БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Г. А. Расолько Е. В. Кремень Ю. А. Кремень

ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ

В трех частях

Часть 2
СПЕЦИАЛЬНЫЕ СРЕДСТВА PASCAL

Рекомендовано
Учебно-методическим объединением
по естественно-научному образованию
в качестве учебно-методического пособия
для студентов учреждений высшего образования,
обучающихся по специальности «математика»

Учебное электронное издание

Минск, БГУ, 2025

ISBN 978-985-881-747-3 (ч. 2)
ISBN 978-985-881-745-9

УДК 004.432Pascal(075.8)(076.5)
ББК 32.973.26-018.1я73-5

Р е ц е н з е н т ы:
кафедра инженерной психологии и эргономики
Белорусского государственного университета
информатики и радиоэлектроники
(заведующий кафедрой доктор психологических наук,
член-корреспондент Международной академии
психологических наук, профессор *Т. В. Казак*);
доктор физико-математических наук, доцент *А. С. Кравчук*

Расолько, Г. А. Практикум по программированию [Электронный ресурс] : учеб.-метод. пособие. В 3 ч. Ч. 2. Специальные средства Pascal / Г. А. Расолько, Е. В. Кремень, Ю. А. Кремень. – Минск : БГУ, 2025. – 1 электрон. опт. диск (CD-ROM). – ISBN 978-985-881-747-3.

Представлен учебный материал для проведения занятий по дисциплине «Практикум по программированию». Приведена краткая теория, рассмотрены задачи по темам: модули пользователя, файлы, специальные средства алгоритмического языка, видеодоступ.

Предназначено для студентов учреждений высшего образования, обучающихся по специальности «математика».

Минимальные системные требования:
PC, Pentium 4 или выше;
RAM 1 Гб; Windows XP/7/10; Adobe Acrobat.

Оригинал-макет подготовлен в программе MicroSoft Word.

Ответственный за выпуск *Т. М. Турчиняк*. Дизайн обложки *В. П. Явуз*.
Технический редактор *В. П. Явуз*. Компьютерная верстка *В. П. Явуз*.
Корректор *Е. О. Алёшина*

Подписано к использованию 25.03.2025. Объем 1,37 МБ.

Белорусский государственный университет.
Управление редакционно-издательской работы.
Пр. Независимости, 4, 220030, Минск.
e-mail: urir@bsu.by
http://elib.bsu.by

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящий практикум направлен на формирование у студентов навыков решения различных типов задач с использованием современных информационных технологий. Основная цель издания – развитие алгоритмического мышления, изучение современных методов программирования и приобретение навыков работы с современными вычислительными средствами.

В процессе занятий студенты должны:

- развивать математическое, логико-алгоритмическое и программистское мышление;
- применять практические знания и умения, современные методы и системы программирования;
- осваивать приемы и основы методологии структурного и модульного программирования;
- формировать творческий подход к конструированию алгоритмов, что способствует развитию аналитических способностей.

Основное внимание уделяется не только кодированию программ, но и их проектированию, где особый акцент делается на современных технологиях: проектирование сверху вниз, модульное программирование с использованием подпрограмм и модулей, анализ эффективности и оптимизация программ, широкое использование рекурсии. Все это направлено на привитие определенного стиля программирования и дальнейшее развитие навыков посредством обучения объектно ориентированному программированию.

В качестве основного языка программирования выбран объектно ориентированный Pascal, который позволяет освоить как классические приемы, так и современные технологии программирования.

Во второй части учебно-методического пособия рассмотрены такие темы, как модули пользователя, работа с файлами, специальные средства алгоритмического языка. Эта часть даст студентам более глубокое понимание программирования и управления данными, которые необходимы для создания эффективных и надежных программ.

В результате изучения второй части практикума обучающие должны знать:

- основы создания и использования модулей, а также работу с комплексными и рациональными числами;
- файловые системы, операции над файлами и методы работы с текстовыми и типизированными файлами;

- создание и уничтожение динамических переменных и массивов, программирование одномерных и двумерных массивов;
- возможности видеодоступа и графического программирования.

Указанные навыки и знания позволят студентам разрабатывать сложные программные проекты, эффективно управлять данными и использовать современные технологии программирования.

Тема 7
МОДУЛИ ПОЛЬЗОВАТЕЛЯ
(6 часов)

Модуль, кроме заголовка `Unit Unitname`, имеет три части – *интерфейсную*, *реализации* и *инициализации*.

В *интерфейсной части* модуля собраны описания объектов, которые доступны из других программ. Это видимые вне модуля объекты.

В *части реализации* содержатся рабочие объекты – скрытые, или невидимые.

В *части инициализации* описываются действия, которые будут выполнены при подключении модуля.

Общая структура модуля

```
unit unitname;
interface
```

Описание видимых объектов	Описание типов, констант, переменных, заголовков процедур, функций.

```
implementation
```

Описание скрытых объектов	Реализация процедур и функций, которые описаны в интерфейсной части, а также вспомогательных алгоритмов, типов, констант и переменных.

```
begin
```

Операторы инициализации объектов модуля	Установка начальных значений переменных модуля перед его использованием.

```
end.
```

При описании модуля используются служебные слова: `unit, interface, implementation`.

Подключение модуля происходит в начале программы оператором `uses`.

Спецификация `uses` должна идти непосредственно после заголовка программы, так как она содержит определения данных и подпрограмм.

Если некоторый модуль подключается в модуль в раздел `interface` или `implementation`, тогда `uses` следует размещать сразу после служебных слов `interface` или `implementation`.

7.1. Подпрограммы в модулях

Заголовок подпрограммы содержит всю информацию, необходимую для вызова: ее имя, число и типы параметров, а для функций – также тип результата. Заголовок подпрограммы является ее интерфейсом (видимый), а тело – реализацией (невидимо), так как тело подпрограммы раскрывает алгоритм.

Значит, в интерфейсной части модуля представлены только заголовки процедур и функций, которые видимы (доступны) для других программ. Это позволяет механизму модулей спрятать детали реализации тех или иных программных подсистем.

Модуль компилируется таким же образом, как и программа, но поскольку модуль не является непосредственно выполняемой единицей, то в результате его компиляции образуется дисковый файл с расширением «.TPU» (Turbo Pascal Unit). Имя TPU-файла должно совпадать с именем файла исходного кода модуля.

Модуль – это автономно компилируемая программная единица.

Задачи

В следующем модуле объявлены средства для работы с комплексными числами: описание типа Complex; инициализация комплексного числа по двум вещественным числам; операции сложения, умножения, деления двух комплексных чисел; печать комплексного числа.

Необходимо запрограммировать заявленные подпрограммы и расширить данный модуль для:

- решения приведенного квадратного уравнения на множестве комплексных чисел;
- решения приведенного биквадратного уравнения на множестве комплексных чисел;
- подсчета тригонометрических функций от комплексного аргумента.

```
unit CmplVals;
interface
type
      Complex = record
                  Re, Im : Real
                end;
    {заголовки процедур, которые реализуют
      операции над комплексными числами}
```

```
procedure  InitC  (R, I : Real;      var C : Complex);
procedure  AddC   (C1, C2 : Complex; var R : Complex);
procedure  MultC  (C1, C2 : Complex; var R : Complex);
procedure  DivC   (C1, C2 : Complex; var R : Complex);
procedure  WriteC (C : Complex);
function   ModC   (C : Complex) : Real;

implementation
                {полное описание процедур}
procedure  InitC;
begin
 with C do
     begin
      Re := R;      Im := I
     end
end;
{здесь запрограммировать реализации заявленных подпрограмм}
end.
```

7.2. Использование модуля, содержащего средства для работы с комплексными числами

Подключение модуля происходит в начале программы операторам `uses`.

Спецификация `uses` должна находиться сразу после заголовка программы, так как она содержит определения данных и подпрограмм.

Если некоторый модуль подключается в модуль в раздел `interface` или `implementation`, тогда `uses` должен идти сразу после служебных слов `interface` или `implementation`.

При подключении модуля нужно выполнить следующие действия:

- первый вариант: открыть файл, содержащий текст программы, которая использует модуль `CmplVals`;
- в меню `Options` выбрать команду `Directories` и в диалоговом окне в поле `UnitDirectories` указать путь к модулю `CmplVals`;
- второй вариант: разместить модуль в текущем каталоге;
- далее через команду `Run` можно запустить выполнение главной программы.

Пример программы, которая использует модуль CmpVals:

```
program UsingComplex;
uses CmplVals;
  {
   чтобы получить доступ к интерфейсным объектам модуля,
   необходимо указать в программе имя нужного TPU-файла
   }
var c1, c2, c3 : Complex;
   {
     заданный в модуле тип Complex,
     доступен здесь так же,
     как будто он определен в программе
    }
begin
 InitC(1,  2, c1);
 InitC(3,  4, c2);
 MultC(c1, c2, c3);
 WriteC(c3);
 DivC(c1,  c2, c3);
 WriteC(c3)
end.
```

Задачи

1. В текстовом файле имеются вещественные числа, количеством, которое кратно трем. Пока не закончится информация из текстового файла, нужно инициализировать коэффициенты полного квадратного уравнения и решить его на множестве комплексных чисел. Результаты вывести на экран.

2. В текстовом файле имеются вещественные числа, количеством, которое кратно трем. Пока не закончится информация из текстового файла, нужно инициализировать коэффициенты полного биквадратного уравнения и решить его на множестве комплексных чисел. Результаты вывести на экран.

7.3. Создание и использование модуля по работе с рациональными числами

Представим рациональное число в виде пары чисел: целое число – числитель дроби, натуральное число – знаменатель дроби.

По аналогии с предыдущим примером нужно запрограммировать модуль `Rational`, содержащий ресурсы по работе с рациональными числами:

- инициализация рационального числа по двум вещественным числам;
- приведение дроби `a/b` к несокращаемому виду `p/q`;
- сложение двух рациональных чисел;
- вычитание двух рациональных чисел;
- умножение двух рациональных чисел;
- деление двух рациональных чисел;
- решение приведенного квадратного уравнения на множестве рациональных чисел;
- решение приведенного биквадратного уравнения на множестве рациональных чисел;
- печать рационального числа.

Задачи

1. Пусть имеется модуль по работе с рациональными числами, в котором описана процедура `Сокр(a, b, p, q)` с целыми параметрами (`b<>0`), приводящая дробь `a/b` к несокращаемому виду `p/q`.

Используйте процедуру `Сокр` для приведения значения выражения `1+1/2+1/3+…+1/20` к несократимому виду `c/d`.

2. Длинные числа объявляются как запись, состоящая из двух полей: текущая длина числа, массив цифр числа в некоторой системе счисления (например, десятичной).

По аналогии с предыдущим примером нужно запрограммировать модуль **`long_number`**, содержащий ресурсы по работе с длинными числами:

- инициализация числа;
- печать числа;
- умножение числа на целое;
- сложение двух длинных чисел;
- вычитание двух длинных чисел;
- получение значения n! для больших n.

Тема 8
ФАЙЛЫ
(8 часов)

8.1. Файловая система

Для описания файловой переменной используются слова `text`, `file` или словосочетание `file of`. Например, описание переменной

```
var   f : file of Integer;
```

понимается как определение под именем `f` последовательности, расположенной на внешнем запоминающем устройстве и состоящей из целых чисел, которые называются *компонентами* или *элементами файла*.

При работе с файлом с каждой переменной файлового типа согласуется текущий указатель файла.

Текущий указатель обозначает (указывает на) конкретный элемент файла, доступный для обработки.

Все действия с файлом (чтение, запись) выполняются поэлементно, причем в этих действиях участвует тот элемент файла, на который указывает текущий указатель.

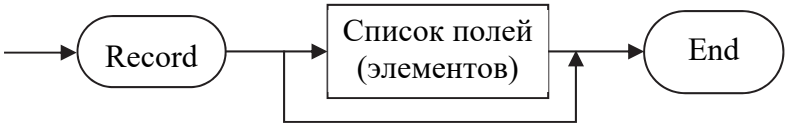
Записи

При работе с файлами с типом часто используют структурированный тип данных – запись.

Запись – это объединение данных разных типов посредством задания именованных полей. В отличие от массивов и множеств, поля записей могут иметь различные типы.

Для определения записи применяют служебные слова `record` и `end`.

Запись:



Пример 1.

```
Type
     zap = record
          x, y : real
         end;
var a : zap;
```

Пример 2.

```
Type
   Person=record
            f_i_o     : string;
                  {фамилия, имя, отчество ≤ 255 символов}
            ves, rost : real;
            telephone : longint; {номер телефона}
            ozenki    : array[1..4] of byte;
     end;
```

Тип `Person` сейчас задает анкету из строки (`f_i_o`), двух чисел с дробной частью (`ves, rost`), одного длинного целого числа (`telephone`) и массива на четыре байта (`ozenki`).

Если задано объявление

```
var  kto_to : person;
```

то под именем `kto_to` понимается данное с конкретными значениями.

Доступ к полям осуществляется по имени при помощи селектора записи согласно следующему формату:

`имя_переменной.имя_поля`

`kto_to.f_i_o` – это значение строки с фамилией.

`Kto_to.telephone` – это значение длинного целого числа с заданным номером телефона.

Над записями как единым целым допускаются следующие действия: присвоить значение одной записи другой, проверить их на равенство или неравенство. Такие записи должны иметь одинаковый тип. Ввод-вывод записи происходит только по их полям.

Доступ к элементам (полям) записей происходит при помощи конструкции, которая называется *селектор записи*, имеющей вид

`R.f`,

где `R` – переменная типа запись, `f` – идентификатор поля.

Операции над файлами

В отличие от переменных других типов, язык Pascal не содержит встроенные операции над собственно файловыми переменными.

Операции с файлами реализованы в виде стандартных подпрограмм, собранных в модуле `System`, который присоединяется автоматически.

Операции с файловыми переменными можно разбить на четыре основные группы:

- установочные и завершающие операции;
- специальные операции;
- собственно ввод-вывод;
- перемещение по файлу.

Установочные и завершающие операции

В группу установочных и завершающих операций входят следующие процедуры:

1) `Assign(f, Name)` – связывает файл `Name` с файловой переменной `f`;
2) `Reset(f)` – открывает доступ к существующему файлу `f`;
3) `Rewrite(f)` – создает и открывает новый файл `f`;
4) `Flush(f)` – переписывает данные из буфера в файл;
5) `Close(f)` – закрывает файл.

Здесь `f` – переменная файлового типа.

Рассмотрим процедуры подробнее.

1. Процедура `Assign(f, Name)` предназначена для установления связи между конкретным файлом (набором данных) на внешнем носителе и переменной файлового типа. Эта процедура всегда предшествует другим процедурам работы с файлами.

Например:

```
Assign(f, 'd:\mydir\myfile.dat');
```

После выполнения этого оператора предполагается, что файловая переменная `f` будет связана с дисковым файлом `myfile.dat`, расположенным в каталоге `mydir` корневого каталога диска `d`. (Проверка на корректность не делается.)

2–3. Процедуры `Reset(f)` и `Rewrite(f)` предназначены для открытия файла, где `f` – файловая переменная, предварительно назначенная оператором `Assign`.

Под открытием понимается: поиск файла на внешнем носителе; создание специальных системных буферов для обмена с ним; установка текущего указателя файла на его начало.

Все элементы файла с типом условно нумеруются: 0, 1, 2, ...

Процедура `Reset(f)` предполагает, что файл, который открывается, уже существует; в противном случае возникает ошибка.

Процедура `Rewrite(f)` допускает, что файл, который открывается, может не существовать, тогда она создает заданный файл. Если же файл существует, тогда `Rewrite` очищает его.

В обоих случаях, если файл `f` был открыт, он предварительно закрывается. Однако связь с существующим файлом не нарушается (она настроена через `Assign`).

Как бы файл с типом ни был открыт (`Reset` или `Rewrite`), в модуле `System` описана переменная `FileMode`, значение которой становится равной 2 (значение 0 означает только чтение, значение 1 – только запись). Оно показывает, что из файла можно и читать, и в файл можно записывать данные.

4. Процедура `Flush(f)` завершает обмен с файлом без его закрытия.

Обмены с файлами всегда реализуются через некоторый буфер в оперативной памяти, поэтому в процессе записи в файл последние элементы, которые записываются, могут еще остаться в буфере. Процедура `Flush` вызывает принудительное сбрасывание этих элементов в файл.

5. Процедура `Close(f)` завершает действие с файлом `f`:

- переписывает информацию из буфера в файл (если файл для вывода);
- устраняет внутренние буферы, созданные при открытии этого файла;
- устраняет связь файла с файловой переменной `f` (освобождает обработчик файла для других работ).

После этого файловую переменную можно связать с помощью `Assign` с другим файлом или устройством.

Операции ввода-вывода для файлов с типом

В эту группу входят две операции (процедуры) – `Read` и `Write`, которые реализуют чтение информации из файла и запись информации в файл соответственно.

Обмен данными происходит через буфер ввода-вывода, размер которого устанавливается автоматически, исходя из размера элементов файла.

В отличие от многих других процедур, `Read` и `Write` могут вызываться с разным числом параметров. Формат операторов:

`Read(f, v1, ..., vn)` и `Write(f, v1, ..., vn)`.

Список `v1, ..., vn` может состоять из нескольких или одной переменной базового типа файла. Многоточие в формате показывает, что здесь будут перечислены переменные через запятую.

Процедура `Read` предназначена для чтения значений из файла. Первый параметр – имя файловой переменной. Далее должны идти переменные, в которые будут помещены значения из файла. Тип этих переменных должен совпадать с базовым типом файла.

Выполнение процедуры `Read` происходит следующим образом. Начиная с текущей позиции указателя файла, последовательно читаются значения, находящиеся в файле, и присваиваются очередной переменной из списка ввода. После каждого действия по чтению данных из файла указатель перемещается на следующий элемент.

Если в процессе выполнения процедуры `Read` текущий указатель файла будет установлен на позицию последнего элемента файла и этот элемент будет прочитан, тогда возникнет ситуация *«конец файла»*.

Возникновение ситуации «конец файла» можно проверить при помощи встроенной логической функции `EoF(F)` с параметром `F` – файловой переменной.

Функция `EoF(F)` возвращает логическое значение `true`, если достигнут конец файла, и `false` – в противном случае. При обращении к файлу для чтения, у которого `EoF` имеет значение `true`, система дает фатальную ошибку 100 (считывание после конца файла).

При записи в файл, истинность функции `EoF(f)` означает, что очередная операция записи поместит информацию в конец файла.

Процедура `write` позволяет записывать информацию в файл. Первым параметром этой процедуры должна быть файловая переменная, которая была открыта процедурой `reset` или `rewrite`. Далее должен идти список переменных. При записи в файл записывается внутреннее представление очередного элемента. В файле элементы занимают один и тот же объем памяти в соответствии со своим типом.

Задачи

1. В текстовом файле имеются целые числа, количеством, которое кратно шести. Пока не закончится информация из текстового файла, нужно инициализировать коэффициенты полного квадратного уравнения с рациональными коэффициентами и решить его на множестве рациональных чисел. Результаты вывести на экран.

2. В текстовом файле имеются целые числа, количеством, которое кратно шести. Пока не закончится информация из текстового файла, нужно инициализировать коэффициенты полного биквадратного уравнения с рациональными коэффициентами и решить его на множестве рациональных чисел. Результаты вывести на экран.

3. Объявите запись, которая содержит информацию о студентах какого-то учебного заведения: фамилия, имя, отчество, курс, группа, полученные зачеты и оценки. Проинициализируйте ее поля.

4. Объявите запись, которая содержит информацию о сотрудниках какого-то заведения: фамилия, имя, отчество, дату рождения, дату поступления на работу, оклад, причем, поле даты должно иметь тип запись. Проинициализируйте ее поля.

5. Пусть имеется константа *n*_max.

Объявите запись, первое поле которой есть реальный размер одномерного массива, а другое – массив, в котором будут храниться вещественные коэффициенты некого многочлена степени не выше *n*_max. Проинициализируйте ее поля.

6. Пусть имеется константа *n*_max.

Объявите массив записей размера не выше *n*_max, который содержит информацию о сотрудниках какого-то заведения: фамилия, имя, отчество, дату рождения, дату поступления на работу, оклад, причем, поле даты должно иметь тип запись. Проинициализируйте их поля.

7. Объясните работу следующих программ:

а)
```
type    arr4 = array[1..4] of byte;
const
   x : record x, y : integer end = (x:315; y: -26);
begin
  write(arr4(x)[1],‘ ‘,arr4(x)[2],‘ ‘,
        arr4(x)[3],‘ ‘,arr4(x)[4]);
end.
```

б)
```
Var
    x: record
         x, y : integer
       end;
type
    arr4 = array[1..4] of byte;
begin
   arr4(x)[1] := 3;
   arr4(x)[2] := 2;
   arr4(x)[3] := 9;
   arr4(x)[4] := 7;
   write(x.x, x.y);
end.
```

8. Объясните работу следующих программ. Одинаковую ли информацию они напечатают?

а)
```
type
 point    = record
              x, y : real;
            end;
      complex = record
                  re, im : real;
                end;
 var
 p    : point;
      z, w : complex;
      re   : real;
 begin
  z.re := 0;    z.im := 1;       w   := z;     re  := 2;
  z.re := 1;    z.im := -w.im;   p.x := re;    p.y := 2;
  writeln(z.re, z.im, w.re, w.im, p.x, p.y);
  writeln(re);
  end.
```

б)
```
type
 point   = record
             x, y : real;
           end;
      complex = record
                  re, im : real;
                end;
  var
 p     : point;
 z, w  : complex;
 re, y : real;
 begin
  with z do im := 1;
  re := 0;   w := z;
  re := 2;
  with z do re := 1;
  im := -w.im;
  with p do y  := 2;   x  := re;
  writeln(z.re, z.im, w.re, w.im, p.x, p.y);
  writeln(re);
 end.
```

в)
```
type
 point   = record
             x, y : real;
           end;
      complex = record
                  re, im : real;
                end;
 var
 p    : point;
 z, w : complex;
 re   : real;
 begin
  with z do begin re := 0; im := 1; end;
  w  := z;
  re := 2;
  with z do begin re := 1; im := -w.im; end;
  with p do begin x := re; y := 2; end;
  writeln(z.re, z.im, w.re, w.im, p.x, p.y);
  writeln(re);
 end.
```

9. Исправьте ошибки в следующем фрагменте программы и выполните ее:

```
type
    point1 = array[ (x , y) ] of real;
    point2 = record
               x, y : real;
             end;
 var
     p1   : point1;
     p2   : point2;
     d    : real;
begin
   p1[x]:= 2;
   p1[y]:=4;
   p2.x:=4;
   p2.y:=6;
   d := sqrt( sqr(p1[x]-p2.x)+sqr(p1[y]-p2.y));
   writeln(d:7:0);
end.
```

8.2. Последовательный и прямой доступ к файлу с типом

Данная группа операций позволяет произвольно изменять последовательность выполнения операций чтения и записи. Сюда входят следующие процедуры:

- `Seek(f, N)` – устанавливает указатель на элемент с номером `N`, где `N` имеет тип `Longint`;
- `Truncate(f)` – уничтожает все элементы файла, начиная с места текущего указателя.

Можно пользоваться и двумя дополнительными функциями:

- `FileSize(f)` – возвращает размер файла (количество элементов);
- `FilePos(f)` – возвращает номер элемента, на который установлен текущий указатель.

Процедура `Seek` позволяет явно изменить значение текущего указателя, установив его на элемент файла с заданным номером (нумерация начинается с нуля). Это фактически прямой доступ к элементу с заданным номером. После выполнения процедуры `Seek` дальнейшие операции чтения или записи будут проводиться, начиная с установленной позиции указателя.

Примеры

`Seek(f, 0)` → установка указателя на начало файла.

`Seek(f, FilePos(f)+1)` → пропуск одного элемента.

`Seek(f, FileSize(F))` → установка текущего указателя непосредственно за последним элементом файла.

Рассмотрим типовые алгоритмы для работы с файлами.

1. Общая структура фрагмента программы, предназначенной для чтения из файла с целью последующей обработки данных:

```
Assign(f, '…');
Reset(f);
while not eof(f) do
   begin
    Read(f, a);
    …
   end;
Close(f);
```

2. Общая структура фрагмента программы, предназначенной для записи в файл:

```
Assign(f, '…');
Rewrite(f);
 {Организовать цикл на количество записываемых элементов}
    begin
       {Получение данного для записи}
       Write(f, a);
    end;
Close(f);
```

3. Общая структура фрагмента программы, предназначенной для добавления в файл:

```
Reset(f);
Seek(f, FileSize(f));
 {Организовать цикл на количество записываемых элементов}
     begin
       {Получение данного для записи}
       Write(f, a);
     end;
Close(f);
```

Специальные операции

Подробнее о таких операциях можно прочесть в фирменных руководствах по языку. Рассмотрим только две операции:

- `Erase(f)` – уничтожение файла на диске, который был связан с файловой переменной `f`. Если файл не существует, то возникает ошибка ввода-вывода;
- `Rename(f, name)` – переименование файла.

Эти процедуры работают с неоткрытыми файлами: требуется выполнить только процедуру `Assign`, но не выполнять `Reset` или `Rewrite`.

`Rename(f, newname)` переименовывает неоткрытый через `Reset` или `Rewrite` файл `f`. Новое имя задается строкой `newname`. При этом нельзя изменять имя диска и путь к файлу, изменяется только собственное имя физического файла.

Задачи

1. Создайте файл *f*1 из целых случайных чисел, принадлежащих отрезку [*a*, *b*], а затем перепишите его элементы в файлы *f*2 и *f*3, помещая в *f*2 только положительные, а в *f*3 − только отрицательные числа.

2. Необходимо вставить новый элемент в целочисленный файл *f*, отсортированный по уменьшению значений элементов, таким образом, чтобы упорядоченность не нарушилась.

3. Объедините два отсортированных по уменьшению значений элементов файла *f*1 и *f*2 в файл *f*3, элементы которого упорядочены по уменьшению.

4. Разработайте программу, которая с помощью отдельных подпрограмм выполняет следующие действия:

- формирует файл сведений о студентах (курс, группа, фамилия, имя, отчество, номер зачетки, отметки за последнюю сессию);
- распределяет информацию по новым файлам таким образом, чтобы каждый из них касался только одного курса;
- позволяет просмотреть на экране произвольный из созданных новых файлов, касающийся соответствующего курса;
- печатает сведения о студентах, которые принадлежат выбранному курсу и группе;
- проводит корректировку особых сведений (курс, группа, оценки);
- подсчитывает средний балл за последнюю сессию для выбранных группы и курса;
- записывает в файл соответствующие сведения о новых студентах.

5. Разработайте программу создания и корректировки файла, который содержит сведения о книгах, находящихся в читальном зале библиотеки. Каждый элемент этого файла содержит следующую информацию: фамилия и инициалы автора, название книги, издательство, год издания, количество страниц.

6. Задано целое число *n* и вещественное число *x*. Получите квадратную матрицу порядка *n*×*n* (середина заполняется нулями) и запишите, двигаясь по строкам, ее элементы в файл *g*:

$$\begin{bmatrix} 1 & x & \cdots & x^{n-2} & x^{n-1} \\ x & 0 & \cdots & 0 & x^{n-2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x^{n-2} & 0 & \cdots & 0 & x \\ x^{n-1} & x^{n-2} & \cdots & x & 1 \end{bmatrix}$$

7. Известно, что имеют место следующие предельные равенства:

а) $\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$;

б) $\frac{1}{2} - \frac{\pi}{8} = \sum_{n=1}^{\infty} \frac{1}{(4n-1)(4n+1)}$;

в) $\frac{\pi^2}{8} = \sum_{n=1}^{\infty} \frac{1}{(2n-1)^2}$;

г) $\frac{\pi^2}{6} = \sum_{n=1}^{\infty} \frac{1}{n^2}$;

д) $\frac{\pi^2}{12} = -\sum_{n=1}^{\infty} \frac{(-1)^n}{n^2}$;

е) $\frac{\pi^4}{96} = \sum_{n=1}^{\infty} \frac{1}{(2n-1)^4}$.

Запишите в файл *h* пары чисел (i, n_i), где i – номер варианта от 1 до 6; n_i – номер того слагаемого, на котором остановился подсчет бесконечной суммы для получения числа π с точностью до некоторого заданного $\varepsilon(\varepsilon > 0)$.

8.3. Текстовые файлы

При записи в текстовый файл данных (чисел, строк, логических значений и т. д.) они превращаются в символьный вид.

При чтении из текстового файла данные автоматически преобразуются из текстового представления во внутреннее.

Текстовые файлы можно обрабатывать только последовательно.

Ввод и вывод нельзя выполнять одновременно для одного и того же текстового файла.

Представителем текстового файла в программе является переменная файлового типа, которая должна быть описана так:

```
var
    Texinf : text;
```

Чтобы наглядно показать различие в представлении информации для двух видов файлов – текстовых и файлов с типом, обратимся к записям с типом `String`.

Представление строк в текстовом файле следующее:

Первая строка файла:	`МАМА#13#10`
Вторая строка файла:	`ПОШЛА#13#10`
Третья строка файла:	`В МАГАЗИН#13#10`

Общая длина фразы для текстового файла будет 25 байт: (6 байт – первая строка, 7 – вторая, 11 – третья, 1 байт – символ конца файла).

Представление строк в типизированных файлах с типом `String`:

Первая запись:	`#4МАМА#0#0#0#0#0#0#0`
Вторая запись:	`#5ПОШЛА#0#0#0#0#0#0`
Третья запись:	`#9В МАГАЗИН#0#0`

Здесь на месте символа #0 может стоять любой другой.

Общая длина фразы для типизированных файлов составит 36 байт.

Для выполнения работ над строками в режиме записи или чтения предпочтение отдается текстовым файлам.

Чтобы создать текстовый файл, лучше всего пользоваться любым текстовым редактором.

Если программа требует ввода трех и более чисел, то рекомендуется создать вспомогательные текстовые файлы с тестовыми наборами данных, чтобы не тратить время на ввод данных с клавиатуры при каждом запуске программы. Но возможны и другие способы.

Процедуры для работы с текстовыми файлами

Процедуры `Assign`, `Close`, `Flush`, функция `EoF` одинаковы для типизированных и текстовых файлов.

`Append(F)` – открывает существующий файл `F` для добавления. Если файл отсутствует на диске, то возникает ошибка ввода-вывода.

`Rewrite(F)` – создает новый текстовый файл, к которому можно только добавлять строки. Если файл с таким именем уже существует на диске, то он уничтожается, и создается новый текстовый файл.

`Reset(F)` – применяется только к существующему файлу, после чего из этого файла можно читать только последовательно.

Если новый текстовый файл закрывается, к нему автоматически добавляется маркер конца файла.

Второе обращение к `Reset(F)` установит текущий указатель снова в начало файла `F`.

Процедуры чтения `(Read, Readln)` из текстового файла и записи `(Write, Writeln)` в файл рассмотрены ранее.

Функции для работы с текстовыми файлами

`Eof[(F)] : Boolean` – возвращает `true`, если следующим за последним прочитанным символом является маркер конца файла `F`.

Если файловая переменная `F` отсутствует, тогда подразумевается стандартный файл ввода.

`Eoln[(F)] : Boolean` – возвращает `true`, если следующим за последним прочитанным символом является маркер конца строки (или `Eof`).

При отсутствии аргумента подразумевается стандартный файл ввода.

Если `Eof(F)` – `true`, то и `Eoln(F)` – `true`.

`SeekEoln[(F)] : Boolean` – проводит поиск конца текущей строки.

Эта функция пропускает все символы-разделители (пробелы и символы табуляции) в строке и устанавливает текущий указатель файла `F` или на конец строки (тогда функция возвращает `true`), или на первый значащий символ (тогда функция возвращает `false`).

`SeekEof[(F)] : Boolean` – возвращает `true`, когда указатель файла `F` находится на маркере конца файла.

Функция пропускает все символы-разделители, а также символы концов строк (т. е. переходит со строки на строку в поисках конца файла или первого значащего символа).

Задачи

1. Перепишите в текстовый файл *t*1 содержимое текстового файла *t*2, но без строк, в которых находятся цифры.

2. Создайте на основе текстового файла *t*1 текстовый файл *t*2, разбивая *t*1 на строки так, чтобы каждая строка или оканчивалась точкой с запятой, или содержала 30 произвольных символов, кроме точки с запятой. В качестве *t*1 используйте файл с текстом разработанной программы.

3. Перекодируйте текстовый файл, заменяя в нем все прописные буквы на строчные.

4. Выведите на экран текстовый файл таким образом, чтобы выполнялось выравнивание по правой границе путем вставки между отдельными словами необходимого количества пробелов. В качестве исходного файла используйте файл с текстом разработанной программы.

5. Заданы два текстовых файла – *t*1 и *t*2. В файле *t*1 содержится произвольный текст, где слова в тексте разделены пробелами и знаками препинания. Файл *t*2 содержит не более 40 слов, которые разделены запятыми. Эти слова образуют пары: каждое первое слово считается заменяемым, каждое второе – заменяющим. Найдите в файле *t*1 все заменяемые слова и замените их соответствующими заменяющими. Результат разместите в файле *g*.

6. Найдите и удалите из файла самое длинное и самое короткое предложение. Концом предложения является один из разделителей ”.”, ”…”, ”!”, ”:”, ”;”, ”?”, ”,”.

7. Разбейте программу на части, отделяя их строкой {*…*}. Каждая часть – это процедура, функция или основной блок.

8. Уплотните программу, помещая в одну строку несколько операторов так, чтобы длина строки не превышала 80 символов и завершалась символом ”;”.

9. В зависимости от номера запроса отредактируйте файл:
0 – поменять местами строку *k* со строкой *m*;
1 – переставить строку *k* после строки *m*;
2 – переставить строку *k* перед строкой *m*.
Числа *m* и *n* задаются во время выполнения программы и могут образовывать некорректную ситуацию.

8.4. Файлы без типа

Файл без типа состоит из элементов одинакового размера, структура которых не имеет значения или неизвестна. Файлы без типа применяются для высокоскоростных программ обмена между оперативной и внешней памятью. Нетипизированный файл объявляется так:

```
var    f : file;
```

Для работы с такими файлами используют следующие процедуры и функции:

```
Assign      Blockread      Eof,
Rewrite     Blockwrite     FileSize
Reset       Seek           FilePos
```

Обмен с типизированным файлом осуществляется следующим образом: вводится-выводится значение данного в форме его внутреннего представления. Вспомним, что при обмене через текстовый файл идет преобразование последовательности символов во внутреннее представление данного при вводе и наоборот – при выводе.

Нетипизированный файл в языке Pascal рассматривается как совокупность байт-блоков.

Любой файл, который был подготовлен как файл `text` или файл с типом, можно открыть и начать работу с ним как с нетипизированным набором данных.

Для таких файлов не требуется преобразовать данные или искать управляющие последовательности. Достаточно считать содержимое файла в определенную память.

Для нетипизированных файлов самым важным параметром является длина блока в байтах, и этот параметр либо присутствует в процедурах открытия файла, чтения, записи, либо берется по умолчанию. Однако суммарный объем разового обмена не должен превышать 64 кб.

Рассмотрим подробнее перечисленные выше процедуры.

`Rewrite(F, Size)` или же `Rewrite(F)` – создает новый файл `F`. Если файл уже существует, то он уничтожается, и создается новый. Второй параметр `Size` определяет размер блока в файле. Если `Size` отсутствует, то размер блока по умолчанию равен 128 байт.

`Reset(F, Size)` или `Reset(F)` – открывает существующий файл `F`. В файл можно записывать информацию и читать из него. Второй параметр аналогичен предыдущей процедуре.

`Blockread(f, Buf, count, Result)` или `Blockread(f, Buf, count)` – читает из файла `F` `count`-записей (блоков) в буфер ввода `Buf`. Если есть четвертый параметр `Result`, то он получает значение реального числа прочитанных записей (блоков). Если читается последняя порция `count`-записей из файла, то она может быть либо пустой, либо неполной.

Если четвертый параметр отсутствует, и количество требуемых (`count`) и реально прочитанных записей (блоков) не совпадает, то возникает ошибка ввода-вывода. Если четвертый параметр присутствует, то ненормальное завершение ввода-вывода фиксируется четвертым параметром.

`Blockwrite(f, Buf, count, Result)` или `Blockwrite(f, Buf, count)` – записывает в файл `count`-записей из буфера ввода `Buf`. Если указан четвертый параметр `Result`, то он получает реальное число записанных записей (блоков). Если, например, на диске не хватает места, то количество записанных элементов может не совпадать с заданным в `count`. Если четвертый параметр не указан и количество требуемых и реально выведенных записей не совпадает, то возникает ошибка ввода-вывода.

Пример. Рассмотрим использование файлов без типа для ускорения обмена данными между внешней и оперативной памятью.

```
program pr_best;
var
     a, b, c : array[1..2000] of real;
                    {компилятор распределяет их в цепочку
                     один за другим (специфика pascal)}
     f       : file;
     rez     : integer;
begin
      ...                {заполняем массивы}
 assign(f, 'abc.dat');
```

```
rewrite(f, sizeof(a));
blockwrite(f, a, 3, rez);
                        {или три раза: из a, из b, из c}
if rez = 3 then writeln('okey')
           else writeln('error');
close(f); …
end.
```

Если нужно быстро сбросить информацию из файла в память, тогда возможно следующее использование файла без типа:

```
… reset(f, sizeof(a));
blockread(f, a, 3, rez);
if rez = 3 then writeln('okey')
           else writeln('error');
                       {дальше с массивами a, b, c
                        можно работать по отдельности}
close(f);…
```

Сравните описанный здесь вариант обмена данными между внешней и оперативной памятью с обменом при помощи типизированных файлов.

Задачи

1. В некотором файле имеется вещественное число a и коэффициенты многочлена $P_n(x)$ степени n. Получите коэффициенты следующих многочленов и запишите их в файл:

а) $(x - a)P_n(x)$;

б) $(x^2 + 2ax + 3)P_n(x)$;

в) $(x^2 - a^2)P_n(x)$.

2. В некотором файле имеются коэффициенты многочлена $P_n(x)$ степени n. Получите коэффициенты многочлена $P_n'(x)$ и запишите их в файл. Подсчитайте $P_n'(1)$, $P_n'(2)$, $P_n'(3)$.

3. В некотором файле заданы вещественные числа s, t и коэффициенты многочлена $P_n(x)$. Найдите значение $\int_s^t P_n(x)\,dx$.

4. Даны вещественные числа a_0, a_1, …, a_5. Получите коэффициенты многочлена шестой степени $(x - a_0)\,(x - a_1)\cdots(x - a_5)$. Запишите их в файл.

Задачи для самопроверки

1. Чему равен размер файла `f` в следующей программе, если функция `FileSize(f)` возвратила значение `1000`?

```
Type
  dat  = record
           den : 1..31;
           mes : 1..12;
           god : integer;
         end;
  stud = record
           fio  : string[10];
           dr   : dat;
           note : array[1..5] of byte;
         end;
var  f  : file  of  stud;
```

Варианты ответа:

а) `1000 байт`;
б) `20000 байт`;
в) `19000 байт`;
г) `22000 байт`.

2. В файл `f1.dat` записывается последовательность чисел. Определите, сколько чисел будет записано в файл `f2.dat` в результате выполнения следующего фрагмента программы.

```
var
  f1      : file  of  integer;
  f, f2   : file;
  i, k, x : integer;
  m       : array[1..4] of integer;
begin
   assign(f1, 'f1.dat');
   rewrite(f1);
   assign(f2, 'f2.dat');
   rewrite(f2,2);
   for i :=1  to  8 do
    begin
      x:=random(66);
      write(f1,x);
    end;
   k:=filesize(f1) div  4;
```

```
   close(f1);
   assign(f, 'f1.dat');
   reset(f,2);
   for i :=1  to  k do
    begin
     blockread(f, m, 4);
     blockwrite(f2, m, 4)
    end;
   close(f);
   close(f2);
 end.
```

Варианты ответа:

а) 8;
б) 4;
в) 16;
г) 2.

3. Пусть имеется программа, создающая файл `t2.dat`:

```
var  f : file of integer;
     i : integer;
begin
 assign(f,'t2.dat');
 rewrite(f);
 read(i);
 while i<>0 do
  begin
    write(f,i);
    read(i);
  end;
 close(f);
end.
```

Ответьте на следующие вопросы.

1. Можно ли прочитать пятую запись файла, минуя четыре предыдущие?
2. Изменить пятую запись файла?
3. Вставить за пятой записью новую, не переписывая файла?
4. Изменить описание переменной i на word?

Варианты ответа:

а) 1 – да, 2 – да, 3 – нет, 4 – нет;
б) 1 – да, 2 – да, 3 – да, 4 – нет;
в) 1 – да, 2 – да, 3 – нет, 4 – да;
г) 1 – нет, 2 – нет, 3 – да, 4 – да.

4. Определите результат выполнения программы:

```
Var F : Text;
Begin
 Assign(F, ’1111.txt’);
 ReWrite(F);
 WriteLn(F, ’1234’);
 Close(F);
End.
```

Варианты ответа:

а) дописывает в файл "1234";
б) ошибка компиляции;
в) стирает содержимое файла и записывает туда "1234";
г) ошибка времени выполнения.

5. Как удалить последнюю запись файла, который описан так:

```
Var      FV: file of integer;
```

Варианты ответа:

а) `Delete(FV, FileSize(FV) - 1);`
б) `Seek(FV, FileSize(FV) - 1);`
`Delete(FV)`
в) `Seek(FV, FileSize(FV) - 1);`
`Truncate(FV)`
г) `Такого способа не существует.`

Тема 9
СПЕЦИАЛЬНЫЕ СРЕДСТВА АЛГОРИТМИЧЕСКОГО ЯЗЫКА
(12 часов)

9.1. Создание динамических переменных

При выполнении программы каждая объявленная в ней переменная получила свой адрес в оперативной памяти.

В языке Pascal существуют два способа распределения памяти для переменных: статический и динамический.

При статическом распределение памяти всем переменным, объявленным в программе или в интерфейсной части модуля, выделяются в определенном месте оперативной памяти фиксированные участки оперативной памяти в соответствии с их типом.

При динамическом распределении памяти есть возможность создавать новые, заранее не объявленные переменные и размещать их на свободных участках динамической области оперативной памяти. Это достигается путем использования указателей.

Указатель – это элемент данных, представляющий собой ссылку (адрес) на начало определенного блока оперативной памяти (ячейку), где может содержаться значение данных заявленного типа. Сам указатель занимает 4 байта.

Переменные, которые размещаются в динамической области оперативной памяти (в куче – она же Heap) при помощи указателя, называются *динамическими переменными.*

Указатель может принимать значения, равные всем адресам оперативной памяти, доступным для записи данных. Указатель может быть равен стандартному значению `nil` (пусто), тогда говорят, что соответствующая переменная в оперативной памяти отсутствует, или же, что указатель не получил значение адреса в оперативной памяти.

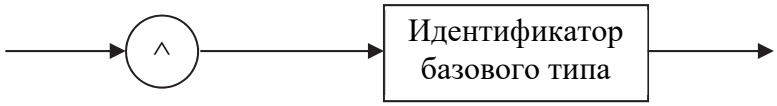
Указатель объявляется при помощи специального символа каре «^», за которым записывается идентификатор типа динамической переменной в соответствии со следующими форматами:

```
type
     имя_ссылочного типа = ^тип;
var
     имя_переменной: имя_ссылочного типа;
```

или

```
var имя_переменной: ^тип;
```

Ссылочный тип



Это ссылочные переменные. К статическим переменным обращение осуществляется по имени, к динамическим (ссылочным) – по адресу.

Указатели нужно проинициализировать – присвоить значение.

Получить адрес статического данного можно либо с помощью функции

`Addr(x) : Pointer`,

где *x* – любая переменная, имя процедуры или функции, либо используя унарную операцию @, а именно *@x*.

Пример 1.

```
type
     NameStr  = String [30];
     ArrayInt = array [1 .. 20] of Integer;
var
        RealP    : ^Real;
{указатель на значение типа Real, требующего 6 байт памяти}
        NameStrP : ^NameStr;
        ArrIntP  : ^ArrayInt;
```

Turbo Pascal допускает и такое объявление:

```
type
         PtrType  = ^BaseType;
         BaseType = record
                      x, y : Real;
                      next : PtrType;
                    end;
```

Для других определений здесь была бы ошибка, так как при описании типа `PtrType` идет использование типа `BaseType`, который еще не описан, но для типа «указатель» это возможно.

Пример 2.

```
var
       i     : Integer;
       A     : array [1..10] of Integer;
```

```
 P1    : ^Integer;
Begin  …
i := 7;
P1 := Addr(i);
{указатель на целое получил адрес переменной i}
   …
 P1 := @A[i];
       {указатель на i-е целое в массиве A}
       {можно P1 := Addr(A[i])            };
 … End.
```

Операции

Над значениями ссылочных типов (т.е. над указателями) допускаются две операции сравнения – на равенство и неравенство:

```
if P1 < > nil then …
 {проверка, получил ли указатель адрес в памяти}
if P1 = nil then …
 {проверка, ссылается ли этот указатель на nil}
```

Сравнение указателей между собой (`P1 = P2`) производить не рекомендуется ввиду неоднозначного хранения значения адреса (в виде адреса сегмента и адреса смещения).

При сравнении указателей на «<>» или «=» типы, с которыми связаны указатели, должны быть совместимыми. При присваивании значения одного указателя другому типы, с которыми связаны указатели, должны быть совместимыми по присваиванию.

```
var
     P, P1 : ^Integer;
     Pr    : ^Real;
то
     P1    := nil;  {можно}
     P     := P1;   {можно}
     Pr    := P;    {так нельзя – не совпадает
                      базовый тип}
```

Доступ к переменной по указателю

Рассмотрим ситуацию

```
P1:=@i;
```

Чтобы обратиться к переменной `i`, есть две возможности:

- использовать идентификатор `i`;
- использовать адрес этой переменной, который находится в `P1`.

В последнем случае – при косвенном доступе к переменной через указатель – используется конструкция, которая называется *разыменование.*

Разыменование означает, что для получения доступа к самой переменной через ее указатель необходимо после имени переменной-указателя поставить знак «^».

После выполнения оператора `P1 := @i` следующие операторы полностью эквивалентны:

```
i := i + 2   и   P1^ := P1^ + 2.
```

Поскольку `P1` – указатель на тип `Integer`, то `P1^` – переменная целого типа. `P1^` нужно понимать так: это переменная, на которую ссылается указатель `P1`.

В результате использования указателя на указатель возможно многократное разыменование.

Пример.

```
type   P  =  ^Integer;
var
       P1  : ^Integer;
       PP1 : ^P;
       i   : Integer;
begin          …
PP1 := @P1;    …
P1 := @i;      …
i := 2;
Write(PP1^^);        {напечатается 2?}.
```

Заметим, что следующая ситуация потенциально опасна:

```
P1 := nil; P1^ := 2;
```

потому что константа `nil` указывает на отсутствие динамической переменной.

Действия над динамическими переменными: их создание и уничтожение

Основные действия над динамическими переменными – это их создание и уничтожение.

Процедура

```
New(P)
```

предназначена для создания динамической переменной, т. е. для выделения памяти в куче, которая будет использоваться динамической переменной.

При этом динамической переменной отводится блок памяти, соответствующий размеру типа, на который ссылается указатель P, а указателю P присвайвается адрес начала блока памяти (адрес динамической переменной).

Если в ходе вычислительного процесса динамическая переменная становится ненужной, ее следует уничтожить процедурой

```
Dispose(P).
```

Эта процедура освобождает память, занятую динамической переменной, и делает значение ее указателя P неопределенным.

При использовании процедуры New надо иметь в виду, что возможен недостаток (исчерпание) памяти для размещения динамической переменной.

В этой ситуации указатель P не получает значения, а программа продолжает выполнение, и никаких сообщений не выдает.

Значит, необходимо контролировать текущее состояние динамической памяти перед каждым обращением к New.

Функция

```
MaxAvail : Longint
```

возвращает размер (в байтах) наибольшего непрерывного блока в динамической области оперативной памяти, где может быть размещена динамическая переменная.

Функция

```
MemAvail : Longint
```

возвращает суммарный размер (в байтах) свободной области динамической памяти.

Пример.

```
type
       Person  = record … end;
       PPerson = ^Person;
var P : PPerson;
     …
 if MaxAvail >=Sizeof (Person)
 then New(P);
 {можно разместить и так: P:=New(PPerson)};
     …
  Dispose (P);
```

В приведенном фрагменте программы оператор New(P) выделяет память под указатель P, который ссылается на переменную типа Person, а адрес начала выделенной памяти хранится в P. Отметим, что к New можно обращаться как к функции: P:=New(PPerson).

Оператор Dispose(P) освобождает память, которая связана с указателем P, уничтожает динамическую переменную, и значение указателя P становится неопределенным.

Задачи для самопроверки

1. Имеется описание:

```
Type
     real=^integer;
Var
     p,q: real;
     i,j: integer;
Begin
    P := @i;
    Q := @j;    ...
    i:= 5;
    j:= 3;
```

Что будет напечатано в результате выполнения следующих операторов?

```
    P^:= q^;
    if p=q then p:= nil
           else
               if p^=q^ then q:= p;
    if p=q then q^:= 4;
    Writeln(p^);
```

Сделайте рисунок к этой задаче.

2. Почему недопустимы некоторые из следующих описаний и как их исправить?

```
Type
       TA = ^0..9;
       TB = record
              p: real;
              q: TC
            end;
       TC = ^B;
```

3. Имеется описание:

```
Var
      p, q : ^integer;
      r : ^char;
```

Какие из следующих операторов ошибочные и почему:

а) `p:= q;  q:= r;  p:= nil;`
б) `r:= nil;  q:= p^;  p^:= nil;`
в) `r^:= p^;  q^:= ord(r^);`
г) `if r<>nil then r^:= nil^;`
д) `if q>nil then q^:= p^;`
е) `if q=p then write(q);`
ж) `if q<>r then read(r^).`

4. Для полинома $P_n(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$ степени n, в котором отсутствует много одночленов, задаются пары чисел $\{i, a_i\}$, причем пара, у которой $a_i = 0$, отсутствует. Требуется:

• ввести коэффициенты полинома;

• распечатать в виде полинома по спаданию степеней, например в таком виде: $5 * x\text{^}30 - x\text{^}10 + x + 1$;

• распечатать в виде полинома от меньшей степени к большей, относительно заданной в параметре переменной;

• подсчитать значение полинома при различных значениях аргумента;

• подсчитать сумму коэффициентов полинома;

• подсчитать суммы коэффициентов полинома при четных и нечетных степенях.

5. Написать модуль, в котором собрать программные ресурсы для работы с многочленом, описанным в предыдущей задаче.

9.2. Создание и уничтожение динамических массивов

При разработке программ, выполняющих одинаковые преобразования над большим количеством однотипных данных, используются массивы.

В программе при объявлении каждого статического массива всегда необходимо указывать его размеры.

Это требование делает программы зависимыми от конкретных размеров массивов, что снижает потребительские свойства программ. В таких программах используются динамические переменные-массивы.

Существуют процедуры для работы с динамическими переменными.

Процедура

```
GetMem(P, size)
```

создает новую динамическую переменную размером `size` байт, устанавливая значение указателя `P` на начало выделенной ей динамической области.

Процедура

```
FreeMem(P, size)
```

уничтожает динамическую переменную, освобождая `size` байт. После выполнения процедуры значение указателя `P` становится неопределенным.

Задачи

1. Описать процедуры для создания и освобождения в динамической памяти массива вещественных чисел, размеры которого задаются во время выполнения программы.

2. Описать процедуры для создания и освобождения в динамической памяти матрицы целых чисел в виде одномерного массива, размеры которой задаются во время выполнения программы.

Задачи для самопроверки

1. Что решает следующая программа и решает ли?

```
type   mas = array[1..1] of integer;
 var      a1              : ^mas;
          p               : pointer;
          n, i            : integer;
          s1,s2           : real;
          k1,k2           : integer;
begin
write('Введите 2<=n<=16000 '); readln(n);
mark(p);
getmem(a1^,n*sizeof(integer));
for i:=1 to n do a1^[i]:=100+random(900);
s1 := 0;   s2 := 0;   k1 := 0;   k2 := 0;
for  i := 1  to  n  do
  if i mod 2 =0
  then begin s1:=s1+a1^[i]; k1:=k1+1; end
  else begin s2:=s2+a1^[i1; k2:=k2+1; end;
release(p);
s1:=s1/k1;
s2:=s2/k2;
writeln(s1,' ', s2);
end.
```

Варианты ответа

А. Ошибок нет. Находится среднее арифметическое значение случайных чисел.

Б. Есть ошибки компиляции. Логических ошибок нет.

В. Есть ошибки выполнения. Ошибок компиляции нет.

2. Что решает следующая программа и решает ли?

```
type   item = real;
       row  = array[1..1] of item;
       prow = ^row;
       mas  = array[1..1] of prow;
       pmas = ^mas;
var    r          : pmas;
       m, n, i, j : integer;
begin
write('Введите количество строк массива ');
readln(m);
write('Введите количество столбцов массива ');
readln(n);
{1} getmem(r,m*sizeof(prow));
for i:=1 to m do
{2}   getmem(r^[i],n*sizeof(item));
for i:=1 to m do
 begin
  writeln;
  for j:=1 to n do
   begin
    r^[i]^[j] :=random(10);
    write(r^[i]^[j],' ');
   end;
  end;
{3} for i:=1 to m do
{4}  freemem(r^[i], n*sizeof(item));
{5} freemem(r,m*sizeof(prow));
end.
```

Варианты ответа

А. Нет ошибок. Создает динамический массив для размещения матрицы.

Б. Есть ошибки компиляции в 1, 2.

В. Есть ошибки в 3, 4, 5.

Г. Есть логические ошибки.

9.3. Программирование алгоритмов для работы с одномерными массивами

Процедура

```
GetMem(P, size)
```

создает новую динамическую переменную размером `size` байт, устанавливая значение указателя `P` на начало выделенной ей динамической области.

Процедура

```
FreeMem(P, size)
```

уничтожает динамическую переменную, освобождая `size` байт. После выполнения процедуры значение указателя `P` становится неопределенным.

Задачи

1. Создайте в динамической памяти массив вещественных чисел, размеры которого задаются во время выполнения программы. Заполните его случайными числами из указанного диапазона. Выведите на экран. Запишите в типизированный файл.

2. Многочлен $P_n(x) = a_n x^n + ... + a_1 x + a_0$ с вещественными коэффициентами представьте в динамической памяти. Создайте соответствующий тип данных для хранения степени многочлена и указателя на массив его коэффициентов. Определите следующие функции и процедуры для работы с такими многочленами:

• процедуру Init(*P*, name) – читает из входного файла степень многочлена и его коэффициенты, создает в динамической памяти соответствующий многочлен *P*;

• логическую функцию Equiv(*P*, *Q*) – проверяет многочлены *P* и *Q* на равенство;

• функцию Value(*P*, *x*) – подсчитывает значение многочлена *P* в точке *x*;

• процедуру Sum(*P*, *Q*, *R*) – строит многочлен *P* как сумму многочленов *Q* и *R*;

• процедуру Show(*P*, *v*) – печатает многочлен *P* в виде многочлена от переменной, односимвольное имя которой является значением символьного параметра *v*.

3. Имеется описание:

```
type
     point=^real;
     vect=array[1..100] of point;
var
     x: vect;
```

Считая, что все элементы вектора *x* не равны `nil`, опишите:

• функцию max(*x*) для нахождения наибольшего из чисел, на которые ссылаются элементы вектора *x*;

• функцию negl(*x*), значением которой является первый из элементов вектора *x*, ссылающийся на отрицательные числа, или `nil`, если таких элементов нет;

• процедуру unique(*x*), заменяющую в векторе *x* все элементы, которые ссылаются на одинаковые числа, на первый из этих элементов.

9.4. Программирование алгоритмов для работы с двумерными массивами

Рассмотрим различные подходы при работе с массивами на примере следующих задач.

Задача 1. Выполните транспонирование матрицы A_{nn} на месте ее размещения.

Задача 2. Упорядочьте строки матрицы так, чтобы элементы *k*-го столбца (*k* – задается при выполнении программы) образовывали невозрастающую последовательность.

Решение этих задач зависит от того, на каком этапе задаются размеры матрицы и какие это размеры.

Возможны, например, такие ситуации.

1. Известны размеры *n*, *m* матрицы A_{nn} (задача 1) и A_{nm} (задача 2), такие, что матрицы помещаются в сегменте данных (в статической памяти).

Эта ситуация не требует распределения матриц в `Heap`, их можно объявить следующим образом:

1)
```
const
       n=50; {задача 1}
  var
       A : array [1..n, 1..n] of Integer;
```
2)
```
const
       n=50; m=30;{задача 2}
  var
       A : array [1..n, 1..m] of Integer;
```

Заметим, что при этом действия компилятора следующие: в памяти матрица распределится цепочкой по строкам:

$$A: a_{11}, a_{12}, \ldots, a_{1n}, a_{21}, a_{22}, \ldots, a_{2n}, \ldots, a_{n1}, a_{n2}, \ldots, a_{nn}.$$

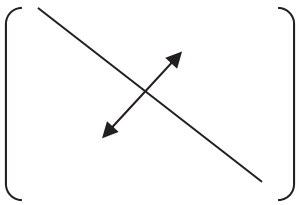
Используем это свойство потом в одном из решений.

2. Если известны размеры n, m матрицы A_{nn} (задача 1) и A_{nm} (задача 2), такие, что Sizeof($A_{n\times n}$) ≥ 64 кб и Sizeof($A_{n\times m}$) ≥ 64 кб, то матрицы необходимо распределить в динамической области по частям.

Если размеры n, m матрицы A_{nn} (задача 1) и A_{nm} (задача 2) задаются во время выполнения программы, то матрицы необходимо распределить в динамической области.

Рассмотрим более подробно решение задачи 1.

Схема транспонирования матрицы на своем месте следующая:



Отсюда получается последовательность действий:

1) размеры матрицы будем задавать при выполнении программы, учитывая, что Sizeof($A_{n\times m}$) < 64 кб;

2) матрицу A будем рассматривать как одномерный массив V, образованный строками одна за другой;

3) местоположение элемента $a_{ij}, i = \overline{1, n}, j = \overline{1, n},$ будем вычислять относительно начала массива по формуле $k := (i-1) * n + j$;

4) соответственно местоположение элемента $a_{ji}, i = \overline{1, n}, j = \overline{1, n},$ будем вычислять относительно начала массива по формуле $l := (j-1) * n + i$;

5) поскольку V содержит элементы матрицы A, для обмена элементов a_{ij} и a_{ji} нужно выполнить обмен элементов V_k и V_l.

Алгоритм.

1. Объявить матрицу как одномерный массив.
2. Задать значения ее элементов.
3. Напечатать исходную матрицу.
4. Транспонировать матрицу.
5. Напечатать полученную матрицу.

```
program transp_ukaz;        {транспонирование}
uses crt;
type
       tvect  = array [1..1] of real;
       tpvect = ^tvect;
var
       masv : tpvect;
       i, n : byte;
{$r-}           {отключили контроль выхода за пределы
             индексов объявленного массива         }

  procedure init(var masv : tpvect; n : byte);
  var
      i : integer;
  begin
   for i := 1 to n * n do
           masv^[i]:= random*100-50;
  end;

 procedure show(masv : tpvect; n : byte);
  var
     i, j : integer;
  begin
   writeln;
   writeln('матр');
   for i := 1 to n do
       begin
       for j := 1 to n do
           write (masv^[(i - 1) * n + j]:6:0);
       writeln;
       end;
  end;

 procedure transp(masv:tpvect; n:byte);
  var
       i, j, k, l : integer;
       r          : real;
      begin
   for i := 1 to n do
       for j := i + 1 to n do
           begin
 {1}         k       := (i - 1) * n + j;
```

```
 {2}          l       := (j - 1) * n + i;
              r       := masv^[k];
              masv^[k]:=masv^[l];
              masv^[l]:=r;
            end;
  end;
begin
  clrscr;
  writeln('n – ?');
  readln(n);
  if maxavail < sizeof(real)* sqr(n) then halt;
  getmem(masv, sizeof(real)* sqr(n));
  init(masv, n);
  show(masv, n);
  transp(masv, n);
  show(masv, n);
  readln;
  freemem(masv, sizeof(real) * sqr(n));
end.
```

Целесообразно было бы определить функцию

```
function ord(i, j : integer) : integer;
 begin
  ord := (i - 1) * n + j;
 end;
```

и операторы {1}, {2} заменить вызовом функции `Ord` от соответствующих аргументов: `Ord(i,j)` и `Ord(j,i)`.

Далее рассмотрим решение задачи 2.

Алгоритм.

1. Объявить матрицу.
2. Задать значения ее элементов.
3. Напечатать исходную матрицу.
4. Упорядочить строки матрицы по условию задачи.
5. Напечатать полученную матрицу.

Будем использовать *сортировку выбором*. Находим среди элементов $\{a_{1k}, ..., a_{nk}\}$ наибольший, например a_{pk}. Переставляем *p*-ю строку с первой строкой. Среди оставшихся элементов $\{a_{2k}, ..., a_{nk}\}$ снова находим наибольший. Переставляем строку, в которой он находится, со второй строкой.

Продолжаем процесс, пока не будут сравниваться $a_{n-1,k}$ и a_{nk}. На этом процесс остановится.

Получилась последовательность действий:

1) образуем цикл по $i = \overline{1, n-1}$;

2) устанавливаем $p := i$;

3) образуем цикл по $j = \overline{i+1, n}$;

4) если $a_{pk} < a_{jk}$, тогда $p := j$;

5) конец цикла по j;

6) переставляем i-ю и p-ю строки;

7) конец цикла по i.

Чтобы написать программную реализацию этого алгоритма, нужно определиться, как будет размещена матрица в памяти. Это зависит не только от ее размера, но и от того, когда (на каком этапе) задаются n и m.

Пусть n и m – константы и Sizeof(A_{nm}) < 64 кб. Матрицу распределяем в сегменте данных.

```
program primer_statika;
const
       n=10;     m=10;     k=5;
type
       titem = integer;
       tmatr = array[1..n,1..m] of  titem;
var
       a : tmatr;

procedure show(a:tmatr; n, m: byte; str:string);
{процедура печати матрицы}
var
     i, j : byte;
begin
 writeln(str:40);
 for i := 1 to n do
     begin
      writeln;
      for j := 1 to m do
         write (a[i, j] :5);
     end;
 writeln;
end;

procedure init(var a:tmatr; n, m: byte);
{процедура инициализации матрицы}
 var
     i, j : byte;
```

```
 begin
  for i := 1 to n do
     for j := 1 to m do
         a[i, j] := random(101);
 end;
procedure sort(var a:tmatr; n, m: byte);
{процедура сортировки}
  procedure swap(i, p: byte);
   {процедура обмена строк матрицы}
   var
        j : byte;
        r : titem;
   begin
        for j:=1 to m do
            begin
             r       := a[i, j];
             a[i, j] := a[p, j];
             a[p, j] := r
            end;
   end;
var
        i, j, p : byte;
begin
 for  i := 1 to n-1 do
    begin
     p := i;
     for j := i+1 to  n  do
         if a[p, k] < a[j, k]  then  p := j;
     if p<>i then swap(i, p);
    end;
 end;
begin
 randomize;
 init(a, n, m);
 show(a, n, m, 'a – исходная');
 sort(a, n, m);
 show(a, n, m, 'a – упорядоченная');
end.
```

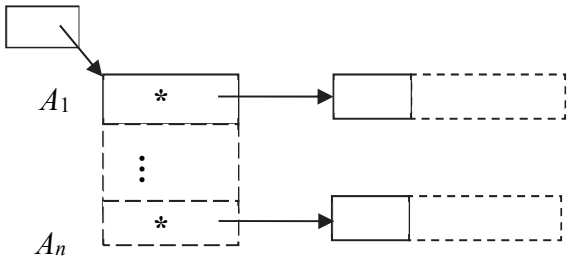
При решении задачи 2 рассмотрим ситуацию, когда размеры матрицы неизвестны и становятся известными во время выполнения программы (вводятся).

Дальнейшее решение задачи зависит от того, поместится ли матрица целиком в сегмент (Sizeof(A_{nm}) < 64 кб). Если она помещается, задачу можно реализовать, представив матрицу как одномерный массив сначала из одного элемента, а затем, если ее размеры станут известны, распределить в динамической памяти всю матрицу.

Предлагаем еще один вариант решения задачи, который будет подходить и для случая, когда Sizeof(A_{nm}) < 64 кб и Sizeof(A_{nm}) > 64 кб.

Заведем генеральный указатель на массив указателей из одного элемента на строку матрицы, которая также состоит из одного элемента.

Тогда схема представления матрицы будет следующей:



и обращение к элементу a_{ij} будет реализовываться так:

$$a_{ij} \Rightarrow A^{[i]}^{[j]}.$$

Как и прежде, используем текст предыдущей программы. Измененные операторы обозначаем комментарием {~}:

```
program  primer5;
const
         k : integer =  5;
type
         titem  = integer;
         trad   = array[1..1]  of  titem;
         tprad  = ^trad;                                  {~}
{$r-}
         tmas  = array[1..1]  of tprad;                   {~}
         tpmas = ^ tmas;                                  {~}
var
         a       : tpmas;                                 {~}
         n, m, i : word;                                  {~}

procedure show(a: tpmas; n, m: word; str:string);         {~}
var
     i, j : word;
 begin
  writeln(str:40);
```

```
  for i := 1 to n do
    begin
     writeln;
     for j := 1 to m do
         write (a^[i]^[j] :5);                          {~}
    end;
  writeln;
 end;

procedure init(a: tpmas; n, m: word);                   {~}
 var
     i, j : word;
 begin
  for i := 1 to n do
     for j := 1 to m do
         a^[i]^[j] := random (101);                     {~}
 end;

procedure sort(a: tpmas; n, m: word);                   {~}
 var
       i, j, p : word;
       r       : tprad;                                 {~}
 begin
  for i := 1 to n - 1 do
    begin
     p := i;
     for j := i + 1 to n do
         if a^[p]^[k] < a^[j]^[k] then p := j;          {~}
         if p<>i then
           begin
            r    := a^[i];                              {~}
            a^[i]:= a^[p];                              {~}
            a^[p]:= r
           end                                          {~}
     end;
 end;
begin
 writeln('n, m - ??');
 readln(n, m);
 getmem(a, n * sizeof(tmas));                           {~}
 for  i := 1  to  n  do                                 {~}
     getmem(a^[i], m * sizeof(tprad));                  {~}
 randomize;
```

```
  init(a, n, m);
  show(a, n, m, 'a – исходная');
  sort(a, n, m);
  show(a, n, m, 'a – упорядоченная');
  for  i := 1  to  n  do                                    {~}
     freemem(a^[i], m * sizeof(tprad));                     {~}
  freemem(a, n * sizeof(tmas))                              {~}
end.
```

Задачи

1. Создайте в динамической памяти матрицу вещественных чисел, размеры которой задаются во время выполнения программы. Заполните ее случайными числами из указанного диапазона. Выведите на экран. Запишите в типизированный файл.

2. Задайте функцию Sum(A), которая подсчитывает величину $x_1x_n + +x_2x_{n-1} + ... + x_nx_1$, где x_i – максимальный элемент i-й строки матрицы A.

3. Задана вещественная матрица размером $n \times m$. Определите числа массива $b_1, \ldots, b_n$ из нулей и единиц, если $b_k = 1$, когда:

а) элементы k-й строки образуют возрастающую последовательность;

б) элементы k-й строки образуют либо возрастающую, либо убывающую последовательность;

в) число отрицательных элементов k-й строки меньше числа положительных элементов;

г) число ненулевых элементов k-й строки равно 1.

4. Задана вещественная матрица размером $n \times m$. Определите числа $b_1, \ldots, b_n$, равные:

а) суммам элементов строк;

б) произведениям элементов строк;

в) наименьшим значениям элементов строк;

г) значениям средних арифметических элементов строк;

д) разностям наибольших и наименьших значений элементов строк.

5. Выполните следующие действия:

а) в заданной квадратной матрице переставьте строку с наибольшим элементом на главной диагонали со столбцом с наименьшим элементом на побочной диагонали;

б) в заданной прямоугольной матрице переставьте столбцы в порядке возрастания суммы модулей их элементов;

в) в заданной квадратной матрице разделите все ее элементы на наибольшую сумму модулей элементов в строках;

г) в заданной прямоугольной целочисленной матрице подсчитайте количество разных элементов;

д) в заданной прямоугольной матрице найдите и напечатает все одинаковые элементы и их индексы.

6. Поменяйте местами элементы матрицы так, чтобы каждая ее строка была отсортирована по возрастанию и ни один элемент в i-й строке не был больше любого элемента в $(i + 1)$-й строке.

Замечание. Матрицу нужно рассматривать как одномерный массив и выполнить его сортировку.

7. В матрице $A_{n, m}$, состоящей из вещественных чисел, поменяйте местами первый и последний, второй и предпоследний и т. д. столбцы. Значения n и m задаются во время выполнения программы.

8. В матрице $A_{n, m}$, состоящей из вещественных чисел, поменяйте местами первую и последнюю, вторую и предпоследнюю и т. д. строки. Значения n и m задаются во время выполнения программы.

9. В матрице $A_{n, m}$, состоящей из целых чисел, поменяйте местами центрально симметричные квадраты. Значения n и m задаются во время выполнения программы.

Тема 10
ВИДЕОДОСТУП
(8 часов)

10.1. Модуль CRT

Подключение модуля CRT осуществляется с помощью строки

```
uses CRT
```

Аббревиатура CRT расшифровывается как «электронно-лучевая трубка». В модуле CRT описаны типы, константы, переменные, а также реализованы специальные процедуры и функции для работы с текстовой информацией на дисплее, позволяющие:

- управлять текстовыми режимами;
- организовывать окна вывода на экране;
- настраивать цвета символов;
- управлять движением курсора.

Кроме того, модуль включает функции опроса клавиатуры и процедуры управления встроенным в ПК динамиком.

Работа с клавиатурой

Главным средством ввода информации в ПК является клавиатура. Все клавиши клавиатуры можно разделить на шесть групп.

Назначение	Пример
1. Алфавитно-цифровые и знаковые	`A .. Z, a ..z, 0 ..9, +, –, *, /, Esc, Tab, Enter, BackSpace, …`
2. Функциональные	`F1 ..F12`
3. Служебные для управления перамещением курсора и редактирования	`↑, →, ↓, ←, End, Home, PageUp, PageDown, Del, …`
4. Служебно-управляющие	`Alt, Ctrl, Shift`
5. Служебные для фиксации регистров	`Caps Lock, Scroll Lock, Num Lock, Ins`
6. Вспомогательные	`Prt_Sc, Pause/Break …`

Каждая клавиша имеет свой номер, называемый `Scan`-кодом (кодом считывания): Esc – 1; 1! – 2 и т. д. Поэтому `Shift` левый и `Shift` правый имеют разные `Scan`-коды.

В системной памяти компьютера находится буфер клавиатуры. Это циклическая очередь из 15 двухбайтовых символов.

Клавиатура имеет свой самостоятельный микропроцессор, который по Scan-коду анализирует, какая клавиша нажата (или отпущена), и передает информацию в процессор компьютера.

Клавиши разделяют на три типа.

1. Клавиши и комбинации клавиш, которые после нажатия пересылают в буфер клавиатуры ASCII-код.

Это алфавитно-цифровые клавиши, нажатые одновременно с Shift или без, а также нажатие комбинации клавиш: Ctrl и алфавитно-цифровых; Ctrl и некоторых специальных символов.

2. Клавиши и комбинации клавиш, нажатие которых посылает в буфер клавиатуры расширенный код. Это, например, функциональные клавиши: F1, …, F12; функциональные нажатые одновременно с Shift: Shift + F1, …, Shift + F12; функциональные нажатые одновременно с Ctrl: Ctrl + F1, … Ctrl + F12; функциональные нажатые одновременно с Alt: Alt + F1, …, Alt + F12; алфавитно-цифровые нажатые одновременно с Alt: Alt + алфавитно-цифровые; управляющие ↓ → … и некоторые другие.

Расширенный код состоит из двух символов: #0 и ASCII-код.

3. Клавиши и комбинации клавиш, нажатие которых не посылает в буфер клавиатуры никаких кодов. Состояние их или фиксируется в специальном месте оперативной памяти: это клавиши Shift, Alt, Ctrl, или распознается комбинация клавиш как вызов подпрограммы: например, Ctrl+Alt+Del; Alt+PrintScreen, или игнорируются.

После обработки прерывания, поступившего от клавиатуры, или изменится информация о состоянии нажатия клавиш Shift, Alt, Ctrl, Caps Lock, или в буфер клавиатуры запишется простой или расширенный код. Управление передается на функциональный уровень, если отдельным клавишам программным путем ставились в соответствие собственные функции, реализуемые при нажатии этих клавиш.

Опрос клавиатуры

Основой алгоритмов управления клавиатурой является анализ буфера клавиатуры в программе. Считывать символы из буфера можно как процедурой Read, Readln, так и функцией модуля CRT

```
Readkey : Char.
```

Команда ch: = Readkey как бы вынимает последовательно введенные в буфер клавиатуры символы – по одному за каждое обращение.

Особенности работы функции Readkey:

• полученные функцией символы не отображаются на дисплее;

• режим работы Readkey зависит от состояния буфера ввода:

– если в буфере есть символы, то функция прочитает очередной символ из буфера (тот, который был введен первым) и удалит его из буфера;

– если буфер пуст, то функция, а вместе с ней и программа, ожидает ввода символа.

Функция

```
KeyPressed : Boolean
```

возвращает значение true, если в буфере ввода с клавиатуры имеется хотя бы один символ; false, если буфер пуст.

Следующая группа операторов очищает буфер клавиатуры:

```
while KeyPressed do ch := ReadKey;
```

Ожидание нажатия клавиши (клавиш), которая вырабатывает простой или расширенный код, осуществляется таким образом:

```
repeat … until KeyPressed.
```

При старте программы буфер обычно пуст.

При работе с клавиатурой возможны следующие ситуации. Если пользователь нажмет комбинацию клавиш:

• может возникнуть программное прерывание (если система включила (on) анализ этих ситуаций):

1) Ctrl + Alt + Del;
2) Ctrl + Break;
3) Alt + PrtScr;
4) Ctrl + Alt;

• может образоваться расширенный код;

• комбинация клавиш может быть проигнорирована. Если сразу нажать Ctrl, Alt, Shift (последовательно), то приоритет имеет Alt, затем Ctrl, потом Shift;

• обычные (символьные) клавиши, даже если они нажаты одновременно с клавишами Ctrl или Shift, выдают символы с ASCII-кодом в диапазоне 1..127 (на русифицированных ПК диапазон до 255). При нажатии клавиш Ctrl + алфавитная результат определяется ее латинским названием (даже если выбрана русская раскладка клавиатуры): Ctrl + A → 1; ...; Ctrl + Z → 26. Это управляющие символы из таблицы кодов. Клавиши Esc и «пробел» в любой комбинации дают одни и те же коды: 27 и 32 соответственно.

Введенные символы остаются в буфере клавиатуры и доступны для дальнейшей обработки.

Опрос расширенных кодов

Алфавитно-цифровые клавиши, нажатые одновременно с клавишей `Alt`, а также функциональные клавиши посылают в буфер ввода с клавиатуры сразу два символа: первый – #0 (нулевой символ), второй – с числовым кодом. Вторая часть расширенного кода может совпадать с кодировкой некоторого символа. Так, функциональная клавиша F1 дает расширенный код #0#59. Если не учесть, что это расширенный код, то код #59 можно трактовать как символ «;».

Подобный механизм значительно повышает информационную отдачу клавиатуры.

Рассмотрим примеры обработки нажатия клавиш клавиатуры.

Пример 1. Программа вывода кодов нажатых клавиш.

```
program PR_key;
uses CRT;
var
    CH : Char;
begin
clrscr;
repeat
      CH := Readkey;
      if CH = #0 then
         begin                              {расширенный код}
         CH := Readkey;
         Write(' спецклавиша ');
         end
         else
             begin
              Write (' Char= ');     {алфавитно-цифровой}
              if CH >= #32 then Write(CH)
              else Write(' ^ ', CHR(ORD(CH) + 64));
             end;              {управляющий символ}
      Writeln (' ASCII= ', ORD(CH));
{т.к. код может совпадать с управляющим, поэтому печатаем номер}
until CH = #27;                              {выход по ESC}
end.
```

Если код клавиши расширенный, то к буферу клавиатуры надо обращаться два раза. Опишем функцию, которая будет возвращать символ и знак расширенного кода.

Пример 2.

```
program PR_KeyD;
uses CRT;
const
      ESC    = #27;        DEL  = #83;
      INSKEY = #82;        HOME = #71;
      F1     = #59;        F10  = #68;
      SF1    = #84;        SF10 = #93;
      CF1    = #94;        CF10 = #103;
      AF1    = #104;       AF10 = #113;
var
     ExtendKey : Boolean;
     CH        : Char;
function GetKey(var ExtendKey : Boolean) : Char;
 var
      ch : Char;
 begin
 ExtendKey := false;
 ch        := Readkey;
 if  ch = #0    then
     begin
     ExtendKey := true;
     ch        := Readkey;
     end;
 GetKey := ch
 end;
begin
ClrScr;
repeat
  ch := GetKey(ExtendKey) ;
  if  not  ExtendKey  then
      Writeln ('символ клавиши с кодом=', Byte(ch))
  else
    case ch of
      SF1..SF10 :
        Writeln('Нажата SHIFT+функциональная клавиша');
      CF1..CF10 :
        Writeln('Нажата CTRL +функциональная клавиша');
      F1 .. F10 :
        Writeln('Нажата функциональная клавиша');
      INSKEY    :
        Writeln('Нажата клавиша «вставка» ');
      DEL :
        Writeln('Нажата клавиша «del» ');
```

```
      HOME :
        Writeln('Нажата клавиша «home» ');
      else Writeln('Расширенный код #00+', Byte(ch));
   end;
until ch = ESC
end.
```

За каждой клавишей можно условно закрепить определенную ноту и таким образом на компьютере имитировать музыкальное устройство.

Пример 3. Имитация музыкального инструмента.

За цифровыми клавишами 0, 1, …, 8, коды которых #48-#55, закрепим ноты 262, 294, 330, 349, 392, 440, 494, 523. Нажатие любой цифровой клавиши позволит воспроизвести соответствующий звук.

```
program DemoInstrument;
uses CRT;
const
      M : array [1 .. 8] of Integer =
                (262, 294, 330, 349, 392, 440, 494, 523);
var    I  : Integer;
       Ch : Char;
begin
while true do
 begin
  Ch:= Readkey;
  case Ch of
    #48       : Halt;                  {клавиша 0}
    #49.. #55 : I := Ord(ch) – 48;
   else
     Write('клавише звук не назначен', ^G, 'Повторите!');
  end;
  Sound(M[i]);
  Delay(100); {здесь можно repeat until KeyPressed}
  NoSound
 end
end.
```

Управление курсором

Работа процедур Write и Writeln вызывает перемещение курсора по экрану дисплея.

Процедура

GOTOXY(x, y)

перемещает курсор в позицию x (столбец) и y (строка) относительно текущего окна (по умолчанию $1 \le x \le 80$, $1 \le y \le 25$ – это весь экран).

Функции

WhereX, WhereY

дают соответственно значение x и y – координат курсора относительно текущего окна.

```
Write('Курсор находится в столбце', WhereX);
Write('Курсор находится в строке ', WhereY);
```

Пример. Звуковое сопровождение украсило и следующую программу. Что выводит следующая программа?

```
uses crt;
type
    Stroka = String[160];
var
    vhod   : Stroka;
procedure Gostring(x, y : Byte; inst : Stroka);
var
       st1 : stroka;
       i   : Byte;
procedure Zwon;
 begin  Sound(1000); Delay(50); Nosound;  end;
Begin
 St1:='';
 Clrscr;
 St1:=st1+inst;
 Writeln(st1);
 Writeln;
 for i:=1 to Length(st1) do
  begin
   Delete(st1,1,1);
   Gotoxy(x, y);
   Write(st1);
   Zwon;
   Delay(500);
   DelLine;
{удаляет с экрана строку, в которой находится курсор}
  Gotoxy(20,10);
  Writeln;
  Writeln(st1);
end;
begin
 Gostring(10, 20,'abcdefghijklmnopqrstuvwxyz');
end.
```

Задачи

1. В зависимости от нажатия клавиш ↓ → ↑ ← заставьте двигаться символ * и при помощи этого движения напишите свое имя.

2. Запрограммируйте несложную мелодию и проигрывайте ее до той поры, пока не будет нажата клавиша ESC.

3. Осуществите генерацию мелодий путем закрепления за отдельными клавишами соответствующих нот.

4. Напишите программу, которая имитирует:
   - пение птиц;
   - звук сигнала «скорой помощи»;
   - завывание ветра за окном;
   - сигналы точного времени;
   - звук взлетающего вертолета.

5. Движение строки по горизонтали.

6. Движение строки по вертикали.

7. Движение строки по границе экрана.

8. Неожиданно появляется ваша фамилия…

10.2. Видеодоступ. Работа с экраном

Основным устройством для отображения информации, которая вводится и выводится, является *дисплей*. Поддержка его работы осуществляется при помощи соответствующего *адаптера* – специальной платы для подключения дисплея к компьютеру. Изображение на экране формируется в двух основных режимах: текстовом и графическом. В первом – на экран выводятся символы, во втором – изображение выводится как объединение цветовых точек – *пикселей*.

Каждый пиксель, или точка, при цветном режиме может светиться разными цветами. Пиксели обладают такими размерами, что промежутки между соседними пикселями почти отсутствуют. Если группа смежных пикселей светится, то они воспринимаются как цельный участок.

Адаптер связывает микропроцессор с дисплеем через устройство, которое называется контроллером электронно-лучевой трубки. В составе контроллера: *программируемые порты ввода-вывода, знакогенератор, оперативная электронная память.*

Сейчас используются VGA (Video Graphics Adapter) и SVGA-адаптеры.

Образ одного экрана хранится в электронной оперативной памяти в закодированном виде. *Страница* – образ экрана в памяти ПК (полная копия экрана), которую можно отобразить мгновенно в любое время.

Существует жесткая взаимосвязь между размером электронной памяти (видеобуфером), разрешением экрана, количеством видеостраниц и «диапазоном» цветов пикселя.

В текстовом режиме под один символ дается поле – матрица определенных размеров. Размеры матрицы зависят от разрешения экрана (обычно 8×8, 8×14, 8×16, 9×14, 9×16).

Если разрешение экрана 640×200, то выбрано 25 строк и 80 столбцов при матрице 8×8 (640×200 = 25×8×80×8).

Разрешение экрана для VGA-адаптера

Адаптер	Разрешение	Количество цветов	Число видеостраниц	Объем видеобуфера
SVGA, VGA	640×200 640×350 640×480	16 16 $>2^8$	4 2 1	256K

В VGA-адаптере видеобуфер – 256 кб и более. Начало видеобуфера зависит от адаптера.

Текстовый режим работы дисплея поддерживает модуль CRT, графический – Graph.

Установка текстового режима

Текстовые режимы служат для отображения символов из таблицы кодов ПК (ASCII-кодов) и характеризуются количеством символов в строке и строк на экране. Возможна установка режима в соответствии с описанными выше режимами работы текущего адаптера.

Процедура

```
TextMode(M),
```

где M – типа Word, переключает текстовые режимы вывода информации на дисплей.

Специально для этой процедуры в модуле CRT определены следующие константы (можно пользоваться как названиями, так и числовыми эквивалентами):

Значение M		Разрешение	Замечание
2	BW80	80×25	Черно-белый при цветном адаптере
3	CO80 = C80	80×25	Цветной
256	Font8×8	80/40×43	Цветной, адаптор EGA
		80/40×50	Цветной, адаптор VGA

Все остальные числовые значения до 65 535, которые не совпадают ни с одной из указанных констант, включают процедурой TextMode режим C80.

Константа Font8×8 используется в адаптерах EGA и VGA, является дополнительной для них. Например, TextMode(CO80+Font8×8) изменит режим разрешимости экрана с 80×25 на режим 80×43 (EGA) или 80×50 (VGA), поскольку константа Font8×8 обеспечивает построение символов не из матриц 8×14 и 8×16, а из матриц 8×8. Когда основной режим не задан, то по умолчанию берется режим BW80.

Процедура TextMode очищает экран текущим цветом, устанавливает курсор в левую верхнюю позицию с координатами (1, 1), выполняет действия для наилучшего отображения информации на экране.

Значение установленного режима запоминается в переменной LastMode типа Word, описанной в CRT.

Очистка экрана и управление строками на экране

ClrScr – полностью очищает экран или окно, помещая курсор в левый верхний угол (цвет определяется текущим цветом фона).

ClrEol – удаляет все символы в строке от позиции курсора до конца строки, при этом можно изменить цвет фона в строке. Курсор остается на месте.

DelLine – удаляет с экрана строку, в которой находится курсор. Все строки, расположенные ниже, перемещаются на одну строку вверх. В нижнюю часть активного окна добавляется новая строка.

InsLine – в рамках текущего окна вставляет пустую строку с позиции расположения курсора.

Вывод на экран

Для установки цветов символов, которые выводятся, используется процедура TextColor(Color), где 0≤Color≤15. Для установки цвета фона – процедура TextBackGround(Color), где 0≤Color≤7.

Эти процедуры связаны с описанной в модуле CRT переменной TextAttr типа Byte, где фиксируется установка цветов символа и фона, а изменение цветов на экране происходит при выполнении каких-либо операторов работы с экраном, например Write, Clrscr и др.

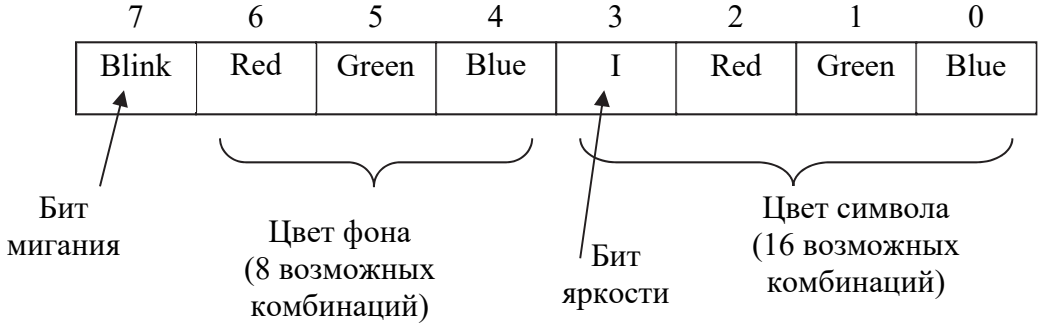
Если цвет символа совпадает с цветом фона, то символ становится невидимым. Так можно «спрятать» текст на определенное время.

Для управления яркостью используются стандартные процедуры LowVideo (наименьшая), NormVideo (нормальная, восстанавливает первоначальный режим яркости), HighVideo (повышенная).

Установка атрибутов цвета символа и фона

Если дисплей работает в текстовом режиме, тогда каждой позиции символа на экране отводится два байта памяти. Первый байт содержит номер кода `ASCII` символа, а второй – атрибуты символа. Цветные адаптеры позволяют выводить в цвете как сам символ, так и фон. Преобразование двух байт, в которых размещены код символа и цвет символа, в пиксельное представление выполняется специальным аппаратным устройством – генератором символов – знакогенератором.

Рассмотрим, как формируется байт атрибута `TextAttr`. По умолчанию устанавливается палитра – набор цветов – на основе цветов: `Red`, `Green`, `Blue`.



`Blink=2`^{7}`=128`

Основная палитра имеет следующие комбинации цветов.

Код цвета	Номер цвета в палитре	Название цвета
Основные цвета		
0000	0	Черный (black)
0001	1	Синий (blue)
0010	2	Зеленый (green)
0100	4	Красный (red)
Смесь основных цветов		
0011	3	Циан (бирюзовый) (cyan)
0101	5	Фиолетовый (magenta)
0110	6	Коричневый (brown)
0111	7	Светло-серый (неярко-белый) (LightGray)
Дополнено интенсивностью		
1000	8	Серый (DarkGray)
1001	9	Светло-синий (LightBlue)
1010	10	Светло-зеленый (LightGreen)
1011	11	Светло-циан (LightCyan)
1100	12	Светло-красный (LightRed)
1101	13	Светло-фиолетовый (LightMagenta)
1110	14	Светло-коричневый (желтый) (yellow)
1111	15	Ярко-белый (white)

Выводы

Номер бита	Значение бита	Толкование
0	1	Синий включен в основной цвет
1	1	Зеленый включен в основной цвет
2	1	Красный включен в основной цвет
3	1	Символ выводится с высокой интенсивностью
4	1	Синий включен в основной цвет фона
5	1	Зеленый включен в основной цвет фона
6	1	Красный включен в основной цвет фона
7	1	Символы мигают

Описанные комбинации битов образуют 16 регистров 0-й палитры.

Бывают и другие палитры, которые переключаются при помощи прерываний.

Процедуры установки цвета символа `TextColor` и цвета фона `TextBackGround` связаны с переменной `TextAttr`, но можно сразу менять значение переменной:

`TextAttr:=$1F;` 00011111 – ярко-белый на синем фоне;

`TextAttr:=$71;` 01110001 – синим на белом фоне;

`LowVideo` ⇔ `TextAttr:=TextAttr and $F7` (11110111)
(установка в ноль бита яркости и контроль его до отмены).

`HighVideo` ⇔ `TextAttr:=TextAttr or $08` (00001000)
(установка в 1 бит яркости).

Вместо процедур:
```
TextColor (Yellow+Blink);
TextBackGround (Red);
```
можно написать: `TextAttr:=Yellow+Blink+Red shl 4;`

Задача 1. Рассмотрим программу вывода 40 окон, координаты которых и цвет фона выбираются случайно.

```
program demorandomwindow;
uses crt;
const  randwx = 80;  randwy = 25;
var    x, y, i : byte;
begin
randomize;
clrscr;
for i:=1  to  40  do
   begin
    x := succ (random(randwx));
    y := succ (random(randwy));
    window(x, y, x+random(20), y+random(8));
    textbackground (random(8));
```

```
    clrscr;
    write(' window', ^g, i);
    delay(300);
   end;
window (1, 1, 80, 25);
clrscr;
gotoxy (33, 25);
write ('автор – я!');
window (1, 1, 80, 24);
clrscr;
gotoxy (33, 15);
write ('okey!');
delay(300);
clrscr;                    {останется – "автор – я!"}
end.
```

Текстовые окна

Модуль CRT поддерживает возможность в любой момент работы программы использовать для вывода определенную часть экрана, так называемое *окно*. Размеры окна задает программист процедурой

Window (X1, Y1, X2, Y2),

где параметры X1, X2, Y1, Y2 типа Byte. (X1, Y1) – координаты верхнего левого, (X2, Y2) – координаты нижнего правого угла окна.

Если следующие условия не выполняются: $1 \le X1 \le X2 \le X_{max}$ и $1 \le Y1 \le$ $\le Y2 \le Y_{max}$, то окно создано не будет. После выполнения процедуры Window окно становится текущим. Это значит, что все операции с экраном относятся к той его части, которая определена координатами открытого окна. При этом перемещение курсора происходит только в пределах текущего окна, и позиция с координатами (1, 1) – это левый верхний угол окна.

После активизации процедуры Window модуль CRT формирует две специальные переменные WindMin и WindMax типа Word, в которых фиксируются размеры текущего окна (отсчет при этом ведется от (0, 0)). Можно получить значения текущего окна

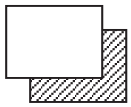
X1: = Lo (WindMin)+1, Y1: = Hi (WindMin),

X2: = Lo (WindMax)+1, Y2: = Hi (WindMax).

Здесь Lo – младший байт младшего слова своего целочисленного аргумента. Hi – старший байт младшего слова своего аргумента. Эти коорди-

наты фиксируются в специальных местах памяти. Местоположение курсора и его вид также фиксируются в оперативной памяти. Переменные `WindMin` и `WindMax` нам нужны, если окно открыто через случайные координаты.

Задание. В середине экрана сделать окно с тенью следующего вида и окантовать его.



Алгоритм. Создаем темное окно (открываем и закрашиваем). Открываем поверху первого второе окно и закрашиваем светлым цветом; стандартная процедура `ClrScr` после открытия окна стирает в видеопамяти все, что попало «под окно». Окантовываем последнее окно символами псевдографики. Открываем новое, третье, окно так, чтобы окантовка попала в предыдущее окно.

Задача. Напишите программу, которая создает несколько неналоженных окон и затем совершает переход из одного окна в другое.

Решение. Для сохранения координат окон, которые создаются, опишем специальные типы и данные.

```
uses crt;
type
      winrecord = record
                  xl, yl, xr, yr : byte
                  end;
const     maxwin = 3;
type      tmw    = array[1..maxwin]  of  winrecord;
var       i      : integer;
const
      mw : tmw = ((xl : 10; yl : 5; xr : 15; yr : 10),
                  (xl : 20; yl : 5; xr : 25; yr : 10),
                  (xl : 30; yl : 5; xr : 35; yr : 10));
begin
clrscr;
for i := 1  to  maxwin  do
    begin
    with mw[i] do window(xl, yl, xr, yr);
         clrscr;
         delay(500);
         end;
    readln;      {окна созданы по неслучайным координатам.
                           обращение делаем аналогично.}
```

```
    for i := 1  to  maxwin  do
        begin
        with mw[i] do
             window (xl, yl, xr, yr);
             delay(5000);
             writeln ('old');
             end;
end.
```

Отметим, что на экране может находиться несколько окон, но только одно из них является активным в каждый момент времени. Именно с этим окном связан курсор, и все процедуры ввода-вывода выполняются относительно активного окна.

Задачи

1. Изобразите флаг Греции.

2. Изобразите флаг Швеции.

3. Изобразите полосатый коврик.

4. Изобразите мозаичный коврик.

5. В разных окнах выведите: динамически распределенную матрицу, проинициализированную числами из файла, ее размеры и фамилию с инициалами автора программы.

6. Выведите на экран текстовую информацию в виде «бегущей строки».

7. Разработайте программу, которая при нажатии клавиши F9 выводит на экран текстовое окно, а затем позволяет при нажатии клавиш со стрелками перемещать соответствующие границы и уменьшать размеры этого окна.

8. Выведите на экран простейшее горизонтальное меню, в котором выбранный вариант должен выделяться яркостью изображения или цветом.

9. Выведите на экран простейшее вертикальное меню, в котором выбранный вариант должен выделяться яркостью изображения или цветом.

10. Задана строка символов. Выведите эту строку на экран так, чтобы каждый символ появлялся на экране справа и перемещался в горизонтальном направлении налево к первой незанятой позиции. После того как этот символ займет свое место, должен начинаться вывод следующего символа и т. д.

10.3. Графическое программирование

Стандартное состояние ПК после включения – работа экрана в текстовом режиме. Любая программа, использующая графические средства компьютера, должна определенным образом инициировать графический режим работы дисплейного адаптера.

Более 90 подпрограмм, содержащихся в модуле `Graph` в файле `GRAPH.TPU`, являются мощными средствами для работы с графической информацией. Подключение модуля стандартное:

```
uses Graph.
```

Настройка графических процедур на работу с конкретным адаптером происходит путем подключения соответствующего графического драйвера. *Драйвер* – это специальная программа, которая осуществляет управление теми или иными техническими средствами ПК. Загрузочные модули драйверов хранятся в специальном файле с расширением `BGI` (`Borland Graphics Interface`). Обычно такие модули записаны в поддиректории `BGI` директории, содержащей `Borland Pascal`.

Для `EGA` и `VGA` адаптеров используется драйвер `EGAVGA.BGI`. В библиотеке модуля `Graph` нет драйверов для новейших адаптеров, и поэтому приходится использовать драйвер `EGAVGA.BGI` и довольствоваться его относительно скромными возможностями.

В графическом режиме экран дисплея условно делится прямоугольной сеткой, каждый элемент которой имеет свои координаты (`x`, `y`). Максимальная величина `x` и `y` зависит от типа дисплея, драйвера и режима его работы. Графическому режиму, как и текстовому, свойственно понятие текущего указателя-курсора, который в графическом режиме невидим.

Образ любого изображения на экране хранится в видеопамяти и восстанавливается примерно через каждые 1/25 с (в текстовом режиме – 1/60 с). Там каждому пикселю отводится свой участок (бит, 2 бита, 4 бита, 8 битов и т. д.), где хранится информация о его состоянии и цвете.

Драйвер имеет несколько режимов работы. При инициализации графического режима процедурой `InitGraph` надо указать драйвер, который используется, желательный режим и путь к драйверу.

Если используется драйвер `VGA` (`GraphDriver` = 9), то значение режима можно выбирать из следующего множества, приведенного в таблице.

Название режима	Значение константы режима	Разрешение	Количество цветов	Количество видеостраниц
VGA	VGAL0=0	640×200	16	4
	VGAMed=1	640×350	16	2
	VGAHi=2	640×480	16	1

Драйвер можно задавать зарезервированной константой `detect` (= 0). Это означает, что при инициализации графического режима происходит определение доступного драйвера и установка лучшего разрешения экрана.

Инициализация и выход из графического режима

Каждый драйвер находится в отдельном файле на диске и содержит выполняемый код и данные. Процедура

`InitGraph(Graphdriver, Graphmode, Pathdriver)`

анализирует графическое аппаратное обеспечение, загружает в динамическую память и инициализирует соответствующий графический драйвер, переводит систему в графический режим.

Процедура `CloseGraph` освобождает память от драйвера и восстанавливает предыдущий видеорежим. Процедура `RestoreCrtMode` выполняет переход в текстовый режим, а `SetGraphMode` – в графический. `SetGraphMode` устанавливает по умолчанию все параметры (палитру, текущий указатель, цвет символов, фон и т. д.).

Очистка экрана проще выполняется процедурой `ClearDevice`, а сложнее – `GraphDefaults`. Последняя процедура неявно вызывается при инициализации графического режима.

Пример.

```
program Test;
uses Graph;
var
      GraphDriver : Integer;                    {драйвер}
      GraphMode   : Integer;                    { режим }
      ErrorCode   : Integer;
begin
DirectVideo  :=  false;
GraphDriver  :=  detect;
    {detect  –  для  автоматического  определения  типа
 драйвера аппаратных средств. Если стоит detect, то GraphMode
установится автоматически, иначе надо задавать GraphMode }

InitGraph(GraphDriver, GraphMode, '');
    {если драйвер в текущем каталоге, то путь задается
     пустой строкой, иначе ставим путь поиска библиотеки}

ErrorCode := GraphResult;
```

```
        {после опроса этой функции она сбрасывает
         свои результаты в 0. Так как повторный вызов
         функции GraphResult дал бы не тот результат, то мы
           код ошибки сохраняем в переменной ErrorCode }
  if ErrorCode <> grOk then            { Ошибка!}
    begin
      Writeln('Ошибка графики:',GraphErrorMsg(ErrorCode));
      Writeln ('Программа завершена');
      Halt(1);
     end;
        {*******Работа в графическом режиме*******}
  Outtext(' Графический режим. Нажмите <Enter>');
  Readln;
        {Задержка экрана, пока не нажмем <ввод> }

  RestoreCrtMode;

        {********Работа в текстовом режиме******}
  Writeln('Текстовый режим. Нажмите <Enter>');
  Readln;
  SetGraphMode (GraphMode);
  Outtext('Снова графический режим. Нажмите <Enter>');
  Readln;
  Closegraph;
  end.
```

Через соответствующие подпрограммы до или после инициализации графического режима можно получить сведения об установленном режиме.

`GetDriverName` – функция возвращает имя графического драйвера.

`GetGraphMode` – функция возвращает номер графического режима.

`GetMaxMode` – функция возвращает максимальное значение номера режима для текущего загруженного драйвера.

`DetectGraph` – процедура возвращает значение номера текущего драйвера и режима и служит для тестирования графического адаптера.

`GetModeName` – функция возвращает имя заданного графического текущего режима (строка типа '640 × 200 VGA').

Эти процедуры и функции используются для организации диалогового управления графическими режимами.

Управление видеостраницами

Память адаптеров (видеобуфер) EGA и VGA поддерживает более одной видеостраницы. Обычно страницы нумеруются начиная с 0. Для работы с видеостраницами предназначены две процедуры: SetActivePage и SetVisualPage. Их часто используют при создании анимационных программ.

Процедура

```
SetVisualPage(Page)
```

устанавливает в качестве текущей, т. е. такой, которая отображается на экране, видеостраницу с номером Page.

Процедура

```
SetActivePage(Page)
```

делает страницу с номером Page активной, на которой будут происходить все графические операции.

Перемещение курсора

Текущий указатель в графическом режиме невидим, но он присутствует как курсор. Его координаты фиксируются относительно текущей страницы в определенном месте оперативной памяти.

Процедура

```
MoveTo(x, y)
```

перемещает текущий указатель в точку с координатами (x, y). Например, `MoveTo(200, 100).`

Процедура

```
MoveRel(dx, dy)
```

перемещает текущий указатель на dx точек по горизонтали и на dy – по вертикали. Например, MoveRel (5, 10).

Функции GetX и GetY – возвращают соответственно значения x- и y-координат текущего указателя. Чтобы координаты не выходили за пределы экрана (в таком случае система не дает ошибки, а отсекает выход), требуется их контролировать функциями

```
GetMaxX : Integer и GetMaxY : Integer,
```

которые возвращают максимально допустимые значения для координат x и y соответственно.

При помощи этих функций можно следующим образом контролировать принадлежность координат точки экрана:

```
if not((x>GetMaxX) or (y>GetMaxY)) then MoveTo(x, y).
```

Для вычисления координат центра получим

```
Xcenter := GetMaxX div 2;
Ycenter := GetMaxY div 2;
```

Вывод точки и определение параметров пикселя

Изображение точки можно получить процедурой

```
PutPixel(x, y, Color),
```

где `x, y : Integer` – заданные координаты, `Color : Word` – цвет, который задается по номеру или по названию из таблицы цветов.

Например, изображение точки в центре экрана зеленым цветом получим вызовом процедуры

```
PutPixel(Xcenter, Ycenter, 2).
```

Функция

```
GetPixel(x, y) : Word
```

дает номер цвета пикселя с координатами (`x`, `y`).

Вывод отрезка

Процедура

```
Line(x1, y1, x2, y2)
```

рисует отрезок прямой, с началом в точке `(x1, y1)` и концом в точке `(x2, y2)` текущим цветом.

Цвет можно предварительно задать процедурой

```
SetColor(Color),
```

где `Color` задается номером или именованной константой.

Из текущей точки с приращением `dx`, `dy` линию чертит процедура

```
LineRel(dX, dY).
```

Из текущей точки в точку с координатами `(x, y)` линию чертит процедура

```
LineTo(X, Y).
```

Turbo Pascal позволяет чертить линии различного стиля: тонкие, широкие, штриховые, пунктирные и т. д., а также задавать собственный режим вывода. Стиль линии устанавливается процедурой

```
SetLineStyle(LineStyle, Pattern, Thickness);
```

Здесь `LineStyle : Word` – тип линии, `Pattern : Word` – образец, `Thickness : Word` – толщина.

Тип линии `LineStyle` может принимать значения, которые приведены в таблице.

Константа типа линии	Тождественное значение	Описание линии
Solidln	0	Непрерывная
Dottedln	1	Линия из точек
Centerln	2	Штрих-пунктир
Dashedln	3	Штриховая линия
UserBitln	4	Тип программиста

Если тип линии определяется из стандартных типов, тогда `Pattern=0`; если задается пользовательский, т. е. с номером 4, тогда `Pattern≠0`. В этом случае вторым параметром указывается примитив.

Например, примитив \$5555 (=0101010101010101_2) на месте `Pattern` показывает, что линия будет перемежаться пробелами и точками по принципу: там, где стоит 0, будет выводиться пиксель цветом фона, а где стоит 1 – цветом линии.

`Thickness` – толщина линии принимает следующие значения:

`NormWidth` (= 1) – нормальная толщина в 1 пиксель;

`ThickWidth` (= 3) – жирная линия в 3 пикселя.

Если какие-либо параметры, кроме последнего, имеют недопустимые значения, то процесс игнорируется, а `graphresult` принимает значение 11. Если `Thickness` задан некорректно, то толщина берется как `NormWidth`.

В ряде случаев при использовании `Line`, `LineRel`, `LineTo` и некоторых других процедур требуется устанавливать режим вывода линии на экран (копирование или использование логических операций). Для установки режима вывода кусочно-линейных примитивов предназначена процедура

`SetWriteMode(Mode)`.

Значение параметра `Mode` определяется стандартными константами:

```
const
       CopyPut = 0;
       XORPut  = 1;
       ORPut   = 2;
       ANDPut  = 3;
       NOTPut  = 4.
```

Если `Mode=0`, то пиксели, расположенные на отрезке прямой линии, переопределяют пиксели на экране, и, таким образом, линия на экране имеет заданный текущий цвет.

Если `Mode=1`, то пиксели, образующие линию, имеют код цвета, который образуется операцией `xor` кода текущего цвета и кода цвета пикселов на экране, через которые линия проходит.

Как частный случай такого поведения можно стереть выведенную линию с экрана, исполнив вывод линии еще раз, так как

```
a xor b xor b = a.
```

Для определения текущих характеристик линий служит процедура

```
getlinesettings(lst),
```

где `lst : linesettingstype`, а тип `linesettingstype` такой:

```
type
     linesettingstype = record
                         linestyle : integer;
                         upattern  : word;
                         thickness : integer;
                        end;
```

Задачи

1. Выведите декартовую систему координат. Обозначьте оси и масштабные отрезки.

2. Напечатайте график функции, заданной в полярной системе координат, в декартовой системе координат.

3. Сделайте графики и фигуры в графическом режиме.

Необходимо разделить экран на четыре части (создать четыре окна), в каждой части нарисовать систему координат, нанести линии масштаба на все оси, отобразить наименьшее и наибольшее значения на каждой оси, нарисовать графические фигуры по вариантам а), б), в), г) (соответственно) и напечатать названия фигур.

Кривые в основном задаются в полярных координатах. Связь между декартовыми (x, y) и полярными (r, φ) координатами следующая:

$$x = r\cos\varphi,\ \ y = r\sin\varphi,\ \ r = \sqrt{x^2 + y^2},$$
$$\cos\varphi = x / \sqrt{x^2 + y^2},\ \ \sin\varphi = y / \sqrt{x^2 + y^2}.$$

Если кривые задаются параметрически $x = \varphi(t)$, $y = \psi(t)$, предполагается, что параметр принимает все возможные значения из области допустимых значений, если это не оговорено отдельно.

При решении создайте отдельный модуль, собирающий такие ресурсы: функции подсчета координат $x = \varphi(t)$, $y = \psi(t)$ или $x = r\cos\varphi$, $y = r\sin\varphi$, подпрограммы подсчета наименьших и наибольших значений соответствующих функций и подсчета масштаба; подпрограммы для рисования осей и их штриховки; подпрограммы вывода собственно графики.

Варианты

1. а) лемниската Бернулли $(x^2 + y^2)^2 = 2a^2(x^2 - y^2)$ или $r^2 = 2a^2\cos(2\varphi)$;
   б) лист трилистника $r = 4(1 + \cos(3\varphi)) - 4\sin^2(3\varphi)$;
   в) лист щавеля $r = 4(1 + \cos(3\varphi)) + 4\sin^2(3\varphi)$;
   г) лист плюща $r = 3(1 + \cos^2\varphi) + 2\cos\varphi + \sin^2\varphi - 2\sin^2 3\varphi\cos^4(\varphi/2)$.

2. Овалы Кассини $r^2 = c^2\left(\cos(2\varphi) \pm \sqrt{\cos^2(2\varphi) + (\frac{a^4}{c^4} - 1)}\right)$:

   а) $c = a$;
   б) $0 < a < c$;
   в) $a > c > 0$;
   г) $a > c\sqrt{2} > 0$ (например, $c\sqrt{2} = a - \varepsilon,\ \varepsilon = 1,\ 0{,}1,\ 0{,}01,\ \ldots$).

3. а) гиперболическая спираль $r = a / \varphi$;
   б) логарифмическая спираль $r = ae^{\varphi\,\mathrm{ctg}\,a}$;
   в) спираль Архимеда $r = a\varphi$ или $\sqrt{x^2 + y^2} = a\,\mathrm{arctg}(y / x)$;
   г) спираль Галилея $r = a\varphi^2 - q, q \geq 0$.

4. «Розы»:
   а) $r = \sin(\sqrt{2}\varphi)$; в) $r = \sin(3\varphi)$;
   б) $r = \sin(4 / 3\varphi)$; г) $r = \sin(5 / 4\varphi)$.

5. Кардиоиды:
   а) $r = \cos(\varphi/2)$; в) $r = \sin(\varphi/2)$;
   б) $r = a(\cos(\varphi) - 1),\ a < 0$; г) $r = a(\cos(\varphi) + 1),\ a > 0$.

6. Улитки Паскаля $(x^2 + y^2 - 2ax)^2 - l^2(x^2 + y^2) = 0$, или
   $r = 2a\cos\varphi + l$, или $x = 2a\cos^2 t + l\cos t,\quad y = 2a\cos t\sin t + l\sin t$:
   а) $l > 2a$; в) $l < 2a$;
   б) $l = 2a$; г) $r = 2\cos(t) - 1,\ x \geq 0$.

7. «Розы» симметрические:
   а) $r = |\sin(2\varphi)|$; в) $r = |\sin(3\varphi)|$;
   б) $r = |\sin(3 / 4\varphi)|$; г) $r = |\sin(2 / 5\varphi)|$.

8. Фигуры $x = a\cos^2(t) + b\cos(t);\ y = a\cos(t)\sin(t) + b\sin(t),\ a > 0,\ b > 0,$ $t \in [0; 2\pi)$:

а) $b \geq 2a$; в) $a > 2b$;

б) $a < b < 2a$; г) $2b > a > b$.

9. Фигуры:

а) $r = a\sqrt[4]{2/(2 - \sin^2(2\varphi))}$, или $x^4 + y^4 = a^4$;

б) $r^2 = -\cos(2\varphi)/(\cos^4(\varphi) + \sin^4(\varphi))$, или $x^4 + y^4 + x^2 - y^2 = 0$;

в) $r = \sqrt{2/(2 - \sin^2(2\varphi))}$, или $x^4 + y^4 = x^2 + y^2$;

г) $r = 4\sin(2\varphi)\sin(\varphi)/(\cos^4(\varphi) + \sin^4(\varphi))$, или $x^4 + y^4 = 8xy^2$.

10. Эпициклоида $x = (a + b)\cos(t) - a\cos((a + b)t/a)$;
$y = (a + b)\sin(t) - a\sin((a + b)t/a),\ a > 0,\ b > 0$:

а) $b/a = 3,\ t \in [0; 2\pi)$;

б) $b/a = 3/2,\ t \in [0; 4\pi)$;

в) астроида $x = b\cos^3(t),\ y = b\sin^3(t),\ t \in [0; 2\pi)$;

г) циссоида $x = at^2/(1 + t^2),\ y = at^3(1 + t^3),\ a > 0$.

11. Фигуры:

а) строфоида $y^2 = \dfrac{x(x-a)^2}{2a - x},\ a > 0,$ или $x = \dfrac{2at^2}{t^2 + 1},\ y = \dfrac{at(t^2 - 1)}{t^2 + 1},$
или $r = a(1 \pm \sin t)/\cos t$;

б) – г) конхоида Никомеда $(x^2 + y^2)(y - a)^2 - l^2y^2 = 0,\ a, l > 0,$
или $r = a/\sin\varphi + l$;

б) $l < a$;

в) $l > a$;

г) $l = a$.

12. «Розы»:

а) $r = \cos(2\varphi)$; в) $r = \cos(3\varphi)$;

б) $r = \cos(3/4\varphi)$; г) $r = \cos(\sqrt{5}\varphi)$.

13. «Розы» симметрические:

а) $r = |\cos(2\varphi)|$; в) $r = |\cos(\sqrt{3}\varphi)|$;

б) $r = |\cos(4/3\varphi)|$; г) $r = |\cos(5/4\varphi)|$.

14. Фигуры:

а) кардиоида $x = 2a\cos\varphi - a\cos 2\varphi,\ y = 2a\sin\varphi - a\sin 2\varphi,$ или $r = 2a(1-\cos(\varphi));$

б) циссоида Диоклеса: $y^2 = \dfrac{x^3}{2a-x},$ или $r = \dfrac{2a\sin^2(\varphi)}{\cos\varphi},$ или

$$x = \frac{a}{1+t^2},\ y = \frac{a}{t(1+t^2)};$$

в) волосы Аньези: $y = \dfrac{a^3}{x^2+a^2}\ (a>0),$ или $x = t,\ y = \dfrac{a^3}{t^2+a^2};$

г) спираль Ферма: $r = a\sqrt{\varphi}.$

15. Циклоиды:

а) $x = a\left(t - \lambda\sin t\right),\ y = a\left(1 - \lambda\cos t\right),\ \lambda > 1;$

б) $x = a\left(t - \lambda\sin t\right),\ y = a\left(1 - \lambda\cos t\right),\ \lambda < 1;$

в) спираль $r = a/\sqrt{\varphi};$

г) декартов лист $x^3 + y^3 = 3axy,$ или $x = 3at/(1+t^3),\ y = 3at^2/(1+t^3),$

или $r = \dfrac{3a\cos\varphi\sin\varphi}{\cos^3\varphi + \sin^3\varphi}.$

16. Трактриссы и канхоиды:

а) $x = a\ln\operatorname{tg}(t/2) + a\cos t,\ y = a\sin t;$

б) $r = c/\sin\varphi \pm l,\ -l = c;$

в) $r = c/\sin\varphi \pm l,\ -l < c;$

г) $r = c/\sin\varphi \pm l,\ -l > c.$

17. Фигуры:

а) трисектриса Маклорена: $r = a/\cos(\varphi/3);$

б) трисектриса Лоншама: $r = a/\cos(3\varphi);$

в) крестообразная кривая: $r = a/\cos(2\varphi);$

г) колосья: $r = a/\cos(k\varphi)$ или $r = a/\sin(k\varphi),\ k$ – рациональное число.

10.4. Другие возможности графики

Цвета для разных адаптеров

В CGA-адаптерах используется так называемая RGBI-система (Red, Green, Blue, Intensiv) для работы с цветом. На основе трех основных цветов, синего, зеленого и красного, путем смешивания и установки различной яркости свечения формируются четыре палитры. Если для задания цвета служат четыре бита, то три из них используются для указания цветовой составляющей: синего (0001), зеленого (0010) и красного (0100). Четвертый бит управляет яркостью свечения: 1000 – высокая интенсивность свечения, 0000 – низкая. По этому принципу формируется 16 цветов, которые поддерживает для адаптера CGA драйвер CGA.BGI. Такую таблицу цветов мы рассматривали в текстовом режиме.

Новые модели видеоадаптеров – EGA / VGA – имеют ряд конструктивных особенностей, позволяющих значительно расширить возможности работы с цветом. Для задания цвета пикселя здесь используются уже шесть битов, а не четыре, как в CGA, что значительно расширяет гамму цветов. Принцип их формирования примерно тот же, но в качестве основы используется система RrGgBb, где RGB – красный, зеленый и синий цвета нормальной яркости, а rgb – те же цвета, но яркость их в два раза меньше.

Для EGA/VGA-адаптеров драйвер EGAVGA.BGI устанавливает 64 цвета. Именованные константы для них начинаются словосочетанием EGA.

Основная палитра имеет следующие комбинации цветов.

Константа	Код цвета	Значение	Название цвета	Компоненты цвета
EGABlack	000000	0	Черный	.....
EGABlue	000001	1	Синий	.....B
EGAGreen	000010	2	Зеленый	….G.
EGACyan	000011	3	Голубой	….GB
EGARed	000100	4	Красный	…R..
EGAMagenta	000101	5	Фиолетовый	…R.B
EGABrown	010100	20	Коричневый	.g.R..
EGALightGray	000111	7	Светло-серый	…RGB
EGADarkGray	111000	56	Темно-серый	rgb…
EGALightBlue	111001	57	Светло-синий	rgb..B
EGALightGreen	111010	58	Светло-зеленый	rgb.G.
EGALightCyan	111011	59	Светло-голубой	rgb.GB
EGALightRed	111100	60	Светло-красный	rgbR..
EGALighMagentat	111101	61	Светло-фиолетовый	rgbR.B
EGAYellow	111110	62	Желтый	rgbRG.
EGAWhite	111111	63	Белый	rgbRGB

Установка цвета

Для различных типов адаптеров количество цветов, которые одновременно отображаются на экране в графическом режиме, отличается. Однако для всех BGI-драйверов она ограничена диапазоном целочисленных значений от 0 до 15. Для определения максимального номера цвета, который воспринимается данным адаптером в текущем графическом режиме, нужно использовать функцию GetMaxColor: WORD. По умолчанию для изображения используется цвет с максимальным номером (белый), а для фона – с минимальным (черный). Если в процедуре SetColor(Color) в качестве Color указан недопустимый номер, текущий цвет не меняется.

Процедура

```
SetBkColor(Color)
```

устанавливает новый цвет фона, который определяется параметром Color.

Получить значения текущих установок цвета можно с помощью двух специальных функций

GetColor: Word и GetBkColor: Word.

Установка палитры

Палитрой называется максимальный набор цветов, которые поддерживаются драйвером. Она включает 16 цветов (от 0 до 15), известных как «программные» цвета или цветовые атрибуты, используемые как в текстовом, так и в графическом режимах.

Каждому программному цвету соответствует «аппаратный» цвет из полной палитры.

В модуле предусмотрен ряд процедур, которые охватывают практически все возможные операции с цветами палитры.

Процедуры и функции для работы с палитрой

Название	Предназначение
GetDefaultPalette(Palette)	Получение информации о текущей палитре
GetPalette(Palette)	Размещает в переменную Palette информацию о текущей палитре
SetPaletteSize:Integer	Возвращает число цветов в текущей палитре
SetPalette(ColorNum,Color)	Используется для установки одного цвета палитры
SetAllPalette(Palette)	Используется для установки всех цветов палитры

Установка стиля заполнения

Для заполнения внутренних или внешних областей графических фигур в модуле Graph встроена группа наперед определенных (стандартных) комбинаций символов-заполнителей, которые можно назвать *маской*.

Маска может окрашиваться в допустимые цвета. Комбинацию цвет-маска называют *стилем заполнения*.

Стандартные стили

Константа	Значение	Маска (заполнение)
EmptyFill	0	Цвет фона
SolidFill	1	Текущий цвет
LineFill	2	Символы – горизонтальные линии
LtSlashFill	3	Символы наклонные линии // нормальной толщины
SlashFill	4	Символы // удвоенной толщины
BkSlashFill	5	Символы \\ удвоенной толщины
LtBkSlashFill	6	Символы \\ нормальной толщины
HatchFill	7	Вертикально-горизонтальная штриховка
XhatchFill	8	Штриховка крест-накрест по диагоналям редкими тонкими линиями
InterLeaveFill	9	Штриховка крест-накрест по диагоналям частыми тонкими линиями
WideDotFill	10	Заполнение «частыми» точками
CloseDotFill	11	Заполнение «редкими» точками
UserFill	12	Заполнение по определенной пользователем маске заполнения

Процедура

```
SetFillStyle(Pattern, Color)
```

устанавливает стиль Pattern и цвет Color. Значение маски Pattern приведено в предыдущей таблице и может быть задано константой или числом, Color берется из установленной палитры. Например:

```
SetFillStyle(SlashFill, Yellow);
Bar(10, 10, 50, 150);
            {столбец заполнен слешами // желтого цвета}
```

Только когда Pattern = UserFill (Pattern = 12), становится активным шаблон (маска), заданный в SetFillPattern.

Если не подходят предопределенные маски и нужно установить свою, используется процедура

```
SetFillPattern(PattMatrix, Color).
```

PattMatrix : FillPatternType – новая маска, Color – ее цвет. Маска занимает матрицу 8×8 (64 пиксела). Для ее задания используется

8 байт (64 бит), причем каждый бит «зажигает» или «гасит» соответствующий пиксель в матрице 8×8. Для этого применяют предопределенный 8-байтовый массив типа

```
type FillPatternType = array[1..8] of Byte;
```

Пример.

```
const
  MyPattern : FillPatternType=
                ($10, $38, $7C, $FE, $7C, $38, $10, $00);
```

В двоичной системе счисления этот массив имеет вид ромбика:

```
00010000
00111000
01111100
11111110
01111100
00111000
00010000
00000000
```

Необходимую информацию о нестандартных стилях можно получить при помощи процедуры

```
GetFillPattern(inf),
```

которая выгружает в массив `Var inf : FillPatternType` все содержимое маски.

Процедура

```
GetFillSetting(inf),
```

где `inf` типа `FillSettingType`, а

```
type
     FillSettingType = record
                         Pattern : Word;
                         Color : Word
                       end
```

дает информацию о текущем стиле.

Процедура

```
FloodFill(x, y, border)
```

служит для заполнения внутренней или внешней области фигур (эллипсов, окружностей, многоугольников) при помощи стиля, который был установлен процедурами `SetFillStyle` и `SetFillPattern`.

Если x, y – координаты внутри фигуры, контур которой имеет цвет border, то закрашивается внутренняя часть фигуры, а наоборот – выполняется внешнее заполнение текущим образцом зарисовки.

10.5. Построение графических фигур

Библиотека Graph содержит ряд процедур, формирующих разные фигуры на основе заданных параметров. Цвет, стиль и толщина линии для вычерчивания берутся по умолчанию или устанавливаются соответственно процедурами SetColor, SetFillPattern, SetFillStyle.

Прямоугольники

Изображение контура одномерного прямоугольника дает процедура

```
Rectangle(x1, y1, x2, y2),
```

где x1, y1 – координаты левого верхнего угла, а x2, y2 – правого нижнего. Область внутри прямоугольника не закрашивается и совпадает по цвету с фоном.

Закрашенный прямоугольник установленным наперед определенным заполнителем и цветом дает процедура

```
Bar(x1, y1, x2, y2).
```

Трехмерный закрашенный прямоугольник (параллелепипед) изображает процедура

```
Bar3D(x1, y1, x2, y2, Delph, Top).
```

Параметр Delph – это количество пикселей, которое задает глубину трехмерного контура (чаще всего Delph := (x2-x1) div 4). Параметр Top:Boolean определяет, строить над прямоугольником вершину (Top=True) или нет (Top=False).

Многоугольники

Прямоугольники, не ориентированные относительно сторон экрана, можно рисовать разными способами, например при помощи Line или LineTo. Однако следующая процедура позволяет строить любые многоугольники линией текущего цвета, стиля и толщины:

```
DrawPoly (NumPoints, PolyPoints).
```

Параметр NumPoints: Word задает количество целочисленных координат, которые попарно записываются в нетипизированном параметре переменной PolyPoints.

Замечание. Для вычерчивания замкнутого n-угольника нужно передать (n + 1) координату ((n + 1)-я совпадает с первой).

При выполнении этой процедуры будет использоваться тип PointType, который введен в модуле Graph следующим образом:

```
type
      PointType = record
                   x, y : Integer;
                  end;
```

Например,

```
const
      T : array[1..5] of PointType = ((X :  10; Y :  50),
                  (X : 100; Y :  60), (X : 100; Y : 100),
                  (X :  10; Y : 110), (X :  10; Y :  50));
           …
begin
        DrawPoly (5, T); … end.
```

Точки будут соединяться следующим образом: первая со второй, вторая с третьей и т. д. Чтобы замкнуть фигуру, задали массив из пяти элементов, где последний элемент равен первому.

Пример. В центре экрана программа чертит красными линиями треугольник.

```
program demodrawpoly;
uses crt, graph;
var
       dv, mv               : integer;
       pp                   : array [1..4] of pointtype;
       xm, ym, ym4, xm4     : word;
begin
dv      :=  detect;
initgraph(dv, mv, '');
xm      := getmaxx;
ym      := getmaxy;
xm4     := xm div 4;
ym4     := ym div 4;              {координаты вершин}
pp[1].x := xm4;
pp[1].y := ym4;
pp[2].x := xm-xm4;
pp[2].y := ym4;
pp[3].x := xm div 2;
pp[3].y := ym-ym4;
```

```
pp[4]   := pp[1];
setcolor(lightred);                {цвет для вычерчивания}
drawpoly(4, pp);
readln;
closegraph
end.
```

При рисовании этой фигуры опросили соответствующими подпрограммами максимальную разрешимость экрана и потом получили образ.

Задание. В центре экрана изобразите окружность случайного радиуса $r \in [10; 20]$ и опишите около нее правильный треугольник.

Предыдущий треугольник можно зарисовать, т. е. изменить фон внутри треугольника. Для этого вместо `DrawPoly` используют другую процедуру, но с такими же параметрами:

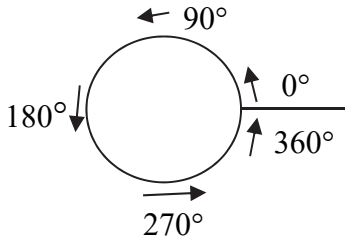
```
FillPoly (NumPoints, PolyPoints)
```

Задача. Нарисуйте четырехконечную звезду.

```
program demofillpoly;
uses
      crt, graph;
const
       star : array[1..18] of integer =        {9 пар}
              (75, 0, 100, 50, 150, 75, 100, 100, 75,
               150, 50, 100, 0, 75, 50, 50, 75, 0);
var
      dv, mv     : integer;
begin
dv           := detect;
initgraph    (dv, mv, '');
setfillstyle (1, green);
fillpoly (9, star);         {количество пересечений =8+1}
readln;
closegraph;
end.
```

Построение дуг и окружностей

При построении дуг и окружностей используется полярная система координат.



Процедура

```
Circle(x, y, R)
```

чертит окружность текущим цветом, где (x, y) – центр окружности, R – радиус (все типа Word). Например:

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

Процедура

```
Arc(x, y, St, End, R)
```

чертит дугу окружности, где St и End – начальный и конечный углы (в градусах) дуги окружности с центром в точке (x, y) и радиусом R (если St = 0, End = 360, получим полную окружность).

Информацию о координатах последнего обращения к Arc дает следующая процедура:

```
GetArcCoords(ArcCoords).
```

Здесь

```
var arccoords : arccoordstype,
arccoordstype = record
                  x, y           : integer;
                  xstart, ystart : integer;
                  xend, yend     : integer
                end;
```

где (x, y) – координаты центра; (xStart, yStart) – начальная позиция; (xEnd, yEnd) – последняя позиция последней команды Arc.

Эта информация, например, бывает нужной для построения сложных фигур.

Для построения эллиптических дуг предназначена процедура

```
Ellipse (x, y, St, End, xR, yR),
```

где x, y – координаты центра эллипса; xR, yR – длины горизонтальной и вертикальной осей; St, End – начальный и конечный углы (все типа Word).

При St = 0, End = 360 получается полный эллипс. Фон внутри эллипса не изменяется. Например,

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

Следующая процедура строит и заливает эллипс:

```
FillEllipse (x, y, xR, yR).
```

Заполнитель устанавливается процедурой SetFillStyle или SetFillPattern.

В программах деловой графики часто требуется разделить окружность или эллипс на секторы и залить их. Это делается процедурами:

```
PieSlice (x, y, St, End, R)     {сектор в окружности}
Sector (x, y, St, End, xR, yR)     {сектор в эллипсе}
```

Секторы рисуются текущим цветом, а при заливке используются тип и цвет, заданные через SetFillStyle и SetFillPattern.

Пример. В следующей программе на экране имитируется вид часового циферблата, однако ход часов выбран без связи с системными часами.

```
uses graph, crt;
var
    d, r, r1, r2, rr, c, x1, y1, x2, y2, x01, y01, k
                      : integer;
          xasp, yasp : word;
               xyasp : real;
begin
d := detect;
initgraph (d, r, '');
c := graphresult;
if c <> grok then writeln (grapherrormsg(c))
   else
   begin
   x1 := getmaxx div 2;
   y1 := getmaxy div 2;
   getaspectratio(xasp, yasp);     {подсчет радиусов}
   xyasp := yasp / xasp;
   r := round(3 * getmaxy / 8/xyasp);
   r1 := round(0.9 * r);           {часовые деления }
   r2 := round(0.95 * r);          {минутные деления}
   circle(x1, y1, r);              {циферблат       }
   circle(x1, y1, round(1.02 * r));
   for k := 0 to 59 do
      begin
```

```
        if k mod 5=0 then rr := r1 {часовые деления }
           else rr := r2;          {минутные деления}
        x01 := x1+round(rr*sin(2*pi*k/60));
        x2  := x1+round(r*sin(2*pi*k/60));
        y01 := y1-round(rr*cos(2*pi*k/60)*xyasp);
        y2  := y1-round(r*xyasp*cos(2*pi*k/60));
        line(x01, y01, x2, y2);     {деления}
       end;
    setwritemode(xorput);
    setlinestyle(solidln, 0, thickwidth);
    r := 0;         {счетчик минут в каждом часе}
    repeat
      for k := 0 to 59 do
      if not keypressed then
       begin             {координаты часовой стрелки}
        x2  :=x1+round(0.85*r1*sin(2*pi*r/60/12));
        y2  :=y1-round(0.85*r1*xyasp*cos(2*pi*r/60/12));
        x01 :=x1+round(r2*sin(2*pi*k/60));
        y01 :=y1-round(r2*xyasp*cos(2*pi*k/60));
                          {координаты минутной стрелки}
        line(x1, y1, x2,  y2);
        line(x1, y1, x01, y01);
        delay(1000);
        line(x1, y1, x2, y2);
        line(x1, y1, x01, y01);   {стерли вторым выводом}
        inc(r);
        if r=12*60 then r := 0;
       end;
    until keypressed;
    if readkey=#0 then k := ord(readkey);
                               {очистили буфер клавиатуры}
    closegraph;
    end
end.
```

10.6. Работа с текстом

Задание шрифтов

В комплект поставки Turbo Pascal включается набор штриховых шрифтов. Файлы штриховых шрифтов имеют расширение `*.CHR`. В штриховых шрифтах при построении символа используется не матричный (как в стандартных шрифтах для текстового режима), а векторный способ. Это дает

широкие возможности манипуляции размерами шрифтов без ухудшения качества их изображения. Для доступности штриховых шрифтов нужно, чтобы при инициализации графического режима были доступны файлы с расширением *.CHR, которые их содержат. По умолчанию подключается матричный (битовый) шрифт.

Имена встроенных шрифтов приведены в таблице.

Наперед определенные константы в модуле Graph	Значение	Пояснение
DefaultFont	0	8×8 стандартный битовый шрифт
TriplexFont	1	Штриховые шрифты
SmalFont	2	Малый шрифт
SansSerifFont	3	Шрифт без засечек
GothicFont	4	Готический шрифт

Все эти шрифты содержат только первую часть ASCII-таблицы (0÷127), и только DefaultFont имеет символы кириллицы (если системно подключена вторая часть таблицы с кириллицей).

Установить нужный шрифт можно процедурой

SetTextStyle(Font, Direction, CharSize).

Здесь Font : Word – номер выбранного шрифта (из таблицы), CharSize: Word – размер выводимых символов (1≤CharSize≤ 10). Если шрифт матричный (Font = DefaultFont), тогда при CharSize = 4 каждый символ, закодированный матрицей 8×8, будет выводиться на основе матрицы 32×32 пикселя. Direction : Word – направление:

Horizdir	0	Слева направо
Vertdir	1	Снизу вверх (на 90° повернутый)

Векторные шрифты в основе построения символа реализуют такую идею: в некоторой системе координат описывается последовательность прохождения контура, который образует символ, относительно предыдущей точки контура. Значит, и модификация шрифта (увеличение, расширение и т. д.) происходит простым умножением этих координат на соответствующее число.

Нужный размер шрифта можно установить процедурой

SetUserCharSize (multX, divX, multY, divY),

где отношения n = multX / divX и m = multY / divY установят ширину и высоту нового шрифта соответственно. Это означает, что, например, при матричном шрифте (8×8) каждый пиксель отображается матрицей n×m.

Функции

```
TextHeight (Textstring) : Word  и
TextWidth  (Textstring) : Word
```

дают сведения о высоте и толщине строки `Textstring` в пикселях. Эта информация нужна, чтобы рассчитать размеры текста для вывода на экран.

Текст выводится относительно текущей позиции (`x, y`). Текст можно выводить, совмещая координаты вывода с центром строки, правым или левым краем строки, а также относительно толщины строки, строку поднять вверх или опустить вниз.

Вариант ориентировки задается процедурой

```
SetTextJustify (Horizontal, Vertical),
```

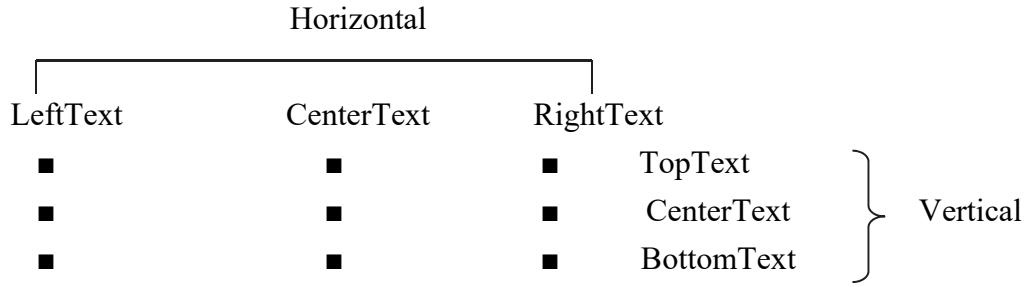
где `Horizontal : Word` содержит значения:

```
0 = LeftText;        1 = CenterText; 2 = RightText;
```

а `Vertical : Word` содержит значения:

```
0 = BottomText; 1 = CenterText; 2 = TopText;
```

На схеме знак ■ показывает координаты точки (`x, y`)



Вывод текста

Процедура

```
Outtext (TextString)
```

выводит текст `Textstring : String`, начиная с позиции `(x,y)` текущего указателя, а процедура

```
OuttextXY (x, y, Textstring)
```

выводит текст относительно заданных координат `(x,y)` установленным стилем, цветом и ориентировкой. Если текст не помещается в окно, то он отсекается (в случае шрифта `defaultfont` теряется вовсе).

Отметим, что текстовые процедуры `GotoXY`, `Write/Writeln` и установка цвета текста (`TextBackGround`, `TextColor`) в графическом режиме работают,

только если переменная Crt.DirectVideo=false (или модуль Crt не подключен). Ввод Read/Readln действует всегда, но при этом текст ввода стирает часть экрана.

Процедура GetTextSettings дает сведения о шрифте, направлении вывода, размерах символа и ориентации текущего текста.

Для вывода числовых значений их предварительно нужно перевести в строку символов процедурой Str.

Пример.

```
Max := 34.56;
Str(Max : 6 : 2, Smax);
OuttextXY (400, 40, 'max='+Smax);
```

Задача. В центре экрана вывести 8 строк разными шрифтами и обвести их рамками.

```
program demotextframe;
uses graph;
var
     driver, mode  : integer;
     st, st1       : string;
     height, width,
     cx, cy, x1, x2,
     y1, y2, i     : integer;
     tinf          : textsettingstype;
begin
     st      :=  'aabbcc…';
     driver  :=  detect;
     initgraph(driver, mode, '');
     cx      :=  getmaxx div 2;
     cy      :=  getmaxy div 2;
     settextjustify(centertext, centertext);
     for i := 1 to 8 do
       begin
          settextstyle (i, 0, 1);
          clearviewport;
          gettextsettings(tinf);
          str(tinf.font : 2, st1);
          st1    := 'font : '+st1+'  '+st;
          height := (textheight(st1)+4) div 2;
          width  := (textwidth (st1)+4) div 2;
          x1     := cx-width;
          x2     := cx+width;
```

```
            y1     := cy-height;
            y2     := cy+height;
            setcolor(15);
            rectangle(x1, y1, x2, y2);
            setcolor(11);
            outtextxy (cx, cy, st1);
            readln;
          end;
        closegraph;
    end.
```

Экран и окно

Окно – это прямоугольная область экрана, которая выполняет все функции полного экрана.

После установки окна вся остальная площадь экрана становится неактивной, и весь ввод-вывод осуществляется только через окно. В каждый момент времени может быть активным только одно окно. Если окон несколько, за переключение ввода-вывода в нужное окно отвечает программист.

Процедура

```
SetViewPort(x1, y1, x2, y2, Clip)
```

создает окно, где `x1, y1` – координаты левого верхнего угла, а `x2, y2` – правого нижнего. Параметр `Clip: Boolean` определяет, будет ли изображение отсекаться при выходе за пределы окна (`Clip := true`) или нет (`Clip := false`). Значит, создав окно, можно получить локальную систему координат и пользоваться отрицательными координатами.

Процедура

```
SetBrColor(Color)
```

задает цвет фона.

Процедура

```
ClearViewPort
```

очищает текущее окно.

Окно существует независимо от текущей страницы и не привязано к ней. По умолчанию окно занимает весь экран, его размеры задаются процедурой инициализации `InitGraph`.

После очистки окна текущий указатель устанавливается в левый верхний угол окна с координатами (0, 0) – это внутренние координаты окна.

Координатную систему полного экрана можно вернуть, установив новое окно `SetViewPort(0, 0, GetMaxX, GetMaxY, True)`, или выполнить процедуру `GraphDefauls`.

Атрибуты текущего окна можно получить при помощи процедуры

GetViewSetting(Vp).

Здесь параметр Vp описан так: var Vp : ViewPortType, а ViewPortType – следующий тип:

```
type
       ViewPortType = record
                        x1, y1, x2, y2 : Integer;
                        Clip               : Boolean;
                        end;
```

Процедура ClearViewPort заливает текущим фоном весь экран и поэтому не выделяет границ окна. «Заливки» графического окна цветом, который отличается от общего фона экрана, делают так:

- процедурой SetFillStyle устанавливается шаблон и цвет заполнения;
- процедурой Bar выводится залитый прямоугольник;
- открывается окно с теми же параметрами, что и в Bar.

Пример.

```
SetFillStyle (1, 4);      {1 – заполнение сплошным цветом,
                           4 – номер цвета Red (красный)}
Bar(100, 50, 500, 200);
SetViewPort(100, 50, 500, 200);
```

Следующая программа под случайное музыкальное сопровождение рисует разноцветные окна:

```
program demo_setviewport;
uses
      crt, graph;
var
         i      : integer;
         dv, dm : integer;
begin
         dv := detect;
         initgraph (dv, dm, '');
         setviewport (10, 10, 630, 320, true);
         i := 1;
         repeat
                  i := i+1;
                  sound (random(180)+40+i);
                  delay (random(170));
                  setfillstyle (random(4), random(16));
                  bar (10, 10, 630, 320);
```

```
                nosound;
                delay(100)
        until keypressed;
        readln;
        closegraph;
end.
```

10.7. Манипулирование фрагментами образов

Использование оперативной памяти для хранения графических образов позволяет быстро восстановить их на экране, что полезно для программирования образов, которые движутся.

Процедура

```
GetImage(x1, y1, x2, y2, BitMap)
```

сохраняет в оперативной памяти в переменной `BitMap` образ прямоугольной области графического экрана:

- `(x1, y1)` – координаты верхнего левого угла прямоугольной области;
- `(x2, y2)` – координаты нижнего правого угла.

Размер `BitMap` должен быть на 4 байта больше, чем необходимо для сохранения образа заданного объема (в этих байтах запоминаются ширина и высота прямоугольной области экрана, которая сохраняется). Переменная `BitMap` обычно располагается в `Heap`, где ей предварительно процедурой `GetMem` отводится память.

Функция

```
ImageSize(x1, y1, x2, y2) : Word
```

возвращает количество байт, необходимое для сохранения прямоугольной области графического образа экрана с координатами (`x1`, `y1`), (`x2`, `y2`). Если нужно памяти более 64 кб, то `ImageSize` = 0, а `GraphResult` = –11 и сохранять образ нужно будет по частям. Величина образа зависит от драйвера и режима, т. е. от количества битов в видеопамяти для одного пикселя.

Процедура

```
PutImage(x, y, BitMap, Mode)
```

выводит на экран сохраненный в `BitMap` образ прямоугольной области графического экрана. Место вывода задается координатами (`x`, `y`) – верхний левый угол (ширина и высота находятся в `BitMap`). Переменная `Mode` задает режим вывода на экран. Ее значение соответствует одному из вариантов следующего списка констант:

```
copyput   =0;        {mov}
normalput =0;        {mov}
```

```
xorput    =1;        {xor}
orput     =2;        {or}
andput    =3;        {and}
notput    =4;        {not}
```

Каждая из этих констант соответствует записанной в комментарии бинарной операции между байтами видеопамяти и байтами сохраненного образа. Режим XORPut обеспечивает вывод с видимостью на любом фоне. Повторный вывод с операцией xor даст предварительное значение экрана, поскольку A xor B xor B = A.

Задача. Создайте простой вариант движения закрашенного круга, который «падает» с левого верхнего угла в правый нижний.

Решение.

```
program pr_graph;
uses crt, graph;
const
      r  = 20;
      sx : word=20;
      sy : word=20;
var
      gd, gm : integer;
      p      : pointer;
      size   : word;
begin
       gd := detect;
       initgraph (gd, gm, '');
       if graphresult < > grok then halt(1);
       circle (sx, sy, r);
       floodfill(sx, sy, getcolor);
                                        {зарисовали круг}
       size := imagesize (sx-r, sy-r, sx+r, sy+r);
                          {сколько памяти надо запросить}
       getmem (p, size);
             {выделили память для хранения образа в heap}
       getimage (sx-r, sy-r, sx+r, sy+r, p^);
                                {записали в буфер в heap}
       repeat
           delay(300);
           putimage (sx-r, sy-r, p^, xorput);
                     {стираем изображение на старом месте}
           inc (sx, 6);
           inc (sy, 3);
```

```
            putimage (sx-r, sy-r, p^, xorput);
                    {пересылаем изображение на новое место}
        until (sx > getmaxx-2*r) or (sy > getmaxy-2*r);
        readln;
        closegraph;
        freemem (p, size);
end.
```

10.8. Анимация

Первый вариант. Если драйвер обеспечивает работу со страницами графических образов, тогда можно чередовать страницы на экране и получать анимационные фрагменты.

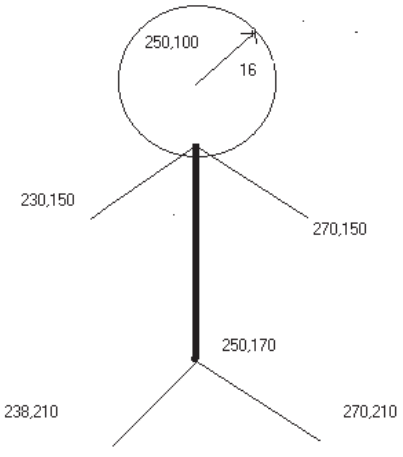
Второй вариант. Выводим рисунок, потом выводим его же цветом, совпадающим с цветом фона, – изображение исчезает, затем таким же методом выводим его в другом месте. Можно при выводе рисунка использовать операцию `XOR`. Такой вариант подходит для небольших изображений на одноцветном фоне.

Третий вариант. Сначала выводим изображение, очищаем экран (`ClearViewPort`). Выводим новый образ, очищаем экран и т. д. Такой метод не дает качественных рисунков.

Четвертый вариант. Выводим рисунок, процедурами `PutImage` и `GetImage` какую-то его часть храним в динамической памяти и нужное количество раз восстанавливаем на новых местах.

Простая анимация

Рассмотрим пример простой анимации движения конечностей мультяшного человечка. В системе экранных координат рассчитаем координаты образа и получим, например, следующую схему для анимации.



Соответственно схеме напишем программу, уточнив некоторые детали.

```
program demo_simple_anim;
uses
         crt, graph;
const
       ver : array [1 .. 3] of integer=(150, 112, 76);
       hor : array [1 .. 3] of integer=(230, 200, 191);
var       drivervar,  modevar,  i : integer;
begin
          drivervar  :=  detect;
          initgraph (drivervar, modevar, '');
          while not keypressed do
           begin
               setcolor(15);
               circle(250, 100, 16);          {     голова}
               line(250, 117, 250, 170);      {   туловище}
               line(250, 170, 238, 210);      {левая  нога}
               line(250, 170, 270, 210);      {правая нога}
               for i := 1 to 3 do
                begin
                  setcolor(15);
                  line(250, 120, hor[i], ver[i]);
                  line(250, 120, 250+(250-hor[i]), ver[i]);
                  delay(1000);
                  setcolor(0);
                  line(250, 120, hor[i], ver[i]);
                  line(250, 120, 250+(250-hor[i]), ver[i]);
                 end;
           end;
closegraph;
end.
```

Задание. Человечек должен подкидывать ногами мяч, который движется по дуге окружности или синусоиды.

Задачи для самопроверки

1. Какой результат вы получите, выполнив программу?

```
uses crt;
type
     direction = (right,down,left,up);
var
```

```
     c,x,y,h,k : integer;
procedure go(d : direction; n : integer);
begin
 for k:=1 to n do
 begin
  case d of
   right   : x:=x+2;
   down    : y:=y+1;
   left    : x:=x-2;
   up      : y:=y-1;
  end;
  c:=random(15)+1;
  textcolor(c);
  gotoxy(x,y);
  write(‘ *‘);
  delay(30);
 end;
end;
begin
 clrscr;
 x:=39;
 y:=13;
 h:=1;
 gotoxy(x,y);
 write(‘ *‘);
 repeat
  go(right,h);
  go(down,h);
  h:=h+1;
  go(left,h);
  go(up,h);
  h:=h+1;
 until h>24;
end.
```

Варианты ответа

А. Множество разноцветных прямоугольников из звездочек.
Б. Множество разноцветных окружностей из звездочек.
В. Множество звездочек разного цвета.
Г. Спираль из разноцветных звездочек.

2. Выберите оптимальный вариант программы, закрашивающей прямоугольник.

Варианты ответа

А.
```
uses graph;
   var d, m : integer;
   begin
      d:=detect;
      initgraph(d, m, '');
      setfillstyle(solidfill, red);
      rectangle(0, 0, 100, 100);
      closegraph;
   end.
```

Б.
```
uses graph;
var d, m : integer;
begin
  d:=detect;
  initgraph(d, m, '');
  setcolor(white);
  setfillstyle(solidfill, red);
  rectangle(0, 0, 100, 100);
  floodfill(1, 1, white);
  closegraph;
end.
```

В.
```
uses graph;
var
  d, m : integer;
begin
  d:=detect;
  initgraph(d, m, '');
  setfillstyle(solidfill, red);
  bar(0, 0, 100, 100);
  closegraph;
end.
```

Г.
```
uses graph;
var
    i, d, m : integer;
begin
   d:=detect;
   initgraph(d, m, '');
   setcolor(red);
```

```
    for i:=0 to 99 do
       line(0, i, 100, i)
    closegraph;
end.
```

Задачи для программирования

1. Напишите процедуру, которая строит двумерную систему координат с осями x и y.

Входными данными являются:

• XB и YB – точка начала системы координат;
• XS – размер шага по оси x;
• XV – первое значение градации по оси x;
• CX – количество шагов по оси x;
• YS – размер шага по оси y;
• YV – первое значение градации по оси y;
• CY – количество шагов по оси y;
• SX – строка названия измерения по оси x;
• SY – строка названия измерения по оси y.

2. Напишите процедуру, которая строит линейный график в двумерной системе координат.

Входными сведениями являются:

• XB, YB, XS, XV, CX, YS, YV, CY, SX, SY – параметры из предыдущей задачи;
• Cline – атрибут цвета для вывода осей;
• Cname – атрибут цвета для вывода названий;
• ArrY – массив значений по оси y для ключевых точек в оси x размера CX;
• Cchart – атрибут цвета для линейного графика.

3. Напишите процедуру, которая строит два линейных графика в одной системе координат.

4. Напечатайте двумерную или трехмерную столбиковую диаграмму, которая соответствует следующим входным данным:

• XB, YB, XS, XV, CX, YS, YV, CY, SX, SY, Cline, Cname – параметры из задачи 2;
• Wx – ширина столбика по оси x;

• `ArrY` – массив значений по оси `y` для ключевых точек в оси `x` размера `CX`;

• `Pcol` – код заполнителя для столбика;

• `Ccol` – атрибут цвета для столбика.

5. В одной системе координат постройте две двумерные (трехмерные) столбиковые диаграммы.

6. Напишите процедуру, которая строит секторную диаграмму. Входными сведениями для нее являются:

• `Xp, Yp` – координаты точки секторной диаграммы;

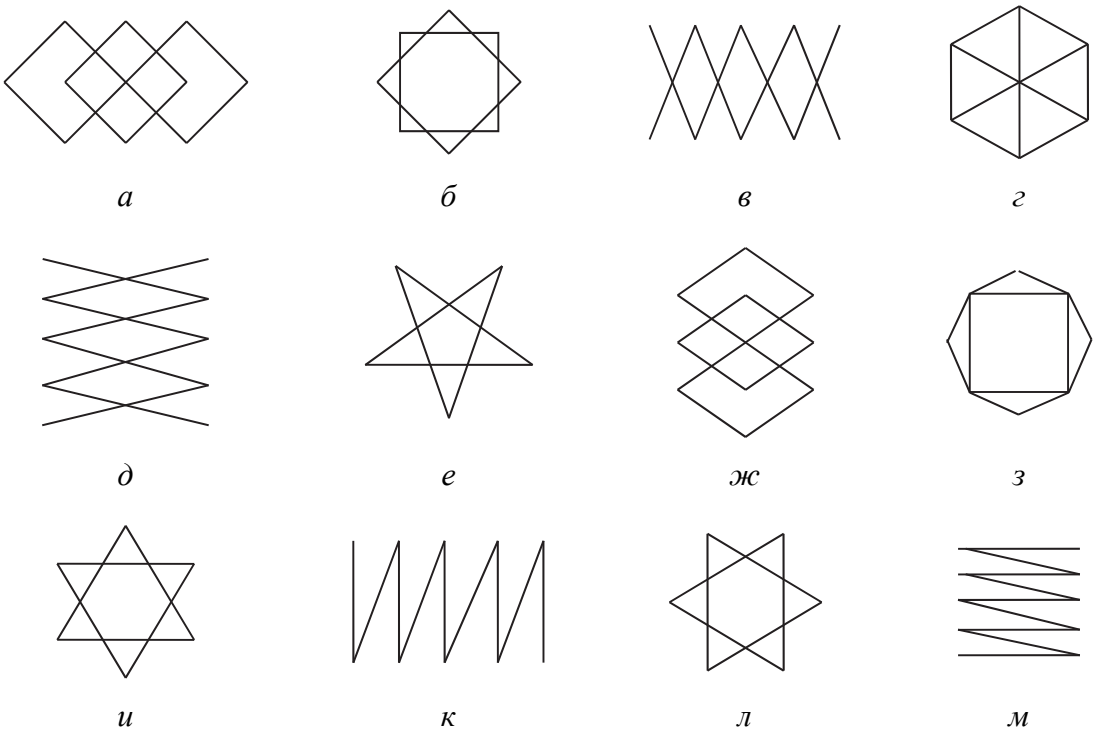
• `Radius` – радиус сектора;

• `Cp` – количество секторов или компонент;

• `P` – массив значений процентного содержания компонент размера `Cp`;

• `Namp` – массив названий компонент размера `Cp`.

7. Составьте процедуры рисования следующих фигур:

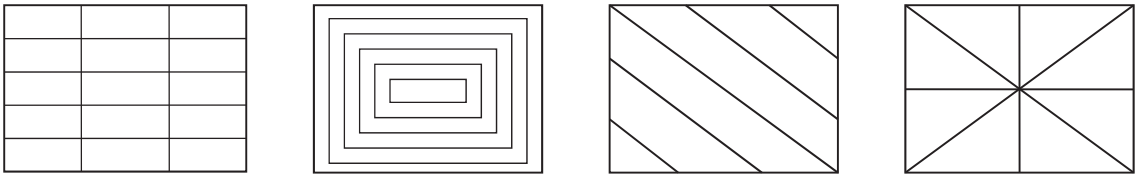


а *б* *в* *г*

д *е* *ж* *з*

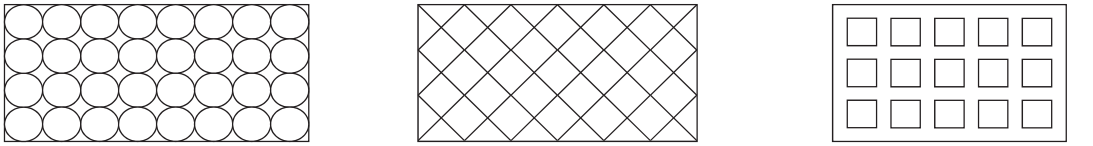
и *к* *л* *м*

8. Разделите экран в соответствии со следующими рисунками и выполните заливку каждой замкнутой области разными цветами.

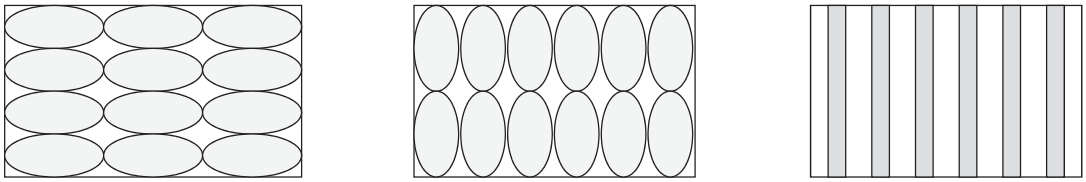


а *б* *в* *г*

9. Составьте процедуру заполнения прямоугольной области экрана орнаментом. В параметры процедуры включите координаты области и размер элемента орнамента.

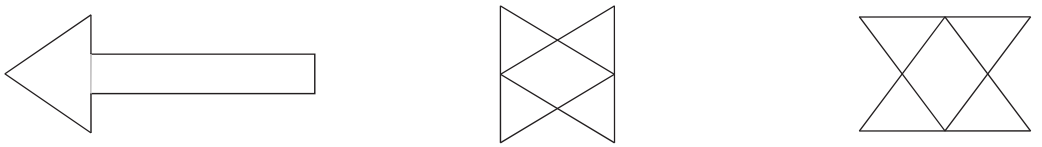


а *б* *в*

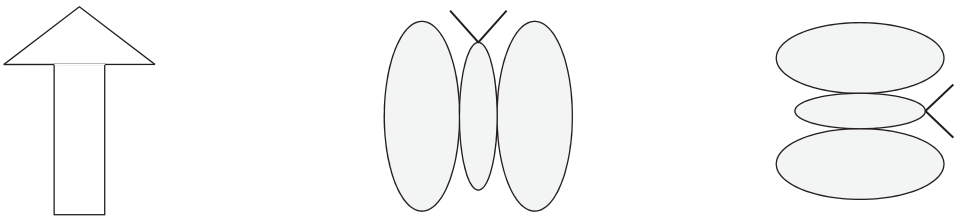


г *д* *е*

10. Составьте процедуру рисования фигуры. В параметры процедуры включите координаты узловой точки, размер и цвет фигуры. Используя полученную процедуру, заполните экран, выведя случайно *n* – const фигур.



а *б* *в*



г *д* *е*

СПИСОК ЛИТЕРАТУРЫ

Аляев, Ю. А. Практикум по алгоритмизации и программированию на языке Pascal : учеб. пособие / Ю. А. Аляев, В. П. Гладков, О. А. Козлов. – М. : Финансы и статистика, 2004.

Долинский, *М. С*. Алгоритмизация и программирование на Turbo Pascal: от простых до олимпиадных задач / М. С. Долинский. – СПб. : Питер, 2005.

Расолько, *Г. А*. Теория и практика программирования на Pascal / Г. А. Расолько, Ю. А. Кремень. – Минск : Выш. шк., 2015.

Расолько, *Г. А*. Теория и практика программирования на языке Pascal / Г. А. Расолько, Ю. А. Кремень. – Минск : Выш. шк., 2022.