ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ОБРАБОТКИ ДАННЫХ

- *Основные принципы построения программ*
- *Модульное и объектно-ориентированное программирование*
- *Алгоритмы компьютерной обработки данных*
- *Усложненные структуры данных*
- *Примеры и упражнения*

$$h_i(k) = h_0(k) + ci + di^2$$

repeat

procedure

УЧЕБНОЕ ПОСОБИЕ

bhv

ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ОБРАБОТКИ ДАННЫХ

Допущено учебно-методическим объединением на базе Санкт-Петербургского государственного университета Министерства образования Российской Федерации в качестве учебного пособия по специальности "Математическое обеспечение и администрирование информационных систем" — 351500

Санкт-Петербург

«БХВ-Петербург»

2003

УДК 681.3.06
ББК 32.973.26
У75

Ускова О. Ф. и др.

У75 Программирование алгоритмов обработки данных / Авторы: Ускова О. Ф., Огаркова Н. В., Воронина И. Е., Бакланов М. В., Мельников В. М. / — СПб.: БХВ-Петербург, 2003. — 192 с.: ил.

ISBN 5-94157-391-X

Учебное пособие для тех, кто уже приобрел начальные навыки программирования. В качестве базового используется язык Turbo Pascal. Объясняются понятия модульного и объектно-ориентированного программирования, дается представление о различных видах программ, в т. ч. рекурсивных, с возвратами. Рассматривается большое количество алгоритмов сортировки, таких как внутренние — методом подсчета, вставками, методом Шелла, быстрая, методом "пузырька", выбором и пр., и внешние — с помощью слияния, многофазная, каскадная. Приводятся также алгоритмы доступа к данным и выполняется их анализ. Введенные понятия иллюстрируются на примерах программ. Книга содержит большое количество задач и упражнений для самостоятельной работы.

Для программистов

УДК 681.3.06
ББК 32.973.26

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. гл. редактора	*Людмила Еремеевская*
Зав. редакцией	*Григорий Добин*
Редактор	*Дмитрий Фунт*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 28.08.03.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 15,5.
Тираж 3 000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-391-X

Содержание

Введение

В последнее время на книжном рынке появилось большое количество изданий, посвященных популярным языкам и системам программирования. Однако далеко не каждое из этих изданий может быть рекомендовано в качестве учебного пособия для профессионального обучения программированию, главным образом по одной причине: эти книги описывают конкретные особенности того или иного языка, а не основополагающие принципы программирования.

Известно, что на рынке компьютерных приложений и программного обеспечения наибольшим спросом пользуются большие программы и программные системы со сложными данными. Систематический и научный подход к построению таких программ очень важен, поскольку программисты могут избежать большого количества ошибок, если со знанием дела будут применять те или иные методы программирования. Практика показывает, что невозможно реализовать все эти методы без знаний о различных вариантах структурирования данных. Н. Вирт в книге "Алгоритмы + структуры данных = программы" пишет: "Решения о структурировании данных нельзя принимать без знания алгоритмов, применяемых к этим данным, и наоборот, структура и выбор алгоритмов существенным образом зависят от структуры данных". В данном учебном пособии авторы попытались изложить классические алгоритмы и методы программирования, показывая на примерах, какая структура данных может быть использована в том или ином случае. Книга подготовлена на основе многолетнего опыта преподавания основных и специализированных дисциплин компьютерного цикла на факультете прикладной математики, информатики и механики Воронежского государственного университета.

Настоящее издание является логическим продолжением учебного пособия тех же авторов "Программирование на языке ПАСКАЛЬ: задачник", которое вышло в издательстве "Питер" в 2002 году и имеет гриф Министерства образования Российской Федерации.

Книга "Программирование алгоритмов обработки данных" предназначена для студентов вузов, углубленно изучающих информатику, преподавателей информатики, а также специалистов в области информационных технологий.

Книга состоит из двух частей. В *первой части* рассматриваются технологии реализации алгоритмов компьютерной обработки информации: модули и объекты.

Во *второй части* представлены наиболее употребительные алгоритмы компьютерной обработки данных: рекурсивные алгоритмы, задачи поиска (алгоритмы с возвратом, хеширование), алгоритмы внутренних и внешних сортировок, алгоритмы работы с сильно ветвящимися деревьями.

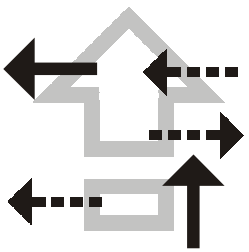
В *приложениях* приведены стандартные процедуры, необходимые для понимания многочисленных примеров программ, рассмотренных в книге.

Каждая глава начинается с краткого изложения теоретического материала, которое носит справочный характер. Последний раздел каждой главы содержит набор задач для самостоятельного решения. Практически все главы содержат примеры больших программ, которые служат хорошей иллюстрацией теоретического материала. В некоторых примерах не только приведен текст программы с комментариями, но и отражены этапы ее проектирования, развития и реализации. Авторы надеются, что их книга поможет в становлении математика-программиста и будет способствовать повышению культуры мышления. Книга предназначена для овладения компьютерными методами обработки информации путем развития профессиональных навыков разработки, выбора и преобразования алгоритмов, что является важной составляющей эффективной реализации программного продукта.

Представленный в книге материал удовлетворяет требованиям основной образовательной программы подготовки специалистов в области прикладной математики и компьютерных наук и поддерживает вузовские курсы, в которых изучаются структуры и алгоритмы обработки данных.

Авторы благодарны студентам и выпускникам факультета прикладной математики, информатики и механики Воронежского государственного университета, которые были самыми пристрастными и внимательными ценителями и судьями. Своими вопросами и замечаниями они помогли исправить шероховатости изложения материала и способствовали совершенствованию методики подачи материала.

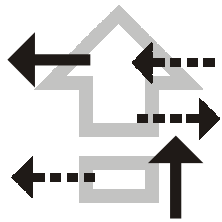
Коллектив авторов будет признателен за любые замечания, предложения, пожелания, направленные по адресу: **voronina@amm.vsu.ru** или **nataly@amm.vsu.ru**, 396006 г. Воронеж, Университетская пл., 1, факультет ПММ, тел.: (0732) 789-698.



ЧАСТЬ I

ТЕХНОЛОГИИ РЕАЛИЗАЦИИ АЛГОРИТМОВ

Глава 1



Модульный подход в программировании

Модуль (`unit`) — это именованная совокупность констант, типов, переменных, процедур и функций. В отличие от подпрограммы (процедуры или функции) модуль хранится отдельно от основной программы и компилируется независимо от нее. Сам по себе модуль не является выполняемой программой — его объекты используются другими программными единицами.

Стандартные модули

Turbo Pascal имеет восемь стандартных модулей, в которых содержатся все системные процедуры и функции (`SYSTEM`, `DOS`, `CRT`, `PRINTER`, `OVERLAY`, `GRAPH`, `GRAPH3`, `TURBO3`).

Все перечисленные стандартные модули хранятся в системном файле Turbo.tpl. Кроме того, каждый стандартный модуль находится в одноименном TPU-файле — в системном каталоге Turbo Pascal. Имена всех используемых в программе модулей, кроме `SYSTEM`, необходимо указать в спецификации использования по обычным правилам. Модуль `SYSTEM` можно не указывать в спецификации использования, так как все его ресурсы подключаются автоматически к любой программе. Рассмотрим стандартные модули подробнее:

- в модуль `SYSTEM` входят все процедуры и функции стандартного языка и подпрограммы, ориентированные на конкретную операционную среду;
- модуль `DOS` содержит средства доступа к операционной системе и является программным представлением системного интерфейса MS DOS;
- модуль `CRT` обеспечивает возможность доступа к экрану дисплея в текстовом режиме. В этом модуле сосредоточены средства чтения информации с клавиатуры, включая расширенные коды клавиш, и простейшего управления звуком;
- модуль `PRINTER` имеет единственный интерфейсный элемент *Lst* типа *text*, системно связанный с логическим устройством PRN, то есть с печатающим устройством, если оно есть в конфигурации;

- модуль OVERLAY предоставляет средства для организации оверлейных программ больших размеров, превышающих объем доступной оперативной памяти;
- модуль GRAPH объединяет многочисленные программные средства управления графическим режимом работы дисплея, обеспечивает использование всех возможностей наиболее распространенных типов дисплеев CGA, EGA, VGA, HERCULES;
- модули TURBO3, GRAPH3 обеспечивают совместимость данной версии с более ранними версиями.

Общая структура модуля

Структурно модуль имеет заголовок и разделен на две части: интерфейс (Interface) и реализация (Implementation). Объявления, указанные в интерфейсе, являются доступными для любой программы, использующей этот модуль. Поэтому интерфейс называется открытой частью. В части реализации помещаются невидимые или скрытые объекты. Модуль в языке Turbo Pascal имеет следующий вид:

```
Unit <имя>;

  Interface {открытая часть (интерфейс)}
  Uses <список имен модулей>;
    {общие объявления}

  Implementation  {скрытая часть (реализация)}
    Uses <список имен модулей>;
    {локальные объявления}
    {процедуры и функции}

begin
  {операторы инициализации}
end.
```

Список модулей интерфейсной части содержит имена модулей, используемых в данном модуле или в вызывающей программе. Вызывающая программа может использовать эти модули, обратившись к ним с помощью **Uses** <имя>, так же, как если бы они были описаны в самой программе. Все вызванные величины являются для вызывающей программы глобальными.

Список модулей в разделе реализации делает доступными соответствующие модули только в разделе реализации данного модуля. Раздел **Implementation**

может содержать инициализирующую часть. Она, в частности, предназначена для установки начальных значений переменным модуля. В случае отсутствия операторов эта часть состоит из завершающего модуль **end**.

Таким образом, модуль состоит из заголовка и трех основных частей, любая из которых может быть пустой.

Компиляция модулей

Как и программа, модуль помещается в отдельный файл и компилируется. Имя файла, содержащего исходный текст модуля, должно совпадать с именем этого модуля. Компилятор после трансляции помещает код модуля в файл с таким же именем и расширением tpu (от английского Turbo Pascal Unit). Имя модуля, как и имя файла, должно содержать не более восьми символов. В одном файле может размещаться только один модуль. Если необходимо хранить код модуля в файле с другим именем, то можно использовать директиву `$U` для переопределения имени файла. Эта директива имеет один параметр, который трактуется как настоящее имя файла с данным модулем. Она должна находиться перед первым использованием модуля в спецификации использования. Например, конструкция

```
Uses {$U Nain} ABC;
```

приведет к тому, что компилятор будет искать код модуля ABC в дисковом файле Nain.tpu.

В среде Турбо Паскаля определены три режима компиляции:

- COMPILE;
- MAKE;
- BUILD.

При компиляции модуля или основной программы в режиме COMPILE все упоминающиеся в предложении `Uses` модули должны быть предварительно откомпилированы и результаты компиляции помещены в одноименные файлы с расширением tpu. Например, если в программе (модуле) имеется предложение

```
Uses Global;
```

то на диске в каталоге, объявленном опцией OPTIONS\DIRECTORIES уже должен находиться файл Global.tpu. Файл с расширением tpu создается автоматически в результате компиляции модуля. Так, если основная программа может компилироваться без создания исполняемого EXE-файла, то компиляция модуля всегда приводит к созданию TPU-файла.

В режиме MAKE компилятор проверяет наличие TPU-файлов для каждого объявленного модуля. Если какой-нибудь из файлов не обнаружен, система пытается отыскать и скомпилировать одноименный файл с расширением

pas, то есть файл с исходным текстом модуля. Кроме того, в этом режиме система следит за возможными изменениями исходного текста любого используемого модуля. Если в PAS-файлы внесены изменения, то система всегда осуществляет их компиляцию перед компиляцией основной программы. Более того, если изменения внесены в интерфейсную часть модуля, то будут перекомпилированы также и все другие модули, обращающиеся к нему.

В режиме BUILD существующие TPU-файлы игнорируются и система отыскивает и компилирует соответствующий PAS-файл для каждого объявленного в предложении `Uses` модуля.

Пример 1. Операции с комплексными числами

Создать модуль, реализующий средства работы с комплексными числами (листинг 1.1).

Листинг 1.1. Модуль, реализующий средства работы с комплексными числами

```
Unit Cmplx;
Interface
Type {Описание комплексного числа}
     Complex = record
                 re, im : real
               end;
  {Процедура ввода комплексного числа z}
  Procedure ReadC  (var z: Complex);
  {Процедура сложения двух комплексных чисел x и y}
  {Результат запоминается в переменной z}
  Procedure AddC   (x, y: Complex; var z: Complex);
  {Процедура вычитания двух комплексных чисел x и y}
  {Результат запоминается в переменной z}
  Procedure SubC   (x, y: Complex; var z: Complex);
  {Процедура умножения двух комплексных чисел x и y}
  {Результат запоминается в переменной z}
  Procedure MulC   (x, y: Complex; var z: Complex);
  {Процедура деления двух комплексных чисел x и y}
  {Результат запоминается в переменной z}
  Procedure DivC   (x, y: Complex; var z: Complex);
```

```
  {Функция сравнения двух комплексных чисел x и y}
  {Если числа равны, то функция возвращает}
  {значение true, иначе — значение false}
  Function EqvC (x, y: Complex): boolean;
  {Процедура вывода комплексного числа z}
  Procedure WriteC (z: Complex);

Const C: Complex = (re: 0.1; im: -2);

Implementation

  Procedure ReadC;
  begin
    with z do readln (re, im)
  end;

  Procedure AddC;
  begin
    z.re := x.re + y.re;
    z.im := y.im + x.im
  end;

  Procedure SubC;
  begin
    z.re := x.re - y.re;
    z.im := x.im - y.im
  end;

  Procedure MulC;
  begin
    z.re := x.re * y.re - x.im * y.im;
    z.im := x.re * y.im + x.im * y.re
  end;

  Procedure DivC;
  Var v: real;
  begin
    v := sqr(y.re) + sqr(y.im);
```

```
    if v=0 then
      begin
        writeln ('Ошибка! Деление на ноль'); readln; halt(0)
      end
    else
      begin
        z.re := (x.re * y.re + x.im * y.im) / v;
        z.im := (x.re * y.im - x.im * y.re) / v
      end;
  end;

  Function EqvC;
  begin
    EqvC := (x.re = y.re) and (x.im = y.im)
  end;

  Procedure WriteC;
  begin
    with z do
      if (re=0) and (im=0) then write('0')
      else
        begin
          if re<>0 then write (re:2:2);
          if im<>0 then
            begin
              if (im>0) and (re<>0) then write('+'); write(im:2:2,'i')
            end;
        end;
    writeln;
  end;
end. {конец модуля Cmplx}
```

Текст этого модуля необходимо поместить в файл Cmplx.pas. После того как он будет откомпилирован, вызывающей программе станут доступны процедуры из новой библиотеки. Например, в следующей программе (листинг 1.2) проверяется, совпадает ли результат арифметических операций сложения, вычитания, умножения над парой комплексных чисел с заданным комплексным числом.

Листинг 1.2. Пример программы, использующей модуль с комплексными числами

```
Program ComplexValue;
Uses Cmplx, Crt;
Var  a, b, c, d : Complex;
begin
  ClrScr;
  writeln('Введите действительную и мнимую части первого числа');
  ReadC(a);
  write('Первое число ');
  WriteC(a);

  writeln ('Введите действительную и мнимую части второго числа');
  ReadC(b);
  write('Второе число ');
  WriteC(b);

  writeln('Введите действительную и мнимую части третьего числа');
  ReadC(c);
  write('Третье число ');
  WriteC(c);

  AddC(a, b, d);
  write('Сумма двух первых чисел равна ');
  WriteC(d);
  If EqvC(c,d) then
     writeln('она совпадает с третьим числом')
  else
     writeln('она не совпадает с третьим числом');

  SubC(a, b, d);
  write('Разность двух первых чисел равна ');
  WriteC(d);
  If EqvC(c,d) then
     writeln('она совпадает с третьим числом')
  else
     writeln('она не совпадает с третьим числом');

  MulC(a, b, d);
```

```
  write('Произведение двух первых чисел равно ');
  WriteC(d);
  If EqvC(c,d) then
     writeln('оно совпадает с третьим числом')
  else
     writeln('оно не совпадает с третьим числом');
  readln;
End.
```

После объявления **Uses Cmplx** программе стали доступны все объекты, объявленные в интерфейсной части модуля **Cmplx**. При необходимости можно переопределить любой из этих объектов, как это произошло с объявленной в модуле типизированной константой *C*. Переопределение объекта означает, что вновь объявленный объект закрывает ранее определенный в модуле одноименный объект. Чтобы получить доступ к закрытому объекту, нужно воспользоваться составным именем: перед именем объекта поставить имя модуля и точку. Например, оператор

```
Writeln(cmplx.c.re:5:1, cmplx.c.im:5:1, 'i');
```

выведет на экран содержимое закрытой типизированной константы из приведенного примера.

Пример 2. Операции над матрицами

Задача 1

Вычислить выражение вида $f(X) = 2*X^3 - 5*X^2 + 3*X - 2*I$, где X — вводимая пользователем матрица размерности 3×3. Решение задачи дано в листинге 1.4.

Задача 2

Вычислить значение выражения $(A + B^T) * B$, где A — матрица размерности 2×3, B — матрица размерности 3×2, B^T — транспонированная матрица B. Решение задачи дано в листинге 1.5.

Решение данных задач базируется на общем модуле, в котором реализованы операции над матрицами (листинг 1.3).

Листинг 1.3. Реализация процедур для работы с матрицей

```
unit Matrix;
interface
const
```

```
  MDim  = 10;   {Максимальная размерность матрицы}
type
  TElem = Real {Integer};          {Тип элементов матрицы}
  Dimension = 1..MDim;             {Интервал индексов матрицы}
  TL = array [Dimension] of TElem; {строка матрицы}
  TM = array [Dimension] of TL;    {матрица}

  TMatrix = record
    {Действительные значения размерностей матрицы}
    Dim1, Dim2: Dimension;
    {Матрица}
    M: TM
  end;

  {Возвращает в переменной Result единичную матрицу}
  {размерности (Dim x Dim)}
  procedure MatrixI( Dim: Dimension; var Result: TMatrix );
  {Транспонирование матрицы A}
  procedure MatrixTranspose( A: TMatrix; var Result: TMatrix );
  {Ввод матрицы}
  {Размерности Dim1,Dim2 определяются при вызове процедуры}
  {( Dim1>0, Dim2>0) или вводятся пользователем}
  procedure MatrixInput(var Result: TMatrix; Dim1, Dim2: Integer);
  {Вывод матрицы}
  procedure MatrixOutput( A: TMatrix );
  {Сложение матриц. Матрицы должны быть одинаковой размерности}
  procedure MatrixSum( A,B: TMatrix; var Result: TMatrix );
  {Умножение матрицы A на число N}
  procedure MatrixNumberMul(A:TMatrix; N:TElem; var Result:TMatrix);
  {Вычитание матриц. Матрицы должны быть одинаковой размерности}
  procedure MatrixSub( A,B: TMatrix; var Result: TMatrix );
  {Умножение матриц. Result(i x k) = СУММА_ПО_j A(i x j)*B(j x k)}
  procedure MatrixMul( A, B: TMatrix; var Result: TMatrix );
  {Сравнение матриц на равенство}
  function MatrixEqual( A, B: TMatrix ): Boolean;
  {Вычисление нормы матрицы: ||A|| = MAX_ПО_i_j |Aij|}
  function MatrixNorm( A: TMatrix ): TElem;
  {Возведение матрицы A в степень P}
```

```
  {Матрица A должна быть размерности (N x N)}
  procedure MatrixPower(A: TMatrix; P:Integer; var Result:TMatrix);

implementation

{Устанавливает размерность матрицы}
procedure SetDimension(var A: TMatrix; Dim1, Dim2: Dimension);
begin
  A.Dim1:=Dim1;
  A.Dim2:=Dim2
end;

{Возвращает в переменной Result единичную матрицу размерности
(Dim x Dim)}
procedure MatrixI( Dim: Dimension; var Result: TMatrix );
var k, j: Dimension;
begin
  for k:=1 to Dim do
    begin
      Result.M[k,k]:=1;          {на главной диагонали - единицы}
      {остальные элементы равны нулю}
      for j:=k+1 to Dim do
        begin
          Result.M[k,j]:=0;
          Result.M[j,k]:=0;
        end;
    end;
  SetDimension(Result, Dim, Dim)
end;

{Транспонирование матрицы A}
procedure MatrixTranspose( A: TMatrix; var Result: TMatrix );
var i, j: Dimension;
begin
  for i:=1 to A.Dim1 do
    for j:=1 to A.Dim2 do
      Result.M[j,i]:=A.M[i,j]; {внимательно: (i,j) -> (j,i)}
  SetDimension (Result, A.Dim2, A.Dim1)
```

```
end;

{Ввод матрицы}
{Размерности Dim1,Dim2 определяются при вызове процедуры}
{(Dim1>0, Dim2>0) или вводятся пользователем}
procedure MatrixInput( var Result: TMatrix; Dim1, Dim2: Integer );
var i, j: Dimension;
begin
  WriteLn('< Ввод матрицы >');

  if (Dim1<=0) or (Dim1>MDim) or (Dim2<=0) or (Dim2>MDim) then
  begin
    {ввод первой размерности}
    repeat
      Write('Введите первую размерность матрицы: ');
      ReadLn(Dim1)
    until (Dim1>0) and (Dim1<=MDim);

    {ввод второй размерности}
    repeat
      Write ('Введите вторую размерность матрицы: ');
      ReadLn (Dim2);
    until (Dim2>0) and (Dim2<=MDim)
  end;

  {сохраняем введенные размерности}
  SetDimension (Result, Dim1, Dim2);

  WriteLn ('Введите матрицу размерности ', Dim1, 'x', Dim2, ':');
  for i:=1 to Result.Dim1 do
  begin
    for j:=1 to Result.Dim2 do
      Read (Result.M[i,j]);
    ReadLn  {матрица вводится построчно}
  end
end;

{Вывод матрицы}
```

```
procedure MatrixOutput( A: TMatrix );
var i, j: Dimension;
begin
  for i:=1 to A.Dim1 do
  begin
    for j:=1 to A.Dim2 do
      Write(A.M[i,j]:8:2,' ');
    WriteLn
  end
end;

{Сложение матриц. Матрицы должны быть одинаковой размерности}
procedure MatrixSum( A, B: TMatrix; var Result: TMatrix );
var i, j: Dimension;
begin
  for i:=1 to A.Dim1 do
    for j:=1 to A.Dim2 do
      Result.M[i,j] := A.M[i,j] + B.M[i,j];
  SetDimension (Result, A.Dim1, A.Dim2)
end;

{Умножение матрицы A на число N}
procedure MatrixNumberMul( A: TMatrix; N: TElem; var Result:TMatrix);
var i,j: Dimension;
begin
  for i:=1 to A.Dim1 do
    for j:=1 to A.Dim2 do
      Result.M[i,j] := A.M[i,j] * N; {умножаем каждый элемент}
  SetDimension( Result, A.Dim1, A.Dim2 )
end;

{Вычитание матриц. Матрицы должны быть одинаковой размерности}
{Замечание. Данную операцию можно реализовать также через}
{умножение матрицы B на -1 и сложение с матрицей A}
procedure MatrixSub( A, B: TMatrix; var Result: TMatrix );
var i,j: Dimension;
begin
  for i:=1 to A.Dim1 do
```

```
    for j:=1 to A.Dim2 do
      Result.M[i,j] := A.M[i,j] - B.M[i,j];
  SetDimension (Result, A.Dim1, A.Dim2)
end;

{Умножение матриц. Result(i x k) = СУММА_ПО_j A(i x j)*B(j x k)}
procedure MatrixMul( A, B: TMatrix; var Result: TMatrix );
var i,j,k: Dimension;
    s: TElem;
begin
  SetDimension (Result, A.Dim1, B.Dim2);
  for i:=1 to Result.Dim1 do
    for k:=1 to Result.Dim2 do
    begin
      s := 0;
      for j:=1 to A.Dim2 do
        s := s + A.M[i,j] * B.M[j,k];
      Result.M[i,k] := s
    end
end;

{Сравнение матриц на равенство}
function MatrixEqual( A, B: TMatrix ): Boolean;
var i,j: Integer;
begin
  MatrixEqual := False;
  {размерности у равных матриц должны совпадать}
  if (A.Dim1=B.Dim1) and (A.Dim2=B.Dim2) then
    begin
      i:=0;
      repeat
        i:=i+1;
        j:=0;
        repeat
          j:=j+1
        until (j=A.Dim2) or (A.M[i,j]<>B.M[i,j])
      until (i=A.Dim1) or (A.M[i,j]<>B.M[i,j]);
      MatrixEqual := (A.M[i,j]=B.M[i,j]);
```

```
    end
end;

{Поиск максимума в строке}
function LineMax( Vector: TL; Dim: Dimension ): TElem;
var i : Dimension;
    max: TElem;
begin
  max := Vector[1]; {начальное значение}
  for i:=2 to Dim do
    if max < Vector[i] then max := Vector[i];
  LineMax := max
end;

{Вычисление нормы матрицы: ||A|| = MAX_ПО_i_j |Aij|}
function MatrixNorm( A: TMatrix ): TElem;
var i  : Dimension;
    max, Mmax: TElem;
begin
  Mmax := LineMax( A.M[1], A.Dim2 );  {начальное значение}
  for i:=2 to A.Dim1 do
  begin
    max := LineMax( A.M[i], A.Dim2 ); {находим максимум в строке}
    if max > Mmax then Mmax := max
  end;
  MatrixNorm := Mmax
end;

{Возведение матрицы A в степень P}
{Матрица A должна быть размерности (N x N)}
procedure MatrixPower( A: TMatrix; P: Integer; var Result: TMatrix );
var i: Dimension;
begin
  Result := A;
  for i:=2 to P do
    MatrixMul (A, Result, Result)
end;
end.{Конец модуля Matrix}
```

Листинг 1.4. Решение задачи 1

```
program MatrixTest1;
uses Matrix, Crt;
const Dim = 3;
var
  X: TMatrix;
  h1, h2, h3, h4: TMatrix;  {вспомогательные матрицы}
begin
  ClrScr;
  MatrixInput  (X, Dim, Dim);     {ввод матрицы}
  MatrixPower      (X, 3, h1);    {h1 = X^3}

  MatrixNumberMul (h1, 2, h1);    {h1 = 2*X^3}
  MatrixPower      (X, 2, h2);    {h2 = X^2}

  MatrixNumberMul (h2, 5, h2);    {h2 = 5*X^2}
  MatrixNumberMul (X, 3, h3);     {h3 = 3*X}

  MatrixI (Dim, h4);              {h4 = I}
  MatrixNumberMul (h4, 2, h4);    {h4 = 2*I}

  MatrixSub (h1, h2, X);    {вычисление заданного выражения}
  MatrixSum (X, h3, X);
  MatrixSub (X, h4, X);

  writeln ('Результат выражения 2*X^3-5*X^2+3*X-2*I равен ');
  MatrixOutput (X);         {вывод результата}
  WriteLn ('Нажмите клавишу <Enter> ...');
  ReadLn
end.
```

Листинг 1.5. Решение задачи 2

```
program MatrixTest2;
uses Matrix, Crt;
const
  Dim1 = 2;
```

```
  Dim2 = 3;
var
  A, B: TMatrix;
  h1: TMatrix;  {вспомогательная матрица}
begin
  ClrScr;
  MatrixInput  (A, Dim1, Dim2);   {ввод матрицы A}
  MatrixInput  (B, Dim2, Dim1);   {ввод матрицы B}

  MatrixTranspose (B, h1);         {транспонирование матрицы B}
  MatrixSum       (A, h1, h1);     {h1 = A+Bт}
  MatrixMul       (h1, B, h1);     {h2 = (A+Bт)*B}

  WriteLn ('¦(A+Bт)*B¦ = ', MatrixNorm(h1):8:4 );
  WriteLn ('Нажмите клавишу <Enter> ...' );
  ReadLn
end.
```

Пример 3. Работа с очередью

Дана строка текста, символы которой по одному вводятся с клавиатуры. Распечатать сначала все символы строки, затем все знаки препинания в том порядке, в котором они встречались в тексте. Решение задачи дано в листинге 1.7.

Для решения задачи воспользуемся такой структурой данных, как очередь, позволяющей хранить и выводить данные в том порядке, в котором они были занесены в очередь. Описание и все процедуры для работы с очередью вынесем в отдельный модуль `DinQueue` (листинг 1.6).

Листинг 1.6. Реализация процедур для работы с очередью

```
Unit DinQueue;
Interface
{Описание очереди на основе динамических структур данных}
type
  TElem = char;  {Тип элементов очереди}
  {Список, предназначенный для хранения элементов очереди}
  TList = ^TElement;
  TElement = record          {Элемент списка}
               Info: TElem;  {Информационная часть}
```

```
                Next: Tlist   {Указатель на следующий элемент списка}
              end;

  TQueue = record     {Представление очереди}
              Head: TList;  {"Голова" очереди}
              Tail: Tlist   {"Хвост" очереди}
            end;

{Описание процедур и функций для работы с очередью}
{Инициализация очереди Q}
procedure Queue_Init( var Q: TQueue );
{Проверка очереди Q на пустоту}
{Результат функции: true - очередь пуста, false - очередь не пуста}
function Queue_IsEmpty( Q: TQueue ): boolean;
{Поместить элемент E в "хвост" очереди Q}
procedure Queue_Push( var Q: TQueue; E: TElem );
{Извлечь элемент из очереди Q}
{Результат функции: извлеченный из "головы" очереди элемент}
function Queue_Pop( var Q: TQueue ): TElem;

{Реализация процедур и функций для работы с очередью}
Implementation

{Инициализация очереди Q}
procedure Queue_Init( var Q: TQueue );
begin
  {Изначально список, хранящий элементы очереди, пуст}
  Q.Head := nil
end; {Queue_Init}

{Проверка очереди Q на пустоту}
{Результат функции: true - очередь пуста, false - очередь не пуста}
function Queue_IsEmpty( Q: TQueue ): boolean;
begin
  {Очередь пуста, если пуст соответствующий список}
  Queue_IsEmpty := ( Q.Head = nil )
```

```
end; {Queue_IsEmpty}

{Поместить элемент E в "хвост" очереди Q}
procedure Queue_Push( var Q: TQueue; E: TElem );
var Z: TList;
begin
  new( Z ); {Создаем новое звено списка}
  Z^.Info := E;
  Z^.Next := nil;
  if Queue_IsEmpty( Q )
    then Q.Head := Z
    else Q.Tail^.Next := Z;
  Q.Tail := Z
end; {Queue_Push}

{Извлечь элемент из очереди Q}
{Результат функции: извлеченный из "головы" очереди элемент}
function Queue_Pop( var Q: TQueue ): TElem;
const ErrorCode = 255;
var Z: TList;
begin
  if Queue_IsEmpty( Q ) then
    begin
      writeln ('Извлечение элемента из пустой очереди !!! ');
      {Аварийное завершение программы}
      Halt (ErrorCode)
    end
  else
    begin
      Z := Q.Head;           {Сохраняем указатель на звено списка}
      Q.Head := Z^.Next;     {Перемещаемся к следующему звену}
      Queue_Pop := Z^.Info;  {Удаляем звено списка}
      dispose( Z )
    end
end; {Queue_Pop}
end. {UnQueue}
```

Листинг 1.7. Главная программа

```
program TestQueue;
Uses Crt, DinQueue;
Const {Знаки пунктуации}
      PunctMark = ['.',',',':','-','!','?'];
var   Queue : TQueue;  {Очередь}
      E : TElem;       {Очередной символ}
      flag : boolean;  {Флаг, который показывает,}
                       {надо ли выводить фразу "Результат: "}

begin
  ClrScr;
  {Инициализация очереди}
  Queue_Init (Queue);
  write ('Введите строку текста: ');
  flag := true;
  {Помещение знаков пунктуации в очередь или печать остальных символов}
  while not eoln do {Пока не конец строки}
    begin
      {Читаем следующий символ}
      read( E );
      if Flag then
        begin
          write('Результат: ');
          Flag := false;
        end;
      {Если символ - знак пунктуации}
      if E in PunctMark then
        {Помещаем его в очередь}
        Queue_Push (Queue, E)
      else
        {иначе - печатаем его}
        write(E)
    end;
  readln; {Читаем признак конца строки}
  {Печатаем знаки пунктуации}
  while not Queue_IsEmpty( Queue ) do
```

```
    {Пока очередь не пуста, извлекаем символ из очереди и выводим его}
    write (Queue_Pop(Queue));
  writeln;
  write ('Нажмите <Enter> ... ');
  readln
end.
```

Задания для самостоятельной работы

1. Составить модуль, реализующий следующие операции над векторами:
   - ввод компонент вектора с клавиатуры и из файла;
   - вывод компонент вектора на дисплей и в файл;
   - вычисление нормы вектора в различных метриках;
   - определение минимальной (максимальной) координаты вектора и ее порядкового номера;
   - вычисление скалярного произведения двух векторов;
   - проверка на возрастание или убывание последовательности координат.
2. Даны два вектора X и Y размерности $n = 30$. Используя составленный в задании 1 модуль, вычислить

   а) $$\frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{(n\sum x_i^2 - (\sum x_i)^2)(n\sum y_i^2 - (\sum y_i)^2}};$$

   б) $$\frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{(k\sum x_i^2 - (\sum x_i)^2)(p\sum y_i^2 - (\sum y_i)^2}},$$

   где k — порядковый номер максимальной координаты вектора X, p — порядковый номер минимальной компоненты вектора Y.
3. Даны три вектора X, Y, Z, каждый размерности $n = 30$. Используя составленные в задании 1 модули, вычислить

   а) minX * (A, A) + maxY * (B, C),

   где A — тот из данных трех векторов, минимальный элемент которого имеет самый большой номер (считать, что такой вектор единственный), B и C — два других вектора; $minX$ — минимальный элемент вектора X, $maxY$ — максимальный элемент вектора Y;

   б) k * (A, A) + p * (B, C),

   где A — тот из данных трех векторов, координаты которого упорядочены по возрастанию (считать, что такой вектор единственный), B и C — два

других вектора; k — номер минимального элемента вектора Z, p — номер максимального элемента вектора Y.

4. Используя модуль, разработанный для операций с комплексными числами, вычислить значение квадратного трехчлена с комплексными коэффициентами $ax^2 + bx + c$ в комплексной точке x.
5. Используя модуль, разработанный для операций с комплексными числами, вычислить с заданной точностью e значение комплексной функции

$$e^z = 1 + \frac{z}{1!} + \frac{z^2}{2!} + \ldots + \frac{z^n}{n!}.$$

6. Реализовать модуль для работы с множеством, число элементов в котором больше, чем 256. Примените представление "большого" множества в виде массива множеств. Модуль должен содержать процедуры инициализации множества, включения, исключения элементов и проверки принадлежности элемента множеству.
7. Используя модуль для работы с множеством, описанный в предыдущей задаче, составить программу нахождения целых чисел из диапазона 1..10000, удовлетворяющих представлению $n^2 + m^2$, где n и m — целые числа.
8. Реализовать модуль для работы со стеком. Напечатать в обратном порядке символы слова наибольшей длины из заданного текстового файла.
9. Реализовать модуль для работы с очередью. Напечатать наибольшее по длине предложение из заданного текстового файла.
10. Создать модуль, реализующий следующие операции над символьными строками:
    - поиск первого вхождения в строку заданного символа;
    - поиск последнего вхождения в строку заданного символа;
    - поиск последнего вхождения в строку заданной подстроки;
    - поиск в строке первого символа, отличного от пробела;
    - поиск в строке последнего символа, отличного от пробела.

    Написать программу, иллюстрирующую работу полученного модуля.
11. Создать модуль, содержащий описание типа "дата" и операций над ним:
    - проверка корректности совокупности полей (день, месяц, год);
    - увеличение даты на заданное число дней;
    - уменьшение даты на заданное число дней;
    - преобразование типа "строка" → "дата";
    - преобразование типа "дата" → "строка".

    Написать программу, иллюстрирующую работу полученного модуля.

12. Создать модуль, реализующий представление длинного целого числа в виде строки символов, а также операции *, /, −, +. Написать программу, иллюстрирующую работу полученного модуля.
13. Создать модуль, реализующий представление полиномов в виде списка, а также операции *, /, −, +. Написать программу, иллюстрирующую работу полученного модуля.
14. Создать модуль, реализующий построение четырех видов диаграмм:
    - столбчатая;
    - круговая;
    - 3D столбчатая;
    - график.

    Написать программу, иллюстрирующую работу полученного модуля.
15. Создать модуль для работы с файлом сотрудников, где каждая запись содержит поля "фамилия", "имя", "отчество", "должность", "оклад" и признак удаленности записи. Реализовать следующие операции с файлом:
    - добавление;
    - удаление;
    - поиск по фамилии, имени, отчеству, должности и окладу;
    - расчет суммарного и среднего оклада всех сотрудников.