МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Программирование на языке Pascal

Практическое пособие в двух частях

Часть 1

Гомель 2005

УДК 004.43(075.8)
ББК 32.973–018.1я73
П 784

Авторы: Е.А. Ружицкая, доцент, кандидат физико-математических наук; Г.Л. Карасёва, доцент, кандидат физико-математических наук; В.В. Орлов, доцент, кандидат технических наук; Т.М. Дёмова, ассистент

Рецензенты: О.И.Еськова, доцент, кандидат технических наук
М.С.Долинский, доцент, кандидат технических наук

Рекомендовано к изданию научно-методическим советом Учреждения образования «Гомельский государственный университет имени Франциска Скорины».

П 784 Программирование на языке Pascal: практическое пособие для студентов математических специальностей университета: В 2ч. Ч.1./ Е.А. Ружицкая, Г.Л. Карасёва, В.В. Орлов, Т.М. Дёмова; М-во образов. РБ, Гомельский государственный университет имени Франциска Скорины. – Гомель: ГГУ им. Ф.Скорины, 2005. – 108с.

Практическое пособие содержит описание языка программирования Turbo Pascal 7.0, приведены примеры решения широко распространенных в практике задач. Оно составлено в соответствии с учебными программами курсов «ЭВМ и программирование» для студентов 1 курса специальности 1–31 03 03–02 «Прикладная математика» (научно-педагогическая деятельность) и «Методы программирования и информатика» для студентов 1 курса специальности 1–31 03 01 «Математика». Пособие ориентировано на самостоятельное изучение и предназначено для студентов математических специальностей университета.

УДК 004.43(075.8)
ББК 32.973–018.1я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Система программирования Turbo Pascal является одной из самых популярных систем программирования. Именно с неё начинают изучать программирование студенты. Практическое пособие предназначено для студентов первого курса математического факультета, начинающих изучать язык программирования Pascal

В первой части практического пособия рассмотрены следующие темы: основные команды операционной системы MS DOS, работа в сервисной программе Norton Commander, элементы теории алгоритмов, среда программирования Turbo Pascal и основы языка программирования Pascal. Приведены методы решения трёх прикладных задач: приближенное вычисление функций, представленной разложением в ряд; методы решения алгебраических и трансцендентных уравнений; приближенные вычисления определенных интегралов. Пособие содержит теоретический материал, необходимый для выполнения лабораторных работ и заданий по вычислительной практике и примеры решения типовых задач по лабораторным работам. Оно является дополнением к лекционному материалу по курсам «ЭВМ и программирование», «Методы программирование и информатика» и ориентировано на самостоятельное изучение.

Пособие составлено в соответствии с учебными программами курсов «ЭВМ и программирование» для студентов 1 курса специальности 1–31 03 03–02 «Прикладная математика» (научно-педагогическая деятельность) и «Методы программирования и информатика» для студентов 1 курса специальности 1–31 03 01 «Математика», утвержденными научно-методическим Советом Учреждения образования «Гомельский государственный университет имени Франциска Скорины».

Учебное издание

Ружицкая Елена Адольфовна
Карасёва Галина Леонидовна
Орлов Владимир Васильевич
Дёмова Тамара Максимовна

Программирование на языке Pascal.
Практическое пособие в двух частях. Часть 1.

В авторской редакции

Подписано в печать 01.10.2005 г. (№131). Формат издания 60x84 1/16.
Бумага писчая №1. Печать на ризографе. Гарнитура Таймс.
Усл.п.л. 5,7. Уч-изд.л. 3,5. Тираж 20 экз.

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»,
246019 г.Гомель, ул. Советская, 104

Отпечатано в учреждении образования
«Гомельский государственный университет
имени Франциска Скорины»,
246019 г.Гомель, ул. Советская, 104

ЛИТЕРАТУРА

1. Бородич Ю.С. и др. Паскаль для персональных компьютеров: Справ. Пособие / Ю.С.Бородич, А.Н.Вальвачев, А.И.Кузьмич. – Мн.: Выш. шк.: БФ ГИТМП «НИКА», 1991. – 365 с.
2. Вальвачев А.Н., Крисевич В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС: Справ. пособие. – Мн.: Выш.шк., 1989. – 223 с.: ил.
3. Офицеров Д.В. и др. Программирование на персональных ЭВМ: Практикум: Учеб. Пособие / Д.В.Офицеров, А.Б. Долгий, В.А.Старых; Под общ. ред. Д.В.Офицерова. – Мн.: Выш.шк., 1993. – 256 с.
4. Немнюгин С.А. Turbo Pascal: практикум – СПб: Питер, 200. – 256 с.:ил.
5. Пантелеева З.Т. Графика вычислительных процессов: Учеб.пособие. – М.: Финансы и статистика, 1983. – 167 с., ил.
6. Фаронов В.В. Турбо Паскаль 7.0. Начальный курс. Учебное пособие. – М.: «Нолидж», 1997. – 616 с., ил.
7. Фигурнов В.Э. IBM PC для пользователя. Изд. 7-е, перераб. и доп. – М.: ИНФРА – М, 1997. – 640 с.: ил.

ОПЕРАЦИОННАЯ СИСТЕМА MS DOS

Работа с файловой системой

Программное обеспечение персонального компьютера можно разделить на 3 группы:

- операционные системы и сервисные программы;
- системы программирования;
- прикладные системы.

Операционная система – это комплекс программ по управлению работой аппаратной части ПЭВМ и организации взаимодействия пользователя и ПЭВМ.

Системы программирования – это языки программирования высокого уровня, машинно-ориентированные языки и инструментальные средства.

Прикладные программы – составляют категорию программных средств, обращенных к пользователям персональных компьютеров – людям, которые не обязаны уметь программировать и знать устройство машины.

Файл – это поименованная область внешней памяти, которой присвоено свое конкретное имя. Файлы используются для хранения текстов программ, данных и т.д. В соответствии с характером хранимой информации файлу присваивается тип (расширение имени файла). Имя и тип, разделенные точкой, используют совместно для идентификации файла. Тип можно не указывать, при отсутствии типа точка не обязательна.

Полное имя файла должно быть уникальным для каждого файла. Файлы, относящиеся к одной задаче, имеют одинаковые имена, но разные типы, например:

PROG.PAS – программный файл на языке Pascal;
PROG.OBJ – программный файл в объектных кодах;
PROG.COM – программный файл в машинных кодах.

Существуют стандартные типы файлов, которые позволяют определить характер файла визуально: COM – командный файл, BAT – пакетный файл, SYS – системный файл, TXT – текстовый файл, PAS – программа на языке Паскаль, BAK – старая версия файла и др.

При работе в MS DOS часто используются так называемые *шаблоны имени файла* – символы “?” и “*”. Шаблоны употребляются в командах для обозначения сразу нескольких файлов или сокращения записи имен файлов. Знак “*” в имени или типе файла заменяет любое число любых символов, а знак “?” – любой одиночный символ. Вопро-

сительный знак, помещенный последним в шаблоне имени или типа файла, воспринимается как любой одиночный символ или отсутствие такового. Примеры шаблонов:

P*.* – все файлы, начинающиеся с буквы “P”;
*.PAS – все файлы типа PAS;
. – все файлы на диске;
*. – все файлы, у которых отсутствует тип;
??n.EXE – все файлы типа EXE, у которых имя файла состоит из трех символов и последний символ имени буква n;
A??B.* – все файлы, имена которых состоят их 4-х символов, начиная с буквы A и последняя буква имени B;
D?.* – все файлы, имена которых начинаются буквой D и содержат один или 2 символа;
???.* – все файлы, имена которых содержат один, два или три любых символа.

NC при задании имен файлов позволяет использовать *регулярные выражения*, которые задаются набором символов в квадратных скобках:

[*символы*] – любой из указанных символов (строчные и прописные буквы различаются между собой);

[*символ1—символ 2*] – любой символ в указанном алфавитном диапазоне;

[^*символы*] – любой символ кроме указанных;

[^*символ1 –символ2*] – любой из символов, кроме символов в указанном диапазоне;

Примеры:

[A-CFX-Z]*.* – файлы с именем, начинающемся с букв A, B,C,F,X,Y,Z;

[^YZ]*.* – файлы с любым именем, кроме начинающемся на Y или Z и с любым расширением;

[^0-9]*.* – файлы с именем, не начинающемся с цифры и с любым расширением;

Каталог – это группа файлов на одном носителе, объединенных по какому-либо признаку. Каталог имеет свое имя и в свою очередь может быть зарегистрирован в другом каталоге, в этом случае говорят, что он является подчиненным каталогом (подкаталогом). Так образуется иерархическая файловая система. Цепочки вложенных друг в друга каталогов обозначаются их именами, разделенными знаком \.

\DEMO
\DEMO\VIKONT


```
Procedure Trap(a1,b1:Real; n:Integer;
                eps1:Real;f:Func; var tk:Real);
  Var
     I          :Integer;
     x,t0,s,h   :Real;
Begin
   t0:=f(a1)+f(b1)/2;
{t0 – предыдущее значение интеграла }
{tk – последующее значение интеграла}
  While True do
  begin
    h:=(b1-a1)/n;   {шаг}
    s:=f(a1)+f(b1)/2;
    For i:=1 to n-1 do
    begin
      x:=a1+i*h;
      s:=s+f(x);
    end;
    tk:=s*h;
    If abs(tk-t0)<eps then Exit;
    t0:=tk;
    n:=2*n;   {удваиваем количество разбиений}
  end;
End;
Begin
  Clrscr;
  Write('Введите пределы интегрирования a,b');
  ReadLn (a,b);
  nn:=6;  {начальное значение количества разбиений}
  Trap(a,b,nn,eps,F1,integral);
  WriteLn('Значение интеграла ',integral:8:4);
  Repeat Until KeyPressed;
End.
```

```
{t0–предыдущее значение интеграла}
{tk – последующее значение интеграла}
  While True do
  begin
    h:=(b1-a1)/n;   {шаг}
    s:=f(a1)+f(b1)/2;
    For i:=1 to n-1 do
    begin
      x:=a1+i*h;
      s:=s+f(x);
    end;
    tk:=s*h;
    If abs(tk-t0)<eps then Exit;
    t0:=tk;
    n:=2*n;   {удваиваем количество разбиений}
  end;
End;
Begin
  Clrscr;
  Write('Введите пределы интегрирования a,b');
  ReadLn (a,b);
  nn:=6; {начальное значение количества разбиений}
  Trap(a,b,nn,eps,integral);
  WriteLn('Значение интеграла ',integral:8:4);
  Repeat Until KeyPressed;
End.

{Вычисление интеграла по формуле трапеций       }
{с передачей имени функции в качестве параметра  }
Program Integ2;
  Uses Crt;
  Const eps=0.01;
  Type Func=Function (z:Real):Real;
  Var
   a,b,eps,integral :Real;
   nn               :Integer;
{подинтегральная функция}
Function F1(z:Real):Real; Far;
Begin
   F1:=1/(z-1);
end;
{процедура вычисления интеграла}
```


При сложной древовидной структуре файлов на диске для указания файла необходимо, кроме имени указать его местоположение – цепочку подчиненных каталогов. Такая цепочка называется *маршрутом* или *путем* по файловой системе. Маршрут отделяется от имени знаком \. Файл полностью задается следующими элементами: именем диска; местоположением (маршрутом); собственно именем файла.

D:\LEX\DOC\LEX.ARC

Команды MS DOS

MS DOS – это аббревиатура слов MicroSoft Disk Operating System, то есть дисковая операционная система фирмы Microsoft. Когда DOS готова к диалогу с пользователем, она выдает на экран *приглашение*, которое, как правило, содержит информацию о текущем дисководе и о текущем каталоге, например: A:\1>

Работа с файлами

1. Получение справки о командах DOS: sys /?
2. Вызов встроенного справочника: help
3. Создание небольших текстовых файлов: copy con *имя_файла*

После ввода этой команды нужно поочередно вводить строки файла. В конце каждой строки нажимать клавишу *Enter,* а после ввода последней – *F6.*

4. Удаление файлов: del *имя_файла*
5. Переименование файлов: ren *старое_имя новое_имя*
6. Копирование файлов:
   copy *имя_исходного_файла имя_полученного_файла*
7. Соединение фалов:
   copy *имя_файла1+имя_файла2 имя_результирующего_файла*
8. Перемещение файлов в другой каталог
   move *имя_файла имя_каталога*
   move *имя_файла* (*дисковод*:)(*путь*)*новое_имя_файла*
9. Поиск фалов на диске: filefind *имя_файла*

Работа с каталогами

1. Смена текущего дисковода: *дисковод:*
2. Изменение текущего каталога: cd *(дисковод:)путь*
3. Просмотр каталога:dir *(дисковод:)(путь\)(имя_файла)(параметры)*
   Параметры:

/P – поэкранный вывод оглавления,
/W – вывод данных в широком формате,
/ON – сортировка по имени файла,
/OE – сортировка по расширению,
/OS – сортировка по размеру файла,
/OD – сортировка по дате и времени создания.

4. Создание каталога: md (*дисковод*:) *путь*
5. Уничтожение пустого каталога: rd (*дисковод*:) *путь*
6. Удаление каталога со всем его содержимым
   deltree *имя_файла_или_каталога*
7. Переименование каталога:
   move *имя_каталога новое_имя_каталога*

Работа с экраном и принтером

1. Вывод файла на экран: type *имя_файла*
2. Очистка экрана: cls
3. Вывод файла на печать: copy *имя_файла* prn

$$\int_a^b f(x)dx \approx \frac{3}{8}h(y_0 + y_{3m} + 2(y_3 + y_6 + \ldots + y_{3m-3}) + 3(y_1 + y_2 + y_4 + y_5 \ldots + y_{3m-2} + y_{3m-1})),$$

где $y_i=f(x_i)$, $x_i=a+ih$, $h = \frac{b-a}{n} = \frac{b-a}{3m}$.

Интегралы считаются с помощью квадратурных формул с точностью ε. Для того, чтобы достичь требуемой точности вычисления ε, используется *способ двойного пересчета*: интеграл вычисляют по выбранной квадратурной формуле дважды, сначала с некоторым шагом h, затем с шагом $\frac{h}{2}$, т.е. удваивают число n (количество точек разбиения $[a,b]$).

Обозначим результаты разбиений через J_n и J_{2n} соответственно и сравним их. Если $|J_n-J_{2n}|<\varepsilon$, где ε – погрешность вычислений, то в качестве результата берут J_{2n}. Если $|J_n-J_{2n}|\geq\varepsilon$, то вычисления повторяют с шагом $\frac{h}{4}$ и т.д.

Пример: С помощью формулы трапеций вычислить интеграл $\int_2^3 \frac{dx}{x-1}$ с точностью ε=0.01.

```
Program Integ1;
{Вычисление интеграла по формуле трапеций        }
{без передачи имени функции в качестве параметра }
  Uses Crt;
  Const eps=0.01;
  Var
    a,b,eps,integral :Real;
    nn               :Integer;
{подинтегральная функция}
Function F(z:Real):Real;
Begin
   f:=1/(z-1);
End;
{процедура вычисления интеграла}
Procedure Trap(a1,b1:Real; n:Integer;
                eps1:Real; var tk:Real);
  Var
    i         :Integer;
    x,t0,s,h  :Real;
Begin
  t0:=f(a1)+f(b1)/2;
```

ПРИБЛИЖЕННЫЕ ВЫЧИСЛЕНИЯ ОПРЕДЕЛЕННЫХ ИНТЕГРАЛОВ

Для вычисления определенного интеграла $\int_a^b f(x)dx$ используют квадратурные формулы вида

$$\int_a^b f(x)dx \approx \sum_{k=0}^{n} A_k f(x_k) + R,$$

где x_k и A_k определяются квадратурной формулой, R – остаточный член или погрешность квадратурной формулы.

Отрезок интегрирования $[a,b]$ разбивается на n равных частей системой равноотстоящих точек $x_i=x_0+ih$, где $i=0,1,2,...,n$; $x_0=a$, $x_n=b$, $h=\frac{b-a}{n}$ — шаг разбиения. Затем вычисляем подынтегральную функцию в полученных узлах: $y_i=f(x_i)$.

Квадратурные формулы для равноотстоящих узлов:

1) формула левых прямоугольников:

$$\int_a^b f(x)dx \approx h(y_0 + y_1 + ... + y_{n-1}),$$ где $y_i=f(x_i)$, $x_i=a+ih$;

2) формула правых прямоугольников:

$$\int_a^b f(x)dx \approx h(y_1 + y_2 + ... + y_n),$$ где $y_i=f(x_i)$, $x_i=a+ih$;

3) формула центральных прямоугольников:

$$\int_a^b f(x)dx \approx h(y_0 + y_1 + ... + y_{n-1}),$$ где $y_i=f(x_i)$, $x_i = a + \left(i + \frac{1}{2}\right)h$;

4) формула трапеций:

$$\int_a^b f(x)dx \approx h(\frac{y_0 + y_n}{2} + y_1 + y_2 + ... + y_{n-1}),$$ где $y_i=f(x_i)$, $x_i=a+ih$;

5) формула Симпсона (формула парабол):

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

где $y_i=f(x_i)$, $x_i=a+ih$, $h = \frac{b-a}{n} = \frac{b-a}{2m}$;

6) формула Ньютона (правило $\frac{3}{8}$):

СЕРВИСНАЯ ПРОГРАММА NORTON COMMANDER (NC)

После запуска NC появляются два прямоугольных окна, ограниченные двойной рамкой (панели). Ниже располагается командная строка. В последней строчке экрана строка-подсказка о назначении функциональных клавиш NC.

В каждой панели NC может содержаться либо оглавление каталога на дискете, либо дерево каталогов, либо информация о диске. Имена файлов выводятся строчными буквами, а подкаталоги – заглавными. Если подвести курсор к какому-либо подкаталогу и нажать клавишу *Enter*, то NC "войдет" в этот каталог и выведет его оглавление. Для выхода из этого подкаталога надо выделить .. и нажать *Enter*. Если выделить файл с расширением .COM, .EXE, .BAT и нажать *Enter*, то начинается его выполнение.

Перемещение по панелям и каталогам

- Переход на другую панель – *Tab*.
- Переход в другой каталог – надо выделить этот каталог и нажать *Enter*.
- Переход в корневой каталог – *Ctrl* – \.
- Переход в подкаталог – *Ctrl–PageUp*.

Выбор группы файлов

- Включить файл в группу – *Insert*.
- Исключить файл из группы – повторное нажатие *Insert*.
- Включить в группу файлы по маске – нажать + на правой части клавиатуры и ввести маску (шаблон имен файлов).
- Исключить из группы файлы по маске – нажать – (знак минус) на правой части клавиатуры и ввести маску (шаблон).
- Сделать выбранные файлы невыбранными, а невыбранные выбранными – нажать * на правой части клавиатуры.

Выбранные файлы изображаются желтым цветом.

Действия с выбранной группой файлов

Выбранную группу файлов можно:

- *F5* – скопировать;
- *F6* – переименовать или переместить в другой каталог;
- *F8* – удалить;
- *Alt–F5* – поместить в архивный файл;
- *Alt–F6* – разархивировать (файлы должны быть архивами);

- *Ctrl–F10* – объединить в один файл.

Управление панелями NC

- *Ctrl–O* – убрать панели с экрана / вывести панели на экран;
- *Ctrl–P* – убрать одну из панелей (не текущую) с экрана / вывести панель на экран;
- *Ctrl–U* – поменять панели местами;
- *Ctrl–L* – сделать неактивную панель информационной (или отменить это);
- *Ctrl–Q* – сделать неактивную панель панелью быстрого просмотра (или отменить это);
- *Ctrl–Z* – вывести в неактивную панель паспорт каталога или отменить это;
- *Ctrl–F1* – убрать или вывести левую панель;
- *Ctrl–F2* – убрать или вывести правую панель;
- *Ctrl–F3* – сортировать файлы по имени;
- *Ctrl–F4* – сортировать файлы по расширению;
- *Ctrl–F5* – сортировать файлы по времени;
- *Ctrl–F6* – сортировать файлы по размеру;
- *Ctrl–F7* – не сортировать файлы на текущей панели;
- *Ctrl–F8* – синхронизация каталогов;
- *Alt–F1* – сменить дисковод на левой панели;
- *Alt–F2* – сменить дисковод на правой панели.

Назначение функциональных клавиш

1. Клавиши *F1–F10*:
   - *F1* – получение помощи;
   - *F2* – вызов меню команд пользователя;
   - *F3* – просмотр файла;
   - *F4* – редактирование файла;
   - *F5* – копирование файла или группы файлов;
   - *F6* – переименование или перемещение файла или каталога;
   - *F7* – создание каталога;
   - *F8* – удаление файла, группы файлов или каталога;
   - *F9* – меню Norton Commander;
   - *F10* – выход из Norton Commander.
2. Клавиши *Ctrl–F1 – Ctrl–F10*:
   - *Ctrl–F1* – убрать или вывести левую панель;
   - *Ctrl–F2* – убрать или вывести правую панель;
   - *Ctrl–F3* – сортировать файлы по имени;
   - *Ctrl–F4* – сортировать файлы по расширению;

Определим, какой из концов отрезка выбрать в качестве начального приближения по методу хорд, и по методу касательных. Для этого вычислим $f'(x)=5x^4-1$, $f''(x)=20x^3$, $f(1)f''(1)<0$, значит положим $x1=1$ – начальное приближение по методу хорд, $x2=1.1$ – начальное приближение для метода касательных.

```
Program Komb; {комбинированный метод}
  Const eps=0.0005;
  Var
   a,b,x,x1,x2,y1,y2,delta :Real;
   n                       :Integer;
Function F(z:Real):Real;
Begin
  F:=sqr(z)*sqr(z)*z-z-0.2;
End;
Function F1(z:Real):Real;
Begin
  F1:=5*sqr(z)*sqr(z)-1;
End;
Function F2(z:Real):Real;
Begin
  F2:=20*sqr(z)*z;
End;
Begin
  Write ('Введите отрезок [a,b]-->');
  ReadLn (a,b);
  If F(a)*F2(a)<0 then begin x1:=a; x2:=b;
                           end
                  else begin x1:=b; x2:=a;
                           end;
  n:=0;
  Repeat
    y1:=x1-F(x1)/(F(b)-F(x1))*(x2-x1);
    y2:=x2-F(x2)/F1(x2);
    delta:=abs(y2-y1);
    n:=n+1;
    x1:=y1; x2:=y2;
  Until delta<eps;
  x:=(x1+x2)/2.0;
  WriteLn('Корень уравнения x=',x:8:4);
  WriteLn('Проверка  f(',x:8:4,')=',F(x):8:5);
  WriteLn ('Количество приближений n=',n);
End.
```

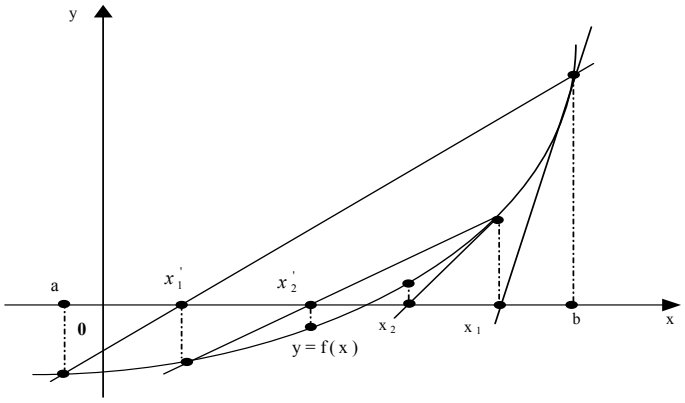
```
  ReadLn (x);
  n:=0;
  Repeat
    y:=x-F(x)/(F(b)-F(x))*(b-x);
    delta:=abs(y-x);
    n:=n+1;
    x:=y;
  Until delta<eps;
  WriteLn('Корень уравнения x=',x:8:4);
  WriteLn('Проверка  f(',x:8:4,')=',F(x):8:5);
  WriteLn('Количество приближений n=',n);
End.
```

Комбинированный метод

Пусть $f(a)f(b)<0$ и $f'(x)$ и $f''(x)$ сохраняют постоянные знаки на $[a,b]$. Соединяя метод хорд и касательных, получаем метод, на каждом этапе которого находится значение по недостатку и по избытку точного корня уравнения $f(x)=0$.

Процесс вычисления прекращаем, когда длина отрезка, на котором находится корень уравнения, будет меньше заданной точности ε. За значение корня лучше принять среднее арифметическое полученных последних значений.

Геометрическая интерпретация комбинированного метода:



Пример: Вычислить с точностью $\varepsilon=0.0005$ комбинированным методом корень уравнения $x^5-x-0.2=0$ лежащий на интервале (1,1.1).

- *Ctrl–F5* – сортировать файлы по времени;
- *Ctrl–F6* – сортировать файлы по размеру;
- *Ctrl–F7* – не сортировать файлы на текущей панели;
- *Ctrl–F8* – синхронизация каталогов;
- *Ctrl–F9* – печать выделенного файла;
- *Ctrl–F10* – делить или объединять файлы.

3. Клавиши *Alt–F1 – Alt–F10*:
   - *Alt–F1* – сменить дисковод на левой панели;
   - *Alt–F2* – сменить дисковод на правой панели;
   - *Alt–F3* – просмотр текстового файла;
   - *Alt–F4* – редактирование файла альтернативным редактором;
   - *Alt–F5* – копирование в архивный файл;
   - *Alt–F6* – извлечение файлов из архива;
   - *Alt–F7* – поиск файла на диске;
   - *Alt–F8* – просмотр и выполнение выполненных ранее команд;
   - *Alt–F9* – переключение с 25 на 43 или 50 строк на экране;
   - *Alt–F10* – быстрый переход в другой каталог.
4. Клавиши *Shift–F1 – Shift–F10*:
   - *Shift–F1* – уборка диска (удаление резервных, временных файлов, старых версий файлов);
   - *Shift–F2* – сетевые утилиты;
   - *Shift–F3* – просмотр файла (имя запрашивается);
   - *Shift–F4* – редактирование файла (имя запрашивается);
   - *Shift–F5* – копирование файла (имя запрашиваются);
   - *Shift–F6* – переименование файла (имя запрашиваются);
   - *Shift–F7* – создание каталога;
   - *Shift–F8* – удаление (имя файла запрашивается) ;
   - *Shift–F9* – сохранение конфигурации NC;
   - *Shift–F10* – вызов меню (последнего использованного пункта).

Другие комбинации клавиш

- *Ctrl–E* – вывод в командную строку последней выполненной команды;
- *Ctrl–Enter* – ввод в командную строку имени файла, на котором установлен курсор.

ЭЛЕМЕНТЫ ТЕОРИИ АЛГОРИТМОВ

Понятие алгоритма, относящееся к фундаментальным основам информатики, возникло задолго до появления компьютеров и является одним из основных понятий математики.

Слово «АЛГОРИТМ» произошло от имени выдающегося средневекового ученого МУХАМЕДА ибн МУСА ал-ХОРЕЗМИ(IX век н.э.) (в переводе с арабского «Мухамед сын Мусы из Хорезма»), сокращенно АЛ-ХОРЕЗМИ, уроженца Хивы. В одном из своих трудов Ал-Хорезми описал десятичную систему счисления и впервые сформулировал правила выполнения арифметических действий над целыми числами и простыми дробями. В латинском переводе арифметического труда Ал-Хорезми правила начинались словами DIXIT ALGORIZMI (Алгоризми сказал), в других латинских переводах автор именовался ALGORITHMUS (Алгоритмус).

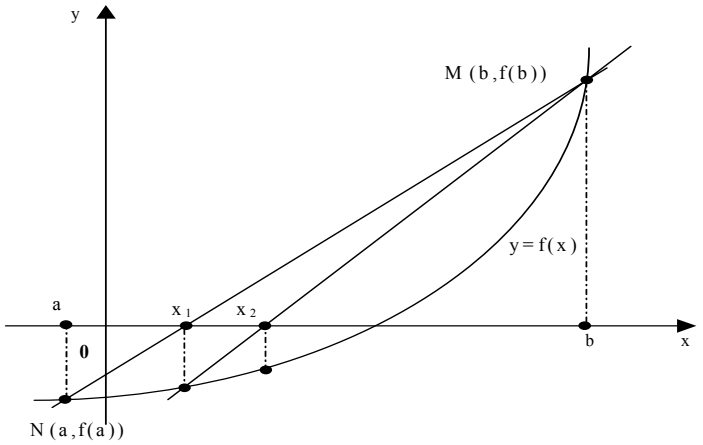
Для пояснения понятия «алгоритм» важное значение имеет определение понятия «исполнитель алгоритма». Алгоритм формулируется в расчете на конкретного исполнителя; алгоритм является руководством к действию для исполнителя, поэтому значение слова «алгоритм» близко по смыслу к значению слов «указание» или «предписание». Можно сказать, что *АЛГОРИТМ* – понятное и точное предписание (указание) исполнителю совершить определенную последовательность действий для достижения указанной цели или решения поставленной задачи или *АЛГОРИТМ* – точное предписание, которое задает вычислительный процесс, начинающийся с произвольного исходного данного из некоторой совокупности возможных для этого процесса данных и направленный на получение полностью определяемого этими исходными данными результата. Сказанное не является определением в математическом смысле, а лишь отражает интуитивное понимание алгоритма (в математике нет понятия «предписание», неясно, какова должна быть точность, что такое «понятность» и т.д.).

Основные свойства алгоритма

1. Алгоритм имеет некоторое число входных величин – аргументов, задаваемых до начала исполнения. Цель выполнения алгоритма – получение результата (результатов), имеющего вполне определенное отношение к исходным данным. Можно сказать, что алгоритм указывает последовательность действий по переработке исходных данных в результаты. Для алгоритма можно выбирать различные наборы входных данных из множества допустимых для этого процесса данных, т.е. можно применять алгоритм для

Процесс нахождения последовательных приближений продолжается до тех пор, пока не выполнится условие $|x_n–x_{n-1}|\le\varepsilon$. Для сходимости метода в качестве начального приближения нужно выбирать тот из концов отрезка, для которого выполняется условие $f(x)f''(x)<0$. Неподвижен тот конец отрезка, для которого $f(x)f''(x)>0$.

Геометрическая интерпретация метода хорд



Пример: Методом хорд найти корень уравнения $x^3–0.2x^2–0.2x–1.2=0$ расположенный на отрезке [1,2]с точностью $\varepsilon=0.001$.

Определим неподвижный конец $f'(x)=3x^2–0.4x–0.2$, $f''(x)=6x–0.4$, $f(2)f''(2)>0$ значит неподвижной будет точка $x=2$, в качестве начального приближения возьмем $x=1$.

Замечание: На каждом этапе необходимо помнить лишь два соседних приближения, поэтому приближение x_n обозначим через x, а приближение x_{n+1} через y.

```
Program Xord; {метод хорд}
  Const eps=0.001;
  Var
   x,y,delta :Real;
   n         :Integer;
Function F(z:Real):Real;
Begin
  F:=z*z*z+0.2*z*z-0.2*z-1.2;
End;
Begin
  Write ('Введите начальное приближение x=');
```

```
Begin
  F1:=(cos(z)-1.0);
End;
Begin
  Clrscr;
  Write ('Введите начальное приближение x=');
  ReadLn (x);
  eps:=0.1*sqrt(eps);
  n:=0;
  Repeat
    y:=x-F(x)/F1(x);
    delta:=abs(y-x);
    n:=n+1;
    x:=y;
  Until delta<eps;
  WriteLn('Корень уравнения x=',x:8:4);
  WriteLn('Проверка  f(',x:8:4,')=',F(x):8:5);
  WriteLn('Количество приближений n=',n);
  Repeat Until KeyPressed;;
End.
```

Метод хорд

Пусть корень уравнения $f(x)=0$ лежит на отрезке $[a,b]$. Для определенности положим $f(a)<0$, $f(b)>0$.

Разделим отрезок $[a,b]$ в отношении $-f(a):f(b)$. Это дает приближенное значение корня $x_1 = a - \dfrac{f(a)}{f(b)-f(a)}(b-a)$.

Далее применим этот прием к тому из отрезков $[a,x_1]$ и $[x_1,b]$ на концах которого функция $f(x)$ имеет противоположные знаки, получим второе приближение корня x_2 и т.д.

Геометрически способ пропорциональных частей эквивалентен замене кривой $y=f(x)$ хордой, проходящей через точки $N\{a,f(a)\}$ и $M\{b,f(b)\}$.

Если конец b отрезка $[a,b]$ неподвижен, то $x_0=a$ и

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(b)-f(x_{n-1})}(b-x_{n-1});$$

если конец a отрезка $[a,b]$ неподвижен, то $x_0=b$ и

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(x_{n-1})-f(a)}(x_{n-1}-a).$$

решения целого класса задач одного типа, различающихся исходными данными. Это свойство алгоритма обычно называют *МАССОВОСТЬЮ*. Однако существуют алгоритмы, применимые только к единственному набору данных. Можно сказать, что для каждого алгоритма существует свой класс объектов, допустимых в качестве исходных данных. Тогда свойство *МАССОВОСТИ* означает применимость алгоритма ко всем объектам этого класса.

2. Чтобы алгоритм можно было выполнить, он должен быть понятен исполнителю. *ПОНЯТНОСТЬ АЛГОРИТМА* означает знание исполнителя о том, что надо делать для исполнения этого алгоритма.
3. Алгоритм представляется в виде конечной последовательности шагов (алгоритм имеет *ДИСКРЕТНУЮ* структуру) и его исполнение расчленяется на выполнение отдельных шагов (выполнение очередного шага начинается после завершения предыдущего).
4. Выполнение алгоритма заканчивается после выполнения конечного числа шагов. При выполнении алгоритма некоторые его шаги могут повторяться многократно. В математике существуют вычислительные процедуры, имеющие алгоритмический характер, но не обладающие свойством *КОНЕЧНОСТИ*.
5. Каждый шаг алгоритма должен быть четко и недвусмысленно определен и не должен допускать произвольной трактовки исполнителем. Следовательно, алгоритм рассчитан на *ЧИСТО МЕХАНИЧЕСКОЕ ИСПОЛНЕНИЕ*. Именно *ОПРЕДЕЛЕННОСТЬ* алгоритма дает возможность поручить его исполнение *АВТОМАТУ*.
6. Каждый шаг алгоритма должен быть выполнен точно и за конечное время. В этом смысле говорят, что алгоритм должен быть *ЭФФЕКТИВНЫМ*, т.е. действия исполнителя на каждом шаге исполнения алгоритма должны быть достаточно простыми, чтобы их можно было выполнить точно и за конечное время. Обычно отдельные указания исполнителю, содержащиеся в каждом шаге алгоритма, называют *КОМАНДАМИ*. Таким образом, эффективность алгоритма связана с возможностью выполнения каждой команды за конечное время. Совокупность команд, которые могут быть выполнены конкретным исполнителем, называется *СИСТЕМОЙ КОМАНД ИСПОЛНИТЕЛЯ*. Следовательно, алгоритм должен быть сформулирован так, чтобы содержать только те команды которые входят в систему команд исполнителя. Кроме того, эффективность означает, что алгоритм может быть выполнен не просто за конечное, а за разумно конечное время.

Приведенные выше комментарии поясняют интуитивное понятие алгоритма, но само это понятие не становится от этого более четким и строгим. Тем не менее в математике долгое время использовали это понятие. Лишь с выявлением алгоритмически неразрешимых задач, т.е. задач, для решения которых невозможно построить алгоритм, появилась настоятельная потребность в построении формального определения алгоритма, соответствующего известному интуитивному понятию. Интуитивное понятие алгоритма в силу своей неопределенности не может быть объектом математического изучения, поэтому для доказательства существования или несуществования алгоритма решения задачи было необходимо строгое формальное определение алгоритма.

Построение такого формального определения было начато с формализации объектов (операндов) алгоритма, так как в интуитивном понятии алгоритма его объекты могут иметь произвольную природу. Ими могут быть, например, числа, показания датчиков, фиксирующих параметры производственного процесса, шахматные фигуры и позиции и т.п. Однако предполагая, что алгоритм имеет дело не с самими реальными объектами, а с их изображениями, можно считать , что *ОПЕРАНДЫ АЛГОРИТМА* есть слова в произвольном алфавите. Тогда получается, что алгоритм преобразует слова в произвольном алфавите в слова того же алфавита. Дальнейшая формализация понятия алгоритма связана с формализацией действий над операндами и порядка этих действий. Одна из таких формализаций была предложена в 1936 г. английским математиком А.Тьюрингом, который формально описал конструкцию некоторой абстрактной машины (*МАШИНЫ ТЬЮРИНГА*) как исполнителя алгоритма и высказал основной тезис тезис о том, что всякий алгоритм может быть реализован соответствующей машиной Тьюринга. Примерно в это же время американским математиком Э.Постом была предложена другая алгоритмическая схема – *МАШИНА ПОСТА*, а в 1954 г. советским математиком А.А.Марковым была разработана теория классов алгоритмов, названных им *НОРМАЛЬНЫМИ АЛГОРИФМАМИ*, и высказан основной тезис о том, что всякий алгоритм нормализуем.

Эти алгоритмические схемы эквиваленты в том смысле, что алгоритмы, описываемые в одной из схем, могут быть также описаны и в другой. В последнее время эти теории алгоритмов объединяют под названием *ЛОГИЧЕСКИЕ*.

Логические теории алгоритмов вполне пригодны для решения теоретических вопросов о существовании или несуществовании алгоритма, но они никак не помогают в случаях, когда требуется получить хороший алгоритм, годный для практических применений. Дело в том,

Следующие приближения находим по формулам: $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}, \ldots, x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$.

Процесс вычислений прекращается при выполнении условия $|x_n - x_{n-1}| < \sqrt{\frac{2m_1\varepsilon}{M_2}}$, где $m_1 = \min_{[a,b]}|f'(x)|, M_2 = \max_{[a,b]}|f''(x)|$. Если $\frac{2m_1}{M_2} \geq 10^{-2}$, то можно пользоваться более простым соотношением: $|x_n - x_{n-1}| \leq \varepsilon_1 = 0.1\sqrt{\varepsilon}$.

Для сходимости метода начальное приближение x_0 выбирают так, чтобы выполнялось условие $f(x_0)f''(x_0)>0$. В противном случае сходимость метода не гарантируется.

Пример: Методом Ньютона найти корень уравнения $\sin(x)-x+0.15=0$ на отрезке [0.5,1] с точностью $\varepsilon=0.0001$.

Найдем $f'(x)=\cos(x)-1$, $f''(x)=-\sin(x)$. Расчетная формула имеет вид:

$$x_n = x_{n-1} - \frac{\sin(x) - x + 0.15}{\cos(x) - 1}.$$

В качестве начального приближения возьмем $x_0=1$, т.к. $f(1)f''(1)>0$.

Вычислим m_1 и M_2: $m_1=|\cos(0.5)-1|=0.12$, $M_2=|-\sin(1)|=0.84$, значит $\frac{2m_1}{M_2} \geq 10^{-2}$, и для проверки точности вычислений воспользуемся соотношением: $|x_n - x_{n-1}| \leq \varepsilon_1 = 0.1\sqrt{\varepsilon}$.

Замечание: На каждом этапе необходимо помнить лишь два соседних приближения, поэтому приближение x_n обозначим через x, а приближение x_{n+1} через y.

```
Program Kasat; {метод касательных }
  Uses Crt;
  Const
    eps=0.0001;
  Var
   x,y,delta :Real;
   n         :Integer;
Function F(z:Real):Real;
Begin
  F:=sin(z)-z+0.15;
End;
Function F1(z:Real):Real;
```

Метод Ньютона (касательных)

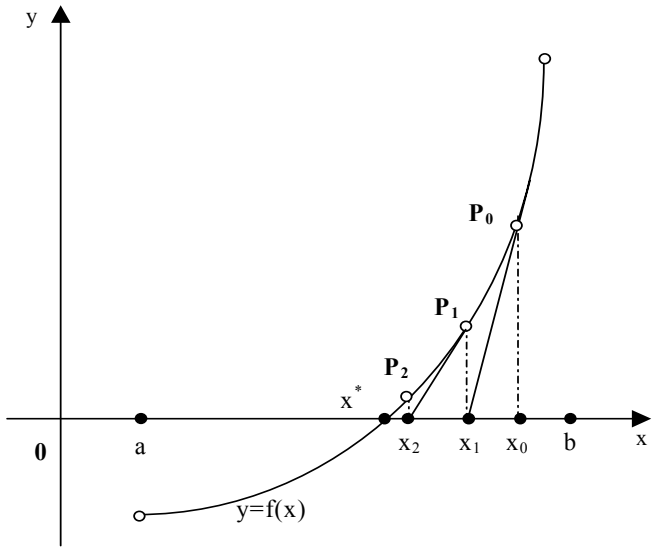
Пусть уравнение $f(x)=0$ имеет один корень на отрезке $[a,b]$, причем $f'(x)$ и $f''(x)$ определены, непрерывны и сохраняют постоянные знаки на отрезке $[a,b]$.

Рассмотрим геометрическую интерпретацию этого метода. Возьмем некоторую точку x_0 отрезка $[a,b]$ и проведем в точке $P_0\{x_0,f(x_0)\}$ графика функции касательную к кривой $y=f(x)$ до пересечения с осью Ox. Абсциссу x_1 точки пересечения можно взять в качестве приближенного значения корня. Проведя касательную через новую точку $P_1\{x_1,f(x_1)\}$ и находя точку ее пересечения с осью Ox, получим второе приближение корня x_2. Аналогично определяются последующие приближения.

Выведем формулу для последовательных приближений к корню. Уравнение касательной, проходящей через точку P_0 имеет вид:

$$y = f(x_0) + f'(x_0)(x - x_0).$$

Полагая $y=0$, находим абсциссу x_1 точки пересечения касательной с осью Ox: $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$.



что с точки зрения логических теорий алгоритмы, предназначенные для практических применений, являются алгоритмами в интуитивном смысле. Поэтому при решении проблем, возникающих в связи с созданием и анализом таких алгоритмов, нередко приходится руководствоваться лишь интуицией, а не строгой математической теорией. Таким образом, практика поставила задачу создания содержательной теории, предметом которой были бы алгоритмы как таковые и которая позволяла бы оценивать их качество, давала бы практически пригодные методы их построения, эквивалентного преобразования, доказательства правильности и т.п.

Содержательная (аналитическая) теория алгоритмов стала возможной лишь благодаря фундаментальным работам математиков в области логических теорий алгоритмов. Развитие такой теории связано с дальнейшим и расширением формального понятия алгоритма, которое слишком сужено в рамках логических теорий. Формальный характер понятия позволит применять к нему математические методы исследования, а его широта должна обеспечить возможность охвата всех типов алгоритмов, с которыми приходиться иметь дело на практике.

Средства записи алгоритмов

Средства, используемые для записи алгоритмов, в значительной степени определяются тем, для какого исполнителя предназначается алгоритм. Если исполнителем алгоритма является человек, то запись алгоритма может быть не полностью формализована, на первое место здесь выдвигаются понятность и наглядность, поэтому для записи подобных алгоритмов может использоваться естественный или графический язык. Однако в случае исполнителя-автомата естественные языки неприменимы ввиду их неточности, неоднозначности и противоречивости, в таких случаях необходимо применять специально разработанные формальные языки.

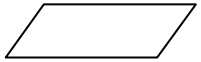
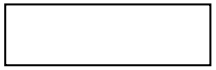
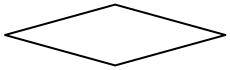
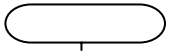
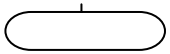
1. СЛОВЕСНАЯ ЗАПИСЬ АЛГОРИТМОВ

   Словесная форма записей алгоритмов на естественных языках применяется при ориентации на исполнителя-человека. Команды алгоритма нумеруют для возможности ссылки на них. Форма записи команд не формализуется. В командах помимо слов могут использоваться специальные символы и формулы.

2. ГРАФИЧЕСКИЕ СХЕМЫ АЛГОРИТМОВ

   Схемы представляют алгоритм в наглядной графической форме. Команды алгоритма помещаются внутрь блоков, представляющих собой стандартные геометрические фигуры и связанных между собой ломаными линиями с указанным направлени-

ем движения. Существуют государственные стандарты изображения геометрических фигур-блоков и правил записи графических схем алгоритмом. На практике наиболее часто используются блоки:

Наименование	Обозначение	Функции
1. Ввод-вывод		Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод)
2.Процесс		Выполнение операции или группы операций, в результате которых изменяется значение, форма представления или расположение данных
3. Решение		Выбор направления выполнения алгоритма или программы в зависимости от некоторых переменных условий
4. Пуск		Начало выполнения программы
5. Останов		Конец выполнения программы

Для записи внутри блока команды используется естественный язык с элементами математической символики. Графические схемы алгоритмов обладают большей наглядностью по сравнению со словесной формой записи, однако это преимущество исчезает при записи сколько-нибудь большого алгоритма.

3. ПСЕВДОКОД

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов. Он занимает промежуточное положение между естественными и формальными языками. С одной стороны он близок к естественному языку, с другой – в псевдокоде используются формальные конструкции и математическая символика, приближающие его к формальным языкам и математической формализации. В псевдокоде не приняты строгие синтаксические правила записи команд, что дает возможность использовать более

Метод половинного деления

Для нахождения корня уравнения $f(x)=0$, принадлежащего отрезку $[a,b]$, делим отрезок пополам, т.е. выбираем начальное приближение равным $x_0 = \frac{a+b}{2}$. Если $f(x_0)=0$, то x_0 является корнем уравнения. Если $f(x_0)\neq 0$, то выбираем тот из отрезков $[a,x_0]$ или $[x_0,b]$, на концах которого функция $f(x)$ имеет противоположные знаки. Полученный отрезок снова делим пополам и проводим те же рассуждения.

Процесс деления отрезков пополам продолжаем до тех пор, пока длина отрезка, на концах которого функция имеет противоположные знаки, не будет меньше заданной точности ε.

Пример: Методом половинного деления найти корень уравнения $x-\sqrt{9+x}+x^2-4=0$ на отрезке [2,3] с заданной точностью ε=0.0001.

```
Program Poldel; {метод половинного деления}
  Uses
    Crt;
  Const
    eps=0.0001;
  Var
    x,a,b:Real;
Function F(z:Real):Real;
Begin
  F:=z-sqrt(9+z)+z*z-4;
End;
Begin
  Clrscr;
  Write ('Ведите отрезок a,b -->');
  ReadLn (a,b);
  Repeat
  x:=0.5*(b+a);
  If f(x)=0 then Break;
  If f(a)*f(x)<0 then b:=x
                 else a:=x;
  Until b-a<eps;
  WriteLn ('Корень уравнения x=',x:8:4);
  WriteLn ('Проверка f(',x:8:4,')=',f(x):8:5);
  Repeat Until KeyPressed;
End.
```

Пример: Методом итераций найти корень уравнения $f(x)=\arcsin(2x+1)-x^2=0$ расположенный на отрезке [-0.5,0] с точностью $\varepsilon=10^{-4}$.

Уравнение преобразуем к виду $y=\varphi(x)$ следующим образом: $\arcsin(2x+1)=x^2$, $\sin(\arcsin(2x+1))=\sin(x^2)$, $2x+1=\sin(x^2)$, $x=0.5(\sin(x^2)-1)$.

Значит $\varphi(x)= 0.5(\sin(x^2)-1)$, $\varphi'(x)=x\cos(x^2)$, $|\varphi'(x)|=|x\cos(x^2)|\le 0.5$ для $x\in[-0.5,0]$. Метод итераций сходится.

Замечание: на каждом этапе необходимо помнить лишь два соседних приближения, поэтому приближение x_n обозначим через x, а приближение x_{n+1} через y.

```
Program Iter; {метод итераций}
  Uses Crt;
  Const eps=0.0001;
  Var
   fx,x,y,delta :Real;
   n            :Integer;
Function Fi(z:Real):Real;
Begin
  Fi:=0.5*(sin(z*z)-1)
End;
Function F(z:Real):Real;
Begin
  F:=2*z+1-sin(z*z);
End;
Begin
  Clrscr;
  Write ('Введите начальное приближение x=');
  ReadLn (x);
  n:=0;
  Repeat
    y:=Fi(x);
    delta:=abs(y-x);
    n:=n+1;
    x:=y;
  Until delta<eps;
  WriteLn('Корень уравнения x=',x:8:4);
  WriteLn('Проверка  f(',x:8:4,')=',f(x):8:5);
  WriteLn('Количество итераций n=',n);
  Repeat Until KeyPressed;
End.
```


широкий набор команд, рассчитанный на абстрактного исполнителя на стадии проектирования. Однако здесь используются стандартные конструкции, присущие формальным языкам, что облегчает переход от записи алгоритма на псевдокоде к записи на формальном языке. В псевдокоде фиксируются служебные слова, смысл которых определен раз и навсегда. Они выделяются жирным шрифтом (печатный вариант) или подчеркиванием (рукописный вариант). Формального определения псевдокода не существует, поэтому возможны его различные варианты, отличающиеся набором служебных слов и основных (базовых) конструкций.

алгоритм *ЕВКЛИД* ;
начало
  пока первое число не равно второму числу
  начало
    если первое число больше второго числа
    то заменить первое число на разность первого и второго чисел
    иначе заменить второе число на разность второго и первого чисел
    все
    взять первое число в качестве ответа;
  конец
конец

Пример записи на псевдокоде алгоритма Евклида нахождения наибольшего общего делителя двух натуральных чисел.

4. ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Для записи алгоритмов, ориентированных на исполнителя-автомат, которым является компьютер, были разработаны формальные языки, получившие наименование *ЯЗЫКИ ПРОГРАММИРОВАНИЯ*.

Структуры алгоритмов

1. ПРОСТЫЕ КОМАНДЫ

Элементарной структурной единицей любого алгоритма является *ПРОСТАЯ КОМАНДА*, обозначающая один элементарный шаг переработки или передачи информации. При исполнении алгоритма переработка информации состоит в изменении значений величин, которыми оперирует алгоритм. Все величины подразделяют на постоянные (кон-

станты) и переменные. Значение константы не может быть изменено в процессе исполнения алгоритма в отличие от переменных величин, значения которых могут быть изменены. Для обозначения величин используются *ИМЕНА,* или *ИДЕНТИФИКАТОРЫ*. Как правило, в качестве идентификаторов используют последовательности букв, цифр и других допустимых символов.

Значение переменной может быть изменено, например, с помощью команды присваивания

<идентификатор>:=<выражение>

Здесь и далее в угловых скобках записываются основные понятия, которые в реальных командах заменяются на конкретные имена и конкретные выражения. Знак присваивания (:=) обозначает указание исполнителю вычислить значение выражения в правой части команды и присвоить это значение переменной, стоящей слева от знака присваивания.

Переменной величине может быть присвоено новое значение и при выполнении исполнителем алгоритма команды ввода, которая предполагает получение исполнителем значения от внешнего источника информации. Например, команда

ввод (x,y,z)

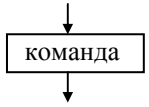
означает получение исполнителем от внешнего источника трех значений, которые должны быть присвоены переменным x, y и z .

Аналогичная команда

вывод (m,n)

означает передачу исполнителем значений переменных m и n внешнему приемнику информации.

Простая команда при графическом способе записи алгоритмов представляется в виде функционального блока, имеющего один вход и один выход, например



2. СОСТАВНЫЕ КОМАНДЫ

2.1. КОМАНДА СЛЕДОВАНИЯ

Эта команда образуется из последовательности команд, следующих одна за другой. При записи на псевдокоде команды отделяются друг от друга точкой с запятой. При исполнении алгоритма команды выполняются одна за другой в естественном порядке их записи. Для обозначения начала и конца команды следования используются служебные слова начало и конец.

чек пересечения этих графиков тоже являются приближенными корнями уравнения $f(x)=0$.

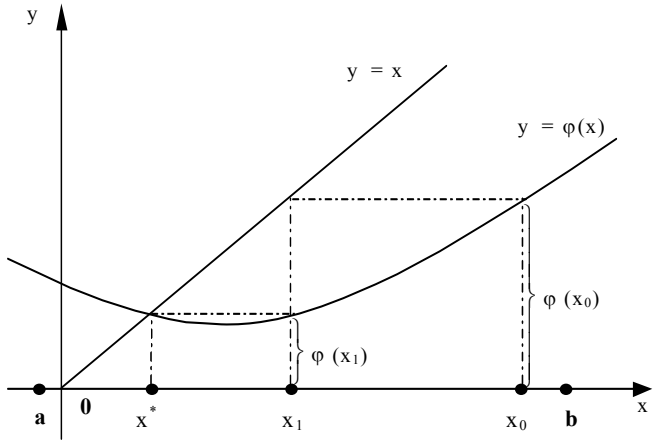
Метод итераций

Уравнение $f(x)=0$ представляем в виде $x=\varphi(x)$. Выбираем на отрезке $[a,b]$ произвольную точку x_0 в качестве начального приближения и строим последовательность: $x_1=\varphi(x_0)$, $x_2=\varphi(x_1)$, ..., $x_n=\varphi(x_{n-1})$. Процесс последовательного вычисления чисел x_n (n=1,2,3,...) называется методом итераций.

Если на отрезке $[a,b]$ выполнено условие $|\varphi'(x)|\le q<1$, то процесс итераций сходится, т.е. увеличивая n, можно получить приближение, сколь угодно мало отличающееся от истинного значения корня уравнения.

Процесс итераций продолжается до тех пор, пока не будет выполнено условие $|x_n-x_{n-1}|\le\frac{1-q}{q}\varepsilon$, где ε — заданная точность вычислений. Если $q\le 0.5$, то для прекращения процесса итераций можно пользоваться более простым соотношением $|x_n-x_{n-1}|\le\varepsilon$.

Геометрическая интерпретация метода итераций:



МЕТОДЫ РЕШЕНИЯ АЛГЕБРАИЧЕСКИХ И ТРАНСЦЕНДЕНТНЫХ УРАВНЕНИЙ

На практике часто приходится решать уравнения вида $f(x)=0$, где $f(x)$ определена и непрерывна на некотором конечном или бесконечном интервале $(a;b)$. Если функция $f(x)$ представляет собой многочлен, то уравнение называется *алгебраическим*, если же в функцию $f(x)$ входят элементарные (тригонометрические, логарифмические, показательные и др.) функции, то такие уравнения называются *трансцендентными.*

Всякое значение x^*, обращающее функцию в 0, т.е. такое, что $f(x^*)=0$, называется корнем уравнения.

Найти корни уравнения точно удается лишь в частных случаях. Кроме того, часто уравнение содержит коэффициенты, известные лишь приблизительно, и, следовательно, сама задача о точном определении корней уравнения теряет смысл. Поэтому разработаны методы численного решения уравнений $f(x)=0$, которые позволяют отыскать приближенные значения корней этого уравнения. При этом приходится решать 2 задачи:

- Отделение корней, т.е. отыскание достаточно малых областей, в каждой из которых находится только один корень уравнения.
- Вычисление корней с заданной точностью.

При выделении областей, содержащих действительные корни уравнения $f(x)$, можно воспользоваться тем, что если на концах некоторого отрезка непрерывная функция $f(x)$ принимает значения разных знаков, то на этом отрезке уравнение $f(x)=0$ имеет хотя бы один корень. Отделение корней можно выполнять графически и аналитически.

Аналитический способ отделения корней:

1) находим производную $f'(x)$;

2) составляем таблицу знаков функции $f(x)$, полагая x равным: критическим точкам функции или близким к ним; граничным значениям, исходя из области допустимых значений неизвестного;

3) определяем интервалы, на концах которых функция принимает значения противоположных знаков; внутри этих интервалов содержится по одному корню.

Графически корни можно отделять двумя способами:

1) строится график функции $y=f(x)$, абсциссы точек пересечения которого с осью Ox являются приближенными корнями уравнения $f(x)=0$;

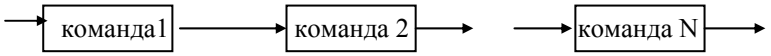
2) уравнение $y=f(x)$ записывается в эквивалентном виде $\varphi(x)=\psi(x)$, так, чтобы графики функций $\varphi(x)$, $\psi(x)$ строились проще; абсциссы то-

Общий вид команды следования

начало <команда 1> ; <команда 2> ; … ; <команда N> **конец** ,

где <команда 1> ;<команда 2> ; … ; <команда N> - простые или составные команды. На практике команды, образующие составную команду, записываются в столбец одна под другой.

Служебные слова начало и конец выполняют роль скобок. Их наличие позволяет рассматривать команду следования как одну команду в тех случаях, когда синтаксис языка описания алгоритмов не допускает использования составных команд.

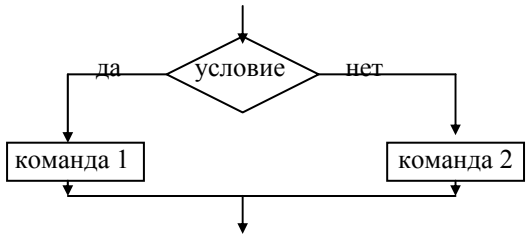


2.2. КОМАНДА ВЕТВЛЕНИЯ (РАЗВИЛКА)

Простейшая форма ветвления – это *АЛЬТЕРНАТИВА*, где есть два возможных пути и выбор зависит от того, верно или неверно некоторое *УСЛОВИЕ*

если < условие >
то < команда 1 >
иначе <команда 2 >
все

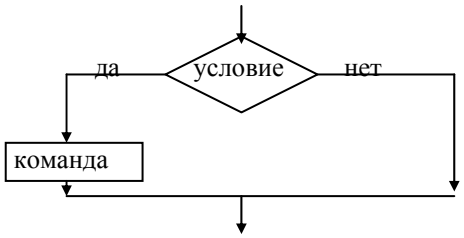
или при использовании графических схем



Это так называемая *ПОЛНАЯ УСЛОВНАЯ КОНСТРУКЦИЯ*. Может использоваться и команда ветвления в сокращенной форме – *НЕПОЛНАЯ УСЛОВНАЯ КОНСТРУКЦИЯ (КОРРЕКЦИЯ)*, когда в случае невыполнения указанного в команде условия никакое действие не выполняется :

если < условие >
то < команда >
все

или на языке графических схем:



Часто приходится выбирать не из двух, а из нескольких возможностей. Такую ситуацию называют *МНОГОЗНАЧНЫМ ВЕТВЛЕНИЕМ (ПЕРЕКЛЮЧАТЕЛЕМ)* и записывают:

выбрать
< условие 1 > : < команда 1 > ;
< условие 2 > : < команда 2 > ;
…
< условие N > : < команда N >
иначе
< команда 0 > ;

Этот порядок предусматривает, что выполняется команда i, если соответствующее условие i *ВЕРНО*; или, если ни одно из условий i (i = 1,2, … ,n) неверно, выполняется команда 0 (при наличии ветви иначе). При использовании *МНОГОЗНАЧНОГО ВЕТВЛЕНИЯ* следует учитывать тот факт, что либо никакие два из условий не являются верными одновременно, либо несколько условий являются верными одновременно.

2.3. КОМАНДА ПОВТОРЕНИЯ (ЦИКЛА)

Многие алгоритмы содержат серии команд, которые должны реализовываться исполнителем многократно. Если такие алгоритмы записывать в виде составной команды следования, то каждую повторяемую команду пришлось бы записать столько раз, сколько раз она повторяется (если вообще известно количество повторений). Однако это неэффективный способ записи алгоритмов. Поэтому для обозначения многократно повторяемых действий используют специальную конструкцию – *ЦИКЛ*. Составная *КОМАНДА ЦИКЛА,* называемая также *КОМАНДОЙ ПОВТОРЕНИЯ,* содержит условие, значение которого определяет количество повторений.

ния: $u_{n+1} = (-1)^{n+2} x^{2n+2}$, $\frac{u_{n+1}}{u_n} = \frac{(-1)^{n+2} x^{2n+2}}{(-1)^{n+1} x^{2n}} = \frac{(-1)^{n+1}(-1) x^{2n} x^2}{(-1)^{n+1} x^{2n}} = -x^2$. Значит

$u_{n+1} = -x^2 u_n$.

```
Program Sum_While;  {Вычисление суммы ряда}
  Uses Crt;
  Const eps=0.001;
  Var
        a,b,h,x,s,q,u,f :Real;
        k,n             :Integer;
Begin
  Clrscr;
  Write ('Введите отрезок [a,b]->'); ReadLn (a,b);
  Write ('Введите значение k->'); Readln (k);
  h:=(b-a)/k;
  x:=a;
WriteLn('       Таблица значений функции ');
WriteLn('┌──────┬──────────┬──────────┬──────────┬─────┐');
WriteLn('│  x   │    S     │    f     │    Q     │  n  │');
WriteLn('├──────┼──────────┼──────────┼──────────┼─────┤');
  While (x<=b) do
  begin
    s:=0;
    n:=1;
    u:=x*x;
    q:=u/2;
    While (abs(q)>eps) do
    begin
      s:=s+q;
      n:=n+1;
      u:=u*(-x*x);
      q:=u/((2*n-1)*2*n);
    end;
    f:=x*arctan(x)-ln(sqrt(1+x*x));
    WriteLn('│',x:4:2,'│',s:8:4,'│',f:8:4,'│',
            q:8:4,'│',n:3,'│');
    x:=x+h;
  end;
WriteLn ('└──────┴──────────┴──────────┴──────────┴─────┘');
  Repeat Until Keypressed;{задержка экрана пока}
End.                      {не нажата любая клавиша}
```

2. Общий член ряда $u_n = (-1)^n \frac{x^{2n+1}}{(2n+1)!}$, тогда $u_{n+1} = (-1)^{n+1} \frac{x^{2(n+1)+1}}{(2(n+1)+1)!} = (-1)^{n+1} \frac{x^{2n+3}}{(2n+3)!}$. Найдем отношение

$$\frac{u_{n+1}}{u_n} : \frac{u_{n+1}}{u_n} = \frac{(-1)^{n+1} x^{2n+3}}{(2n+3)!} \cdot \frac{(2n+1)!}{(-1)^n x^{2n+1}} = \frac{(-1)^n (-1) x^{2n+1} x^2}{(2n+1)!(2n+2)(2n+3)}$$

$$\cdot \frac{(2n+1)!}{(-1)^n x^{2n+1}} = -\frac{x^2}{(2n+2)(2n+3)}. \text{ Тогда } u_{n+1} = u_n \cdot \left(-\frac{x^2}{(2n+2)(2n+3)} \right).$$

Если при подстановке n=1 в общий член суммы будет получен первый член суммы, то начальное значение $S=0$, если же будет получен второй член суммы, то за начальное значение S_n принимают значение первого члена суммы.

Начальное значение для рекуррентных соотношений определяется из первых членов суммы ряда путем выделения в них той части, которая вычисляется рекуррентно.

Вычисление суммы организовывается в цикле. Когда при прохождении цикла номер члена суммы изменяется на 1, то сумма изменяется на его n-й член, т.е. $S_{n+1}=S_n+Q_{n+1}$, S_n — сумма n членов. Вычисления проводят до тех пор, пока не будет выполнено неравенство $|Q_n| \le \varepsilon$.

Пример: С точностью ε =0,001 подсчитать значение *Sn* функции *F(x)*, представленной разложением в ряд *S=S(x)*:

$$S = \frac{x^2}{2} - \frac{x^4}{12} + \ldots + (-1)^{n+1} \frac{x^{2n}}{(2n-1)2n} + \ldots$$

Результат сравнить со значением функции $F = x \arctan(x) - \ln\sqrt{1+x^2}$.

Вычисления произвести в диапазоне изменения аргумента x: $0.1 \le x \le 1$ с заданным шагом $h=(b-a)/k$, k=10. На печать выдавать в виде таблицы: аргумент x, значения S и f, количество членов ряда n, обеспечивающих заданную точность и значения члена ряда u_n.

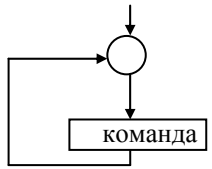
В данном примере общий член ряда $Q_n = (-1)^{n+1} \frac{x^{2n}}{(2n-1)2n}$ принадлежит к третьему типу. Представим его в виде произведения двух сомножителей $Q_n = u_n p_n$, где $u_n = (-1)^{n+1} x^{2n}$ вычисляется рекуррентно, а $p_n = \frac{1}{(2n-1)2n}$ – непосредственно. Найдем рекуррентные соотноше-

2.3.1. БЕСКОНЕЧНЫЙ ЦИКЛ

В некоторых случаях целесообразно использовать так называемый *БЕСКОНЕЧНЫЙ ЦИКЛ*:

повторять

< команда > ;



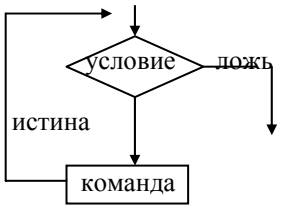
БЕСКОНЕЧНЫЙ ЦИКЛ играет фундаментальную роль в области процессов реального времени, операционных систем и т.д.

2.3.2. КОМАНДА ПОВТОРЕНИЯ С ПРЕДУСЛОВИЕМ (ЦИКЛ-ПОКА)

Обычно бывает необходимо повторять некоторое действие не бесконечно, а только *ПОКА* верно некоторое условие. В этом случае используют *КОМАНДУ ПОВТОРЕНИЯ С ПРЕДУСЛОВИЕМ* или *ЦИКЛ – ПОКА*:

пока < условие >**повторять**

< команда >;



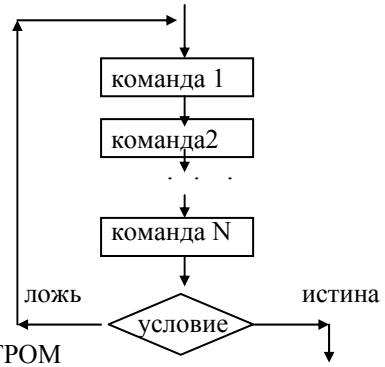
где условие – выражение, принимающее значение логических констант истина или ложь. Вычислительный процесс, представленный в виде *ЦИКЛА – ПОКА*, будет обладать свойством конечности, если команда предусматривает действия, изменяющие значение условия или если изначально значение условия - ложь.

Главная проблема, которая возникает в связи с использованием *ЦИКЛА – ПОКА*, это проблема его *ОКОНЧАНИЯ*: как можно гарантировать, что цикл завершится после некоторого числа итераций, для некоторого класса данных? Эта проблема не имеет решения в общем случае; на практике, однако, часто бывает возможно найти свойства условия и команды, обеспечивающие окончание циклического процесса.

2.3.2. КОМАНДА ЦИКЛА С ПОСТУСЛОВИЕМ (ЦИКЛ – ДО)

В отличие от предыдущего случая, когда команда может не выполниться ни разу, *ЦИКЛ С ПОСТУСЛОВИЕМ* или *ЦИКЛ – ДО* предусматривает выполнение команды по крайней мере один раз:

повторять
< команда 1 > ;
< команда 2 > ;
. .
< команда N >
до < условие >;



2.3.3. ЦИКЛ С ПАРАМЕТРОМ

Значительный интерес с практической точки зрения представляет базовая конструкция – *ЦИКЛ С ПАРАМЕТРОМ*, которая может использоваться в различных модификациях:

для всякого элемента x принадлежащего M **выполнить**
< команда > ;
для x принадлежащего M **пока** < условие > **повторять**
< команда > ;
для x **от** m **до** n **повторять**
< команда > ;
для x **от** m **до** n **шаг** h **повторять**
< команда > ;

3. КОМПОЗИЦИИ БАЗОВЫХ СТРУКТУР

В соответствии с так называемой «структурной теоремой», изложенной в классической работе итальянских математиков К.Бома и Г.Джакопини (1965 г.), всякая программа (алгоритм) может быть построена с использованием только трех управляющих конструкций (структур): следование, развилка и цикл. Каждая из этих конструкций имеет один вход и один выход. В силу этого развилка или цикл могут рассматриваться как обобщенный функциональный блок, т.е. «черный ящик» с одним входом и одним выходом. Таким образом, в конструкциях цикл и развилка функциональные блоки сами могут быть конструкциями такого же типа, поэтому возможны вложенные конструкции. При этом какова бы ни была глубина вложенности, важно, что любая конструкция в конечном итоге имеет один вход и один выход.

Также можно утверждать, что конструкции следование и цикл-пока принципиально достаточны, чтобы описать действие любой программы (алгоритма). Не пытаясь доказать этот общий результат, по-

ПРИБЛИЖЕННОЕ ВЫЧИСЛЕНИЕ ФУНКЦИЙ

Функцию $f(x)$, $n+1$ раз дифференцируемую на интервале (a,b), содержащем точку c, можно разложить в ряд Тейлора, т.е. представить в виде суммы многочлена n-ой степени и остаточного члена R_n:

$$f(x) = f(c) + \frac{f'(c)}{1!}(x-c) + \frac{f''(c)}{2!}(x-c)^2 + \cdots + \frac{f^{(n)}(c)}{n!}(x-c)^n + R_n.$$

Выражение $Q_n = \frac{f^{(n)}(c)}{n!}(x-c)^n$ называется общим членом ряда. Необходимым условием сходимости ряда является то, что общий член ряда по абсолютной величине стремится к нулю.

Вычислять значение суммы будем следующим образом: зададим начальное значение суммы, вычислим первый член суммы и добавим его к начальному значению, вычислим второй член суммы, третий и т.д., до тех пор, пока значение n-го члена суммы, по абсолютной величине, не будет меньше заданной точности ε (добавление его к сумме не повлияет на значение суммы). Формула общего члена ряда может принадлежать к одному из трех типов:

1) $\frac{\cos nx}{n}$; $\frac{\sin(2n-1)x}{2n-1}$; $\frac{\cos 2nx}{4n^2-1}$; ... 2) $\frac{x^n}{n!}$; $(-1)^n\frac{x^{2n+1}}{(2n+1)!}$; $\frac{x^{2n}}{(2n)!}$; ...

3) $\frac{x^{4n+1}}{4n+1}$; $(-1)^n\frac{\cos nx}{n^2}$; $\frac{n^2+1}{n!}\cdot\left(\frac{x}{2}\right)^n$; ...

В первом случае каждый член суммы вычисляется непосредственно по общей формуле. Во втором случае для вычисления суммы лучше всего использовать рекуррентные соотношения, т.е. выражать последующий член ряда через предыдущий. В последнем случае член суммы представляется в виде произведения двух сомножителей, первый из которых вычисляется с использованием рекуррентных соотношений, а второй вычисляется непосредственно.

Пример. Найдем рекуррентные соотношения для 2-х общих членов ряда:

1. Общий член ряда $u_n = \frac{x^n}{n!}$, тогда $u_{n+1} = \frac{x^{n+1}}{(n+1)!}$. Найдем отношение

$\frac{u_{n+1}}{u_n} : \frac{u_{n+1}}{u_n} = \frac{x^{n+1}}{(n+1)!}\cdot\frac{n!}{x^n} = \frac{x^n\cdot x}{n!\cdot(n+1)}\cdot\frac{n!}{x^n} = \frac{x}{n+1}$. Значит $u_{n+1} = u_n\cdot\frac{x}{n+1}$.

```
  WriteLn ('Поиск');
End;
Procedure Perest;
Begin
  Window(1,1,80,25);
  Frame(58,4,79,15,Blue,Yellow);
  WriteLn ('Перестановка');
End;
Procedure Sort;
Begin
  Window(1,1,80,25);
  Frame(58,4,79,15,blue,yellow);
  WriteLn ('Сортировка');
End;
Begin {основная программа}
  Clrscr;
  While True do
    Begin
      Window(1,1,80,25);
      GorMenu(1,1,4,Menu,reg);
      Case reg of
       1:Vvod;
       2:Vivod;
       3:begin
           Window(1,1,80,25);
           VertMenu(32,4,4,Menu3,reg3);
           case reg3 of
             1:Poisk;
             2:Perest;
             3:Sort;
           end;
           {очищаем подменю}
           Window(1,1,80,25);
           Window(32,4,55,10);
           TextBackGround(Blue);
           Clrscr;
           Continue;
         end;
       4:Exit
       end;
  end
End.
```


смотрим, например, как можно обойтись без *АЛЬТЕРНАТИВЫ* если … то … иначе:

если < условие >
то < команда 1 >
иначе < команда 2 >
все;

Пусть лог1 и лог2 – логические переменные. Тогда конструкция

лог1 : = < условие > ;
лог2 : = ~ < условие > ;
пока лог1 **повторять**
начало
< команда 1 > ;
лог1 : = ложь
конец ;
пока лог2 **повторять**
начало
< команда 2 > ;
лог2 : = ложь
конец;

эквивалентна предыдущей конструкции.

СРЕДА ПРОГРАММИРОВАНИЯ
Turbo Pascal

Система программирования Turbo Pascal представляет собой единство двух начал: компилятора с языком программирования Pascal и некоторой инструментальной программной оболочки. Для краткости условимся в дальнейшем называть реализуемый компилятором язык программирования Pascal языком Turbo Pascal, а разнообразные сервисные услуги, представляемые программной оболочкой — средой Turbo Pascal.

Для вызова системы Turbo Pascal следует дать команду:

turbo

По этой команде операционная система запустит на исполнение программу из файла *turbo.exe*: загрузит программу в оперативную память и передаст ей управление.

После успешного вызова системы экран компьютера приобретает вид:

Верхняя строка содержит меню возможных режимов работы Turbo Pascal, нижняя — краткую информацию о назначении основных функциональных клавиш. Вся остальная часть экрана принадлежит окну редактора. Новому файлу присваивается имя *noname00.pas*. В среде Turbo Pascal можно работать одновременно с несколькими программами, каждая из которых может располагаться в отдельном окне редактора. Кроме окна редактора, используются следующие окна: отладочного режима, вывода результатов работы программы, справочной службы, и т.д. По желанию они могут вызываться на экран поочередно или присутствовать на нем одновременно.

Пример основной программы формирования меню с подменю следующего вида:

```
Ввод      Вывод      Обработка      Выход
```

```
Поиск
Перестановка
Сортировка
Выход
```

```
Program Menu_with_menu;
  Uses
    Crt,Mybibl;
  Const
  Menu:mas_string=('Ввод      ',
                   'Вывод     ',
                   'Обработка',
                   'Выход     ','');
  Menu3:mas_string=('Поиск       ',
                    'Перестановка',
                    'Сортировка  ',
                    'Выход       ','');
  Var
   reg,reg3,i,j,k :Byte;
   kod            :Char;
Procedure Vvod;
Begin
  Window(1,1,80,25);
  Frame(1,4,25,15,Blue,Yellow);
  WriteLn ('Ввод матрицы');
End;
Procedure Vivod;
Begin
  Window(1,1,80,25);
  Frame(1,4,25,15,Blue,Yellow);
  WriteLn ('Вывод матрицы');
End;
Procedure Poisk;
Begin
  Window(1,1,80,25);
  Frame(58,4,79,15,Blue,Yellow);
```

```
    kod:Char;
Procedure Vvod;
Begin
  Window(1,1,80,25);
  Frame(40,1,70,10,Blue,Yellow);
  WriteLn ('Ввод матрицы');
  {окно с рамкой с заданным фоном}
  ...
End;
Procedure Vivod;
Begin
  Window(1,1,80,25);
  Frame(40,1,70,10,Blue,Yellow);
  WriteLn ('Вывод матрицы');
  ...
End;
Procedure Obrabotka;
Begin
  Window(1,1,80,25);
  Frame(40,11,70,20,Blue,Yellow);
  WriteLn ('Обработка');
  ...
End;
Begin {основная программа}
  Clrscr;
  While True do
  begin
    Window(1,1,80,25);
    VertMenu(1,1,4,Menu,reg);
    Case REG of
     1:Vvod;
     2:Vivod;
     3:Obrabotka;
     4:Exit
    end;
  end
End.
```

Для формирования горизонтального меню необходимо лишь заменить вызов процедуры вертикального меню на процедуру горизонтального меню.

Функциональные клавиши

Функциональные клавиши используются для управления средой Turbo Pascal. Они обозначаются *F1*, *F2*, ..., *F12* и располагаются в верхнем ряду клавиатуры.

Действия почти всех функциональных клавиш можно модифицировать тремя особыми клавишами: *Alt* (дополнительный), *Ctrl* (управление), *Shift* (сдвиг). Назначения основных функциональных клавиш:

F1 – обратиться за справкой к встроенной справочной службе (*Help* – помощь);

F2 – запись редактируемого текста в дисковый файл;

F3 – прочитать текст из дискового файла в окно редактора;

F4 – используется в отладочном режиме;

F5 – распахнуть активное окно на весь экран;

F6 – сделать активным следующее окно;

F7 – используется в отладочном режиме;

F8 – используется в отладочном режиме;

F9 – компилировать программу, но не выполнять ее;

F10 – перейти к диалоговому выбору режима работы с помощью главного меню;

Ctrl-F9 – выполнить прогон программы, компилировать программу, находящуюся в редакторе, загрузить ее в оперативную память и выполнить, после чего вернуться в среду Турбо Паскаля;

Alt-F5 — сменить окно редактора на окно вывода результатов работы программы.

Порядок работы с Pascal– программой

1. Набор текста программы. Текстовый редактор среды Turbo Pascal предоставляет пользователю удобные средства создания и редактирования текстов программы. Из режима редактирования можно перейти к любому другому режиму работы Turbo Pascal с помощью функциональных клавиш или выбора нужного режима из главного меню.

Для перехода от состояния выбора режима из главного меню в состояние редактирования нужно нажать клавишу *Esc*, а для перехода к выбору из главного меню – *F10*.

Для создания текста программы нужно ввести этот текст с помощью клавиатуры. После заполнения очередной строки следует нажать на клавишу *Enter*, чтобы перейти курсором на следующую строку.

Окно редактора имитирует длинный и достаточно широкий лист бумаги, фрагмент которого виден в окне. Окно можно смещать относительно листа с помощью клавиш:

PgUp – на страницу вверх,
PgDn – на страницу вниз,
Home – в начало текущей строки,
End – в конец текущей строки,
Ctrl - PgUp – в начало текста,
Ctrl - PgDn – в конец текста.

Если вы ошиблись при выборе очередного символа, его можно стереть с помощью клавиши *Backspace*. Клавиша *Del* стирает символ, на который в данный момент указывает курсор. Команда *Ctrl-Y* удаляет всю строку, на которую указывает курсор. Команда *Ctrl–Q-L* восстанавливает текущую строку (действует, если курсор не покидал измененную строку).

Начальный режим работы редактора – режим вставки. Редактор также может работать в режиме наложения. Переключает эти режимы клавиша *Insert*.

Обычно редактор работает в режиме автоотступа. В этом режиме каждая новая строка начинается в той же позиции на экране, что и предыдущая. Отказ от автоотступа – команда *Ctrl-O-I*. Повтор этой команды восстанавливает режим автоотступа.

Команды работы с блоками:
Ctrl-K-B – пометить начало блока,
Ctrl-K-K – пометить конец блока,
Ctrl-K-Y – стереть блок,
Ctrl-K-C – копировать блок,
Ctrl-K-V – переместить блок,
Ctrl-K-W – записать блок в дисковый файл,
Ctrl-K-R – прочитать блок из дискового файла,
Ctrl-K-P – напечатать блок.

2. Запись программы на диск. Основной формой хранения текстов программ вне среды являются файлы. Если вы создали новую программу, то среда Turbo Pascal еще не знает такого файла. Сохранить текст программы можно двумя способами:

1. Нажать функциональную клавишу *F2*.
2. Войти в главное меню (клавиша *F10*), выбрать команду *File* и в появившемся подменю команду *Save* или *Save as*.

На экране появится небольшое окно запроса с надписью в верхней части: *Save File as* (сохранить файл с именем).


```
      {вывод строки меню}
      GotoXY((i-1)*15+1,1);
      Write(Stor[i]);
  end;
  {отображение выбора с помощью стрелок}
  kod:=ReadKey;{считывание символа}
  {если нажата функциональная клавиша}
  If kod=#0 then
  begin
    {считывание второго байта}
    kod:=ReadKey;
    {если нажата стрелка ввлево}
    If kod=#75 then
    begin
      If k>1 then k:=k-1
             else k:=L
    end;
    {если нажата стрелка вправо}
    If kod=#77 then
    begin
      If k<L then k:=k+1
             else k:=1;
    end;
  end;
  regim:=k;
  end;
end;

BEGIN
END.
```

Пример основной программы, вызывающей вертикальное меню:

```
Program Prim_Vert_Menu;
  Uses
    Crt,Mybibl;
  Const {задаем пункты меню}
   Menu:mas_string=('Ввод      ',
                    'Вывод     ',
                    'Обработка',
                    'Выход     ','');
  Var
    reg,i,j,k:Byte;
```

```
    kod:=ReadKey;
    {если нажата стрелка вверх}
    If kod=#72 then
    begin
      If k>1 then k:=k-1
             else k:=L
    end;
    {если нажата стрелка вниз}
    If kod=#80 then
    begin
       If k<L then k:=k+1
              else k:=1;
    end;
  end;
  regim:=k;
  end; {while}
end;

Procedure GorMenu(x1,y1,L:Byte;Stor:mas_string; var
regim:Byte);
  {процедура формирования горизонтального меню}
  Var
    k,i:Byte;
    kod:Char;
Begin
  Frame (x1,y1,79,3,Blue,Yellow); {задаем окно}
  k:=1;
  kod:=' ';
  While kod<>#13 do
  begin
    For i:=1 to L do
    begin
      If i=k then
               begin
                 TextBackGround(Green);
                 TextColor(Yellow);
               end
             else
               begin
                 TextBackGround(Blue);
                 TextColor(Yellow);
               end;
```


Ниже надписи располагается поле для ввода имени файла, в которое можно написать любое имя и нажать клавишу *Enter*. Текст будет сохранен.

3. Компиляция программы. Откомпилировать программу можно двумя способами:

1. Нажать комбинацию клавиш *Alt-F9*.
2. Войти в главное меню (клавиша *F10*), выбрать команду *Compile* и в появившемся подменю команду *Compile.*

Если транслятор обнаружит синтаксическую ошибку, он прервет компиляцию, выдаст соответствующее сообщение на экран, указав место ошибки. При наличии ошибки необходимо ее исправить, записать измененный текст программы и заново откомпилировать.

4. Выполнение программы и просмотр результатов. После успешной компиляции программы можно попытаться выполнить ее. Прогон программы реализуется двумя способами:

1. Нажать комбинацию клавиш *Ctrl-F9*.
2. Войти в главное меню (клавиша *F10*), выбрать команду *Run* и в появившемся подменю команду *Run.*

Если во время выполнения программы обнаружена ошибка, среда прекращает дальнейшие действия, восстанавливает окно редактора и помещает курсор на ту строку программы, где была обнаружена ошибка. Сложные ошибки выявляются с помощью пошагового исполнения программы, связанного с клавишами *F4*, *F7*, *F8*.

Просмотр результатов выполнения программы осуществляется двумя способами:

1. Нажать комбинацию клавиш *Alt-F5.*
2. Войти в главное меню (клавиша *F10*), выбрать команду *Run* и в появившемся подменю команду *User Screen.*

5. Выход из системы Турбо Паскаль. Выйти из среды Турбо Паскаль можно:

1. Нажав комбинацию клавиш *Alt-X.*
2. Войти в главное меню (клавиша *F10*), выбрать команду *File* и в появившемся подменю команду *Exit.*

ЯЗЫК ПРОГРАММИРОВАНИЯ Pascal

АЛФАВИТ И СЛОВАРЬ ЯЗЫКА

При записи алгоритма решения задачи на языке программирования необходимо знать правила написания и использования элементарных информационных и языковых единиц.

Программа на языке Паскаль формируется с помощью конечного набора знаков, образующих *алфавит* языка, и состоит из букв, цифр, специальных символов.

В качестве *букв* используются прописные и строчные буквы латинского алфавита и знак подчеркивания; в качестве *цифр* – арабские цифры от 0 до 9.

При написании программ применяются *специальные символы*: +, -, *, /, =, <, >, [ ], ( ), @, { }, :, ;, ', # (номер), $ (знак денежной единицы), ^ (тильда), пробел, точка и запятая.

Комбинации специальных символов могут образовывать *составные символы*: := (присваивание), <> (не равно), .. (диапазон значений), <= (меньше или равно), >= (больше или равно), (* *) — альтернатива {}, (..) — альтернатива [].

Неделимые последовательности знаков алфавита образуют слова, отделенные друг от друга разделителями и несущие определенный смысл в программе. Разделителем может служить пробел, символ конца строки, комментарий. Слова подразделяются на зарезервированные, стандартные идентификаторы и идентификаторы пользователя.

Зарезервированные слова являются составной частью языка и их нельзя использовать в качестве идентификаторов. В языке Паскаль зарезервированными являются следующие слова: `and, array, begin, case, const, div, do, downto, else, end, file, for, forward, function, goto, if, in, label, mod, nil, not, of, or, packed, procedure, program, record, repeat, set, shl, shr, string, then, to, type, unit, until, uses, var, while, with, xor.`

Стандартные идентификаторы служат для обозначения заранее определенных разработчиками языка типов данных, констант, процедур и функций.

Идентификаторы пользователя используются для обозначения меток, констант, типов, переменных, процедур и функций, определенных самим программистом.


```
  {задание окна внутри рамки}
  Window (x1+1,y1+1,x2-1,y2-1);
  TextColor(cvet);
  TextBackground(fon);
  Clrscr;
End;

Procedure     VertMenu(x1,y1,L:Byte;Stor:mas_string;
var regim:Byte);
  {процедура формирования вертикального меню}
  Var
    k,i:Byte;
    kod:Char;
Begin
  {задаем окно}
  frame (x1,y1,x1+20,y1+L+1,blue,Yellow);
  k:=1; {номер режима}
  kod:=' ';
  While kod<>#13 do
  begin
   For i:=1 to L do
   begin
      If i=k then
                 begin
                    TextBackGround(Green);
                    TextColor(Yellow);
                 end
              else
                 begin
                    TextBackGround(Blue);
                    TextColor(Yellow);
                 end;
      {вывод строки меню}
      GotoXY(1,i);
      Write(Stor[i]);
  end;
  {отображение выбора с помощью стрелок}
  kod:=ReadKey;{считывание символа}
  {если нажата функциональная клавиша}
  If kod=#0 then
  begin
     {считывание второго байта}
```

```
IMPLEMENTATION

Procedure Frame(X1,y1,x2,y2,fon,cvet:Integer);
{Процедура черчения рамок заданного цвета и фона}
{X1,Y1,X2,Y2 - координаты соответственно
 левого верхнего и правого нижнего угла рамки}
  Const
    {для черчения двойной линии}
    A=#186; B=#187; C=#188;
    D=#200; E=#201; F=#205;
    {для черчения одинарной линии
    A=#179; B=#191; C=#217;
    D=#192; E=#218; F=#196;}
  Var
    i,j:Integer;
Begin
  TextColor(cvet);
  GotoXY(X1,Y1);
  Write(E);    {левый верхний угол}
  {горизонтальная линия}
  For i:=X1+1 to X2-1 do Write(F);
  Write(B);     {правый верхний угол}
  {вертикальные линии}
  For i:=Y1+1 to Y2-1 do
  begin
    GotoXY(x1,i);  {переходим к левой границе}
    Write(A);      {левая черта              }
    GotoXY(X2,i);  {правая граница           }
    Write(A);      {правая черта             }
  end;
  GotoXY(x1,y2);
  Write(D);       {левый нижний угол}
  {расширяем вниз на одну строку координаты окна, иначе вывод в
правый нижний угол вызовет прокрутку окна вверх}
  Window(X1,Y1,X2,Y2+1);
  {возвращаем курсор из левого верхнего угла окна в нужное место}
  GotoXY(2,Y2-Y1+1);
  {горизонтальная рамка}
  For i:=X1+1 to X2-1 do Write(F);
  Write(C);     {правый нижний угол}
  { Определяем внутреннюю часть окна}
```


Правила написания идентификаторов

1. Идентификатор начинается с буквы или символа подчеркивания (исключение составляют метки, которые могут начинаться и с цифры, и с буквы).
2. Идентификатор может состоять из букв, цифр и знака подчеркивания (пробелы, точки и другие специальные символы недопустимы). При написании идентификаторов можно использовать как прописные, так и строчные буквы, однако Турбо Паскаль 7.0 не различает прописные и строчные буквы, поэтому записи `Writeln, WRITELN, WriteLn` эквивалентны.
3. Между двумя идентификаторами должен быть по крайней мере один пробел.
4. Максимальная длина идентификатора — 127 символов, но значимы только первые 63 символа.
5. Идентификаторы нужно делать “осмысленными”. Для создания идентификаторов, состоящих из двух слов, можно воспользоваться большими буквами (например, `ReadText`) или символом подчеркивания (`Read_Text`) (это гораздо лучше чем `RT`).
6. Все структуры языка имеют англоязычные идентификаторы. Можно использовать русские идентификаторы (записанные английскими литерами, например `Privetstvie`), но для удобства лучше выполнять не транслитерацию русских слов в английские, а перевод их на английский язык (например, `Hello`).

ДАННЫЕ

Все данные, в зависимости от способа их хранения и обработки, можно разделить на две группы: *константы* и *переменные.*

Константами называются элементы данных, значения которых установлены в описательной части программы и в процессе выполнения программы они не изменяются.

Формат описания констант:

```
Const
    идентификатор=значение;
```

Стандартные виды констант

1. *Целочисленные* – определяются посредством чисел, записанных в десятичном или шестнадцатеричном формате, не содержащих десятичной точки.

2. *Вещественные* – определяются посредством чисел, записанных в десятичном формате данных.
3. *Символьные* – это любой символ персонального компьютера, заключенный в апострофы.
4. *Строковые* – определяются последовательностью произвольных символов, заключенных в апострофы.
5. *Логические* – это либо `False`, либо `True`.
6. *Типизированные* – это переменные с начальным значением. Каждой типизированной константе ставится в соответствие имя, тип и начальное значение.

   Описание типизированных констант:

```
Const
    идентификатор:тип=значение;
```

7. *Зарезервированные* константы:

Идентификатор	Тип	Значение	Описание
`Pi`	`Real`	3.1415926536	Число π
`True`	`Boolean`	True	Истина
`False`	`Boolean`	False	Ложь
`Maxint`	`Integer`	32767	Максимальное целое число

Пример:

```
Const
    A=2;                    {целая}
    B=2.35;                 {вещественная}
    St='g';                 {символьная}
    R='ПРИВЕТ !';           {строковая}
    Year:Integer=2005;      {типизированная}
```

Переменные в отличие от констант могут менять свои значения в процессе выполнения программы. Каждая константа и переменная принадлежат к определенному типу данных. Тип констант автоматически распознается компилятором без предварительного описания. Тип переменной должен быть описан перед тем, как с переменными будут выполняться какие-либо действия.

Формат описания переменных:

```
Var
    идентификатор:тип;
```



```
      Write ('Такого режима нет. Повторите!');
    end {else}
  end {case}
end {while}
End.
```

Горизонтальное меню представляет собой строку из списка режимов, один из которых (обычно первый) выделен цветом. Выбор нужного пункту меню осуществляется нажатием клавиш →, ←, активизация режима – клавиша *Enter*.

ГОРИЗОНТАЛЬНОЕ МЕНЮ

Ввод	Вывод	Обработка	Выход

Вертикальное меню – отличается от горизонтального расположением списка режимов.

ВЕРТИКАЛЬНОЕ МЕНЮ

Ввод
Вывод
Обработка
Выход

Модуль Mybibl, содержащий горизонтальное и вертикальное меню, процедуру формирования рамки и установки цветов фона и символов:

```
Unit MYBIBL;

INTERFACE
Uses CRT;
Type
    mas_string=array[1..5] of String;
Procedure Frame(X1,Y1,X2,Y2,fon,cvet:INTEGER);
Procedure    VertMenu(x1,y1,L:Byte;Stor:mas_string;
var regim:Byte);
Procedure GorMenu(x1,y1,L:Byte;Stor:mas_string; var
regim:Byte);
```

```
    '4 - выход');
  Var
    i,j: Byte;
Procedure Zast;
Begin
   {процедура заставка}
End;
Procedure Vvod;
Begin
   {процедура ввода}
End;
Procedure Vivod;
Begin
   {процедура вывода}
End;
Procedure Obrabotka;
Begin
   {процедура обработки}
End;
Begin
  Zast;
  While True do
  begin
    Clrscr;
    GotoXY(24,4);
    Write ('ГЛАВНОЕ МЕНЮ');
    For j:=1 to 4 do
    begin
      GotoXY(25,7+j);
      Write (Nameregim[j]);
    end;
    GotoXY(20,15);
    Write ('Выберете режима и нажмите Enter');
    ReadLn(i);
    Case i of
    1: Vvod;
    2: Vivod;
    3: Obrabotka;
    4: Exit
    else
    begin
      GotoXY(20,17);
```


Типы данных

Тип – это множество значений, которые могут принимать объекты программы, и совокупность операций, допустимых над этими значениями.

Целые типы. Диапазон возможных значений целых типов зависит от их внутреннего представления.

Тип	Название	Длина, байт	Диапазон значений
Byte	Длиной в байт	1	0..255
ShortInt	Короткое целое	1	-128..127
Word	Длиной в слово	2	0..65535
Integer	Целое	2	-32768..32767
LongInt	Длинное целое	4	-2147483648.. 2147483647

Логический тип (Boolean). Значениями логического типа может быть одна из констант False или True.

Символьный тип (Char). Значениями символьного типа является множество всех символов персонального компьютера. Для кодировки используется код ASCII (American Standart Code for Information Interchange – американский стандартный код для обмена информации).

Перечисляемый тип. Перечисляемый тип задается перечислением тех значений, которые он может получить. Каждое значение именуется некоторым идентификатором и располагается в списке, обрамленном круглыми скобками.

Формат задания типа:

```
Type
    имя_типа=тип_значений;
Var
    идентификатор: имя_типа;
```

Пример задания перечисляемого типа:

```
Type
    Colors=(black, red, white);
Var Col:colors;
```

Переменные перечисляемого типа можно объявлять без предварительного описания типа:

```
Var
    Col:(black, white, green);
```

Тип-диапазон. Тип-диапазон есть подмножество своего базового типа, в качестве которого может выступать любой скалярный тип, кроме вещественного и типа-диапазона. Тип-диапазон задается границами своих значений внутри базового типа:

```
минимальное_значение .. максимальное_значение
```

Пример:

```
Type
    Digit='0'..'9';
    Dig2=48..57;
Var
    D1:Digit;
    D2:Dig2;
```

Тип-диапазон можно непосредственно указывать при объявлении переменной. *Пример*:

```
Var
    Date:1..31;
    Month:1..12;
```

Вещественные типы. Значения вещественных типов определяют произвольное вещественное число с некоторой конечной точностью, зависящей от внутреннего формата числа.

Тип	Название	Длина, байт	Кол-во цифр мантисы	Диапазон десятичного порядка
Real	Вещественный	6	11..12	-39..38
Single	С одинарной точностью	4	7..8	-45..38
Double	С двойной точностью	8	15..16	-324..308
Extended	С повышенной точностью	10	19..20	-4932..4932
Comp	Сложный	8	10..20	$-2*10^{63}+1..$ $2*10^{63}-1$

Программа в общем случае состоит из заставки, набора процедур, функций и глобального блока (функционирующего посредством меню).

Заставка является визитной карточкой программы. Она выводится на экран сразу после старта программы и содержит информацию о названии программы, ее назначении, авторе и т.д.

Меню – это перечисление возможностей системы, из которого пользователь выбирает нужную в текущий момент. Меню должно быть простым в работе и понятым для самого неподготовленного пользователя. Более или менее сложная система обычно имеет несколько меню. Среди них выделяется главное, наиболее общее меню. Каждый элемент главного меню может генерировать новое (вложенное) меню, являющееся второстепенным по отношению к главному. В свою очередь второстепенное меню может также активизировать подчиненное ему меню и т.д. Уровень вложения меню ограничивается только логической структурой решаемой задачи.

Виды меню

Простой запрос представляет собой наиболее несложный вид меню. Выбор режима меню осуществляется нажатием цифры выбранного режима меню.

ГЛАВНОЕ МЕНЮ

1 – ввод
2 – вывод
3 – обработка
4 – выход

Выберете режим и нажмите *Enter*

Пример простого меню:

```
Program Simple_Menu;
Uses
  Crt;
Const
  K=4; {Количество режимов}
  Nameregim:array[1..K] of String[26]=
   ('1 - ввод',
    '2 - вывод',
    '3 - обработка',
```

В режиме *Build* существующие *tpu*-файлы игнорируются, и система пытается отыскать и компилировать соответствующий pas-файл для каждого объявленного в разделе `Uses` модулей.

Стандартные модули

В системе программирования Pascal имеется 8 стандартных модулей, в которых содержится большое число разнообразных типов, констант, процедур и функций: *System*, *Dos*, *Crt*, *Printer*, *Graph*, *Overlay*, *Turbo3* и *Graph3*. Модули *Graph*, *Turbo3* и *Graph3* выделены в отдельные tpu-файлы, а остальные входят в состав библиотечного файла *Turbo.tpl*. Лишь модуль `System` подключается к любой программе автоматически, все остальные становятся доступны только после указания их имен разделе `Uses`.

В модуль *System* входят все процедуры и функции стандартного языка программирования Pascal, а также встроенные процедуры и функции, которые не вошли в другие стандартные модули.

Модуль *Printer* делает доступным вывод текстов на принтер.

В модуле *Crt* сосредоточены процедуры и функции, обеспечивающие управление текстовым режимом работы экрана.

Модуль *Graph* содержит большой набор типов, констант, процедур и функций для управления графическим режимом работы экрана.

В модуле *Dos* собраны процедуры и функции, открывающие доступ программам к средствам дисковой операционной системы MS-DOS.

Модуль *Overlay* используется при разработке громоздких программ с перекрытиями.

Модули *Turbo3* и *Graph3* введены для совместимости с ранней версией 3.0 системы Turbo Pascal.

СОЗДАНИЕ ЛИЧНОЙ БИБЛИОТЕКИ ПРОГРАММИСТА

Каждый опытный программист со временем накапливает определенное количество процедур и функций, которые использует как готовые блоки при разработке различных программ. Организуем из них библиотечный модуль Mybibl и откомпилируем его с размещением на диске результата (опция компиляции *Build*). После выполнения компиляции на диске создается файл Mybibl.tpu, доступный для использования без каких-либо дополнительных описаний. Этот файл можно подключать в любую программу следующим образом:

```
Uses Mybibl;
```


Пример:

```
Var
    Min:Real;
```

Операции

Операции делятся на 4 категорий согласно приоритету:

1) унарный минус, `not` — логическое отрицание; @ – получение адреса операнда;
2) операции группы умножения — * (умножить), `mod` (деление по модулю), `div` (целочисленное деление), / (деление), `and` (логическое "и"), `shl` – сдвиг влево, `shr` – сдвиг вправо;
3) операции группы сложения: + (сложение), – (вычитание), `or` (логическое "или"), `xor` (исключающее "или");
4) операции отношения: = (равно), <> (не равно), > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), `in` (проверка принадлежности множеству).

Пример:

```
F:=2*sin(x)+3/sqrt(1+x*x);
```

Операции целочисленной арифметики

Целочисленное деление (`div`) – возвращает целую часть частного, дробная часть отбрасывается. Результат целочисленного деления всегда равен нулю, если делимое меньше делителя.

Деление по модулю (`mod`) – возвращает остаток, полученный в результате целочисленного деления.

Пример:

```
11 div 5 = 2;  2 div 3=0;
11 mod 5=1;    14 mod 5=4
```

Таблица истинности для логических операций

Операция	Пример	Значение A	Значение B	Результат
not Логическое отрицание	Not A	True False		False True
and Логическое умножение	A and B	True True False False	True False True False	True False False False

Операция	Пример	Значение A	Значение B	Результат
or Логическое сложение	A or B	True True False False	True False True False	True True True False
xor Исключающее или	A xor B	True True False False	True False True False	False True True False

Сдвиговые операции (shl, shr)*:*

`i shl j` – сдвиг содержимого i на j разрядов влево; освободившиеся младшие разряды заполняются нулями (сдвиг влево на 1 разряд эквивалентен умножения числа *i* на 2);

`i shr j` – сдвиг содержимого i на j разрядов вправо; освободившиеся старшие разряды заполняются нулями (сдвиг вправо на 1 разряд эквивалентен делению числа *i* на 2).

Выражения

Выражения — алгоритмические конструкции, задающие правила вычисления значений. Выражения состоят из операндов и знаков операций.

Стандартные математические функции:

`abs(x)` – $|x|$,

`arctan(x)` – arctg$x$,

`cos(x)` – cos$x$,

`exp(x)` – e^x,

`int(x)` – целая часть выражения x,

`frac(x)` – дробная часть выражения x,

`ln(x)` – ln$x$,

`sin(x)` – sin$x$,

`sqr(x)` – x^2,

`sqrt(x)` – $\sqrt{x}$,

`Random` – псевдослучайное число, равномерно распределенное в диапазоне 0..1;

`Random(x)` – псевдослучайное число, равномерно распределенное в диапазоне 0..x-1;

`Randomize` – инициация генератора псевдослучайных чисел.


```
    End;
BEGIN                          {инициирующая часть}
    …
END.
```

После слова `Unit` записывается имя модуля. Оно служит для связи модуля с основной программой и другими модулями. Имя модуля должно совпадать с именем дискового файла, где находится исходный текст программы.

В секции `INTERFACE` описываются *глобальные* данные, заголовки процедур и функций, доступные основной программе и другим модулям.

В секции `IMPLEMENTATION` реализуется программный код глобальных процедур и функций и описываются локальные данные, процедуры и функции, недоступные основной программе и другим модулям.

Инициирующая часть завершает модуль. Она может отсутствовать вместе с начинающим её словом `BEGIN` или быть пустой. В инициирующей части размещаются исполняемые операторы, содержащие некоторый фрагмент программы. Эти операторы исполняются до передачи управления основной программе и обычно используются для подготовки её работы.

Компиляция модулей

В среде программирования Pascal определены 3 режима компиляции: *Compile*, *Make*, *Build*. Режимы отличаются только способом связи компилирующего модуля или основной программы с другими модулями.

При компиляции модуля или основной программы в режиме *Compile* все упоминающиеся в предложении `Uses` модули должны быть предварительно откомпилированы и результаты компиляции помещены в одноименные файлы с расширением *tpu*.

В режиме *Make* компилятор проверяет наличие *tpu*–файлов для каждого объявленного модуля. Если какой-либо из файлов не обнаружен, система пытается отыскать одноименный файл с расширением pas, т.е. файл с исходным текстом модуля, и, если исходный файл найден, приступает к его компиляции. Кроме того, в этом режиме система следит за возможными изменениями исходного текста любого используемого модуля, и если в исходный текст были внесены изменения, то модуль будет перекомпилирован.

`Sound(Hz);` – включает внутренний динамик. Hz задает частоту генерируемого динамиком сигнала в герцах. Звуковой сигнал подается до тех пор, пока не будет выключен процедурой `NoSound`.

`NoSound;` – отключает внутренний динамик.

МОДУЛИ

Модуль – это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции) и, возможно, некоторые исполняемые операторы инициирующей части. Модули используются для разработки библиотек прикладных программ. Важная особенность модулей заключается в том, что компилятор Паскаля размещает их программный код в отдельном сегменте памяти. Максимальная длина сегмента не может превышать 64 Кбайта, однако количество одновременно используемых модулей ограничивается лишь доступной памятью.

Структура модуля

Модуль имеет следующую структуру:

```
UNIT имя_модуля;                    {заголовок модуля}
{$директивы_компилятора}
INTERFACE                           {интерфейсная часть}
   Uses                             {имена подключаемых модулей}
   Const                            {раздел описания констант}
   Type                             {раздел описания типов}
   Var                              {раздел описания переменных}
   Procedure имя (параметры);       {заголовки процедур}
   Function имя (параметры):тип_результата;{заголовки функций}
IMPLEMENTATION                      {исполняемая часть}
   Uses
   Const
   Type
   Var
   Procedure имя;                   {тело процедуры}
   Begin
   …
   End;
   Function имя;                    {тело функции}
   Begin
   …
```


Скалярные функции обрабатывают данные любого скалярного типа, кроме вещественного:

`Pred (S)` — возвращает элемент, предшествующий S;

`Succ (S)` — возвращает значение, следующее за S;

`Odd (I)` — возвращает значение булевского типа, равное `True`, если I нечетное, и `False`, если I четное.

Функции преобразования типов:

`Round(x)` – возвращает значение *x*, округленное до ближайшего целого числа, результат имеет целочисленный тип;

`Trunc(x)` – возвращает ближайшее целое число меньшее или равное *x*, если $x \geq 0$, большее или равное *x*, если $x<0$. Результат имеет целочисленный тип;

`Chr(i)` – возвращает символ стандартного кода обмена информацией с номером, равным значению *i*.

`Ord(s)` – возвращает порядковый номер значения *s* во множестве, определенном типом *s*.

Пример:

```
Round(3.7);          результат  4
Round(-46.317)       результат –46
Trunc(3.7);          результат 3
```

Стандартные процедуры и функции, применяемые к целым типам:

`Dec(i)` – уменьшает значение переменной I на единицу (аналог оператора i:=i-1);

`Inc(i)` – увеличивает значение переменной I на единицу (аналог оператора i:=i+1);

`Hi(i)` – возвращает старший байт аргумента (тип аргумента byte или word);

`Lo(i)` – возвращает младший байт аргумента (тип аргумента byte или word);

`Swap(i)` – меняет местами байты в слове.

Стандартные функции, применимые к символьному типу:

`Chr(I)` – возвращает символ стандартного кода обмена информацией с номером, равным значению I;

`UpCase(ch)` – функция меняет регистр латинских букв со строчной на прописную.

Стандартные функции, поддерживающие работу с типами-диапазонами:

`High(x)` – возвращает максимальное значение типа-диапазона, к которому принадлежит переменная x;

`Low(x)` – возвращает минимальное значение типа-диапазона.

СТРУКТУРА ПРОГРАММЫ

```
Program имя_программы;
{РАЗДЕЛ ОПИСАНИЙ}
    Uses        {подключаемые модули        }
    Label       {объявление глобальных меток }
    Const       {объявления констант         }
    Type        {объявления типов            }
    Var         {объявления переменных       }
    Procedure   {описание процедур           }
    Function    {описание функций            }
{РАЗДЕЛ ОПЕРАТОРОВ}
Begin
    {операторы}
End.
```

Любой из разделов, кроме раздела операторов может отсутствовать. В любом месте программы могут содержаться комментарии, которые заключаются в { } или (* *).

ОПЕРАТОРЫ

Операторы выполняются в том порядке, в котором они записаны в программе. Разделителем операторов служит точка с запятой.

Все операторы разделяются на 2 группы: простые и структурные.

Операторы, не содержащие внутри себя других операторов, называются *простыми.* К ним относятся операторы присваивания, безусловного перехода, пустой оператор и оператор вызова процедур. *Структурные* операторы представляют собой структуры, построенные из других операторов по строго определенным правилам. Все структурные операторы подразделяются на 3 группы:

- составные,
- условные операторы,
- операторы повтора.

Коды управляющих клавиш и их сочетаний со сдвиговыми

Клавиша или комбинация клавиш	Первая часть кода	Вторая часть кода
Home	0	71
↑	0	72
Page Up	0	73
←	0	75
→	0	77
End	0	79
↓	0	80
Page Down	0	81
Insert	0	82
Delete	0	83
Ctrl←	0	115
Ctrl→	0	116
Ctrl–End	0	117
Ctrl–Rage Down	0	118
Ctrl–Home	0	119
Ctrl–Page Up	0	132

Процедуры управления строками на экране

`CrtExit` – восстанавливает режим, который был установлен при загрузке операционной системы.

`CrtInit` – выводит на экран строку инициализации терминала, определенную при установке системы.

`CrlEol` – стирает все символы в строке, начиная с текущей позиции курсора до конца строки.

`ClrScr` – полностью очищает экран и помещает курсор в левый верхний угол экрана.

`DelLine` – полностью стирает содержимое строки, в которой расположен курсор, все нижестоящие строки перемещаются на одну позицию вверх.

`InsLine` – вставляет пустую строку в место расположения курсора.

Эти процедуры обычно используются для изменения части экрана в сочетании с процедурой управления перемещением курсора `GotoXY`.

Процедуры управления звуком

`Delay(Time);` – вызывает задержку выполнения программы на Time милисекунд;

няющая те же функции, что и полный экран. После определения окна все координаты задаются относительно активного окна (начиная с первой позиции его левого верхнего угла), а не полного экрана. `x1, y1` – координаты левого верхнего угла окна, `x2, y2` – координаты правого нижнего угла окна.

`Clrscr;` – очищает активное окно и устанавливает курсор в левый верхний угол;

`ClrEol;` – очищает строку активного окна от текущей позиции курсора до конца строки без изменения позиции курсора;

`GotoXY(x,y);` – перемещает курсор в позицию с координатами x, y в рамках активного окна;

Функции работы с окнами

`WhereX;` – возвращает x-координату текущей позиции курсора (относительно активного окна);

`WhereY;` – возвращает y-координату текущей позиции курсора (относительно активного окна);

Программирование клавиатуры

`KeyPressed;` – функция возвращает значение типа `Boolean`, указывающее состояние буфера клавиатуры: `False` означает, что буфер пуст, а `True` – что в буфере есть хотя бы один символ, еще не прочитанный программой.

`ReadKey;` – функция считывает код символа с клавиатуры и возвращает значение типа `Char`. При обращении к этой функции анализируется буфер клавиатуры: если в нем есть хотя бы один не прочитанный символ, код этого символа берется из буфера и возвращается в качестве значения функции, в противном случае функция будет ожидать нажатия на любую клавишу.

Специальные клавиши генерируют расширенные коды клавиш, состоящие из двух значений, причем первое всегда равно #0. При нажатии специальной клавиши функция возвращает сначала нулевой символ #0, а затем вторую (расширенную) часть кода.

Функция `ReadKey` игнорирует нажатие на так называемые сдвиговые клавиши *Shift*, *Ctrl*, *Alt* и переключающие клавиши *Caps Lock*, *Num Lock*, *Scroll Lock* и клавиши *F11*, *F12*.

Чтобы получить код клавиши, можно воспользоваться функцией `Ord(ReadKey)`.

Совместимость типов

Два типа считаются совместимыми, если:

- оба есть один и тот же тип;
- оба вещественные;
- оба целые;
- один тип есть тип-диапазон второго типа;
- оба являются типами-диапазонами одного и того же базового типа.

Виды операторов

1. ***Оператор присваивания*** выполняет выражение, заданное в его правой части, и присваивает результат переменной, идентификатор которой расположен в левой части.

Формат оператора:

```
идентификатор:=выражение;
```

Такое присваивание возможно лишь в следующих случаях:

- идентификатор и значение выражения принадлежат к одному и тому же типу;
- идентификатор и значение выражение являются совместимыми типами и значение выражения лежит в диапазоне возможных значений идентификатора;
- идентификатор вещественного типа – значение выражения целочисленного типа;
- идентификатор – строка, выражение – символ.

В программе данные одного типа могут преобразовываться в данные другого типа. Такое преобразование может быть явным и неявным. *Явное преобразование* типов может осуществляться двумя способами:

1. Использование функций преобразование типов.
2. Преобразование типов может осуществляться применением идентификатора (имени) стандартного типа или типа определенного пользователем, как идентификатора функции преобразования к выражению преобразуемого типа, например:

```
Var
    k:Integer;
Begin
    Byte(k); {переменная k преобразована в целочисленный тип}
End.
```

Неявное преобразование типов возможно в двух случаях:

- в выражениях, составленных из вещественных и целочисленных переменных, целочисленные переменные автоматически преобра-

зуются к вещественному типу, и все выражение в целом приобретает вещественный тип;
- одна и та же область памяти попеременно трактуется как содержащая данные то одного, то другого типа (совмещение в памяти данных разного типа).

2. Оператор безусловного перехода `Goto`. Служит для передачи управления оператору, помеченному меткой. Метка отделяется от оператора двоеточием. Оператор `Goto` применяется в случаях, когда после выполнения некоторого оператора надо выполнить не следующий по порядку, а какой-либо другой, отмеченный меткой оператор.

Формат оператора:

```
Goto метка;
```

Формат описания меток:

```
Label
  имя_метки;
```

При записи оператора `Goto` необходимо помнить следующее:

1. Метка, на которую передается управление, должна быть описана в разделе описания меток того блока процедуры, функции, основной программы, в котором эта метка используется.
2. Областью действия метки является тот блок, в котором она описана.
3. Попытка выйти за пределы блока или передать управление внутрь другого блока вызывает программное прерывание.

Пример:

```
Goto met1;
  …
met1: оператор;
```

Обычно оператор `Goto` применяется для преждевременного выхода из цикла или при обработке ошибок. В других случаях его использовать *не рекомендуется.*

3. Пустой оператор не содержит ни одного символа и не выполняет никаких действий. Он может быть использован там, где синтаксис языка требует наличие оператора, но никакие действия выполнять не нужно. Пустой оператор – это лишняя точка с запятой (;).

Пример:

```
Goto m7;
...
m7: ;
```


CRT. При вызове процедуры `TextMode` сбрасываются все ранее сделанные установки цвета и окон, экран очищается и курсор переводится в его левый верхний угол. По умолчанию устанавливается режим 2.

Процедуры управления выводом текстовой информации

`LowVideo;` – устанавливает режим минимальной яркости свечения выводимых на экран символов;

`NormVideo;` – устанавливает режим нормальной яркости свечения выводимых на экран символов;

`HighVideo;` – устанавливает режим наибольшей яркости свечения выводимых на экран символов;

`TextBackGround (Color);` – устанавливает цвет фона;

`TextColor (Color);` – устанавливает цвет выводимых символов;

Параметр `Color` это выражение целого типа, соответствующее одной из констант цветов:

Цвет	Наименование константы	Значение константы
Черный	Black	0
Синий	Blue	1
Зеленый	Green	2
Бирюзовый	Cyan	3
Красный	Red	4
Малиновый	Magenta	5
Коричневый	Brown	6
Светло-серый	LigthGray	7
Темно-серый	DarkGray	8
Светло-голубой	LigthBlue	9
Светло-зеленый	LigthGreen	10
Светло-бирюзовый	LigthCyan	11
Светло-красный	LigthRed	12
Светло-малиновый	LigthMagenta	13
Желтый	Yellow	14
Белый	White	15
Мерцание	Blink	16

Для цвета фона используются константы от 0–7. Для цвета символов – 0–15.

Процедуры работы с окнами

`Window(x1,y1,x2,y2);` – определяет на экране новое активное текстовое окно. Окно – это ограниченная область экрана, выпол-

```
      r:=M[k];
      M[k]:=M[i_min];
      M[i_min]:=r;
  end;
End;
Begin
   WriteLn ('Введите количество строк:');
   ReadLn (n);
   WriteLn ('Введите строки:');
   For i:=1 to n do
     ReadLn (St[i]);
   WriteLn ('Исходный массив строк:');
   For i:=1 to n do
     WriteLn (St[i]);
   Sortstring(St,n);
   WriteLn ('Отсортированный массив строк');
   For i:=1 to n do
     WriteLn (St[i]);
End.
```

МОДУЛЬ CRT

Стандартный модуль CRT устанавливает режим работы адаптера дисплея, организует прямой вывод в буфер экрана, регулирует яркость свечения символов и выполняет другие необходимые функции.

Установка текстовых режимов

Текстовые режимы служат для отображения символов кодовой таблицы ПЭВМ и характеризуются количеством символов в строке и строк на экране. Минимальной единицей управления служит символ. Символ строится из нескольких точек (пикселей), преобразование которых в символ происходит на аппаратном уровне. Для задания одного из возможных текстовых режимов используется процедура `TextMode`.

```
TextMode (Mode);
```

Здесь Mode – код текстового режима. В качестве значений этого выражения могут использоваться следующие константы:

0 – черно-белый режим 40x25;

1 – цветной режим 40x25;

2 – черно-белый режим 80x25;

3 – цветной режим 80x25.

Код режима, установленного с помощью вызова процедуры `TextMode`, запоминается в глобальной переменной `LastMode` модуля

4. Составной оператор представляет собой группу из произвольного числа операторов, отделенных друг от друга точкой с запятой и ограниченную операторными скобками `begin` и `end`. Составной оператор воспринимается как единое целое и может находиться в любом месте программы, где синтаксис языка допускает наличие оператора.

Формат оператора:

```
begin
   Оператор 1;
          …
   Оператор N;
end;
```

5. Условные операторы обеспечивают выполнение или невыполнение некоторого оператора, группы операторов или блока в зависимости от заданных условий. В Паскале используются два условных оператор: `If` и `Case`.

Оператор условия `If`. Может принимать одну из форм:

```
If условие then оператор_1
          else оператор_2; {полная условная конструкция}
If условие then оператор; {неполная условная конструкция}
```

В первом случае говорят о полном операторе `If`, а во втором – о неполном операторе `If`. Условие – это выражение булевского типа. Оно может быть простым или сложным. Сложные условия образуются с помощью логических операций. При записи условия могут использоваться все возможные операции отношения.

В первом случае если условие истинно, то выполняется оператор_1, если условие ложно – оператор_2. Во втором случае если условие истинно, выполняется оператор, если ложно – оператор, следующий за оператором `If`.

Оператор `If` может входить в состав другого оператора `If`. В этом случае говорят о вложенности операторов:

```
If условие then
    If условие then оператор1
               else  оператор2;
```

При вложенности операторов каждое `else` соответствует тому `then`, которое непосредственно ему предшествует.

Пример:

```
If ((A>B) and (C<D)) then
```

```
    If (Z>=X) then Writeln ('Норма')
              else Writeln ('Превышение нормы')
 else  WriteLn ('Недобор');
```

Оператор выбора `Case` является обобщением оператора `If` и позволяет сделать выбор из произвольного числа имеющихся вариантов. Он состоит из выражения, называемого селектором, и списка параметров, каждому их которых предшествует список констант выбора (список может состоять и из одной константы). Как и в операторе `If`, здесь может присутствовать слово `Else`, имеющее тот же смысл. Формат оператора:

```
Case выражение-селектор of
Список_1: оператор_1;
Список_2: оператор_2;
   ...
Список_N: оператор_N
else оператор
end;
```

Оператор `Case` передает управление тому оператору, с одним из значений списка которого совпало значение выражения-селектора. Если ни одно из значений списков не совпадает со значением селектора, то либо такой оператор `Case` эквивалентен пустому оператору и затем выполняется оператор, следующий за словом `End`, либо выполняется оператор, следующий за словом `Else`.

Выражение-селектор может иметь любой скалярный тип. Список констант выбора состоит из произвольного числа значений или диапазонов, отделенных друг от друга запятыми.

Примеры:

```
Case X of      {выражение-селектор целого типа}
1,2,3  : A:=B+C;
   4   : A:=B-C;
5..9   : A:=B*C
else   A:=B/C
end;

Case CH of {выражение-селектор литерного типа}
'A'..'z': WriteLn ('Введена латинская буква');
'0'..'9': WriteLn ('Введена цифра')
end;
```

6. Операторы циклов. *Цикл* – это последовательность операторов, которая может выполняться более одного раза. Если количество


```
end;
Begin
  Insort (1,n,inm);
End;
Begin {основная программа}
  WriteLn ('Введите количество строк');
  ReadLn (n);
  WriteLn ('Введите строки');
  For i:=1 to n do
    ReadLn (st[i]);
  WriteLn ('Исходный массив строк:');
  For i:=1 to n do
    WriteLn (st[i]);
  Sortstring(st,n);
  WriteLn ('Отсортированный массив строк');
  For i:=1 to n do
    WriteLn (st[i]);
End.
```

7. Дан массив строк. Отсортировать его по длине строк.

```
Program sortd;
Type
  indata=String[80];
  mas=array[1..80] of indata;
Var
  St   :mas;
  i,n  :integer;
Procedure Sortstring(var M:mas; n:integer);
Var
  min,i_min,k,i,j  :integer;
  r                :indata;
Begin
  For k:=1 to n-1 do
  begin
     min:=Length(M[k]);
     i_min:=k;
     For i:=k to n do
        If Length(M[i])<min then
        begin
           min:=Length(M[i]);
           i_min:=i;
        end;
```

```
        begin
          z:=A[sl];
          A[sl]:=A[sl+1];
          A[sl+1]:=z;
          break;
        end;
        If A[sl][i]<A[sl+1][i] then break;
      end;
    end;
  WriteLn ('Упорядоченный список:');
  For i:=1 to k do Write(A[i],' ');
  Repeat Until Keypressed;
End.
```

6. Дан массив строк. Отсортировать его в алфавитном порядке.

```
Program Sorta;
  Type
    indata=String[80];
    mas=array[1..80] of indata;
  Var
    st  :mas;
    i,n :integer;
Procedure Sortstring(var inm:mas; n:Integer);
Procedure Insort(L,R:Integer; var M:mas);
Var
  a,b:indata;
  i,j:integer;
Begin
  i:=L;
  j:=R;
  a:=M[(l+r)div 2];
  Repeat
    While M[i]<a do i:=i+1;
    While A<m[j] do j:=j-1;
    If i<=j then
    begin
      b:=M[i]; M[i]:=M[j];
         M[j]:=b; i:=i+1; j:=j-1;
      end;
   Until i>j;
   If l<j then Insort(l,j,m);
   If l<r then Insort(i,r,m);
```


повторов известно заранее, используется оператор `For`, если количество повторов неизвестно, применяются операторы `Repeat` и `While`.

Оператор повтора `For` состоит из заголовка и тела цикла. Он может быть представлен в двух форматах:

For параметр_цикла:=начальное_значение **to** конечное_значение **do**
оператор;

For параметр_цикла:=начальное_значение **downto** конечное_значение **do**
оператор;

Тело цикла может быть простым или составным оператором. Оператор `For` обеспечивает выполнение тела цикла до тех пор, пока не будут перебраны все значения параметра цикла от начального до конечного.

Параметр цикла, его начальное и конечное значения должны принадлежать к одному и тому же типу данных. При этом допустим любой скалярный тип, кроме вещественного. В цикле `For…to` значение параметра цикла увеличивается на единицу, а в цикле `For…downto` значение параметра цикла уменьшается на единицу.

Пример. Найти сумму чисел в диапазоне от 0 до 100.

```
Program DemoFor;
    Var
        I, Sum: Integer;
Begin
    Sum:=0;
    For i:=0 to 100 do
        Sum:=Sum+I;
   WriteLn ('Сумма чисел равна ',Sum);
End.
```

Оператор повтора repeat состоит из заголовка (`Repeat`), тела и условия окончания (`Until`).

Формат оператора:

Repeat
оператор_1;
...
оператор_n;
Until условие;

Условие – выражение логического типа. Операторы, заключенные между словами `Repeat` и `Until`, являются телом цикла. Вначале выполняется тело цикла, затем проверяется условие выхода из цикла. Ес-

ли результат равен False, тело цикла активизируется еще раз, если результат True – происходит выход из цикла.

Оператор Repeat имеет три характерные особенности:

- выполняется по крайней мере один раз;
- тело цикла выполняется пока условие равно False;
- в теле может находиться произвольно число операторов без операторных скобок Begin...End.

Пример. Найти сумму нечетных чисел в диапазоне от 0 до 100.

```
Program DemoRepeat;
    Var
        I, Sum: Integer;
Begin
    I:=3;
   Sum:=3;
    Repeat
        Sum:=Sum+I;
        I:=I+2;
    Until (I>100);
   WriteLn ('Сумма нечетных чисел равна ',Sum);
End.
```

Оператор While аналогичен оператору Repeat, но проверка условия выполнения тела цикла производится в самом начале цикла.

Формат оператора:

While условие **do** оператор;

Условие – выражение логического типа, тело цикла – простой или составной оператор. Перед каждым выполнением оператора вычисляется значение выражения условия. Если результат условия равен True, оператор выполняется и снова вычисляется выражение условия. Если результат равен False, происходит выход из цикла и переход к первому после While оператору. Если перед первым выполнением цикла значение выражения было False, оператор вообще не выполняется и происходит переход на следующий оператор.

Операторы For, While, Repeat могут быть вложенными, т.е. в теле цикла может быть другой оператор цикла.

Пример. Найти сумму четных чисел в диапазоне от 0 до 100.

```
Program DemoWhile;
   Var
      I, Sum: Integer;
Begin
```



```
   n,k     :Integer;
Begin
  WriteLn('Введите исходный текст: ');
  ReadLn (txt);
  WriteLn ('Введите удаляемый текст: ');
  ReadLn (str);
  n:=Length(str);
  Repeat
    k:=Pos(str,txt);
    If k<>0 then
      Delete (txt,k,n);
  Until k=0;
  WriteLn('Полученный текст: ');
  WriteLn (txt);
End.
```

5. Дана строка, содержащая список фамилий, разделенных одним пробелом. Отсортировать его в алфавитном порядке.

```
Program alf;
Var
  st,z           :String;
  A              :array[1..10] of String[10];
  i,k,j,sl,ds    :Byte;
Begin
  k:=1;
  WriteLn('Введи список фамилий: ');
  ReadLn(st);
  {получение из строки массива слов}
  For i:=1 to Length(st)-1 do
    If (st[i]=' ') then inc(k)
                   else A[k]:=A[k]+st[i];
  writeln;
  {сортировка массива слов}
  For j:=1 to k-1 do
    For sl:=1 to k-j do
    begin
      If Length(A[sl])<Length(A[sl+1]) then
        ds:=Length(A[sl+1])
      else ds:=Length(A[sl]);
      For i:=1 to ds do
      begin
        If A[sl][i]>A[sl+1][i] then
```

```
      If (fl<>0) then
                 begin
                   insert (str1,txt,j);
                   n:=n+n1;
                 end;
   m1:end;
  WriteLn ('Полученный текст:');
  WriteLn (txt);
End.
```

3. Дан текст Txt. Заменить любое вхождение строки St1 на строку St2.

```
Program zam;  {замена}
  Label m1;
  Var
    d2,k              : Integer;
    s,str1, str2,txt  : String;
Begin
  WriteLn('Введите исходный текст:');
  ReadLn (txt);
  WriteLn ('Введите замещающий текст:');
  ReadLn (str1);
  WriteLn ('Введите текст, вместо которого
нужна замена:');
  ReadLn (str2);
  d2:=Length(st2);
  Repeat
    k:=Pos(str2,txt);
    If k<>0 then
    begin
      Delete (txt,k,d2);
      Insert(st1,txt,k);
    end;
  Until k=0;
  WriteLn('Полученный текст: ');
  WriteLn (txt);
End.
```

4. Дан текст Txt. Удалить строку St из текста.

```
Program Udal;  {удаление}
Var
   str,txt :String;
```



```
   I:=2;
   Sum:=0;
   While (i<=100) do
   begin
      Sum:=Sum+I;
      I:=I+2;
   end;
   WriteLn ('Сумма четных чисел равна ',Sum);
End.
```

7. Операторы управления работой циклов. Для управления работой цикла используются специальные процедуры-операторы `Continue`, `Break`.

Вызов процедуры `Continue` в теле цикла прерывает выполнение цикла и переходит к проверке условия, минуя все операторы, расположенные ниже `Continue`.

```
Repeat
  оператор;
  …
  Continue;
  …
  оператор;
Until условие;
```

```
While условие do
begin
  оператор;
  …
  Continue;
  …
  оператор;
end;
```

```
For i:=1 to N do
begin
  оператор;
  …
  Continue;
  …
  оператор;
end;
```

Вызов процедуры `Break` прерывает выполнение цикла и передает управление первому оператору, расположенному за оператором цикла.

```
Repeat
  оператор;
  …
  Break;
  …
  оператор;
Until условие;
оператор;
```

```
While условие do
begin
  оператор;
  …
  Break;
  …
  оператор;
end;
оператор;
```

```
For i:=1 to N do
begin
  оператор;
  …
  Break;
  …
  оператор;
end;
оператор;
```

8. Оператор вызова процедуры.

ПРОЦЕДУРЫ ВВОДА-ВЫВОДА

Для выполнения операций ввода-вывода служат 4 процедуры: `Read, ReadLn, Write, WriteLn.`

Процедура чтения `Read` обеспечивает ввод числовых данных, символов, строк и т.д. для последующей их обработки программой. Формат:

```
Read (X1,X2,…,Xn);
```

где `X1, X2, …, Xn` – переменные допустимых типов.

Значения `X1,…,Xn` набираются минимум через один пробел на клавиатуре пользователем и высвечиваются на экране. После ввода данных для одной процедуры `Read` нажимается клавиша ввода Enter. Значения переменных должны вводиться в строгом соответствии с синтаксисом языка.

Если в программе имеется несколько процедур `Read`, данные для них вводятся потоком, т.е. после считывания значений переменных для одной процедуры `Read` данные для следующей процедуры `Read` набираются в той же строке, что и для предыдущей до окончания строки, затем происходит переход на следующую строку.

Пример.

```
Var
    A,B: Integer;
    C,D: Real;
Begin
    Read (A,C);
    Read (B,D);
End.
```

Поток ввода:

```
25 2.34<Enter> 451 25.685<Enter>
```

Процедура чтения `ReadLn` аналогична процедуре `Read`, единственное отличие заключается в том, что после считывания последнего в списке значения для одной процедуры `ReadLn` данные для следующей процедуры `ReadLn` будут считываться с начала новой строки. Заменим в предыдущем примере процедуры `Read` на процедуры `ReadLn`:

```
Var
    A,B: Integer;
    C,D: Real;
Begin
    ReadLn (A,C);
    ReadLn (B,D);
End.
```

Поток ввода:

```
25 2.34 <Enter>
451  25.685 <Enter>
```



```
            st1:=Copy(st,ns,ds);
          end;
          ns:=i+1;
        end;
    WriteLn ('Слово максимальной длины: ',st1);
End.
```

2. Дан текст Txt. Вставить строку St1 в текст Txt после строки St2.

```
Program Vstavka;  {вставка}
Label m1;
Var
   i,j,n,n1,n2,k,fl :Integer;
   str1, str2,txt   :String;
Begin
  WriteLn ('Введите исходный текст:');
  ReadLn (txt);
  WriteLn ('Введите вставляемый текст:');
  ReadLn (str1);
  WriteLn ('Введите текст, после которого нуж-
на вставка:');
  ReadLn (str2);
  n:=Length (txt);
  n1:=Length(str1);
  n2:=Length (str2);
  For i:=1 to n do
  begin
    fl:=0;
    If (txt[i]=str2[1]) then
    begin
      j:=i;
      for k:=1 to n2 do
      begin
        if (txt[j]=str2[k]) then fl:=1
                            else
                            begin
                              fl:=0;
                              goto m1;
                            end;
                        j:=j+1;
              end;
      end;
```

Значение St	Выражение	Результат
'ABCDEFG'	`Copy (St, 2, 3)`	'BCD'

`Concat (Str1, Str2, ..., StrN)` – выполняет сцепление строк Str1, Str2, ..., StrN в том порядке, в каком они указаны в списке параметров.

Выражение	Результат
Concat ('AA', 'XX', 'Y')	'AAXXY'

`Length (St)` – вычисляет длину в символах строки St.

Значение St	Выражение	Результат
'123456789'	`Length (St)`	9

`Pos (Str1, Str2)` – обнаруживает первое появление в строке Str2 подстроки Str1. Результат имеет целочисленный тип и равен номеру той позиции, где находится первый символ подстроки Str1. Если в Str2 подстроки Str1 не найдено, результат равен 0.

Значение Str1	Выражение	Результат
'abcdef'	`Pos('de', Str1)`	4
'abcdef'	`Pos('r', Str1)`	0

Примеры программ обработки строк

1. Дана строка. Слова в строке разделены одним пробелом, в конце строки точка. Распечатать слово максимальной длины.

```
Program Max_Length;
Var
   St,St1:String;
   n,i,ns,ds,d,d_max:Integer;
Begin
   WriteLn ('Введите строку');
   ReadLn(St);
   d:=Length(st);
   ns:=1;
   d_max:=0;
   For i:=1 to d do
      If ((St[i]=' ') or (St[i]='.')) then
      begin
         ds:=i-ns;
         If ds>d_max then
         begin
            d_max:=ds;
```


Процедура записи `Write` производит вывод числовых данных, символов, строк и булевских значений. Формат:

```
Write (Y1,…,Yn);
```

где `Y1,…,Yn` – выражения целочисленного, вещественного, символьного, строкового, булевского и др. типов.

Форматы вывода:

Процедура	Значение	Поток вывода
`Write (I);`	`134`	`134`
`Write (I:5);`	`134`	`__134`

(Для вывода целого числа отводится 5 позиций. При выводе число автоматически прижимается к правому краю).

`Write (R);`	`715.342`	`__7.153420000E+02`
`Write (R:12);`	`46.78`	`4.678000E+01`

(На вещественное число отводится 12 позиций).

`Write (R:6:2);`	`46.78`	`_46.78`

(На вещественное число отводится 6 позиций, из них 2 знака после запятой. Число прижимается к правому краю).

`Write (CH);`	`'X'`	`X`
`Write (CH:4);`	`'X'`	`___X`
`Write (S);`	`'Привет !'`	`Привет !`
`Write (S:10);`	`'Привет !'`	`___Привет !`
`Write (B);`	`True`	`True`
`Write (B:6);`	`True`	`__True`

Процедура записи `WriteLn` аналогична процедуре `Write`, но после вывода последнего в списке значения для текущей процедуры `WriteLn` происходит перевод курсора к началу следующей строки. Процедура `WriteLn`, записанная без параметров, вызывает перевод строки.

Пример:

```
   Write ('A=',A:3);
   WriteLn ('_C=',C:3);
   WriteLn ('D=',D:6:2);
```

Поток вывода

```
A=_23_C=_34
D=__3.24
```

Пример. Вычислить значения функций f и g:

$$f=\begin{cases}\dfrac{1}{\sqrt{x^2+y^2}}, & x+y<0,\\ 2xy, & 0\le x+y\le 5,\\ \dfrac{1}{x+y}, & x+y>5,\end{cases}\quad g=\begin{cases}x-y, & x+y<0,\\ 2x+3, & 0\le x+y\le 5,\\ \dfrac{1}{\sqrt{x^2+1}}, & x+y>5.\end{cases}$$

Написать две программы, использующие:

- вещественные значения;
- целочисленную арифметику (целочисленные данные, промежуточные и конечные результаты, операции целочисленной арифметики, константу MAXINT, диапазоны оператора выбора Case).

```
Program example1;
{Программирование ветвящихся вычислительных процессов}
{использование вещественных значений                  }
  Uses Crt;
  Var
    x,y,f,g:Real;
Begin
  Clrscr; {Очистка экрана}
  WriteLn('Введите значения x,y');
  ReadLn (x,y);
  WriteLn ('При x=',x:5:2,'  y=',y:5:2);
  If x+y<0 then
           begin
             WriteLn ('x+y=',x+y:5:2,'<0');
             f:=1/sqrt(x*x+y*y);
             g:=x-y;
           end
           else
           If x+y<=5 then
                     begin
            WriteLn ('0<=x+y=',x+y:5:2,'<=5');
                       f:=2*x*y;
                       g:=2*x+3;
                     end
                     else
                     begin
             WriteLn ('x+y=',x+y:5:2,'>5');
                       f:=1/(x+y);
```


Процедуры обработки строк

`Delete (St, Poz, N)` – удаление N символов строки St, начиная с позиции Poz.

Значение St	Выражение	Результат
'абвгде'	`Delete (Str,4,2)`	'абве'

`Insert (Str1, Str2, Poz)` – вставка строки Str1 в строку Str2, начиная с позиции Poz.

Пример:

```
Var
   S1,S2,S3 : String[11];
   ...
   S1:=' Pentium ';
   S2:='V';
   S3:=Insert (S1, S2, 10);
```

В результате выполнения последнего выражения значение строки S3 станет равным 'Pentium V'.

`Str (I, St)` – преобразование числового значения величины I (целого или вещественного типа) и помещение результата в строку St. После I может записываться формат, аналогичный формату вывода. Если в формате указано недостаточное для вывода количество разрядов, поле вывода расширяется автоматически до нужной длины.

Значение I	Выражение	Результат
1500	`Str (I:6, St)`	' 1500'

`Val (St, I, Cod)` – преобразует значение St в величину целочисленного или вещественного типа и помещает результат в I. Значение St не должно содержать незначащих пробелов в начале и в конце. Cod — целочисленная переменная. Если во время операции преобразования ошибки не обнаружено, значение Cod равно нулю, если ошибка обнаружена (например, литерное значение переводится в цифровое), Cod будет содержать номер позиции первого ошибочного символа, а значение I не определено.

Значение St	выражение	Результат	
'1450'	`Val (St, I, Cod)`	1450	Cod=0

Функции

`Copy (St, Poz, N)` – выделяет из строки St подстроку длиной N символов, начиная с позиции Poz.

```
Const
    St:String='Сегодня хорошая погода !';
```

Строковые выражения

Выражения, в которых операндами служат строковые данные, называются *строковыми выражениями.* Они состоят из строковых констант, переменных, указателей функций и знаков операций. Над строковыми данными допустимы операция сцепления и операции отношения.

Операция сцепления (+) применяется для сцепления нескольких строк в одну результирующую строку.

Пример:

Выражение	Результат
'E'+'C'+' 18' +'40'	'EC 1840'

Длина результирующей строки не должна превышать 255.

Операции отношения (=, <>, <, >, >=, <=) проводят сравнение двух строковых операндов и имеют приоритет более низкий, чем операция сцепления, т.е. вначале всегда выполняются все операции сцепления, если они присутствуют, и лишь потом реализуются операции отношения. Сравнение строк производится слева направо до первого несовпадающего символа, и та строка считается больше, в которой первый несовпадающий символ имеет больший номер в стандартной таблице обмена информацией. Результат выполнения операций отношения над строковыми операндами всегда имеет булевский тип и принимает значение True, если выражение истинно, и False, если выражение ложно.

Пример:

Выражение	Результат
'COSM1'<'COSM2'	True

Если строки имеют различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная. Строки считаются равными, если они полностью совпадают по длине и содержат одни и те же символы.

Для присваивания строковой переменной результата строкового выражения используется оператор присваивания (:=).

Допускается смешение в одном выражении операндов строкового и литерного типа. Если при этом литерной переменной присваивается значение строкового типа, длина строки должна быть равна единице, иначе возникает ошибка выполнения.

Для обработки строковых данных используются стандартные процедуры и функции.


```
                    g:=1/sqrt(x*x+1);
                  end;
  WriteLn ('f=',f:6:4);
  WriteLn ('g=',g:6:4);
  ReadLn; {Задержка экрана}
End.

Program example2;
{Программирование ветвящихся вычислительных процессов}
{использование целочисленной арифметики              }
  Uses Crt;
  Var
    x,y,f,g:Integer;
Begin
  Clrscr; {Очистка экрана}
  WriteLn('Введите значения x,y');
  ReadLn (x,y);
  WriteLn ('При x=',x,'  y=',y);
  Case x+y of
  -Maxint..-1: begin
                 WriteLn ('x+y=',x+y,'<0');
                f:=1 div round(sqrt(x*x+y*y));
                g:=x-y;
               end;
   0..5      : begin
                WriteLn ('0<=x+y=',x+y,'<=5');
                f:=2*x*y;
                g:=2*x+3;
               end;
   6..Maxint : begin
                 WriteLn ('x+y=',x+y,'>5');
                 f:=1 div (x+y);
                 g:=1 div round (sqrt(x*x+1));
               end
  end;
  WriteLn ('f=',f);
  WriteLn ('g=',g);
  ReadLn; {Задержка экрана}
End.
```

МАССИВЫ. ЗАДАЧИ КОМБИНИРОВАННОЙ ОБРАБОТКИ МАССИВОВ

Массив — это структурированный тип данных, состоящий из фиксированного числа элементов одного типа. Тип элементов массива называется *базовым*. Число элементов массива фиксируется при описании и в процессе выполнения программы не меняется.

Доступ к элементу массива реализуется указателем имени массива и в квадратных скобках индекса. Индексы элементов массива это выражения любого скалярного типа кроме вещественного.

Определить массивы можно двумя способами:

1. **Var**
   имя_массива : **array** [тип_индексов] **of** тип_элементов;
2. **Type**
   имя_типа = **array** [тип_индекса] **of** тип_элементов;

Var
   имя_массива : имя_типа;

Тип индекса (это тип-диапазон) определяет границы изменения значений индекса. Если задан один индекс, то массив называется одномерным, если два – двумерным, если n – n-мерным. Одномерные массивы используются для представления векторов, двумерные – для представления матриц.

Пример.

1-ый способ:

```
Var
A,B: array [1..10] of Real; {одномерные массивы}
C: array[1..5,1..10] of Integer; {двумерный массив}
```

2-ой способ:

```
Type
    Mas1=array [1..10] of Real;
    Mas2=array [1..5,1..10] of Integer;
Var
    A,B:Mas1;
    C:Mas2;
```

Диапазоны индексов можно задать константами, которые описаны в разделе описания констант:

```
Const
    N=5;
    M=10;
Var
    C: array [1..N,1..M] of Integer;
```



```
End; {MyFunc}
Var
   i:Integer;
begin
   i:=1;
   i:=2*MyFunc(i)-100; {Стандартный вызов функции}
   MyFunc(i)           {Расширенный вызов функции}
end.
```

С помощью расширенного синтаксиса нельзя вызывать стандартные функции. Компиляция с учетом расширенного синтаксиса включается активным состоянием опции EXTENDED SYNTAX диалогового окна OPTION/COMPILER или глобальной директивой компилятора `{$X+}`.

ОБРАБОТКА СИМВОЛЬНОЙ ИНФОРМАЦИИ

Строка – это последовательность символов. При использовании в выражениях строка обязательно заключается в апострофы. Количество символов в строке (длина строки) может динамически изменяться от 0 до 255. Определение строкового типа устанавливает максимальное количество символов, которое может содержать строка.

Формат описания строк:

1-ый способ:

```
Type
   имя_типа = String [максимальная_длина_строки];
Var
   идентификатор : имя_типа;
```

2-ой способ:

```
Var
   идентификатор : String [максимальная_длина_строки];
```

Длина строки может не указываться, в этом случае принимается максимально возможная длина строки, равная 255 символов.

Пример:

```
Type
   Stroka:String[50];
Var
   St:Stroka;
   St1:String;
   St2:String[25];
```

Строку можно описать с помощью типизированной константы:

При написании рекурсивных подпрограмм необходимо обращать внимание на выход из подпрограммы в нужный момент, так как возможен выход значений из допустимого диапазона. Часто встречается и другой вариант рекурсии, когда первая подпрограмма вызывает вторую, которая в момент вызова еще не определена. Такая ситуация называется *косвенной рекурсией.* Для реализации косвенной рекурсии используется так называемое предварительное описание процедур и функций.

Предварительное описание подпрограмм

Для реализации алгоритмов с косвенной рекурсией в языке Паскаль предусмотрена специальная директива предварительного описания подпрограмм `Forward`.

Предварительное описание состоит из заголовка процедуры и следующего за ним зарезервированного слова `Forward`. Позже процедура или функция описываются без повторения списка формальных параметров, атрибутов или возвращаемых типов.

```
Program DF;
Procedure P2 (формальные параметры); Forward;
Procedure P1;
Begin
   P2(фактические параметры);
End;
Procedure P2; {список формальных параметров не повторяется}
Begin
   P1;
End;
Begin {основная программа}
   P1
End.
```

Расширенный синтаксис вызова функции

В языке программирования Pascal есть возможность вызывать функцию и не использовать то значение, которое оно возвращает, т.е. вызов функции может внешне выглядеть как вызов процедуры, например:

```
{$X+} {Включаем расширенный синтаксис}
Function MyFunc(var x:Integer):Integer;
Begin
   If x<0 Then x:=0
          Else MyFunc:=x+10;
```


Массив можно описать с помощью типизированных констант:

```
Const
Vect:array[1..5] of Byte=(1,6,3,8,5);
Matr:array[1..4,1..6] of Integer=
       ((1,6,3,5,2,4),
       (7,2,5,4,3,2),
       (3,1,6,3,8,5),
       (5,2,8,5,5,4));
```

Элементы массива располагаются в памяти последовательно. Многомерные массивы располагаются таким образом, что самый правый индекс возрастает самым первым. Например, массив A[3,3] будем располагаться следующим образом: A[1,1], A[1,2], A[1,3], A[2,1], A[2,2], A[2,3], A[3,1], A[3,2], A[3,3].

Действия над массивами

Для работы с массивом как единым целым, используется идентификатор массива без указания индекса в квадратных скобках. Массивы, участвующие в этих действиях должны иметь одинаковые типы индексов и одинаковые типы компонент. Над массивом как единым целым можно произвести следующие действия:

1. A=B (проверить массивы на равенство);
2. A<>B (проверить массивы на неравенство);
3. A:=B.

Действия над элементами массива

1. Инициализация массива (заключается в присвоении каждому элементу массива одного и того же значения)

- одномерного

```
For i:=1 to N do
    A[i]:=0;
```

- двумерного

```
For i:=1 to N do
   For j:=1 to M do
      B[i,j]:=0;
```

2. Ввод элементов массива

- одномерного

```
Write ('Введите размерность массива N=');
ReadLn (N);
WriteLn ('Введите элементы массива');
For i:=1 to N do
```

```
    begin
          Write ('A[',i,']=');
          ReadLn (A[i]);
    end;
```

- двумерного

```
Write ('Введите размерность массива N, M');
ReadLn (N, M);
WriteLn ('Введите элементы массива');
For i:=1 to N do
   For j:=1 to M do
      begin
         Write ('B[',i,',',j,']=');
         ReadLn (B[i,j]);
      end;
```

3. Вывод элементов массива

- одномерного

```
WriteLn ('Вектор A:');
For i:=1 to N do
   Write (A[i]:5);
WriteLn;
```

- двумерного

```
WriteLn ('Матрица B:');
For i:=1 to N do
begin
   For j:=1 to M do
      Write (B[i,j]:5);
   WriteLn;
end;
```

4. Поиск нулевых элементов в массиве

- одномерном

```
k:=0;
For i:=1 to N do
   If A[i]=0 then k:=k+1;
```

- двумерном

```
k:=0;
For i:=1 to N do
   For j:=1 to M do
   If B[i,j]=0 then k:=k+1;
```


Область действия идентификаторов

Объекты, описанные в основной программе являются *глобальными* и могут использоваться во всех вложенных блоках. Вложенные блоки – это процедуры и функции. Описанные в них объекты являются *локальными* и недоступны во внешних блоках.

Правила использования идентификаторов:

- каждый идентификатор должен быть описан перед тем, как он будет использован;
- областью действия идентификатора является тот блок, в котором он описан;
- все идентификаторы в блоке должны быть уникальными, т.е. не повторяться;
- один и тот же идентификатор может быть по-разному определен в каждом отдельном блоке;
- если идентификатор подпрограммы пользователя совпадает с именем стандартной процедуры или функции, то последние недоступны в пределах области действия подпрограммы, объявленной пользователем, т.е. стандартная функция игнорируется, а выполняется подпрограмма пользователя.

Рекурсивные подпрограммы

Во многих случаях оптимизация алгоритма решения задачи требует вызова для выполнения подпрограммы из раздела операторов той же самой подпрограммы, т.е. подпрограмма вызывает саму себя. Такой способ вызова называется *рекурсией.* Рекурсия полезна прежде всего в случаях, когда основную задачу можно разделить на подзадачи, имеющие ту же структуру, что и первоначальная задача. Подпрограммы, реализующие рекурсию, называются *рекурсивными*.

Классический пример использования рекурсии – вычисление факториала целого положительного числа.

```
Program Demorec;
   Var N: Integer;
Function Fact (k:Integer):Integer;
Begin
   If k=0 then Fact:=1
          else Fact:=k*Fact(k-1);
End;
Begin
   Write ('Введите число ->');  Readln (N);
   Writeln ('N!=',Fact(N));
End.
```

```
    Begin
        p:=f(x)+f(2*x);
        Writeln ('Значение функции =',p:8:3);
    End;
Begin {основная программа}
    Clrscr;
    Fn(3,sin1);
    Fn(5,cos1);
End.
```

В программе могут быть объявлены переменные процедурных типов:

```
    Var
        P1:Proc1;
        F1:Func1;
```

Переменным процедурных типов допускается присваивать в качестве значений имена соответствующих подпрограмм. После такого присваивания имя переменной становится синонимом имени подпрограммы, например:

```
Program DemoProc1;
    Type
        Func=Function (x:Real):Real;
    Var
        F1:Func;
    Function sin1(x:Real):Real; far;
    Begin
        Sin1:=sin(x);
    End;
    Procedure Fn(x:Real; f:Func);
        Var
            p:Real;
    Begin
        p:=f(x)+f(2*x);
        Writeln ('Значение функции =',p:8:3);
    End;
Begin {основная программа}
    Clrscr;
    F1:=Sin1;
    Fn(3,F1);
End.
```


5. Нахождение минимального элемента массива и его места

- одномерного

```
min:=A[1];
i_min:=1;
For i:=1 to N do
   If A[i]<min then
                  begin
                    min:=A[i];
                    i_min:=i;
                  end;
```

- двумерного

```
min:=B[1,1];
i_min:=1;
j_min:=1;
For i:=1 to N do
   For j:=1 to M do
      If B[i,j]<min then
                     begin
                        min:=B[i,j];
                        i_min:=i;
                        j_min:=j;
                     end;
```

6. Перестановка минимального и первого элементов в массиве

- одномерном

```
r:=A[1];
A[1]:=A[i_min];
A[i_min]:=r;
```

- двумерном

```
r:=B[1,1];
B[1,1]:=B[i_min,j_min];
B[i_min,j_min]:=r;
```

7. Нахождение суммы положительных элементов массива

- одномерного

```
sum:=0;
For i:=1 to N do
   If A[i]>0 then sum:=sum+A[i];
```

- двумерного

```
sum:=0;
For i:=1 to N do
   For j:=1 to M do
```

```
      If B[i,j]>0 then sum:=sum+B[i,j];
```

8. Нахождение произведения нечетных элементов

- одномерного

```
prod:=1;
For i:=1 to N do
   If (A[i] mod 2) <> 0 then
prod:=prod*A[i];
```

- двумерного (нахождение произведения нечетных элементов)

```
prod:=1;
For i:=1 to N do   {Функция Odd(X) возвращает значение}
   For j:=1 to M do    {истина, если X нечетно          }
      If Odd(B[i,j]) then
        prod:=prod*B[i,j];
```

9. Нахождение суммы положительных элементов выше главной диагонали (включая элементы диагоналей).

Элементы на главной диагонали характеризуются тем, что индексы этих элементов равны, т.е. *i=j*. Для элементов побочной диагонали для любого *i* индекс столбца *j=n-i+1*. Элементы областей выше, ниже главной или побочной диагоналей можно задать или порядком изменения индексов или условиями, накладываемыми на индексы:


i=1,2,...,n; j=1,2,...,i
или *i>=j*


i=1,2,...,n; j=i,i+1,...,n
или *i<=j*


i=1,2,...,n; j=1,2,... n-i+1;
или *n-i-j+1>=0*


i=1,2,...,n;j=n-i+1,...,n
или *n-i-j+1<=0*

```
sum:=0;
For i:=1 to n do
   For j:=i to n do
      If B[i,j]>=0 then sum:=sum+B[i,j];
```

10. Поменять местами максимальный элемент на главной диагонали и минимальный элемент ниже побочной.

```
Max:=B[1,1];
I_max:=1;
For i:=1 to n do
   If B[i,i]> max then
   begin
```



```
Function St (s:InType):OutType;
```

Для передачи строк *произвольной длины* можно установить режим компиляции, при котором отключается контроль за совпадением длины фактического и формального параметра–строки. При передаче строки меньшего размера формальный параметр будет иметь ту же длину, что и параметр обращения; передача строки большего размера приведет к её усечению до максимального размера формального параметра. Контроль включается только при передаче строки, объявленной как формальный параметр–переменная. Если соответствующий параметр объявлен параметром–значением, эта опция игнорируется и длина не контролируется.

Процедурные типы. Параметры–функции и параметры–процедуры. *Процедурные типы* позволяют передавать имена процедур и функций в подпрограммы в качестве параметров. Для объявления процедурного типа используется заголовок процедуры (функции), в котором отсутствует ее имя:

```
Type
   Proc1=Procedure (a,b,c:Real; var d:Real);
   Proc2=Procedure;
   Func1=Function (var a:Integer):real;
```

Существуют два процедурных типа: тип-процедура и тип-функция. При передачи имен процедур и функций в качестве параметров, эти подпрограммы должны быть откомпилированы с расчетом на дальнюю модель памяти, т.е. сразу за заголовком процедуры или функции должна быть использована директива компилятора Far.

Пример. Вычислить значения функций: $f_1=\sin(x)+\sin(2x)$; $f_2=\cos(x)+\cos(2x)$.

```
Program DemoProc;
   Type
      Func=Function (x:Real):Real;
   Function sin1(x:Real):Real; far;
   Begin
      Sin1:=sin(x);
   End;
   Function cos1(x:Real):Real; far;
   Begin
      Cos1:=cos(x);
   End;
   Procedure Fn(x:Real; f:Func);
      Var
         p:Real;
```

```
Type
    MasType=array [1..10] of Integer;
Procedure S (mas:MasType);
```

При передаче подпрограмме массивов *переменной* длины используются так называемые *открытые массивы*. Открытый массив представляет собой формальный параметр подпрограммы, описывающий базовый тип элементов массива, но не определяющий его размерности и границы:

```
Procedure MyProc(OpenArray:array of Integer);
```

Внутри подпрограммы такой параметр трактуется как одномерный массив с нулевой нижней границей. Верхняя граница открытого массива возвращается функцией `High`. Используя нуль как минимальный индекс и значение, возвращаемое функцией `High`, как максимальный индекс, подпрограмма может обрабатывать одномерные массивы произвольной длины.

Пример. Программа вывода на экран содержимого двух одномерных массивов различной длины.

```
Program DemoMass;
   Const
   A: array[-1..2] of Integer=(0,1,2,3);
   B: array[5..7] of Integer=(4,5,6);
{Процедура обработки открытого массива}
Procedure ArrayPrint(Mas:array of Integer);
   Var
       i:Integer;
Begin
   For i:=0 to High(Mas) do
      Write(Mas[i]:8);
   Writeln;
End;
Begin {основная программа}
   ArrayPrint(A);
   ArrayPrint(B);
End.
```

Параметры–строки. Поскольку строка *фиксированной длины* является фактически своеобразным массивом, её передача в подпрограмму осуществляется аналогичным образом:

Пример:

```
Type
    InType=String[15];
    OutType=String[30];
```



```
      Max:=B[i,i];
      I_max:=i;
   end;
Min:=B[1,n];
I_min:=1;
J_min:=n;
For i:=1 to n do
   For j:=n-i+1 to n do
      If B[i,j]< Min then
      begin
         Min:=B[i,j];
         I_min:=i;
         J_min:=j;
      end;
R:=B[I_max,I_max];
B[I_max,I_max]:=B[I_min,J_min];
B[I_min,J_min]:=R;
```

11. Дана квадратная матрица B размерности nxn. Построить вектор A, где a_i – сумма положительных элементов i-ой строки матрицы.

```
For i:=1 to n do
Begin
   S:=0;
   For j:=1 to n do
      If B[i,j]>0 then S:=S+B[i,j];
   A[i]:=S;
End;
```

12. Дана квадратная матрица B размерности nxn. В каждом столбце оставить без изменения максимальный элемент столбца, остальные элементы заменить нулями.

```
For j:=1 to n do
begin
   Max:=B[1,j];
   I_max:=1;
   For i:=1 to n do
      If B[i,j]> Max then
      begin
         Max:=B[i,j]
         I_max:=i;
      end;
```

```
      For i:=1 to n do
        If i_max<>i then B[i,j]:=0;
    end;
```

Пример: Дан целочисленный вектор $A(n)$, поменять местами максимальный и минимальный элементы вектора. На печать выдавать исходный вектор, максимальный, минимальный элементы, полученный вектор.

```
Program Example_Vect;
  Uses Crt;
  Const
    N_max=10;
  Var
    N,i,max,i_max,min,i_min,r:Integer;
    A:array [1..N_max] of Integer;
Begin
  Clrscr;
  Write('Введите размерность массива N (<',N_max,')=');
  ReadLn (N);
  WriteLn ('Введите элементы массива');
  For i:=1 to N do   {ввод элементов вектора}
  begin
    Write ('A[',i,']=');
    ReadLn (A[i]);
  end;
  WriteLn ('Исходный вектор A:');
  For i:=1 to N do    { вывод вектора}
    Write (A[i]:5);
  WriteLn;
  min:=A[1];       {нахождение минимального элемента}
  i_min:=1;        {и его индекса}
  For i:=1 to N do
    If A[i]<min then
                begin
                  min:=A[i];
                  i_min:=i;
                end;
  max:=A[1];
  i_max:=1;            {нахождение максимального элемента}
  For i:=1 to N do     {и его индекса}
    If A[i]>max then
```


раметра означает изменение фактического параметра. В параметры, передаваемые по ссылке, помещаются результаты работы программы.

Третий способ описания основан на использовании *параметров–констант*, которые задаются с использованием слова `const`:

```
Procedure MyProc(const a,b,c: Integer; d: Real);
Function MyFunc(const a,b,c: Integer; d:Real):Real;
```

В этом случае используется механизм передачи параметров по ссылке, но внутри подпрограммы формальные параметры изменять нельзя. Это означает, что в случае параметра–константы в подпрограмму также передается адрес области памяти, в которой располагается переменная или вычисленное значение. Однако компилятор блокирует любые присваивания параметру–константе нового значения в теле подпрограммы. Этот способ используется для передачи в подпрограмму данных большого объема.

Возможен смешанный вариант, когда указываются и посылаемые в подпрограмму данные, и возвращаются результаты, например:

```
Procedure MyProc(a,b,c: Integer; var d: Real);
Function MyFunc(a,b,c:Integer; var d:Real):Real;
```

Нетипизированные параметры–переменные. Существует разновидность ссылочных параметров без типа. Они называются *нетипизированными* и предназначены для передачи и приёма данных любого типа. Параметр считается нетипизированным, если тип формального параметра–переменной в заголовке подпрограммы не указан, при этом соответствующий ему фактический параметр может быть переменной любого типа. Нетипизированными могут быть только параметры–переменные.

Нетипизированные параметры описываются с помощью ключевых слов `var` и `const`, при этом тип данных опускается:

```
Procedure MyProc(const y; var x);
Function MyFunc(const y; var x):Real;
```

Внутри подпрограммы тип таких параметров неизвестен, поэтому программист должен сам позаботиться о правильной интерпретации переданных данных. В остальном это обычные параметры, передаваемые по ссылке.

Параметры–массивы. Для передачи массива *фиксированной* длины в подпрограмму необходимо предварительно описать его тип.

Пример:

памяти определяет возможность вызова процедуры из различных частей программы: если используется *ближняя модель*, вызов возможен только в пределах 64 Кбайт (в пределах одного сегмента кода, который выделяется основной программе и каждому используемому в ней модулю); при *дальней модели* вызов возможен из любого сегмента.

Параметры процедур и функций

Параметры служат для передачи в процедуры и функции исходных данных и для приёма результатов. В зависимости от назначения они описываются в заголовке подпрограммы тремя основными способами:

Первый способ предполагает указание списка параметров и их типа, например:

```
Procedure MyProc (a,b,c: Integer; d: Real);
Function MyFunc (a,b,c: Integer; d: Real):Real;
```

В этом случае в процедуру и функцию передаются *значения* фактических параметров, указанных при вызове подпрограммы. Такие параметры называются *параметрами-значениями*. Формальный параметр является переменной, локальной в блоке. Если параметр определен как параметр–значение, то перед вызовом подпрограммы это значение вычисляется, полученный результат копируется во временную память и передается подпрограмме. Изменение значения формальных параметров внутри процедуры или функции не приводит к изменению фактических. Фактический параметр может быть выражением того же типа, что и соответствующий ему формальный параметр.

Второй способ. Фактические параметры передаются в процедуру или функцию по *ссылке*. В этом случае перед списком параметров располагается слово `var`, например:

```
Procedure MyProc(var a,b,c: Integer; d: Real);
Function MyFunc(var a,b,c: Integer; d: Real):Real;
```

Когда параметр передается посредством *ссылки*, фактический параметр является переменной. Формальный параметр обозначает эту фактическую переменную в течение всего времени активизации блока. Параметры, переданные посредством ссылки, называются *параметрами-переменными*. Если параметр определен как параметр–переменная, то при вызове подпрограммы передается сама переменная, а не её копия (фактически в этом случае подпрограмме передается *адрес* переменной). Их характерный признак – любое изменение формального па-


```
                    begin
                      max:=A[i];
                      i_max:=i;
                    end;
 WriteLn('Минимальный элемент A[',i_min,']=',min);
 WriteLn('Максимальный элемент A[',i_max,']=',max);
  r:=A[i_min];     {перестановка}     {другой способ:}
  A[i_min]:=A[i_max];                 { A[i_min]:=A[i_max];}
  A[i_max]:=r;                        { A[i_max]:=min;}
  WriteLn ('Полученный вектор A:');
  For i:=1 to N do     {вывод полученного вектора}
  Write (A[i]:5);
  WriteLn;
End.
```

ИСПОЛЬЗОВАНИЕ ПОДПРОГРАММ

Подпрограммой называется именованная логически законченная группа операторов языка, которую можно вызвать для выполнения по имени любое количество раз из различных мест программы. В языке Pascal для организации подпрограмм используются процедуры и функции.

Все процедуры и функции языка Pascal подразделяются на две группы: встроенные и определенные пользователем.

Процедуры и функции пользователя организовываются самим программистом в соответствии с синтаксисом языка. Предварительное (перед использованием) описание процедур и функций пользователя обязательно.

В соответствии с областями применения различают 9 основных групп встроенных процедур и функций: арифметические, скалярные, преобразования типов, управления строками на экране, специальные, обработки строк, обработки файлов, управления памятью для динамических переменных, управления графикой.

Специальные процедуры

`Delay (Time)` – организует задержку выполнения программы на `Time` мс.

`Exit` – обеспечивает выход из выполняемого блока в окружающую среду. Если текущий блок является процедурой или функцией, выход производится во внешний блок. Если `Exit` указана в оператор-

ной части основной программы, программа прекращает работу и управление передается системе программирования.

`Halt` – прекращает выполнение программы и передает управление системе программирования.

Специальные функции

`KeyPressed` – возвращает результат `True`, если на клавиатуре была нажата какая-либо клавиша, и `False` в противном случае.

`SizeOf (IT)` – вычисляет объем основной памяти в байтах, которую занимает указанная переменная или тип. Результат имеет целочисленный тип. IT – идентификатор переменной или типа данных.

Процедура пользователя представляет собой именованную группу операторов, реализующую определенную часть общей задачи и вызываемую при необходимости для выполнения по имени из любой позиции раздела операторов. Описание процедуры включает заголовок и тело процедуры. Заголовок состоит из зарезервированного слова `Procedure`, идентификатора (имени) процедуры и необязательного заключенного в скобки списка формальных параметров с указанием типа каждого параметра. Имя процедуры – идентификатор, уникальный в пределах программы. Тело процедуры представляет собой локальный блок, по структуре аналогичный программе:

Формат описания процедуры:

`Procedure` имя_процедуры (формальные_параметры); {заголовок}
  раздел описаний
`Begin` {тело процедуры}
  раздел операторов
`End`;

Для обращения к процедуре используется оператор вызова процедуры. Он состоит из идентификатора (имени) процедуры и списка фактических параметров, отделенных друг от друга запятыми и заключенных в круглые скобки. Список параметров может отсутствовать, если процедуре не передается никаких значений.

Формат обращения к процедуре:

имя_процедуры (фактические_параметры);

Параметры обеспечивают механизм замены, который позволяет выполнять процедуру с различными начальными данными. Между фактическими параметрами в операторе вызова процедуры и формальными параметрами в заголовке описания процедуры устанавливается взаимно-однозначное соответствие по типу, количеству и порядку следования.

Функция, определенная пользователем, состоит из заголовка и тела функции. Заголовок содержит зарезервированное слово Function, идентификатор (имя) функции, заключенный в круглые скобки необязательный список формальных параметров и тип возвращаемого функцией значения.

Формат описания функции пользователя:

`Function` имя (формальные_параметры): тип_результата;
  разделы описаний
`Begin` {тело функции}
  раздел операторов
`End`;

В разделе операторов должен находиться по крайней мере один оператор, присваивающий идентификатору функции значение. Если таких присваиваний несколько, то результатом выполнения функции будет значение последнего оператора присваивания.

Обращение к функции осуществляется по имени с необязательным указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам, указанным в заголовке, и иметь тот же тип.

Директивы компилятора

Сразу за заголовками подпрограммы может следовать одна из стандартных директив компилятора, которые уточняют его действия и распространяются на всю подпрограмму и только на нее:

`Assembler` – тело подпрограммы написано на ассемблере;

`External` – с помощью этой директивы объявляется внешняя подпрограмма;

`Far` – компилятор должен создать код подпрограммы, рассчитанный на дальнюю модель вызова;

`Near` – компилятор создает код подпрограммы, рассчитанный на ближнюю модель памяти (используется по умолчанию);

`Forward` – используется при опережающем описании подпрограмм для сообщения компилятору, что описание подпрограммы следует дальше по тексту программы (но в пределах текущего программного модуля);

`Inline` – тело подпрограммы реализуется с помощью встроенных машинных инструкций;

`Interrupt` – используется при создании процедур обработки прерываний.

В соответствии с архитектурой микропроцессора, в программе могут использоваться две *модели памяти*: ближняя и дальняя. Модель