Московский Государственный университет
имени М.В.Ломоносова
Химический факультет

Калугина О.Б., Люцарев В.С.

СПРАВОЧНИК

ОСНОВЫ ПРОГРАММИРОВАНИЯ И ИСПОЛЬЗОВАНИЯ ЧИСЛЕННЫХ МЕТОДОВ

Методическая разработка

Москва - 1996

Методическая разработка содержит справочный материал к заданиям по основным темам, изучаемым в рамках дисциплины “Программирование и решение задач на ЭВМ” студентами 1-го курса общего потока Химического факультета МГУ.

В первом разделе методической разработки описаны некоторые приемы программирования на алгоритмическом языке Pascal. Второй ее раздел – краткий обзор основных численных методов.

С о д е р ж а н и е

Алгоритмический язык Pascal. Краткий справочник……..…4

Тема 1:
Константы и переменные. Арифметические выражения.
Использование стандартных математических функций.
Операторы присвоения, ввода и вывода значений……5

Тема 2:
Организация циклов в программах. Оператор For … do …
Составной оператор……....19

Тема 3:
Операторы повторений Repeat ... until ... и While ... do …
Правила организации вложенных циклов……23

Тема 4:
Организация разветвлений в программах;
оператор If ... then ... else ... ……28

Тема 5:
Организация и использование функций (Function)
и процедур (Procedure) в программах……...……31

Тема 6:
Массивы переменных. Взаимодействие программы с внешними
файлами данных…….……..37

Численные методы. Краткий обзор и основные
математические формулы……44

Методы численного интегрирования…….45

Численное решение нелинейных уравнений ……....48

Линейная регрессия…….53

Решение дифференциальных уравнений…….…..56

Приложение 1: Зарезервированные слова в Pascal’е…….…….60

Приложение 2: Список некоторых стандартных
математических функций ……..62

Приложение 3: Диагностика некоторых ошибок……..……..63

Литература……..69

Алгоритмический язык Pascal.
Краткий справочник.

В данном разделе описаны некоторые приемы программирования на алгоритмическом языке Pascal. Здесь умышленно не дается полное описание языка, которое можно найти в литературе [1-5], а большее внимание уделено его практическому использованию, в частности, для решения задач из раздела I.

При изучении программирования на каком-либо алгоритмическом языке (в том числе, Pascal'e) важно помнить о том, что, как и в любом языке вообще, здесь есть строгие правила и ряд ограничений, которых обязательно придерживаются при составлении программ любой степени сложности. Вместе с тем, одна и та же вычислительная задача каждым программистом может быть реализована "по-своему" с учетом его знаний и навыков.

Программы, которые приводятся в качестве примеров, умышленно очень просты. Их основная цель - сконцентрировать внимание начинающего пользователя на специфике работы того или иного оператора Pascal’я, а не искусстве построения сложных алгоритмов.

Тема 1

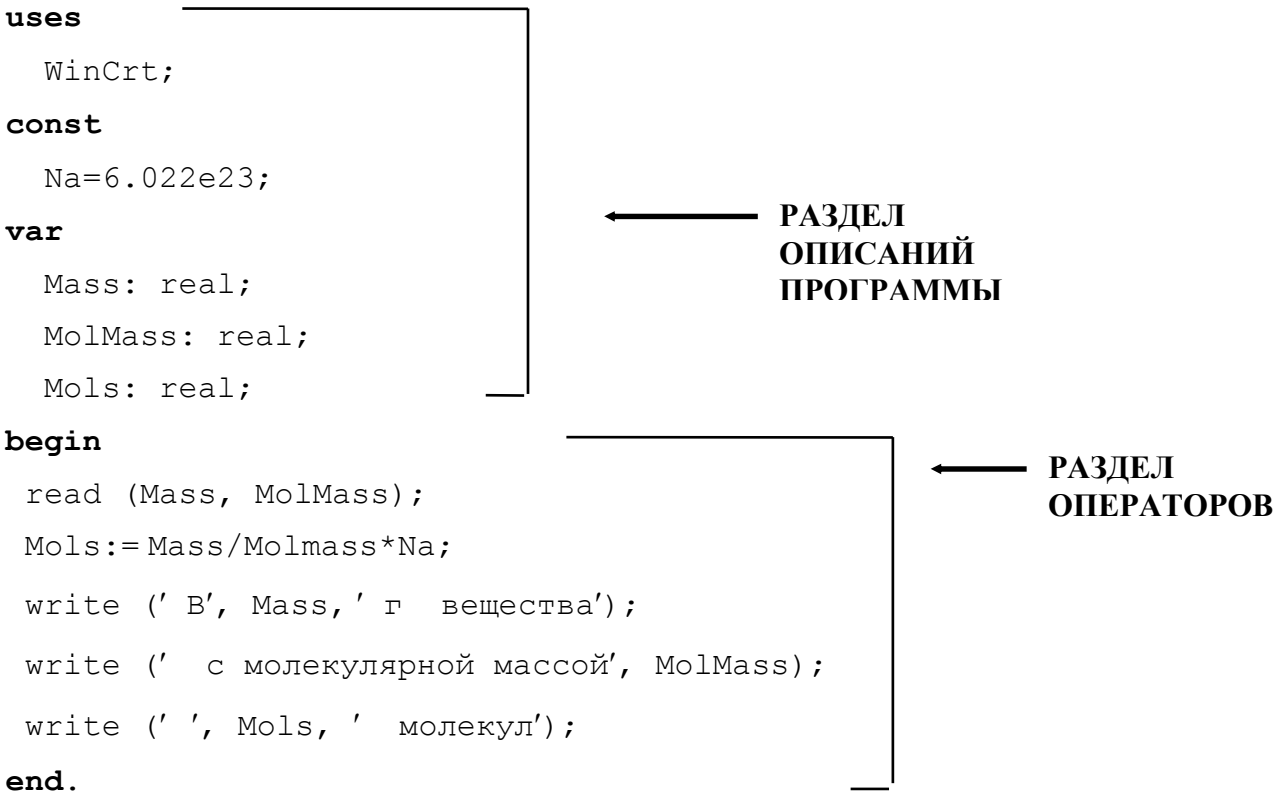
Константы и переменные. Арифметические выражения. Использование стандартных математических функций. Операторы присвоения, ввода и вывода значений

Структура программы

Любая программа на Pascal'е состоит из *раздела описаний* и *раздела операторов*.

В первом из них описывают все величины (константы, переменные и т.д.) и подпрограммы (процедуры, функции), используемые в программе. В разделе операторов, который всегда начинается со слова **`begin`**, последовательно указываются те действия, которые должна выполнить программа. Заканчивается программа на Pascal'е словом **`end.`** (с точкой) - это также обязательное требование.

Приведем в качестве примера текст программы для вычисления количества молей в навеске вещества.



Раздел описаний обычно состоит из нескольких *секций*. Каждая секция начинается со слова, которое ее характеризует (**uses, const, var,** …).

uses

Этим служебным словом начинается *секция используемых программных модулей*. В ней указывается одно или несколько перечисленных через запятую названий модулей. Например, строка

```
uses WinCrt;
```

указывает на необходимость использования программного модуля `WinCrt`, отвечающего за диалог с пользователем при работе в среде Windows.

const

Так всегда начинается *секция описания констант*. В ней указываются имена и значения постоянных величин, которые будут использоваться в программе. В общем случае каждое описание константы имеет вид:

<*имя*> = <*константное выражение*>;

ПРИМЕРЫ

```
. . .
const
  Na = 6.022e23;
  R = 8.2056e-5;
. . .
```

Обозначения (имена) различным объектам в программе (константам, переменным, процедурам, функциям) присваивает программист.

Правила записи имен

- Именем может быть любая последовательность из латинских букв, цифр, знаков подчеркивания. Другие символы, например пробелы, в именах использовать нельзя!
- Имя не может начинаться с цифры.
- Внутри имен недопустимы пробелы.
- Прописные и строчные буквы не различаются.
- Зарезервированные слова [1] нельзя использовать в качестве имен каких-либо объектов программы.

[1] назначение некоторых специальных слов строго определено в PASCAL'е и не меняется от программы к программе. Это так называемые зарезервированные слова: const, var, begin, end … Список зарезервированных слов приведен в приложении 1.

ПРАВИЛЬНО ☺	НЕПРАВИЛЬНО ☹
a	1a
A1	A(1)
A__1	A 1
m34F12a	3412
molecularmass	молекул_масса

Константное выражение - это значение, которое известно заранее и не меняется в процессе выполнения программы. Простейшим примером константного выражения являются *числовые константы*.

Правила записи чисел

Для записи чисел в Pascal'е используются три формы.

Целое число записывается в виде последовательности цифр и знака минус (для отрицательных чисел):

```
0       345      -15
```

Для записи дробных чисел используется десятичная точка:

```
3.1415926      -2.718
```

Очень большие и очень маленькие числа обычно записываются в экспоненциальной форме:

`6.022e23`	$(= 6.022 \cdot 10^{23})$	Латинские буквы "`E`" и "`e`"
`-5E-7`	(= -0.0000005)	не различаются!

Показатель степени, который ставится после латинской буквы "E", может быть только целым. Мантисса должна обязательно присутствовать. Например, число 10^9 записывается как `1e9`, а запись `e9` воспринимается как имя, а не как число.

var

Этим служебным словом начинается *секция описания переменных*. Она служит для указания имен и типа переменных величин, которые будут использоваться в программе.

Переменная - это величина, значение которой может меняться в процессе выполнения программы. Любая переменная должна быть обозначена именем.

Чаще всего, значениями переменных являются числа. Но это не единственная возможность: значением может быть буква, строка символов, допустимы и другие варианты.

Для каждой переменной множество возможных ее значений строго фиксировано и называется *типом* переменной. Тип переменной также обязательно указывается при ее описании в секции **var**.

Условимся использовать для решения учебных задач следующие типы числовых величин:

Тип величин	Диапазон представления
REAL (вещественные)	0, а также действительные числа по абсолютному значению от 2.9e-39 до 1.7e38 с указанием 11 значащих цифр
INTEGER (целые)	Целые числа от -32768 до 32767

В общем случае описание переменных имеет вид:

<*имена переменных*>**:** <*тип*>;

<*имена переменных*> – это одно или несколько перечисленных через запятую имен.

ПРИМЕРЫ

```
var                              var
     Mass, Molmass: real;             I: integer;
     Mols: real;                      a, Sum: real;
```

begin

Как уже отмечалось выше, этим служебным словом начинается *раздел операторов* – последовательность инструкций о том, какие действия должен выполнить компьютер. Операторы должны быть отделены друг от друга точкой с запятой. Заканчивается программа на Pascal'е всегда служебным словом **end** и точкой.

Кроме этих обязательных служебных слов (первого **begin** и последнего **end**.) внутри программы могут быть также использованы пары этих слов для объединения в группы сразу нескольких операторов[1]. Поэтому иногда служебные слова **begin** и **end** называют "операторными скобками".

```
begin                  ┐
  begin                │
    begin              │
      . . .            │   РАЗДЕЛ ОПЕРАТОРОВ
    end;               │
  end;                 │
end.                   ┘
```

[1] Подробнее см. "Составной оператор" (стр.8)

Выполнение любой программы - это последовательное исполнение ее операторов.

Вернемся к примеру программы (стр.6) о количестве молей в навеске вещества. Первым ее оператором стоит оператор `read`. Его назначение - считывание информации из входного потока. По умолчанию источником данных для входного потока является клавиатура компьютера. Поэтому сразу после запуска исполнение программы приостанавливается, пока во входной поток информации не будут отправлены пользователем два числа.

Считывание информации оператором `read`

Оператор `read` имеет вид:

`read (`<*переменная*>`,`<*переменная*>`, ... ,`<*переменная*>`)`

Оператор `read` принимает информацию из входного потока, интерпретирует ее как значения и присваивает эти значения соответствующим переменным. Количество значений и порядок их следования во входном потоке должны находиться в строгом соответствии со списком переменных в операторе `read`. Если эти значения являются числами, то каждое из них набирается по правилам записи чисел (стр.8). Между значениями во входном потоке должен стоять один или несколько разделительных символов, которыми могут быть только пробел или символ новой строки. Если при считывании данных оператор `Read` не обнаруживает в строке числа, он считывает следующую строку и т.д. Если же число набрано неправильно или в его записи встречаются недопустимые символы (например, запятая или точка с запятой), программа прекращает работу и выдает сообщение об

ошибке (invalid numeric format)[1] . Отметим, что при вводе с клавиатуры информация попадает во входной поток программы только после того, как нажата клавиша 

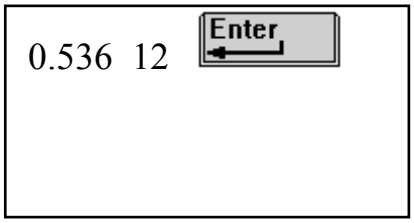
. Если при наборе допущена ошибка и информация еще не отправлена во входной поток, можно воспользоваться клавишей 

для внесения исправлений. После нажатия 

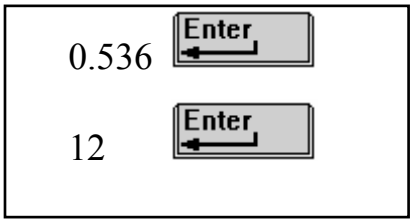
изменения значений произвести уже нельзя.

ПРИМЕРЫ

Числовые значения для списка переменных в операторе `Read(a,i)`:



или



В любом из этих случаев числа попадут во входной поток программы, оператор `read` присвоит переменным соответствующие значения, с которыми и будут произведены последующие вычисления.

Печать чисел оператором `write`

Оператор `write` может быть представлен как

`write` (<*элемент*>,<*элемент*>, … , <*элемент*>)

[1] Список некоторых ошибок, возможных при запуске программы, приведен в приложении 3.

Его исполнение заключается в последовательном выводе указанных элементов в выходной поток. В качестве элементов вывода в простейшем случае можно указывать следующие конструкции:

- *<строка>*

  - последовательность символов, заключенная в апострофы. Все символы между апострофами, включая пробелы, скобки и т.д., переносятся в выводимую строку. Например:

```
write('Результаты расчета по программе:')
```

- *<переменная>*

  - указывается имя переменной. При исполнении оператора `write` будет напечатано значение переменной. Если это число, то оно представлено в экспоненциальной форме с максимально возможным количеством цифр после десятичной точки. Например:

```
. . .
a:=pi;
write (a); ————→ будет напечатано:  3.1415926536E+00
```

- *<переменная>*: *<формат>*

  Если после имени переменной стоит двоеточие и целое число (формат), то значение переменной по-прежнему печатается в экспоненциальной форме, но общее количество знаков в записи числа не превышает указанного формата. Например:

```
. . .
a:=pi;
write (a:9); ————→ будет напечатано:  3.14E+00[1]
```

[1] При печати любого числа одна из отведенных форматом позиций всегда отводится под знак числа. Для положительных чисел знак “+” опускается.

Экспоненциальная форма вещественного числа иногда затрудняет восприятие. Поэтому в Pascal'е предусмотрена возможность изменения вида распечатываемого числа.

<*переменная*> : <*формат*> : <*дробная часть*>

В такой конструкции явно указывается, сколько позиций из заданных форматом отводится под запись дробной части выводимого числа. Если количество знаков в числовом значении окажется меньше указанного формата, то перед числом будут оставлены пробелы.

ПРИМЕРЫ

Согласно операторам

```
. . .
a:=pi;
write (a:10:4);
. . .
```

будет напечатано значение:

```
( ( (-3.1416
```

т.к. реально число заняло 7 позиций, то слева от него будут оставлены пробелы.

Перечисленные выше варианты элементов вывода в операторе `write` не исчерпывают всех возможностей. В частности, в любом из них вместо переменной может стоять арифметическое выражение, например:

```
write (Mass/MolMass*Na:15:5, '  молекул')
```

Кроме оператора `write`, для вывода информации в выходной поток часто используют оператор `writeln`:

`writeln` (<*элемент*>,<*элемент*>, … , <*элемент*>)

Правила формирования элементов вывода для этого оператора идентичны описанным выше правилам для `write`. Принципиальное отличие этих операторов друг от друга заключается в том, что оператор `writeln` после перечисленных в скобках элементов вывода информации передает в выходной поток символ новой строки.

При оформлении результатов расчета по программе для вставки пустой строки можно использовать оператор:

```
writeln;
```

без списка элементов. В этом случае в выходной поток информации будет передан только символ новой строки.

ПРИМЕРЫ

Согласно операторам:

```
. . .
a:=-pi;
write (a:10:4);
write ((a+1):10:4);
. . .
```

будут напечатаны значения:

```
( ( (-3.1416 ( ( (-2.1416
```

```
. . .
a:=-pi;
writeln (a:10:4);
writeln ((a+1):10:4);
. . .
```

будут напечатаны значения:

```
   ( ( (-3.1416
( ( (-2.1416
```

Оператор присвоения значения переменной

В общем случае оператор присвоения имеет вид:

<*переменная*> **:=** <*выражение*>

Знак присвоения состоит из двух символов: двоеточия и равенства.

Например,

```
Mols := Mass / MolMass*Na
```

Действие оператора заключается в том, что сначала на основании текущих значений переменных вычисляется выражение, стоящее справа от этого знака, а получившийся результат становится новым значением (присваивается) переменной, указанной слева. Например, в результате исполнения оператора присвоения:

```
Sum := Sum+Term
```

значение переменной `Sum` увеличится на значение `Term`.

Арифметические выражения

Арифметическое выражение - это задание на вычисление по формуле. Значением выражения является результат вычисления. Выражение записывается в строчку и строится из:

- констант;
- переменных;
- знаков операций;
- вызовов функций;
- круглых скобок.

В Pascal’е определены следующие арифметические операции:

+	сложение;	*	умножение;	**div**	деление нацело;
-	вычитание;	**/**	деление	**mod**	остаток от деления.

Все вычислительные операции внутри арифметического выражения выполняются последовательно, согласно своему приоритету:

1. Вызовы функций;
2. Умножение и деление;
3. Сложение и вычитание.

Если имеет место несколько операций одного ранга, они выполняются последовательно слева направо. Изменить порядок выполнения операций внутри арифметического выражения можно с помощью круглых скобок. Например, вычисление арифметического выражения `(A+B)*C` даст произведение суммы значений `A` и `B` на значение `C`, а арифметического выражения `A+B*C` - сумму значения `A` и произведения `B*C`.

Математическая запись	Арифметическое выражение на Pascal'е
$\frac{a \cdot c}{b}$	`A*C/B` или `a/b*c`
$\frac{a}{b \cdot c}$	`A/B/C` или `A/(B*C)`
$\frac{a+b}{2} \cdot c$	`(a+b)/2*c` или `(a+b) *c/2`

Более сложные вычисления выполняют с помощью функций. Вызов функции представляет собой имя функции, вслед за которым в круглых скобках указываются параметры.

ПРИМЕРЫ

`ln(a+b)` - производится сложение значений переменных `a` и `b`; от результата берется натуральный логарифм.

`sin(a)*sqrt(3)/2+cos(a)/5` - вычисление по формуле:

$$sin(a) \cdot \frac{\sqrt{3}}{2} + \frac{cos(a)}{5}$$

Стандартные математические функции перечислены в приложении 2.

Как вычислить на Pascal'е выражение x^Y? Вот несколько вариантов:

- `exp(y*ln(x))`, исходя из тождества $x^Y = e^{Y \ln X}$. Однако, это справедливо лишь для $x > 0$;
- x^2 записывается как `x*x` или `sqr(x)`;
- x^4 можно записать как `sqr(sqr(x))`;
- x^3 - это x*`sqr(x)`;
- $x^{1/2}$- это `sqrt(x)`, так что $x^{3/2}$ можно вычислить как `x*sqrt(x)`, $x^{1/4}$ - как `sqrt(sqrt(x))` и т.д.;

Тема 2

Организация циклов в программах.

Оператор For … do …

Составной оператор

Очень часто в программе необходимо выполнить одну и ту же последовательность операторов несколько раз. В этих ситуациях используют операторы повторений (циклов).

Оператор For … do …

Оператор цикла **for** в общем виде записывается так:

for <*параметр*> **:=** <*нач.знач-е*> **to** <*кон.знач-е*> **do** <*оператор*>

Элемент оператора **for**:	Что может быть использовано:	Допустимый тип:
<*параметр*>	Только переменная	INTEGER
<*нач. знач-е*> <*кон. знач-е*>	Константы, переменные или арифметические выражения, результат которых - целая величина.	INTEGER
<*оператор*>	Вложенный оператор, составляющий тело цикла	

Программа 2.

```
(* Нахождение суммы всех целых чисел от 1 до N *)
uses
  WinCrt;
var
  I,n: integer;
```

```
  s: integer;
begin
  write(' Введите n:');
  read (n);
  s:=0;
  for  i:= 1 to n do
    s:=s+i;
  writeln(' Сумма целых чисел от 1 до', n:3, 'равна:',
          s:6);
end.
```

В этой программе в первой строке использован комментарий - поясняющая надпись для пользователя, которая может находиться в любом месте программы. Комментарий начинается символами (* и заканчивается *). Он может занимать неограниченное количество строк и не влияет на исполнение программы.

Исполнение оператора **for**

Исполнение оператора for начинается с того, что его параметр принимает указанное начальное значение и вложенный оператор исполняется первый раз. После этого значение параметра цикла автоматически увеличивается на единицу [1], и вложенный оператор выполняется снова. Так происходит до тех пор, пока параметр цикла не станет равным заданному в строке **for** конечному значению. При этом цикл выполняется последний раз, а затем исполнение переходит к следующему оператору программы. В программе 2 цикл **for** выполняется n раз (n - вводимое при запуске программы целое число). По завершении

[1] Самостоятельное переопределение значения параметра цикла внутри этого цикла является грубой ошибкой!

цикла в переменной S будет накоплена сумма всех целых чисел от 1 до n.

В операторе **for** вместо слова **to** может стоять слово **downto**. В этом случае при каждом прохождении цикла значение счетчика будет уменьшаться на 1. Если в программе 2 на месте приведенного выше оператора **for** записать:

```
for i := n downto 1 do
```

результат ее выполнения не изменится.

Составной оператор

Часто в цикле необходимо повторить не один, а несколько операторов. Такая последовательность операторов должна быть объединена в составной оператор служебными словами **begin** и **end**. Проиллюстрируем сказанное программой 2а:

```
(* Нахождение сумм всех целых чисел и их квадратов от 1 до N *)
  uses
    Wincrt;
  var
    i, n: integer;
    s1, s2: integer;
  begin
    write (' n?');
    read (n);
    s1:=0; s2:=0;
```

```
  for i:=1 to n do
  begin
    s1:=s1+i;
    s2:=s2+i*i;
  end;
  writeln (' Сумма целых чисел от 1 до',n:3, ' равна:',
s1:6);
  writeln;  (* Пропуск пустой строки при печати *)
  writeln (' Сумма квадратов этих чисел равна:', s2:6);
end.
```


СОСТАВНОЙ ОПЕРАТОР

Тема 3

Операторы повторений

Repeat ... until ...;

While ... do …;

Правила организации вложенных циклов

Логические величины

Величины логического типа (`BOOLEAN`) - особый вид величин в Pascal'е. Они могут принимать только два значения: `True` (истина) и `False` (ложь). Где и как используются эти величины? Например, результатом выполнения операции отношения (сравнения между собой значений двух или более величин) всегда является логическая величина. В Pascal'е предусмотрены следующие операции отношения:

=	равно	>	больше
<>	не равно	>=	больше или равно
<	меньше	<=	меньше или равно

Операции отношения применимы к операндам уже рассмотренных выше типов: вещественного (`REAL`) и целого (`INTEGER`).

Кроме операций отношения определены следующие логические операции:

not - логическое отрицание ("не")

and - логическое умножение ("и")

or - логическое сложение ("или")

xor - исключающее или.

Разберите следующие примеры:

Условие	Логическое выражение
$a \neq b$	`a <> 0`
$a \geq b + c$	`a <= b+c`
$a^2 < \sqrt{b}$	`a*a < sqrt(b)`
$0 \leq \alpha < 2\pi$	`(alpha >= 0) and (alpha < 2*pi)`

Логические операции применимы к операндам логического (BOOLEAN) типа и их результатом также является логическая величина:

A	B	not A	A and B	A or B	A xor B
True	True	False	True	True	False
True	False	False	False	True	True
False	True	True	False	True	True
False	False	True	False	False	False

Отметим, что логические операции имеют более высокий приоритет, чем арифметические операции и операции отношения. Поэтому в случае составления сложных логических выражений необходимо использовать круглые скобки.

П Р А В И Л Ь Н О ☺	Н Е П Р А В И Л Ь Н О ☹
`(x=0) or (y=0)`	`x or y = 0`
`not (a > eps)`	`x = 0 and y = 0`
`((f1<=0) and (f2>0)) or ((f2<=0) and (f1>0))`	`f1<=0 and f2>0 or f2<=0 and f1>0`

Операторы повторений, использующие логические величины

Мы рассмотрели оператор повторений **for**, исполнение которого связано с изменением параметра цикла. Иногда использование этого оператора в программах оптимально (например, когда целочисленный параметр цикла **for** используется в качестве некоторого счетчика и т.п.), но чаще более универсальной является организация циклов с помощью двух других операторов повторений, использующих логические величины. Первый из них имеет вид:

while <*условие*> **do** <*оператор*>

На месте условия может стоять любое выражение логического типа. До тех пор пока его значение истинно, повторяется выполнение оператора, стоящего за служебным словом **do** (обычно используется составной оператор). Как только условие перестанет выполнятся, произойдет выход из цикла.

Приведем в качестве примера программу суммирования ряда с заданной точностью $(\varepsilon < 10^{-5})$: $\sum_{i=1}^{\infty} \frac{1}{i!}$

```
uses
  WinCrt;
const
  eps=1e-5;
var
  i: integer;
  a, sum: real;
begin
  sum:=0;
  a:=1;  i:=0;
```

```
  while a>eps do
    begin
      i:=i+1;
      a:=a/i;
      sum:=sum+a;
    end;
  writeln('Сумма:', sum:12,'найденная с точностью', eps:8,
  'включает',i:5,'слагемых');
end.
```

Может ли вложенный оператор цикла **while** ни разу не выполниться в программе? В каких случаях?

Другой оператор повторений, использующий логические величины, имеет вид:

repeat <*операторы*> **until** <*условие*>

Отличие его от оператора **while** в том, что сначала происходит выполнение вложенных операторов[1] цикла и лишь после этого проверяется поставленное условие. Если его результат “истина”, происходит выход из цикла, если “ложь” - исполнение вложенных операторов повторяется.

Вот как может выглядеть вариант предыдущей программы для суммирования ряда с заданной точностью:

```
uses  WinCrt;
const  eps=1e-5;
var
  i: integer;  a, sum: real;
begin
```

[1] На месте слова <*операторы*> обычно стоят несколько операторов. Слова `Repeat .. until` выполняют здесь роль `begin .. end` для составного оператора.

```
  sum:=0;
  a:=1;  i:=0;
  repeat
    i:=i+1;
    a:=a/i;
    sum:=sum+a;
  until a<eps;
  writeln('Сумма:', sum:12,'найденная с точностью', eps:8,
          'включает',i:5,'слагемых');
end.
```

Тема 4

Организация разветвлений в программах; оператор If ... then ... else ...

Условный оператор

Для организации разветвлений в программах используется условный оператор:

if <*условие*> **then** <*оператор* 1> **else** <*оператор* 2>

if , then , else являются зарезервированными словами.

<*условие*> любое выражение логического типа (стр. 23).

Программа 3.

```
(* Проверка на неотрицательность вводимого с клавиатуры
 числа *)
uses
  WinCrt;
var
   a: real;
begin
   write (' Введите число:');
   readln (a);
   if a >= 0 then
     writeln (' число', a, 'неотрицательно')
     else
       writeln ('число', a, 'отрицательно');
end.
```

Исполнение логического оператора **if**

Прежде всего, проверяется <*условие*>. Если результат проверки - “истина”, то будет выполнен <*оператор* 1>, указанный за служебным словом **then.** Стоящий после слова **else** <*оператор* 2> пропускается, а исполнение переходит к следующему за **if** оператору программы. Если результатом проверки условия является “ложь”, все происходит наоборот: игнорируется <*оператор* 1>, а будет выполнен стоящий после **else** <*оператор* 2>.

Замечания:

1. Перед служебным словом **else** никогда не ставится точка с запятой.
2. Если на месте <*оператор* 1> или <*оператор* 2> алгоритмом программы предусмотрено выполнение не одного, а нескольких операторов, используют составной оператор.

Иногда в программах удобнее использовать сокращенный вариант оператора **if**:

if <*условие*> **then** <*оператор*>

Если поставленное условие не выполнено (результат логического выражения - “ложь”), то <*оператор*>, следующий за **then**, будет просто пропущен.

Программа 4.

```
(* Подсчет количества неотрицательных чисел среди 10
   вводимых с клавиатуры *)
uses
  WinCrt;
var
  a: real;
  i, count: integer;
begin
  count:=0;
  for i:=1 to 10 do
  begin
    write (' число ', I:2,'?'); readln (a);
    if a>=0 then count:=count+1;
  end;
  writeln;
  writeln(' Неотрицательных чисел:', count:10:5);
end.
```

Тема 5

Организация и использование функций (Function) и процедур (Procedure) в программах

Процедуры и функции представляют собой отдельные фрагменты программ, оформленные определенным образом и реализующие часть общего алгоритма, и поэтому могут быть названы *подпрограммами*.

Каждая подпрограмма имеет свое имя для обращения к ней. Описание подпрограммы состоит из *заголовка* и *тела*. В заголовке кроме имени указываются необходимые параметры. Тело подпрограммы имеет в точности такую же структуру, как и программа [1], только в конце после слова **end** ставится точка с запятой.

Описание всех процедур и функций программы должно находиться в разделе описаний до раздела операторов.

Function

Заголовок описания функции имеет вид:

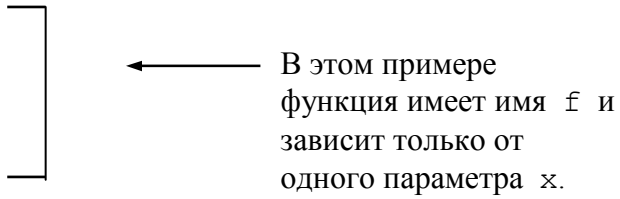
function <*имя*> (<*параметры*>): <*тип результата*>;

Разберите пример:

```
(* Программа с использованием функции *)
uses
  Wincrt;
const
  a:=0.5;
  b:=1.5;
var
```

[1] В описание процедур и функций в свою очередь также при необходимости можно включать “свои” подпрограммы.

```
  s,dx: real;
  n,I: integer;
Function f(x:real): real;
begin
  f:=1/x;
end;
begin
  readln(n);
  dx:=(b-a)/n;
  s:=0.5*(f(a)+f(b));
  for I:=1 to n-2 do
    s:=s+f(a+I*dx);
  writeln(s*dx);
end.
```



После своего исполнения функция должна возвращать некоторое значение, например, в операторе

```
s:=0.5*(f(a)+f(b));
```

функция с именем `f` вызывается дважды, после чего складываются два возвращенных значения: `f(a)` и `f(b)`. Возвращаемое значение передается через имя `f`, для чего в теле описания функции должен присутствовать соответствующий оператор присвоения, например, `f:=1/x`. В приведенном выше операторе присвоения `f(a)` и `f(b)` – соответствующие значения функции `f(x)` при $x = a$ и $x = b$.

Параметр `x`, указанный в заголовке описания функции, называется формальным, параметры `a` и `b`, указанные при вызове функции, – фактическими.[1]

[1] В теле функции описывается формальная последовательность действий, фактически эти действия будут происходить с величинами, заданными в качестве параметров при вызове этой функции.

Procedure

Заголовок описания процедуры имеет вид:

Procedure <*имя*> (<*параметры*>)

Пример:

```
(* Использование процедуры для печати заголовка таблицы *)
Uses
  WinCrt;
var
  i: integer;
  h,a: real;
procedure Header (n: integer);
var
  i: integer;

begin
  writeln;
  writeln (' Таблица №', n);
  for i:=1 to 20 do
    write ('-');
  writeln;
end;
function f1 (x:real);
begin
  f1:=sin(x);
end;
function f2 (x:real);
begin
  f2:=cos(x);
end;
begin
  Header(1);
```

```
  writeln (' Введите начальное значения аргумента и шаг');
  for i:=0 to 10 do
    writeln (a+i*h:8:3, f1(a+i*h):12:7);
  Header(2);
  for i:=0 to 10 do
    writeln (a+i*h:8:3, f2(a+i*h):12:7);
end.
```

Вызов процедуры представляет собой отдельный оператор, состоящий из имени процедуры и списка фактических параметров. Вызов процедуры отделяется от других операторов программы точкой с запятой (например, `Header(1);` Здесь `Header` - имя процедуры, `1` - фактический параметр, соответствующий формальному параметру `n`).

Отметим, что процедура может и не иметь параметров.

Формальные и фактические параметры

Необходимо соблюдать строгое соответствие между формальными и фактическими параметрами по их количеству, типам и порядку следования в списке.

Формальным параметром может быть только переменная, тип которой обязательно указывается в заголовке описания функции или процедуры.

Если формальных параметров несколько и они имеют один и тот же тип, то их удобно объединить в группу:

<имя> , *<имя>* ,…, *<имя>* : *<тип>*;

Если в списке формальных параметров несколько подобных групп, они отделяются друг от друга точкой с запятой.

Фактическим параметром может быть константа, переменная или выражение [1], но в любом случае фактический параметр обязательно должен совпадать по типу с соответствующим ему формальным.

Неизменяемые и изменяемые (**`var`**-) параметры

В приведенных выше примерах программ и в процедурах, и в функциях использовались неизменяемые параметры. Такой способ называется *передачей параметров по значению*. Другими словами, при вызове процедуры или функции на место формального параметра подставляется текущее значение соответствующего ему фактического параметра и именно с этим значением выполняются действия, описанные в теле подпрограммы. По завершении подпрограммы никаких изменений в значении фактического параметра в вызывающую программу не передается: его значение остается таким, каким оно было перед вызовом процедуры или функции.

Другой способ взаимодействия параметров называется *передачей параметров по ссылке*. В этом случае перед именами формальных параметров в заголовке описания процедуры или функции следует указывать служебное слово **`var`**. При выполнении процедуры или функции вместо формального **`var`**-параметра подставляется переменная соответствующего фактического параметра. Все изменения, произошедшие с этой переменной в подпрограмме, будут переданы в вызывающую программу.

[1] В этом случае вычисляется результат выражения, который и ставится в соответствие формальному параметру.

Неизменяемые параметры (передача параметров по значению)	**Изменяемые (var-) параметры** (передача параметров по ссылке)

var
  a:real;
procedure one (x: real);
begin
  writeln(x);
  x:=1;
  writeln(x);
end;
begin
  a:=2;
  one (a);
  writeln (a);
end.
	var
	  a:real;
	procedure one (var x: real);
	begin
	  writeln(x);
	  x:=1;
	  writeln(x);
	end;
	begin
	  a:=2;
	  one (a);
	  writeln (a);
	end.

Результат работы программы: 2 1 2	Результат работы программы: 2 1 1

Глобальные и локальные переменные

Все переменные, указанные в разделе описаний программы, доступны ее процедурам и функциям. Такие переменные называются *глобальными.*

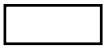
Переменные, описанные внутри процедуры или функции, являются *локальными,* т.е. существующими только на время работы этой подпрограммы (например, переменная `i`, описанная в процедуре `Header` из программы на стр. 33).

Если имя локальной переменной совпадает с именем глобальной переменной, то доступ к этой глобальной переменной потерян для данной подпрограммы (примером может служить та же переменная `i` из процедуры `Header`).

Тема 6

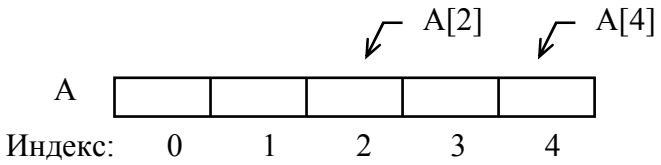
Массивы переменных. Взаимодействие программы с внешними файлами данных

Простая переменная - это ячейка памяти, обозначенная именем.

< Имя > 

Массив переменных (переменные с индексом) - это несколько ячеек памяти, обозначаемых одним идентификатором. Отдельные ячейки памяти в массиве различаются *индексом.* В качестве индекса обычно используют целые величины.

Например, массив из 5 ячеек памяти, обозначаемый идентификатором A , можно представить следующей схемой:



Описание массива переменных должно присутствовать в секции описания переменных. Синтаксис описания соответствует схеме:

<*имя*> : **array** [<*нижний предел*> **..** <*верхний предел*>] of <*тип*> ;

Здесь:

<*имя*>	- имя, определяемое как массив переменных;
<*нижний предел*> и <*верхний предел*>	- пределы изменения индекса массива
<*тип*>	- тип элементов массива.

Например, описание массива A из пяти ячеек памяти типа REAL может выглядеть так:

```
A: array [0..4] of real;
```

Количество ячеек в массиве определяется величинами <*нижний предел*> и <*верхний предел*>, в качестве которых можно использовать числовые или символьные константы целого типа, например:

```
const
  N = 64;
var
  Y: array [0..N] of real;
```

Нельзя в качестве пределов в описании массивов использовать переменные. Это связано с тем, что память под массивы выделяется на стадии трансляции, когда значения переменных еще не известны. Поэтому, если требуемый размер массива заранее неизвестен, выделяют сразу "достаточно большое" количество ячеек.

Используются элементы массива в точности так же, как обычные переменные - в операторах присвоения, в качестве параметров процедур и функций и т.п. При этом обозначение элемента массива состоит из имени массива и индекса в квадратных скобках. Например, обозначение Y[2] для описания, приведенного выше, соответствует третьему по порядку элементу массива Y.

Индексом элемента массива может быть вычисляемое в процессе работы программы арифметическое выражение.

Пример фрагмента программы:

```
...
A[0]:= 1;
for i:=1 to 4 do A[i]:=A[i-1]+i;
...
```

После его исполнения в массиве A окажутся числа: 1, 2, 4, 7, 11.

Массивы переменных удобно использовать для хранения вычисляемых значений функции, например:

```
...
read(x0,dx);
for i:=1 to 20 do Y[i]:=func(x0+i*dx);
...
```

Поиск минимального или максимального по значению элемента массива

Пусть в некоторый массив помещен ряд чисел. Как найти среди них то, которое имеет наименьшее значение?

Идея соответствующего алгоритма проста. Вначале предположим, что таким элементом является первый и запомним его. Затем будем сравнивать его с остальными. Если по мере просмотра обнаружится меньший - заменим запомненный на вновь найденный и в дальнейшем будем сравнивать уже с ним. На Паскале это можно записать так:

```
...
Ymin := Y[0];
for i:=1 to N do
if Y[i]< Ymin then Ymin:=Y[i];
...
```

Работа с файлами

Часто в программе ставится задача обработки большого объема информации с использованием некоторой математической модели. Очевидно, что в этих случаях удобно организовать хранение данных в массивах переменных. Считывание данных в массив обычно организуется осуществить через цикл **for**, например:

```
var
  n,i: integer;
  X,Y: array [1..100] of real;
  . . .
  . . .
begin
  writeln ('Задайте количество элементов в массивах X и Y',
  '(не более 100)');
  readln(n);
  for i:=1 to n do
    readln(X[I],Y[I]);
  . . .
  . . .
```

Было бы идеально, чтобы написанная программа при первом же запуске выдала безошибочный результат. Если же возникнет необходимость внести в исходные данные какие-либо исправления, потребуется при повторных запусках снова набирать все значения, что повышает вероятность ошибок при вводе информации. В подобных ситуациях в программе организуется работа с файлами данных, подготовленными до запуска использующей их программы.

Обычно (если не указано иначе) считывание информации происходит из файла клавиатуры, а вывод - в файл экрана. За связь программы с этими файлами отвечают файловые переменные `input` (ввод) и `output` (вывод), которые заранее зарезервированы для этих действий.

Как организовать доступ программы к другим файлам? Можно воспользоваться одной из указанных файловых переменных, переключив ее со стандартного файла на другой, необходимый программе. Можно же ввести новую файловую переменную, указав в секции описания переменных в программе, например, так:

```
var
  f:text;
```

Связывание файловой переменной с именем нужного файла осуществляет процедура:

assign (<*файловая переменная*>,<*имя файла*>)

Примеры

`assign(f,'1.dat');` - устанавливает через файловую переменную `f` связь с файлом `1.dat`

Перед организацией считывания из этого файла данных необходимо вызвать специальную процедуру:

`reset` (<*файловая переменная*>);

Например,

```
reset(f);
```

В операторах считывания из файла необходимо не забывать указывать ту же файловую переменную:

ПРИМЕРЫ

```
Read (f,a,b);
```

- Этот оператор означает считывание значений переменных `a` и `b` из файла, который связан с данной программой через файловую переменную `f`.

Для переключения на работу с другим файлом данных (например, вернуться к работе с клавиатурой) можно воспользоваться в программе строками:

```
...
close(f);
assign(input, 'con');
reset(input);
...
```

Установить связь с файлом ‘con’-файлом клавиатуры

Открыть этот файл на считывание

Для записи какой-либо информации в файл после установки связи с этим файлом (процедура `assign`) необходимо вызвать встроенную процедуру

`rewrite` (<*файловая переменная*>);

а в последующих операторах вывода `write` или `writeln` список переменных начать с указания этой файловой переменной, например:

```
...
assign(f,'2.dat');
rewrite(f);
writeln(f,'Результаты:',a:10,b:10);
...
```

Обратите внимание на то, что если в указанном файле хранилась какая-либо информация, она будет потеряна, т.к. при выполнении процедуры `rewrite` старый файл уничтожается.

Чтобы дописать информацию в непустой файл, вместо процедуры `rewrite` можно воспользоваться процедурой:

`append (<`*файловая переменная*`>);`

Как подготовить файл для считывания из него информации

Файл подготавливается средствами любого текстового редактора (например, встроенного редактора Turbo Pascal'я). Информация (например, числовые значения переменных) должна располагаться в строках согласно правилам считывания их операторами `read` (см. стр.11). Порядок строк ввода строго соответствует порядку операторов считывания в программе.

Если при подготовке файла данных были допущены ошибки, их легко устранить соответствующими командами текстового редактора до запуска программы.

Численные методы.
Краткий обзор и основные математические формулы.

Часто при проведении вычислительного эксперимента возникают ситуации, когда точное решение той или иной математической задачи невозможно. Например, в вычислениях присутствует некоторая функция, которая выражена не аналитически (в элементарных функциях), а задана в виде таблицы числовых значений, поставленных в соответствие дискретному множеству значений аргумента. Другой случай связан с отсутствием конкретной формулы для непосредственного решения поставленной задачи. В этих и аналогичных ситуациях для математической обработки данных используются численные (приближенные) методы решения.

Обычно при проведении расчетов с использованием численных методов задается необходимая точность вычислений, с которой сравнивается очередное полученное решение.

Различают абсолютную и относительную погрешности вычислений.

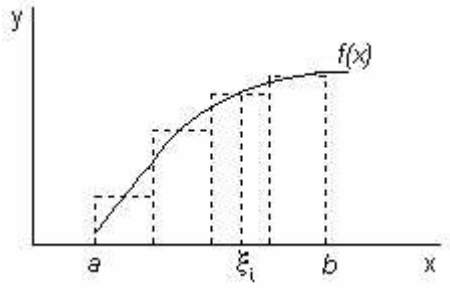
Абсолютная погрешность - это абсолютное значение разности между точным и приближенным решениями.

Относительная погрешность есть отношение разности точного и приближенного решений к точному решению, взятое по абсолютному значению.

Методы численного интегрирования

Пусть на отрезке *[a,b]* задана функция *f(x)*. С помощью точек $x_0, x_1, x_2, \ldots, x_n$ разобьем отрезок *[a,b]* на *n* элементарных отрезков $[x_{i-1}, x_i]$ (i=1,2,...,n), причем $x_0=a$, $x_n=b$. На каждом из этих отрезков выберем точку ξ_i и найдем произведение s_i значения функции $f(\xi_i)$ на длину элементарного отрезка $\Delta x_i = x_i - x_{i-1}$:

$$s_i = f(\xi_i)\,\Delta x_i \qquad (1)$$



Сумма таких произведений:

$S_n = \sum_{i=1}^{n} s_i$ называется интегральной суммой.

Определенным интегралом от функции f(x) на отрезке [a,b] называется предел интегральной суммы при неограниченном увеличении числа точек разбиения. При этом длина наибольшего из элементарных отрезков стремится к нулю:

$$\int_a^b f(x)dx = \lim_{\max \Delta x_i \to 0} \sum_{i=1}^{n} f(\xi_i)\Delta x_i$$

Во многих случаях, когда подынтегральная функция задана в аналитическом виде, определенный интеграл вычисляют по формуле Ньютона-Лейбница, согласно которой определенный интеграл равен приращению первообразной *F(x)* на отрезке интегрирования. На

практике такое решение часто неприемлимо по двум основным причинам:

1. Вид функции не допускает непосредственного интегрирования, т.е. первообразную нельзя выразить в элементарных функциях.
2. Значения функции заданы *f(x)* заданы таблично (множество x_i конечно).

В этих случаях применяют методы численного интегрирования.

Важным частным случаем в методах численного интегрирования является тот, когда величина элементарного отрезка Δx_i постоянна и может быть вынесена за знак интегральной суммы. Эта величина называется шагом интегрирования и обычно обозначается h.

I. **<u>Метод прямоугольников</u>** непосредственно использует замену определенного интеграла интегральной суммой. В качестве точек ξ_i могут выбираться левые (x_{i-1}) или правые (x_i) границы элементарных отрезков. Обозначив $y_i = f(x_i)$, расчетные формулы можно записать:

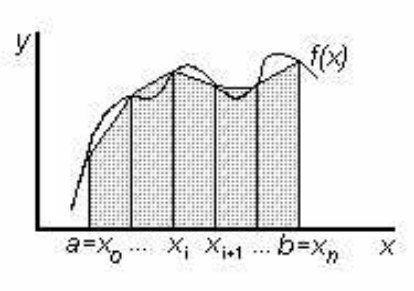
$$\int_a^b f(x)dx \approx h\,(y_0 + y_1 + y_2 + \text{K} + y_{n-1}) = h\sum_{i=0}^{n-1} y_i$$

при выборе левых границ или

$$\int_a^b f(x)dx \approx h\,(y_1 + y_2 + y_3 + \text{K} + y_n) = h\sum_{i=1}^{n} y_i$$

при выборе правых границ.

II. В **<u>методе трапеций</u>** график функции *f(x)* аппроксимируется ломаной, соединяющей точки с координатами (x_i, y_i).



Искомое значение определенного интеграла представляется в виде суммы площадей трапеций, построенных на каждом из элементарных отрезков:

$$\int_a^b f(x)dx \approx h\left(\frac{1}{2}y_0 + y_1 + y_2 + \text{K} + y_{n-1} + \frac{1}{2}y_n\right) = h\left(\frac{y_o + y_n}{2} + \sum_{i=1}^{n-1} y_i\right)$$

Здесь y_0 и y_n - значения функции *f(a)* и *f(b)* соответственно.

III. В **<u>методе парабол</u>** (**формула Симпсона**) на каждом из элементарных отрезков по трем известным значениям функции $f(x_i)$ строится парабола, заданная уравнением ax^2+bx+c.

Формула для нахождения определенного интеграла может быть выведена из условия равенства значений: $y_i = ax_i^2+bx_i+c$:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}(y_0 + y_n + 4(y_1 + y_3 + y_5 + \text{K} + y_{n-2}) + 2(y_2 + y_4 + y_6 + \text{K} + y_{n-1})) =$$

$$= \frac{h}{3}(y_0 + y_n + 4\sum_{i=1}^{(n-1)/2} y_{2i-1} + 2\sum_{i=1}^{(n-1)/2} y_{2i})$$

Численное решение нелинейных уравнений

Уравнения:

линейное: $a + bx = 0$;

квадратное: $a + bx + cx^2 = 0$;

трансцендентное: $e^x + x = 0$

В общем виде любое уравнение записывается как: $f(x) = 0$.

Например, уравнение $\frac{x^5}{e^{(1+x)}} = 1 - x$ в общем виде будет выглядеть:

$x^5 + e^{(1+x)} \cdot x - e^{(1+x)} = 0$

Решение уравнений

> Решить уравнение - значит найти значение переменной x, при котором заданная функция равна нулю.

Для решения линейных, квадратных и кубических уравнений существуют аналитические формулы. Однако, например, третье уравнение из приведенных выше примеров аналитически решить невозможно. Это так называемое *иррациональное* уравнение, и для его приближенного решения можно воспользоваться численными методами. Существует большое количество методов численного решения нелинейных уравнений, которые различаются точностью получаемого результата, скоростью его получения, надежностью и сложностью.

Рассмотрим два простейших метода:

- *метод дихотомии (деления отрезка пополам)*
- *метод Ньютона (метод касательных).*

I. Метод дихотомии

Пусть функция *f(x)* отрицательна в точке *a* *(f(a) < 0)*, положительна в точке *b (f(b) > 0)*, и непрерывна на отрезке *[a,b]*. Тогда, очевидно, на *[a,b]* график функции пересекает ось *X*, т.е. на этом отрезке имеется *корень уравнения* - точка, в которой *f(x) = 0*. Тот же вывод следует, если *f(a) > 0* и *f(b) < 0*. В обоих случаях в точках *a* и *b* функция *f(x)* принимает значения разных знаков.

Если нам известен хотя бы один такой отрезок, пусть и большой длины, мы можем построить процедуру быстрого и сколь угодно точного поиска корня уравнения. Найдем значение функции в точке *c*, находящейся в середине отрезка: $c = \frac{a+b}{2}$. Знак *f(c)* совпадает со знаком функции на одном из концов отрезка и противоположен знаку функции на другом конце. Предположим, что разные знаки *f(x)* в точках *a* и *c*. Значит на *[a,c]* наверняка есть искомый корень уравнения.

Таким образом, мы получили задачу, эквивалентную исходной, но теперь длина отрезка, на котором, как нам известно, находится корень, в два раза короче. Отрезок *[a,c]* опять можно разделить пополам и оставить в рассмотрении только один из двух получившихся (учитывая знаки значений *f(x)* на концах этих отрезков). Выбранный отрезок вновь разделить, и продолжать так до тех пор, пока отрезок, на котором находится корень, не станет достаточно мал.

Как написать программу на Pascal'е, соответствующую этому методу? Отметим основные моменты в оформлении такой программы:

– Прежде всего, функцию, корень которой мы ищем, лучше всего описать в виде отдельной `function`.
– Во время выполнения программа должна запросить начальные значения границ отрезка: *a* и *b*.
– Далее нужно проверить, что `f(a)` и `f(b)` действительно разных знаков. Это условие можно записать двумя разными способами:

  1) `(f(a)<=0) and (f(b)>0)) or ((f(b)<=0) and (f(a)>0))`

  2) `f(a)*f(b)<=0`

  Если условие не выполнено, нужно только напечатать сообщение об ошибке, иначе можно приступать к поиску корня.
– Поиск проще всего оформить в виде цикла

```
repeat ... until abs(b-a)<eps
```

  Здесь `eps` - константа, равная, например, 10^{-5}.

  На каждой итерации цикла нужно вычислить значение точки *c* и сравнить знак функции в этой точке со знаком *f(a)*. Если знаки разные, о прежней точке *b* можно "забыть": `b:=c`. Иначе "забываем" о точке *a*. После окончания цикла остается лишь напечатать найденный корень. Ближе всего к корню будет, скорее всего, середина последнего отрезка *[a,b]*, длина которого не превышает заданной точности `eps`.

II. Метод касательных.

Метод Ньютона, или метод касательных, основан на вычислении не только значений функции, но и значений ее производной. Разложим *f(x)* в ряд Тейлора и отбросим члены ряда выше второго порядка:

$$f(x) = f(x_0) + f'(x_0) \cdot (x - x_0)$$

В этом приближении корень функции легко найти по формуле:

$$f(x) = 0 \quad \Rightarrow x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Для нелинейных функций это, конечно, не точное равенство, но способ получить значение, более близкое к корню. Таким образом, метод Ньютона состоит в построении последовательности итераций:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

. . .

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

. . .

На какой итерации завершить процесс нахождения корня? Возможны два варианта: либо когда значение функции будет достаточно мало по абсолютной величине: $|f(x_i)| < eps$, либо когда практически перестанет меняться значение x: $\left|\frac{f(x_i)}{f'(x_i)}\right| < eps$.

Преимущества и недостатки рассмотренных методов.

Метод дихотомии обладает 100% - ной надежностью: если найдены точки *a* и *b*, в которых функция имеет разные знаки, корень наверняка будет найден. Однако, иногда найти такие две точки бывает трудно.

Можно ли методом дихотомии найти корень уравнения $x^2 = 0$?

Метод Ньютона обладает более быстрой сходимостью и не накладывает ограничений на начальное приближение. Однако, для применения метода, кроме вычисления функции, требуется и вычисление производной. Кроме того, сходится этот метод не для всех функций.

Проверьте на практике применимость метода Ньютона для решения уравнения $arctg(10x) = 0$.

Линейная регрессия

В процессе обработки результатов эксперимента часто встает задача аппроксимации полученных данных линейной зависимостью. При этом обычно известно, что теоретически результаты должны укладываться на прямую линию, однако из-за ошибок эксперимента этого не происходит. В общем случае эта задача формулируется так. Имеется n пар чисел (x_i, y_i), $i = \overline{1, n}$, являющихся результатом эксперимента. Требуется найти коэффициенты прямой линии $y(x) = ax + b$, которая проходит ближе всего сразу ко всем экспериментальным точкам.

Прежде всего, следует математически определить критерий, по которому мы можем судить о "качестве" той или иной прямой линии. Мерой удаленности одной точки от прямой может служить величина отклонения $D_i(a,b) = y(x_i) - y_i = ax_i + b - y_i$, которая зависит от параметров a и b. Для характеристики "суммарного" отклонения прямой от всех точек можно использовать сумму квадратов индивидуальных отклонений.

$$S(a,b) = \sum_{i=1}^{n} D_i^2 \qquad (1)$$

Подумайте, почему нельзя использовать просто сумму отклонений.

Таким образом, мы получили "качество" прямой линии как функции параметров последней. Для того чтобы найти параметры, обеспечивающие минимальное отклонение прямой от

экспериментальных точек, нужно найти минимум функции $S(a,b)$.

Пусть нам известно оптимальное значение параметра *a*. Тогда *S* зависит только от *b*, и для того, чтобы найти ее минимум, надо приравнять нулю частную производную:

$$S_b' = \sum_{i=1}^{n} 2D_i D_{i_b}' = 2 \cdot (a\sum_{i=1}^{n} x_i + bn - \sum_{i=1}^{n} y_i) = 0 \qquad (2)$$

Отсюда получаем

$$b = \frac{\sum_{i=1}^{n} y_i}{n} - a\frac{\sum_{i=1}^{n} x_i}{n} = \bar{y} - a\bar{x} \qquad (3)$$

Подставим получившееся выражение для b в (1), и получим "качество" как функцию только от a:

$$S(a) = \sum_{i=1}^{n} (a \cdot (x_i - \bar{x}) - (y_i - \bar{y}))^2 \qquad (4)$$

Приравняв нулю производную от этой функции, получим выражение для оптимального значения *a*:

$$a = \frac{\sum_{i=1}^{n} (x_i - \bar{x}) \cdot (y - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \qquad (5)$$

Теперь мы можем составить алгоритм расчета оптимальных значений a и b:

– вычислить средние значения $\bar{x}$ и $\bar{y}$.

– вычислить суммы $S_1 = \sum_{i=1}^{n}(x_i - \bar{x})\cdot(y_i - \bar{y})$

и

$S_2 = \sum_{i=1}^{n}(x_i - \bar{x})^2$

– вычислить по (5) $a = \frac{S_1}{S_2}$

– вычислить по (3) $b = \bar{y} - a\bar{x}$

Коэффициент корреляции позволяет оценить степень линейной зависимости в исходных данных. Он может быть вычислен по формуле:

$$r = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{\left(n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2\right)\left(n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2\right)}}$$

Если $r = 1$, связь между x и y чисто линейная;

при $r = 0$ линейной связи в массиве данных нет.

Решение дифференциальных уравнений

Дифференциальное уравнение содержит функцию и ее производные. В общем виде дифференциальное уравнение можно записать так:

$$F(x, y(x), y'(x), y''(x), \text{K}\ , y^{(n)}(x)) = 0;$$

Номер *(n)* высшей производной, входящей в уравнение, называется порядком уравнения.

Примерами дифференциальных уравнений могут служить:

- кинетическое уравнение для зависимости концентрации вещества от времени:

$$c'(t) = k \cdot c^n(t) \tag{1}$$

- второй закон Ньютона для траектории частицы, т.е. координаты как функции времени:

$$x''(t) = \frac{f(x,t)}{m} \tag{2}$$

Решить дифференциальное уравнение - значит найти функцию *y(x)*, которая при подстановке в уравнение даст тождество.

Например, одним из решений уравнения *y'(x)=y(x)* является функция *exp(x)*, так как при подстановке ее вместо *y(x)* уравнение превращается в тождество.

Простейшим типом дифференциальных уравнений являются линейные дифференциальные уравнения первого порядка, которые в общем случае можно записать так:

$$y'(x) = F(y(x),x) \qquad (3)$$

Практическим примером может служить рассмотренное выше кинетическое уравнение (1).

Прежде чем обсуждать методы решения, необходимо отметить некоторые общие свойства этих решений.

Во-первых, уравнение (3) имеет бесконечно много решений. Например, уже рассмотренное уравнение $y'(x) = y(x)$ имеет в качестве решений не только $\exp(x)$, но и $2 \cdot exp(x)$, $-1.5 \cdot exp(x)$, и вообще, $a \cdot exp(x)$, где a - произвольное число.

Во-вторых, через каждую точку на плоскости проходит ровно одно решение уравнения (3).

Поэтому однозначной задачей является не просто поиск решения уравнения (3), но поиск решения, которое в точке x_0 принимает значение y_0.

$$\begin{cases} y'(x) = F(y(x),x) \\ y(x_0) = y_0 \end{cases}$$

Такая постановка называется ***задачей Коши***.

Метод Эйлера.

Метод Эйлера является одним из простейших методов приближенного решения задачи Коши. Пусть нам известно, что в точке

x_0 некоторая функция имеет значение $y(x_0)$ и производную $y'(x_0)$. Тогда, воспользовавшись определением производной, мы можем получить приближенное значение функции в соседней точке $x_1 = x_0 + h$:

$$y(x_1) = y(x_0) + h \cdot y'(x_0) \qquad (4)$$

На этой идее и основан метод Эйлера. Начиная с точки x_0 , в которой значение функции нам известно по условию, мы можем получить ее приближенное значение в $x_1 = x_0 + h$, затем в $x_2 = x_1 + h$ и так далее. Таким образом строится решение в любом интервале.

Для вычислений по (4) кроме значения самой функции, нам требуется еще и значение ее производной. Откуда его взять? Из дифференциального уравнения (3): подставив в его правую часть имеющиеся значения x_i и $y_i(x_i)$, мы определим, какое значение производной должно иметь искомое решение. Окончательные формулы для вычислений выглядят так:

$$\begin{cases} x_{i+1} = x_i + h \\ y_{i+1} = y_i + F(y_i, x_i) \end{cases}$$

В практических задачах чаще фигурируют не отдельные уравнения типа (1), а системы уравнений, решениями которых являются сразу несколько функций $y_1(x)$, $y_2(x)$,…, $y_m(x)$:

$$\begin{cases} y_1'(x) = F_1(y_1(x), y_2(x), \text{K}, y_m(x), x); & y_1(x_0) = y_{10} \\ y_2'(x) = F_2(y_1(x), y_2(x), \text{K}, y_m(x), x); & y_2(x_0) = y_{20} \\ \cdots \\ y_m'(x) = F_m(y_1(x), y_2(x), \text{K}, y_m(x), x); & y_m(x_0) = y_{m0} \end{cases}$$

В частности, уравнения высших порядков можно свести к системам уравнений первого порядка. Например, уравнение Ньютона (2) можно записать в виде системы из двух уравнений:

$$\begin{cases} x'(t) = v(t) \\ v'(t) = \dfrac{F(x,t)}{m} \end{cases}$$

Решаются подобные системы в точности так же, как и одиночные уравнения, но построение всех функций $y_i(x)$ должно производиться одновременно. Это значит, что на каждом шаге метода Эйлера должны быть известны значения в текущей точке x_i всех функций y_k . Тогда мы можем их подставить в правые части уравнений и получить все производные. Зная для каждой функции ее значение и производную, можно, как и прежде, найти соседнее значение.

Приложение 1

Зарезервированные слова в Pascal’е

A	**absolute**	**array**	**and**
B	**begin**		
C	**case**	**const**	
D	**div**	**downto**	**do**
E	**else**	**external**	**end**
F	**file**	**function**	**forward**
	for		
G	**goto**		
I	**if**	**implementation**	**in**
	inline	**interface**	**interrupt**
L	**label**		
M	**mod**		

N	**nil**	**not**	
O	**of**		
P	**packed**	**procedure**	**program**
R	**record**	**repeat**	
S	**set**	**shl**	**shr**
	string		
T	**then**	**type**	**to**
U	**unit**	**until**	**uses**
V	**var**		
W	**while**	**with**	
X	**xor**		

Приложение 2

Список некоторых стандартных математических функций

`Arctan(x)`	Арктангенс (результат в радианах)
`Cos(x)`	Косинус (аргумент в радианах)
`Exp(x)`	Экспонента (e^x)
`Frac(x)`	Дробная часть числа
`Ln(x)`	Натуральный логарифм
`Pi`	Значение числа π
`Round(x)`	Округление до ближайшего целого (тип результата Integer)
`Sin(x)`	Синус (аргумент в радианах)
`Sqr(x)`	Возведение в квадрат
`Sqrt(x)`	Извлечение квадратного корня
`Trunc(x)`	Целая часть числа (тип результата Integer)

Приложение 3

Диагностика некоторых ошибок, возникающих при компиляции программ.

1	Out of memory	Выход за границы памяти
2	Identifier expected	Не указано имя. Возможна попытка использовния зарезервированного слова
3	Unknown identifier	Неизвестное имя: не описано в разделе описаний программы.
4	Duplicate identifier	Повторное описание имени
5	Syntax error	Синтаксическая ошибка
6	Error in real constant	Ошибка в вещественной константе
7	Error in integer constant	Ошибка в целой константе
8	String constant exceeds line	Строковая константа превышает размер строки. Возможно, отсутствует кавычка в конце строковой константы.
10	Unexpected end of file	Неправильный конец файла. Скорее всего, количество слов **`begin`** не соответствует количеству слов **`end`**
11	Line too long	Строка слишком длинная. Максимальная длина строки = 126 символам
12	Type identifier expected	Отсутствует указание типа имени

14	Invalid filename	Неверное имя файла
15	File not found	Файл не найден
20	Variable identifier expected	Требуется правильно описать имя переменной
25	Invalid string length	Неверная длина строковой константы
26	Type mismatch	Несоответствие типов
30	Integer constant expected	Необходима целая константа
31	Constant expected	Требуется константа
32	Integer or real constant expected	Требуется целая или вещественная константа
34	Invalid function result type	Неправильный тип результата функции
36	BEGIN expected	Отсутствует **`begin`**
37	END expected	Отсутствует **`end`**
38	Integer expression expected	Выражение должно иметь тип `Integer`
40	Boolean expression expected	Выражение должно иметь тип `Boolean`
41	Operand types do not match operator	Типы операндов не соответствуют оператору
42	Error in expression	Ошибка в выражении
43	Illegal assignment	Неверное присвоение
54	OF expected	Отсутствует **`of`**
57	THEN expected	Отсутствует слово **`then`**
58	TO or DOWNTO expected	Отсутствует **`to`** или **`downto`**

62	Division by zero	Деление на ноль
76	Constant out of range	Константа нарушает границы допустимого диапазона
79	Integer or real expression expected	Выражение должно иметь целый или вещественный тип
85	“;” expected	Отсутствует точка с запятой
86	“:” expected	Отсутствует точка
87	“,” expected	Отсутствует запятая
88	“(“ expected	Отсутствует открывающая круглая скобка
89	“)” expected	Отсутствует закрывающая круглая скобка
90	“=” expected	Отсутствует знак равенства
91	“:=” expected	Отсутствует знак присвоения
92	“[“ or “(.” Expected	Отсутствует открывающие квадратная скобка или круглая с точкой
93	“]” or “.)” expected	Отсутствует закрывающие квадратная скобка или точка с круглой
94	“.” expected	Отсутствует точка
95	“..” expected	Отсутствует многоточие
96	Too many variables	Слишком много переменных
97	Invalid FOR control variable	Неправильная переменная параметра цикла **`for`**
98	Integer variable expected	Переменная должна иметь целый тип
102	String constant expected	Отсутствует строковая константа

103	Integer or real variable expected	Отсутствует целая или вещественная переменная
124	Statement part too large	Слишком большой раздел операторов
133	Cannot evaluate this expression	Невозможно вычислить данное выражение
135	Invalid format specifier	Неверная спецификация формата
140	Invalid floating-point operation	Недопустимая операция с вещественными числами, которая привела к переполнению (выход за границы представления вещественных чисел) или делению на ноль

Диагностика некоторых ошибок возникающих при запуске программы.

2	File not found	Файл не найден. Ошибка происходит при выполнении процедур `Reset` или `Append` в случае адресации к несуществующему файлу.
3	Path not found	Указан неверный путь к файлу
5	File access denied	Доступ к файлу закрыт
100	Disk read error	Ошибка при чтении с диска. Возникает при попытке осуществить считывание после достижения конца файла
101	Disk write error	Ошибка записи на диск
102	File not assigned	Файлу не присвоено имя. Наверное, отсутствует вызов процедуры `assign`
103	File not open	Файл не открыт
104	File not open for input	Файл не открыт на считывание
105	File not open for output	Файл не открыт на запись
106	Invalid numeric format	Неверный числовой формат. Ошибка возникает, если считанное числовое значение не соответствует правильному числовому формату
200	Division by zero	Деление на ноль

201	Range check error	Ошибка при проверке границ. Скорее всего, неверно указаны границы массива переменных
205	Floating point overflow	Переполнение при выполнении операции с плавающей точкой (вещественными числами)
207	Invalid floating point operation	Недопустимая операция с вещественными числами

Литература:

1. К.Эберт, Х.Эдерер. Компьютеры. Применение в химии. М., «Мир», 1988.
2. К.Джонсон. Численные методы в химии. М., «Мир», 1983.
3. Т.Шуп. Решение инженерных задач на ЭВМ. М., «Мир», 1982.
4. К.Йенсен, Н.Вирт. Паскаль. Руководство для пользователя. М., «Компьютер», 1993
5. Боон. Паскаль для всех. М., «Энергоатомиздат», 1988
6. Ж.Джонс, К.Харроу. Решение задач в системе Турбо Паскаль. М., «Финансы и статистика», 1991
7. С.А.Абрамов, Е.В.Зима. Начала программирования на языке Паскаль. М., «Наука», 1987
8. Р.Форсайт. Паскаль для всех. М., «Машиностроение», 1986
9. Ю.С.Бородич, А.Н.Вальвачев, А.И.Кузьмич. Паскаль для персональных компьютеров. Минск, 1991
10. Д.Мак-Кракен, У.Дорн. Численные методы и программирование на Фортране. М., «Мир», 1977
11. В.И.Крылов, Б.В.Бобков, П.И.Монастырский. Вычислительные методы. М., «Наука», 1976