Министерство образования и науки
Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ
(ТУСУР)

Н.В. Зариковская

Информатика

Учебное пособие

2012

Зариковская Н.В.
Информатика. Учебное пособие. – Томск: Томский государственный университет систем управления и радиоэлектроники (ТУСУР)., 2012. – 194 с.

СОДЕРЖАНИЕ

1 Введение

Информатика – это техническая наука, систематизирующая приемы создания, хранения, воспроизведения, обработки и передачи данных средствами вычислительной техники, а также принципы функционирования этих средств и методы управления ими.

Предмет информатики составляют следующие понятия:

- аппаратное обеспечение средств вычислительной техники;
- программное обеспечение средств вычислительной техники;
- средства взаимодействия аппаратного и программного обеспечения;
- средства взаимодействия человека с аппаратными и программными средствами.

В данном учебном пособии будут рассмотрены вопросы связанные с программным обеспечением средств вычислительной техники, разработкой и созданием программ на языке программирования Turbo Pascal.

Хотелось бы сразу отметить, что в основе го решения задачи с использованием программных средств, лежит главным образом математика, скорее даже способ мышления, в сочетании со знаниями языков программирования. В первую очередь, чтобы научиться программировать необходимо, чтобы ваши мысли могли быть изложены на математическом (формальном) языке. Далее наступает момент написания данного решения на языке программирования, с использованием структур данных и алгоритмов.

2 Среда Turbo Pascal

В настоящее время, Turbo Pascal не является одним из лучших языков программирования, но простота его синтаксиса идеально подходит для обучения, и именно поэтому мы начинаем именно с него.

И начнем с запуска программы. Файл программы – Turbo.exe. При запуске появляется окно, представленное на рисунке 2.1.

Pascal позволяет работать, как в полноэкранном режиме, так и в оконном. Для того чтоб перейти из полноэкранного режима в оконный и обратно используется сочетание клавиш <Alt+Enter>.

После запуска программы необходимо убедиться в ее работоспособности, проверить правильно ли прописаны директории в options/directories (там должны быть указаны пути относительно директории в которой находится сам Pascal).

Примечание. Часто бывает удобно прописать в директориях пустые строки, а все используемые файлы размещать в той папке, где лежит Turbo.exe.

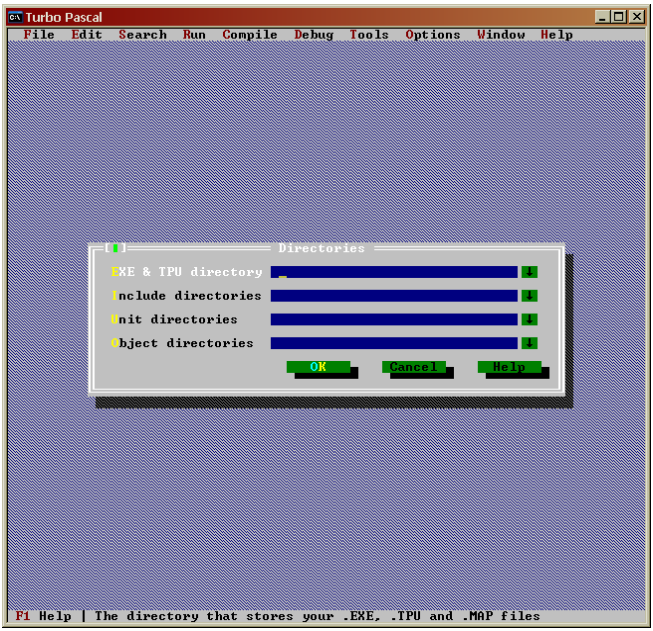


Рисунок 2.1 Оконный режим программы

Убедитесь в том, что в меню compile/destination у вас установлен disk (то место где будет сохраняться исполняемый файл.

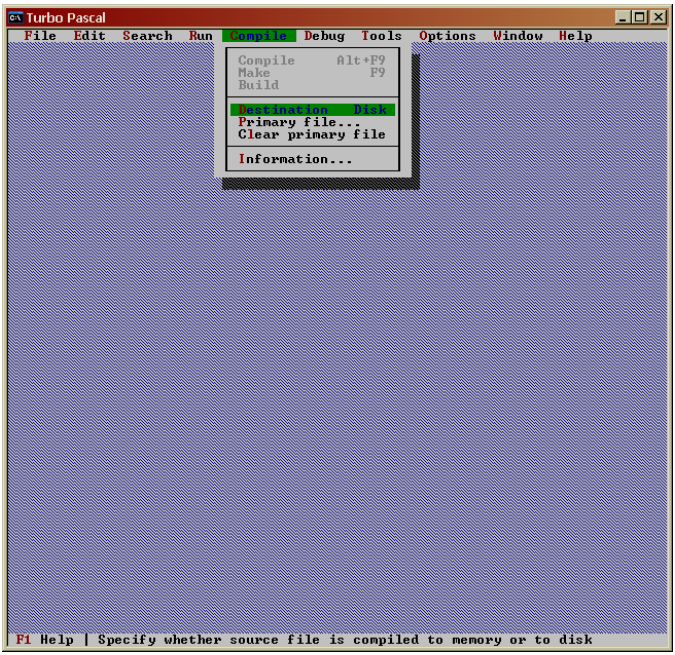


Рисунок 2.2 Настройки Turbo Pascal

Теперь Pascal готов для работы. С остальными особенностями работы в Паскале предлагаем ознакомиться самостоятельно.

Чтобы создать программу, зайдите в меню file/new. Создастся файл и появится окно редактора. Рекомендуется сразу сохранять код программы (F2). По ходу внесения изменений в код программы, следует периодически сохранять эти изменения. После того как вы напишите код программы, необходимо её скомпилировать (создать исполняемый файл). Полезно знать: F9 – скомпилировать; Ctrl+F9 – запустить программу.

Рассмотрим теперь подробнее основные принципы программирования на языке Turbo Pascal.

2.1 Основные понятия описания языка

Три составляющие любого языка: алфавит, синтаксис, семантика.

Алфавит языка – это множество символов, которые могут быть использованы в текстах этого языка.

Синтаксис – это набор правил, определяющих возможные сочетания (конструкции) из букв алфавита. Для описания синтаксиса языка, как правило, используют другой язык или синтаксические диаграммы.

Семантика – это набор правил, определяющих значение (смысл) отдельных конструкций языка.

2.2 Алфавит языка

Алфавит языка включает следующие символы.

Заглавные и строчные латинские буквы и символ «_» (который тоже считается буквой). Используются при создании Идентификаторов и служебных слов.

- Цифры от 0 до 9. Используются при записи и идентификаторов.

- Специальные символы: `+ - * / = < > . , ; : @ ^ ‘ ( ) [ ] { } $ #` используются как знаки операций, синтаксические разделители, при записи выражений, комментариев.

Символы алфавита языка используются для построения лексем – элементарных единиц языка, имеющих самостоятельный смысл. К лексемам относятся служебные слова, идентификаторы, знаки операций, разделители, изображения.

Служебные слова – это конечный набор зарезервированных слов, смысл которых определён в языке. Служебные слова нельзя использовать в качестве идентификаторов. Служебные слова:

ABSOLUTE EXPORTS LIBRARY
SET
ASSEMBLER EXTERNAL MOD
SHL
AND FAR NAME
SHR
ARRAY FILE NIL
STRING
ASM FOR NEAR
THEN
ASSEMBLER FORWARD NOT
TO
BEGIN FUNCTION OBJECT
TYPE
CASE GOTO OF
UNIT
CONST IF OR
UNTIL
CONSTRUCTOR IMPLEMENTATION PACKED
USES
DESTRUCTOR IN PRI-
VATE VAR
DIV INDEX PROCDURE
VIRTUAL
DO INHERITED PROGRAM
WHILE
DOWNTO INLINE PUBLIC
WITH
ELSE INTERFACE RECORD
XOR
END INTERRUPT REPEAT
EXPORT LABEL RESIDENT

Идентификаторы – это имена переменных, констант, процедур, функций, типов, меток. Составляются из букв, цифр и знаков подчёркивания в соответствии со следующими правилами:
- длина идентификатора может быть произвольной, однако компилятор воспринимает только первые 63 символа;
- первым символом обязательно должна быть буква или знак подчёркивания.

Например, правильными считаются идентификаторы:

`A_54  _123`

Поскольку Pascal нечувствителен к регистру, идентификатор ***A*** соответствует ***a***.

Неправильными будут идентификаторы:

`1Add begin 23_2345` и т. д.

В текстах программ возможно наличие пояснений, которые называются комментариями и не меняют смысл программы. Комментарий может находиться между двумя любыми лексемами программы, представляет из себя любую последовательность символов (т.е. может включать буквы русского алфавита), заключённую в фигурные скобки.

2.3 «Выражение» и «Оператор»

«Выражение» в алгоритмическом языке состоит из элементарных конструкций (минимальные единицы языка, имеющие самостоятельный смысл) и символов, оно задает правило вычисления некоторого значения.

«Оператор» - задает полное описание некоторого действия, которое необходимо выполнить. Для описания сложного действия может потребоваться группа операторов. В этом случае операторы объединяются в «составной оператор» или «блок» (осуществляется при использо-

вании программных скобок осуществляется при использовании программных скобок begin end).

Простейшие операторы:

`:=` - оператор присваивания. Применяется для задания какого-либо значения переменной.

`*` - умножение.

`/` - деление.

`+` - сложение.

`-` - вычитание.

`()` - оператор скобки. Применяется для изменения приоритетов в выражении.

`;` - «пустой» оператор.

Среди арифметических операторов порядок выполнения действий соответствует порядку аналогичных действий в математике.

Лишняя пара скобок, как и лишние пробелы, не изменяет скорости работы программы и не отражается на производительности, поэтому хорошим тоном программирования считается выделение с помощью скобок в математических выражениях составных частей (при этом, однако, следует следить за правильностью последовательности действий). Например, для выражения

```
rX:=7*iIma-byCount/54+65*byGong
```

более удобным будет эквивалентное представление в виде:

```
rX := (7*iIma) – (byCount/54) + (65*byGong)
```

2.4 Структура программы

В программировании хорошим тоном является в самом начале программы записать комментариями (используя `{}`) цель программы, входные и выходные данные. Это поможет вам в дальнейшем, когда у вас будет наработано некоторое количество программ, разбираться

с ними и использовать в качестве процедур для более сложных программ.

В целом, всегда старайтесь не скупиться на комментарии.

2.4.1 Тело программы

```
Begin
  тело программы
End.
```

Можно принять, что вся программа это один «большой и толстый» составной оператор, состоящий в свою очередь из множества других операторов, которые также могут быть составными.

Следует отметить, что после последнего end в программе ставится «.», означающая конец программы. Код, написанный после точки игнорируется при компиляции и не включается в исполняемый файл.

До тела программы, при необходимости следует описание подключаемых модулей, описание меток, констант, типов, переменных, процедур и функций, о чём речь пойдёт дальше.

2.4.2 Название программы

```
Program program1;
```

Имя программы является не обязательным, хотя в некоторых случаях полезно в использовании, например организация рекурсии.

Также рекомендуется в имени программы использовать содержащие смысл слова.

```
Program First_Program;
Begin
End.
```

2.4.3 Подключаемые модули

```
Uses
```

Здесь описываются подключаемые модули.

Модуль – совокупность программных ресурсов (констант, типов, переменных, подпрограмм), предназначенных для использования другими модулями и программами.

Другими словами, модуль – это часть программы, сохранённая в отдельном файле, процедуры, функции, константы и типы которого можно использовать в своей программе, при его подключении. Чтобы подключить модули в разделе Uses через запятую перечисляются имена файлов модулей.

Например:

```
Program Simple_Uses;
Uses
  crt, graph;
begin
  clrscr;{процедура очищения экрана из модуля crt}
end.
```

В дальнейшем будет рассматриваться создание собственных модулей.

2.4.4 Метки

```
Label
```

Здесь описываются метки, они перечисляются через запятую.

Если в программе поставить метку (название метки:), то по ходу программы можно будет переходить на неё, при помощи служебного слова goto.

На самом деле использование меток считается «дурным тоном» программирования, т.к. в любом случае

можно обойтись и без них, а при их использовании структура программы становится сложной и вносить в неё коррективы и изменения становится задачей довольно сложной. Код программы становится нечитабельным. Например:

```
Program Simple_Label;
Uses
  Crt;
Label
  Metka;
Begin
  Metka:
  Clrscr; {процедура модуля crt – очиска экрана}
  Goto Metka;
End.
```

Но на начальном этапе обучения, их использование может быть оправдано.

2.4.5 Константы

```
Const
```

Здесь описываются константы.

Использование констант иногда упрощает читабельность программы.

Константы описываются следующим образом:
Имя константы = значение константы;
Тип константы указывать не нужно, он определяется по её значению. Заданные константы нельзя менять в ходе программы.

```
Program Simple_const;
Uses
  Crt;
Const
  e=2.71828;
```

```
  Pi=3.1415926536;
  {Задание значений констант для чи-
сел e и пифагора}
Begin

End.
```

Нужно отметить, немного забегая наперед, что в разделе `const` Можно описывать переменные с начальными значениями. Для этого нужно написать имя переменной, через «:» ее тип, а затем через равно ее значение.

```
Program const_variables;
Const
  iSinmp: integer=2;
begin
end.
```

2.4.6 Описание типов

`Type`

Здесь описываются ваши собственные типы (они строятся на базовых типах). Так как использование базовых типов, в случаях, когда используется их суперпозиция, загромождает программу, имеет смысл описать собственные типы.

```
Program Simple_Type;
Type
  Tint=integer;
Begin
End.
```

2.4.7 Описание переменных

`Var`

Здесь описываются переменные.

Имя переменной: тип;

Значение указывать не обязательно.

Здесь в программе происходит резервирование памяти под переменные. Количество этой памяти зависит от типа переменной. И поэтому привыкайте не заводить лишних переменных, так, как Pascal ограничивает использование памяти.

Полезно знать

В настоящее время в профессиональном программировании принято записывать имена переменных с использованием так называемой венгерской нотации.

Венгерская нотация - это соглашение о наименованиях переменных и функций. Соглашение широко используется при программировании на языках Pascal, C и в среде Windows.

Венгерская нотация основывается на следующих принципах:

-имена переменных и функций должны содержать префикс, описывающий их тип;

-имена переменных и функций записываются полными словами или словосочетаниями или их сокращениями, но так, чтобы по имени можно было понять назначение переменной или действие, выполняемое функцией.

Префиксы записываются малыми буквами, первая буква каждого слова - заглавная, префиксы и слова записываются либо слитно, либо через символ _ (подчеркивание).

Для языка Pascal могут быть рекомендованы следующие префиксы для скалярных переменных и функций:

Префикс	**Тип**
By	Byte
Sh	Shortint
I	Integer

W	Word
L	Longint
R	Real
Si	Single
D	Double
E	Extended
C	Comp
Ch	Char
B	Boolean
P	Pointer
x,y	координаты символа или точки на экране

Для величин структурированного типа могут быть использованы следующие префиксы:

Префикс	Тип
A	Array
S	String
Sz	Stringz
Se	Set
Re	Record
F	File
T	Text

```
Program Simple_Variables;
Type
  Tint=Integer;
Var
  iSum: integer;
  bySume, byCount: byte;
  Look: Tint;
Begin
  iSum := 0;
  iSum := iSum + 5;
```

```
  bySum := 20;
  byCount := bySum + iSum;
  Look := bySum + byCount + iSum;
End.
```

2.4.8 Основные единицы программирования

Основными единицами программирования являются цикл и действие при условии. На этом строится все программирование, и суть мышления программиста заключается в умении представления своих идей на языке: «Сделать при условии», «Сделать многократно» и их сочетаниях. «Сделать» - означает выполнение какого-либо оператора.

2.4.8.1 Условие

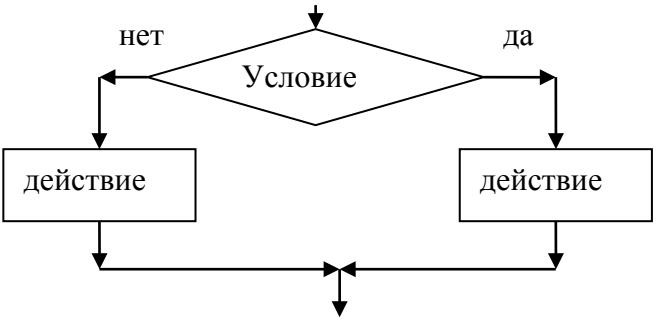
Как уже было сказано выше, действие при условии является основным методом программирования, на нем строится математическая логика. Общая структура:

Если «условие» тогда «действие» в противном случае «действие»

В Паскале это реализуется следующими способами:

1. `If` “условие” `then` “действие” `else` “действие”, где «условие» – выражение, значением которого является булевская величина.

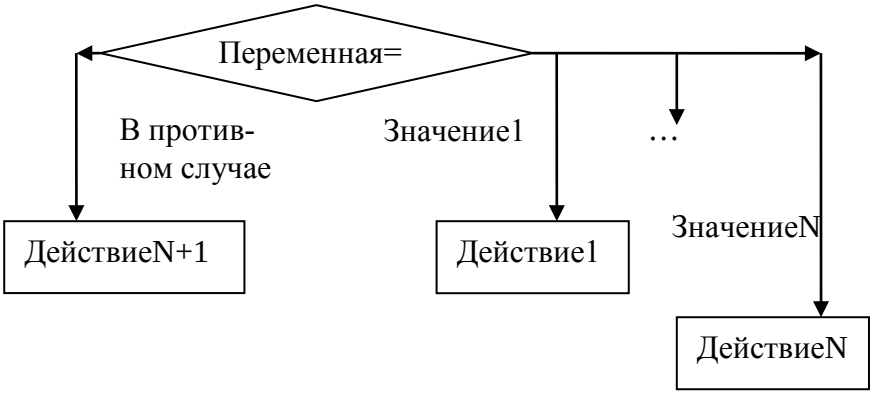
«действие» - оператор (простой или составной). В данной конструкции else «действие» является не обязательной.




```
If iSum>15 then
Begin
  xPos := iSum-2;
  yPos := iSum+2;
End
Else
  iSum := 8;
```

```
2. case «переменная» of
    Значение1: Действие1;
    ..........
          ЗначениеN: ДействиеN
    Else действиеN+1;
  End;
```

Здесь `else` “действиеN+1” так же является необязательным.




```
Case iX of
  1: iY := 2;
  2..15: iY := 3;
  16,18,25: iY := 8;
  Else iY := 0;
End;
```

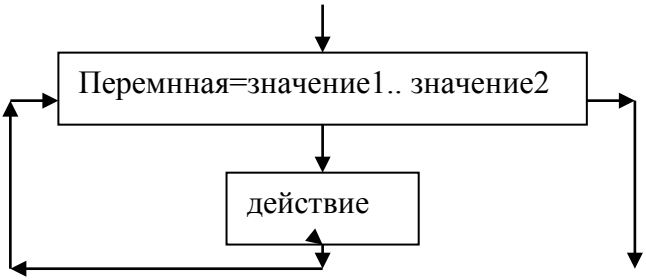
2.4.8.2 Циклы

Цикл – еще один основной метод программирования. Суть – Многократное повторение какого-либо оператора (простого либо составного). Pascal предусматривает три типа циклов: цикл с параметром, цикл с предусловием, цикл с постусловием.

1.`For` “переменная”:=”значение1” `to` ”значение2” `do` “действие”;

Что означает, для всех целых значений переменной начиная от “значения1” до “значения2”, выполнять

“действие”. Причём “значение1” и “значение2” - целые числа.



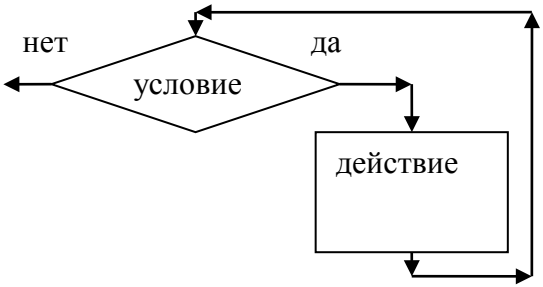
Данная конструкция цикла является удобной в тех случаях, когда мы знаем точно сколько раз будет проведено действие. При этом происходит наращение переменной от значения1 до значения 2. Если же требуется наоборот, уменьшение переменной – вместо to используется downto

```
iP := 1;
For byCount1 := 1 to 100 do
Begin
  iX := 0;
  For byCount2 := 100 Downto 1 do
    iX :=  iX + byCount2 + byCount1;
  iP := iP*iX;
End;
```

В цикле с параметром нельзя искусственно изменять параметр, иначе компилятор сообщит об ошибке.

2.`While` “условие” `do` “действие”;

Что означает, до тех пор, пока выполняется “условие”, выполнять “действие”.

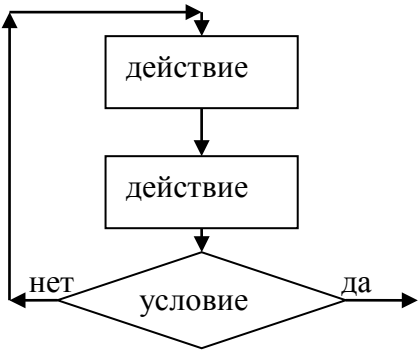


{приближённое вычисление суммы бесконечно убывающей прогрессии с первым членом 100 и со знаменателем 1/2}

```
rQ := 0.5;
rB := 100;
While rB>0.0001 do
Begin
  rS := rS + rB;
  rB := rB*rQ;
End;
```

3.`Repeat` “блок действий” `until` “условие”;

Что означает, выполнение “блока действий” до тех пор пока не выполнится “условие”.



{пример реализации предыдущего алгоритма с использованием цикла repeat}

```
rQ := 0.5;
rB := 100;
Repeat
  rS := rS + rB;
  rS := rB*rQ;
Until rB<0.0001;
```

Практически во всех случаях эти три цикла взаимозаменяемы, однако существуют такие условия, в которых применение какого-то определённого цикла упрощает понимание кода программы.

2.4.8.3 Процедуры ввода-вывода

Ввод и вывод данных также является основной единицей программирования, потому что любая программа – это обработка информации, которую необходимо предварительно ввести, и результат обработки которой необходимо вывести.

Для ввода и вывода данных используются стандартные процедуры ввода и вывода `Read` и `Write`

(Readln, Writeln). Окончание "-ln" означает лишь возврат каретки (переход на следующую строку).

Read(a1,a2,...,an); - считывает с клавиатуры введённые значения в переменные a1,a2,...,an. Для a1,a2,...,an допустимы типы:

целые;
вещественные;
символьные;
строковые.

Если в процедуру ReadLn не передаются никакие параметры (т.е. записано «ReadLn;»), происходит считывание значения «вникуда».

Write(a1,a2,...,an); - выводит значения переменных a1,a2,...,an без пробелов между ними. Для a допустимы типы:

целые;
вещественные;
символьные;
строковые;
булевские.

Кроме переменных в процедуре Write /Writeln могут использоваться также выражения данных типов.

Если в процедуру WriteLn не передаются никакие параметры (т.е. записывается «WriteLn;»), то происходит возврат каретки.

Вывод каждого значения в строку экрана происходит в соответствии с шириной поля вывода, определяемой конкретной реализацией языка.

Оператор вывода позволяет задать ширину поля вывода для каждого элемента списка вывода. В этом случае элемент списка вывода имеет вид A:K, где A - выражение или строка, K - выражение либо константа целого типа. Если выводимое значение занимает в поле вывода

меньше позиций, чем К, то перед этим значением располагаются пробелы. Если выводимое значение не помещается в ширину поля К, то для этого значения будет отведено необходимое количество позиций. Для величин действительного типа элемент списка вывода может иметь вид А:К:М, где А - переменная или выражение действительного типа, К - ширина поля вывода, М - число цифр дробной части выводимого значения. К и М - выражения или константы целого типа. В этом случае действительные значения выводятся в форме десятичного числа с фиксированной точкой.

```
WriteLn('A = ', 5);
```

При этом на экран будет выведено:

```
A = 5
```

```
ReadLn(iA, iB);
```

При этом на экране появится курсор. При нажатии Enter («Ввод»), введённое значение будет присвоено переменной Ia (если была введена пустая строка, т.е. нажата клавиша Enter, значение будет запрошено ещё раз, а если тип введённых данных не соответствует типу переменной – произойдёт ошибка), затем операция повторится для Ib.

2.4.8.4 Операторы выхода

В некоторых случаях возникает необходимость завершения выполнения программы, самой программой.

Для завершения работы программ, процедур и функций без предварительного перехода по меткам к закрывающему end в Pascal введены процедуры `Exit` и `Halt`.

Вызов Exit завершает работу своего программного блока и передает управление вызывающей программе. Если Exit выполняется в подпрограмме, то выполнение этой подпрограммы прекратится, и далее будет выполняться следующий за вызовом этой подпрограммы оператор. Если Exit выполняется в основной программе, выход из нее будет эквивалентен ее нормальному завершению.

Вызов процедуры Halt, где бы она не находилась, завершает работу программы и передает управление операционной системе.

Процедура Halt имеет структуру Halt(n), где n - код возврата, который может быть проанализирован операционной системой с помощью команды If Errorlevel. Значение n=0 соответствует нормальному завершению работы программы. Вызов процедуры Halt без параметра эквивалентен
вызову Halt(0).

Пример программы на паскале:

{программа предназначена для демонстрации структуры программ
на Паскале}

```
program simple;

uses

  {Подключение модулей crt и graph}
  crt, graph;

label
  start_of_program;

const
  {задание значения константе пи}
```

```
  Pi = 3.14159265;
  {задание значения константе n}
  N = 10;



type
  {объявление собственного типа, в частности далее будет описано, что это описание типа массива вещественных чисел}
  Tmass = array[1..10] of real;


var
  arMass: Tmass; {объявление переменной mass типа Tmass}
  iI, iJ,
  iZelaya: integer;
  rVeshestv: real;
  byA, byB, byC, byS: byte;

begin
  start_of_program: {метка}
  readln(byA);     {ввод данных}
  readln(byB);
  readln(byC);
  readln(byS);
  for Ii:=1 to n do  {пример использования цикла с параметром}
  begin
  {пример составного оператора (начинается со слова begin и заканчивается словом end)}
    arMass[iI]:=57;
```

```
    arMass[iI]:=(arMmass[iI]-
by+byA*byB)*byC;
  end;
  {пример цикла с предусловием}
  while arMass[5]<100 do
    arMass[5]:=arMass[5]+abs(rVeshestv);
  iJ:=1;
  repeat       {пример цикла с пост усло-
вием}
    writeln(arMass[iJ]);
    inc(iJ);
  until iJ=10;
  iZelaya:=48;
  case iZelaya of
    1           : writeln(iZelaya);{вывод
данных}
    2..10       : writeln(iZelaya mod 2);
    11..50, 90 : iZelaya:=-iZelaya;
    else         exit;
  end;
  if arMass[1]=trunc(arMass[iI]) then go-
to   start_of_program;{переход   к   метке
начала программы}
end.
```

3 Типы данных

В математике принято классифицировать переменные в соответствии с некоторыми важными характеристиками. Производится строгое разграничение между вещественными, комплексными и логическими переменными, между переменными, представляющими отдельные значения и множество значений и так далее.

При обработке данных на ЭВМ такая классификация еще более важна. В любом алгоритмическом языке

каждая константа, переменная, выражение или функция бывают определенного типа.

В языке Pascal существует правило: ип явно задается в описании переменной или функции, которое предшествует их использованию. Концепция типа языка Pascal имеет следующие основные свойства:

-любой тип данных определяет множество значений, к которому принадлежит константа, которые может принимать переменная или выражение, или вырабатывать операция или функция;

-тип значения, задаваемого константой, переменной или выражением, можно определить по их виду или описанию;

-каждая операция или функция требует аргументов фиксированного типа и выдает результат фиксированного типа.

Отсюда следует, что транслятор может использовать информацию о типах для проверки вычислимости и правильности различных конструкций.

Тип определяет:

-возможные значения переменных, констант, функций, выражений, принадлежащих к данному типу;

-внутреннюю форму представления данных в ЭВМ;

-операции и функции, которые могут выполняться над величинами, принадлежащими к данному типу.

Обязательное описание типа приводит к избыточности в тексте программ, но такая избыточность является важным вспомогательным средством разработки программ и рассматривается как необходимое свойство современных алгоритмических языков высокого уровня.

-к любому простому типу может быть применена стандартная функция `Ord`, которая в качестве результата возвращает порядковый номер конкретного значения в данном типе;

-к любому простому типу могут быть применены стандартные функции `Pred` и `Succ`, которые возвращают предыдущее и последующее значения соответственно;

-к любому простому типу могут быть применены стандартные функции `Low` и `High`, которые возвращают наименьшее и наибольшее значения величин данного типа.

В языке Pascal введены понятия эквивалентности и совместимости типов.

Два типа T1 и T2 являются эквивалентными (идентичными), если выполняется одно из двух условий:

-T1 и T2 представляют собой одно и то же имя типа;

-тип T2 описан с использованием типа T1 с помощью равенства или последовательности равенств. Например:

```
type
  T1 = Integer;
  T2 = T1;
  T3 = T2;
```

Менее строгие ограничения определены совместимостью типов. Например, типы являются совместимыми, если:

-они эквивалентны;

-являются оба либо целыми, либо действительными;

-один тип - интервальный, другой - его базовый;

-оба интервальные с общим базовым;

один тип - строковый, другой - символьный.

В Pascal ограничения на совместимость типов можно обойти с помощью приведения типов. Приведение типов позволяет рассматривать одну и ту же величину в

памяти ЭВМ как принадлежащую разным типам. Для этого используется конструкция

Имя Типа(переменная или значение).

Например,

```
Integer('Z')
```

представляет собой значение кода символа `'Z'` в двухбайтном представлении целого числа, а

```
Byte(534)
```

даст значение 22, поскольку целое число 534 имеет тип `Word` и занимает два байта, а тип `Byte` занимает один байт, и в процессе приведения старший байт будет отброшен.

3.1 Простые типы данных в паскале

3.1.1 Логический тип

Логический тип (`Boolean`) определяет те данные, которые могут принимать логические значения `True` и `False`.

Значения булевского типа возвращают операторы отношения

= - равно
< - Больше
> - Меньше
>= - Больше либо равно
<= - Меньше либо равно
<> - Неравно.

В выражениях, операции отношения имеют более низкий приоритет, чем все математические действия.

Значение True возвращается в том случае, если выполняется равенство, в противном случае возвращается False.

К булевским операндам применимы следующие логические операции:

`Not` – («Не») противоположное значение. Применяется к одному значению.

`And` – Логическое умножение. Применяется к двум значениям, возвращает True, если оба значения True, и возвращает False, если хотя бы одно значение False.

`Or` – Логическое сложение. Применяется к двум значениям, возвращает False, если оба значения False, и возвращает True, если хотя бы одно значение True.

`Xor` – Логическое вычитание. Применяется к двум значениям, возвращает True, если значения относительно которых применена данная операция не равны, и False в противном случае.

Что касается приоритета выполнения этих операций в выражениях:

`Not` имеет самый высокий приоритет. And имеет такой же приоритет как и * и /. `Or` и `Xor` имеет такой же приоритет, как и + и -. И поэтому надо быть аккуратными в составлении длинных логических выражений. Например запись

```
If iA+iB=iS and bU or iM=8 then
Begin
End;
```

Является вообще логически неправильной, потому что = имеет меньший приоритет, чем and и or. И данная запись будет эквивалентна следующей:

```
If ((iA+iB)=(((iS and bU) or
iM))=8then
Begin
```

```
End;
```

Здесь уже четко прослеживается, что iS and bU не допустимо, потому что iS не является Булевским, (iS and bU) or iM так же не корректно, потому что iM не Булевского типа.

Правильными вариантами могут быть:

```
1 if (iA+iB=iS) and bU or (iM=8) then
  Begin
  End;
2 if (iA+iB=iS) and (bU or (iM=8))
then
  Begin
  End;
```

Заметим, что если `(iM=8)=true`, `bU=false` и `(iA+iB=iS)=false` то в первом случае общее условие выполнится, а во втором нет.

Логический тип определен таким образом, что `False < True`. Это позволяет применять к булевским операндам все операции отношения.

В Паскале введены еще разновидности логического типа:
`ByteBool`, `WordBool` и `LongBool`, которые занимают в памяти ЭВМ один, два и четыре байта соответственно.

```
Program Simple_boolean;
Var
  bA, bB, bC: Boolean;
Begin
  bA:=2>3;
  bB:=9>=10;
  bC:=bA and bC;
  If not bC then
  Begin
          End;
```

```
End.
```

3.1.1.2 Битовая арифметика

Битовая или поразрядная арифметика введена в Pascal для обеспечения возможности работы с двоичными разрядами (битами). Операции битовой арифметики применимы только к целым типам.

Первая группа операций - логические операции not, and, or и xor.

Операция `not` является одноместной, она изменяет каждый бит целого числа на обратный.

Операции `and`, `or` и `xor` - двуместные, операнды этих операций – целые величины одинаковой длины. Операции выполняются попарно над всеми двоичными разрядами операндов.

Вторая группа операций - это операции сдвига влево shl и сдвига вправо shr:

```
I shl N
I shr N.
```

Эти операции сдвигают двоичную последовательность значения I влево или вправо на N двоичных разрядов. При этом биты, уходящие за пределы разрядной сетки, теряются, а освободившиеся двоичные разряды заполняются нулями. При сдвиге вправо отрицательных значений освободившиеся разряды заполняются единицами.

Что касается приоритета, то он для этих операций такой же как и для * и /.

3.1.2 Целые типы

Целые типы определяют константы, переменные и функции, значения которых реализуются множеством целых чисел, допустимых в данной ЭВМ.

Тип	Диапазон значений	Требуемая память
Shortint	-128.. 127	1 байт
Integer	-32768.. 32767	2 байта
Longint	-2147483648.. 2147483647	4 байта
Byte	0.. 255	1 байт
Word	0.. 65535	2 байта

Над целыми операндами можно выполнять следующие арифметические операции:

+ сложение

- вычитание

* умножение

`div` целая часть от деления

`mod` получение остатка от деления

В выражениях div и mod имеют такой же приоритет, как и * и /.

Результат арифметической операции над целыми операндами есть величина целого типа. Результат выполнения операции деления целых величин есть целая часть частного. Результат выполнения операции получения остатка от деления - остаток от деления целых.

К целым операндам применимы все операции отношения.

К аргументам целого типа применимы следующие стандартные (встроенные) функции, результат выполнения которых имеет целый тип:

`Abs(X)` модуль

`Sqr(X)` квадрат

Следующая группа стандартных функций для аргумента целого типа дает результат вещественного типа:

`Sin(X)` синус от аргумента, в радианах

`Cos(X)` косинус от аргумента, в радианах

`ArcTan(X)` арктангенс
`Ln(X)` натуральный логарифм
`Exp(X)` e^X
Sqrt(X) корень квадратный

Полезно знать. Данные функции разложимы в ряд Тейлора, именно поэтому они запрограммированы.

Функции прямого возведения в степень в паскале нет, однако, с помощью вышеуказанных функций это можно обойти:

$$y^x = \exp(x * \ln(y)).$$

Результат выполнения функции проверки целой величины на нечетность Odd(X) имеет значение истина, если аргумент нечетный, и значение ложь, если аргумент четный:

Для быстрой работы с целыми числами определены процедуры:

```
Inc(X)      X:=X+1
Inc(X, N)   X:=X+N
Dec(X)      X:=X-1
Dec(X, N)   X:=X-N
```

Полезно знать.

При использовании `Inc(X)` и `Dec(X)` в циклах с пред и постусловием, зависящим от X, имеет смысл для облегчения понимания кода сторонними программистами перейти к циклу с параметром. Например, цикл

```
Ii:=1;
While Ii<=10 do
Begin
  Inc(Ii);
End;
```

Эквивалентен циклу

```
For Ii:=1 to 10 do
Begin
End;
```

В Pascal предусмотрена функция случайных чисел `Random(iA: integer): integer;` Которая генерирует случайные числа в промежутке длиной iA, начиная от нуля. Чтобы случайные числа были «более» случайными, необходимо предварительно выполнить процедуру `randomize;`

```
Program simple_random;
Var
  iA:=integer;
begin
  randomize;
  iA:=random(100);
end.
```

3.1.3 Вещественные типы

Вещественные типы определяет те данные, которые реализуются подмножеством действительных чисел, допустимых в данной ЭВМ.

Тип	Диапазон значений	Количество цифр мантиссы	Требуемая память (байт)
`Real`	2.9e-39 .. 1.7e+38	11	6
`Single`	1.5e-45 .. 3.4e+38	7	4
`Double`	5.0e-324 .. 1.7e+308	15	8

Extended	3.4e-4932 .. 1.1e+4932	19	10
Comp	-9.2e+18 .. 9.2e+18	19	8

Здесь 2.9e-39 означает 2.9×10^{-39}

Тип Comp хотя и относится к действительным типам, хранит только длинные целые значения.

Над действительными операндами можно выполнять следующие арифметические операции, дающие действительный результат:

+ сложение
– вычитание
* умножение
/ деление

К величинам действительного типа применимы все операции отношения, дающие булевский результат.

Один из операндов, участвующих в этих операциях, может быть целым.

К действительным аргументам применимы функции, дающие действительный результат:

Abs(X) абсолютное значение
Sqr(X) квадрат
Sin(X) синус
Cos(X) косинус
ArcTan(X) арктангенс
Ln(X) натуральный логарифм
Exp(X) e^X
Sqrt(X) корень квадратный
Frac(X) возвращает дробную часть X
Int(X) возвращает вещественное значение целой части X

Pi возвращает значение числа Пи действительного типа.

`Trunc(X)` выделяет целую часть действительного аргумента путем отсечения дробной части

`Round(X)` округляет аргумент до ближайшего целого

3.1.4 Символьный тип

Символьный тип (`Char`) определяет упорядоченную совокупность символов, допустимых в данной ЭВМ. Значение символьной переменной или константы - это один символ из допустимого набора.

Символьная константа может записываться в тексте программы тремя способами:

1. как один символ, заключенный в апострофы (`'A' 'a' 'Ю' 'ю'`)
2. с помощью конструкции вида #K, где K - код соответствующего символа, при этом значение K должно находиться в пределах 0..255;
3. с помощью конструкции вида ^C, где C - код соответствующего управляющего символа, при этом значение C должно быть на 64 больше кода управляющего символа.

К величинам символьного типа применимы все операции отношения.

Для величин символьного типа определены две функции преобразования.

`Ord(C)` определяет порядковый номер символа C в наборе символов

`Chr(K)` определяет по порядковому номеру `K` символ, стоящий на K-ом месте в наборе символов Порядковый номер имеет целый тип.

`Pred(C)` определяет предыдущий символ

`Succ(C)` определяет последующий символ

При отсутствии предыдущего или последующего символов значение соответствующих функций не определено.

Для литер из интервала `'a'..'z'` применима функция `UpCase(C)`, которая переводит эти литеры в верхний регистр `'A'..'Z'`.

К символьным переменным применимы также операции отношения. При этом надо помнить, что сравниваются они по коду символов. Например, `‘a’ < ‘z’`.

3.1.5 Перечисляемый тип данных

Перечисляемый тип представляет собой ограниченную упорядоченную последовательность скалярных констант, составляющих данный тип. Значение каждой константы задается ее именем. Имена отдельных констант
отделяются друг от друга запятыми, а вся совокупность констант, составляющих данный перечисляемый тип, заключается в круглые скобки.

Программист объединяет в одну группу в соответствии с каким – либо признаком всю совокупность значений, составляющих перечисляемый тип. Например, перечисляемый тип `Cveta` объединяет скалярные значения `Red, Orange, Yellow, Green, Light_blue, Blue, Violet.` Перечисляемый тип `Traffic_Light` объединяет скалярные значения `Red, Yellow, Green`.

Перечисляемый тип описывается в разделе описания типов, который начинается со служебного слова `type`, например:

```
type
  Cveta = (Red, Orange, Yellow,
Green, Light_blue, Blue, Violet);
```

Каждое значение является константой своего типа и может принадлежать только одному из перечисляемых типов, заданных в программе. Например, перечисляемый тип `Traffic_Light` не может быть определен в одной программе с типом `Rainbow`, так как оба типа содержат одинаковые константы.

Описание переменных, принадлежащих к скалярным типам, которые объявлены в разделе описания типов, производится с помощью имен типов. Например:

```
type
  Traffic_Light=(Red,        Yellow,
Green);
Var
  Section: Traffic_Light;
```

Это означает, что переменная `Section` может принимать значения `Red, Yellow` или `Green`.

Переменные перечисляемого типа могут быть описаны в разделе описания переменных, например:

```
Var
  Section: (Red, Yellow, Green);
```

При этом имена типов отсутствуют, а переменные определяются совокупностью значений, составляющих данный перечисляемый тип.

К переменным перечисляемого типа может быть применим оператор присваивания:

```
Section:= Yellow;
```

Упорядоченная последовательность значений, составляющих перечисляемый тип, автоматически нумеруется, начиная с нуля и далее через единицу. Отсюда следует, что к перечисляемым переменным и константам могут быть применены операции отношения и стандартные функции `Pred, Succ, Ord.`

Переменные и константы перечисляемого типа не могут быть элементами списка ввода или вывода.

3.1.6 Ограниченный тип данных

Отрезок любого порядкового типа может быть определен как ограниченный или интервальный тип. Отрезок задается диапазоном от минимального до максимального значения констант, разделенных двумя точками. В качестве констант могут быть использованы константы, принадлежащие к целому, символьному, логическому или перечисляемому типам. Скалярный тип, на котором строится отрезок, называется базовым типом.

Минимальное и максимальное значения констант называются нижней и верхней границами отрезка, определяющего ограниченный тип.

Нижняя граница должна быть меньше верхней.

Над переменными, относящимися к ограниченному типу, могут выполняться все операции и применяться все стандартные функции, которые допустимы для соответствующего базового типа.

При использовании в программах ограниченных типов данных может осуществляться контроль за тем, чтобы значения переменных не выходили за границы, введенные для этих переменных в описании ограниченного типа.

```
Program simple_inter;
Type
```

```
  Numb=1..10;
Var
  N: numb;
  alfa: ‘a’..’z’;
begin
end.
```

3.2 Составные типы данных

3.2.1 Регулярные типы данных (массивы)

`Array`

Бывают задачи, в которых нельзя обойтись некоторым достаточно небольшим набором переменных, а если выразиться точнее, то редко бывают такие задачи, в которых не требуется обработка больших наборов данных. Для этого в первую очередь используются массивы.

Массивы представляют собой ограниченную упорядоченную совокупность однотипных величин. Каждая отдельная величина называется компонентой массива. Тип компонент может быть любым, принятым в языке Pascal (и составным в том числе) кроме файлового типа. Тип компонент называется базовым типом. Проще говоря, массив – ряд переменных одного типа. Для всех компонент в массиве существует одно имя, а для доступа к отдельным из них используется индекс.

Для того чтобы объявить массив в паскале и используется следующая конструкция:

`Array[индекс1, индекс2, ..]of Базовый тип`.

`[индекс1, индекс2, ..]` по сути определяет размер массива (количество его элементов). `индекс1, индекс2, ..` – Интервалы, указанные перечисляемым типом, базовым для которого могут являться Булевский, Символьный и Целочисленный типы.

В зависимости от количества индексов массивы бывают одномерные (Вектора), двухмерные (матрицы), трехмерные и т.д.

Объявить массив можно как предварительным описанием нового типа в разделе `type`, так и в разделе описания переменных (`var`).

Например:

```
Program Arrays;
Type
  TcMass=array[‘a’.. ‘z’] of char;
  TMass=array[false.. true]of
          Array[false..  true]of  in-
teger;
Var
  acB, acC: TcMass;
  aaiM: Tmass;
  arV:  arra[1..10,  1..10,  1..10]of
real;

Begin
  arV[1, 1, 1]:=0.6;
End.
```

При обращение к несуществующей ячейке массива программа выдаст ошибку.

В операторной части программы один массив может быть присвоен другому, если их типы идентичны, например для предыдущей программы можно использовать:

```
acB:=acC;.
```

Чтобы создать массив, и прописать в него начальные значения всех его элементов испоьзуется следующая конструкция:

```
Type
  Dim10 = Array[1..10] of Real;

const
  raM10: Dim10 = ( 0, 2.1, 4, 5.65,
6.1, 6.7, 7.2, 8, 8.7, 9.3 );
```

Для двумерных массивов значения компонент каждого из входящих в него одномерных массивов записывается в скобках:

```
Type
  Dim3x2= Array[1..3,1..2] of Inte-
ger;

const
  iaM3x2: Dim3x2= (  (1, 2)
                     (3, 4)
                     (5, 6) );
```

Отметим, что в вышеприведенных случаях, если не указывать имя типа, то мы будем иметь константный массив.

Pascal допускает два типа составных структур, в частности - массивов: упакованные и распакованные. Упакованные позволяют экономить оперативную память, однако в значительной степени снижают быстродействие программы. Для объявления упакованного используется служебное слово packed:

```
arMase: packed array [1..150] of re-
al;
```

Все вышеприведённые массивы относятся к типу распакованных, которые применяются в подавляющем большинстве случаев.

Массивы занимают в памяти весьма значительные объёмы, поэтому целесообразно освоить некоторые приёмы работы с памятью.

Функция `SizeOf(V): word;`

Эта функция возвращает размер памяти, выделенной под переменную `V`. Например, если `V: array[1..10] of real`, то функция вернёт значение 40 (10 элементов, объёмом 4 байта каждый).

Процедура `FillChar( var V; NBytes: Word; B: Byte );`

Эта процедура заполняет все байты участка памяти, начиная с адреса переменной `V`, количеством `NBytes`, значением переменной B.

Например, для обнуления массива `A: array[1..10] of Real` можно записать:

```
FillChar(A, 40, 0);
```

или

```
FillChar(A, SizeOf(A), 0);
```

Процедура `Move(const Source; var Dest; Count: Integer);`

Эта процедура копирует Count байт начиная с адреса переменной Source в область памяти, начиная с адреса переменной Dest. Это, например, можно использовать, когда требуется приравнять друг к другу массивов разных типов.

Оператор @ - взятие адреса (имеет приоритет равный приоритету not при составлении выражений).

```
Program Simple_memory;
Var
  alA: array[1..10] of LongInt;
  arB: array[1..10] of Real;
begin
  FillChar(alA, 40, 0);
  Move(alA, arB, SizeOf(arB));
end.
```

3.2.2 Строки

Строковые переменные - это одномерные пакованные массивы символов, для описания которых в Pascal введен тип String.

Например, если строка содержит до 30 символов, ее тип будет определен как

```
type
  s= String[30];
```

Длина строки не может содержать более, чем 255 символов.

В Pascal определено понятие строки переменной длины, в этом случае ее описание задается как

```
Type
  s= String;
```

Тип String без указания длины совместим со всеми типами строк.

Особенностью строковых переменных является то, что к ним можно обращаться как к скалярным переменным, так и к массивам. Во втором случае применяется конструкция "переменная с индексом", что обеспечивает доступ к отдельным символам строки. При этом нижняя

граница индекса равна 1. Отдельный символ строки совместим с типом Char.

В памяти ЭВМ строка занимает количество байтов, на единицу большее ее длины. Нулевой байт строки содержит ее длину.

Строки могут быть элементами списка ввода - вывода, при этом записывается имя строки без индекса.

При вводе строковых переменных количество вводимых символов может быть меньше, чем длина строки. В этом случае вводимые символы размещаются с начала строки, а оставшиеся байты заполняются пробелами. Если количество вводимых символов превышает длину строки, лишние символы отбрасываются.

Для работы со строками в Pascal включены процедуры и функции, которые обеспечивают редактирование и преобразование строк.

Строки можно складывать. Для сравнения строк применяются все операции отношения. Сравнение строк происходит посимвольно, начиная с первого символа. Строки равны, если имеют одинаковую длину и посимвольно эквивалентны.

`Length(s: string): byte;` - возвращает длину строки.

Функция конкатенации (объединения строк)

`Concat(s1, [s2,…,sn]: string): string;`

Эта функция эквивалентна

`s1 + s2 + … + sn`

однако работает она значительно быстрее.

Функция

`copy(s: string; i, count: byte): string;`

Возвращает count символов строки `s`, начиная с `i`-того.

Процедура

Delete(var s: string; i, count: byte);

удаляет count символов из строки s, начиная с i-того.

Процедура

Insert(var s1, s2: string; i: byte);

вставляет строку s1 в s2 после i-того символа.

Функция

Pos(SubStr, Str: string): byte;

возвращает номер символа, с которого начинается первое вхождение подстроки SubStr в строке Str. Если Str не содержит SubStr, возвращается 0.

```
Program simple_string;
Var
   szStr, szSubstr, szCopy, szDelete,
   szInsert, szS1, szS2: string;
   byPos, byI, byCount: byte;
Begin
  byI:=5;
  byCount:=18;
  {Операции конкатенации}
  szStr:=sS1+sS2;
  szStr:=concat(sS1, sS2);
  {Копирование}
  szStr:=copy(szCopy, byI, byCount);
  {Определение позиции}
  byPos:=pos(szSubstr, szStr);
  {Удаление символов}
  delete(szStr, byI, 10);
  {Вставка}
  insert(szStr, szSubstr, 10);
  {сравнение строк}
  if szStr<szSubstr then
                          szStr:=sz
                    SubStr;
```

```
  byCount:=length(szStr);
End.
```

3.2.3 Множества

Понятие множества в языке Pascal основывается на математическом представлении о множествах: это ограниченная совокупность различных элементов. Для построения конкретного множественного типа используется перечисляемый или интервальный тип данных. Тип элементов, составляющих множество, называется базовым типом.

Множественный тип описывается с помощью служебных слов `Set of`, например:

```
type
  TM=Set of Byte;
```

Здесь `TM` - множественный тип, `Byte` - базовый тип.

Пример описания переменной множественного типа:

```
type
  TM= Set of 'A'..'D';
var
  seS: TM;
```

Принадлежность переменных к множественному типу может быть определена прямо в разделе описания переменных:

```
var
  seC: Set of 0..7;
```

Константы множественного типа записываются в виде заключенной в квадратные скобки последователь-

ности элементов или интервалов базового типа, разделенных запятыми, например:

```
['A', 'C']      [0, 2, 7]      [3, 7, 11..14].
```

Константа вида

```
[]
```

означает пустое подмножество.

Множество включает в себя набор элементов базового типа, все подмножества данного множества, а также пустое подмножество. Если базовый тип, на котором строится множество, имеет К элементов, то число подмножеств, входящих в это множество, равно 2 в степени К. Пусть имеется переменная Р интервального типа:

```
var
  P: 1..3;
```

Эта переменная может принимать три различных значения - либо 1, либо 2, либо 3. Переменная seMno множественного типа

```
var
  seMno: Set of 1..3;
```

может принимать восемь различных значений:

```
[ ]        [1,2]
[1]        [1,3]
[2]        [2,3]
[3]        [1,2,3]
```

Порядок перечисления элементов базового типа в константах безразличен.

Значение переменной множественного типа может быть задано конструкцией вида [Т], где Т - переменная базового типа.

К переменным и константам множественного типа применимы операции

присваивания(:=),
объединения(+),
пересечения(*)
и вычитания(-):

```
['A','B']+['A','D'] даст ['A','B','D']
['A']*['A','B','C'] даст ['A']
['A','B','C'] - ['A','B'] даст ['C']
```

Чтобы добавить или отнять от множества константу (по сути множество состоящее из одного элемента) Лучше использовать вместо простого сложения и вычитания процедуры

`Include(var seMno: set of T, a: T);` - Добавление во множество seMno, a, сдесь T – Базовй тип множества.

`Exclude(var seMno: set of T, a: T);` - соответсвенно удаление.

Результат выполнения всех выше перечисленных операций есть величина множественного типа.

К множественным величинам применимы операции: тождественность (=), нетождественность (<>), содержится в (<=), содержит (>=). Результат выполнения этих операций имеет логический тип, например:

```
['A','B'] = ['A','C'] даст False
['A','B'] <> ['A','C'] даст True
['B'] <= ['B','C'] даст True
['C','D'] >= ['A'] даст False.
```

Кроме этих операций для работы с величинами множественного типа вязыке Pascal используется операция

in

проверяющая принадлежность элемента базового типа, стоящего слева от знака операции, множеству, стоящему справа от знака операции. Результат выполнения этой операции - булевский. Операция проверки принадлежности элемента множеству часто используется вместо операций отношения, например:

```
‘A’ in ['A', 'B'] даст  True,
2 in [1, 3, 6]  даст  False.
```

При использовании в программах данных множественного типа выполнение операций происходит над битовыми строками данных. Каждому значению множественного типа в памяти ЭВМ соответствует один двоичный разряд. Например, множество

```
['A', 'B', 'C', 'D']
```

представлено в памяти ЭВМ битовой строкой

```
1 1 1 1.
```

Подмножества этого множества представлены строками:

```
['A','B','D']   1 1 0 1
['B','C']       0 1 1 0
['D']           0 0 0 1
```

Величины множественного типа не могут быть элементами списка ввода - вывода.

В каждой конкретной реализации транслятора с языка Pascal количество элементов базового типа, на ко-

тором строится множество, ограничено. В Pascal количество базовых элементов не должно превышать 256.
Чтобы создать множество с начальными значениями используется конструкция:

```
const
  seLit: Set of 'A'..'D'= [];
```

Пример программы.

```
Program simple_mno;
Var
  iI: intger;
  szS: string;
  seA: set of char;

begin
  seA:=[];
  readln(szS);
  for iI:=1 to length(s) do
  begin
    if not (szS[iI] in seA) then
      include(seA, szS[iI]);
  end;
end.
```

3.2.4 Записи

Часто бывают задачи, в которых требуется обработка логически связанных, но разнотипных данных. В таких случаях мы не можем использовать массивы.

Запись представляет собой совокупность ограниченного числа логически связанных компонент, принадлежащих к разным типам. Компоненты записи называются полями, каждое из которых определяется именем. Поле записи содержит имя поля, вслед за которым через двоеточие указывается тип этого поля. Поля записи могут от-

носиться к любому типу, допустимому в языке Pascal (в том числе и к составному типу), за исключением файлового типа.

Описание записи в языке Pascal осуществляется с помощью служебного слова `Record`, вслед за которым описываются компоненты записи. Завершается описание записи служебным словом `End`.

Для наглядности рассмотрим импользование записей на примере телефонной книги. Книжка является удобной, если в ней предусмотрены разные поля: фамилия, имя, телефон, e-mail, и т.д. Для этого в Pascal будем использовать конструкцию:

```
type
  TAdressBook=Record
                name: String[20];
                numb: String[10]
              end;
var
  AdressBook: TadressBook;
```

Описание записей возможно и без использования имени типа (сразу в разделе `var`), например:

```
var
  AdressBook: Record
                name: String[20];
                numb: String[10]
              end;
```

Это достаточно простая конструкция, которая впринципе может быть заменена массивом (конечно это ухудшит читаемочть кода программы, но тем не менее:

```
Var
```

AdressBook: array[1, 2]of string;, здесь мы полагаем AdressBook[1] – Имя, AdressBook[2] – телефон. Согласитесь, что это достаточно не удобно).

Обращение к записи в целом допускается только в операторах присваивания, где слева и справа от знака присваивания используются имена записей одинакового типа. Во всех остальных случаях оперируют отдельными полями записей. Чтобы обратиться к отдельной компоненте записи, необходимо задать имя записи и через точку указать имя нужного поля, например:

AdressBook.name AdressBook.numb (Замете как нагляден становится код программы)

Такое имя называется составным. Компонентой записи может быть также запись, в таком случае составное имя будет содержать не два, а большее количество имен. И вообще надо сказать, что чаще как рас и используются более сложные конструкции. Ниже в примере это будет рассмотрено.

Обращение к компонентам записей можно упростить, если воспользоваться оператором присоединения with.

Он позволяет заменить составные имена, характеризующие каждое поле, просто на имена полей, а имя записи определить в операторе присоединения:

```
with AdressBook do
begin
  name:='Саня';
  numb:='89606739217';
end;
```

Чтобы при создании записи изначально записать в нее значения, используется конструкция:

```
type
  RecType= Record
             x,y: Word;
             ch: Char;
             dim:   Array[1..3]   of
Byte
           end;

const
  Rec: RecType= ( x: 127; y: 255;
                  ch: 'A';
                  dim: (2, 4, 8) );
```

Пример программы создания Телефонной книги на 100 номеров.

```
Program phone;
Type
  TAdressBook=record
               szName,          szFamil:
string[20];
               szNumb: string[10];
               birth: record
                        szMonth: string;
                        day: 1..31;
                        year:     1900..
2100;
                      end;
            End;
Var
  AdressBook:   array[1..100]of   TAdress-
Book;
```

```
Begin
  {например, чтобы заполнить все поля для
человека, пятого в списке:}
  With AdressBook[5] do
  Begin
    szName:='Александр';
    szFamil:='Пак';
    szNumb:='89609739217';
    birth.szMonth:='Сентябрь';
    birth.day:=22;
    birth.year:=1987;
  End;
End.
```

3.2.5 Файлы

Введение файлового типа в язык Pascal вызвано необходимостью обеспечить возможность работы с периферийными (внешними) устройствами ЭВМ, предназначенными для ввода, вывода и хранения данных.

Файловый тип данных или файл определяет упорядоченную совокупность произвольного числа однотипных компонент.

Общее свойство массива, множества и записи заключается в том, что количество их компонент определено на этапе написания программы, тогда как количество компонент файла в тексте программы не определяется и может быть произвольным.

Понятие файла достаточно широко. Это может быть обычный файл на диске, коммуникационный порт ЭВМ, устройство печати, клавиатура или другие устройства.

При работе с файлами выполняются операции ввода - вывода. Операция ввода означает перепись данных с внешнего устройства (из входного файла) в основную память ЭВМ, операция вывода - это пересылка дан-

ных из основной памяти на внешнее устройство (в выходной файл).

Файлы на внешних устройствах часто называют физическими файлами. Их имена определяются операционной системой. В программах на языке Pascal имена файлов задаются с помощью строк. Например, имя файла надиске может иметь вид:

```
'A:\LAB1.DAT'
‘C:\Windows\Regedit.exe'
'lab3.pas'.
```

Операционная система MS-DOS не делает особого различия между файлами на дисках и лентах и устройствами ЭВМ и портами коммуникаций. В Pascal могут использоваться имена устройств и портов, определенные в MS-DOS, например:

```
'CON', 'LPT1', 'PRN', 'COM1', 'AUX',
'NUL'.
```

Для работы с файлами в программе необходимо определить файловую переменную. Pascal поддерживает три файловых типа: текстовые файлы, компонентные файлы, бестиповые файлы.

Pascal вводит ряд процедур и функций, применимых для любых типов файлов: `Assign, Reset, Rewrite, Close, Rename, Erase, Eof, IOResult.`

Процедура `Assign(var f; FileName: String);`связывает логический файл f с физическим файлом, имя которого задано в строке FileName. Существует два способа написания имени файла: абсолютный и относительный.

В случае, когда используется абсолютный путь всё просто: достаточно указать полный путь к файлу, начиная с буквы раздела, например

```
'C:\Windows\Regedit.exe'
```

Если же используется относительный путь, то необходимо уяснить, что отсчёт ведётся с папки, в которой лежит исполняемый файл (если же вы проводите проверку работоспособности программного кода в компиляторе, то отсчёт ведется с текущей папки компилятора, которую можно заменить в этом смысле на иную при помощи `File\Change dir`…Однако с этим также нужно обращаться внимательно, т.к. придётся перезадать пути для подключаемых модулей и драйверов(`Options\Dirrectories`)).

Процедура `Reset(var f)` открывает логический файл f для последующего чтения данных или, как говорят, открывает входной файл. После успешного выполнения процедуры `Reset` файл готов к чтению из него первого элемента.

Процедура `Rewrite(var f)` открывает логический файл f для последующей записи данных (открывает выходной файл). После успешного выполнения этой процедуры файл готов к записи в него первого элемента.

Процедура `Close(var f)` закрывает открытый до этого логический файл. Вызов процедуры `Close` необходим при завершении работы с файлом. Если по какой-то причине процедура Close не будет выполнена, файл все же будет создан на внешнем устройстве, но содержимое последнего
буфера в него не будет перенесено. Для входных файлов использование оператора закрытия файла необязательно.

Логическая функция `Eof(var f): Boolean` возвращает значение `True`, когда при чтении достигнут конец файла. Это означает, что уже прочитан последний элемент в файле или файл после открытия оказался пуст.

Процедура `Rename(var f; NewName: String)` позволяет переименовать физический файл на диске, связанный с логическим файлом f. Переименование возможно после закрытия файла.

Процедура `Erase(var f)` уничтожает физический файл на диске, который был связан с файловой переменной f. Файл к моменту вызова процедуры Erase должен быть закрыт.

Функция `IOResult: Integer` возвращает целое число, соответствующее коду последней ошибки ввода-вывода. При нормальном завершении операции функция вернет значение 0. Значение функции IOResult необходимо присваивать какой - либо переменной, так как при каждом вызове функция обнуляет свое значение. Функция IOResult работает только при выключенном режиме проверок ошибок ввода - вывода или с ключом компиляции {$I-}.

3.2.5.1 Текстовые файлы

Особое место в языке Pascal занимают текстовые файлы, компоненты которых имеют символьный тип. Для описания текстовых файлов в языке определен стандартный тип Text:

```
Var
  TF1, TF2: Text;
```

Текстовые файлы представляют собой последовательность строк, а строки - последовательность символов. Строки имеют переменную длину, каждая строка завершается признаком конца строки.

С признаком конца строки связана функция `EOLn(var T:Text):Boolean`, где T - имя текстового файла. Эта функция принимает значение `True`, если

достигнут конец строки, и значение `False`, если конец строки не достигнут.

Для работы с текстовыми файлам в Pascal определены следующие процедуры:

`Read(-Ln), Write(-Ln), SetTextBuf, Append, Flush, SeekEOLn, SeekEOF.`

Процедуры `Read(-ln)` и `Write(-Ln)` работают с файлами аналогично работе с дисплеем. Только первым параметром при обращении к этим процедурам следует указать файловую переменную.

Процедура `SetTextBuf( var f: Text; var Buf; BufSize: Word )` служит для увеличения или уменьшения буфера ввода - вывода текстового файла. Значение размера буфера для текстовых файлов по умолчанию равно 128 байтам. Увеличение размера буфера сокращает количество обращений к диску. Рекомендуется изменять размер буфера до открытия файла. Буфер файла начнется с первого байта переменной Buf. Размер буфера задается в необязательном параметре BufSize, а если этот параметр отсутствует, размер буфера определяется длиной переменной Buf.

Процедура `Append(var f: Text)` служит для специального открытия выходных файлов. Она применима к уже существующим физическим файлам и открывает из для дозаписи в конец файла.

Процедура `Flush(var f: Text)` применяется к открытым выходным файлам. Она принудительно записывает данные из буфера в файл независимо от степени его заполнения.

Функция `SeekEOLn(var f: Text): Boolean` возвращает значение `True`, если до конца строки остались только пробелы.

Функция `SeekEOF(var f: Text): Boolean` возвращает значение `True`, если до конца файла остались строки, заполненные пробелами.

3.2.5.2 Компонентные файлы

Компонентный или типизированный файл - это файл с объявленным типом его компонент. Компонентные файлы состоят из машинных представлений значений переменных, они хранят данные в том же виде, что и память ЭВМ.

Описание величин файлового типа имеет вид:

```
Type
  M= File Of T;
```

где `M` - имя файлового типа, `T` - тип компоненты.

Описание файлов можно задавать в разделе описания переменных. Например:

```
Var
  F1: file of integer;
```

Компонентами файла могут быть все типы, кроме файловых. Например:

```
Type
  Tmassive=array[1..34] of char;

Var
  Fchr: File of Tmassive;
```

Для ввода - вывода в компонентные файлы используются процедуры:

```
Read(f, X);
Write(f, X);
```

где f - имя логического файла, X – переменная, тип которой должен совпадать с типом компоненты файла.

Выполнение процедуры `Read(f,X)` состоит в чтении с внешнего устройства одной компоненты файла и запись ее в X. Повторное применение процедуры `Read(f,X)` обеспечит чтение следующей компоненты файла и запись ее в X.

Выполнение процедуры `Write(f,X)` состоит в записи X на внешнее устройство как одной компоненты. Повторное применение этой процедуры обеспечит запись X как следующей компоненты файла.

Для работы с компонентными файлами введена расширенная форма операторов ввода и вывода:

```
Read(f,X1,X2,...XK
Write(f,X1,X2,...XK)
```

Здесь `f` - компонентный файл, а переменные `X1, X2,...XK` должны иметь тот же тип, что и объявленный тип компонент файла `f`.

3.2.5.3 Бестиповые файлы

Бестиповые файлы позволяют записывать на диск произвольные участки памяти ЭВМ и считывать их с диска в память.

Бестиповые файлы объявляются при помощи служебного слова `File`. Например:

```
Var
  Fileator: File;
```

При открытии файла длина буфера устанавливается по умолчанию в 128 байт. Pascal позволяет изменить

размер буфера ввода - вывода, для чего следует открывать файл расширенной записью процедур

Для бестиповых файлов применимы расширенные формы процедур Reset и Rewrite

```
Reset(var f: File; BufSize: Word )
Rewrite(var f: File; BufSize: Word )
```

Параметр BufSize задает число байтов, считываемых из файла или записываемых в него за одно обращение. Минимальное значение BufSize – один байт, максимальное - 64 К байт.

Чтение данных из бестипового файла осуществляется процедурой

```
BlockRead( var f: File; var X; Count: Word; var QuantBlock: Word );
```

Эта процедура осуществляет за одно обращение чтение в переменную X количества блоков, заданное параметром `Count`, при этом длина блока равна длине буфера. Значение `Count` не может быть меньше 1. За одно обращение нельзя прочесть больше, чем 64 К байтов.

Необязательный параметр `QuantBlock` возвращает число блоков (буферов), прочитанных текущей операцией `BlockRead`. В случае успешного завершения операции чтения `QuantBlock` = `Count`, в случае аварийной ситуации параметр `QuantBlock` будет содержать число удачно прочитанных блоков. Отсюда следует, что с помощью параметра `QuantBlock` можно контролировать правильность выполнения операции чтения.

Запись данных в бестиповой файл выполняется процедурой

```
BlockWrite( var f: File; var X;
Count: Word; var QuantBlock: Word );
```

которая осуществляет за одно обращение запись из переменной X количества блоков, заданное параметром `Count`, при этом длина блока равналине буфера.

Необязательный параметр `QuantBlock` возвращает число блоков (буферов), записанных успешно текущей операцией `BlockWrite`.

3.2.5.4 Прямой и последовательный доступ

Смысл последовательного доступа заключается в том, что в каждый момент времени доступна лишь одна компонента из всей последовательности. Для того, чтобы обратиться (получить доступ) к компоненте с номером К, необходимо просмотреть от начала файла К-1 предшествующую
компоненту. После обращения к компоненте с номером К можно обращаться к компоненте с номером К+1. Отсюда следует, что процессы формирования (записи) компонент файла и просмотра (чтения) не могут произвольно чередоваться. Таким образом, файл вначале строится при помощи последовательного добавления компонент в конец, а затем может последовательно просматриваться от начала до конца.

Рассмотренные ранее средства работы с файлами обеспечивают последовательный доступ.

Pascal позволяет применять к компонентным и бестиповым файлам, записанным на диск, способ прямого доступа. Прямой доступ означает возможность заранее определить в файле блок, к которому будет применена операция ввода - вывода. В случае бестиповых файлов блок равен размеру буфера, для компонентных файлов блок - это одна компонента файла.

Прямой доступ предполагает, что файл представляет собой линейную последовательность блоков. Если файл содержит n блоков, то они нумеруются от 1 через 1 до n. Кроме того, вводится понятие условной границы между блоками, при этом условная граница с номером 0 расположена перед блоком с номером 1, граница с номером 1 расположена перед блоком с номером 2 и, наконец, условная граница с номером n находится после блока с номером n.

Реализация прямого доступа осуществляется с помощью функций и процедур `FileSize, FilePos, Seek` и `Truncate`.

Функция `FileSize(var f): Longint` возвращает количество блоков в открытом файле f.

Функция FilePos(var f): Longint возвращает текущую позицию в файле f. Позиция в файле - это номер условной границы. Для только что открытого файла текущей позицией будет граница с номером 0. Это значит, что можно записать или прочесть блок с номером 1. После чтения или записи первого блока текущая позиция переместится на границу с номером 1, и можно будет обращаться к блоку с номером 2. После прочтения последней записи значение `FilePos` равно значению `FileSize`.

Процедура `Seek(var f; N: Longint)` обеспечивает назначение текущей позиции в файле (позиционирование). В параметре N должен быть задан номер условной границы, предшествующей блоку, к которому будет производиться последующее обращение. Например, чтобы работать с блоком 4, необходимо задать значение N, равное 3. Процедура `Seek` работает с открытыми файлами.

Процедура `Truncate(var f)` устанавливает в текущей позиции признак конца файла и удаляет (стирает) все последующие блоки.

3.3 Подпрограммы. (Процедуры, Функции)

Алгоритм решения задачи проектируется путем декомпозиции всей задачи в отдельные подзадачи. Обычно подзадачи реализуются в виде подпрограмм. Это делает код программы более удобочитаемым, более гибким. Такие программы легче редактировать, и в них легче находить ошибки.

Подпрограмма – Фактически самостоятельная программа со своими переменными, со своими типами, константами, а в некоторых случаях еще и со своими подпрограммами, но оформленная отдельно внутри программы. Структура подпрограмм точно такая же как и у программ.

В Pascal существуют два типа подпрограмм - процедуры и функции. Описываются они служебными словами `procedure` и `function` соответственно.

Их принципиальное отличие – Функция возвращает значение, а процедура нет.

3.3.1 Процедуры

`Procedure`

Каждая процедура должна иметь уникальное имя, по которому к ней можно обращаться по ходу программы.

Также существует возможность передачи процедурам параметров. Для этого, при объявлении процедуры в скобках после имени указываются имена переменных и их типы (при вызове процедуры, эти переменные будут *локальными*).

Существует два способа передачи параметров процедурам:

```
Procedure Sample(var x: integer);
```

и

```
Procedure Sample(x: integer);
```

В первом случае, процедуре передаётся лишь ссылка на адрес в памяти, где находится значение переменной (т.е. при её изменении в ходе работы процедуры, по её завершении, изменится значение той переменной, которая была передана; таким образом обязательно указание переменной, а не только лишь значения). Во втором же – процедура создаёт в памяти новую переменную.

Вызывать процедуру следует по её имени, указывая, если это необходимо, параметры (которые должны соответствовать типам, указанным для параметров при объявлении процедуры).

Типы передаваемых параметров – любые (даже процедуры и функции, что, однако, требует применения директив и не будет рассмотрено в рамках данного пособия, так как эта тема требует отдельного освещения).

При вызове процедуры, все переменные, которые создаёт сама процедура, помещаются в стек – специальную область памяти. Если их имена совпадают с именами глобальных переменных – доступ к ним из данной процедуры невозможен, хотя их значения для всей программы остаются неизменными.

Таким образом, существует кардинальное различие между переменными, объявленными в процедурах (они называются локальными) и переменными, объявленными в самой программе в разделе описания переменных (глобальные переменные).

К глобальным переменным имеется доступ из любой части программы, кроме случая совпадения имён переменных процедуры собственно с именами глобальных переменных, о чём было сказано выше.

К локальным переменным доступ имеется лишь только в рамках той процедуры, в которой они объявлены, причём, если имеется случай вложенных процедур, то речь идёт уже о локальных переменных разного уровня

вложенности и для внутренней процедуры переменные внешней будут иметь «глобальный» характер, а для всей программы в целом – «локальный» характер. Об этих особенностях следует помнить при использовании процедур.

```
Program Bugaga;

Var
  iX: integer;

Procedure Free(var iX: integer);
Begin
  iX := 3;
End;

Begin
  Free(iX);
End.
```

3.3.2 Функции

`Function`

Как было сказано выше, функция отличается от процедуры лишь тем, что она возвращает значение.

```
Function    Tretij_Den_Bez_Otdyha_8(
var rSidim: real; iUstali, iOchen: inte-
ger): Boolean;
```

Вышеприведённая запись означает, что функция при вызове возвратит булевское значение.

Механизм можно представить так: внутри функции существует ещё одна переменная, имя которой совпадает с именем функции, также совпадает её тип и тип, возвращаемый функцией. По окончании работы функции будет возвращено значение именно этой переменной.

Вызвать функцию можно следующим образом:

```
Program Demonstrate_Calling_Function;

Var
  iY: integer;

Function Shtopor: Integer;
Begin
  Shtopor := iY*iY;
End;

Begin
  iY := 3;
  iY := Shtopor;
End.
```

3.3.3 Рекурсия

Рекурсия – это такой способ использования процедур, либо функций, при котором процедура/функция вызывает сама себя напрямую, либо посредством других процедур/функций.

Наглядно объяснить смысл рекурсии можно на примере использования толкового словаря. Когда Вы встречаете непонятное слово Вы пользуетесь толковым словарём (вызов функции). Вы находите в нём непонятное слово и читаете его описание. Если все слова описания понятны – вы заканчиваете работу со словарём (работа с функцией закончена). Однако возможен случай, когда некоторые слова в описании будут непонятны. Тогда Вы оставляете закладку на странице и ищете в словаре непонятное в описании слово (вызов функцией самой себя). Однако и в его описании может встретиться непонятное слово и Вы, снова оставив закладку, начнёте искать уже

это слово в словаре. В конце концов, Одно из определений в итоге не вызовет у Вас никаких сложностей и Вы вернётесь к предыдущему и т.д. до тех пор, пока все слова в первом определении не станут ясны. Тогда необходимый результат достигнут (возврат функцией конечного значения).

Аналогично обстоит дело и в программировании, достаточно только определить рекурентную функцию и условие, при котором она будет себя вызывать. К тому же, в отличие от случая со словарём, Вам не нужно ставить «закладки» - об этом уже позаботились разработчики среды программирования. Каждая новая «копия» процедуры/функции работает независимо от предыдущей, а переменные остаются в стеке.

Составим программу, которая рекурсивно вычисляет факториал числа. Для начала надо определить функцию:

Мы знаем, что факториал числа N есть факториал числа N-1 умноженного на N, а именно N!=N(N-1)!. Пусть f(N)=N!, тогда можно записать, что f(N)=f(N-1)*N.

Теперь нужно определить условие, при котором фукнция будет обращаться к самой себе по выведенному нами правилу: Эта рекуррентная последовательность ведет к уменьшению аргумента, и мы знаем, что по определению 1!=1, 0!=1, отсюда условие: если N>1 тогда вызвать функцию, в противном случае значение функции 1.

```
Program simple_recurs;
Function fuck(iN: integer): longint;
Begin
  If (iN=0) or (iN=1) then fuck:=1
  Else fuck:=iN*fuck(iN-1);
End;
Begin
  Writeln(fuck(6));
```

```
End.
```

Применяется рекурсия достаточно часто. Например, когда речь идет о любых рекурентных последовательностях, оптимальный вариант – рекурсия.

3.4 Указатели. Динамические переменные

Операционная система MS - DOS все адресуемое пространство делит на сегменты. Сегмент - это участок памяти размером 64 К байт. Для задания адреса необходимо определить адрес начала сегмента и смещение относительно него.

В Pascal определен адресный тип Pointer - указатель. Переменные типа Pointer

```
var
  p: Pointer;
```

содержат адрес какого - либо элемента программы и занимают 4 байта, при этом адрес хранится как два слова, одно из них определяет сегмент, второе - смещение.

Переменную типа указатель можно описать другим способом.

```
type
  NameType= ^T;

var
  p: NameType;
```

Здесь p - переменная типа указатель, связанная с типом T с помощью имени типа NameType. Описать переменную типа указатель можно непосредственно в разделе описания переменных:

```
var
  p: ^T;
```

Необходимо различать переменную типа указатель и переменную, на которую этот указатель ссылается. Например если p - ссылка на переменную типа T, то p^ - обозначение этой самой переменной.

Для переменных типа указатель введено стандартное значение NIL, которое означает, что указатель не ссылается ни к какому объекту. Константа NIL используется для любых указателей.

Над указателями не определено никаких операций, кроме проверки на равенство и неравенство.

Переменные типа указатель могут быть записаны в левой части оператора присваивания, при этом в правой части может находиться либо функция определения адреса Addr(X), либо выражение @ X, где @ - унарная операция взятия адреса, которая имеет в выражениях приоритет унарного минуса, т.е. самый высокий. X - имя переменной любого типа, в том числе процедурного.

Переменные типа указатель не могут быть элементами списка ввода-вывода.

Процедуры, определённые над указателями:

`New, Dispose, GetMem, FreeMem, Mark, Release, MaxAvail, MemAvail, SizeOf`.

Процедура `New(var p: Pointer)` выделяет место в динамической области памяти для размещения динамической переменной p^ и ее адрес присваивает указателю p.

Процедура `Dispose(var p: Pointer)` освобождает участок памяти, выделенный для размещения динамической переменной процедурой `New`, и значение указателя p становится неопределенным.

Проуедура `GetMem(var p: Pointer; size: Word)` выделяет участок памяти, присваивает адрес его

начала указателю p, размер участка в байтах задается параметром size.

Процедура `FreeMem(var p: Pointer; size: Word)` освобождает участок памяти, адрес начала которого определен указателем p, а размер параметром `size`. Значение указателя p становится неопределенным.

Процедура `Mark(var p: Pointer)` записывает в указатель p адрес начала участка свободной динамической памяти на момент ее вызова.

Процедура `Release(var p: Pointer)` освобождает участок динамической памяти, начиная с адреса, записанного в указатель p процедурой `Mark`, то-есть, очищает ту динамическую память, которая была занята после вызова процедуры `Mark`.

Функция `MaxAvail: Longint` возвращает длину в байтах самого длинного свободного участка динамической памяти.

Функция `MemAvail: Longint` полный объем свободной динамической памяти в байтах.

Вспомогательная функция `SizeOf(X): Word` возвращает объем в байтах, занимаемый X, причем X может быть либо именем переменной любого типа, либо именем типа.

Рассмотрим некоторые примеры работы с указателями.

```
var
  p1, p2: ^Integer;
```

Здесь p1 и p2 - указатели или переменные ссылочного типа.

```
p1:=NIL;
p2:=NIL;
```

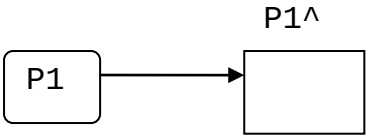
После выполнения этих операторов присваивания указатели p1 и p2 не будут ссылаться ни на какой конкретный объект.

```
New(p1);
New(p2);
```

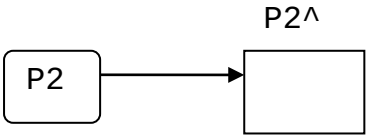
Процедура `New(p1)` выполняет следующие действия:

-в памяти ЭВМ выделяется участок для размещения величины целого типа;

-адрес этого участка присваивается переменной p1:



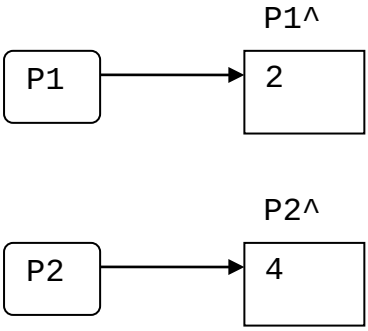
Аналогично, процедура `New(p2)` обеспечит выделение участка памяти, адрес которого будет записан в p2:



После выполнения операторов присваивания

```
p1^:=2;
p2^:=4;
```

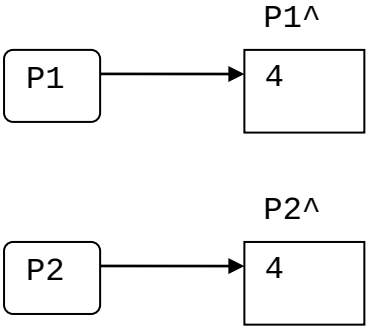
в выделенные участки памяти будут записаны значения 2 и 4 соответственно:



В результате выполнения оператора присваивания

```
p1^:=p2^;
```

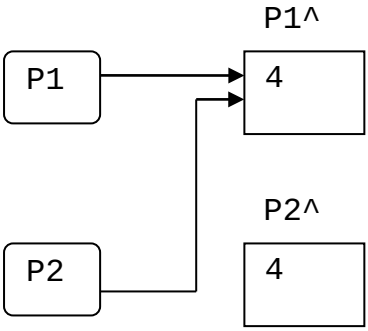
в участок памяти, на который ссылается указатель p1, будет записано значение 4:



После выполнения оператора присваивания

```
p2:=p1;
```

оба указателя будут содержать адрес первого участка памяти:



Переменные p1^, p2^ являются динамическими, так как память для них выделяется в процессе выполнения программы с помощью процедуры New.

Динамические переменные могут входить в состав выражений, например:

```
p1^:=p1^+8;
Write('p1^=',p1^:3);
```

Пример. В результате выполнения программы:

```
Program DemoPointer;
Var
  p1, p2, p3:^Integer;
begin
  p1:=NIL;
  p2:=NIL;
  p3:=NIL;
  New(p1);
  New(p2);
  New(p3);
```

```
        p1^:=2;
        p2^:=4;
        p3^:=p1^+Sqr(p2^);
        writeln('p1^=', p1^, ' p2^=' ,p2^,
' p3^=', p3^);
        p1:=p2;
        writeln('p1^=',p1^,'  p2^=',p2^)
      end.
```

на экран дисплея будут выведены результаты:

```
p1^=2 p2^=4 p3^=18
p1^=4 p2^=4
```

Работа с динамическими переменными через указатели требует большей тщательности и аккуратности при проектировании программ. В частности, следует стремиться освобождать выделенные области сразу же после того, как необходимость в них отпадает, иначе "засорение" памяти ненужными динамическими переменными может привести к быстрому ее исчерпанию.

Кроме того, необходимо учитывать еще одну проблему, связанную с противоречием между стековым принципом размещения статических переменных и произвольным характером создания и уничтожения динамических переменных. Рассмотрим следующий схематический пример программы:

```
      type
        PPerson=^Person;
        Person= record
                  ...
                end;
      procedure GetPerson;
      var
        P:PPerson;
```

```
begin
  new(P)
end;

begin
  Writeln(MemAvail);
  GetPerson;
  Writeln(MemAvail)
end.
```

Вызов New в процедуре GetPerson приводит к отведению памяти для динамической переменной типа Person. Указатель на эту переменную присваивается переменной P. Рассмотрим ситуацию, возникающую после выхода из процедуры GetPerson. По правилам блочности все локальные переменные подпрограммы перестают существовать после ее завершения. В нашем случае исчезает локальная переменная P. Но, с другой стороны, область памяти, отведенная в процессе работы `GetPerson`, продолжает существовать, так как освободить ее можно только явно, посредством процедуры `Dispose`. Таким образом, после выхода из GetPerson отсутствует какой бы то ни было доступ к динамической переменной, так как единственная "ниточка", связывающая ее с программой, - указатель P - оказался потерянным при завершении `GetPerson`. Вывод на печать общего объема свободной памяти до и после работы `GetPerson` подтверждают потерю определенной области.

Pascal, как и многие другие языки программирования, не имеет встроенных средств борьбы с засорением памяти неиспользуемыми динамическими переменными. Во всяком случае, нужно придерживаться правила, согласно которому при выходе из блока необходимо или освободить все созданные в нем динамические переменные, или сохранить каким-то образом ссылки на них

(например, присвоив эти ссылки глобальным переменным).

К описанной проблеме примыкает коллизия другого рода, заключающаяся в ситуации, когда некоторая область памяти освобождена, а в программе остался указатель на эту область. Например, пусть ссылка p указывает на элемент списка, и был выполнен оператор `p:=nil;` или `dispose(p).` Несмотря на это, можно (неправильно) использовать далее в программе выражение `p^.next`, но его значение непредсказуемо.

3.4.1 Применение динамических переменных. Динамические структуры данных

Структурированные типы данных, такие как массивы, множества и записи иногда бывают неудобными в использовании, потому что для них память выделяется сразу при запуске программы, она не меняется и ее изначальный размер ограничен 64КБ, чего иногда для выполнения задачи оказывается недостаточно.

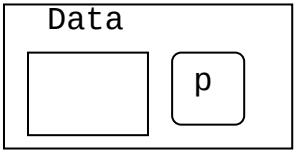
Часто требуется, чтобы структуры данных могли менять свои размеры по ходу программы и могли содержать данные больше чем на 64КБ. Для этого используют динамические переменные. Смысл заключается в том, что в динамическую переменную, помимо данных, которые требуется хранить, записывается адрес другой ячейки динамической структуры. Конечно, это усложняет работу с данными, но тем не мене иногда это бывает полезным.

Каждая компонента любой динамической структуры представляет собой запись, содержащую, по крайней мере, два поля: одно поле типа указатель, а второе - для размещения данных. В общем случае запись может содержать не один, а несколько указателей и несколько полей данных. Поле данных может быть переменной, массивом, множеством или записью.

3.2.1.1 Линейные динамические структуры данных

В том случае, когда каждая компонента содержит ссылку только на одну следующую компоненту, конструкция называется линейной.

Рассмотрения отдельную компоненту в виде:



где поле `p` - указатель; поле `Data` - данные.

Описание этой компоненты дадим следующим образом:

```
type
  Pointer = ^Comp;
  Comp=record
    D: T;
    pNext: Pointer
  end;
```

здесь T – Любой тип данный предусмотренный в Pascal.
Как мы видим, тип `Pointer` задается рекуррентно, поэтому работать с такими конструкциями проще всего рекурсивными процедурами.

Однако, в случае использования рекурсии, вряд ли можно будет добиться оптимизации памяти и поэтому следует прибегать к рекурсивным процедурам только тогда, когда их использование обусловлено алгоритмом, а не малыми объёмами доступной памяти.

Теперь познакомимся с отдельными приемами работы с линейными структурами данных.

3.4.1.1.1 Стеки

Самая простая обработка линейных динамических структур – обработка по принципу стека:

```
LIFO (Last-In, First-Out) -
```

«поступивший последним, обслуживается первым».
В этом случае добавление компоненты и исключение компоненты производится из одного конца, который называется вершиной стека. А саму линейную конструкцию будем называть стеком.

Рассмотрим три основные операции, проделываемые со стеками:

-формирование стека (запись первой компоненты);

-добавление компоненты в стек;

-выборка компоненты (удаление).

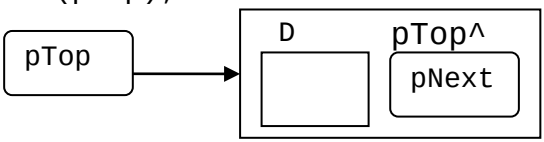
Для формирования стека и работы с ним необходимо иметь две переменные типа указатель, первая из которых определяет вершину стека, а вторая - вспомогательная. Пусть описание этих переменных имеет вид:

```
type
  Comp=^PComp;
  PComp=record
          D: SomeType;
          pNext: Comp;
        End;

var
  pTop, pAux: Comp;
```

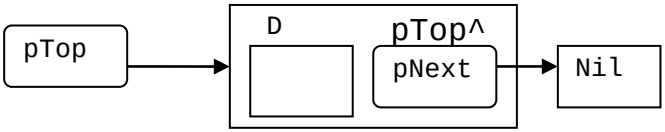
где `pTop` - указатель вершины стека;
`pAux` - вспомогательный указатель.

Начальное формирование стека выполняется следующим образом:

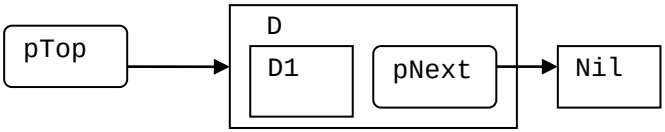
```
New(pTop);
```




```
pTop^.pNext:=NIL;
```



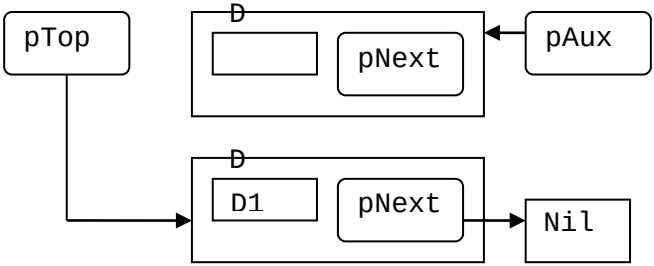

```
pTop^.D:=D1;
```



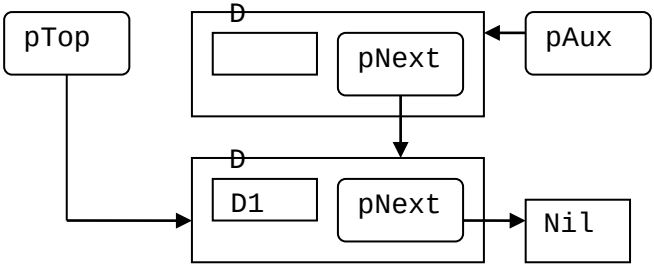
Последний оператор или группа операторов записывает содержимое поля данных первой компоненты.

Добавление компоненты в стек производится с использованием вспомогательного указателя:

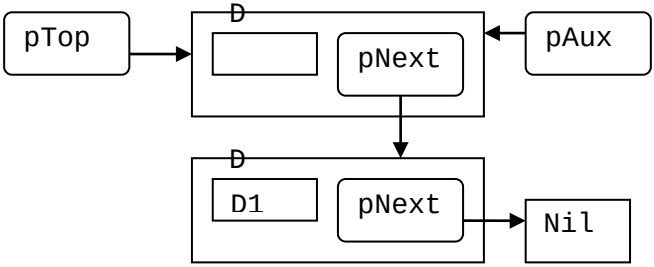
```
New(pAux);
```



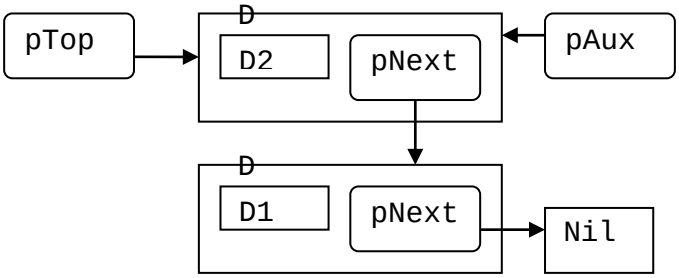

```
pAux^.pNext:=pTop
```




```
pTop:=pAux;
```

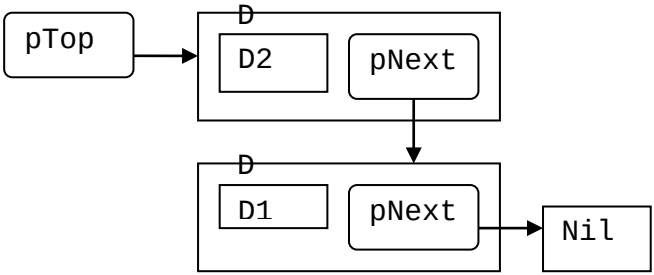



```
pTop^.D:=D2;
```



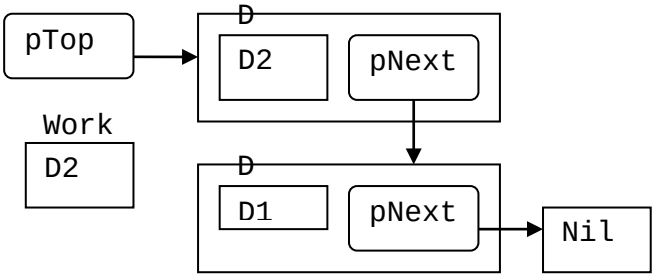
Добавление последующих компонент производится аналогично.

Рассмотрим процесс выборки компонент из стека. Пусть к моменту начала выборки стек содержит две компоненты:

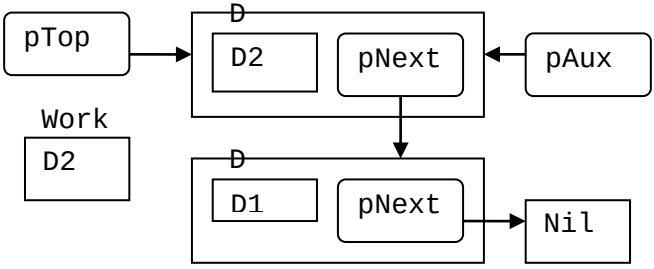


Сначала производится обработка, данных конца стека. Затем нужно Указать новую вершину стека, и позаботиться о том чтобы старая была удалена.

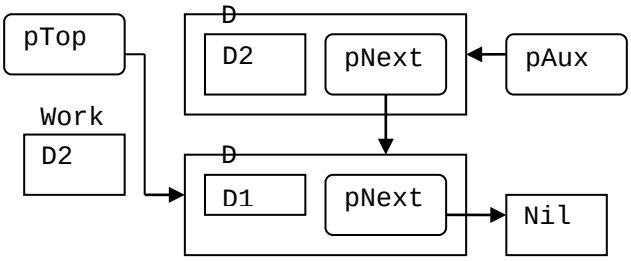
```
Work:=pTop^.D;
```



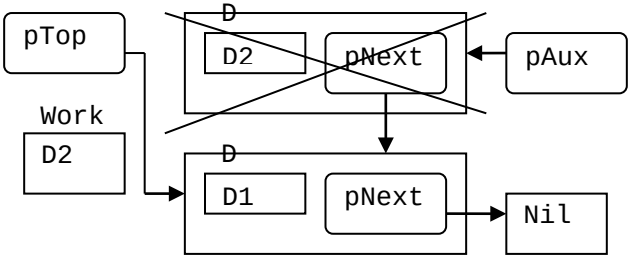

```
pAux:=pTop;
```




```
pTop:=pTop^.pNext;
```




```
dispose(pAux);
```



Как видно из рисунка, при чтении компонента удаляется из стека. Ни в коем случае нельзя забывать о `Dispose`, иначе память будет исчерпана неиспользуемыми значениями, к которым нет доступа.

Пример. Составить программу, которая формирует стек, добавляет в него произвольное количество компонент, а затем читает все компоненты
и выводит их на экран дисплея, В качестве данных взять строку символов. Ввод данных - с клавиатуры дисплея, признак конца ввода – строка символов END.

```
Program STACK;

uses
  Crt;
```

```
      type
        Alfa=String[10];
        PComp=^Comp;
        Comp=Record
               sD: Alfa;
               pNext: PComp
             end;

      var
        pTop: PComp;
        sC: Alfa;

      Procedure   CreateStack(var   pTop:
PComp; var sC: Alfa);
      begin
        New(pTop);
        pTop^.pNext:=NIL;
        pTop^.sD:=sC
      end;


      Procedure  AddComp(var  pTop:  PComp;
var sC: Alfa);
      var
        pAux: PComp;
      begin
        NEW(pAux);
        pAux^.pNext:=pTop;
        pTop:=pAux;
        pTop^.sD:=sC
      end;

      Procedure  DelComp(var  pTop:  PComp;
var sC:ALFA);
      var
        pAux: PComp;
```

```
begin
  pAux:=pTop;
  sC:=pTop^.sD;
  pTop:=pTop^.pNext
  dispose(pAux);
end;

begin
  Clrscr;
  writeln('  ВВЕДИ СТРОКУ ');
  readln(sC);
  CreateStack(pTop,sC);
  repeat
    writeln('  ВВЕДИ СТРОКУ ');
    readln(sC);
    AddComp(pTop,sC)
  until sC='END';
  writeln('****** ВЫВОД  РЕЗУЛЬТАТОВ
******');
  repeat
    DelComp(pTop,sC);
    writeln(sC);
  until pTop = NIL
end.
```

3.4.1.1.2 Очереди

Использование динамических структур по принципу очереди:

```
FIFO (First-In, First-Out) -
```

«поступивший первым, обслуживается первым».

В этом случае добавление компонент происходит с одного конца, а выборка осуществляется с другого. Линейная конструкция при этом называется очередбю.

Для формирования очереди и работы с ней необходимо иметь три переменные типа указатель, первая из которых определяет начало очереди, вторая - конец очереди, третья - вспомогательная.

Описание компоненты очереди и переменных типа указатель дадим следующим образом:

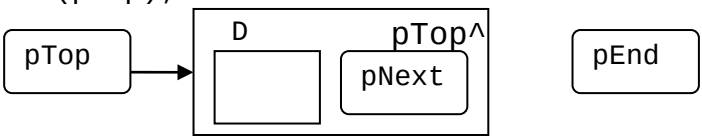
```
type
  Comp=^PComp;
  PComp=record
          D: SomeType;
          pNext: Comp
        end;
var
  pTop, pEnd, pAux: PComp;
```

где pTop - указатель начала очереди, pEnd - указатель конца очереди, pAux - вспомогательный указатель.

Тип SomeType определяет тип данных компоненты очереди.

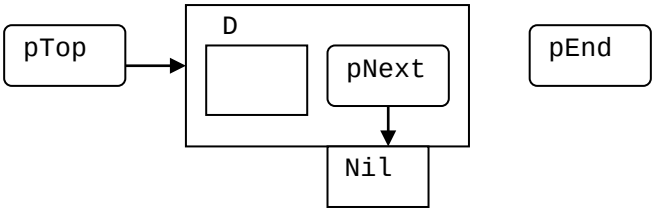
Начальное формирование очереди выполняется следующим образом:

```
New(pTop);
```

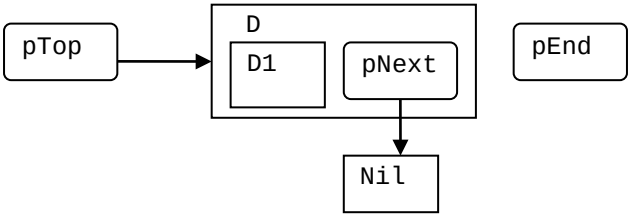



```
pTop^.pNext:=NIL;
```

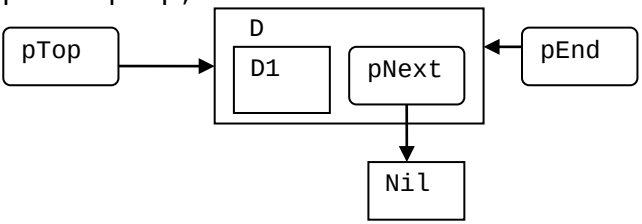
pTop^



pTop^.D:=D1;

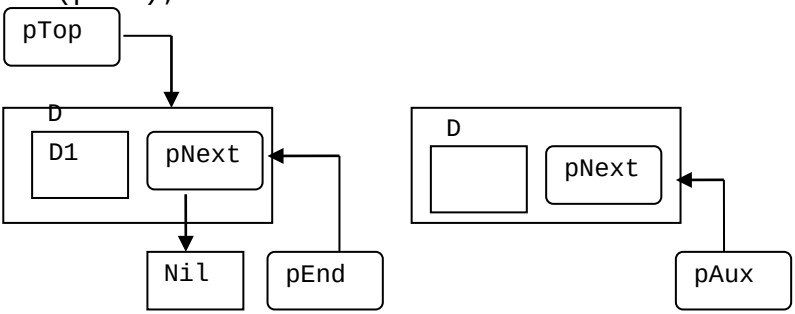


pEnd:=pTop;

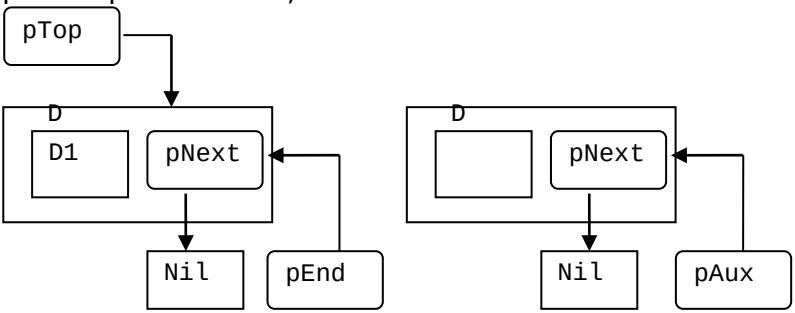


Добавление компоненты в очередь производится в конец очереди:

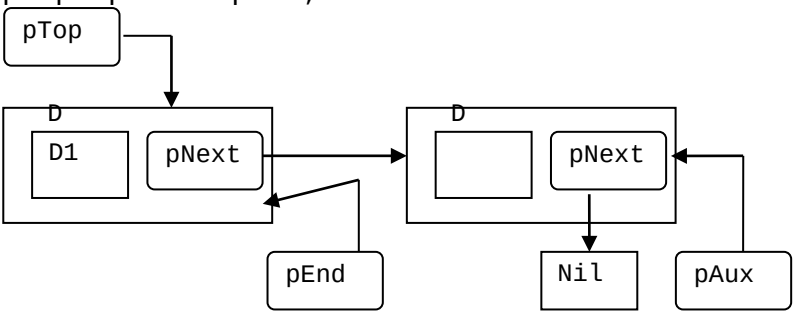
New(pAux);



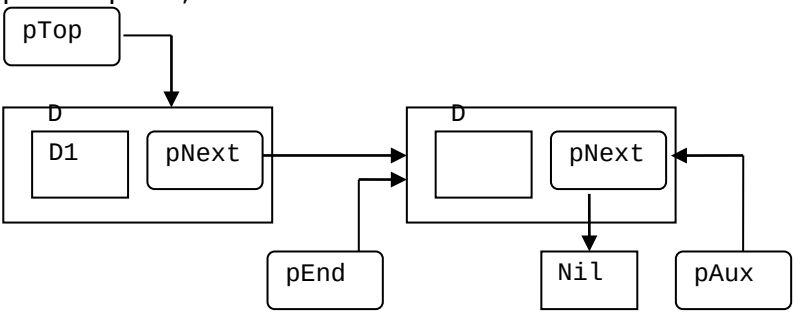
pAux^.pNext:=NIL;



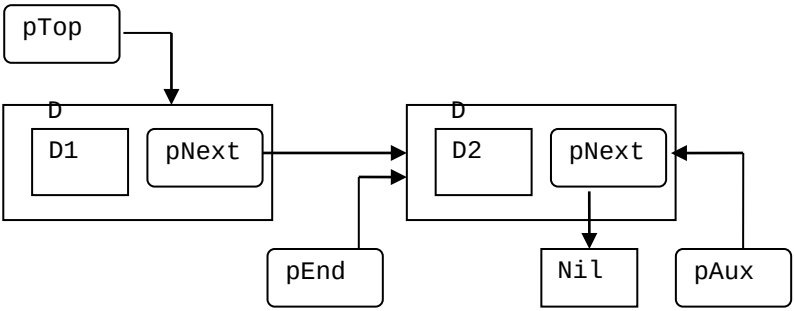
pTop^.pNext:=pAux;



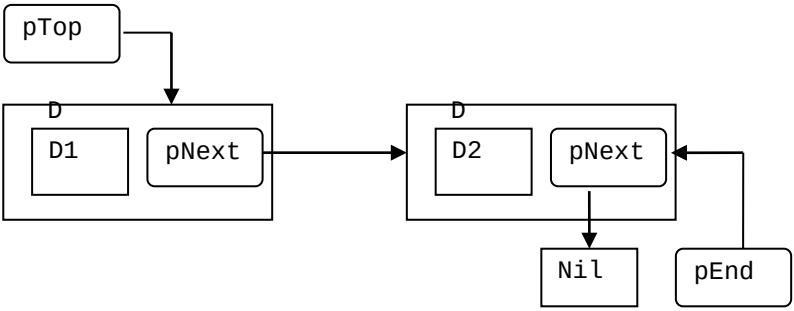
pEnd:=pAux;



pEnd^.D:=D2;

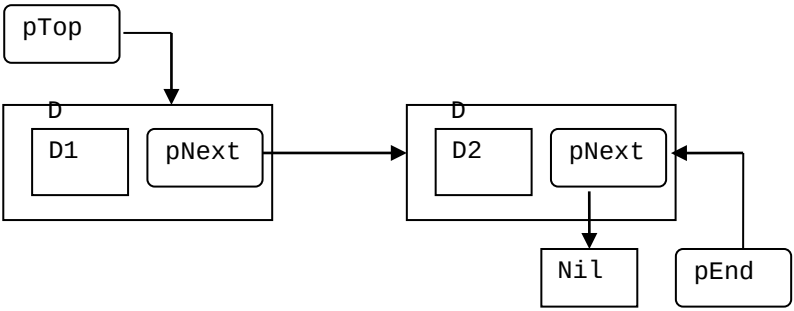


В итоге имеем:



Добавление последующих компонент производится аналогично.

Выборка компоненты из очереди осуществляется из начала очереди, одновременно компонента исключается из очереди. Пусть в памяти ЭВМ сформирована очередь, состоящая из двух элементов:



Выборка компоненты выполняется таким же образом как и для стека:

```
Work:=pTop^.D;
pAux:=pTop;
pTop:=pBop^.pNext;
dispose(pAux);
```

Пример. Составить программу, которая формирует очередь, добавляет в нее произвольное количество компонент, а затем читает все компоненты и выводит их на экран исплея. В качестве данных взять строку символов. Ввод данных - с клавиатуры дисплея, признак конца ввода - строка символов END.

```
Program QUEUE;

uses
  Crt;

type
  Alfa=String[10];
  PComp=^Comp;
  Comp=record
         sD:Alfa;
         pNext:PComp
```

```
            end;
      var
        pTop, pEnd: PComp;
        sC: Alfa;

      Procedure    CreateQueue(var    pTop,
pEnd: PComp; var sC: Alfa);
      begin
        New(pTop);
        pTop^.pNext:=NIL;
        pTop^.sD:=sC;
        pEnd:=pTop
      end;

      Procedure  AddQueue(var  pEnd:PComp;
var sC:Alfa);
      var
        pAux: PComp;

      begin
        New(pAux);
        pAux^.pNext:=NIL;
        pEnd^.pNext:=pAux;
        pEnd:=pAux;
        pEnd^.sD:=sC
      end;

      Procedure  DelQueue(var  pTop:  PComp;
var sC: Alfa);
      var
        pAux: pComp;
      begin
        sC:=pTop^.sD;
        pAux:=pTop;
        pTop:=pTop^.pNext;
        dispose(pAux);
```

```
end;

begin
  Clrscr;
  writeln(' ВВЕДИ СТРОКУ ');
  readln(sC);
  CreateQueue(pTop,pEnd,sC);
  repeat
    writeln(' ВВЕДИ СТРОКУ ');
    readln(sC);
    AddQueue(pEnd,sC)
  until sC='END';
  writeln('  *****  ВЫВОД  РЕЗУЛЬТАТОВ
*****');
  repeat
    DelQueue(pTop,sC);
    writeln(sC);
  until pTop=NIL
end.
```

3.4.1.1.3 Списки

Работа по принципу списка – самая гибкая обработка линейных динамических структур. В этом случае добавление и выборка компонент производится по ключу. Ключ – какая-либо отличительная черта ячейки. При такой обработке линейную структуру будем называть Связанный список, или просто список.

Для формирования списка и работы с ним необходимо иметь пять переменных типа указатель, первая из которых определяет начало списка, вторая - конец списка, остальные вспомогательные.

Описание компоненты списка и переменных типа указатель дадим следующим образом:

```
type
  PComp= ^Comp;
  Comp= record
          D: SomeType;
          pNext: PComp
        end;
 var
    pTop,  pEnd,  pCKey,  pPreComp,
pAux: PComp;
```

где pTop - указатель начала списка, pEnd - указатель конца списка,

pCKey, pPreComp, pAux - вспомогательные указатели.

Начальное формирование списка, добавление компонент в конец списка выполняется так же, как и при формировании очереди.

Для чтения и вставки компоненты по ключу необходимо выполнить поиск компоненты с заданным ключом:

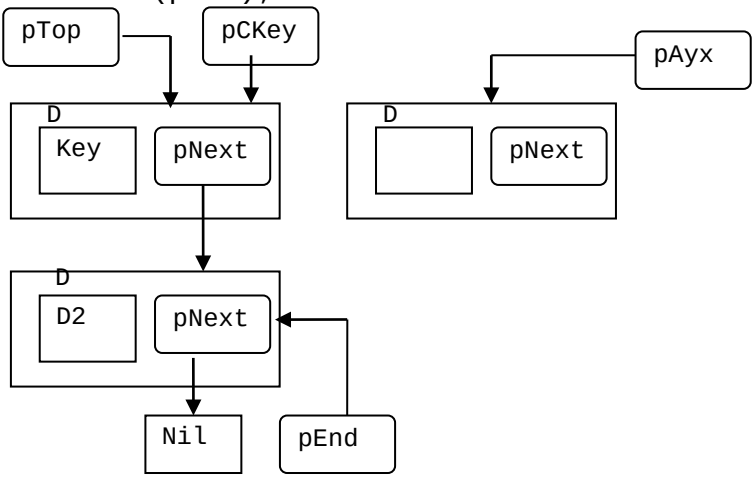
```
pCKey:=pTop;
while (pCKey<>nil) and (Key<>pCKey^.D) Do
  pCKey:=pCKey^.pNext;
```

Здесь Key - ключ, тип которого совпадает с типом данных компоненты, или с типом компоненты, специально отведенным под ключ.

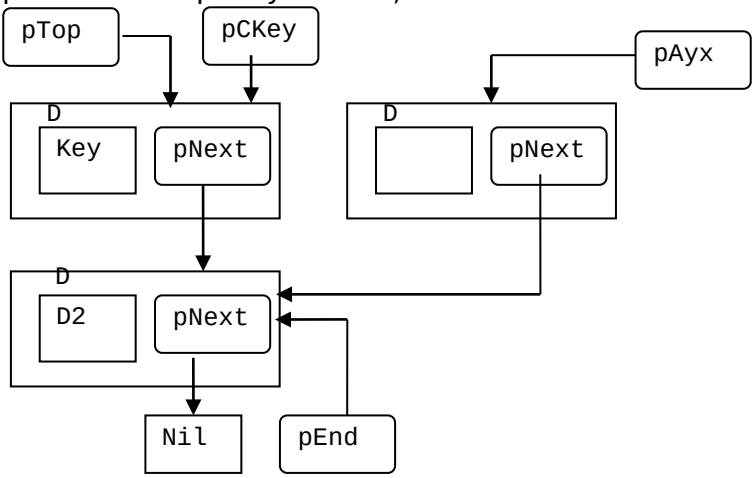
После выполнения этих операторов указатель pCKey будет определять компоненту с заданным ключом или такая компонента не будет найдена (`pCKey` будер равным `nil`).

Пусть pCKey определяет компоненту с заданным ключом. Вставка новой компоненты после нее выполняется следующим образом:

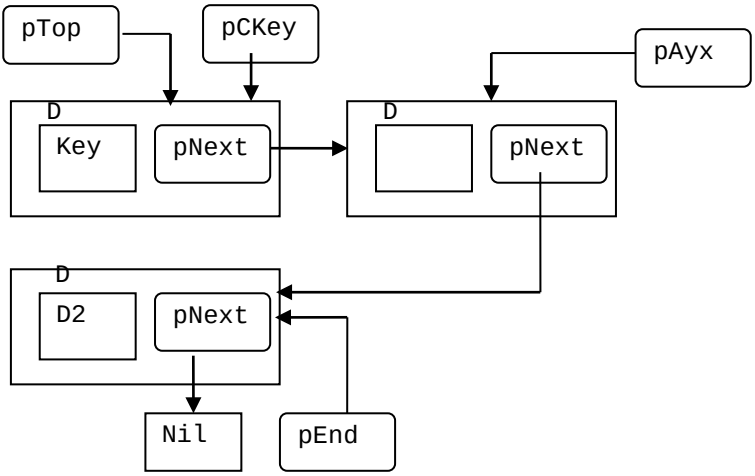
New(pAux);



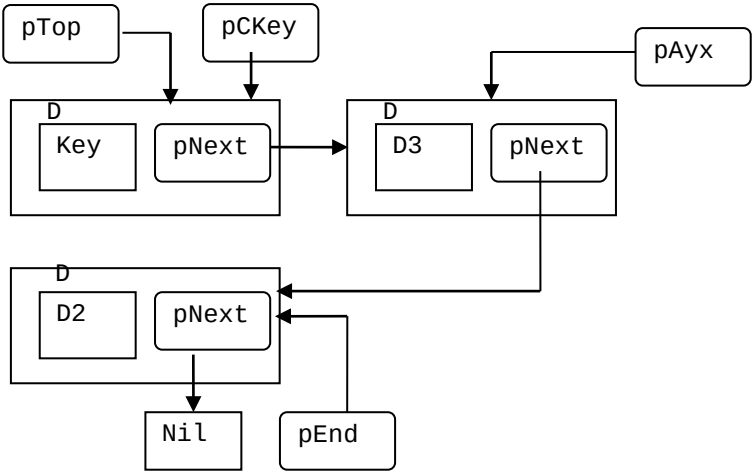
pAux^.nex:=pCKey^.next;



pCKey^.next:=pAux;




```
pAux^.D:=D3;
```



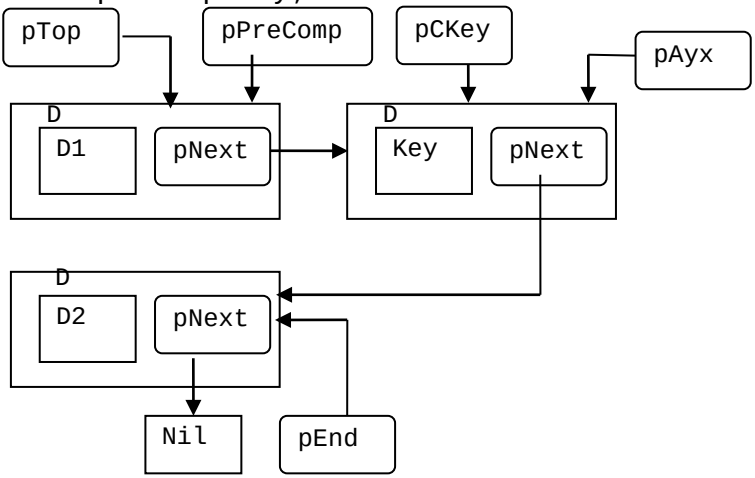
Для удаления компоненты с заданным ключом необходимо при поиске нужной компоненты помнить адрес предшествующей:

```
      pCKey:=pTop;
      while        (pCKey<>NIL)        and
(Key<>pCKey^.D) do
      begin
        pPreComp:=pCKey;
        pCKey:=pCKey^.pNext
      end;
```

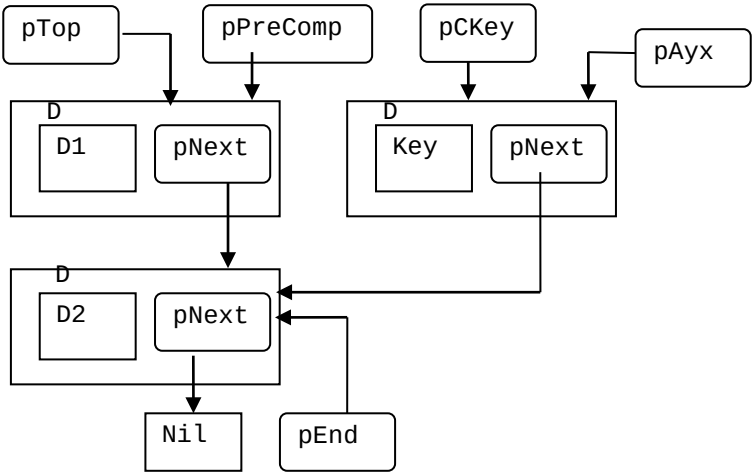
Здесь указатель `pCKey` определяет компоненту с заданным ключом, указатель `pPreComp` содержит адрес предидущей компоненты.

Удаление компоненты с ключом Key выполняется следующим образом:

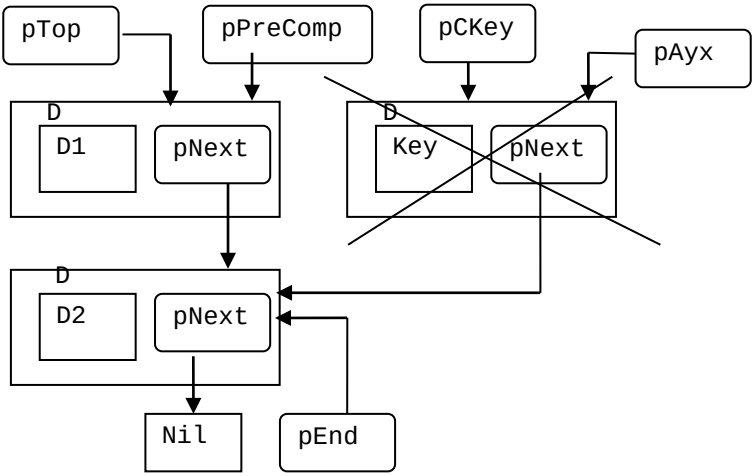
```
      pAux:=pCKey;
```




```
      pPreComp^.pNext:=pCKey^.next;
```




```
dispose(pAux);
```



Пример. Составить программу, которая формирует список, добавляет в него произвольное количество компонент, выполняет вставку и удаление компоненты по ключу, а затем читает и выводит весь список на экран дисплея. В качестве данных взять строку символов. Ввод

данных – с клавиатуры дисплея, признак конца ввода - строка символов END.

```
Program LISTLINKED;
uses
  Crt;
type
  Alfa= String[10];
  PComp=^Comp;
  Comp= record
          sD:Alfa;
          pNext:PComp
        end;
var
  pTop, pEnd, pAux, pCKey, pPreComp:
PComp;
  sC, sKey: Alfa;
  bCond: Boolean;


Procedure  CreateLL(var  pTop,pEnd:
PComp; var sC: Alfa);
begin
  New(pTop);
  pTop^.pNext:=NIL;
  pTop^.sD:=sC;
  pEnd:=pTop
end;

Procedure AddLL(var pEnd: PComp; var
sC: Alfa);
var
  pAux: PComp;
begin
  New(pAux);
  pAux^.pNext:=NIL;
```

```
        pEnd^.pNext:=pAux;
        pEnd:=pAux;
        pEnd^.sD:=sC
      end;

      Procedure Find(var sKey: Alfa; var
pTop,pCKey,pPreComp: PComp;
                        var bCond: Boole-
an);
      begin
        pCKey:=pTop;
        while (pCKey <> NIL) and (sKey <>
pCKey^.D) do
        begin
          pPreComp:=pCKey;
          pCKey:=pCKey^.pNext
        end;
        if (pCKey = NIL) and (sKey <> pCK-
ey^.sD) then bCond:=False

else bCond:=True
      end;

      Procedure InsComp(var sKey,sC: Al-
fa);
      var
        pAux:PComp;

      begin

Find(sKey,pTop,pCKey,pPreComp,bCond);
        New(pAux);
        pAux^.sD:=sC;
        pAux^.pNext:=pCKey^.pNext;
        pCKey^.pNext:=pAux
      end;
```

```
     Procedure  DelComp(var  sKey:  Alfa;
var pTop: PComp);
     Var
       pAux: pComp;
     begin

Find(sKey,pTop,pCKey,pPreComp,bCond);
       pAux:=pCKey;
       pPreComp^.pNext:=pCKey^.pNext
       dispose(pAux);
     end;

     begin
       ClrScr;
       writeln('  ВВЕДИ СТРОКУ ');
       readln(sC);
       CreateLL(pTop,pEnd,sC);
       repeat
         writeln('ВВЕДИ СТРОКУ ');
         readln(sC);
         AddLL(pEnd,sC)
       until sC='END';
       writeln('  *****  ВЫВОД  ИСХОДНОГО
СПИСКА *****');
       pAux:=pTop;
       repeat
         writeln(pAux^.sD);
         pAux:=pAux^.pNext;
       until pAux=NIL;
       writeln;
       writeln('ВВЕДИ  КЛЮЧ  ДЛЯ  ВСТАВКИ
СТРОКИ');
       readln(sKey);
       writeln('ВВЕДИ  ВСТАВЛЯЕМУЮ  СТРО-
КУ');
```

```
        readln(sC);
        InsComp(sKey,sC);
        writeln;
        writeln('ВВЕДИ    КЛЮЧ    УДАЛЯЕМОЙ
СТРОКИ');
        readln(sKey);
        DelComp(sKey,pTop);
        writeln;
        writeln('  *****  ВЫВОД  ИЗМЕНЕННОГО
СПИСКА *****');
        pAux:=pTop;
        repeat
           writeln(pAux^.sD);
           pAux:=pAux^.pNext;
        until pAux=NIL
      end.
```

3.4.1.1.4 Циклические списки

Циклически связанный список (сокращенно- циклический список) – ещё один принцип использования линейных динамических структур данных. Отличается он от простого списка только тем, что его последняя ячейка ссылается не на nil, а на первую. В этом случае можно получить доступ к любому элементу, находящемуся в списке, отправляясь от любой заданной точки; одновременно достигается также полная симметрия, и теперь уже не приходится различать в списке "последний" или "первый" узел.

Знаний об использовании линейных динамических структур данных по принципу списка вполне достаточно, чтобы организовать работу по принципу циклического списка. Стоит отметить, что для работы по этому принципу вполне достаточно четырёх переменных, одна из кото-

рых – указатель на какую-либо компоненту списка, а три другие вспомогательные.

```
type
  PComp= ^Comp;
  Comp= record
          D: SomeType;
          pNext: PComp
        end;
var
   pTop, pCKey, pPreComp, pAux: PComp;
```

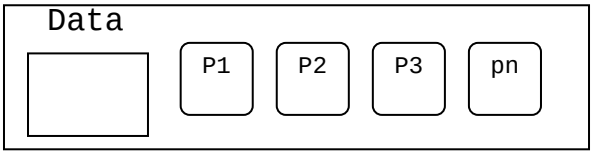
Также отметим, что при обходе такого списка его концом можно считать ту ячейку, с которой начался его обход.

Предлагается самостоятельно составить программу, аналогичную рассмотренной в предыдущем примере, используя циклические списки.

3.4.1.2 Нелинейные динамические структуры

В том случае, когда каждая компонента динамической структуры данных содержит более одной ссылки на другие её компоненты, такая структура является нелинейной.

Рассмотрения отдельную компоненту в виде:



В общем случае эту ячейку можно описать следующим образом:

```
type
  PComp= ^Comp;
```

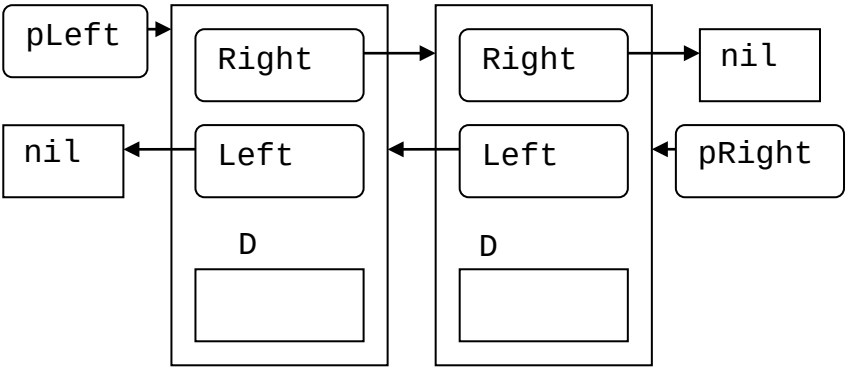
```
Comp= record
        D: SomeType;
        p: array [1..n] of PComp
      end;
```

Общий случай использования таких структур слишком сложен (хотя бы одно название «Гиперобъёмное циклическое пространство» - говорит само за себя) для краткого его описания, поэтому предлагается ознакомиться с ним самостоятельно. Мы же будем рассматривать только бинарные (содержит только два указателя на другие ячейки) нелинейные структуры, а именно список с двумя связями и бинарное дерево.

3.4.1.2.1 Списки с двумя связями

Двусвязный список - это такой принцип использования бинарных нелинейных динамических структур данных, при котором каждая ячейка имеет ссылку как на следующую, так и на предыдущую ячейку (собственно поэтому в названии фигурирует слово «двусвязный»), что даёт возможность перемещаться по списку как в одну, так и в другую сторону.

```
type
  PComp= ^Comp;
  Comp= record
          D: SomeType;
          Left, right: PComp
        end;
var
  pLeft, pRight: pComp;
```



Выше представлена конструкция и организация двусвязного списка. Предлагается самостоятельно освоить их.

* Задание. Организовать циклический двусвязный список.

3.4.1.2.2 Деревья

Все вышеперечисленные структуры применяются в основном для экономии памяти. Деревья же применяются в виду особенностей выбранного алгоритма решения задачи.

3.4.1.2.2.1 Определение деревьев

Определим формально дерево как конечное множество T, состоящее из одного или более узлов, удовлетворяющих следующим свойствам.

1) Имеется один специальный узел, называемый корнем данного дерева.

2) Остальные узлы (исключая корень) содержатся в m>=0 попарно не пересекающихся множествах T1,...,Tm, каждое из которых в свою очередь является де-

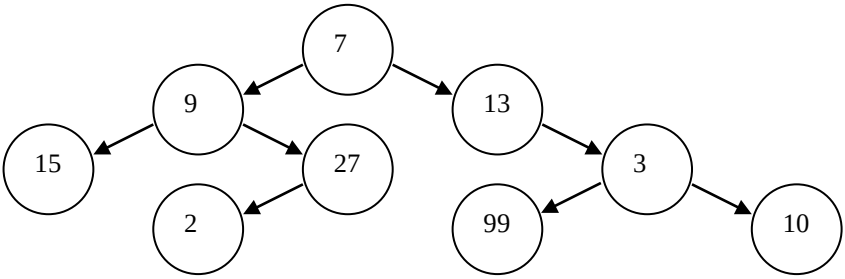
ревом. Деревья T1,...,Tm называются поддеревьями данного дерева.

Из нашего определения следует, что каждый узел дерева является корнем некоторого поддерева, которое содержится в этом дереве. Число поддеревьев данного узла называется степенью этого узла. Узел с нулевой степенью называется концевым узлом или листом. Дерево называется бинарным, если каждый узел имеет два поддерева.

В качестве примера рассмотрим бинарные деревья с базовым типом integer:

Бинарное дерево либо пусто, либо состоит из узла, содержащего целое число, и левого и правого поддеревьев, являющихся бинарными деревьями.

Вот так оно выглядит в представлении чеовека:

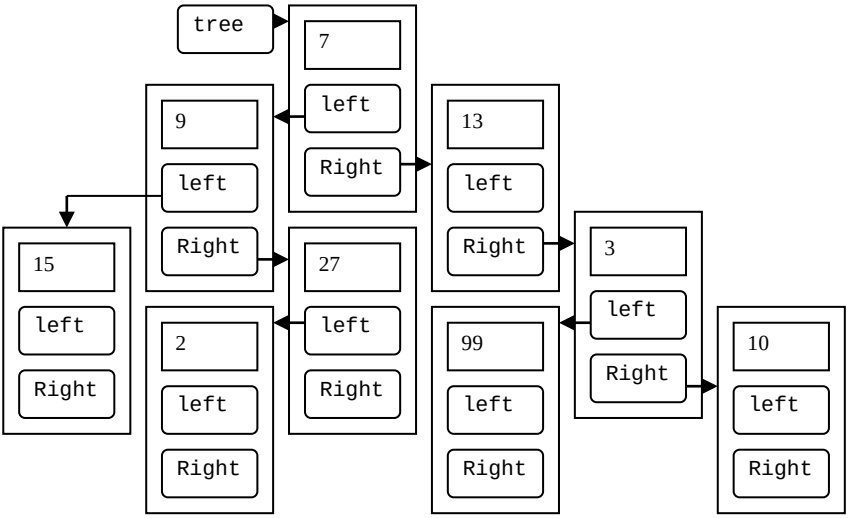


Соответствующее определение типа бинарное дерево, использующее ссылки, следующее:

```
type
  PComp= ^Comp;
  Comp= record
          D: Integer;
          Left, right: PComp
        end;
var
```

```
Tree: Pcomp;
```

А вот так можно отобразить его схематическое представление в памяти ЭВМ:



Здесь подразумевается, что указатели, никуда не ссылающиеся, указывают на nil

Узел с числом 7 называется корнем дерева. Дерево обычно изображают корнем вверх. Пустые деревья не обозначены. Узлы, имеющие только пустые поддеревья, являются листьями (в данном дереве это узлы -15, 2, 99 и 10).

Каждый узел поддерева имеет свой уровень, который определяется рекурсивно:

1) уровень корня бинарного дерева равен 0;

2) если уровень узла равен n, то уровни корней левого и правого поддеревьев равны n+1.

3.4.1.2.2.2 Формирование дерева

Рассмотрим формирование дерева на примере программы, которая сортирует какую-либо последовательность. Она делает это, основываясь на следующем алгоритме: записать число в левую ветвь, поддерева, если оно меньше корня поддерева и в правую – если больше.

```
Program sort_by_tree;

Type
  pTree=^TTree;
  TTree=record
          Data: SomeType;
          Left, right: pTree;
        End;

Var
  Tree: pTree;
  N, i: integer;

Procedure AddToTree(var tree: pTree;
var a: integer);
Begin
  If tree=nil then
  Begin
    New(tree);
    Tree^.data:=a;
  End
  Else
  If a>tree^.data then
         AddToTree(Tree^.right,   a)
else
         AddToTree(Tree^.left, a);
End;
```

```
Begin
  Readln(n);
  For I:=1 to n do
  Begin
    Readln(a);
    AddToTree(tree, a);
  End;
End.
```

3.4.1.2.2.3 Обход дерева

До конца раздела (если не оговорено противное) рассматриваются только бинарные деревья, поэтому прилагательное бинарное будем опускать.

Имеется много задач, которые можно выполнять на древовидной структуре: распространенная задача - выполнение заданной операции P с каждым элементом дерева. Здесь P рассматривается как параметр более общей задачи посещения всех узлов, или, как это обычно называют, обхода дерева.

Если рассматривать эту задачу как единый последовательный процесс, то отдельные узлы посещаются в некотором определенном порядке и могут считаться расположенными линейно. В самом деле, описание многих алгоритмов существенно упрощается, если можно говорить о переходе к следующему элементу дерева, имея в виду некоторое упорядочение.

Существуют три принципа упорядочения, которые естественно вытекают из структуры деревьев. Так же как и саму древовидную структуру, их удобно выразить с помощью рекурсии. Пусть R обозначает корень, а A и B левое и правое поддеревья, тогда можно определить такие три упорядочения:

1) сверху вниз: R, A, B (посетить корень до поддеревьев);

2) слева направо: A, R, B;

3) снизу вверх: A, B, R (посетить корень после поддеревьев).

Обходя дерево из 9 и выписывая числа, находящиеся в узлах, в том порядке, в котором они встречаются, мы получаем следующие последовательности:

1) сверху вниз: 7 9 15 27 2 13 3 100 10;
2) слева направо: 15 9 2 27 7 13 100 3 10;
3) снизу вверх: 15 2 27 9 100 10 3 13 7.

Теперь выразим эти три метода обхода как три конкретные программы с явным параметром t, означающим дерево, с которым они имеют дело, и неявным параметром P, означающим операцию, которую нужно выполнить с каждым узлом. Эти три метода легко сформулировать в виде рекурсивных процедур; они вновь служат примером того, что действия с рекурсивно определенными структурами данных лучше всего описываются рекурсивными алгоритмами.

```
Type
  pTree=^TTree;
  TTree=record
          Data: SomeType;
          Left, right: pTree;
        End;

procedure preorder(Tree: pTree);
begin
  if t <> nil then
  begin
    Obrabotka(Tree);
    preorder(Tree^.left);
    preorder(Tree^.right)
  end
end;
```

```
procedure postorder(Tree: pTree);
begin
  if t<>nil then
  begin
    postorder(Tree^.left);
    postorder(Tree^.right);
    Obrabotka(Tree)
  end
end;

procedure inorder(Tree: pTree);
begin
  if t<>nil then
  begin
    inorder(Tree^.left);
    Obrabotka(Tree);
    inorder(Tree^.right)
  end
end;
```

Теперь когда вы знаете как обходить дерево, попробуйте самостоятельно написать процедуру к программе, описанной в предыдущем разделе, которая выведет на экран упорядоченную последовательность.

Примером не бинарных деревьев могут послужить структура глав данного методического пособия, структура директорий операционной системы Windows.

4 Модульное программирование

Модуль (UNIT) в Pascal - это особым образом оформленная библиотека подпрограмм. Модуль в отличие от программы не может быть запущен на выполнение самостоятельно, он может только участвовать в построении программ и других модулей.

Модули позволяют создавать личные библиотеки процедур и функций и строить программы практически любого размера.

Модуль в Pascal представляет собой отдельно хранимую и независимо компилируемую программную единицу.

В общем случае модуль - это совокупность программных ресурсов, предназначенных для использования другими программами. Под программными ресурсами понимаются любые элементы языка Pascal: константы, типы, переменные, подпрограммы. Модуль сам по себе не является выполняемой программой, его элементы используются другими программными единицами.

Все программные элементы модуля можно разбить на две части:

- программные элементы, предназначенные для использования другими программами или модулями, такие элементы называют видимыми вне модуля;

- программные элементы, необходимые только для работы самого модуля, их называют невидимыми или скрытыми.

В соответствии с этим модуль, кроме заголовка, содержит две основные части, называемые интерфейсом и реализацией.

В общем случае модуль имеет следующую структуру:

```
      unit <имя модуля>;    {заголовок модуля}

      interface
{описание видимых программных элементов модуля}
```

```
    { описание скрытых программных эле-
ментов модуля }

    begin
    {операторы инициализации элементов
модуля}
    end.
```

В частном случае модуль может не содержать части реализации и части инициализации, тогда структура модуля будет такой:

```
    unit <имя модуля>;    {заголовок мо-
дуля}

    interface
    {описание видимых программных эле-
ментов модуля}

    implementation

    end.
```

Использование в модулях процедур и функций имеет свои особенности. Заголовок подпрограммы содержит все сведения, необходимые для ее вызова: имя, перечень и тип параметров, тип результата для функций, эта информация должна быть доступна для других программ и модулей. С другой стороны, текст подпрограммы, реализующий ее алгоритм, другими программами и модулями не может быть использован. Поэтому заголовок процедур и функций помещают в интерфейсную часть модуля, а текст – в часть реализации.

Интерфейсная часть модуля содержит только видимые (доступные для других программ и модулей) заголовки процедур и функций (без служебного слова

forward). Полный текст процедуры или функции помещают в часть реализации, причем заголовок может не содержать список формальных параметров.

Исходный текст модуля должен быть откомпилирован с помощью директивы Make подменю Compile и записан на диск. Результатом компиляции модуля является файл с расширением .TPU (Pascal Unit). Основное имя модуля берется из заголовка модуля.

В том случае, если имена переменных в интерфейсной части модуля и в программе, использующей этот модуль, совпадают, обращение будет происходить к переменной, описанной в программе. Для обращения к переменной, описанной в модуле, необходимо применить составное имя, состоящее из имени модуля и имени переменной, разделенных точкой.

Например, пусть имеется модуль, в котором описана переменная K:

```
unit M;

interface

var
  K: Integer;
implementation
  ................
end.
```

Пусть программа, использующая этот модуль, также содержит переменную K:

```
Program P;

uses M;

var
```

```
  K: Char;

begin
  .............
end.
```

Для того, чтобы в программе P иметь доступ к переменной K из модуля M, необходимо задать составное имя M.K.

Использование составных имен применяется не только к именам переменных, а ко всем именам, описанным в интерфейсной части модуля.

Рекурсивное использование модулей запрещено.

Если в модуле имеется раздел инициализации, то операторы из этого раздела будут выполнены перед началом выполнения программы, в которой используется этот модуль.

5 Модуль Crt

Модуль CRT.

Наиболее часто применимые процедуры и функции модуля CRT:

```
Procedure ClrScr;
Procedure Delay (MS: Word);
Procedure GoToXY (X, Y: integer);
Function KeyPressed: Boolean;
Function ReadKey: Char;
Procedure Sound (Hz: Word);
Procedure NoSound;
Procedure   TextBackground   (Color:
Byte);
Procedure TextColor (Color: Byte);
Function WhereX: Integer;
Function WhereY: Integer;
```

Процедура ClrScr очищает экран в текстовом режиме, что в значительной мере облегчает представление постоянно меняющейся информации пользователю.

Процедура Delay осуществляет при её вызове задержку выполнения дальнейшего кода на MS миллисекунд. Однако из-за особенностей модуля, «миллисекунды» - это лишь выражение для определения времени, на самом деле на компьютерах различной конфигурации задержка функцией Delay различна. Существует множество способов получения «неизменной» задержки. Например, если Вас устраивает точность задержки в 1/18 секунды (один процессорный «тик»), можно использовать следующую процедуру:

```
Procedure Timer(X: LongInt);
{ozhidanie na X tikov}
var
  l: LongInt;
begin
  l := MemL[Seg0040:$6c];
  While MemL[Seg0040:$6c] < l+x do;
end;
```

Процедура GoToXY Перемещает курсор на позицию X строки Y.

Функция KeyPressed возвращает True, если была нажата какая-либо клавиша. Это очень удобно использовать, если программа непрерывно выполняет какое-то действие и это действие в реальном времени зависит от нажатий клавиш.

Функция ReadKey возвращает значение нажатой клавиши. Чаще всего используется в совокупности с функцией KeyPressed.

Процедура Sound воспроизводит звук частотой Hz в герцах. Стоит отметить, что для того, чтобы прекра-

тить воспроизведение звука нужно вызвать процедуру NoSound.

Процедура TextBackground устанавливает цвет фона для символов текста в цвет Color (номер цвета). При этом следует помнить, что цвет фона для символа – это цвет небольшого прямоугольника, окружающего символ, а не цвет всего символьного поля.

Процедура TextColor устанавливает цвет текста равным Color.

Функции WhereX и WhereY возвращают соответственно значения текущей позиции и строки курсора на экране.

6 Модуль Graph

Это один из наиболее распространённых модулей для работы с графикой в Pascal.

6.1 Начало работы

Для работы с графикой помимо модуля Graph необходимы графические драйвера. В теле модуля описана работа только с четырьмя типами графических адаптеров Hercules, CGA, EGA, VGA. Драйвера для графических адаптеров хранятся в файлах с расширением BGI. Таким образом, Вам для работы потребуются файлы:

CGA.BGI, EGAVGA.BGI, HERC.BGI

Однако недостаточно только лишь подключить графический драйвер, так как графические адаптеры зачастую поддерживают несколько режимов. Для инициализации графики требуется указать используемый режим.

Драйвер	Режим	Разрешение	Файл
CGA (1)	CGAC0, CGAHI	320×200 (640×200)	CGA.BGI
EGA (3)	EGALo, EGAHI	640×200 (640×350)	EGAVGA.BGI
VGA (9)	VGALo, VGAHI	640×200 (640×350)	EGAVGA.BGI
HERC	HERCMONOHI	720×348	HERC.BGI

Для того, чтобы подключить драйвер и выбрать режим необходимо применить процедуру

```
InitGraph(var VideoDriver, DriverMode: integer; Path: string);
```

Где `VideoDriver` – переменная, либо константа, содержащая в себе код подключаемого графического драйвера (в таблице он указан в скобках). А `DriverMode` – соответственно переменная или константа, содержащая в себе код режима, в котором будет запущена графика. `Path` – путь к папке, в которой лежит графический драйвер (опять же не стоит забывать об особенностях, связанных с относительными и абсолютными путями в среде Turbo Pascal).

Предусмотрена возможность программного определения наиболее подходящих графического драйвера (при помощи встроенной функции `Detect`, возвращающей код подходящего драйвера) и режима при помощи процедуры `DetectGraph(var VideoDriver, DriverMode: integer);`

По окончании работы с графикой необходимо отключить графический режим. Это осуществляется при помощи встроенной процедуры `CloseGraph.`

```
Uses
```

```
  Graph;
var
  GraphDriver, GraphMode: integer;
begin
  GraphDriver:=Detect;
  GraphMode:=1;
  InitGraph(GraphDriver, GraphMode, '');
  CloseGraph;
End.
```

Память видеобуфера подразделяется на несколько частей - так называемых видеостраниц. Их количество зависит от текущего режима и типа адаптера. Более одной страницы имеют адаптеры EGA, VGA и Hercules. Нумерация страниц начинается с 0.

В каждый отдельный момент на экране может быть отображена только одна страница, она называется **видимой**. По умолчанию видима страница с номером 0. Страничная организация позволяет с помощью графических процедур и функций формировать изображение на любой из страниц. Страница, на которой в данный момент формируется изображение, называется **активной.**

Драйвер	**Режим**	**Цвет**	**Страницы**
EGA (3)	EGALO (0)	16	4
EGA (3)	EGAHI (1)	16	2
VGA (9)	VGALO (0)	16	4
VGA (9)	VGAMED (1)	16	4
HERC (7)	HERCMONOHI (0)	2	2

Процедура `SetActivePage(Page: word)` устанавливает активную страницу для построения изображения. Например:

```
SetActivePage(1);
```

Построение изображения может производиться незаметно для смотрящего на экран (в этом случае активная страница не совпадает с видимой). например, страница может формироваться "подкачкой" данных с диска или с помощью любых процедур Pascal. Сформировав страницу, ее можно показать на экране с помощью процедуры

```
SetVisualPage(Page: word);
```

где `Page` - номер видимой страницы. Например,

```
SetActivePage(0);  {показ страницы 0 на
экране}
OutText('Страница 0'); {строка появляется
на экране}
SetActivePage(1); {активная страница}
OutText('Страница    1');    {формирование
изображения на странице 1, но на экране
ее нет!}
Readln;
SetVisualPage(1);   {показ   страницы   1,
строка на экране}
```

Для контроля за правильностью работы графики определены две функции: GraphResult и GraphErrorMsg. GraphResult возвращает значение 0, если последняя графическая операция выполнилось без ошибок или число в диапазоне -15...-1, если ошибка была. Некоторые ошибки и их коды приведены в таблице

Константа	**Значение**	**Описание**
grOk	0	Нет ошибок
grNoInitGraph	-1	Графика не инициализирова-

		на (используйте InitGraph)
grNotDetected	-2	Графическое устройство не обнаружено
grFileNotFound	-3	Файл драйвера устройства не найден

В качестве примера рассмотрим следующий фрагмент:

```
Uses
  Graph;
Var
  ErrorNumber: integer;
Begin
  ErrorNumber := GraphResult;
```

В переменной `ErrorNumber` содержится код ошибки. Можно пользоваться как кодом ошибки, так и соответствующей ему константой, например:

```
If ErrorNumber <> grOk then
  writeln('Обнаружена ошибка');
```

`GraphErrorMsg` - возвращает строку сообщения об ошибке, соответствующую коду ошибки. Например, процедура

```
writeln(GraphErrorMsg(ErrorNumber));
```

выведет строку `"No error"`, так как в примере графический режим установлен правильно.

6.3 Система координат

Система координат дисплея устроена следующим образом:

Начало координат, находится в левом верхнем углу, x увеличивается слева направо, а y – сверху вниз (на это следует особо обращать внимание при использовании модуля для отображения графиков, поскольку в противном случае графики будут перевёрнуты).

Чтобы построить изображение, необходимо указывать , по крайней мере, точку начала ввода. В графическом режиме видимого курсора нет, но есть невидимый **текущий указатель** CP (Current Pointer). Фактически это тот же курсор, но он невидим.

В графическом режиме для перемещения CP имеется ряд процедур и функций. В первую очередь это `MoveTo` и `MoveRel`. Процедура `MoveTo(x,y)` перемещает текущий указатель в точку с координатами x и y. Процедура `MoveRel(dx,dy)` перемещает CP на dx точек по горизонтали и на dy точек по вертикали.

В ряде программ выполняется постоянный контроль местоположения текущего указателя. Для этого используются функции `GetX` и `GetY`, которые возвращают соответственно значение координаты x и координаты y указателя CP. Например:

```
var
  xpos,ypos:integer;
…
xpos := GetX;
ypos := GetY;
…
```

В процессе управления CP может возникнуть ситуация, когда его координаты выйдут за допустимые пределы. в таких ситуациях используются функции `GetMaxX:integer` и `GetMaxY:integer`, которые

возвращают соответственно максимально возможные для установленного режима значения координат x и y.

6.4 Графические примитивы

Чтобы стереть изображения на экране, т. е. очистить его, используется не имеющая параметров процедура `ClearDevice`. С момента ее выполнения все установки по цвету, фону и т. д. аннулируются и указатель CP переходит в точку с координатами (0,0).

Процедура «Точка».

```
PutPixel(x,y: integer; Color:word);
```

где x и y - экранные координаты расположения точки, `Color` - ее цвет. Построение прямоугольников

Для построения прямоугольных фигур имеется несколько процедур. Первая из них - процедура вычерчивания одномерного прямоугольника:

```
Rectangle(X1,Y1,X2,Y2: integer);
```

где X1, Y1 - координаты левого верхнего угла, X2, Y2 - координаты правого нижнего угла прямоугольника. Это очень полезная процедура, с ее помощью, в частности, можно легко построить любую диаграмму для визуального анализа данных. Область внутри прямоугольника не закрашена и совпадает по цвету с фоном. В качестве примера приведем фрагмент, который выводит на экран 100 вычерченных разным цветом динамически меняющихся по высоте прямоугольников:

```
for i:=1 to 100 do
begin
  SetColor(Green);          {установка
цвета}
```

```
        Rectan-
gle(200,Random(300),250,300);  {i-ый пря-
моугольник}
        Delay(50);{задержка}
        ClearDevice {очистка экрана}
      end;
```

Более эффектные для восприятия прямоугольники можно строить с помощью процедуры

```
Bar(x1,y1,x2,y2: integer);
```

которая рисует закрашенный столбец.

Еще одна весьма эффектная процедура:

```
Bar3D(x1,y1,x2,y2:integer;    Depth:
word; Top: boolean)
```

Вычерчивает закрашенный прямоугольник в так называемом «2,5» измерении. При этом используется тип и цвет закраски, установленные с помощью процедуры `SetFilllStyle`. Параметр `Depth` представляет собой число пикселов, задающих глубину трехмерного контура. Чаще всего его значение равно четверти ширины прямоугольника:

```
Depth := (x2-x1) div 4;
```

параметр Top определяет, строить над прямоугольником вершину (Top=true) или нет (Top=false).

Построение многоугольников

Процедура `DrawPoly` позволяет строить любые многоугольники линией текущего цвета, стиля и толщины. Она имеет следующий формат:

```
DrawPoly(numPoints: word; var PolyPoints);
```

Параметр `polyPoints` является нетипизированным параметром, который содержит координаты каждого пересечения в многоугольнике (последовательностью блоков по 4 байта: 2 байта X и 2 байта Y типом integer). Наиболее удобной конструкцией для `polyPoints` можно считать массив записей, поля которой `X, Y: integer,` причём такая запись определена типом

```
pointtype = record
              X, Y: integer;
            End;
```

Таким образом, предлагается использовать следующую конструкцию:

```
Type
  PolyPoints=array[1.. numPoints] of pointtype;
```

Параметр NumPoints задает число координат в `PolyPoints`. Необходимо помнить, что для вычерчивания замкнутой фигуры с n вершинами нужно передать при обращении к процедуре `DrawPoly` n+1 координату, где координата вершины с номером n будет равна координате вершины с номером 1.

Процедура аналогичная процедуре `DrawPoly` -

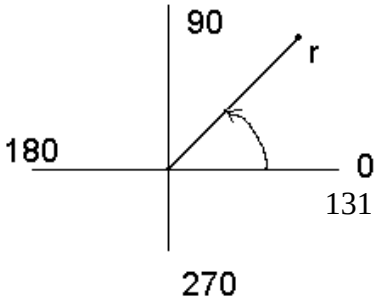
```
FillPoly(NumPoints: word; var PolyPoints);
```

Значение параметров те же, что и в процедуре DrawPoly. Действие тоже аналогично, но фон внутри многоугольника закрашивается. Например вот пример программы, которая строит правильный многоугольник.

```
program Simple_fillpoly;

uses
  graph;

var
  x: real;
  i,
  vd, dm: integer;
  ugol: array[1..10] of pointtype;

begin
  initgraph(vd, dm, '');
  for i:=1 to 9 do
  begin
    x:=2*I*pi/9;

ugol[i].x:=round(100*cos(x))+320;

ugol[i].y:=round(100*sin(x))+240;
  end;
  ugol[10]:=ugol[1];
  drawpoly(10, ugol);
  readln;
  closegraph;
end.
```

Построение дуг и окружностей

Для задания углов используется полярная система координат:



Процедура вычерчивания окружности текущим цветом имеет следующий формат:

```
Circle(x,y, Radius: word),
```

где x и y - центр окружности, а Radius - ее радиус. Например, следующий фрагмент обеспечивает вывод ярко-зеленой окружности с радиусом 50 пикселей и центром в точке 450, 100:

```
SetColor(LightGreen);
Circle(450,100,50);
```

В ряде случаев, в частности для создания псевдо-объемных фигур, используются дуги. Их можно вычертить с помощью процедуры

```
Arc(x,y: integer; StAngle, EnAngle, Radius: word);
```

где x, y - центр окружности, `StAngle` и `EnAngle` - начальный и конечный угол, `Radius` - радиус. Цвет для вычерчивания устанавливается процедурой SetColor. Очевидно, что если `StAngle`=0 и `EndAngle` = 360, то вычерчивается полная окружность.

Для построения эллиптических дуг предназначена процедура

```
Ellipse(X,Y: integer; StAngle, EndAngle: word; xR, yR: word);
```

где x, y - центр эллипса в дисплейных координатах, xR и yR - горизонтальная и вертикальная оси. Дуга эллипса вычерчивается от начального угла `StAngle` до конечного угла `EndAngle` текущим цветом. Значения `StAngle` = 0 и `EndAngle` = 360 приведут к вычерчиванию полного

эллипса. пример построения эллипса, выведенного ярко-голубым цветом:

```
SetColor(LightCyan);
Ellipse(100,100,0,360,30,50);
```

Обратите внимание, что фон внутри эллипса совпадает с фоном экрана. Чтобы создать закрашенный эллипс (в частности, закрашенный круг), используется специальная процедура

```
FillEllipse(x,y: integer; xR,yR: word);
```

где x, y центр эллипса в дисплейных координатах, xR и yR - горизонтальная и вертикальная оси. Заменитель устанавливается процедурой `SetFillStyle`.

```
SetFillStyle(wideDotFill, Green);
{установка стиля заполнения}
SetColor(LightRed); {цвет для вычерчивания эллипса}
FillEllipse(300,150,50,50);
```

В этом фрагменте эллипс вычерчивается ярко-красной кривой и заполняется редкими точками зеленого цвета.

6.5 Стили

Возможные значения `Color` приведены в таблице. Например, оператор

```
for i:=0 to 59 do
  PutPixel(i,0,Red);
```

выведет в первую строку экрана 60 красных точек.

Цвет	Код	Цвет	Код
Black	0	DarkGray	8
Blue	1	LightBlue	9
Green	2	LightGreen	10
Cyan	3	LightCyan	11
Red	4	LightRed	12
Magenta	5	LightMagenta	13
Brown	6	Yellow	14
LightGray	7	White	15

Чтобы узнать цвет точки в конкретной позиции экрана, используется функция

```
GetPixel(x,y:integer):word;
```

Из точек строятся линии (отрезки прямых). Это делает процедура `Line(x1,y1,x2,y2)`, где x1 и y1 - координаты начала, x2 и y2 - координаты конца линии. В процедуре Line нет параметра для установки цвета. В этом и других аналогичных случаях цвет задается процедурой `SetColor(Color)`, где `Color` - цвет, значение которого берется из табл. 7. Например,

```
SetColor(Cyan);
Line(1,1,600,1);
```

Для черчения линий применяются еще две процедуры: `LineTo` и `LineRel`. Процедура `LineTo(x,y)` строит линию из точки текущего указателя в точку с координатами x,y. Процедура LineRel(dx,dy) проводит линию от точки текущего расположения указателя в точку (CPx+dx, Cpy+dy), где CPx и Cpy - текущие координаты CP.

Можно вычерчивать линии самого различного стиля: тонкие, широкие, шриховые, пунктирные и т. д. Установка стиля производится процедурой

```
SetLineStyle (LineStyle : word; Pat-
tern: word; Thickness: word);
```

Параметр `LineStyle` устанавливает стиль линии, возможные значения которого приведены в таблце; `Pattern` - образец, `Thickness` - толщина линии, определяемая константами, указанными в таблице. Если применяется один из стандартных стилей, значение `Pattern` равно 0. Например:

```
SetLineStyle(DottedLn,0,NormWidth);
Line(1,1,600,1);
```

Если пользователь хочет активизировать свой собственный стиль, то значение `LineStyle` равно 4. В этом случае `Pattern` - двухбайтовое число.

Тип линий

Константа	**Значение**	**Описание**
`SolidLn`	0	Непрерывная линия
`DottedLn`	1	Линия из точек
`CenterLn`	2	Линия из точек и тире
`DashedLn`	3	Штриховая линия
`UserBitLn`	4	Тип пользователя

Толщина линий

Константа	**Значение**	**Описание**
`NormWidth`	1	Нормальная толщина (1 пиксель)
`ThickWidth`	3	Жирная линия (3 пикселя)

Цвет закраски устанавливается с помощью `SetFillStyle`.

Функция `SetFillStyle(pattern:word;color:word)` определяет стиль заполнения. Пример использования:

```
SetFillStyle(1,3);
```

Значение `pattern` приведены в таблице и могут быть представлены константой или цифрой, color берется из шкалы цветов

Константа	**Значение**	**Стиль**
EmptyFill	0	Заполнение цветом фона
SolidFill	1	Однородное заполнение цветом
LineFill	2	Заполнение символами "--", цвет - color
LtSlashFill	3	Заполнение символами "//" нормальной толщины, цвет - color
SlashFill	4	Заполнение символами "//" удвоенной толщины, цвет - color
BkSlashFill	5	Заполнение символами "\\" удвоенной толщины, цвет - color
LtBkSlashFill	6	Заполнение символами "\\" нормальной толщины, цвет - color
HatchFill	7	Заполнение вертикально-горизонтальной штриховкой тонкими линиями
XhatchFill	8	Заполнение штриховкой крест-накрест по диагонали "редкими" тонкими линиями, цвет - color
InterLeaveFill	9	Заполнение штриховкой крест-накрест по диагонали "частыми" тонкими линиями, цвет - color

WideDotFill	10	Заполнение "редкими" точками
CloseDotFill	11	Заполнение "частыми" точками
UserFill	12	Заполнение по определенной пользователем маске заполнения, цвет - color

6.6 Работа с текстом

Для вывода текста на экран используются процедуры `OutText` и `OutTextXY`. Процедура

```
OutText(TextString : string)
```

выводит строку текста, начиная с текущего положения CP.

Явный недостаток этой процедуры - нельзя указать произвольную точку начала вывода. Его можно устранить с помощью MoveTo, но лучше воспользоваться процедурой

```
OutTextXY( x,y,Text),
```

где x,y - координаты точки начала вывода текста, Text - константа или переменная типа string.

Для начинающих проблемой является вывод численных данных, ибо в Graph нет предназначенных для этого процедур. Выход прост: сначала преобразовать число в строку с помощью процедуры `Str`, а затем посредством '+' подключить ее к выводимой `OutTextXY` строке. Например:

```
Max := 34.56;
Str(Max:6:2, Smax); {результат преобразования находится в Smax}
OutTextXY(400,40,'Максимум
='+Smax);{+ - конкатенация}
```

Установить нужный шрифт можно процедурой

```
SetTextStyle(Font:word; Direc-
tion:word; CharSize:word)
```

где `Font` - выбранный шрифт, `Direction` - направление (горизонтальное или вертикальное), `CharSize` - размер выводимых символов. При организации вертикального вывода необходимо учитывать, что если программист не установит точку начала вывода с помощью `MoveTo`, то текст начинается с нижней строки экрана и продолжается вверх. величину выводимых символов можно устанавливать с помощью коэффициента `CharSize`. Если `CharSize = 1`, то символ строится в матрице 8×8, если `CharSize = 2`, то используется матрица 16×16 и т. д. до 10-кратного увеличения.

Шрифты

Константа	**Значение**	**Описание**
`DefaultFont`	0	8×8 - битовый шрифт
`TriplexFont`	1	Штриховые шрифты
`SmallFont`	2	Малый шрифт
`SansSerifFont`	3	Сансериф
`GothicFont`	4	Готический

Ориентация шрифтов

Константа	**Значение**	**Описание**
HoizDir	0	Слева направо
VertDir	1	Снизу вверх

7 Математический пакет MathCAD

7.1 Общий вид главного окна

С точки зрения интерфейса система MathCAD 13 представляет собой полноценное приложение, работающее в операционной системе Windows. То есть, открыв сам MathCAD, можно увидеть меню пользователя, разнообразные панели инструментов с кнопками быстрого доступа, строку состояния, вертикальную и горизонтальную полосы прокрутки.

Все действия, вычисления, настройка параметров в MathCAD доступны через главное меню, состоящее из вложенных разнообразных подменю. Большинство действий возможны также и с помощью клавиатуры через комбинации соответствующих клавиш. Панели инструментов прежде всего предназначены для быстрого доступа ко многим командам MathCAD 13 с помощью мыши. Строка состояния сообщает пользователю развёрнутую информацию о текущем действии. Полосы прокрутки позволяют перемещаться по самому документу.

Для того, чтобы передвигаться по документу с помощью клавиатуры, в нём находится специальный курсор. Чтобы его не путать с курсором мыши, дальше он будет называться **курсором.** Данный курсор показывает ваше местоположение в текущем документе, и в зависимости от того, где находится, он может принимать разные формы (перекрестие, перпендикулярные линии, вертикальная черточка) и цвета (красный, синий). Непосредственно передвигать курсор можно при помощи клавиатурных кнопок ВЛЕВО, ВПРАВО, ВВЕРХ, ВВОД, ТАБУЛЯЦИЯ, ПРОБЕЛ, BACKSPACE.

Следует сразу обратить внимание на тот факт, что при нажатии клавиш PAGE UP, PAGE DOWN вы можете

передвигаться по документу постранично, но курсор **останется на прежнем месте**. Курсор переносится автоматически на то место, куда вы нажмёте левой или правой кнопкой мыши.

7.1.1 Главное меню

В MathCAD 13 главное меню состоит из 9 основных меню: File, Edit, View, Insert, Format, Math, Symbolics, Window, Help.

С помощью меню File, можно производить разнообразные операции с документом MathCAD: создать новый документ, открыть уже существующий, закрыть текущий, сохранить текущий, установить параметры печати и напечатать весь документ. Как функциональное дополнение и в это меню включен список ранее открытых документов.

Меню Edit позволяет проводить различные действия с редактором MathCAD: отменить ошибочно введённый текст, повторить последнее действие, вырезать (скопировать или вставить) текст, отыскать (заменить) текст в рабочем документе, проверить орфографию.

В меню View настраивается непосредственный вид главного окна MathCAD: включение/выключение панелей инструментов, изменение масштаба просмотра документа. Также в этом меню предусмотрены опции анимации и воспроизведения звука.

Меню Insert помогает вставить в рабочий документ разнообразные объекты: графики, матрицы, функции, рисунки, комментарии и любые зарегистрированные в операционной системе OLE (Active X) объекты.

В меню Format пользователю представляется возможность менять формат представления разнообразных данных своего документа: уравнения, результата, текста, графиков, цветов и т.д.

Меню Math позволяет пользователю управлять вычислительным процессом MathCAD.

С помощью меню Simbolics возможно управление символьными вычислениями, которые реализуются встроенным символьным транслятором.

Переключение между разными документами в MathCAD возможно с помощью меню Window.

Меню Help содержит самую разнообразную информацию о самой системе MathCAD 13.

7.1.2 Панели инструментов

Стандартная панель инструментов (рисунок 7.1 а) объединяет разнообразные, но наиболее часто используемые команды в виде кнопок быстрого доступа. Если поднести курсор мыши к какой-нибудь кнопке, то смысл ее действия будет пояснен во всплывающей подсказке, а более развернутая информация появится в строке состояния.

Панель форматирования (рисунок 7.1 б) предназначена для выбора шрифта, стиля документа, выравнивания текста.

а

б

Рисунок 7.1 Стандартная панель инструментов (а), панель форматирования (б)

Математическая панель инструментов предоставляет доступ ко всем математическим ресурсам MathCAD в виде дополнительных панелей инструментов. В MathCAD 13 возможно открыть девять дополнительных панелей инструментов:

Арифметическая панель (кнопка ) предоставляет доступ к арифметическим операторам;

Панель графиков (кнопка ) представляет доступ к графическим ресурсам системы MathCAD;

Панель матриц (кнопка ) предоставляет доступ к операторам работы с векторами и матрицами;

Панель вычислений (кнопка ) предоставляет доступ к операторам записи вычислений;

Панель математического анализа (кнопка ) предоставляет доступ к операторам математического анализа;

Панель элементов программирования (кнопка ) предоставляет доступ к операторам программирования;

Панель греческого алфавита (кнопка ) открывает доступ к греческим символам;

Панель символьных вычислений (кнопка ) дает возможность применять операторы символьного вычисления.

7.2 Работа в редакторе документов системы MathCAD 13.

7.2.1 Понятие региона

Любое математическое выражение, текстовый комментарий, рисунок и т.д. в документе MathCAD является независимым объектом и называется **регионом**. Вы можете войти в регион, если нажмете левой кнопкой мыши на него. После этого данный регион выделится черной рамочкой. Перейти в регион также можно, если переместить туда курсор.

В MathCAD 13 все регионы являются независимыми (в смысле редактирования) объектами. Это позво-

ляет располагать **любой регион в любом месте** документа, **копировать, удалять, перемещать** регион в буфер обмена Windows, что делает редактирование самого документа более наглядным и быстрым.

Если вы находитесь в самом регионе, то видите, что его границы обрамляются тонкими черными линиями, а вне региона видно лишь его содержимое без границ. Для того, чтобы были видны границы всех регионов, необходимо включить соответствующую опцию Regions в меню View.

7.2.2 Редактирование математических выражений

Одним из основных достоинств системы MathCAD является естественная математическая запись любого выражения. Например, если необходимо разделить число «a» на число «b», то в самой системе MathCAD это будет выглядеть именно как $\frac{a}{b}$, а не a/b или $a\!\!\diagup\!\!b$ (также это касается и остальных математических операторов). Это, несомненно, делает MathCAD не просто «большим калькулятором» (как, например, Excel), а мощной математической системой для решения серьезных вычислительных задач.

Для создания нового региона с математическим выражением необходимо переместить курсор на пустое (т.е. не занятое другим регионом) место рабочего документа. Курсор должен быть в виде красного перекрестия. Для примера посчитаем с помощью MathCAD следующее выражение: 1+2. Нажмите на клавиатуре цифру «1». Вы увидите на экране саму цифру «1», обведенную в черный прямоугольник. Таким образом, вы автоматически оказались внутри региона математического выражения. В нем курсор приобретает вид синих перпендикулярных линий:

вертикальная показывает текущее положение относительно всей формулы, а горизонтальная – зону охвата формулы.

После того как вы нажмете на клавишу «+», вы увидите, что справа от знака сложения появился маленький черный прямоугольник, обозначающий пустое (но необходимое для заполнения) место. Сам MathCAD подсказывает вам, что оператор сложения имеет два числа. Заполните цифрой «2» данный черный прямоугольник. Теперь для вывода результата вычисления необходимо ввести оператор равенства «=». После его ввода автоматически появится ответ. Аналогичным образом можно ввести любое другое математическое выражение и рассчитать его.

Очень удобно для ввода, редактирования математических выражений применять соответствующие математические палитры. Например, рассчитаем следующее выражение: $\frac{\log(524.236)}{\sin(7)}+\sqrt{58}\approx 11{,}8$. Сделаем это двумя путями: используя клавиатуру и используя арифметическую панель инструментов (предварительно перейдя на пустое место рабочего документа).

Путь первый (используем клавиатуру):

1) последовательно наберите слово «log», открывающую круглую скобку, цифры «524», десятичную точку, цифры «236», закрывающую круглую скобку;

2) наберите косую черту «/». Вы увидите, что ранее набранное выражение «log(524.236)» немного приподнимается вверх, под ним появится горизонтальная черта, а под чертой – черный маленький прямоугольник (курсор автоматически передвинется на него). Таким образом, вы реализовали оператор деления;

3) продолжим набор и запишем в знаменатель синус семи: «sin», «(», «7», «)». Левая часть нашего выражения записана;

4) далее для ввода нашего выражения необходимо набрать его правую часть (корень из 58). Нажмите пробел (необходимо для того, чтобы из знаменателя перейти на уровень охвата всей дроби), «+», «\» (ввод оператора извлечения квадратного корня), «58». Ввод математического выражения закончен. Для получения ответа необходимо нажать «=».

Путь второй (используя арифметическую панель инструментов):

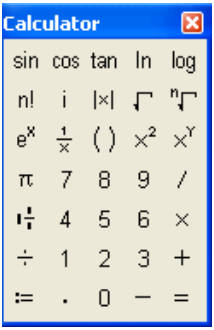


Рисунок 7.2 Арифметическая панель инструментов

1) откройте саму арифметическую панель инструментов (рис. 4), просмотрите ее внимательно. Вы увидите перечень всевозможных арифметических операторов и цифр;

2) попробуйте набрать наше выражение без помощи клавиатуры, используя только мышь и арифметическую панель инструментов.

Для удаления ошибочно набранных букв, цифр или операторов используется клавиша BACKSPACE или клавиша DELETE. Удаление группы символов также возможно, если выделить при помощи мыши или клавиатуры (выделенная область окрашивается в контрастный цвет) то, что необходимо, и нажать клавишу удаления DELETE или BACKSPACE.

7.2.3 Ввод текста

Под текстом в MathCAD понимается любое текстовое выражение, которое самой системой никак не обрабатывается и не вычисляется. Это сделано разработчиками MathCAD для того, чтобы была возможность ввода комментариев в любое свободное место документа.

Перед началом набора текста следует набрать на клавиатуре парные кавычки (или выбрать из меню Insert пункт Text region). Автоматически создается регион для набора текста в виде простой стандартной коробочки редактирования. Курсор принимает вид красной вертикальной черточки. Теперь можно набирать любой текст, вставлять его из любого текстового редактора, удалять, искать конкретные фразы и т.д. Для выхода из текстового региона достаточно нажать левую кнопку мыши в любом месте вне этого региона.

7.2.4 Построение двумерных графиков

С помощью двумерных графиков можно посмотреть вид функциональной зависимости или математических выражений. Для создания региона простого двухмерного графика достаточно нажать клавишу @ (или в графической панели нажать кнопку  , или выбрать в меню Insert в подменю Graph пункт X-Y plot), и вы увидите шаблон (рисунок 7.3).

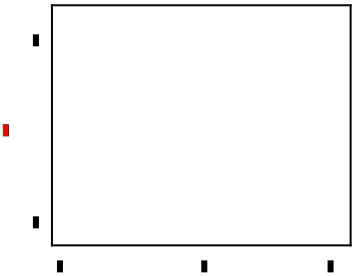


Рисунок 7.3 Шаблон региона двумерного графика

Для примера построим график функции x^2 в диапазоне [-10;10] (рисунок 7.4).

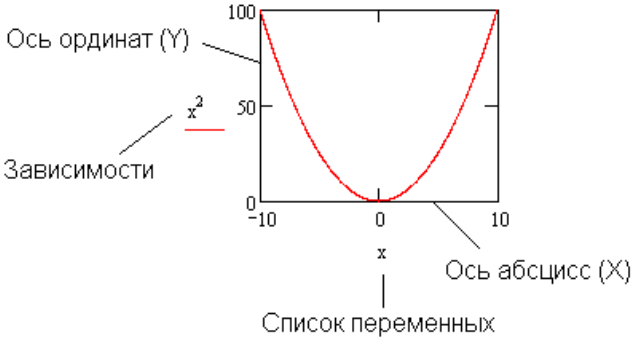


Рисунок 7.4 График функции x^2

В списке переменных графика введите имя переменной (в примере на рис. 6 – это x). На оси абсцисс слева в черном квадратике наберите наименьшее значение нашего диапазона (число минус десять), а справа – наибольшее (число плюс десять). На месте зависимостей наберите функцию x^2 по правилам редактирования математических выражений. Теперь для расчета и просмотра самого графика достаточно нажать клавишу F9 или про-

сто покинуть графический регион. Заметьте, масштабирование значений функции (т.е. минимальное и максимальное) MathCAD подберет автоматически.

7.3 Использование системы MathCAD для вычислений

7.3.1 Особенности языка MathCAD

Будучи математической системой, MathCAD заметно облегчает выполнение самых разнообразных математических расчетов. При решении многих задач MathCAD не требует от пользователя навыков программирования в общепринятом его понимании, т.е. подробного описания решения задачи на некотором специальном языке программирования. Вполне достаточно описать алгоритм решения нужных задач так же, как в математической литературе.

Это достигнуто за счет работы пользователя с промежуточным математически ориентированным языком описания задач, который называется входным. Этот язык интерпретирующего типа и поддерживает простой и удобный диалог с пользователем. Можно считать это важным аспектом общей визуализации вычислений, когда в наглядном и понятном виде не только выводятся результаты вычислений, но и задаются данные для них и описываются этапы решения задач.

К важнейшим типам данных в системе MathCAD относятся константы, обычные и системные переменные, массивы (векторы и матрицы) и данные файлового типа.

7.3.2 Алфавит MathCAD

Как и всякий язык (в том числе программирования), входной язык MathCAD имеет свой алфавит, т.е. набор символов, из которых состоят его объекты. В алфавит MathCAD входят: малые и большие латинские буквы, малые и большие греческие буквы, арабские цифры от 0

до 9, идентификаторы системных переменных, специальные знаки – операторы, имена встроенных функций, спецзнаки, малые и большие буквы кириллицы (для работы с русифицированными документами).

Константы – поименованные объекты, значения которых заведомо предопределены в системе. MathCAD имеет следующие типы констант:

- целочисленные константы (0, 1, 45, -80 и т.д.);

- вещественные числа с мантиссой и порядком ($12{,}3\cdot10^{-5}$ – десятичная константа с мантиссой 12,3 и порядком -5);

- восьмеричные числа с основанием 8 и значениями одного разряда от 0 до 7 (помечаются латинской буквой O от слова octal - восьмеричное);

- шестнадцатеричные числа с основанием 16, каждый разряд которых может иметь значения 0, 1, …, 9, A, B, C, D, E, F (помечаются знаком H или h в конце – от слова hexagonal);

- комплексные числа $Z=\mathrm{Re}Z + i\cdot\mathrm{Im}Z$, где $\mathrm{Re}Z$ – действительная часть комплексного числа Z, $\mathrm{Im}Z$ – его мнимая часть;

- системные константы, хранящие определённые параметры системы;

- строковые константы – любые цепочки символов, заключенные в кавычки, например «string», «2+3» (арифметические выражения в строковых константах рассматриваются как текст и не вычисляются);

- единицы измерения физических величин.

Знак умножения * при выводе числа на экран меняется на привычную точку, а операция возведения в степень (с применением спецзнака ^) отображается путем представления порядка в виде надстрочного числа. Диапазон возможных значений лежит в пределах от 10^{307} 10^{-307} (это есть машинная бесконечность и машинный ноль).

Для проведения физических расчетов в MathCAD может применяться особый вид констант – единицы измерения размерных величин. Помимо своего числового значения они характеризуются еще и указанием на то, к какой физической величине они относятся. Для этого указания используется символ умножения. В системе MathCAD заданы следующие основные типы физических величин: время, длина, масса, заряд и др. При необходимости их можно заменить на иные. Важно отметить, что MathCAD выполняет физические расчёты с соответствующим преобразованием размерных величин.

7.3.3 Переменные

Переменные – поименованные объекты, которым можно присваивать различные значения. Имена констант, переменных и иных объектов называют идентификаторами. В MathCAD тип переменной определяется ее значением – переменные могут быть числовыми, строковыми, символьными и т.д. Поэтому тип переменной предварительно не задаётся.

Идентификаторы в MathCAD могут иметь практически любую длину. При их задании можно использовать латинские и греческие буквы, а также цифры. Однако начинаться идентификатор может только с латинской буквы. Пробелы в идентификаторах вводить нельзя, но допустимо применение некоторых спецсимволов (например, знак подчеркивания «_»). Применение знаков операторов арифметических операций недопустимо, поскольку ведет к неоднозначности идентификации переменной. Малые и большие буквы в идентификаторах различаются. Идентификаторы должны быть уникальными, т.е. они не могут совпадать с именами встроенных или определённых пользователем функций.

Для присваивания переменным значений используется оператор присваивания в виде «:=», вводимый двоеточием. Если переменной не будет присвоено никакого значения, то ее применение в каком-нибудь выражении вызовет ошибку. Все ошибки диагностируются и требуют исправления для продолжения вычислений с незаданной переменной. Для того, чтобы вывести значение какой-нибудь переменной, необходимо набрать после соответствующего идентификатора знак «=».

Пример присваивания переменным значений:

a := 5

b := 10

c := a + b

c = 15

В этом примере переменной a присваивается число пять, переменной b – десять, а переменой c – сумма переменных a и b (строкой ниже выведено значение переменной c).

В математике часто возникает необходимость в задании некоторого ряда значений, чаще всего упорядоченного. Например, для вычисления факториала $N!=1\cdot2\cdot3\cdot\ldots\cdot(N-1)\cdot N$ нужно сформировать ряд целых чисел от 0 до N с шагом 1. Часто ряд значений какой-то переменной (например, абсциссы x) нужен для построения графика функции по точкам обычно соединяемым отрезками прямых. Для создания таких рядов в MathCAD используется так называемые **диапазонные переменные**. Иногда они заменяют управляющие структуры – циклы, но полноценной такая замена не является. В частности потому, что не предусмотрен выбор любого значения диапазонной переменной. В самом простом случае для создания диапазонной переменной используется выражение

NAME := Nbegin .. Nend,

Где Name – имя переменной; Nbegin – ее первоначальное значение; Nend – конечное значение; .. – символ, указывающий на изменение переменной в заданных пределах (он вводится знаком точка с запятой «;»).

Если Nbegin < Nend, то шаг изменения переменной будет равен плюс 1, в противном случае – минус 1.

Для создания диапазонной переменной общего вида используется выражение NAME := Nbegin, (Nbegin + Step) .. Nend.

Здесь Step – заданный шаг изменения переменной (он может быть положительным, если Nbegin < Nend, или отрицательным в противном случае).

Диапазонные переменные широко применяются для представления численных значений функций в виде таблиц, а также для построения графиков. Знак равенства после любого выражения с диапазонными переменными инициирует таблицу вывода значений выражения по диапазонным переменным.

Пример применения диапазонных переменных:

$k := 1 .. 4$; $m := 0, 2 .. 8$;

$$k = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix};\ m = \begin{pmatrix} 0 \\ 2 \\ 4 \\ 6 \\ 8 \end{pmatrix};\ k^2 = \begin{pmatrix} 1 \\ 4 \\ 9 \\ 16 \end{pmatrix}.$$

В данном примере используется две диапазонные переменные: переменная k применяется от одного до четырех с шагом плюс один, а переменная m изменяется от нуля до восьми с шагом плюс два. Также в этом примере показано использование диапазонной переменной в выражении (возведение в квадрат).

В MathCAD имеются и так называемые **системные переменные** с заранее определенными именами и

начальными значениями. Они прежде всего необходимы для инициализации некоторых начальных установок самой системы. Однако значения системных переменных могут быть в дальнейшем изменены пользователем путем переписывания из значений. Основными системными переменными являются:

TOL – значение погрешности численных методов (начальное значение равно 0,001);

ORIGIN – нижняя граница индексации массивов (начальное значение равно 0);

FRAME – переменная счетчика кадров при работе с анимационными рисунками (начальное значение равно 0);

π – число «пи»(начальное значение равно 3,1415926…);

e – основание натурального логарифма (начальное значение равно 2,71…);

∞ - системная бесконечность (начальное значение равно 10^{307}).

В MathCAD существует возможность ввода массива. Сам массив задаётся именем, как и любая переменная. Однако он имеет ряд элементов с определённым порядком расположения. Порядковый номер элемента задается индексом. Нижняя граница индексации определяется значением системной переменной ORIGIN. Таким образом, элементы массива являются **индексированными переменными**. Это значит, что помимо имени такие переменные имеют подстрочный индекс, который вводится с помощью знака «[».

Вектор (массив размера n x 1) или матрица (массив размера n x m) могут быть созданы присваиванием их элементам тех или иных значений. Незаданные элементы по умолчанию являются нулевыми (например, присвоены значения нулевому, первому, третьему элементам вектора и в этом случае второй элемент заменится на нуль, так как

ему значение не присвоено). Вместо указания индекса массива в виде конкретного числа можно использовать диапазонные переменные. Если диапазонная переменная превысит текущую размерность массива автоматически увеличится до максимального значения, которое принимает сама диапазонная переменная.

Пример задания матрицы:

$i := 0 .. 3$; $j := 0 .. 3$;

$A_{i,j} := i - j$;

$$A = \begin{pmatrix} 0 & -1 & -2 & -3 \\ 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 \\ 3 & 2 & 1 & 0 \end{pmatrix};$$

$A_{0,0} = 0$; $A_{1,1} = 0$; $A_{2,3} = -1$;

В этом примере матрица A имеет размерность 4x4, а элементы матрицы вычисляются по определённой формуле. Показан вывод всей матрицы, а также некоторых ее элементов.

7.3.4 Операторы

Операторы – элементы языка, с помощью которых можно создавать математические выражения. К ним, например, относятся операторы арифметических действий, операторы вычисления сумм, произведений, производной, интеграла и т.д. После указания операндов (аргументов соответствующих операций) операторы становятся исполняемыми программными блоками. MathCAD позволяет задавать и новые операторы, определённые пользователем.

Арифметические операторы предназначены для выполнения арифметических действий над численными величинами (операндами) и конструирования математи-

ческих выражений. Самыми распространенными являются операторы арифметических действий («+», «-», «·» и «/»), возведения в степень «^», извлечения квадратного корня. Ниже показаны примеры использования арифметических операторов.

По приведённым примерам видно, что MathCAD оперирует как с действительными, так и с комплексными величинами.

Пример 1 (операции со скалярными величинами):

$a := 2 \qquad b := 12$

$c := a + b \qquad c = 14 \qquad d := a - b \qquad d = -10$

$e := a \cdot b \qquad e = 24 \qquad f := \frac{a}{b} \qquad f = 0.167$

$g := a^{b} \qquad g = 4.096 \times 10^{3}$

$\sqrt{144} = 12 \qquad |3| = 3 \qquad |-3| = 3$

$-a = -2$

$5! = 120$

В данном примере переменным a и b присваиваются определённые значения. Переменной c присваивается сумма a и b, d – разность, e – произведение, f – деление, g – возведение числа a в степень b. Далее показаны применение операторов извлечения квадратного корня из 144, взятия по модулю числа a и оператор расчёта факториала от пяти.

Пример 2 (операции с комплексными числами)

$i := \sqrt{-1}$

$z := 2 + 3 \cdot i \qquad |z| = 3.606 \qquad \arg(z) = 0.983$

$\sqrt{z} = 1.674 + 0.896i$

$a := 4 + 7 \cdot i \qquad b := 9 - 2 \cdot i$

$c := a + b \qquad c = 13 + 5i \qquad d := a - b \qquad d = -5 + 9i$

$e := a \cdot b \qquad e = 50 + 55i \qquad f := \frac{a}{b} \qquad f = 0.259 + 0.835i$

В этом примере вводится мнимая единица как квадратный корень из минус единицы. У мнимого числа z рассчитывается его модуль и аргумент, а также показан результат извлечения квадратного корня. Далее видны результаты сложения, вычитания, умножения и деления комплексных чисел.

MathCAD содержит расширенные арифметические операторы: вычисление суммы и произведения ряда величин, вычисления производной и определённого интеграла. Применение расширенных операторов облегчает решение математических задач – раньше для выполняемых ими действий приходилось писать отдельные программы.

Ниже предлагаются варианты применения этих операторов. Заметим, что выражения с ними возвращают вычисленные значения, поэтому их можно использовать в составе сложных математических выражений.

Пример применения расширенных арифметических операторов:

$i := 1..10;$

$\sum_i i^2 = 385 \qquad \int_0^1 x^3 \, dx = 0.25$

$x := 10$

$\frac{d}{dx}\sin(x) = -0.839$ $\qquad\qquad \cos(x) = -0.839$

В данном примере показаны операторы суммирования, взятия интеграла и операторы дифференцирования с проверкой.

Очень часто в математических расчётах необходимо судить о равенстве и неравенстве величин, например значений переменных или выражений. Для этого служат операторы:

$X > Y$	X больше Y;
$X < Y$	X меньше Y;
$X \geq Y$	X больше или равно Y;
$X \leq Y$	X меньше или равно Y;
$X \neq Y$	X не равно Y;
$X = Y$	X равно Y.

Не следует путать оператор сравнения (знак равенства) с похожим знаком вывода значений переменных. В системе MathCAD знак равенства как оператор отношения имеет больший размер и более жирное написание, чем обычный знак равенства – оператор вывода. Все операторы отношения могут вводиться самостоятельно в месте расположения курсора. В этом случае по обе стороны от них появляются маленькие тёмные прямоугольники. Они являются местами ввода подлежащих сравнению выражений. Например, если ввести знак «меньше», то на экране дисплея появится блок вида

$\blacksquare < \blacksquare$

Знак вывода при этом будет появляться с сообщением об ошибке «Пропущенный операнд».

Необходимо отметить, что выражения с логическими операторами возвращают логическое значение, соответствующее выполнению или невыполнению условия, заданного оператором. Эти значения в MathCAD являются логической единицей 1, если условие выполнено, и логическим нулем 0, если оно не выполнено. Математически значения логической единицы и нуля совпадают со значениями числовых констант 1 и 0.

Например:

2>1=1 – условие выполнено, результат 1;

1>2=0 – условие не выполнено, результат 0.

Указанное свойство логических операторов позволяет строить не совсем обычные выражения, содержащие в себе логические операторы: 2*(5>0)=2.

Нетрудно понять, что выражение 5>0 возвращает единицу, поэтому результат вычисления такого выражения даст число 2. Логические операторы часто используются совместно с условными функциями. Их также используют в программных модулях для организации ветвлений, зависящих от значений обрабатываемых ими данных.

7.3.5 Функция

MathCAD имеет множество встроенных функций. Функции обладают особым свойством – в ответ на обращение к ним по имени с указанием аргумента (или списка аргументов) в круглых скобках они возвращают некоторое значение (символьное, числовое, вектор или матрицу). В систему встроен ряд функций, например вычисление синуса sin(z), логарифма ln(z) и т.д. Наряду с встроенными функциями могут задаваться и функции пользователя, описывающие произвольные, нужные пользователю функции, отсутствующие в наборе встроенных в MathCAD функций. Благодаря встроенным функциям

обеспечивается расширение входного языка MathCAD и его адаптация к задачам пользователя.

Во многих математических расчётах встречается необходимость вычисления **элементарных функций.** MathCAD содержит расширенный набор встроенных элементарных функций. Функции задаются своим именем и значением аргумента в круглых скобках. В ответ на обращения к ним функции возвращают вычисленные значения. Аргумент и значение функций могут быть действительными или комплексными числами.

Пример использования элементарных функций:

$\sin(0) = 0 \qquad \cos(0) = 1$

$\mathrm{asin}(0) = 0 \qquad \mathrm{acos}(1) = 0$

$\ln\left(e^{10}\right) = 10$

В данном примере вызываются следующие встроенные в MathCAD функции: вычисление синуса и косинуса нулевого угла, арксинуса от нуля, арккосинуса от единицы, натурального логарифма от числа e в десятой степени.

Для создания **условных выражений** используется функция **if:**

if (УСЛОВИЕ, ВЫРАЖЕНИЕ1, ВЫРАЖЕНИЕ2).

Если в этой функции условие УСЛОВИЕ выполняется, то будет вычисляться ВЫРАЖЕНИЕ1, в противном случае - ВЫРАЖЕНИЕ2.

Пример применения функции **if:**

$i := 0..4 \qquad j := 0..4$

$Aij := \mathrm{if}(i \geq j, i + j, 0)$

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 2 & 3 & 4 & 0 & 0 \\ 3 & 4 & 5 & 6 & 0 \\ 4 & 5 & 6 & 7 & 8 \end{pmatrix}$$

В этом примере происходит формирование нижней диагональной матрицы A.

Несмотря на довольно широкий набор встроенных функций, нередко возникает необходимость расширить систему новыми функциями, представляющими интерес для пользователя. **Функции пользователя** вводятся с применением следующего выражения:

```
ИмяФункции(СписокПараметров):=Выражение.
```

Имя функции задаётся как любой идентификатор. В скобках указывается список параметров функции – это перечень используемых в выражении переменных, разделяемых запятыми. Выражение – любое выражение, содержащее доступные системе операторы и функции с операндами и аргументами, указанными в списке параметров.

Пример задания функции:

$$\text{fun}(x) := 10 \cdot (1 - \exp(x))$$

$$\text{module}(x, y) := \sqrt{x^2 + y^2}$$

В этом примере задаются две пользовательские функции: функция fun(x) –функция от одной переменной x и функция module(x,y) – функция двух переменных x и y.

Следует отметить особый статус переменных, указанных в списке аргументов функции пользователя. Эти

переменные являются **локальными**, поэтому они могут не определяться до задания функций – фактически, их указание в списке параметров и является заданием определённого статуса этих переменных. Естественно, что локальные переменные могут использоваться только в выражении, описывающем функцию. Их имена могут совпадать и именами глобальных переменных, введённых ранее. Но при этом при выходе из блока задания функции значения этих переменных будут сохранять ранее заданные (для глобальных переменных) значения.

7.3.6 Программные операторы

Вплоть до появления последних версий системы MathCAD возможности программирования в ней были крайне ограниченными. Фактически, MathCAD позволял реализовать лишь линейные программы, осуществляя функциональное программирование, в основе которого лежит понятие функции. Условная функция и диапазонные переменные в отдельных случаях могли заменить условные выражения и циклы, но с серьёзными ограничениями. Эти возможности появились в версии MathCAD 6.0 PLUS и в расширенном варианте имеются в 7.0 и выше версиях.

Программный модуль в системе MathCAD представляет собой самостоятельный модуль, выделяемый в тексте документа жирной вертикальной чертой. Модуль может вести себя как функция без имени и параметров, но возвращающая результат. Программный модуль может выполнять и роль тела функции пользователя с именем и параметрами.

В MathCAD набор программных элементов для создания программных модулей содержит следующие элементы:

Add Line – создаёт и при необходимости расширяет жирную вертикальную линию, справа от которой в шаблонах задаётся запись программного блока;

▮ ← ▮ - символ локального присваивания (в теле модуля);

if – оператор условного выражения;

otherwise – оператор иного выбора (применяется с **if**)

for – оператор задания цикла с фиксированным числом отклонений;

while – оператор задания цикла типа «пока» (цикл выполняется, пока выполняется некоторое условие);

break – оператор прерывания;

continue – оператор продолжения;

return – оператор-функция возврата;

on error – оператор обработки ошибок.

Оператор **Add Line** выполняет функции расширения программного блока. Расширение фиксируется удлинением вертикальной черты программных блоков или их древовидным расширением. Благодаря этому, в принципе, можно создавать сколь угодно большие программы.

Оператор ▮ ← ▮ выполняет функции внутреннего локального присваивания. Например, выражение

x ← 123 присваивает переменной x значение 123. Локальный характер присваивания означает, что такое значение x сохраняет только в теле программы. За пределами тела программы значение переменной x может быть неопределённым либо равно значению, которое задаётся оператором присваивания вне программного блока.

Оператор **if** является оператором для создания условных выражений. Она задаётся в виде: ВЫРАЖЕНИЕ **if** УСЛОВИЕ. Если УСЛОВИЕ выполняется, то возвращается значение ВЫРАЖЕНИЕ. Совместно с этим операто-

ром часто используются оператор прерывания **break** и оператор иного выбора **otherwise**.

Оператор **otherwise** («иначе») обычно используется совместно с оператором **if**. Его использование поясняет следующая программная конструкция:

$$f(x) := \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{otherwise} \end{cases}$$

Функция f(x) возвращает единицу, если x>0, либо возвращает -1 во всех остальных случаях.

Оператор **for** служит для организации циклов с заданным числом повторений. Он записывается в виде:

for ПЕРЕМЕННАЯ ∈НАЧАЛЬНОЕ_ЗНАЧЕНИЕ [, НАЧАЛЬНОЕ_ЗНАЧЕНИЕ +ШАГ] .. КОНЕЧНОЕ_ЗНАЧЕНИЕ

Эта запись означает, что если переменная ПЕРЕМЕННАЯ меняется с шагом ШАГ (по умолчанию принимается равным +1, если НАЧАЛЬНОЕ_ЗНАЧЕНИЕ<КОНЕЧНОЕ_ЗНАЧЕНИЕ; либо -1, если НАЧАЛЬНОЕ_ЗНАЧЕНИЕ> КОНЕЧНОЕ_ЗНАЧЕНИЕ) от значения НАЧАЛЬНОЕ_ЗНАЧЕНИЕ до КОНЕЧНОЕ_ЗНАЧЕНИЕ, то выражение, помещённое в шаблон, будет выполняться. Переменную счётчика ПЕРЕМЕННАЯ можно использовать в выражениях программы.

Оператор **while** служит для организации циклов, действующих до тех пор, пока выполняется некоторое условие. Оператор записывается в виде: while УСЛОВИЕ.

Оператор **break** вызывает прерывание работы программы всякий раз, как он встречается. Чаще всего он используется с оператором условного выражения **if** и операторами циклов **while** и **for**, обеспечивая переход в коней тела цикла.

Оператор продолжения **continue** используется для продолжения работы после прерывания программы. Он

также используется обычно совместно с операторами задания циклов **while** и **for**, обеспечивая после прерывания возврат в начало цикла.

Особый оператор-функция **return** прерывает выполнение программы и возвращает значение своего операнда, стоящего следом за ним.

Оператор обработки ошибок **on error** позволяет создавать конструкции обработчиков ошибок. Этот оператор задаётся в виде:

ВЫРАЖЕНИЕ_1 on error ВЫРАЖЕНИЕ_2.

Здесь если при выполнении ВЫРАЖЕНИЕ_2 возникает ошибка, то выполняется ВЫРАЖЕНИЕ_1. Для обработки ошибок также полезна функция error(S), которая, будучи в программном модуле, возвращает окошко с надписью, хранящейся в символьной переменной S или в символьной константе.

Несмотря на столь скромный набор программных средств, они дают MathCAD именно те возможности, которые ранее просто отсутствовали: задание функций с аппаратом локальных переменных, задание различных видов циклов (в том числе вложенных), упрощение алгоритмов применением операции присваивания и реализацию по классическим алгоритмам итерационных и рекурсивных процедур.

Для нескольких подмодулей, которые должны выполняться в составе циклов, нужно использовать их объединение в виде жирной вертикальной черты. Для этого служит команда **Add Line**, как и для исходного задания блока, добавляющая в модуль дополнительную черту для подмодуля.

Программный модуль, в сущности, является функцией, но описанной с применением упомянутых сугубо программных средств. Она может возвращать значение, определяемое последним оператором. Это значит, что после такого модуля, выделенного как целый блок,

можно поставить знак равенства для вывода значения функции. В блоке могут содержаться любые операторы и функции входного языка системы. Для передачи в блок значений переменных можно использовать переменные документа, которые ведут себя в блоке как глобальные переменные.

Обычно модулю присваивается имя со списком переменных, после которого идёт знак присваивания «:=». Переменные в списке являются локальными, и им можно присваивать значения при вызове функции, заданной модулем. Локальный характер таких переменных позволяет использовать для их имён (идентификаторов0 те же имена, что и у глобальных переменных документа.

Задание программных модулей позволяет реализовать любые специальные приёмы программирования. Оно может служить мощным средством расширения системы путём создания новых функций. Ниже приведены примеры применения программных операторов.

Пример 1 (использование условного оператора):

$$\text{abs}(x) := \begin{cases} -x & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

$$\text{abs}(-10) = 10$$

$$\text{abs}(10) = 10$$

В этом примере задаётся функция abs(x), которая вычисляет модуль аргумента x. Эта функция возвращает значение –x, если аргумент меньше нуля, либо он возвращает x в противном случае.

Пример 2 (использование оператора for):

```
sum(n) := | s ← 0
          | for i ∈ 1..n
          |   s ← s + i
          | s
```

sum(10) = 55

sum(20) = 210

В этом примере sum(n) рассчитывает сумму n первых членов ряда натуральных чисел. В начале программного модуля локальной переменной s (в ней идёт накапливание суммы) присваивается нуль. После этого в операторе **for** в качестве счётчика цикла используется переменная i (она меняется от единицы до n – фактически она показывает номер текущего члена ряда). Внутри оператора цикла текущее значение суммы увеличивается на текущее значение очередного члена ряда. После окончания работы оператора **for** в переменной s храниться сумма n первых членов ряда натуральных чисел. Она-то и будет результатом работы функции sum(n).

Пример 3 (использование оператора **while**):

```
F(n) := | f ← n
        | while n > 1
        |     | F ← F(n − 1)
        |     | n ← n − 1
        | f
```

F(3) = 6

F(10) = 3628800

В данном примере функция F(n) вычисляет факториал числа n. В локальной переменной f хранится текущий результат расчёта факториала (в начале выполнения программного модуля он равен самому числу n). Вход в цикл и работа оператора **while** осуществляется по условию: n>1. Внутри самого цикла выполняются два оператора локального присваивания: изменение текущего результата расчёта факториала и уменьшение числа n не единицу (нужно обеспечить условие выхода из цикла). По окончании работы цикла **while** значению функции F(n) присваивается окончательный результат расчёта факториала.

Пример 4 (использование оператора обработки ошибок **on error**):

$$\mathrm{div}(x,y) := \left| \begin{array}{l} d \leftarrow \infty \quad \text{on error } d \leftarrow \dfrac{x}{y} \\ \\ d \end{array} \right.$$

$$\mathrm{div}(10,5) = 2$$

$$\mathrm{div}(55.5,5) = 11.1 \qquad \mathrm{div}(10,0) = 1 \times 10^{307}$$

В этом примере функция div(x,y) делит число x на y, причём в самой функции предусмотрена обработка ситуации деления на ноль. Если окажется, что поделить x на y не получится (будет ошибка деления на ноль), то тогда переменной d принудительно присваивается максимальное значение.

7.3.7 Графики

Для создания графиков в MathCAD имеется программный графический процессор. Основное внимание

при его разработке было уделено обеспечению простоты задания графиков и их модификации с помощью соответствующих опций. Процессор позволяет строить самые разные графики, например, в декартовой и полярной системах координат, трёхмерные поверхности, графики уровней и т.д.

7.3.8 Символьные вычисления

Для выполнения символьных вычислений MathCAD дополнен символьным ядром (процессором). Символьные операции применяются к целым выражениям, к отдельным переменным в выражениях, к матрицам, применяются и для преобразований Лапласа, Фурье, Z-преобразования. Также предусмотрено вычисление предела аналитически заданной функции.

7.4 Построение графиков функций

7.4.1 Построение графика функции одной переменной в декартовой системе координат

Найти область определения функции. Указать диапазон изменения аргумента с определенным шагом. Диапазон необходимо выбрать таким образом, чтобы он полностью входил в область определения аргумента функции и не содержал особых точек. Определить функцию, зависящую от одного аргумента. Имена функции и аргумента выбирать произвольно. Построить график функции.

$$f(x) := \begin{cases} \dfrac{x+2}{x^2+9}, x < 0 \\ \dfrac{|x-1|}{x^2+4}, x \geq 0 \end{cases}$$

Область определения функции f(x) не имеет разрывов, и функция определена на всей оси Ox. Чтобы построить график этой функции зададим переменную диапазона x, с шагом 0.2.

$$x := -10, -9.8 .. 10$$

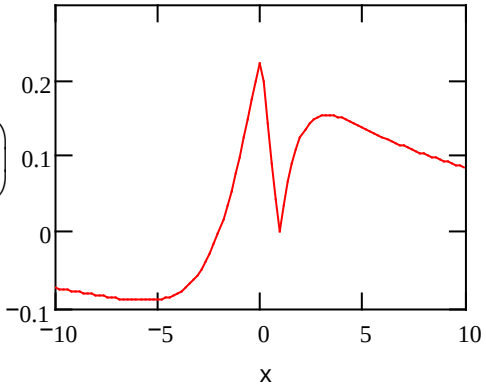
При помощи функции if определим функцию f(x). В случае если выполняется заданное условие, функция будет рассчитываться по первому выражению, во всех остальных случаях по второму.

А теперь построим график этой функции в декартовой системе координат. В окне построения графика в качестве аргумента зададим переменную диапазона x, в качестве функции

$$\text{if}\left(x < 0, \frac{x+2}{x^2+9}, \frac{|x-1|}{x^2+4}\right).$$

При этом ось Ox будет ограничиваться значениями -10 и 10, а ось Oy в оптимальном масштабе.

$$\text{if}\left(x<0,\ \frac{x+2}{x^2+9},\ \frac{|x-1|}{x^2+4}\right)$$



7.4.2 Построение поверхности

Построить поверхность, заданную функцией двух аргументов. Интервал и количество точек выбирать произвольно.

$$z := \frac{3x^2 \cdot e^y - xy}{x^2 + y^2}$$

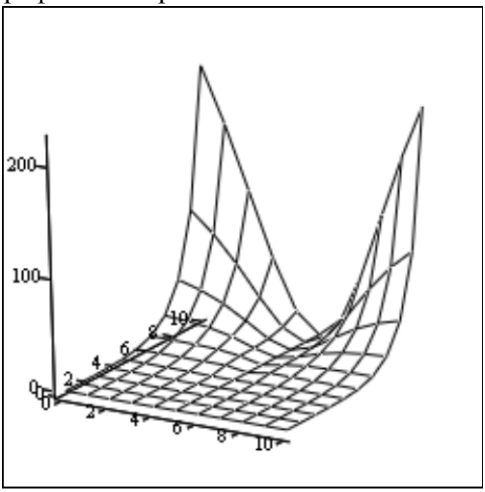
Для построения заданной поверхности воспользуемся двумерным массивом A, размером 10x10. Строки и столбцы его будут выступать в роли координатной сетки плоскости XOУ, а значения - координату z. Интервал по осям Ox и Oy одинаковый, от -5 до 5, но так значения номера строки или столбца не могут быть отрицательными, то значения переменных диапазона x и y будут изменятся от 0 до 10 с шагом 1, при этом аргументы в уравнении плоскости z будут (x-5) и (y-5) соответственно.

$x := 0 .. 10$

$y := 0 .. 10$

$$A_{x,y} := \frac{3(x-5)^2 \cdot e^{y-5} - (x-5)\cdot(y-5)}{(x-5)^2 + (y-5)^2}$$

Теперь отобразим значения массива A в окне построения графика поверхности.



A

7.4.3 Построение графика параметрический заданной функции

Определить y(t) как функцию, зависящую от параметра.

Построить график функции.

$$x := (a + b) \cdot \cos(2t) \qquad y := (a - b) \cdot \sin(4t) \qquad t := 0..2\pi$$

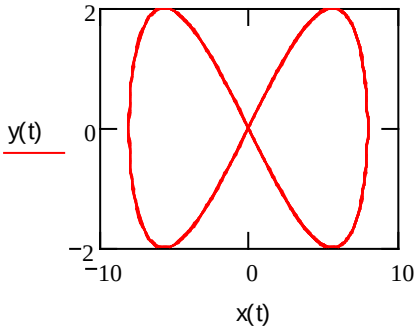
Так как параметры a и b не заданы, определим их самостоятельно. Пусть a = 5, b = 3

$$a := 5 \qquad b := 3$$

$$x(t) := (a + b) \cdot \cos(2t)$$

$$y(t) := (a - b) \cdot \sin(4t)$$

В окне построения графика в качестве аргумента введем x(t), в качестве функции y(x).



7.5 Решение систем линейных уравнений

7.5.1 Решение СЛАУ методом Крамера

Написать при помощи элементов программирования функцию, которая находит решение системы линейных уравнений методом Крамера. Вычисление определителя производить при помощи соответствующей операции MathCad. Найти решение системы при помощи данной функции.

$$A := \begin{pmatrix} 1 & 0 & 8 & 0 \\ 1 & -3 & 9 & -4 \\ 2 & 6 & 1 & 7 \\ 0 & -2 & -6 & -4 \end{pmatrix} \qquad b := \begin{pmatrix} 25 \\ 17 \\ 25 \\ -28 \end{pmatrix}$$

Функция Kram(A,b) содержит два аргумента: A - основная матрица, и b - матрица-столбец свободных чле-

нов. Сначала функция вычисляет ранг матрицы n, это значение меньше на 1 истинного ранга матрицы, т.к MathCad начинает нумерацию с 0, а не с 1. Следующая локальная переменная O, равна определителю основной матрицы A. Затем в цикле со счетчиком i от 0 до n, производится вычисление каждого неизвестного по правилу Крамера. Итогом является матрица столбец значений соответствующих неизвестных.

$$\text{Kram}(A,b) := \left| \begin{array}{l} n \leftarrow \text{rank}(A) - 1 \\ O \leftarrow |A| \\ \text{for } i \in 0..n \\ \quad \left| \begin{array}{l} M \leftarrow A \\ M^{\langle i \rangle} \leftarrow b \\ X_i \leftarrow \dfrac{|M|}{O} \end{array} \right. \\ X \end{array} \right.$$

$$X := \text{Kram}(A,b)$$

$$X = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 2 \end{pmatrix}$$

Сделаем проверку, подставив в каждое уравнение полученные значения переменных.

$$A_{0,0} \cdot X_0 + A_{0,1} \cdot X_1 + A_{0,2} \cdot X_2 + A_{0,3} \cdot X_3 = 25$$

$$A_{1,0} \cdot X_0 + A_{1,1} \cdot X_1 + A_{1,2} \cdot X_2 + A_{1,3} \cdot X_3 = 17$$

$$A_{2,0} \cdot X_0 + A_{2,1} \cdot X_1 + A_{2,2} \cdot X_2 + A_{2,3} \cdot X_3 = 25$$

$$A_{3,0} \cdot X_0 + A_{3,1} \cdot X_1 + A_{3,2} \cdot X_2 + A_{3,3} \cdot X_3 = -28$$

7.5.2 Решение СЛАУ методом Гаусса

Написать функцию, которая находит решение системы линейных уравнений методом Гаусса. Найти решение системы при помощи данной функции. Проверить результат.

$$A := \begin{pmatrix} -10 & -13 & 13 & -1 & 1 & 1 \\ -5 & -4 & -14 & 16 & -1 & -2 \\ -1 & -2 & 15 & 4 & 0 & -5 \\ -14 & 7 & -6 & -4 & -6 & -13 \\ 14 & -9 & -3 & -3 & -5 & -10 \\ -14 & 1 & 3 & 6 & -8 & -6 \end{pmatrix} \qquad b := \begin{pmatrix} 106 \\ 84 \\ 9 \\ 91 \\ -47 \\ 111 \end{pmatrix}$$

Функция Gaus(A,b) содержит два аргумента: A - основная матрица, и b - матрица-столбец свободных членов. Сначала функция объединяет матрицы A и b с помощью функции augment. Затем функцией rref она сводит основою матрицу к единичной. Итогом является матрица столбец значений соответствующих неизвестных, выделенная из последнего столбца расширенной матрицы C.

$$Gaus(A,b) := \left(\begin{array}{l} C \leftarrow augment(A,b) \\ C \leftarrow rref(C) \\ n \leftarrow rank(C) \\ X \leftarrow C^{\langle n \rangle} \\ X \end{array} \right) \qquad X := Gaus(A,b)$$

$$X = \begin{pmatrix} -7 \\ -4 \\ -1 \\ 1 \\ -1 \\ -1 \end{pmatrix}$$

7.6 Матричные операции

Определить матрицы. Размерность матриц подобрать с учетом выражения. Количество строк и столбцов - не менее четырех. Обязательно наличие неквадратных матриц.

$$(A \cdot D - D \cdot C) + B \cdot E$$

Количество столбцов матрицы A и строк матрицы D должно быть равным, пусть оно будет n. С другой стороны число столбцов матрицы D должно быть равно числу строк матрицы C, значит оно тоже равно n. Так как произведения матриц A на D и D на C должно дать матрицы одинаковых размеров и при этом квадратные (чтобы можно посчитать определитель), то размеры матриц A,D и C - одинаковы, nxn. Так как произведение матриц B и E, должно дать также матрицу nxn, то размер B - nxk, а E - kxn. Пусть n будет равно 5, а k - 6. Тогда зададим переменные диапазона n и k, т.к MathCad начинает нумерацию строк и столбцов с 0, то их значения будут 0..4 и 0..5 соответственно.

$n := 0 .. 4$ $k := 0 .. 5$

Заполнить матрицы случайными значениями при помощи встроенных функций (не вручную).

Воспользуемся функцией rnd. Зададим предел значений 1, тогда значения будут варьироваться от 0 до 0.999. Умножим это значение на 10 и округлим до целого при помощи функции round. Так получится значение от 0 до 10. Заполнение будет производится при помощи двух вложенных циклов for, первый будет перебирать строки, второй – столбцы.

$$A := \left| \begin{array}{l} \text{for } i \in n \\ \quad \text{for } j \in n \\ \quad\quad M_{i,j} \leftarrow \text{round}(\text{rnd}(1) \cdot 10) \\ M \end{array} \right. \qquad A = \begin{pmatrix} 0 & 2 & 6 & 4 & 8 \\ 2 & 7 & 3 & 1 & 1 \\ 10 & 1 & 0 & 5 & 6 \\ 2 & 5 & 1 & 8 & 5 \\ 9 & 10 & 5 & 5 & 9 \end{pmatrix}$$

$$D := \left| \begin{array}{l} \text{for } i \in n \\ \quad \text{for } j \in n \\ \quad\quad M_{i,j} \leftarrow \text{round}(\text{rnd}(1) \cdot 10) \\ M \end{array} \right. \qquad D = \begin{pmatrix} 8 & 10 & 6 & 3 & 8 \\ 4 & 7 & 0 & 3 & 6 \\ 8 & 5 & 7 & 5 & 7 \\ 6 & 7 & 6 & 2 & 4 \\ 5 & 8 & 2 & 5 & 7 \end{pmatrix}$$

$$C := \left| \begin{array}{l} \text{for } i \in n \\ \quad \text{for } j \in n \\ \quad\quad M_{i,j} \leftarrow \text{round}(\text{rnd}(1) \cdot 10) \\ M \end{array} \right. \qquad C = \begin{pmatrix} 1 & 1 & 7 & 4 & 10 \\ 2 & 8 & 2 & 8 & 2 \\ 7 & 3 & 7 & 7 & 1 \\ 8 & 5 & 4 & 9 & 5 \\ 5 & 8 & 5 & 10 & 7 \end{pmatrix}$$

$$B := \left| \begin{array}{l} \text{for } i \in n \\ \quad \text{for } j \in k \\ \quad\quad M_{i,j} \leftarrow \text{round}(\text{rnd}(1) \cdot 10) \\ M \end{array} \right. \qquad B = \begin{pmatrix} 2 & 8 & 5 & 0 & 6 & 5 \\ 8 & 7 & 8 & 1 & 3 & 3 \\ 1 & 8 & 6 & 3 & 0 & 8 \\ 2 & 6 & 1 & 8 & 5 & 4 \\ 7 & 4 & 6 & 6 & 5 & 7 \end{pmatrix}$$

$$E := \left| \begin{array}{l} \text{for } i \in k \\ \quad \text{for } j \in n \\ \quad\quad M_{i,j} \leftarrow \text{round}(\text{rnd}(1) \cdot 10) \\ M \end{array} \right. \qquad E = \begin{pmatrix} 2 & 2 & 5 & 1 & 1 \\ 9 & 9 & 2 & 4 & 6 \\ 5 & 6 & 9 & 6 & 6 \\ 2 & 6 & 2 & 6 & 6 \\ 5 & 7 & 6 & 8 & 6 \\ 9 & 7 & 7 & 3 & 3 \end{pmatrix}$$

Вычислить заданное матричное выражение

Для дальнейшей работы с полученной матрицей приравняем её к матрице F.

$$F := (A \cdot D - D \cdot C) + B \cdot E \qquad F = \begin{pmatrix} 162 & 134 & 54 & -50 & 80 \\ 170 & 151 & 124 & 13 & 110 \\ 184 & 224 & 69 & -52 & 104 \\ 159 & 209 & 56 & 19 & 136 \\ 297 & 342 & 205 & 80 & 262 \end{pmatrix}$$

Для результирующей матрицы выполнить следующее

Сложить каждые элемент со скаляром x1.

Исходные данные x1=24.45.

$$x1 := 24.45$$

$$F := F + x1$$

$$F = \begin{pmatrix} 186.45 & 158.45 & 78.45 & -25.55 & 104.45 \\ 194.45 & 175.45 & 148.45 & 37.45 & 134.45 \\ 208.45 & 248.45 & 93.45 & -27.55 & 128.45 \\ 183.45 & 233.45 & 80.45 & 43.45 & 160.45 \\ 321.45 & 366.45 & 229.45 & 104.45 & 286.45 \end{pmatrix}$$

Вычесть из каждого элемента скаляр x2.

Исходные данные x2=69.54.

$x2 := 69.54$

$F := F - x2$

$$F = \begin{pmatrix} 116.91 & 88.91 & 8.91 & -95.09 & 34.91 \\ 124.91 & 105.91 & 78.91 & -32.09 & 64.91 \\ 138.91 & 178.91 & 23.91 & -97.09 & 58.91 \\ 113.91 & 163.91 & 10.91 & -26.09 & 90.91 \\ 251.91 & 296.91 & 159.91 & 34.91 & 216.91 \end{pmatrix}$$

Вычислить определитель матрицы.

Воспользуемся встроенной функцией MathCad.

$|F| = -1.678 \times 10^{8}$

Найти обратную матрицу и проверить результат.

Для нахождения обратной матрицы воспользуемся встроенной функцией MathCad.

$Fobr := F^{-1}$

$$Fobr = \begin{pmatrix} -0.138 & 0.346 & -0.111 & 0.272 & -0.165 \\ -0.044 & 0.061 & -1.165 \times 10^{-3} & 0.047 & -0.031 \\ 0.051 & -0.13 & 0.05 & -0.123 & 0.068 \\ -0.141 & 0.309 & -0.093 & 0.246 & -0.148 \\ 0.206 & -0.44 & 0.109 & -0.33 & 0.212 \end{pmatrix}$$

Проверим результат, зная, что произведение определителей исходной и обратной матриц равно 1

$|Fobr| \cdot |F| = 1$

Найти собственные числа и собственные вектора и проверить результат.

Для нахождения собственных чисел воспользуемся функцией eigenvals, она выводит значения в вектор.

$$Sc := eigenvals(F) \qquad Sc = \begin{pmatrix} 387.409 \\ 122.026 \\ -89.088 \\ 14.444 \\ 2.759 \end{pmatrix}$$

Для нахождения собственных векторов воспользуемся функцией eigenvecs, она выводит значения векторов в столбцы матрицы.

$$Sv := eigenvecs(F) \qquad Sv = \begin{pmatrix} 0.092 & 0.271 & 0.391 & -0.695 & -0.561 \\ 0.267 & 5.028 \times 10^{-4} & -0.263 & 0.197 & -0.058 \\ 0.22 & 0.171 & 0.588 & 0.478 & 0.256 \\ 0.329 & -0.325 & 0.497 & -0.42 & -0.474 \\ 0.874 & -0.89 & -0.43 & 0.271 & 0.626 \end{pmatrix}$$

Для проверки воспользуемся свойством собственных чисел и векторов - $Av=\lambda v$, где A - квадратная матрица, v - ее собственный вектор, λ - соответствующее собственное число.

$$F \cdot Sv^{\langle 0 \rangle} = \begin{pmatrix} 35.647 \\ 103.268 \\ 85.281 \\ 127.444 \\ 338.579 \end{pmatrix} \qquad Sc_0 \cdot Sv^{\langle 0 \rangle} = \begin{pmatrix} 35.647 \\ 103.268 \\ 85.281 \\ 127.444 \\ 338.579 \end{pmatrix}$$

$$F \cdot Sv^{\langle 1 \rangle} = \begin{pmatrix} 33.045 \\ 0.061 \\ 20.911 \\ -39.626 \\ -108.586 \end{pmatrix}$$

$$Sc_1 \cdot Sv^{\langle 1 \rangle} = \begin{pmatrix} 33.045 \\ 0.061 \\ 20.911 \\ -39.626 \\ -108.586 \end{pmatrix}$$

$$F \cdot Sv^{\langle 2 \rangle} = \begin{pmatrix} -34.798 \\ 23.436 \\ -52.383 \\ -44.306 \\ 38.336 \end{pmatrix}$$

$$Sc_2 \cdot Sv^{\langle 2 \rangle} = \begin{pmatrix} -34.798 \\ 23.436 \\ -52.383 \\ -44.306 \\ 38.336 \end{pmatrix}$$

$$F \cdot Sv^{\langle 4 \rangle} = \begin{pmatrix} -1.547 \\ -0.16 \\ 0.706 \\ -1.307 \\ 1.728 \end{pmatrix}$$

$$Sc_4 \cdot Sv^{\langle 4 \rangle} = \begin{pmatrix} -1.547 \\ -0.16 \\ 0.706 \\ -1.307 \\ 1.728 \end{pmatrix}$$

$$F \cdot Sv^{\langle 3 \rangle} = \begin{pmatrix} -10.036 \\ 2.843 \\ 6.903 \\ -6.073 \\ 3.913 \end{pmatrix}$$

$$Sc_3 \cdot Sv^{\langle 3 \rangle} = \begin{pmatrix} -10.036 \\ 2.843 \\ 6.903 \\ -6.073 \\ 3.913 \end{pmatrix}$$

Вычислить ранг матрицы

Вычислим ранг с помощью функции MathCad - rank.

$$\mathrm{rank}(F) = 5$$

Определить вектора, которые состоят из элементов n-ой строки и m-го столбца.

Исходные данные n=3 m=4

Так как MathCad начинает нумерацию строк и столбцов с 0, то зададим переменным n и m значения 2 и 3 соответственно.

$$n := 2 \qquad m := 3$$

Для выделения строки n воспользуемся функцией выделения вектора из столбца матрицы, транспонировав перед этим матрицу. Для выделения столбца m просто воспользуемся этой функцией. Присвоим этим векторам имена v1 и v2.

$$v1 := \left(F^{T}\right)^{\langle n \rangle} \qquad v2 := F^{\langle m \rangle}$$

$$v1 = \begin{pmatrix} 138.91 \\ 178.91 \\ 23.91 \\ -97.09 \\ 58.91 \end{pmatrix} \qquad v2 = \begin{pmatrix} -95.09 \\ -32.09 \\ -97.09 \\ -26.09 \\ 34.91 \end{pmatrix}$$

Вычислить Модуль каждого вектора.

Воспользуемся функцией MathCad для определения модуля вектора.

$$|v1| = 254.506 \qquad |v2| = 146.28$$

Вычислить орт каждого вектора.

С помощью элементов программирования напишем функцию Ort(V,n), определяющую орт вектора V. С помощью цикла for рассчитывается n координат орта, по формуле координата орта равна соответствующей координате вектора деленной на его длину.

$$\mathrm{Ort}(V, n) := \left| \begin{array}{l} \text{for} \quad i \in 0 .. (n - 1) \\ \quad \mathrm{Ort}_i \leftarrow \dfrac{V_i}{|V|} \\ \mathrm{Ort} \end{array} \right.$$

$$\mathrm{Ort}(v1, 5) = \begin{pmatrix} 0.546 \\ 0.703 \\ 0.094 \\ -0.381 \\ 0.231 \end{pmatrix} \qquad \mathrm{Ort}(v2, 5) = \begin{pmatrix} -0.65 \\ -0.219 \\ -0.664 \\ -0.178 \\ 0.239 \end{pmatrix}$$

Проверим результат, зная, что длина орта равна 1.

$$|\mathrm{Ort}(v1, 5)| = 1 \qquad |\mathrm{Ort}(v2, 5)| = 1$$

Вычислить сумму координат каждого вектора.

С помощью элементов программирования напишем функцию Sum(V,n), рассчитывающую с помощью цикла for сумму n координат вектора V.

$$\text{Sum}(V, n) := \left| \begin{array}{l} S \leftarrow 0 \\ \text{for} \quad i \in 0 .. (n - 1) \\ \quad S \leftarrow S + V_i \end{array} \right.$$

$$\text{Sum}(v1, 5) = 303.55 \qquad \text{Sum}(v2, 5) = -215.45$$

Вычислить количество положительных и отрицательных координат каждого вектора.

С помощью элементов программирования напишем функцию Pos(V,n), рассчитывающую с помощью цикла for количество положительных координат вектора V, (n - количество координат вектора V). Аналогично напишем функцию Neg(V,n) для подсчета количества отрицательных элементов.

$$\text{Pos}(V, n) := \left| \begin{array}{l} \text{Pos} \leftarrow 0 \\ \text{for} \quad i \in 0 .. (n - 1) \\ \quad \text{Pos} \leftarrow \text{Pos} + 1 \quad \text{if} \quad V_i \geq 0 \\ \text{Pos} \end{array} \right.$$

$$\text{Neg}(V, n) := \left| \begin{array}{l} \text{Neg} \leftarrow 0 \\ \text{for} \quad i \in 0 .. (n - 1) \\ \quad \text{Neg} \leftarrow \text{Neg} + 1 \quad \text{if} \quad V_i < 0 \\ \text{Neg} \end{array} \right.$$

$$\text{Pos}(v1, 5) = 4 \qquad \text{Pos}(v2, 5) = 1$$

$$\text{Neg}(v2, 5) = 4 \qquad \text{Neg}(v2, 5) = 4$$

Вычислить скалярное произведение векторов.

Воспользуемся функцией скалярного вычисления векторов.

$$v1 \cdot v2 = -1.668 \times 10^4$$

Вычислить $Sin\gamma$ и $Cos\gamma$, где γ - угол между векторами.

Зная, что косинус угла между векторами равен скалярному произведению этих векторов деленному на их длины.

$$Cos\gamma := \frac{v1 \cdot v2}{|v1| \cdot |v2|} \qquad Cos\gamma = -0.448$$

По основному тригонометрическому тождеству вычислим синус угла между векторами.

$$Sin\gamma := \sqrt{1 - Cos\gamma^2} \qquad Sin\gamma = 0.894$$

Вычислить векторное произведение векторов.

Встроенная функция MathCad позволяет вычислить векторное произведение только для трехкомпонентных векторов. Поэтому вычислим векторное произведение, зная, что оно равно произведению длин векторов и синуса угла между ними.

$$|v1| \cdot |v2| \cdot Sin\gamma = 3.328 \times 10^4$$

Вычислить вектор, координаты которого являются произведением соответствующих координат двух векторов.

С помощью элементов программирования напишем функцию VP(V1,V2,n), рассчитывающую с помощью цикла for каждую координату нового вектора.

$$VP(V1, V2, n) := \left| \begin{array}{l} \text{for } i \in 0..(n-1) \\ \quad VP_i \leftarrow V1_i \cdot V2_i \\ VP \end{array} \right.$$

$$VP(v1, v2, 5) = \begin{pmatrix} -1.321 \times 10^4 \\ -5.741 \times 10^3 \\ -2.321 \times 10^3 \\ 2.533 \times 10^3 \\ 2.057 \times 10^3 \end{pmatrix}$$

7.7 Интегрирование

7.7.1 Определенный интеграл

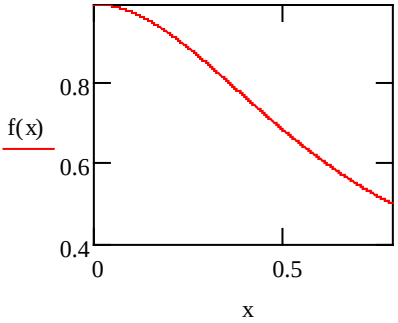
Построить график подынтегральной функции на интервале интегрирования. Вычислить значение интеграла.

$$\int_0^{\pi/4} \frac{dx}{1+2\sin^2 x}$$

Определим функцию f(x) равную подынтегральной функции.

$$f(x) := \frac{1}{1 + 2\cdot\sin(x)^2}$$

Построим график данной функции по оси Ox зададим интервал от 0 до π/4.



$$\int_0^{\frac{\pi}{4}} f(x)\,dx = 0.605$$

7.7.2 Неопределенный интеграл

Найти вручную первообразную для подынтегральной функции. Вычислить неопределенный интеграл при помощи MathCad.

$$\int \frac{3x+2}{\sqrt{x^2+x+2}}dx \qquad \int \frac{2-\sqrt[3]{tgx}\,dx}{\cos^2 x}$$

Вручную найдем первый неопределенный интеграл

$$\int \frac{3x+2}{\sqrt{x^2+x+2}}dx = 3\int \frac{x\,dx}{\sqrt{x^2+x+2}} + 2\int \frac{dx}{\sqrt{x^2+x+2}} = \left|x+\tfrac{1}{2}=t\right| =$$

$$= 3\int \frac{(t-\tfrac{1}{2})dt}{\sqrt{t^2+1\tfrac{3}{4}}} + 2\int \frac{dt}{\sqrt{t^2+1\tfrac{3}{4}}} = \frac{3}{2}\int \frac{d(t^2+1\tfrac{3}{4})}{\sqrt{t^2+1\tfrac{3}{4}}} + \frac{1}{2}\int \frac{dt}{\sqrt{t^2+1\tfrac{3}{4}}} =$$

$$= 3\sqrt{t^2 + 1¾} + ½\ln\left|t + \sqrt{t^2 + 1¾}\right| = 3\sqrt{x^2 + x + 2} + ½\ln\left|x + ½ + \sqrt{x^2 + x + 2}\right| + C$$

Вычислим его при помощи MathCad

$$\int \frac{3x + 2}{\sqrt{x^2 + x + 2}}\,dx \rightarrow 3\cdot\left(x^2 + x + 2\right)^{\frac{1}{2}} + \frac{1}{2}\cdot\operatorname{asinh}\left[\frac{2}{7}\cdot 7^{\frac{1}{2}}\cdot\left(x + \frac{1}{2}\right)\right]$$

MathCad выразил натуральный логарифм через гиперболический арксинус.

Вручную найдем второй неопределенный интеграл

$$\int \frac{2 - \sqrt[3]{tgx}}{\cos^2 x} = \int (2 - tg^{\frac{1}{3}}x)d(tgx) = 2tgx - \frac{3}{4}tg^{\frac{3}{4}}x + C$$

Вычислим его при помощи MathCad

$$\int \frac{2 - \sqrt[3]{\tan(x)}}{\cos(x)^2}\,dx \rightarrow \text{indef\_int}\left(\frac{2 - \tan(x)^{\frac{1}{3}}}{\cos(x)^2}, x\right)$$

MathCad не может вычислить данный неопределенный интеграл.

7.8 Дифференцирование

Найти производную заданной функции. Вычислить значение первой и второй производной в заданной точке.

$$y = \frac{x}{2}\sqrt{a^2 - x^2} + \frac{a^2}{2}\arcsin\frac{x}{2} \qquad x_0 = 0$$

Используя функцию вычисления производной, вычислим ее подставив ее в качестве функции.

$$\frac{d}{dx}\left(\frac{x}{2}\cdot\sqrt{a^2 - x^2} + \frac{a^2}{2}\cdot\operatorname{asin}\left(\frac{x}{2}\right)\right) \to \frac{1}{2}\cdot\left(a^2 - x^2\right)^{\frac{1}{2}} - \frac{1}{2}\cdot\frac{x^2}{\left(a^2 - x^2\right)^{\frac{1}{2}}} + \frac{1}{2}\cdot\frac{a^2}{\left(4 - x^2\right)^{\frac{1}{2}}}$$

Чтобы вычислить значения первой и второй производной зададим значение заданной точки переменной x и воспользуемся встроенными функциями.

$x := 0$

$$\frac{d}{dx}\left(\frac{x}{2}\cdot\sqrt{a^2 - x^2} + \frac{a^2}{2}\cdot\operatorname{asin}\left(\frac{x}{2}\right)\right) \to \frac{1}{2}\cdot\left(a^2\right)^{\frac{1}{2}} + \frac{1}{8}\cdot a^2\cdot 4^{\frac{1}{2}}$$

$$\frac{d^2}{dx^2}\left(\frac{x}{2}\cdot\sqrt{a^2 - x^2} + \frac{a^2}{2}\cdot\operatorname{asin}\left(\frac{x}{2}\right)\right) \to 0$$

7.9 Сплайн-интерполяция

Для заданного набора точек построить линейный, параболический и кубический сплайны.

Отобразить на одном графике исходные точки, линейный, параболический и кубический сплайны.

x	2.046	1.929	9.568	7.388	6.798	9.213	8.443	1.054	9.038	8.927
y	-0.4004	-0.401	-11.85	-6.206	-5.005	-10.81	-8.703	-0.5788	-10.31	-9.998

Запишем координаты по x и y в векторы vx и vy соответственно.

$$vx := \begin{pmatrix} 2.046 \\ 1.929 \\ 9.568 \\ 7.388 \\ 6.978 \\ 9.213 \\ 8.443 \\ 1.054 \\ 9.038 \\ 8.927 \end{pmatrix} \qquad vy := \begin{pmatrix} -0.4004 \\ -0.401 \\ -11.85 \\ -6.206 \\ -5.005 \\ -10.81 \\ -8.703 \\ -0.5788 \\ -10.31 \\ -9.998 \end{pmatrix}$$

Для сортировке их по координате x, объединим их с помощью функции augment. Затем функцией csort отсортируем матрицу по первому(нулевому) столбцу, и потом опять их разъединим.

$V := \text{augment}(vx, vy)$

$W := \text{csort}(V, 0)$

$vx := W^{\langle 0 \rangle} \qquad vy := W^{\langle 1 \rangle}$

vx =

	0
0	1.054
1	1.929
2	2.046
3	6.978
4	7.388
5	8.443
6	8.927
7	9.038
8	9.213
9	9.568

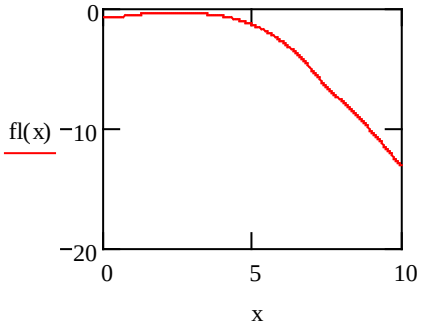
vy =

	0
0	-0.579
1	-0.401
2	-0.4
3	-5.005
4	-6.206
5	-8.703
6	-9.998
7	-10.31
8	-10.81
9	-11.85

Построенные точки линейным сплайном с помощью функции lspine(vx,vy), аргументами которой являются два вектора координат точек по x и по y. Определим функцию fl(x) с помощью функции интерполирования interp.

$$vl := lspline(vx, vy)$$

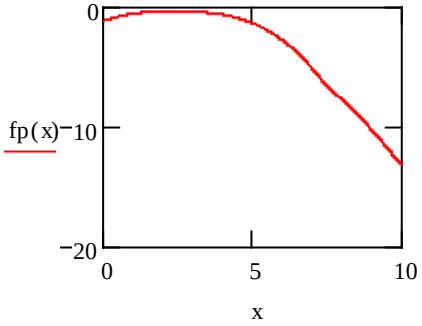
$$fl(x) := interp(vl, vx, vy, x)$$



Аналогично интерполируем параболическим и кубическим сплайном функциями pspline(vx,vy) и csplie(vx,vy).

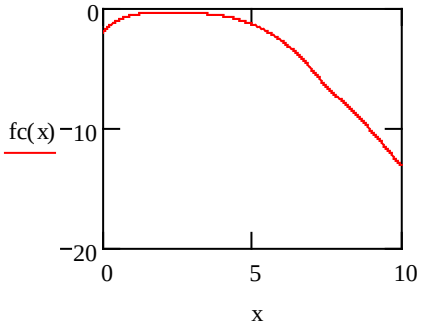
$vp := pspline(vx, vy)$

$fp(x) := interp(vp, vx, vy, x)$

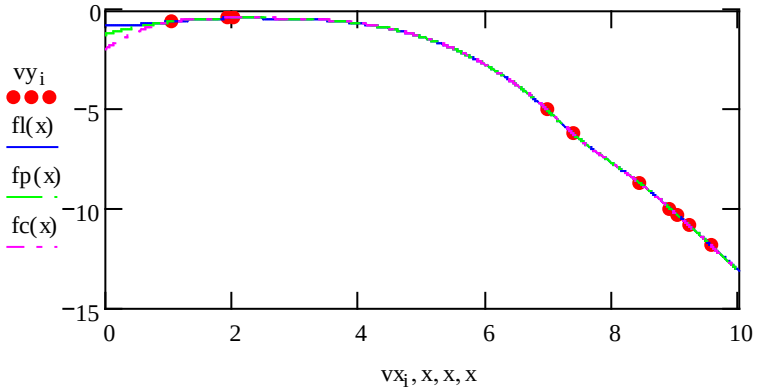


$vc := cspline(vx, vy)$

$fc(x) := interp(vc, vx, vy, x)$



Теперь построим все три сплайна на одном графике, а также отметим заданные точки.



Из графика видно, что на заданном интервале сплайны близки своими значениями, однако за пределами видно существенное расхождение.

Список литературы

1. Марченко А.И., Марченко Л.А.; Программирование в среде Turbo Pascal 7.0 – М.: Бином Универсал, 1997. – 496 с., ил.
2. Грызлов В.И., Грызлова Т.П., Турбо Паскаль 7.0 – М.: ДМК, 1998. - 400 с., ил.
3. Климова. Л.М. Pascal 7.0. Практическое программирование. Решение типовых задач. – М.: КУДИЦ-ОБРАЗ, 2000. - 528 с.
4. Дьяконов В., MathCad 2000: учебный курс – СПб.: Питер, 2001. - 592 с., ил.
5. Дьяконов В., MathCad 2001: специальный справочник – СПб.: Питер, 2002. – 832 с., ил.