль теперь сможет обходиться совсем без топлива? Найти фактическую экономию топлива.

Ответ на первый вопрос возникает естественно: нет, в любом случае какое-то количество бензина машина будет расходовать. Но какое? Простое суммирование или нахождение среднего значения исходных данных даст нам неверный результат. Обратив внимание на то, что исходные значения заданы в процентах, находим в справочнике формулу сложных процентов

$$p_{\Sigma} = \left[\left(1 + \frac{p_1}{100} \right) \left(1 + \frac{p_2}{100} \right) \left(1 + \frac{p_3}{100} \right) + 1 \right] * 100\%,$$

вычисление по которой и даст нам верный результат.

Практически тот же подход используется и при решении большинства задач «на время», на перевод величин в разных системах мер, на экономию операций при выполнении некоторых действий. Здесь хорошими помощниками являются функции остатка от деления и целого деления, а также математические ухищрения.

Решить задачу обмена значениями двух переменных, не используя дополнительных переменных и предполагая, что значениями целых переменных могут быть произвольные целые числа.

Обозначим начальные значения переменных a и b через a0 и b0. Тогда математически очевидно следующее:

```
a=a+b          {a=a0+b0, b=b0}
b=a-b          {a=a0+b0,b=a0)
a=a-b          {a=b0, b=a0}
```

Реализовав эти вычисления на языке программирования, получим решение нашей задачи:

```
CLS
INPUT "ЧИСЛО A"; A
INPUT "ЧИСЛО B"; B
A = A + B
B = A - B
A = A - B
PRINT "A="; A, "B="; B
```

```
var a,b:integer;
begin
      write('Input A: ');
      readln(a);
      write('Input B: ');
      readln(b);
           a:=a+b;
           b:=a-b;
           a:=a-b;
      writeln('A = ',a,' B = ',b);
        readln;
end.
```

Часовая стрелка образует угол φ *с лучом, проходящим через центр циферблата и через точку, соответствующую 12 часам. По значению угла определить время, показываемое часами.*

Циферблат разбит на 12 равных частей (часов). Циферблат, как окружность, имеет 360 градусов. Значит, один полный час равен 30 градусам. То есть количество полных часов определяется выражением φ div 30. Далее. За один полный час, т. е. за 30 градусов хода часовой стрелки, минутная совершает полный оборот, т. е. 360 градусов. Значит, 1 минута равна 2 градусам и количество минут в текущем часе будет равно:

$$(\varphi \bmod 30)*2.$$

Точность определения времени по углу часовой стрелки равна ±1 минута.

```
CLS
INPUT "ВВЕДИТЕ УГОЛ";FI
CHAS=FI\30
MIN=(FI MOD 30)*2
PRINT "ТЕКУЩЕЕ ВРЕМЯ";CHAS;"ЧАС";MIN;"МИН"
```

```
var fi,chas,min: integer;
begin
     write ('Введите угол ');
     readln(fi);
     chas:=fi div 30;
     min:=(fi mod 30)*2;
     write('Текущее время ',chas,' час',min,' мин');
end.
```

Попробуйте самостоятельно решить следующие задачи

1-1. Временной интервал

Заданы моменты начала и конца некоторого промежутка времени в часах, минутах и секундах (в пределах одних суток). Найти продолжительность этого промежутка в тех же единицах измерения.

1-2. Округленное время

Текущее время (часы, минуты, секунды) задано тремя переменными: h, m, s. Округлить его до целых значений минут и часов. Например, 14 ч 21 мин 45 с преобразуется в 14 ч 22 мин или 14 ч, а 9 ч 59 мин 23 с — соответственно в 9 ч 59 мин или 10 ч.

1-3. Угловое время

Задан угол между часовой стрелкой и линией, соединяющей центр циферблата и точку в 12 часов. Найти время, показываемое часами, и определить, сколько времени пройдет до того момента, когда:

а) стрелки часов совпадут;

б) стрелки часов станут перпендикулярны;

в) стрелки часов образуют ровную линию (угол в 180 градусов).

1-4. Русские единицы длины

1 верста = 500 саженей; 1 сажень = 3 аршина; 1 аршин = 16 вершков; 1 вершок = 44,45 мм. Длина некоторого отрезка составляет p метров. Перевести ее в русскую неметрическую систему.

1-5. Хитрая степень

Возвести заданное число x в степень n, не используя многократного умножения и возведения в степень выше квадрата.

1-6. Число наоборот

Данное четырехзначное число записать наоборот. Например: 6584 — 4856.

1-7. Сравните числа

Найти наибольшее и наименьшее из двух заданных чисел A и B.

1-8. Деньги, деньги...

Некоторая сумма денег выражена в копейках, например 317, 5050, 100. Записать данную сумму в рублях и копейках, т. е. в виде: 3 руб. 17 коп., 50 руб. 50 коп, 1 руб. 00 коп. (Количество копеек всегда выражается двумя цифрами.)

Занятие № 2

Условный алгоритм и его модификации

У каждой программы по крайней мере два назначения:
что она должна делать и чего не должна.
А. Перлис

Условный оператор

На практике решение большинства задач не удается описать с помощью программ линейной структуры. Такая ситуация возникает тогда, когда решение задачи зависит от какого-либо условия. При этом после проверки некоторого условия выполняется та или иная последовательность операторов, т. е. происходит нарушение естественного порядка выполнения операторов. Для этих целей используют управляющие операторы.

Условный алгоритм хранит в себе такое количество возможностей, что практически все задачи содержат в себе данную алгоритмическую конструкцию. В то же время следует отметить «своенравность» условного алгоритма. Чтобы «приручить» его, необходимо научиться правильно составлять условия. В принципе все условия основаны на какой-либо закономерности, и именно этому будет посвящено наше сегодняшнее занятие.

Условный оператор используется для реализации разветвлений в программе, которые происходят при выполнении некоторого условия, и имеет следующую структуру, одинаковую для Бейсика и Паскаля:

```
IF <логическое выражение> THEN серия1 ELSE серия2;
```

Данная запись читается как: ЕСЛИ <логическое выражение> ТО ... ИНАЧЕ ...

Серия — один или несколько операторов языка. Если операторов несколько, то в Бейсике их следует разделять знаком двоеточия, а в Паскале — заключать в ***операторные скобки*** **begin...end**, не забывая при этом в конце каждой строки (оператора) ставить точку с запятой.

Если логическое выражение, выступающее в качестве условия, принимает значение `False` (ложь), то выполняются операторы, расположенные после `else` (серия2), если `True` (истина) — операторы, следующие за `then`. При записи логического выражения следует избегать знака = (равно) для действительных (веществен-

ных) переменных, так как они представляются с некоторой точностью (до определенного знака), а поэтому может не произойти совпадения значений выражений, стоящих слева и справа от знака равно. Для устранения указанного недостатка следует проверять выполнения условия с заданной точностью, т. е. вместо отношения $X = Y$ рекомендуется, например, использовать такой прием:

```
ABS(X – Y) < 1E-8,
```

т. е. абсолютная разница сравниваемых величин не превышает некоторой весьма малой величины.

Поскольку развилка может быть неполной, т. е. требуется рассмотрение только одного варианта, то возможна и неполная форма записи условного оператора:

```
IF <логическое выражение> THEN серия;
```

Условный оператор реализует разветвление вычислительного процесса по двум направлениям, одно из которых осуществляется при выполнении условия, другое — в противном случае. Для реализации разветвлений более чем по двум направлениям необходимо использовать несколько условных операторов. Рассмотрим примеры.

Пример 1. *Дано число a. Вычислить f(a), если*

$$f(x) = \begin{cases} 0 \text{ при } x \le 0, \\ x^2 - x \text{ при } 0 < x \le 1, \\ x^2 - \sin \pi x^2 \text{ при других } x \end{cases}$$

```
INPUT "Введите число:"; A
IF A<=0 THEN F=0 ELSE IF A<=1 THEN F=A^2-A
 ELSE F=SQR(A)-SIN(3.14*A^2)
PRINT "Значение функции F(x) при x = ";A;"равно";F
```

```
var a,f:real;
begin
        writeln('Введите число: ');
        readln(a);
        if a <= 0 then
                f:= 0
        else
                if a <= 1 then f: = sqr(a) – a
                else f: = sqr(a) – sin(pi * sqr(a));
writeln('значение функции f(x) при x = ', a:10:4,
' равно ', f:10:4);
end.
```

Логические функции

Рассмотрим следующую задачу.

Пример 2. *Даны действительные числа x, y. Если x и y отрицательны, то каждое значение заменить модулем; если отрицательно только одно из них, то оба значения увеличить на 0,5; если оба значения неотрицательны и ни одно из них не принадлежит отрезку [0,5; 2,0], то оба значения уменьшить в 10 раз; в остальных случаях x и y оставить без изменения.*

Разработаем алгоритм решения задачи, после чего напишем программу.

Алгоритм запишем словесно:

```
1) ввести значения x, y;
2) если x<0 и y<0, найти их модули и перейти к п. 5, иначе
перейти к следующему пункту;
3) если x<0 или y<0, увеличить каждую величину на 0,5 и
перейти к п.5, иначе перейти к следующему пункту;
4) если ни x, ни y не принадлежат отрезку [0,5; 2,0], уме-
ньшить их в 10 раз;
5) вывести значения x и y;
6) конец.
```

Обратите внимание на «связки» в нашем условном алгоритме:

```
если x<0 и y<0,
если x<0 или y<0
если ни x, ни y
```

Эти смысловые союзы образуют смысловую логическую связку двух или более условий в одно. В программировании они называются ***логическими функциями***. Основных логических функций четыре:

AND (ЛОГИЧЕСКОЕ **И**),
OR (ЛОГИЧЕСКОЕ **ИЛИ**)
NOT (ЛОГИЧЕСКОЕ **НЕ**)
XOR (ИСКЛЮЧАЮЩЕЕ **ИЛИ**)

Последняя функция весьма специфична, поэтому ее рассмотрение пока отложим.

Для того чтобы понять, как работают логические функции, используем так называемые ***таблицы истинности***, которые позволяют увидеть, какое значение примет та или иная логическая функция при различных входных условиях.

Логические функции могут принимать только одно значение из двух — ИСТИНА или ЛОЖЬ (TRUE — FALSE). Часто истинность обозначают цифрой 1, а ложность — цифрой 0.

X	Y	X AND Y	X OR Y	NOT X
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Обратите внимание на значения функций. Функция отрицания НЕ дает всегда противоположное исходному значение. Функция И истинна только в том случае, когда оба связанных условия истинны. Функция же ИЛИ истинна, если хотя бы одно из связываемых условий истинно.

Если в таблице истинности между аргументами поставить знак сложения для функции ИЛИ, а для функции И поставить знак умножения и сравнить арифметический результат со значением функций, то станет понятно, почему функцию ИЛИ называют также «логическое сложение», а функцию И — «логическое умножение».

Перенесем наш словесный алгоритм в среду программирования. Отметим следующий факт: при использовании логических функций требуется заключение связываемых условий в обычные скобки.

```
   INPUT "Введите два числа";X,Y
   IF X<0 AND Y<0 THEN X=ABS(X): Y=ABS(Y)
   IF X<0 OR Y<0 THEN X=X+0.5: Y = Y + 0.5
   IF NOT (((X>=0.5) AND (X <= 2))OR((Y>=0.5) AND (Y<=2)))
THEN X=X/10: Y=Y/10
   PRINT "Результат:"
   PRINT "X="; X, "Y=";Y
```

Примечание: оператор IF...THEN... записывается в одну строку без переносов в тексте программы.

```
  var x, y : real;
  begin
     write('введите два числа '); readln(x, y);
     if (x<0) and (y<0) then
              begin x:=abs(x); y:=abs(y) end
     else
              if (x<0) or (y<0) then begin
                   x:=x+0.5; y:=y+0.5 end
              else
                   if not (((x>=0.5) and (x<=2))
                             or ((y>=0.5) and (y<=2))) then
                             begin x:=x/10; y:=y/10 end;
    writeln('результат:'); writeln('x= ', x:6:3);
writeln('y= ', y:6:3)
  end.
```

Попробуйте самостоятельно разобрать следующие задачи.

Пример 3. *Написать программу решения обычного квадратного уравнения.*

```
  INPUT "Введите A,B,C";a,b,c
  d=b^2-4*a*c
  IF d<0 THEN PRINT "действительных корней нет" ELSE
  IF d=0 THEN x=(-b)/(2*a): PRINT "корень уравнения:";x
  ELSE x=(-b+sqr(d))/(2*a): PRINT "1-й корень уравнения:";x:
  x=(-b-sqr(d))/(2*a): PRINT "2-й корень уравнения:";x
```

Примечание: 3, 4, 5, 6 строки текста программы набираются в одну строку без переносов.

```
  var a,b,c,d,x:real;
  begin
      writeln('введите A,B,C');
      readln(a,b,c);
      d:=sqr(b)-4*a*c;
      if d<0 then begin
         writeln('Действительных корней нет');
      end else if d=0 then begin
         x:=(-b)/(2*a);
         writeln('корень уравнения: ',x);
      end else begin
         x:=(-b+sqrt(d))/(2*a);
         writeln('1-й корень уравнения: ',x);
         x:=(-b-sqrt(d))/(2*a);
         writeln('2-й корень уравнения: ',x);
      end
  end.
```

Пример 4. *Какая из двух точек на плоскости, заданная своими координатами, ближе к началу координат?*

```
INPUT "Введите A(X1,Y1) и B(X2,Y2)"; x1,y1,x2,y2
d1:=sqr(y1^2+x1^2)
d2:=sqr(y2^2+x2^2)
IF d1<d2 THEN PRINT "Точка A ближе" ELSE IF d1>d2 THEN
PRINT "Точка B ближе" ELSE PRINT "Одинаково"
```

Примечание: 4, 5 строки текста программы набираются в одну строку без переносов.

```
var x1,y1,x2,y2,d1,d2:real;
begin
     writeln('введите A(X1,Y1) и B(X2,Y2)');
     readln( x1,y1,x2,y2 );
     d1:=sqrt(sqr(y1)+sqr(x1));
     d2:=sqrt(sqr(y2)+sqr(x2));
     if d1<d2 then writeln('Точка A ближе')
     else if d1>d2 then writeln('Точка B ближе')
     else writeln('Одинаково');
end.
```

Структурный условный оператор

Язык Паскаль отличается своей структурированностью, что позволяет писать на нем программы, весьма строгие по своей структуре и более легкие для понимания. Однако версия языка Бейсик, известная как Microsoft Qbasic, также имеет возможности структурного программирования. В частности, условный оператор IF можно записать в таком виде:

```
IF условие THEN
   [серия1]
ELSE
   [серия2]
END IF
```

Например:

```
INPUT "1 или 2", i
IF i=1 OR i=2 THEN
    PRINT "OK"
ELSE
    PRINT "Неверно"
END IF
```

Содержание