Е.К. Липачёв

ВВЕДЕНИЕ В КОМПЬЮТЕРНЫЕ НАУКИ. ОСНОВНЫЕ АЛГОРИТМЫ

КАЗАНЬ — 2003

УДК 519.6
ББК 32.811
Л61

Печатается по решению
кафедры теории функций и приближений
Казанского государственного университета
от 03.07.03 Протокол №1

Рецензент

Кандидат физико-математических наук, доцент
С.Н. Тронин

Липачёв Е.К.

Л61 Введение в компьютерные науки. Основные алгоритмы: Учебно-методическое пособие. – Казань: Казанский государственный университет им. В.И.Ульянова-Ленина, 2003. – с.84.

Предназначено для студентов механико-математического факультета специальностей 010100 – «математика» и 010500 – «механика» как вспомогательный материал на занятиях в рамках компьютерного цикла.

УДК 519.6
ББК 32.811

Оглавление

1. Консольные приложения

1.1. Создание консольных приложений

В тех случаях, когда при решении практической задачи не требуется создавать графический интерфейс, а нужны, например, результаты вычислений, можно построить консольное приложение. Примером консольных приложений являются программы, созданные в Turbo Pascal для DOS. Заметим, что большинство программ, написанных на языке Паскаль, можно выполнить в среде Delphi как консольные приложения. Большую часть учебных задач по программированию, например, предложенных в [1], [3], целесообразно решать как консольные приложения.

Мы будем использовать консольные приложения для описания различных алгоритмов, — интерфейсная часть проекта в этих случаях второстепенна.

Для создания консольного приложения выполнить команду меню **File | New | Other…** . В окне диалога **New** выберем пиктограмму **Console Application.**

В результате получим следующий шаблон приложения:

```
program Project1;
{$APPTYPE CONSOLE}
uses  SysUtils;
begin
 {TODO -oUser-cConsole Main}
 {Запишите код здесь}
end.
```

Для ввода данных в консольных приложениях используются операторы **`Read, Readln,`** а для вывода — **`Write, Writeln`**. Напомним, что оператор **`Readln;`** (без аргументов) можем использовать для задержки (до нажатия клавиши Enter) DOS-окна с сообщениями консольного приложения.

1.2. Как русифицировать консольное приложение

Консольное приложение работает в режиме эмуляции DOS, поэтому пояснительные записи на русском языке, выводимые операторами `Writeln`, будут непонятны из-за несоответствия кодировок. Ситуацию можно исправить преобразованием выводимого текста в кодировку DOS. В следующем примере для этой цели используется функция **`AnsiToOem(),`** переводящая текст из кодировки Windows (Win 1251) в кодировку DOS (OEM 866).

Пример 1.2.1. Перевод из кодировки Win 1251 в кодировку OEM 866.

```
program OemStr;
{$APPTYPE CONSOLE}
uses   SysUtils;
function AnsiToOem(s:String):String;
{Преобразование ANSI в OEM}
Var s_o:String; i:Integer;
    ch_o,ch : Char;
begin
 s_o :='';
 for i :=1 to Length(s) do
  begin
   ch:=s[i];
   Case ch of
   'a'..'п': ch_o := Chr(Ord(ch)-64);
   'p'..'я': ch_o := Chr(Ord(ch)-16);
        'ё': ch_o := Chr(241);
        'Ё': ch_o := Chr(240);
   'A'..'Я': ch_o := Chr(Ord(ch)-64)
   else ch_o := ch
   end;
      s_o := s_o + ch_o;
   end;
   Result:=s_o;
 end; {AnsiToOem}
Begin
{ Вместо}
 Writeln('Консольное приложение');
{ Записываем }
 Writeln(AnsiToOem('Консольное приложение'));
 Readln; {Задержка}
End.
```

1.3. Задачи

1.3.1. Вводится целое положительное число n. Переставить первую и последнюю цифру числа n.

1.3.2. Преобразовать введенное двоичное число в десятичное.

1.3.3. Преобразовать введенное шестнадцатеричное число в десятичное.

1.3.4. Преобразовать введенное десятичное число в шестнадцатеричное.

1.3.5. Получить таблицу температур по Цельсию от -40° до $+40^\circ$ и их эквивалентов по шкале Фаренгейта, используя формулу пересчета

$$t_F = 1.8t_C + 32.$$

1.3.6. Вводится целое положительное число n. Вычислить a_n, где

$$a_0 = 1, \quad a_k = ka_{k-1} + 1/k, k = 1,2,\ldots.$$

1.3.7. Вводится целое положительное число ***n***. Вычислить F_n, где

$$F_0 = 0, F_1 = 1, \quad F_k = F_{k-1} + F_{k-2}, k = 2,3,\ldots.$$

1.3.8. Вводится целое положительное число n. Вычислить последовательность $F_1, \ldots, F_n$, где

$$F_k = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{k+2} - \left(\frac{1-\sqrt{5}}{2}\right)^{k+2}\right), k=1,2, \ldots .$$

Замечание. Числа, рассмотренные в задачах 1.3.7 и 1.3.8, носят название чисел Фибоначчи (см., напр., [8], [11], [16]).

1.3.9. Вводится целое положительное число n. Вычислить последовательность $r_1, \ldots, r_n$, где

$$r_1 = 1, r_{k+1} = (125 r_k) \bmod 8192.$$

Это пример генератора псевдослучайных чисел (подробности см. в [12, гл. 3], [16, стр. 210]).

1.3.10. (**Алгоритм Евклида**). Составить программу нахождения наибольшего общего делителя целых чисел $A, B\,(A > B > 0)$. В алгоритме Евклида, начиная с $r_{-1} = A, r_0 = B$, производятся последовательные деления r_{i-2} на r_{i-1}, в результате чего вычисляется r_i как остаток от деления, т.е.

$$r_{i-2} = d_i\, r_{i-1} + r_{i,} \quad i = 1,2,\ldots,k.$$

Деления выполняются до получения остатка $r_{k+1} = 0$. Тогда r_k — наибольший общий делитель чисел A и B.

1.3.11. (**Расширенный алгоритм Евклида**). Вводятся целые числа $A, B\ (A > B > 0)$. Составить программу вычисления натуральных чисел x и y, таких что $Ax + By = НОД(A,B)$, где через $НОД(A,B)$ обозначен наибольший общий делитель чисел A и B.

Вычисления проводятся как в алгоритме Евклида, но дополнительно вычисляются две последовательности

$$x_{-1} = 1,\ y_{-1} = 0,\ x_0 = 0,\ y_0 = 1,$$

$$x_i = x_{i-2} - d_i\, x_{i-1},\ y_i = y_{i-2} - d_i\, y_{i-1,} \quad i > 0,$$

где через d_i, как и в задаче 1.3.10, обозначено частное от деления r_{i-2} на r_{i-1}. Значения x_k и y_k, при которых $r_k = НОД(A,B)$, будут искомыми.

1.3.12. Для вычисления наибольшего общего делителя целых чисел $A, B\,(A > B > 0)$ можно использовать также следующий алгоритм (подробности см. [4]). Установить начальные значения $r_A = A$ и $r_B = B$. Циклически, пока выполнено условие $r_A \neq r_B$, изменять значения этих переменных: если $r_A > r_B$, заменить r_A на $r_A - r_B$, иначе заменить r_B на $r_B - r_A$. После завершения цикла $НОД(A,B) = r_A = r_B$.

1.3.13. Вводится массив целых чисел. Подсчитать сколько различных чисел в этом массиве.

1.3.14. Вводится массив попарно различных целых чисел. Напечатать все перестановки этих чисел.

Замечание. Задача сводится к нахождению всех перестановок чисел $1,2,\ldots,n$. Перестановки можно порождать следующим образом (подробности см., напр., [15], [19]). Начиная с перестановки $(1,2,\ldots,n)$, строим из $\Pi=(\pi_1,\pi_2,\ldots,\pi_n)$ следующую путем просмотра Π справа налево в поисках самой правой позиции, в которой $\pi_i<\pi_{i+1}$. Найдя такую позицию i, ищем π_j как наименьший элемент, расположенный справа от π_i и больший его. Затем выполняем перестановку элементов π_j и π_i, а отрезок $\pi_{i+1},\ldots,\pi_n$ записываем в обратном порядке. Алгоритм заканчивает работу, когда $i=0$, что происходит, если $\pi_1>\pi_2>\cdots>\pi_n$ (это последняя в лексикографическом порядке перестановка).

1.3.15. Вводится массив целых чисел. Найти число, повторяющееся максимальное количество раз. Если таких чисел несколько, вывести одно из них.

1.3.16. Вводится массив целых чисел. Найти длину самой длинной последовательности подряд идущих элементов массива, равных нулю.

1.3.17. Составить программу вычисления цепной дроби (см., напр., [9])

$$\left[a_0,\frac{b_1}{a_1},\frac{b_2}{a_2},\frac{b_3}{a_3},\cdots\right]\equiv a_0+\cfrac{b_1}{a_1+\cfrac{b_2}{a_2+\cfrac{b_3}{a_3+\cdots}}},$$

где $a_0,a_1,\ldots,a_n,\ b_1,b_2,\ldots,b_n$ — заданные действительные числа.

1.3.18. Вычислить сумму ряда

$$\sum_{i=1}^{\infty}\frac{(-1)^i}{i!}$$

с ошибкой, не превышающей $E>0$ (напр., $E=10^{-6}$). Будем считать, что требуемая точность достигается, если частная сумма ряда отличается от предшествующей частной суммы менее, чем на E.

1.3.19. Вычислить сумму ряда

$$\sum_{i=1}^{\infty}\frac{(-1)^{i+1}}{i(i+1)(i+2)}$$

с ошибкой, не превышающей $E>0$ (напр., $E=10^{-6}$).

1.3.20. Вычислить сумму ряда

$$\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}$$

с ошибкой, не превышающей $E>0$ (напр., $E=10^{-6}$).

1.3.21. Вводится действительное число x. Вычислить сумму ряда

$$\sum_{k=0}^{\infty}\frac{(-1)^k x^{2k+1}}{2k+1}$$

с ошибкой, не превышающей $E>0$ (напр., $E=10^{-6}$).

1.3.22. Вычислить сумму ряда

$$\varphi(x)=\sum_{i=1}^{\infty}\frac{1}{i(i+x)}$$

с ошибкой, не превышающей $E>0$, для значений x от 0 до 1 с шагом 0.1.

Замечание. На примере этого ряда можно показать, что непосредственные вычисления, без предварительного анализа, приводят к медленно работающей программе (см., напр., [24, стр. 42]). При вычислении можем применить прием, называемый ***ускорением сходимости*** (см., напр., [8, стр. 199]). Рассмотрим для этого вспомогательные ряды

$$S^{(m)}:=\sum_{i=1}^{\infty}\frac{1}{i(i+1)\ldots(i+m)}=\frac{1}{m\cdot m!},\quad m=1,2,\ldots$$

Тогда разность $\varphi(x)-S^{(1)}$, с одной стороны, равна $\varphi(x)-1$, а с другой,

$$(1-x)\sum_{i=1}^{\infty}\frac{1}{i(i+1)(i+x)}.$$

Далее, имеем

$$\sum_{i=1}^{\infty}\frac{1}{i(i+1)(i+x)}-S^{(2)}=(2-x)\sum_{i=1}^{\infty}\frac{1}{i(i+1)(i+2)(i+x)}.$$

Следовательно,

$$\varphi(x)=1+(1-x)\frac{1}{4}+(1-x)(2-x)\sum_{i=1}^{\infty}\frac{1}{i(i+1)(i+2)(i+x)}.$$

1.3.23. Вычислить сумму ряда

$$S=\sum_{i=1}^{\infty}\frac{1}{i^2+1}$$

с ошибкой, не превышающей $E>0$ (напр., $E=10^{-6}$).

Замечание. Для вычисления этого ряда можем использовать формулы

$$\sum_{i=1}^{\infty}\frac{1}{i^2}=\frac{\pi^2}{6},\quad \sum_{i=1}^{\infty}\frac{1}{i^4}=\frac{\pi^4}{90}.$$

Из соотношений

$$\frac{1}{i^2+1}=\frac{1}{i^2}-\frac{1}{i^2(i^2+1)},\quad \frac{1}{i^2(i^2+1)}=\frac{1}{i^4}-\frac{1}{i^4(i^2+1)}$$

получаем, что

$$S=\sum_{i=1}^{\infty}\frac{1}{i^2}-\sum_{i=1}^{\infty}\frac{1}{i^2(i^2+1)}=\frac{\pi^2}{6}-\frac{\pi^4}{90}+\sum_{i=1}^{\infty}\frac{1}{i^4(i^2+1)}.$$

1.3.24. Подсчитать количество шагов, необходимых для вычисления ряда из задачи 1.3.23 с заданной точностью, используя оба варианта вычислений — “непосредственный” и с “ускорением”.

1.3.25. Вводится числовая матрица. Элемент матрицы называется седловой точкой, если он является одновременно наименьшим в своей строке и наибольшим в своём столбце. Найти номера строки и столбца какой-нибудь седловой точки.

1.3.26. Вводятся числовые матрицы A и B размера $k\times l$ и $l\times m$. Найти произведение AB.

1.3.27. Вводится числовая матрица A порядка n. Вычислить матрицу A^2.

1.3.28. Вводятся числовые матрицы A и B порядка n. Получить матрицу $AB-BA$.

1.3.29. Вводятся числовые матрицы A, B и C порядка n. Получить матрицу $(A+B)C$.

1.3.30. Вводится числовая матрица A порядка n. Вычислить транспонированную матрицу.

1.3.31. Вводятся действительные числа $x_1,\ldots,x_n$. Составить матрицу Вандермонда $A=[a_{ij}]$, $a_{ij}=x_j^i, i,j=1,\ldots,n$ и вычислить ее определитель согласно формуле

$$\det A=\prod_{1\le j\le n}x_j\prod_{1\le i<j\le n}(x_j-x_i).$$

1.3.32. Даны натуральные числа m и n ($m<n$). Составить программу нахождения всех наборов $(k_1,\ldots,k_m)$, таких что $1\le k_1<k_2<\cdots<k_m\le n$.

1.3.33. Вводятся действительные числа $x_1,\ldots,x_n$. Вычислить матрицу $B=[b_{ij}]$, используя соотношения

$$b_{ij}=(-1)^{j+1}\sum_{\substack{1\le k_1<\cdots<k_{n-j}\le n\\ k_1,\ldots,k_{n-j}\ne i}}(x_{k_1}x_{k_2}\cdots x_{k_{n-j}})/x_i\prod_{\substack{1\le k\le n\\ k\ne i}}(x_k-x_i).$$

Отметим, что матрица B является обратной матрицей к матрице Вандермонда (см., напр., [11, стр. 67]).

1.3.34. Вводятся числовые матрицы A и B порядка n. Вычислить матрицу $C = A \circ B$, состоящую из элементов

$$c_{ik} = \min_{1 \le j \le n} (a_{ij} + b_{jk}).$$

1.3.35. Пусть A числовая матрица порядка n. Вычислить матрицу $D = A^{n-1}$, где возведение в степень выполнено по правилам операции, определенной в предыдущей задаче.

Замечание. Операция умножения, рассмотренная в задачах 1.3.34 и 1.3.35, используется в теории графов. В частности, если A матрица длин ребер некоторого графа, то D — матрица расстояний этого графа (см., напр., [16, стр. 141]).

2. Работа с файлами

2.1. Общие сведения

В Delphi поддерживаются три типа файлов: текстовые, типизированные и двоичные. Текстовые файлы содержат информацию, которую можно редактировать в любом текстовом редакторе. Типизированные файлы содержат данные типа запись (**`record` … `end`**). Остальные файлы можно рассматривать как двоичные.

2.2. Схема работы с текстовыми файлами

1. Описываем файловую переменную

```
Var F:TextFile;
```

2. Связываем файловую переменную с именем файла оператором

```
AssignFile(F,NameFile);
```

например,

```
assignfile(f,'info.txt');
assignfile(f,'c:\examples\data\info.txt');
```

3. Открываем файл.
   a. Для чтения (предполагаем, что файл уже существует).

```
Reset(F);
```

   b. Для записи (если файл существует, то прежняя информация будет удалена).

```
Rewrite(F);
```

   c. Для добавления информации в конец существующего файла

```
Append(F);
```

При открытии файла возможны сбои в работе программы. Исключительная ситуация возникнет, например, при попытке открыть файл на недоступном устройстве (к примеру, записать на CD-R или прочитать файл, которого нет). Если не обрабатывать исключительные

ситуации, то работа программы будет прекращена. Об обработке чуть позднее.

4. Проводим операции с файлом: читаем из файла или записываем в него.
   a. Читаем с помощью операторов
      `Read(F,…..);` **`Readln(F,…..);`**
      С помощью функции **Eof(F)** можно определить, что файл полностью прочитан (**`Eof(F)`** возвращает `True`, если прочитан последний символ в файле). С помощью функции **`Eoln(F)`** можно определить, что закончено чтение строки (**`Eoln(F)`** возвращает True после прочтения последнего символа в строке).
   b. Записываем в файл с помощью операторов
      `Write(F,….);` или **`Writeln(F,…..);`**

   Отметим, что одновременно нельзя читать и записывать в файл.
5. Закрываем файл с помощью оператора
   `CloseFile(F);`

Приведем примерный участок кода, используемый при открытии текстового файла

```
Var f : TextFile;
Begin
   AssignFile(f, 'info.txt');
   Reset(f);
  { РАБОТА С ФАЙЛОМ }
   CloseFile(f);
End;
```

Для записи в новый файл можем применить операторы

```
Var f : TextFile;
Begin
   AssignFile(f, 'info.txt');
   Rewrite(f);
  { РАБОТА С ФАЙЛОМ }
   CloseFile(f);
End;
```

Добавление данных в конец файла выполняется командами

```
Var f : TextFile;
Begin
   AssignFile(f, 'info.txt');
   Append(f);
  { РАБОТА С ФАЙЛОМ }
   CloseFile(f);
End;
```

Пример 2.2.1. Вывод на экран информации из файла “info.txt”.

```
Var f: TextFile;
      s:String;
Begin
      AssignFile(f,’info.txt’);
      Reset(f);
      While Not Eof(f) do
  begin
   Readln(f,s); Writeln(s)
  end;
  CloseFile(f);
End.
```

2.3. Обработка исключительных ситуаций при работе с файлами

С помощью защищенных блоков можем обработать ошибки, возникающие при открытии файла и работе с ним. “Исправим” пример 2.2.1, добавив защищенные блоки.

Пример 2.3.1.

```
Uses
    Dialogs; {для поддержки ShowMessage()}
Var f: TextFile;
    s:String;
Begin
 AssignFile(f,’info.txt’);
 try
  Reset(f);
  While Not Eof(f) do
   begin
    Readln(f,s); Writeln(s)
   end;
  CloseFile(f);
 except
  on E:EinOutError do
   ShowMessage(‘Ошибка при работе с файлом.’+
           ‘Номер ошибки=’ +IntToStr(E.ErrorCode))
 end
End.
```

Можно произвести обработку иначе. Этот способ применялся при работе в Turbo Pascal и сохранился в Delphi. Он состоит в обработке результата функции **IOResult**. Проиллюстрируем тем же примером.

Пример 2.3.2.

```
Var f: TextFile;
    s:String;
```

```
Begin
    AssignFile(f,'info.txt');
  {$I-}
    Reset(f);
 {$I+}
If IOResult<>0 then
        ShowMessage('Ошибка при работе с файлом')
else
  begin
    While Not Eof(f) do
     begin
       Readln(f,s); Writeln(s)
     end;
   CloseFile(f);
  end
End.
```

Функция `IOResult()` возвращает код ошибки, возникшей в ходе выполнения последней операции ввода-вывода. Чтобы получить код ошибки, необходимо с помощью директивы компилятора **`{$I-}`** отключить автоматическую проверку операций ввода-вывода. Если этого не сделать, программа автоматически завершит свою работу. После выполнения операций ввода-вывода, способных привести к ошибке, необходимо с помощью директивы **`{$I+}`** снова включить автоконтроль ввода-вывода.

2.4. Чтение из текстового файла

Пример 2.4.1. Найти в файле `'book.txt'` самое длинное слово. Под словом, в данном случае, будем понимать последовательность символов, ограниченную пробелами или другими разделителями.

```
program Project2_4_1;
{$APPTYPE CONSOLE}
{Поиск самого длинного слова в файле}
uses  SysUtils;
Var
 f:Text;
 lmax:Integer; dlina,i:Integer;
 slovo, maxslovo:String;
 ch:char; Dividers:set of char; flag:Boolean;
begin
 {разделители:}
 Dividers:=[' ',',','.','!','?',';',':'];
 AssignFile(f,'book.txt');
 reset(f);
 lmax:=0;
```

```
 slovo:='';  dlina:=0;
 flag:=False;
 while Not Eof(f) {пока файл не кончился} do
  begin {1}
   read(f,ch);
   if ch in Dividers then
    {встретили разделитель - слово закончилось}
    flag :=True
   else
    if Eoln(F) then { закончилась строка}
      begin
       flag :=True;
       slovo := slovo+ch; Inc(dlina)
      end
    else {продолжаем собирать слово}
       begin
        slovo:=slovo+ch; Inc(dlina)
      end;
   {если очередное слово найдено, сравним его длину:}
   if flag then
    begin {2}
      if dlina>lmax then
       begin
        lmax:=dlina; maxslovo:= slovo;
       end;
      {Для нового слова:}
      slovo:=''; dlina:=0;
      flag:=False
     end {2}
  end; {1}
 closefile(f);
 Writeln('Слово ',maxslovo,' имеет длину ', lmax);
Readln;
End.
```

2.5. Запись в текстовый файл

Текстовые файлы удобно использовать для хранения результатов вычислений.

Пример 2.5.1. Вычислить ***n*** членов последовательности Фибоначчи, где ***n*** (***n*** <93) — натуральное число, вводимое при запуске программы. Результаты сохранить в текстовом файле. Числа Фибоначчи вычисляются по формуле (см., напр., [8], [11], [16])

$$F_0 = 0, F_1 = 1, \quad F_k = F_{k-1} + F_{k-2},\ k > 1$$

(см. задачи 1.3.7 и 1.3.8).

```
program Project2_5_1;
{$APPTYPE CONSOLE}
uses   SysUtils;
 Var g: TextFile; NameFile:String;
     i,n: Integer; u,v,w: Int64;
Begin
  NameFile:='Fibb.txt';
  Writeln('Последовательность Фибоначчи для n=');
  Readln(n);
  AssignFile(g, NameFile);
  Rewrite(g);
  u := 0; v := 1;
  {Запишем в файл первые два числа}
  Writeln(g, 'f_1=',u); Writeln(g,'f_2', v);
  for i:= 2 to n do
    begin
      w := v + u;
      Writeln(g,'f_',i:3,'=',w);
      u := v; v := w;
    end;
  CloseFile(g);
 End.
```

Замечание. В программе предполагается, что ***n***<93. Это ограничение вызвано тем, что количество значащих цифр у последующих чисел Фибоначчи превышает допустимый диапазон значений типа Int64. В разделе 5 показано, как можно справиться с подобной проблемой.

2.6. Компоненты Delphi для работы с файлами

Для выбора файла перед операцией открытия можно использовать компонент **OpenDialog**, размещенный на странице **Dialogs** палитры компонент. Из свойств компонента отметим следующие:

Свойство	*Возможные значения*
DefaultExt	Расширение, добавляемое к имени файла, если у него не указано расширение
FileName	Содержит полное имя выбранного файла
Filter	Текст фильтров, используемых при отображении файлов. Любое количество строковых пар, разделённых символом **“|”**. В каждой паре первая часть содержит поясняющий текст, а вторая часть определяет фильтр, например, "**Все файлы (*.*)|*.***"
InitialDir	Папка, содержимое которой отображается при открытии окна выбора файлов

Пример 2.6.1.

```
{Определим фильтр:}
OpenDialog1.Filter := 'Файлы данных (*.dat)|*.dat'
+'Текстовые файлы (*.txt, *.doc)|*.txt;*.doc' +
 'Все файлы (*.*)|*.*';
if OpenDialog1.Execute then
  Name := OpenDialog1.FileName; {Имя выбранного файла}
```

При сохранении файла можно использовать диалоговое окно "**Save**" ("**Сохранить файл**"), которое выводится с помощью компонента **SaveDialog** (страница **Dialogs** палитры компонент). Этот компонент имеет точно такие же свойства, как и компонент **OpenDialog**.

Пример работы с этими компонентами содержится в разделе 4.5.2.

2.7. Типизированные файлы

Файлы, в которых хранятся структуры данных Object Pascal, называются типизированными файлами (файлами записей). Покажем на примере схему работы с типизированным файлом.

Пример 2.7.1. Создается файл для хранения информации о книгах (автор, название книги, год издания). Введен тип `TBook` для работы с такой информацией.

```
Type
 TBook = record
          Autors : String[80]; {Фамилии авторов}
          Title : String[80]; { Название }
          Year : Integer; {Год издания}
         end;
Var
   F: File of TBook; Book: TBook;
Begin
   AssignFile(F,'BookFile.dat');
   Rewrite(F);
  {Подготовим данные:}
   Book.Autors:='Вирт Н.';
   Book.Title:=
   'Алгоритмы + структуры данных = программы';
   Book.Year:=1985;
  {Запишем в файл:}
   Write(F,Book);
  {Снова данные:}
   Book.Autors:=
        'Грэхем Р., Кнут Д., Паташник О.';
   Book.Title:=
 'Конкретная математика. Основания информатики';
   Book.Year:=1998;
```

```
  {Запишем в файл:}
   Write(F,Book);
   { . . . . }
   CloseFile(F);
End;
```

Пример 2.7.2. Чтение записи из типизированного файла.

```
Type
 TBook = record
          Autors : String[80]; {Фамилии авторов}
          Title : String[80]; { Название }
          Year : Integer; {Год издания}
         end;
Var F: File of TBook; Book: TBook;
Begin
   AssignFile(F,'BookFile.dat');
   Reset(F);
   If Not Eof(F) then  Read(F,Book);
   { . . . . }
   CloseFile(F);
End;
```

Пример 2.7.3. Добавление записи в типизированный файл.

```
Type
 TBook = record
          Autors : String[80]; {Фамилии авторов}
          Title : String[80]; { Название }
          Year : Integer; {Год издания}
         end;
Var F: File of TBook; Book: TBook;
Begin
   AssignFile(F,'BookFile.dat');
   Reset(F);
   {переместим указатель в конец файла:}
   Seek(F, FileSize(F));
   {Подготовим данные:}
   Book.Autors:='Тейксейра С., Пачеко К.';
   Book.Title:=
   'Borland Delphi 6. Руководство разработчика';
   Book.Year:=2002;
  {Запишем в файл:}
   Write(F,Book);
   CloseFile(F);
End;
```

Процедура `Seek()` используется в этом примере для перемещения указателя в конец файла. С помощью этой процедуры можем обращаться к записям типизированного файла “напрямую”, по их порядковому номеру.

Процедура `Seek()` имеет два параметра: первый имеет тип файловой переменной и служит для определения файла, вторым параметром передаем порядковый номер элемента файла, на который должен быть установлен указатель файла (первый элемент файла имеет номер 0).

2.8. Задачи

2.8.1. Дан текстовый файл ***f***, получить копию файла ***f*** в файле ***g***.
2.8.2. Даны текстовые файлы ***f*** и ***g***. Записать в файл ***h*** сначала компоненты файла ***f***, а затем — компоненты файла ***g***.
2.8.3. Дан текстовый файл ***f***. Создать файл ***g***, образованный из файла ***f*** заменой всех прописных букв одноименными строчными. Заметим, что стандартные функции, например, `UpperCase()`, не “справляются” с буквами русского алфавита.
2.8.4. Дан текстовый файл ***f***. Подсчитать количество слов в файле ***f.***
2.8.5. Дан текстовый файл ***f***. Подсчитать в файле ***f*** количество слов заданной длины ***n.***
2.8.6. Подсчитать количество вхождений данного слова ***s*** в текстовый файл**.**
2.8.7. Дан текстовый файл ***f***, содержащий информацию в кодировке DOS (OEM – 866). Преобразовать информацию в кодировку Windows (Win-1251). (Функция, выполняющая обратное преобразование, приведена в примере раздела 1.2.)
2.8.8. Дан текстовый файл ***f***, содержащий информацию на русском языке. Преобразовать текст, заменив символы кириллицы на “подходящие” латинские буквы (для букв, не имеющих аналогов в латинице, использовать сочетания букв, например, “ж” — “zh”, “х” — “kh”, “ц” — “ts”, “ч” — “ch”, “ш” — “sh”, “щ” — “tsh”, “ю” — “ju”, “я” — “ja”). Преобразованный текст записать в файл ***g***.
2.8.9. Дан текстовый файл ***f***. Найти слово (слова), встречающиеся наиболее часто.
2.8.10. Дан текстовый файл ***f***. Вычислить количество вхождений каждого символа в файле ***f***. Результат сохранить в массиве

```
Var  Num:Array[Char] of Integer;
```

так, чтобы элемент `Num[ch]` содержал количество вхождений символа `ch` в файле ***f***.
2.8.11. Дан файл, содержащий сведения о автомобилях. Эти сведения состоят из марки автомобиля, фамилии владельца и гос. номера. Составить программу, в которой по номеру автомобиля выводится фамилия владельца.

3. Динамические массивы

Динамические массивы отличаются от “обычных” тем, что размерность массива можно не задавать явно, а определить во время работы программы. Этот тип данных появился в Delphi 4.

Динамические массивы объявляются с помощью описания

Array of базовый тип;

Например,

Var b : array of Double;

Динамические массивы индексируются с нуля. Размерность массива при объявлении не указывается. Если в программе используется динамический массив, то необходимо выделить память для этого массива с помощью процедуры **SetLength().** В качестве первого параметра процедуры указывается имя массива, а вторым — количество элементов массива. Например, **SetLength(b, 3).**

Пример 3.1.

```
Var  b : Array of Double;
begin
 SetLength(b, 3);
 b[0] := 1;  b[1] := 0.1;  b[2] := 4;
end;
```

3.1. Функции, используемые при работе с динамическими массивами

Length(a) — число элементов динамического массива **a**.
High(a) — наибольший индекс массива **a** (т.е. **Length(a)-1**). Если массив имеет нулевую длину, функция **High()** возвращает значение –1.
Low(a) — наименьший индекс массива (=0).

3.2. Многомерные динамические массивы

Многомерные динамические массивы объявляются с помощью повторного использования конструкции

Array of . . .

Например,

Var a : Array of array of Double;

— объявлен двумерный динамический массив вещественных чисел.

Для выделения памяти для такого массива используется процедура **SetLength()** с тремя параметрами: первый — имя массива, второй — количество строк, третий — количество столбцов. Например,

SetLength(a, 3, 3).

Пример 3.2.1. Можно определить динамические массивы, не являющиеся прямоугольными.

```
Var a : Array of array of double;
    i, j : Integer;
begin
 SetLength(a, 10);
 for  i := 0 to  High(a)  do
  begin
   SetLength(a[i], i + 1);
   for j := 0 do  High (a[i]) do
     a[i, j] :=sin(i+j);
   end;
end;
```

Размер динамического массива можно изменять после его объявления.

Пример 3.2.2. Из матрицы **a** создается новая матрица **b**, в которую включаются строки матрицы **a**, не содержащие нулевых элементов. Память под каждую строку матрицы **b** выделяется в процессе проверки строк матрицы **a.**

```
program Project3_2_2;
{$APPTYPE CONSOLE}
uses   SysUtils;
Type
    TMatr=Array of Array of Double;
procedure ExclN(a:TMatr; var b:TMatr);
Var
    i,j,k,n,m:Integer; flag:Boolean;
begin
  n:=Length(a);//количество строк матрицы  a
  k:=0; //текущее количество строк матрицы b
  for i:=0 to n-1 do
   begin {1}
    {просматриваем i-ю строку матрицы a:}
     flag:=false;{изменим на True, если
                  в строке есть нулевые элементы}
     m:=Length(a[i]); // длина строки
     j:=0;
     repeat
      if a[i,j]=0 then flag:=true;
      Inc(j);
     until (j>=m) or flag;
     if Not flag then {i-я строка без нулей}
      begin
       {i-ю строку матрицы a запишем в
        k-ю строку матрицы b}
```

```
        SetLength(b,k+1,m);{Увеличили память
                            на одну строку}
        for j:=0 to m-1 do b[k,j]:=a[i,j];
        Inc(k);
       end;
    end; {1}
end;
Var a,b:TMatr;  i,j:Integer;
Begin
  SetLength(a,3,3);
  a[0,0]:=1; a[0,1]:=1;a[0,2]:=0;
  a[1,0]:=2; a[1,1]:=2; a[1,2]:=2;
  a[2,0]:=3; a[2,1]:=3; a[2,2]:=3;
  ExclN(a,b);
  for i:=0 to High(b) do
  begin
   for j:= 0 to High(b[i]) do
     write(b[i,j]);
   writeln;
  end;
  Readln;
End.
```

3.3. Параметры функций в виде открытых массивов

В процедурах и функциях допускается использование параметров с описаниями в виде

Array of базовый тип;

Например,

function Sum(b : Array of Double) : Double;

Отметим, что индексация открытых массивов начинается с нуля.

Пример 3.3.1. Использование открытых массивов.

```
program Project3_3_1;
{$APPTYPE CONSOLE}
uses   SysUtils;
function Sum(a:array of double):double;
Var i:Integer; S:double;
begin
  S:=0;
  for i := 0 to High(a) do
    S:= S + a[i];
  Result := s;
end;{Sum}
Var a:array[0..5] of double=(3,1,4,1,5,9);
    b:array[-5..0] of double=(2,6,5,3,5,8);
```

```
    c:array[1..12] of double=(3,1,4,1,5,9,
                              2,6,5,3,5,8);
Begin
  writeln(Sum(a));
  writeln(Sum(b));
  writeln(Sum(c));
  Readln;{задержка}
End.
```

При вызове функций и подпрограмм значения открытого массива можно передать с помощью конструктора массива. Значения элементов массива записываются в виде списка, обрамленного квадратными скобками, запятая служит разделителем элементов.

Пример 3.3.2. Использование конструктора массива.

```
program Project3_3_2;
{$APPTYPE CONSOLE}
uses   SysUtils;
function Sum(a:array of double):double;
Var i:Integer; S:double;
begin
  S:=0;
  for i := 0 to High(a) do
    S:= S + a[i];
  Result := s;
end;{Sum}
Begin
  writeln(Sum([3,1,4,1,5,9]));
  writeln(Sum([2,6,5,3,5,8]));
  writeln(Sum([3,1,4,1,5,9,2,6,5,3,5,8]));
  Readln;{задержка}
End.
```

3.4. Задачи

3.4.1. Вводятся натуральное число n (n>1) и действительные числа $x_0, x_1, \ldots, x_n$. Получить последовательность

$$x_0 - x_n, x_1 - x_n, \ldots, x_{n-1} - x_n.$$

3.4.2. Вводятся натуральное число n и действительные числа $x_0, x_1, \ldots, x_n$. Вычислить

$$x_0 x_n + x_1 x_{n-1} + \ldots + x_n x_0.$$

3.4.3. Вводятся натуральное число n и действительные числа $x_0, x_1, \ldots, x_n$. Вычислить

$$(x_0 + 2x_1 + 3x_2)(x_1 + 2x_2 + 3x_3) \cdot \ldots \cdot (x_{n-2} + 2x_{n-1} + 3x_n).$$

3.4.4. Вводятся натуральное число n и действительные числа $x_0, x_1, \ldots, x_n$. Вычислить среднее арифметическое этих чисел.

3.4.5. Многочлен

$$P_n(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n$$

представляется массивом коэффициентов a. Написать функцию вычисления значения многочлена для заданного x. Вычисления организовать по схеме Горнера

$$P_n(x) = \left(\ldots\left(a_0 x + a_1\right)x + \cdots + a_{n-1}\right)x + a_n \ .$$

3.4.6. Составить процедуру сложения динамических матриц.

3.4.7. Составить процедуру перемножения динамических матриц. С помощью параметра типа `Boolean` осуществить контроль за возможностью выполнения этой операции (количество строк первой матрицы должно совпадать с количеством столбцов второй матрицы).

3.4.8. Дана матрица **A** вещественных чисел. Создать новую матрицу, включив в нее те строки матрицы **A**, в которых нет элементов, равных минимальному значению элементов матрицы **A.**

3.4.9. Дана матрица вещественных чисел. Создать новую матрицу путем включения строк исходной матрицы, содержащих хотя бы один отрицательный элемент. Если все элементы исходной матрицы неотрицательны, вывести соответствующее сообщение.

4. Решение уравнений

В этом разделе рассмотрены некоторые простейшие численные методы приближенного решения линейных и нелинейных уравнений и систем.

Для решения нелинейных уравнений используются метод итераций, метод половинного деления и метод Ньютона (см., например, [2], [9], [24]).

Для решения систем линейных уравнений рассмотрен метод исключения Гаусса, а методы прогонки, квадратного корня, итераций рассмотрены в задачах.

4.1. Метод итераций

Пусть известно, что отрезок $[a,b]$ содержит единственный корень уравнения

$$f(x) = 0.$$

Метод итераций при определенных ограничениях на функцию $f(x)$ позволяет вычислить корень с заданной точностью.

Применение метода итераций требует предварительного приведения уравнения к виду

$$x = g(x).$$

Способы приведения рассмотрены в [9], отметим, что способ приведения влияет на сходимость метода (см., напр., [24]). Будем предполагать, что

$g(x)$ удовлетворяет условию Липшица с постоянной $q<1$, в этом случае метод итераций сходится и скорость сходимости оценивается неравенством

$$\left|x^* - x_n\right| \le \frac{e}{1-q} q^n,$$

где $\left|x_o - g(x_0)\right| \le e$, x^* — решение уравнения, x_0 — начальное приближение. Метод итераций состоит из следующих шагов.

2. Выбирается начальное приближение x_0.
3. Вычисляется последовательность чисел
$$x_n = g(x_{n-1}), \quad n = 1,2,\ldots.$$
4. Вычисления продолжаются до тех пор, пока для двух последовательных приближений x_n и x_{n-1} не будет обеспечено выполнение неравенства
$$\left|x_n - x_{n-1}\right| < (1-q)e/q,$$
где ***e*** — заданная точность вычислений и $\left|g'(x)\right| < q.$
5. Если количество итераций превысит некоторую заранее установленную величину, считают, что метод не сходится.

Пример 4.1.1. Листинг программы.

```
program Example4_1_1;
{$APPTYPE CONSOLE}
uses   SysUtils;
Const
    MaxN=1000; {макс. число шагов метода}
Type   Funct=function(x:double):double;
 function g(x:double):double;
begin
  Result:=-x*x*x*0.04 + 1.08*x + 0.2;
end;
procedure Iterat(g:TFunct; x0,
                 eps : Double;
 Var x : Double; Var N : Integer);
{Метод итерации:}
Var t:double;
  begin
   N:=1;
   repeat
    x:=g(x0);
    t:=abs(x-x0);
    Inc(N);
    x0:=x;
   until  (t<eps) Or (N>MaxN);
end;{Iterat}
```

```
Var   x:Double; N:Integer;
begin
 Iterat(g,2.5,0.000001,x,N);
 Writeln('Методом Iterat найден корень', x, 'за', N,
'шагов');
  Readln;
End.
```

Метод итераций можно использовать для решения систем уравнений

$$\begin{cases} x = \Phi(x, y), \\ y = \Psi(x, y). \end{cases} \qquad (4.1.1)$$

Выбирается начальное приближение (x_0, y_0), итерационный процесс выполняется по формулам

$$\begin{cases} x_{k+1} = \Phi(x_k, y_k), \\ y_{k+1} = \Psi(x_k, y_k), \end{cases} \quad k = 0,1,\ldots \ . \qquad (4.1.2)$$

Вычисления продолжаются до тех пор, пока изменения неизвестных x и y в двух последовательных итерациях не станут малыми. Это условие не гарантирует достижения выбранной точности приближения и, в данном случае, предложено для упрощения программирования (см., напр., [2], [9]).

Достаточным условием сходимости метода итераций является выполнение неравенств (см., напр., [9])

$$\left|\frac{\partial\Phi}{\partial x}\right| + \left|\frac{\partial\Psi}{\partial x}\right| \le q_1 < 1, \quad \left|\frac{\partial\Phi}{\partial y}\right| + \left|\frac{\partial\Psi}{\partial y}\right| \le q_2 < 1.$$

Вместо итерационного процесса (4.1.2) можно использовать "процесс Зейделя":

$$x_{k+1} = \Phi(x_k, y_k), \quad y_{k+1} = \Psi(x_{k+1}, y_k). \qquad (4.1.3)$$

Приведенные в этом разделе формулы достаточно просто обобщаются на случай систем произвольной размерности (подробности в [2], [9]).

4.2. Метод половинного деления

Согласно этому методу, для уравнения $f(x) = 0$ отыскивается интервал $[u,v]$, в котором $f(x)$ меняет знак. В процессе выполнения алгоритма интервал уменьшается до величины, которая соответствует точности вычислений. Метод состоит из следующих шагов:

1. Положить u = a, v = b. Вычислить $f(u)$ и $f(v)$.
2. Положить $w = (u+v)/2$. Вычислить $f(w)$.
3. Если $f(w) = 0$, закончить вычисления (w является корнем).
4. Если $f(w)$ и $f(u)$ имеют одинаковые знаки, то заменить u на w, в противном случае заменить v на w.

5. Если *v* - *u* > *e* (здесь *e* — заданная точность вычислений), то перейти к шагу 2, в противном случае завершить вычисления (считаем *w* приближенным значением корня).

Пример 4.2.1. Листинг программы.

```
procedure Bisect(c,d,eps:Double;
                 Var x:Double; Var i:Integer);
{Метод половинного деления:}
Var
     u, v, f1, f2 : Double;
begin
   i := 1;
   u := c; v := d;
   f1 := f(u);
   repeat
     x := (u + v) * 0.5;
     f2 := f(x);
     if   f2 = 0  then break;{Выход из цикла}
     if  f1*f2 >0   then
        begin
          u := x;
          f1 := f2;
        end
     else v := x;
     Inc(i);
   until (v - u) < eps;
end; {Bisect}
```

4.3. Метод Ньютона

Предполагается, что на отрезке $[c,d]$ уравнение $f(x)=0$ имеет единственный корень, кроме того, $f'(x)$ и $f''(x)$ непрерывны и сохраняют определенные знаки на отрезке $[c,d]$. Алгоритм состоит из следующих шагов:

1. Выбирается начальное приближение x_0.
2. Вычисляется следующее приближение согласно формуле
$$x_n = x_{n-1} - f(x_{n-1})/f'(x_{n-1}), \quad n=1,2,\ldots$$
или (упрощённый метод Ньютона) по формуле:
$$x_n = x_{n-1} - f(x_{n-1})/f'(x_0), \quad n=1,2,\ldots$$
3. Процедура вычислений приближений продолжается до тех пор, пока $|x_n - x_{n-1}| > e,$ где ***e*** — заданная точность вычислений.

Если количество шагов алгоритма превысит некоторую заранее установленную величину, считают, что метод не сходится.

Пример 4.3.1. Листинг программы.

```
procedure Newton(x0, eps : Double;
                 Var x : Double; Var i : Integer);
{Метод Ньютона}
Var t : Double;
begin
 i:=1;
 repeat
   x := x0 - f(x0)/df(x0);
   t := abs(x - x0);
   Inc(i);
   x0 := x;
 until (t<eps) Or (N>MaxI);
end; {Newton}
```

Метод Ньютона позволяет решать системы уравнений (см. [2], [9]). Приведем расчетные формулы для случая двух уравнений

$$\begin{cases} F(x,y)=0, \\ G(x,y)=0. \end{cases}$$

Выбирается начальное приближение (x_0, y_0). Для уточнения решения используются формулы

$$x_{k+1} = x_k - \frac{1}{J(x_k,y_k)}\begin{vmatrix} F(x_k,y_k) & F'_y(x_k,y_k) \\ G(x_k,y_k) & G'_y(x_k,y_k) \end{vmatrix},$$

$$y_{k+1} = y_k - \frac{1}{J(x_k,y_k)}\begin{vmatrix} F'_x(x_k,y_k) & F(x_k,y_k) \\ G'_x(x_k,y_k) & G(x_k,y_k) \end{vmatrix},$$

где $k = 0,1,\ldots.$. На каждом шаге итерации якобиан должен быть отличным от нуля:

$$J(x_k,y_k) = \begin{vmatrix} F'_x(x_k,y_k) & F'_y(x_k,y_k) \\ G'_x(x_k,y_k) & G'_y(x_k,y_k) \end{vmatrix} \neq 0.$$

4.4. Пример решения нелинейного уравнения

Создадим класс, ориентированный на приближенное решение уравнения. В качестве методов этого класса включим уже рассмотренные процедуры нахождения приближенного решения.

Техника использования объектно-ориентированного программирования в вычислительных задачах описана в [18].

Пример 4.4.1. Листинг программы.

```
program Project4_4_1;
{$APPTYPE CONSOLE}
{Класс для решения уравнений}
```

```
Const
  MaxN=1000; {макс. число шагов метода}
Type  TFunct=function(x:double):double;
Type
 TEquation=class
            f: TFunct; {f(x)=0}
            g: TFunct;
          {x-g(x)=0 эквивалентно f(x)=0}
            Df: Tfunct; {производная функции f(x)}
            a,b: double; {границы отрезка}
            x0: double; {прибл. значение корня}
            eps: double; {точность вычислений}
            N: Integer; {Кол. шагов прибл. метода}
   {общий для всех методов конструктор:}
 constructor Create(f_,g_,Df_:TFunct;
                    a_,b_,x0_,eps_:double);overload;
  {конструктор для метода половинного деления:}
 constructor Create(f_:TFunct;
                    a_,b_,eps_:double);overload;
  {конструктор для метода итераций:}
 constructor Create(f_,g_:TFunct;
                    x0_,eps_:double);overload;
  {конструктор для метода Ньютона:}
 constructor Create(x0_,eps_:double;
                    f_,Df_:TFunct);overload;
  destructor Destroy;
  function Bisect:double;
  function Iterat:double;
  function Newton:double;
  function GetN:Integer;
 end;
constructor TEquation.Create(f_,g_,Df_:TFunct;
                         a_,b_,x0_,eps_:double);
 {общий конструктор}
 begin
     f:=f_; g:=g_; Df:=Df_;
     a:=a_; b:=b_;  x0:=x0_; eps:=eps_;
 end;
constructor TEquation.Create(f_:TFunct;
                 a_,b_,eps_:double);
 {конструктор для метода половинного деления}
 begin
     f:=f_;
     a:=a_; b:=b_; eps:=eps_;
 end;
```

```
constructor TEquation.Create(f_,g_:TFunct;
                      x0_,eps_:double);
{конструктор для метода итераций}
 begin
     f:=f_; g:=g_;
     x0:=x0_; eps:=eps_;
 end;
constructor TEquation.Create(x0_,eps_:double;
            f_,Df_:TFunct);
{конструктор для метода Ньютона}
 begin
     f:=f_; Df:=Df_;
     x0:=x0_; eps:=eps_;
 end;
destructor TEquation.Destroy;
 begin
 end;
function TEquation.Bisect:double;
 {Метод половинного деления:}
Var  u, v, f1, f2 : double;
begin
  N := 1;
  u := a; v := b;
  f1 := f(u);
  repeat
   x0 := (u + v) * 0.5;
   f2 := f(x0);
   if   f2 = 0  then break;
   if f1*f2 >0 {т.е. у f1 и f2 одинаковые знаки}
    then
     begin
      u := x0; f1 := f2;
     end
    else v := x0;
   Inc(N);
  until (v - u) < eps;
  Result := x0;
end; {Bisect}
function TEquation.Iterat:double;
   {Метод итераций}
Var x,t:double;
begin
  N := 1;
  repeat
   x := g(x0);
   t := abs(x-x0);
```

```
   Inc(N);
   x0 := x;
  until  (t<eps) Or (N>MaxN);
  Result := x;
end;{Iterat}
function TEquation.Newton:double;
 {Метод Ньютона}
Var t,x:double;
begin
  N := 1;
  repeat
   x := x0-f(x0)/df(x0);
   t := abs(x-x0);
   Inc(N);
   x0 := x;
  until (t<eps) Or (N>MaxN);
  Result := x;
 end; {Newton}
 function TEquation.GetN:Integer;
   begin
    Result := N;
 end;
 { Пример уравнения  }
function f(x:double):double;
{Левая часть уравнения:}
begin
  Result := x*x*x - 2*x - 5;
end;
function g(x:double):double;
{x=g(x) равносильно f(x)=0}
{используется в методе итераций}
begin
  Result := -x*x*x*0.04 + 1.08*x + 0.2;
end;
function df(x:double):double;
{Производная функции f(x)}
{Используется в методе Ньютона}
begin
  Result := 3*x*x - 2;
end;
Var Eq: TEquation;
Begin
  Eq := TEquation.Create(f,2,3,0.000001);
  Writeln('Методом Bisect найден корень',
           Eq.Bisect,'за',Eq.GetN,'шагов');
```

```
  Eq := TEquation.Create(f,g,2.5,0.000001);
  Writeln('Методом итераций найден корень',
           Eq.Iterat, 'за',Eq.GetN,'шагов');
  Eq:= TEquation.Create(2.5,0.000001,f,df);
  Writeln('Методом Ньютона найден корень',
           Eq.Newton,'за',Eq.GetN,'шагов');
Readln; {задержка}
End.
```

Для каждого из численных методов использовался отдельный конструктор `Create()`. Поскольку для метода итераций и метода Ньютона необходимо передать два параметра типа `double` и два параметра типа `TFunct`, пришлось произвести перестановку этих параметров, чтобы сигнатуры конструкторов были различными. Можно использовать общий конструктор, в этом случае основная программа имеет вид

```
Var Eq: TEquation;
Begin
 Eq:= TEquation.Create(f,g,df,2,3,2.5,0.000001);
 Writeln('Методом Bisect найден корень',
          Eq.Bisect,'за',Eq.GetN,'шагов');
 Writeln('Методом итерации найден корень',
          Eq.Iterat,'за',Eq.GetN,'шагов');
 Writeln('Методом Ньютона найден корень',
          Eq.Newton,'за',Eq.GetN,'шагов');
Readln; {задержка}
End.
```

Замечание. В операторах **Writeln()** содержится текст на русском языке, который при запуске приложения будет отражен в ANSI – кодировке (“нечитаемой” в DOS). В разделе 1.2 указано, как решить эту проблему.

4.5. Решение систем линейных уравнений

4.5.1. Метод Гаусса

Наиболее распространенным методом решения систем линейных алгебраических уравнений является метод последовательного исключения неизвестных (см., напр., [2], [9], [24]). Обычно этот метод называют методом исключения Гаусса. Рассматривается система

$$a_{11}\,x_1 + a_{12}\,x_2 + \ \ldots\ + a_{1n}\,x_n = b_1$$
$$a_{21}\,x_1 + a_{22}\,x_2 + \ \ldots\ + a_{2n}\,x_n = b_2$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_{n1}\,x_1 + a_{n2}\,x_2 + \ldots + a_{nn}\,x_n = b_n.$$

Алгоритм решения состоит из двух этапов: ***прямого хода*** и ***обратной подстановки***.

Прямой ход. Сначала с помощью первого уравнения исключается x_1 из остальных уравнений системы. Затем с помощью второго уравнения из последующих уравнений исключается x_2. На ***k***-м шаге кратные ***k***-го уравнения вычитаются из оставшихся уравнений для исключения x_k. В результате за $n-1$ шагов матрица системы будет приведена к треугольному виду. В процессе исключения неизвестных приходится производить деления на коэффициенты a_{kk}. Если на ***k***-м шаге $a_{kk}=0$, то необходимо произвести перестановку последующих уравнений системы. Схема с ***выбором ведущего элемента*** предполагает, что на ***k***-м шаге в ***k***-м столбце матрицы системы отыскивается наибольший по модулю элемент и производится перестановка уравнений так, чтобы этот элемент оказался на месте элемента a_{kk}.

После выполнения прямого хода получим систему

$$\begin{aligned}
\tilde{a}_{11}x_1+\tilde{a}_{12}x_2+\cdots+\tilde{a}_{1n}x_n&=\tilde{b}_1,\\
\tilde{a}_{22}x_2+\cdots+\tilde{a}_{2n}x_n&=\tilde{b}_2,\\
\cdots\quad\cdots\quad\cdots\quad\cdots\quad\cdots\quad\cdots&\\
\tilde{a}_{nn}x_n&=\tilde{b}_n.
\end{aligned}$$

Обратный ход. Состоит в последовательном вычислении неизвестных: решая последнее уравнение, находим

$$x_n=\tilde{b}_n/\tilde{a}_{nn},$$

затем, используя это значение, из предпоследнего уравнения находим x_{n-1} и т.д. Расчетная формула для *k*-го неизвестного имеет вид

$$x_k=\frac{1}{\tilde{a}_{kk}}\left(\tilde{b}_k-\tilde{a}_{k\,k+1}x_{k+1}-\cdots-\tilde{a}_{k\,n}x_n\right),\quad k=n-1,\ldots,1.$$

Пример 4.5.1 Листинг программы.

```
program Project4_5_1;
{$APPTYPE CONSOLE}
{Решение системы линейных уравнений методом Гаусса}
uses   SysUtils;
Type
     TDynMatr=Array of array of double;
Procedure Solve(a:TDynMatr;b:Array of double;
        Var x:Array of double; var Flag: Integer);
Var
    max_k, t:double;
    i,j,k,m,n:integer;
begin
 Flag:=0;{система разрешима, если Flag=0}
 n:=Length(b); {порядок системы}
```

```
 if n>1 then
 begin {1}
  n:=n-1;
  for k:=0 to n-1 do
   begin {2}
   { Найдем ведущий элемент в k-ом столбце:}
     max_k:=a[k,k];m:=k;
     for i:=k+1 to n do
      if abs(max_k)<abs(a[i,k]) then
       begin
        m:=i;max_k:=a[m,k];
       end;
{Ведущий элемент расположен в m-ой строке.
 Система не разрешима, если этот элемент нулевой}
    if abs(max_k)>0.00000001 then
     begin {3}
      if m<>k then {перестановка строк m и k:}
      begin {4}
       for j:=k to n do
        begin
         t:=a[m,j];a[m,j]:=a[k,j];a[k,j]:=t;
        end;
       t:=b[m];b[m]:=b[k];b[k]:=t;
      end;{4}
   { Делим k-ю строку на max_k:}
      for i:=k+1 to n do
       a[k,i]:=a[k,i]/max_k;
      b[k]:=b[k]/max_k;
   { Исключение по столбцам: }
      for i:=k+1 to n do
       begin
        for j:=k+1 to n do
         a[i,j]:=a[i,j]-a[k,j]*a[i,k];
        b[i]:=b[i]-b[k]*a[i,k];
       end;
     end  {3}
    else
      begin
        Flag := m; Exit; {Выходим из процедуры}
      end;
  end;{2}
  {Обратный ход:}
   if abs(a[n,n])>0 then
    begin {2_}
     b[n]:=b[n]/a[n,n];
     x[n]:=b[n];
```

```
     for i:=n-1 downto 0 do
      begin {3_}
       t:=0;
       for j:=i+1 to n do
        t:=t+a[i,j]*x[j];
       x[i]:=b[i]-t;
      end;{3_}
    end  {2_}
   else
    Flag:=n;
  end {1}
 else
  if abs(a[0,0]) >0 then
       x[0]:=b[0]/a[0,0]
  else Flag:=1;
End; {Solve}

Var a:TDynMatr; b:array of double;
    x:array of double;
    i,N:Integer; Flag:Integer;
Begin {Основная программа}
  N:=3; {порядок системы}
  SetLength(a,N,N);
  SetLength(b,N);
  SetLength(x,N);
  {Пример системы:}
  a[0,0]:=1;a[0,1]:=1;a[0,2]:=1;
  a[1,0]:=2;a[1,1]:=3;a[1,2]:=1;
  a[2,0]:=1;a[2,1]:=-1;a[2,2]:=-1;
  b[0]:=4;b[1]:=9;b[2]:=-2;
  Solve(a,b,x,Flag);
  if Flag<>0 then writeln('Error ',Flag)
  else for i:=0 to High(x) do
       writeln('x[',i:2,']=', x[i]);
  readln;
End.
```

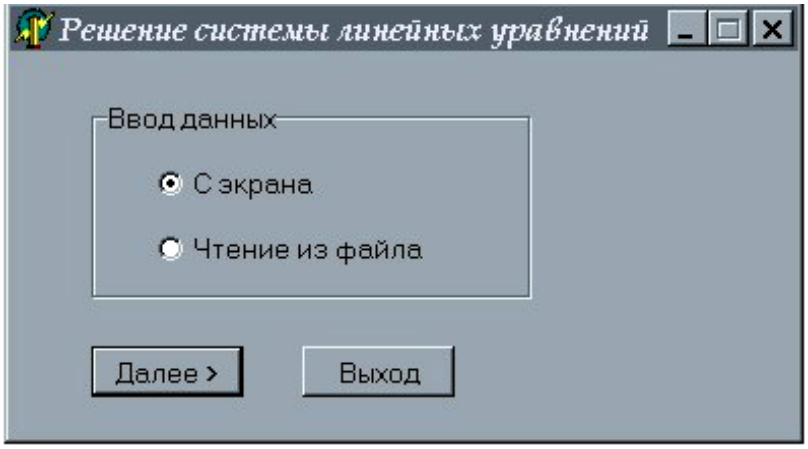
4.5.2. Замечания по проектированию интерфейса приложения

При проектировании интерфейса приходится решать вопросы, связанные с вводом коэффициентов системы, записью и печатью результатов, а также выводом информации о ходе выполнения вычислений.

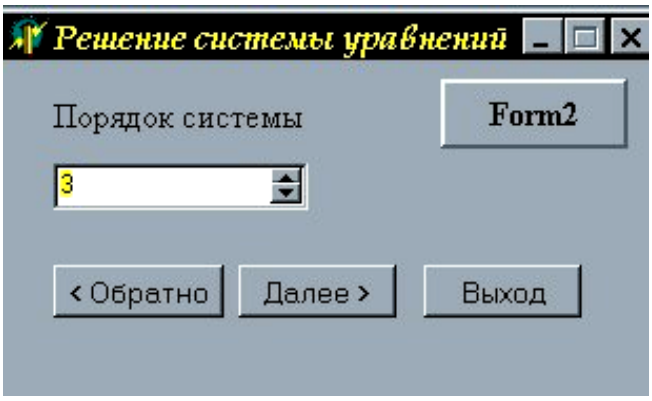
Можем организовать интерфейс приложения, например, в виде “мастера”. Переход на следующий шаг мастера осуществляется выбором

кнопки “**Далее>**”. Кнопка “**<Обратно**” позволяет возвратиться на шаг назад, в случае каких-либо ошибочных действий на текущем шаге.

На первом шаге предлагается выбрать способ ввода данных. Данные можно прочитать из файла или ввести непосредственно во время работы программы.

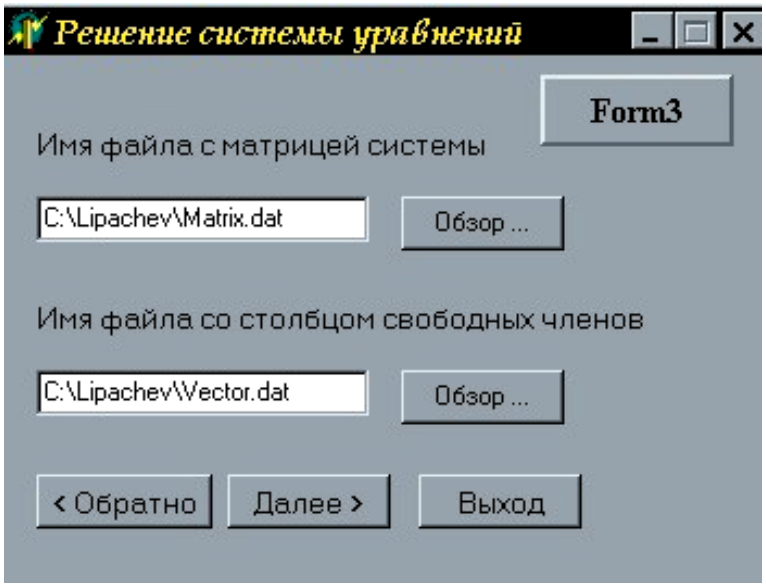


В случае выбора радиокнопки “**С экрана**” предлагается ввести порядок системы уравнений:



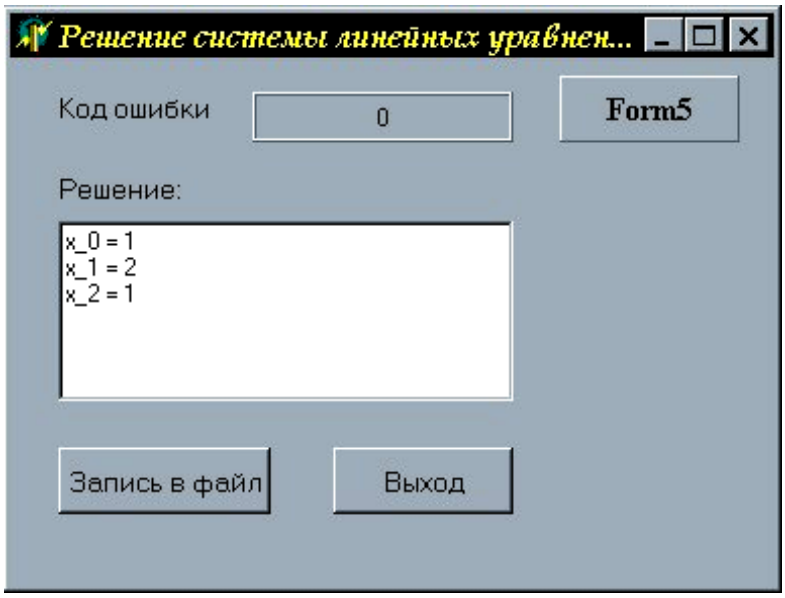
Затем, после выбора кнопки “**Далее>**”, выводится окно для ввода системы уравнений. Данные заносятся в ячейки соответствующих таблиц. Размер таблиц регулируется значением “**Порядок системы**”, установленным на предыдущем шаге. Данная версия программы не допускает ячеек без значений (“пустых” ячеек), хотя незначительная доработка позволит внести эту возможность в программу, после чего можно будет не вводить нулевые значения в таблицы.

Если на первом шаге была отмечена радиокнопка "**Чтение из файла**", то на следующем шаге последует предложение указать файлы с данными:



Имена файлов можно ввести с клавиатуры или воспользоваться кнопкой "**Обзор**...", выбор которой откроет стандартный диалог открытия файлов. Файлы с данными можно подготовить с помощью любого текстового редактора. В приведенном приложении предполагается, что значения матрицы системы записаны построчно, с использованием пробела в качестве разделителя значений. Столбец свободных членов записан в отдельном файле, причем числа записаны по одному в строке.

После выбора кнопки "**Далее>**" в окнах "**Form3**" или "**Form4**" система линейных уравнений будет решена и результаты выведены в окно:



Описание глобальных переменных и код алгоритма исключения разместим в отдельном модуле **UnitDat**, который создадим с помощью команды меню **File|New|Unit**.

```
unit UnitDat;
interface
Type
  TDynMatr= Array of Array of Double;
Var
   a: TDynMatr;
   b: Array of Double;
   x: Array of Double;
   N: Integer;
   Flag: Integer;
Procedure Solve(a:TDynMatr;b:Array of double;
        Var x:Array of double; var Flag: Integer);
implementation
Procedure Solve(a:TDynMatr;b:Array of double;
   Var x:Array of double; var Flag: Integer);
Var max_k,t:real;
    i,j,k,m,n:integer;
begin
  Flag:=0;
  n:=Length(b);
  if n>1 then
   begin {1}
    n:=n-1;
    for k:=0 to n-1 do
     begin {2}
      max_k:=a[k,k];m:=k;
```

```
      for i:=k+1 to n do
       if abs(max_k)<abs(a[i,k]) then
        begin
         m:=i;max_k:=a[m,k];
        end;
        if abs(max_k)>0.00000001 then
         begin {3}
          if m<>k then
           begin {4}
            for j:=k to n do
             begin
              t:=a[m,j];
              a[m,j]:=a[k,j];
              a[k,j]:=t;
             end;
             t:=b[m];b[m]:=b[k];b[k]:=t;
           end;{4}
          for i:=k+1 to n do
           a[k,i]:=a[k,i]/max_k;
          b[k]:=b[k]/max_k;
          for i:=k+1 to n do
           begin
            for j:=k+1 to n do
             a[i,j]:=a[i,j]-a[k,j]*a[i,k];
            b[i]:=b[i]-b[k]*a[i,k];
           end;
         end  {3}
        else     Flag:=m;
     end;{2}
     if abs(a[n,n])>0 then
      begin {2_}
        b[n]:=b[n]/a[n,n];
        x[n]:=b[n];
        for i:=n-1 downto 0 do
          begin {3_}
            t:=0;
            for j:=i+1 to n do
             t:=t+a[i,j]*x[j];
            x[i]:=b[i]-t;
          end;{3_}
     end  {2_}
      else   Flag:=n;
    end {1}
   else x[0]:=b[0]/a[0,0];
 End; {Solve}
End.
```

Во всех формах проекта присутствует кнопка “**Выход**”, выбор которой должен приводить к закрытию приложения. В обработчики этих кнопок запишем всего одну строку

```
Application.Terminate; {Завершить приложение}
```

При выборе кнопки “**Далее>**” необходимо открыть следующее окно мастера и передать значения параметров, определенных на данном шаге мастера.

В форме **Form1** с помощью радиокнопок организуется выбор способа ввода данных. Обработчик кнопки “**Далее>**” этой формы имеет вид

```
procedure TForm1.BitBtn2Click(Sender:TObject);
  begin
     if Form1.RadioButton1.Checked then
        Form2.ShowModal
     else
        Form3.ShowModal;
  end;
```

Для формы **Form2** в обработчик кнопки “**Далее>**” запишем

```
procedure TForm2.BitBtn2Click(Sender:TObject);
  begin
  UnitDat.N:=Form2.SpinEdit1.Value;       {размерность
системы}
  Form4.StringGrid1.ColCount := UnitDat.N;
  Form4.StringGrid1.RowCount := UnitDat.N;
  Form4.ShowModal;
 end;
```

С помощью компонента **SpinEdit** (находится на странице **Samples**) в форме **Form2** пользователю предлагается установить значение порядка системы уравнений. Значение можно установить с помощью кнопок этого компонента или отредактировать вручную. Введенное значение можно “узнать” с помощью свойства **Value**.

Компонент **StringGrid1** в форме **Form4** используется для ввода элементов матрицы системы, а компонент **StringGrid2** — для ввода столбца свободных членов. Эти компоненты находятся на странице **Additional**. Для свойств **FixedCols** (количество фиксированных столбцов) и **FixedRows** (количество фиксированных строк) этих компонент установим нулевые значения. Значения свойств **ColCount** (число столбцов таблицы) и **RowCount** (число строк) устанавливаются в обработчике кнопки “**Далее>**” формы **Form2**, в зависимости от выбранного размера системы уравнений. Чтобы сделать таблицы

редактируемыми, установим значение **True** у опций **goEditing** и **goAlwaysShowEditor** свойства **Options**. Опция **goEditing**, установленная в **True**, разрешает редактирование ячеек. Опция **goAlwaysShowEditor**, установленная в **True**, позволяет редактировать содержимое ячейки после выбора ячейки щелчком мыши, значение **False** означает, что редактирование возможно после двойного щелчка мыши или нажатия клавиши **F2**. Содержимое ячеек таблицы доступно с помощью свойства **Cells[j,i]**, при этом первым параметром указывается номер столбца, а вторым — номер строки таблицы.

В обработчике кнопки "**Далее>**" формы **Form4** с помощью свойства **Cells** значения из таблиц передаются в динамические массивы **a** (матрица системы) и **b** (столбец свободных членов). Полный код обработчика этой кнопки имеет вид

```
procedure TForm4.BitBtn2Click(Sender: TObject);
Var i,j, m: Integer;   s:String;
begin
 Form1.Hide;
 Form4.Hide;
 {Выделяем память под данные:}
 m:=UnitDat.N;
 SetLength(UnitDat.a,m,m);{матрица системы}
 SetLength(UnitDat.b,m); {свободные члены}
 SetLength(UnitDat.x,m); {вектор решений}
 {содержимое ячеек таблиц переносим в a и b}
 for i:= 0 to m-1 do
  for j:=0 to m-1 do
   UnitDat.a[i,j]:=
    StrToFloat(Form4.StringGrid1.Cells[j,i]);
 for i:=0 to m-1 do
  UnitDat.b[i]:=
   StrToFloat(Form4.StringGrid2.Cells[0,i]);
{Решаем систему:}
 UnitDat.Solve(UnitDat.a,UnitDat.b,
               UnitDat.x, UnitDat.Flag);
{решения помещаем в окно Form5.Memo1.}
 Form5.Memo1.Lines.Clear;
 for i:=0 to m-1 do
  begin
  s:='x_'+IntToStr(i)+'='+FloatToStr(UnitDat.x[i]);
  Form5.Memo1.Lines.Insert(i,s);
 end;
 Form5.Panel1.Caption:=IntToStr(UnitDat.Flag);
 Form5.ShowModal;
end;
```

В форме **Form3** предлагается другой вариант ввода данных. Матрица системы и столбец свободных членов вводятся из текстовых файлов. Предполагается, что значения столбца свободных членов записаны по одному в строке, а элементы матрицы записаны в другом файле, причем одна строка файла соответствует строке матрицы, а числа разделены пробелами. Подготовить такие файлы можно с помощью любого текстового редактора. Имена файлов можно ввести в поля **Edit1** и **Edit2** вручную или с помощью кнопок "**Обзор**...", выбор которых открывает стандартный диалог "**Открыть файл**". Для этого размещаем на форме две компоненты **OpenDialog** со страницы **Dialogs**. Обработчик первой кнопки "**Обзор**..." организует диалог для выбора файла со значениями матрицы системы и состоит из операторов

```
procedure TForm3.Button1Click(Sender: TObject);
 begin
  OpenDialog1.Title:='Файл с матрицей системы';
  if OpenDialog1.Execute then
   Edit1.Text:=OpenDialog1.FileName;
 end;
```

Соответственно обработчик для второй кнопки "**Обзор**..." имеет вид

```
procedure TForm3.Button2Click(Sender: TObject);
begin
 OpenDialog2.Title:='Файл со свободными членами';
 if OpenDialog2.Execute then
  Edit2.Text:=OpenDialog2.FileName;
 end;
```

С помощью Object Inspector установим следующее значение свойства **Filter** для компонент **OpenDialog**:

Файлы данных (*.dat)|*.dat|Все файлы (*.*)|*.*

Обработчик кнопки "**Далее>**" формы **Form3** содержит операторы чтения данных, вызов процедуры решения и вывод результатов.

```
procedure TForm3.BitBtn2Click(Sender: TObject);
Var
    NameFile_a, NameFile_b:String;
    F_a, F_b :TextFile;
    i, j, m : Integer;  t: double; s:String;
begin
{Узнаем имена файлов с данными:}
  NameFile_a:=Form3.Edit1.Text;
  NameFile_b:=Form3.Edit2.Text;
  {Узнаем порядок системы:}
  AssignFile(F_b,NameFile_b);
```

```
 try
  Reset(F_b);
  m:=0;
  While Not Eof(F_b) do
   begin
     m:=m+1;
     Readln(F_b,t);
   end;
  CloseFile(F_b);
  if m<>0 then
  begin {m}
   {Выделяем память под данные}
   SetLength(UnitDat.a,m,m);{матрица системы}
   SetLength(UnitDat.b,m); {свободные члены}
   SetLength(UnitDat.x,m); {вектор решений}
  {Читаем данные из файлов:}
   AssignFile(F_a,NameFile_a);
   Reset(F_a);
   for i:=0 to m-1 do
   begin
    for j:=0 to m-1 do Read(F_a,UnitDat.a[i,j]);
    Readln(F_a);{перешли на следующую строку}
   end;
   CloseFile(F_a);
   AssignFile(F_b,NameFile_b);
   Reset(F_b);
   for i:=0 to m-1 do
    Readln(F_b,UnitDat.b[i]);
   CloseFile(F_b);
  {Решаем систему:}
  UnitDat.Solve(UnitDat.a,UnitDat.b,
                UnitDat.x, UnitDat.Flag);
  {решения помещаем в окно Form5.Memo1:}
 Form5.Memo1.Lines.Clear;
 for i:=0 to m-1 do
  begin
   s:='x_'+IntToStr(i)+'='+FloatToStr(UnitDat.x[i]);
  Form5.Memo1.Lines.Insert(i,s);
  end;
 s:='m='+IntToStr(m);
 Form5.Memo1.Lines.Insert(m,s);
 Form5.Panel1.Caption:=IntToStr(UnitDat.Flag);
 Form1.Hide;
 Form3.Hide; Form3.Close;
 Form5.ShowModal;
end; {m}
```

```
 except
  on E:EInOutError do
   ShowMessage('Ошибка #'+IntToStr(E.ErrorCode)+
   'при работе с файлом'+NameFile_b);
 end;
end;
```

Форма **Form5** служит для отображения результатов программы. В компоненту **Panel1** передается код ошибки, для этого используются операторы

```
Form5.Panel1.Caption:=IntToStr(UnitDat.Flag);
```

обработчиков кнопки "**Далее>**" форм **Form3** и **Form4**. Если код ошибки нулевой, то в компоненту **Memo1** записываются решения системы. Свойство **Lines** этого компонента является основным и содержит текст окна в виде списка строк типа **TStrings**. Строки нумеруются с 0 и для доступа к информации, содержащейся в строке с номером **m**, можем использовать оператор **Memo1.Lines[m]**. Для добавления новой строки **s** в позицию **m** списка используется метод **Memo1.Lines.Insert(m,s)**. Перед выводом результатов с помощью метода **Memo1.Lines.Clear()** производится очистка окна **Memo**. В форме **Form5** необходимо разместить компоненту **SaveDialog**, с помощью которой организуется диалог выбора файла для записи результатов. Запись производится после выбора кнопки "**Запись в файл**" и обработчик этой кнопки имеет вид

```
procedure TForm5.BitBtn1Click(Sender: TObject);
  Var f:TextFile; i:Integer;
  begin
   if SaveDialog1.Execute then
    begin
     AssignFile(f,SaveDialog1.FileName);
     Rewrite(f);
     for i:=0 to Memo1.Lines.Count-1 do
     writeln(f,Memo1.Lines[i]);
     CloseFile(f);
    end;
  end;
```

При выборе кнопки "**<Обратно**" фокус ввода передается на форму предшествующего шага, а текущая форма закрывается. Приведем обработчики для этой кнопки каждой из форм.

```
procedure TForm2.BitBtn1Click(Sender: TObject);
  begin
   {Кнопка "<Обратно" формы Form2}
    Form2.Close;
```

```
    Form1.SetFocus;
  end;

procedure TForm3.BitBtn1Click(Sender: TObject);
  begin
   {Кнопка "<Обратно" формы Form3}
   Form3.Close;
   Form1.SetFocus;
  end;

procedure TForm4.BitBtn1Click(Sender: TObject);
  begin
   {Кнопка "<Обратно" формы Form4}
   Form4.Hide; Form4.Close;
   Form2.Show;
  end;
```

4.5.3. Объектно-ориентированный подход

На примере консольного приложения покажем использование технологии объектно-ориентированного программирования для реализации алгоритма решения систем линейных уравнений. Для работы с элементами системы в классе `TLinEquation` используются динамические массивы (на самом деле, создается только один массив — для хранения решения системы, а для матрицы системы и столбца свободных членов при вызове конструктора передаются адреса уже созданных массивов). Вместо динамических массивов можно использовать указатели или специальные классы, например, классы динамический вектор и динамическая матрица, введенные в работе [18].

Пример 4.5.3. Листинг программы.

```
program Project4_5_3;
{$APPTYPE CONSOLE}
 {Решение системы лин. ур. методом исключения}
 {Объектно-ориентированный подход}
uses   SysUtils;
Type
   TDynMatr=Array of array of double;
   TVect=Array of double;
Type
  TLinEquation=class
               NEq:Integer;  {Порядок системы}
               Flag:Integer; {Код ошибки}
               a:TDynMatr;  {Матрица системы}
               b:TVect; {Столбец свободных членов}
               x:TVect; {Решение системы}
```

```
          constructor Create(N_:Integer;
                  a_:TDynMatr;b_:TVect);overload;
          procedure Solve;
          function GetX(i:Integer):double;
          function GetXP: Pointer;
          function GetFlag:Integer;
     end;
constructor TLinEquation.Create(N_:Integer;
            a_:TDynMatr;b_:TVect);
begin
  NEq := N_;
  a := a_;
  b := b_;
  SetLength(x,N);
end;
procedure TLinEquation.Solve;
Var
   max_k,t:double;
   i,j,k,m,n:integer;
begin
  Flag:=0;
  if NEq>1 then
  begin {1}
   n := NEq-1;
 {…  …  …}
 {Код этого метода совпадает с кодом процедуры Solve()
  примера 4.5.1}
 {…  …  …}
end; {Solve}
function TLinEquation.GetX(i:Integer):double;
begin
  Result := x[i];
end;
function TLinEquation.GetXP:Pointer;
begin
  Result := x;
end;
function TLinEquation.GetFlag:Integer;
begin
  Result := Flag;
end;

Var a_:TDynMatr;b_:TVect;
    i,N:Integer;
    Eq:TLinEquation;
```

```
Begin {Основная программа}
  N:=3;
  SetLength(a_,N,N);
  SetLength(b_,N);
  a_[0,0]:=1;a_[0,1]:=1;a_[0,2]:=1;
  a_[1,0]:=2;a_[1,1]:=3;a_[1,2]:=1;
  a_[2,0]:=1;a_[2,1]:=-1;a_[2,2]:=-1;
  b_[0]:=4;b_[1]:=9;b_[2]:=-2;
  Eq := TLinEquation.Create(N,a_,b_);
  Eq.Solve;
  if Eq.GetFlag =0 then
   begin {1}
    for i:=0 to N-1 do
      writeln('x[',i:2,']=',Eq.GetX(i));
   end {1}
  else Writeln('Error Flag=',Eq.GetFlag);
  Readln;
End.
```

В основной программе для доступа к вектору решений можно также использовать метод **GetXP()**. Для этого необходимо объявить динамический массив

```
Var x:TVect;
```

и изменить блок **begin {1} . . . end {1}** следующим образом

```
begin {1}
  x:=Eq.GetXP;
  for i:=0 to N-1 do
     writeln('x_[',i:2,']=',x[i]);
 end {1}
```

4.6. Вычисление определителя матрицы

Метод исключения можно применить для вычисления определителя матрицы. Достаточно выполнить прямой ход метода, убрать операции со столбцом свободных членов и вычислить знак определителя.

Прямой ход состоит из операций перестановки строк, деления строк на их ведущие элементы и вычитания строк. Операция вычитания из одной строки матрицы линейной комбинации других строк не изменяет определителя матрицы. При делении строки матрицы на число определитель также делится на это число. Перестановка любых строк матрицы меняет знак определителя.

После прямого хода метода исключений матрица приводится к треугольному виду, поэтому, в обозначениях пункта 4.5.1, имеем

$$\det A = (-1)^k \tilde{a}_{11} \cdot \tilde{a}_{22} \cdot \ldots \cdot \tilde{a}_{nn},$$

где k — количество перестановок.

Программирование операций вычисления определителя предлагается в качестве упражнения.

4.7. Вычисление обратной матрицы

Пусть $A=\left(a_{ij}\right)$ невырожденная матрица. Для вычисления обратной матрицы $A^{-1}=\left(y_{ij}\right)$ можно использовать метод исключения Гаусса. Равенство $A\cdot A^{-1}=E$, где E — единичная матрица, приводит к системе

$$\sum_{k=1}^{n} a_{ik}y_{kj}=\delta_{ij}\equiv\begin{cases}1, i=j,\\ 0, i\neq j,\end{cases}\quad i,j=1,2,\ldots,n.$$

Это равенство означает, что для нахождения элементов обратной матрицы необходимо решить n систем линейных уравнений с одинаковой матрицей A, но с различными правыми частями. Прямой ход метода исключений проводится только один раз. Затем требуется n раз выполнить обратный ход, предварительно пересчитав правые части, учитывая, при этом, перестановки, связанные с выбором ведущих элементов.

Приведем текст программы, реализующий обращение матрицы. Элементы обратной матрицы обозначены через **`y[i,j]`**. Поскольку эти переменные участвуют только в обратном ходе, в целях сокращения памяти, элементы **`y[i,j]`** используются для хранения правых частей систем уравнений. Поэтому в начале работы массив y совпадает с единичной матрицей **`E`**.

Пример 4.7. Листинг программы.

```
program InverseMatr;
{$APPTYPE CONSOLE}
uses
  SysUtils;
Type
   TDynMatr=Array of array of double;
procedure Inverse(a:TDynMatr;{Исходная матрица}
                  var y:TDynMatr;{Обратная к a}
                  var Flag:Integer);
Var
   max_k,t:double;
   i,j,k,m,n:integer;
begin
 Flag:=0;
 n:=Length(a); {порядок системы}
 if n>1 then
  begin {1}
  n:=n-1;
```

```
for i := 0 to n do
 begin
   for j :=0 to n do  y[i,j]:=0;
   y[i,i] := 1;
 end;
for k:=0 to n-1 do
 begin {2}
 {Найдем ведущий элемент  k-го шага:}
    max_k:=a[k,k];m:=k;
    for i:=k+1 to n do
     if abs(max_k)<abs(a[i,k]) then
       begin
        m:=i;max_k:=a[m,k];
       end;
{Ведущий элемент расположен в m-ой строке,
 если он равен 0, то система не разрешима}
    if abs(max_k)>0.00000001 then
     begin {3}
      if m<>k then
        {перестановка строк m и k:}
         begin {4}
          for j:=k to n do
            begin
            {перестановка в матрице a:}
              t:=a[m,j];a[m,j]:=a[k,j];
              a[k,j]:=t;
            end;
          for j := 0 to n do
            begin
            {перестановка в массиве y:}
              t:=y[m,j];y[m,j]:=y[k,j];
              y[k,j]:=t;
            end;
         end;{4}
 {Делим k-ю строку матрицы a на max_k:}
         for j:=k+1 to n do
           a[k,j]:=a[k,j]/max_k;
 {Делим k-ю строку массива y на max_k:}
         for j:= 0 to n do
          y[k,j]:=y[k,j]/max_k;
 {Исключение по столбцам:}
         for i:=k+1 to n do
           begin
            for j:=k+1 to n do
              a[i,j]:=a[i,j]-a[k,j]*a[i,k];
```

```
              for j:=0 to n do
               y[i,j]:=y[i,j]-y[k,j]*a[i,k];
             end;
            end  {3}
            else
             begin
               Flag:=m;  Exit;
             end;
          end;{2}
  {Вычисление элементов обратной матрицы.
   Для каждого k=0,...,n  проводим обратный ход,
   используя в качестве правых частей системы
   k-й столбец массива y}
  for k :=0 to n do
   begin {1_}
    if abs(a[n,n])>0 then
     begin {2_}
        y[n,k]:=y[n,k]/a[n,n];
        for i:=n-1 downto 0 do
          begin {3_}
            t:=0;
            for j:=i+1 to n do
              t:=t+a[i,j]*y[j,k];
            y[i,k]:=y[i,k]-t;
          end;{3_}
     end  {2_}
      else
       begin
        Flag:=n; Exit;
       end
     end;{1_}
    end {1}
   else
     if abs(a[0,0])>0 then y[0,0]:=1/a[0,0]
     else Flag:=1;
end; {Inverse}

Var
   a,b:TDynMatr;
   i,j,N:Integer; Flag:Integer;
begin
  N:=4; {порядок системы}
  SetLength(a,N,N);
  SetLength(b,N,N);
  a[0,0]:=1.8;a[0,1]:=-3.8;a[0,2]:=0.7; a[0,3]:=-3.7;
  a[1,0]:=0.7;a[1,1]:=2.1; a[1,2]:=-2.6;a[1,3]:=-2.8;
```

```
  a[2,0]:=7.3;a[2,1]:=8.1; a[2,2]:=1.7; a[2,3]:=-4.9;
  a[3,0]:=1.9;a[3,1]:=-4.3;a[3,2]:=-4.9;a[3,3]:=-4.7;
  {Вычисление обратной матрицы:}
  Inverse(a,b,Flag);
  if Flag<>0 then writeln('Error ',Flag)
  else
    for i:=0 to High(b) do
      begin
       for j:=0 to High(b[i]) do
         write(b[i,j]:7:5,'  ');
       writeln;
      end;
readln;
End.
```

4.8. Задачи

4.8.1. Составить программу решения уравнения $f(x)=0$ методом секущих. В этом методе выбирают два начальных приближения x_0 и x_1. Последующие приближения вычисляются по формуле

$$x_{k+1} = x_k - \frac{f_k}{(f_{k-1} - f_k)/(x_{k-1} - x_k)},$$

где $f_k = f(x_k)$.

4.8.2. Составить программу решения системы двух нелинейных уравнений методами итераций и Ньютона.

4.8.3. Составить процедуру вычисления определителя матрицы произвольного порядка на основе прямого хода метода исключения.

4.8.4. ***Метод прогонки*** (см., напр., [9, 24]) является модификацией метода исключения для случая трехдиагональных матриц

$$\begin{aligned}
b_1x_1 + c_1x_2 &= d_1,\\
a_1x_1 + b_2x_2 + c_2x_3 &= d_2,\\
\dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots\ \dots&\\
a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n &= d_{n-1},\\
a_nx_{n-1} + b_nx_n &= d_n.
\end{aligned}$$

На первом этапе (***прямая прогонка***) вычисляются прогоночные коэффициенты

$$A_1 = -c_1/b_1, \quad B_1 = d_1/b_1,$$
$$A_i = -c_i/e_i, \quad B_i = (d_i - a_iB_{i-1})/e_i,$$

где $e_i = a_iA_{i-1} + b_i, \quad i = 2,3,\dots,n-1.$

Значения неизвестных x_i вычисляются в процессе ***обратной прогонки*** с помощью формул

$$x_n = \frac{d_n - a_n B_{n-1}}{b_n + a_n A_{n-1}},$$

$$x_i = A_i x_{i+1} + B_i,\ i = n-1, n-2, \ldots, 2, 1.$$

Составить процедуру решения трехдиагональной системы линейных уравнений произвольного порядка методом прогонки.

4.8.5. (**Метод простой итерации**). Составить программу уточнения решения системы линейных уравнений

$$\vec{x} = C\vec{x} + \vec{f},$$

где $C = \left(c_{ij}\right)$ — некоторая матрица, а $\vec{f} = \left(f_1, \ldots, f_n\right)^t$ — вектор (символ $(\bullet)^t$ означает транспонирование).

Исходя из начального приближения $\vec{x}^{(0)} = \left(x_1^{(0)}, \ldots, x_n^{(0)}\right)$, организуется итерационный процесс

$$\vec{x}^{(k+1)} = C\vec{x}^{(k)} + \vec{f},\quad k = 0, 1, 2, \ldots.$$

Окончание итераций определяется либо заданием максимального числа итераций, либо условием

$$\max_{1 \le i \le n} \left| x_i^{(k+1)} - x_i^{(k)} \right| < \varepsilon,\quad (\varepsilon > 0).$$

4.8.6. ***Метод квадратного корня*** (см., напр., [2], [9]) используется для решения линейной системы

$$A\vec{x} = \vec{b},$$

у которой матрица $A = \left(a_{ij}\right)$ симметрическая, т.е. $a_{ij} = a_{ji}$ для всех i, j. В процессе прямого хода вычисляется треугольная матрица $T = \left(t_{ij}\right)$ такая, что $A = T^t \cdot T$. Для этого используются формулы

$$t_{11} = \sqrt{a_{11}},\quad t_{1j} = \frac{a_{1j}}{t_{11}},\quad j > 1,$$

$$t_{ii} = \left(a_{ij} - \sum_{k=1}^{i-1} t_{ki} t_{kj} \right) / t_{ii},\quad i < j,$$

$$t_{ij} = 0 \ \text{ при } \ i > j.$$

5. Алгоритмы сортировки данных

В этом разделе приведены некоторые алгоритмы упорядочивания данных. Самым простым из них является метод “пузырька”, а наиболее сложным — алгоритм быстрой сортировки. Дополнительные сведения по вопросам сортировки данных можно найти в книгах [5], [6] и [13]. Под

термином *сортировка* в данном случае понимается процедура перестановки элементов множества в определённом порядке, т.е. если даны элементы

$$a_1, \ldots, a_n,$$

то сортировка означает перестановку этих элементов в таком порядке

$$a_{k_1}, a_{k_2}, \ldots, a_{k_n},$$

что при заданной *функции упорядочения* f справедливо соотношение:

$$f\left(a_{k_1}\right) \le f\left(a_{k_2}\right) \le \ldots \le f\left(a_{k_n}\right).$$

5.1. Метод "пузырька"

Будем производить последовательные просмотры массива и каждый раз пару за парой сравнивать соседние числа. Если числа в паре расположены в порядке возрастания, оставляем их без изменения; в противном случае меняем их местами. Затем переходим к следующей паре. Сортировка считается законченной, если в ходе просмотра не была произведена ни одна перестановка (в приведённой далее программе используется переменная `flag`, — в начале каждого просмотра ей присваивается значение True, если в ходе просмотра выполнили хотя бы одну перестановку, значение переменной flag меняется на False, таким образом, по значению этой переменной определяем, нужен или нет ещё один просмотр).

Пример 5.1. Листинг программы.

```
program Project5_1;
{$APPTYPE CONSOLE}
Const N=9; {количество элементов массива}
Type
   TMass = array[1..N] of Double;
Procedure Swap(Var x, y : Double);
{процедура перестановки элементов}
Var
   temp : Double;
begin
 temp := x;  x := y; y := temp;
end;
procedure sort_change(var a:TMass;dim:Integer);
Var
   flag : Boolean;   i, j : Integer;
begin
 j := dim;
 repeat
  flag := True;
  j := j - 1;
```

```
  for i := 1 to j do
   if a[i] > a[i+1] then
    begin
     {перестановка}
     Swap(a[i], a[i+1]);
     flag := False;
    end;
 until  flag;
end;  {sort_change}
{ Основная программа }
Var
   a, b : TMass; i : Integer;
begin
 for i := 1 to N do
  begin
   writeln('a[',i:2,']= ');
   readln(a[i]);
  end;
  {Упорядочим массив};
  Sort_change(a,N);
End.
```

5.2. Метод просеивания

Выполняется так же как метод пузырька, но после перестановки элементов величина с меньшим значением передвигается к началу массива, насколько это возможно. Она сравнивается в обратном порядке со всеми предшествующими элементами массива. Если значение меньше, чем у предшествующего элемента массива, то выполняется обмен. Если же встречается элемент с меньшим значением, то процесс продвижения к началу массива прекращается и нисходящее сравнение возобновляется с той же позиции, с которой начался обратный ход.

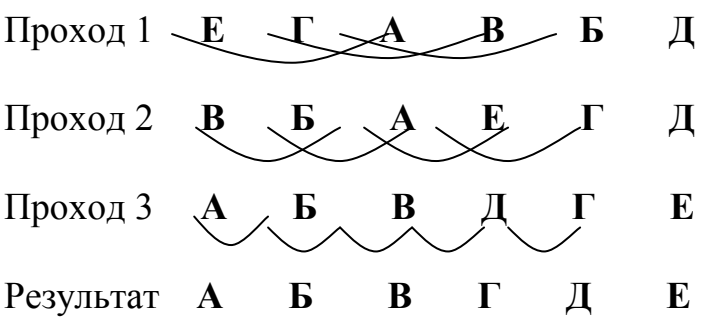
Пример 5.2. Листинг программы.

```
program Project5_2;
{$APPTYPE CONSOLE}
Const N=9; {количество элементов массива}
Type
   TMass=array[1..N] of Double;
Procedure Swap(Var x,y:Double);
{процедура перестановки элементов}
Var
  temp:Double;
begin
 temp:=x; x:=y; y:=temp;
end;
```

```
procedure sort_sift(var a:TMass; dim:Integer);
Var
   i,j,m:Integer;
begin
 m:=dim-1;
 for i:= 1 to m do
  if a[i]>a[i+1] then
   begin
    {перестановка}
    Swap(a[i],a[i+1]);
    {Обратный ход:
     "проталкиваем" a[i]  в начало списка:}
    for j:= i downto 2 do
     if a[j-1] >a[j] then Swap(a[j-1],a[j]);
   end;
end;{sort_sift}
{ Основная программа }
Var a,b:TMass; i:Integer;
begin
 for i:=1 to N do
  begin
   writeln('a[',i:2,']= ');
   readln(a[i]);
  end;
  {Упорядочим массив};
  Sort_sift (a, N);
End.
```

5.3. Метод Шелла

Так же как и метод просеивания, состоит из прямого и обратного хода. Но сравниваются и обмениваются не непосредственные соседи, а элементы, отстоящие на заданном расстоянии. Когда обнаружена перестановка, цепочка вторичных сравнений охватывает те элементы, которые входили в последовательность первичных просмотров.



Каждый последующий просмотр производится с уменьшенным шагом, на последнем просмотре шаг должен равняться 1. Можем использовать следующую процедуру выбора шага. На первом просмотре шаг имеет значение $d = 2^k - 1$, где k выбрано из условия $2^k < n \le 2^{k+1}$. Новый просмотр производим с шагом $d = \left[{(d-1)}/{2} \right]$.

Сортировка заканчивается при $d = 0$.

Пример 5.3. Листинг программы.

```
program Shell;
{$APPTYPE CONSOLE}
uses    SysUtils;
Const n=30;
Type   TMass=Array[1..n] of Integer;
Var   a:TMass;  x:Integer;  i,j, d:Integer;
begin
{Вводим массив:}
for i:= 1 to n do
 begin
   writeln('a[',i:2,']='); readln(a[i]);
 end;
  {Вычисляем d, т.ч. 2^d <n<=2^(d+1)}
    d:=1;
    repeat
      d:=2*d;
    until d>n;
     d:= d-1;
    While d>0 do
     begin
      for i:=d+1 to n do
       begin
         x:=a[i];
         j:=i-d;
         While  (x<a[j])  And (j>0) do
          begin
           a[j+d]:=a[j];
           j:=j-d;
          end;
         a[j+d]:=x;
       end;
       d:= (d-1) div 2;
     end;
 for i:=1 to n do writeln(a[i]);
 readln;
End.
```

5.4. Быстрая сортировка

1. Выбираем в массиве $a_1,\ldots,a_n$ (случайным образом) какой-нибудь элемент x.
2. Просматриваем массив, двигаясь слева направо, пока не найдем элемент $a_i > x$.
3. Просматриваем список, двигаясь справо налево, пока не найдем элемент $a_j < x$.
4. Меняем местами элементы a_i и a_j.
5. Продолжим процесс просмотра, пока два просмотра не встретятся. В результате массив разделится на две части: левую с ключами, меньшими чем x, и правую — с ключами, большими x.
6. Применяем приведённую процедуру для каждой из полученных частей.

Пример. Упорядочим последовательность чисел: 2, 6, 7, 9, 3, 2, 5, 1, 4. Будем обозначать через ***L*** — левую границу просмотра, через ***R*** — правую границу просмотра, в качестве x возьмем элемент a_k с индексом $k = \lceil (L+R)/2 \rceil$.

Первый просмотр.

***L*=1, *R*=9, *x*=a_5**

2, 6, 7, 9, **3**, 2, 5, 1, 4 — $a_2 > x$, $a_8 < x$; поменяем их местами:

2, 1, 7, 9, **3**, 2, 5, 6, 4

2, 1, 7, 9, **3**, 2, 5, 6, 4 — $a_3 > x$, $a_6 < x$; поменяем их местами:

2, 1, 2, 9, **3**, 7, 5, 6, 4

2, 1, 2, 9, **3**, 7, 5, 6, 4 — просмотры встретились

2, 1, 2, **3**, 9, 7, 5, 6, 4.

Список разделился на две части: 2, 1, 2, 3 и 9, 7, 5, 6, 4.

Для каждой из частей применим аналогичную процедуру.

Для левой части: 2, 1, 2, 3. ***L*=1, *R*=4, *x*=a_2.**

2, **1**, 2, 3

1, 2, 2, 3 — левая часть списка упорядочена.

Для правой части: 9, 7, 5, 6, 4. ***L*=1, *R*=5, *x*=a_3.**

9, 7, **5**, 6, 4 — $a_1 > x$, $a_5 < x$; поменяем их местами:

4, 7, **5**, 6, 9

4, 7, **5**, 6, 9 — просмотры встретились

4, **5**, 7, 6, 9.

Список разделился на две части: 4, 5 и 7, 6, 9. Левая часть упорядочена, а для правой требуется один просмотр:

7, **6**, 9

6, 7, 9.

Соединяя упорядоченные части, получаем весь список:
1, 2, 2, 3, 4, 5, 6, 7, 9 .

Пример 5.4. Листинг программы.

```
program Project5_4;
{$APPTYPE CONSOLE}
Const N=9; {количество элементов массива}
Type
    TMass =array[1..N] of Double;
procedure quicksort(var a: TMass;
                    Lo,Hi: integer);
 procedure sort(L,r: integer);
  var
     i, j: integer;   x, y:Double;
 begin
    i:=L; j:=r;
    x:=a[(L+r) DIV 2];
    repeat
      while a[i] < x do i := i + 1;
      while x<a[j] do j:=j-1;
    if i<=j then
       begin
         y := a[i]; a[i] := a[j]; a[j] := y;
         i := i + 1; j := j - 1;
       end;
    until i > j;
    if L < j then sort(L, j);
    if i < r then sort(i, r);
 end;
begin {quicksort}
   sort(Lo,Hi);
end; {quicksort}
{ Основная программа }
Var a, b:TMass; i : Integer;
begin
 for  i := 1 to N do
   begin
    writeln('a[' ,i:2 ,']= ');
    readln(a[i]);
   end;
  {Упорядочим массив};
  quicksort(a,1,N);
End.
```

5.5. Сортировка структурированных данных

В этом разделе показано, как производить сортировку данных типа запись.

Пример 5.5. Пусть имеется массив анкет (записи с полями: “Фамилия”, “Имя”, “Адрес”, “Примечания”). Требуется упорядочить анкеты согласно алфавитному порядку фамилий.

```
program Project5_5;
{$APPTYPE CONSOLE}
Const N=9; {Количество анкет }
Type
    TAnketa = record
              FirstName : String; { Фамилия }
              Name : String; { Имя }
              EMail : String;
              Memo : String; {Примечания}
            end;
    TMass=array[1..N] of TAnketa;
Procedure Swap(Var x, y : TAnketa);
{перестановка анкет}
Var
  temp : TAnketa;
begin
  temp := x; x := y; y := temp;
end;
procedure sort_change(var a : TMass; dim : Integer);
Var
   flag:Boolean;
   i , j : Integer;
begin
   j := dim;
 repeat
  flag := True;
  j := j - 1;
  for  i := 1 to j do
   {сравниваем фамилии:}
   if  a[i].FirstName > a[i+1].FirstName then
    begin
     {перестановка}
     Swap(a[i], a[i+1]);
     flag := False;
    end;
 until  flag;
end;  {sort_change}
Var   Person:TMass; i:Integer;
```

```
begin {Основная программа}
 {Ввод анкет:}
 for i:= 1 to N do
  begin
   Writeln('Anket No ',i:2);
   Writeln('  Name: ');
   Readln(Person[i].Name);
   Writeln('  FirstName: ');
   Readln(Person[i].FirstName);
   Writeln(' Email: ');
   Readln(Person[i].Email);
   Writeln('  Memo: ');
   Readln(Person[i].Memo);
  end;
  {Упорядочим анкеты согласно алфавитному
   порядку фамилий:}
   sort_change(Person,N);
   {Выводим упорядоченные анкеты}
   for  i:= 1 to N do
    begin
     writeln;
     write(Person[i].Name, ' ',
           Person[i].FirstName,' ',
           Person[i].Email,' ',
           Person[i].Memo);
    end;
    readln;
End.
```

5.6. Задачи

3.6.1. Даны два упорядоченных числовых массива ***a*** и ***b***. Написать программу, которая сливает эти массивы в один упорядоченный массив ***c***.
3.6.2. Даны упорядоченные файлы ***f*** и ***g.*** Создать упорядоченный файл ***h*** как слияние файлов ***f*** и ***g***.
3.6.3. Даны упорядоченные файлы ***f*** и ***g.*** Создать упорядоченный файл ***h*** как пересечение файлов ***f*** и ***g***, то есть в файл ***h*** включаются только те элементы файла ***f***, которые содержатся также и в файле ***g***.
3.6.4. Даны упорядоченные файлы ***f*** и ***g.*** Создать упорядоченный файл ***h*** как разность файлов ***f*** и ***g***. Файл ***h*** состоит из компонент файла ***f***, из которых исключены компоненты файла ***g*** (например, если ***f***={1,1,2,2,4,5,6}, а ***g***={1,2,5}, то ***h***={1,2,4,6}). Создание файла ***h*** необходимо выполнить за один просмотр файлов ***f*** и ***g***.
3.6.5. Даны упорядоченные файлы ***f*** и ***g.*** Проверить, содержатся ли все компоненты файла ***f*** в файле ***g***.

3.6.6. Дан файл ***f***. Последовательность $x_i, \ldots, x_k$ компонент файла называется *цепочкой*, если её члены упорядочены $(x_i \le x_{i+1} \le \ldots \le x_{k-1} \le x_k)$. Если, кроме того, $x_{i-1} > x_i$ и $x_k > x_{k+1}$, то цепочка называется максимальной. Найти длину самой длинной максимальной цепочки файла ***f***.

6. Списки

6.1. Приемы работы со списками

Указатель — это переменная, значениями которой являются адреса памяти. С помощью указателей можно разместить в памяти любой тип данных Object Pascal. Указатель называется ***типизированным***, если он связан с некоторым типом данных. Для объявления типизированного указателя используется знак ^, который ставится перед обозначением типа, например,

```
Var  pd:^double;
```

Тип **Pointer** используется для объявления "нетипизированных" указателей, т.е. не связанных с каким-либо конкретным типом.

Память для динамически размещаемой переменной выделяется процедурой **New()**, а освобождается процедурой **Dispose()**.

Присвоение указателю адреса памяти выполняется с помощью оператора @. Доступ к значению динамической переменной осуществляется с помощью символа ^, который помещают после идентификатора указателя, например,

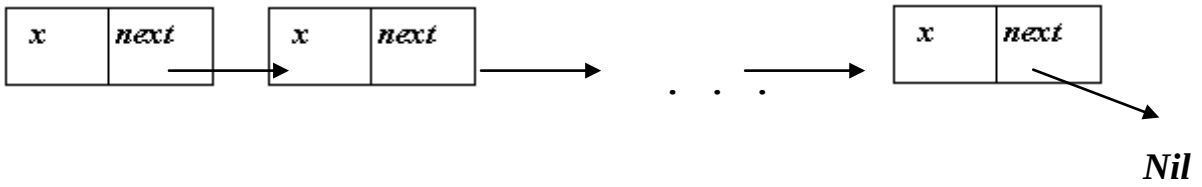
```
Var
   x, y:double; px:^double;
 Begin
   x := 3.141569;
   px := @x;
   y := px^;
 End.
```

С помощью зарезервированного слова **Nil** определяется значение адреса, которое не ссылается ни на какой элемент. Это значение используется при построении списков и деревьев, как признак конца списка или ветви дерева.

Список представляет собой множество, между элементами которого установлено отношение "предыдущий–следующий". Для организации списков стандартными средствами языка Pascal можем применить тип record, содержащий поля, являющиеся ссылками на другие элементы этого же типа. Например, список из вещественных чисел можем создать на основе типа

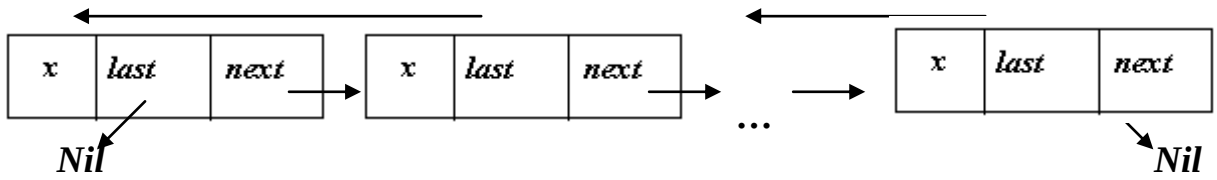
```
Type
    PListDouble=^ListDouble;
    ListDouble=record
                  x:double;
                  next:PListDouble;
               end;
```

Для обозначения линейных списков обычно (см., напр., [5]) используют рисунки



Для работы с линейными списками достаточно знать указатель на первый элемент списка.

Можем также организовать двусвязный список




```
Type
    PListDouble2=^ListDouble2;
    ListDouble2=record
                  x:double;
                  Last:PListDouble2;
                  next:PListDouble2;
               end;
```

Пример 6.1.1. Вводится последовательность действительных чисел (количество чисел до запуска программы неизвестно). Вычислить среднее арифметическое этих чисел.

```
program Project6_1_1;
{$APPTYPE CONSOLE}
uses
     SysUtils;
Type
   PListDouble=^ListDouble;
   ListDouble=record
                 Item:double;
                 next:PListDouble;
              end;
```

```
function CreateList(n:Integer):PListDouble;
Var
    head,p,q:PListDouble;
    x:double; i:integer;
begin
 if n=0 then Result:=Nil {список пуст}
 else
  begin
   New(head);
   p:=head;
   for i:=1 to n do
    begin
     Writeln('x='); Readln(x);
     p^.Item:=x;{занесли данные в список}
     q:=p;
     New(p);
     q^.next:=p; {связали элементы списка}
    end;
   q^.next:=nil; {последняя ссылка в списке}
   dispose(p);
   Result:=head;
end;
end;{CreateList}
Var
    head,p:PListDouble;
    s:double; n:Integer;
begin
 Writeln('n='); Readln(n);
 head:=CreateList(n); {создали список}
 {находим среднее арифметическое:}
 s:=0;
 {проходим по списку:}
 p:=head;
 While p<>Nil do
 begin
    s:=s+p^.Item;
    p:=p^.next; {переходим к следующему элементу}
 end;
 if n>0 then s:=s/n;
 Writeln('s=',s);
 Readln;
End.
```

Пример 6.1.2. Вещественные числа записаны в файл `'info.txt'` по одному в строке. Вычислить $s = x_0 x_n + x_1 x_{n-1} + \cdots + x_n x_0$, где x_i — i–я компонента файла.

```
program Project6_1_2;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Dialogs; {для поддержки ShowMessage}
{Вычислим s=x[0]x[n]+x[1]x[n-1]+...+x[n]x[0],}
{где x[i] - i-я компонента файла F}
Type
   PListDouble2=^ListDouble2;
   ListDouble2=record
                 Item:double;
                 last,next:PListDouble2;
               end;
Var
    head,tail,p,q:PListDouble2;
    x:double; s:double; n:Integer; F:TextFile;
begin
 n:=0;
 AssignFile(F,'info.txt');
 try
   Reset(F);
   if Eof(f) then
    begin {список пуст}
     head:=Nil; tail:=Nil
    end
   else
    begin
     New(head);
     head^.last :=Nil; p:=head;
     While Not Eof(F) do
      begin
       {Заносим в список числа из файла:}
       Read(f,x); Inc(n);
       p^.Item:=x;
       q:=p;
       New(p);
      {свяжем элементы списка:}
       p^.last:=q;
       q^.next:=p;
      end;
      q^.next:=nil; {конец списка}
 {удалим элемент, созданный на последнем шаге:}
    dispose(p);
    tail:=q;
    CloseFile(F);
   end;
```

```
 except on E:EInOutError do
  ShowMessage('Ошибка'+IntToStr(E.ErrorCode));
 end;
 {проводим вычисления:}
  s:=0;
 {проходим по списку:}
 p:=head; q:=tail;
 While p<>Nil do
 begin
    s:=s + (p^.Item) * (q^.Item);
    {переходим к следующей паре элементов:}
    p:=p^.next; q:=q^.last;
 end;
 Writeln('s=',s);
 Readln;
End.
```

6.2. Средства Delphi для поддержки списков

При работе в Delphi списки можно создавать на основе класса **TList**. Свойства и методы этого класса позволяют добавлять, удалять и сортировать элементы списка.

Максимальное число элементов списка (текущая емкость списка) задается с помощью свойства

property Capacity: Integer;

Емкость списка, если она исчерпана, будет автоматически увеличена на фиксированную величину. С помощью этого свойства можно как увеличить, так и уменьшить объем выделенной для списка памяти. Если значение `Capacity` окажется меньше значения `Count`, возникает исключительная ситуация `EListError`.

Число элементов списка регулируется свойством

property Count: Integer;

это свойство изменяется при добавлении или удалении элементов списка.

Доступ к указателям на элементы списка осуществляется с помощью свойства

property Items(Index:Integer):Pointer;

С помощью параметра `Index` задается порядковый номер элемента, при этом первый элемент списка имеет номер 0.

Приведем наиболее важные методы класса `TList`.

Для размещения нового элемента `Item` в конец списка предназначен метод

function Add(Item:Pointer):Integer;

Возвращает индекс размещенного элемента (первый элемент списка имеет индекс 0).

Можно добавить новый элемент Item не только в конец списка, но и в заданную позицию Index. Для этой цели используется метод

procedure Insert(Index:Integer;Item:Pointer);

Индексы всех элементов списка, расположенных после элемента Item, увеличиваются на единицу. Если Index равен 0, то элемент вставляется в начало списка.

Удалить все элементы списка можно с помощью метода

procedure Clear;

Свойствам Capacity и Count присваиваются нулевые значения, но память, выделенная под элементы списка, не освобождается.

Для удаления отдельных элементов списка используется метод

procedure Delete(Index:Integer);

Параметр Index содержит индекс удаляемого элемента. Индекс элементов списка, расположенных за удаляемым элементом, уменьшается на единицу.

Поменять местами элементы списка с индексами Index1 и Index2 можно с помощью метода

procedure Exchange(Item1, Index2:Integer);

Индекс элемента списка с указателем Item можно узнать с помощью метода

function IndexOf(Item:Pointer):Integer;

Пример 6.2.1. Выполним ту же задачу, что и в примере 6.1.1, но с помощью класса **TList**.

```
program Project6_2_1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Classes;{добавили эту ссылку для поддержки типа
           TList}
Var
   List:TList;
   x,s:double; px:^double; i,n:Integer;
begin
 Writeln('n='); Readln(n);
 {Создадим список из n элементов:}
 List:=TList.Create; {вызвали конструктор}
 for i:= 1 to n do
   begin
     New(px);
     Writeln('x='); Readln(x);
     px^:=x;
     List.Add(px);
   end;
```

```
 {найдем среднее арифметическое элементов}
 s:=0;
 {проходим по списку:}
 for i:=0 to List.Count-1 do
 begin
    px:=List[i];
    s:=s+px^;
 end;
 if List.Count>0 then s:=s/List.Count;
 Writeln('s=',s);
 Readln;
End.
```

Пример 6.2.2. Вариант задачи 6.1.2 на основе класса **`TList`**.

```
program Project6_2_2;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Dialogs,{для поддержки ShowMessage}
  Classes;{для поддержки типа TList}
Var
   List : TList;
   px,qx:^double;
   F:TextFile; x,s:double; i,j,n:Integer;
begin
 {Создадим список:}
 List:=TList.Create;
 AssignFile(F,'info.txt');
 try
   Reset(F);
   While Not Eof(F) do
    begin
     {Заносим в список числа из файла:}
     New(px);
     Read(F,x);
     px^:=x;
     List.Add(px);
    end;
   CloseFile(F);
 except
  on E:EInOutError do
      ShowMessage('Ошибка'+IntToStr(E.ErrorCode));
 end;
 {проводим вычисления:}
  s:=0;
  n:=List.Count-1;
```

```
  for i:=0 to n do
   begin
     px:=List[i];
     j:=n-i;
     qx:=List[j];
     s := s + (px^) * (qx^);
   end;
 Writeln('s=',s);
 Readln;
End.
```

С помощью метода

procedure Sort(Compare:TListSortCompare);

можно выполнить сортировку списка. Функция **Compare()** задает функцию упорядочения, т.е. определяет операцию сравнения элементов списка.

Пример 6.2.3. Упорядочим список действительных чисел, считанных из файла 'info.txt'.

```
program Project6_2_3;
{$APPTYPE CONSOLE}
uses
    SysUtils,   Classes,  Dialogs;
Type
    PDouble=^Double;
function Compare(Item1,Item2:Pointer):Integer;
begin
 if PDouble(Item1)^ <PDouble(Item2)^ then
    Result := -1
 else
  if  PDouble(Item1)^ > PDouble(Item2)^ then
     Result := 1
  else Result:=0;
end;
Var  List:TList;
     F:TextFile;
     x:double; px:^double; i:integer;
begin
    List:=TList.Create;
    AssignFile(F,'info.txt');
    try
      Reset(F);
      While Not Eof(F) do
       begin
         New(px);
         Read(F,x);
         px^:=x;
```

```
          List.Add(px);
        end;
        CloseFile(F);
     except
      on E:EInOutError do
ShowMessage('Ошибка'+IntToStr(E.ErrorCode));
     end;
     {список до упорядочения:}
     for i:=0 to List.Count -1 do
      begin
        px:=List[i]; writeln(px^);
      end;
     {список после упорядочения:}
     Writeln('Sort:');
     List.Sort(Compare);
     for i:=0 to List.Count -1 do
      begin
        px:=List[i]; writeln(px^);
      end;
   Readln;
End.
```

7. Вычисления с многократной точностью

Подробное изложение вопросов, связанных с арифметикой многократной точности, можно найти в [12]. Книги [16], [21] содержат разделы, посвященные вычислению числа π с высокой точностью.

Основными в арифметике многократной точности являются операции:

a) сложение и вычитание n-разрядных целых чисел, с получением n-разрядного ответа и цифры переноса;
b) умножение n-разрядного целого числа на m-разрядное целое число, с получением $(m+n)$-разрядного результата;
c) деление $(m+n)$-разрядного целого числа на n-разрядное целое число, с получением $(m+1)$-разрядного частного и n-разрядного остатка.

Д. Кнут предложил называть эти алгоритмы “классическими”, так как, по его замечанию (см. [12, стр. 282]), само слово “алгоритм” в течение многих веков использовалось лишь в связи с этими вычислительными процессами.

7.1. Вычисление числа 2^n для большого n

Для хранения больших чисел можем создать массив, каждая цифра которого будет хранить одну цифру числа. Приведем программу вычисления числа 2^{1000}. Можно вычислить и большую степень, но нужно

изменить значение константы MaxLength, в которой указано количество цифр вычисляемого числа.

7.1.1. Алгоритм вычисления

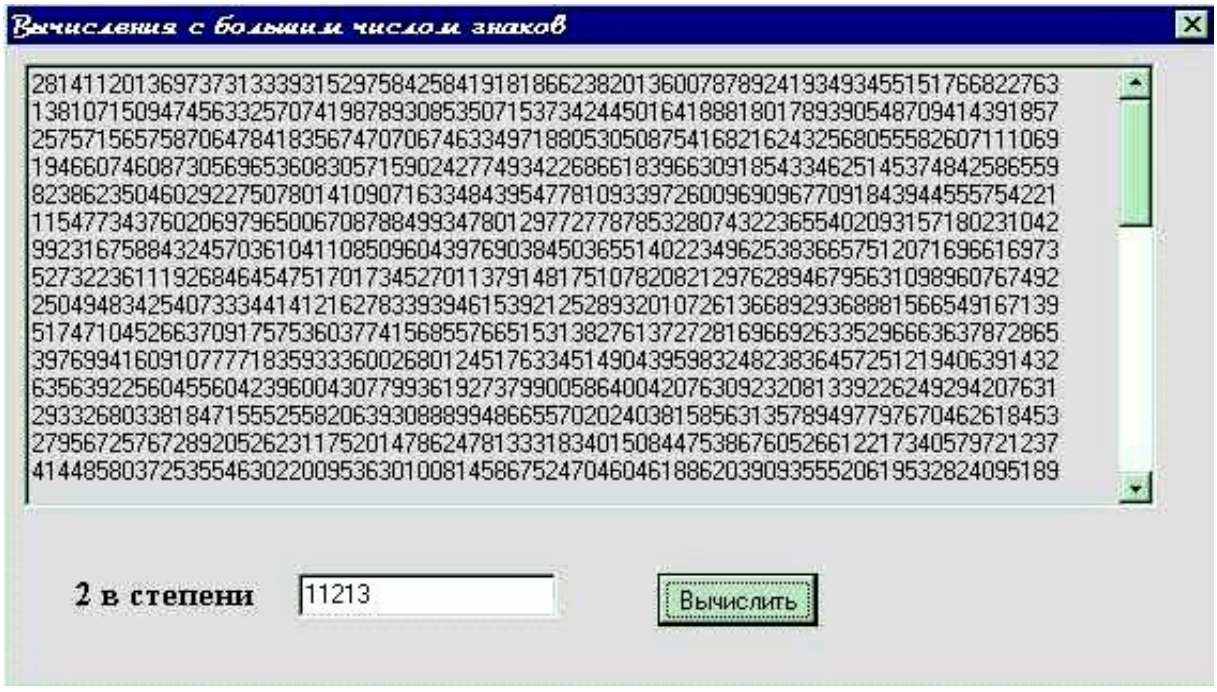
```
program Power7_1_1;
{$APPTYPE CONSOLE}
Const
     MaxLength=302;{Число цифр, достаточное  для
                    хранения числа 2**1000}
Var
  xdigit : array[1..MaxLength] of  0..9;
  xstart : 1..MaxLength;
  N : Integer;{Показатель степени}
  i : 1..MaxLength;
procedure Power2(N:integer);
Var
   i : 1..MaxLength;
   transfer:Integer;{переносимая в след.разряд цифра}
   b,k:Integer;
begin
 xdigit[MaxLength] := 1;
 xstart := MaxLength;
 for k := 1 to N do
  begin{1}
    transfer := 0;
    for i := MaxLength downto xstart do
     begin{2}
      {удвоить цифру xdigit[i]}
      b := 2*xdigit[i];
      {прибавить перенос из предыдущего разряда}
      b := b + transfer;
      {записать цифру единиц суммы в xdigit[i]}
      xdigit[i] := b mod 10;
      {запомнить перенос}
      transfer := b div 10;
     end;{2}
     if transfer <> 0 then
      begin
       xstart := xstart - 1;
       xdigit[xstart] := transfer;
      end;
  end;{1}
end;{Power2}
begin{Основная программа}
 Writeln('Type n (<=1000):');  Readln(N);
```

```
 Power2(N);
 Writeln('2**',N:4,'=');
 for i := xstart to MaxLength do
write(xdigit[i]:1);
  readln;
End.
```

В известной книге М. Гарднера “Математические новеллы” приведена распечатка числа $2^{11213}-1$ (найденное с помощью ЭВМ двадцать третье число Мерсенна). Разумеется, самой трудной задачей при программировании была проверка чисел на простоту (см. по этому поводу [12]). Нас, в данном случае, интересует организация хранения и вывода чисел с большим числом значащих цифр. С помощью только что приведенной программы можем получить распечатку указанного числа, нужно лишь увеличить значение **MaxLength** до 3376 (это количество цифр в числе, но можно было бы взять “с запасом”, например 4000).

7.1.2. Организация интерфейса

Для вывода результата будем использовать компоненту **Memo1** (страница **Standard** палитры компонент). Обычно с помощью этой компоненты создают многострочный редактор текста. Степень числа 2 вводится с помощью компоненты **Edit1** (страница **Standard**). Вычисления производятся после нажатия на кнопку **BitBtn1** (страница **Additional**).


Вычисления с большим числом знаков
2814112013697373133393152975842584191818662382013600787892419349345515176682276
3138107150947456332570741987893085350715373424450164188818017893905487094143918
2575715657587064784183567470706746334971880530508754168216243256805558260711106
1946607460873056965360830571590242774934226866183966309185433462514537484258655
9
8238623504602922750780141090716334843954778109339726009690967709184394455575422
1
1154773437602069796500670878849934780129772778785328074322365540209315718023104
2
9923167588432457036104110850960439769038450365514022349625383665751207169661697
3
5273223611192684645475170173452701137914817510782082129762894679563109896076749
2
2504948342540733344141216278333934615392125289320107261366892936888156654916713
9
5174710452663709175753603774156855766515313827613727281696692633529666363787286
5
3976994160910777718359333600268012451763345149043959832482383645725121940639143
2
6356392256045560423960043077993619273799005864004207630923208133922624929420763
1
2933268033818471555255820639308889948665570202403815856313578949779767046261845
3
2795672576728920526231175201478624781333183401508447538676052661221734057972123
7
4144858037253554630220095363010081458675247046046188620390935552061953282409518
9
```

С помощью **Object Inspector** установим значения свойств размещённых компонент.
```

Очистим поле **Text** у компоненты **Edit1** (из этого поля прочитаем значение степени числа 2, введённое пользователем).

У компоненты **Memo1** выберем значение **True** для свойства **ReadOnly**, запретив тем самым исправления результата вычислений.

Найдем свойство **ScrollBars** компоненты **Memo1** и выберем либо значение **ssBoth** либо **ssVertical** (в случае, если результат не уместится в окне, можно "прокрутить" окно **Memo**1 с помощью полос прокрутки).

Изменим значение свойства **Caption** компонента **BitBtn1** на **"Вычислить"**. Все вычисления сосредоточены в обработчике для кнопки **"Вычислить"**. Поместим в этот обработчик рассмотренную ранее процедуру **Power2()**. Прежде чем выводить результат, очистим поле **Memo1** с помощью оператора **Memo1.Lines.Clear;** Поскольку результат может оказаться длиннее, чем ширина окна **Memo1**, нужно разделить результат на части и выводить в несколько строк. Каждая новая строка добавляется с помощью метода **Insert()**:

Memo1.Lines.Insert(ys, S);

здесь **S** — строка с информацией, а **ys** — номер добавляемой строки (начинается с **0**). Ширина строки устанавливается значением переменной **xsMax**.

Далее приведен полный текст обработчика.

```
procedure TForm1.BitBtn1Click(Sender: TObject);
Const
     MaxLength=4000;
Var
    xdigit:array[1..MaxLength] of Integer;
    xstart:Integer;   N:Integer;
    i:Integer;  xS,xSMax, yS:integer;
    S:String;
procedure Power2(N:integer);
Var
   i, k, transfer, b : Integer;
begin
 xdigit[MaxLength] := 1;
 xstart := MaxLength;
 for k := 1 to N do
  begin{1}
    transfer := 0;
    for i := MaxLength downto xstart do
     begin{2}
      b := 2 * xdigit[i];
      b := b + transfer;
```

```
      xdigit[i] := b mod 10;
      transfer := b div 10;
     end;{2}
     if transfer <> 0 then
      begin
       xstart := xstart - 1;
       xdigit[xstart] := transfer;
      end;
  end;{1}
end;{Power2}
begin
  N := StrToInt(Form1.Edit1.Text);
  Power2(N);
  xS := 1; yS := 0;
  S:=''; {пустая строка}
 xSMax := 80; {количество  символов в строке
               Memo1.Lines[ys]}
 Memo1.Lines.Clear;
 for i := xstart to MaxLength do
  begin
   S := S + IntToStr(xdigit[i]);
    if xS < xSMax then  Inc(xS)
      else
      begin
       Memo1.Lines.Insert(yS,S);
       S := '';{пустая строка}
       xS := 1; Inc(yS);
      end;
   end;
  Memo1.Lines.Insert(yS,S);
 End;
```

7.2. Вычисление числа π

Краткий исторический обзор по анализу и вычислению числа π дан в статье [26], там же приведены наиболее известные расчётные формулы. Отметим книги [16] и [21], содержащие популярное изложение алгоритма вычисления этой математической константы. В Internet самую обширную информацию о числе π, по нашему мнению, можно найти на сайте www.cesm.sfu.ca/pi/.

В 1949 году на одном из первых компьютеров было вычислено 2035 знаков числа π, на что потребовалось около 70 часов машинного времени (см. [27]). Вычисления числа π продолжаются и в настоящее время (см., например, www.cc.u-tokyo.ac.jp и http://wasi.org/PI/).

Самым простым и хорошо известным является разложение

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\dots .$$

Но этот ряд плохо подходит для вычисления π, поскольку сходится медленно. При вычислениях чаще используются формулы (см., напр., [21, 26])

$$\frac{\pi}{4}=4\arctan\frac{1}{5}-\arctan\frac{1}{239},$$

$$\frac{\pi}{4}=8\arctan\frac{1}{10}-4\arctan\frac{1}{515}-\arctan\frac{1}{239},$$

$$\frac{\pi}{4}=3\arctan\frac{1}{4}+\arctan\frac{1}{20}+\arctan\frac{1}{1985}.$$

Для вычисления arctan *x* применяется ряд Грегори

$$\arctan x=\sum_{k=0}^{\infty}\frac{(-1)^k x^{2k+1}}{2k+1},\quad -1\le x\le 1.$$

В работе [28] вычисление числа π производилось по формуле

$$\pi=24\arctan\frac{1}{8}+8\arctan\frac{1}{57}+4\arctan\frac{1}{239},$$

расчеты проверялись с помощью тождества Гаусса

$$\pi=48\arctan\frac{1}{18}+32\arctan\frac{1}{57}-20\arctan\frac{1}{239}.$$

7.2.1. Алгоритм вычисления

```
program PiCalcul;
{Вычисление числа $\pi$ с помощью формулы Мечина
 $\pi /4 = 4 \arctan 1/5 - \arctan 1/239$}
{$APPTYPE CONSOLE}
uses
   SysUtils;
Const
   Basa=10000;{наименьшее число типа SmallInt с
               максимальным количеством разрядов}
Type
   TStrInt=String[4];
Type
   TNaborDigs=Array of SmallInt; {Предназначен для
                    хранения набора значащих цифр}
function SmallIntToStr(x:SmallInt):TStrInt;
Var i,n:Byte; s:TStrInt;
begin
 for i:= 1 to 4 do
  begin
    n := x mod 10;
```

```
    s[5-i] := chr(n+ord('0'));
    x := x div 10;
  end;
 Result := s;
end;

{Процедуры арифметики многократной точности:}
{Сложение:}
procedure AddNabor(Var res:TNaborDigs;
                   increm:TNaborDigs);
Var i,nblock:Integer;
begin
  nblock :=Length(res);
  for i:=nblock-1 downto 0 do
    begin
      res[i] := res[i] + increm[i];
      if res[i] >= Basa then
        begin
          res[i] := res[i] - Basa;
          res[i-1] := res[i-1] + 1;
        end
    end;
end; {AddNabor}
{Вычитание:}
procedure SubNabor(Var res:TNaborDigs;
                   decrem:TNaborDigs);
Var  i,nblock:Integer;
begin
  nblock := Length(res);
  for i := nblock-1 downto 0 do
    begin
      res[i] := res[i] - decrem[i];
      if res[i] < 0 then
        begin
          res[i] := res[i] + Basa;
          res[i-1] := res[i-1] - 1;
        end
    end;
end; {SubNabor}
{Умножение:}
procedure MultNabor(Var res:TNaborDigs;
                    factor:SmallInt);
Var i,nblock:Integer; transfer:SmallInt;
    t:Integer;
begin
  nblock :=Length(res);
```

```
  transfer := 0;
  for i := nblock-1 downto 0 do
    begin
      t:=res[i];
      t := t * factor;
      t := t + transfer;
      transfer := t div Basa;
      res[i] := t mod Basa;
    end;
end; {MultNabor}
{Деление:}
procedure DivideNabor(Var res:TNaborDigs;
                      denom:SmallInt);
Var i,nblock:Integer;
    transfer:SmallInt; t:Integer;
begin
  nblock :=Length(res);
  transfer := 0;
  for i:= 0 to nblock-1 do
    begin
      t:=res[i];
      t := t + transfer * Basa;
      transfer := t mod denom;
      res[i] := t div denom;
    end;
end; {DivideNabor}
{Установка значений: целое число записываем в
 массив TNaborDigs}
procedure IntToNabor(num:SmallInt;
                     Var res:TNaborDigs);
Var i,nblock:Integer;
begin
  nblock :=Length(res);
  for i := 0 to nblock-1 do res[i] := 0;
  res[0] := num;
end; { IntToNabor}
{Копирование многозначного числа в другую переменную:}
procedure CopyNabor(Var res:TNaborDigs;
                    from:TNaborDigs);
Var i,nblock:Integer;
begin
  nblock :=Length(res);
  for i := 0 to nblock-1 do   res[i] := from[i];
end; {CopyNabor}
```

```
{Сравнение с нулем:}
function ZeroNabor(Var res:TNaborDigs):Boolean;
Var i,nblock:Integer;
begin
 nblock :=Length(res);
 for i := 0 to nblock-1 do
    if res[i]<>0  then
     begin
       Result := False;
       Exit
     end;
 Result := True;
end; {ZeroNabor}
{Вычисление arctan от 1/x с большим числом знаков:}
procedure ArctanNabor(Var res:TNaborDigs;
                      denom:SmallInt);
Var
   w1, w2 : TNaborDigs;
   k, denom2 : SmallInt;
   nblock:Integer;
begin
   nblock :=Length(res);
   SetLength(w1,nblock * SizeOf(SmallInt));
   SetLength(w2,nblock * SizeOf(SmallInt));
   IntToNabor(1,res);
   DivideNabor(res,denom);
   CopyNabor(w1,res);
   k := 1;
   Repeat
     DivideNabor(w1,denom); DivideNabor(w1,denom);
     CopyNabor(w2,w1);
     DivideNabor(w2, 2 * k +1);
     if Odd(k) // т.е. k - нечетное
       then SubNabor(res,w2)
       else AddNabor(res,w2);
     k := k + 1;
   Until ZeroNabor(w2);
end; {ArctanNabor}
procedure PiCalc(Var Res:TNaborDigs);
Var
   ResTemp : TNaborDigs;nblock:Integer;
begin
  nblock :=Length(res);
  SetLength(ResTemp,nblock * SizeOf(SmallInt));
  ArctanNabor(Res,5);
  MultNabor(Res,4);
```

```
  ArctanNabor(ResTemp,239);
  SubNabor(Res,ResTemp);
  MultNabor(Res,4);
end; {PiCalc}
procedure PrintNabor(Res:TNaborDigs);
Var  i : Integer; ff:TextFile;
ss:TStrInt;
begin
 AssignFile(ff,'Pi_Smallf.txt');
 Rewrite(ff);
 Writeln(ff,'$\pi=$');
 for i := 0 to High(Res) do
  begin
   ss:= SmallIntToStr(res[i]);
   writeln(ff,ss);
  end;
  CloseFile(ff);
end;{PrintNabor}

Var
   ndigit : Integer; Res : TNaborDigs;
Var
   nblock:Integer;{количество блоков цифр, т.е.
        количество элементов массива типа TNaborDigs}
begin
   Writeln('\pi with N digits, N= ');
   Readln(ndigit);
   if ndigit < 20 then ndigit:=20;
   nblock := ndigit div 4;
   SetLength(Res,nblock * SizeOf(SmallInt));
   PiCalc(Res);
   PrintNabor(Res);
   Readln;
End.
```

Для хранения цифр числа π используется массив `Res` типа `TNaborDigs`. Каждый элемент этого массива содержит 4 значащих цифры числа π. Если тип `TNaborDigs` определить на основе типа `Integer`, то наборы цифр будут содержать по 9 значащих цифр. Значение константы `Basa` в этом случае нужно изменить на 1000000000. Отметим, что также потребуется небольшая корректировка процедуры `MultNabor`(), так как при выполнении операций умножения можем выйти за допустимый диапазон значений используемого типа.

Для вывода результатов многоточных вычислений используется процедура `PrintNabor()`. Эта процедура учитывает “незначащие” нули

чисел, входящих в наборы. Так, число 781 (это 17 набор наших вычислений) будет преобразован в 0781.

7.3. Задачи

7.3.1. Составить программы сложения, вычитания, умножения и деления целых чисел с большим числом значащих цифр.

7.3.2. Вычислить 1000 чисел Фибоначчи (см. задачи 1.3.7, 1.3.8 и пример 2.5.1).

7.3.3. Составить программу вычисления $\sqrt{2}$ с большой точностью, используя последовательность

$$x_1 = 1, \quad x_{k+1} = \frac{x_k}{2} + \frac{1}{x_k}, \quad k = 1, 2, \ldots .$$

Эти соотношения представляют собой метод Ньютона (см. 4.3), примененный к уравнению x^2-2=0.

7.3.4. Вычислить число ***e*** с заданным количеством десятичных знаков, используя формулу

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots .$$

7.3.5. Вычислить число π с заданным количеством десятичных знаков, используя для хранения результата массив типа

```
Type  TNaborDigs=Array of Integer;.
```

8. Операции с датой и временем

8.1. Тип данных TDateTime

В Delphi имеется специальный тип данных **`TDateTime`** для выполнения операций с датой и временем. Этот тип определяет число с плавающей точкой, целая часть которого содержит число дней, отсчитанное от 12 часов 30 декабря 1899 года (“заданное начало”), а дробная часть содержит время, равное части 24-часового дня.

Над данными типа `TDateTime` определены те же операции, что и над вещественными числами.

Для перевода из “привычного” формата “число, месяц, год” к значению типа `TDateTime` используется функция

```
function StrToDate(const S:String):TDateTime;
```

Формат представления даты в строке `S` зависит от установок операционной системы. В MS Windows эти установки можно изменить с помощью вызова утилиты “**Язык и стандарты**” панели управления. Для России, по умолчанию, установлен формат “**`дд.ММ.гггг`**”. Можно использовать формат “**`дд.ММ.гг`**”, если год попадает в интервал (например, от 1930 до 2029), установленный с помощью той же утилиты.

Для преобразования времени из формата “час:мин:сек” в формат TDateTime используется функция

```
function StrToTime(const S:String):TDateTime;
```

Преобразовать дату, записанную как “дд.ММ.гггг час:мин:сек”, в формат TDateTime можно с помощью функции

```
function StrToDateTime(const S:String):TDateTime;
```

Параметр **S** в этих функциях должен отвечать установленному формату для даты и времени, в противном случае возбуждается исключение **EConvertError**.

Для “обратного” преобразования из формата TDateTime в строку предназначены функции

```
function DateToStr(date:TDateTime):String;
function TimeToStr(time:TDateTime):String;
function DateTimeToStr(date_time:TDateTime):String;
```

С помощью функций **DateTimeToString()** и **FormatDateTime()** можно управлять форматом строки, в которую производится преобразование из типа TDateTime.

Кроме того, привести заданную дату к типу TDateTime можно с помощью функции

```
function EncodeDate(Year,Month,Day:Word):TDateTime;
```

Для приведения времени к типу TDateTime можно использовать функцию

```
function EncodeTime(Hour,Min,Sec,MSec:Word):TDateTime;
```

“Обратные” операции по разделению даты и времени на составляющие выполняются с помощью процедур

```
procedure DecodeDate(date:TDateTime;Var
          Year,Month,Day:Word);
procedure DecodeTime(time:TDateTime;Var
          Hour,Min,Sec,MSec:Word);
```

Текущее значение времени определяется с помощью функции

```
function Time:TDateTime;
```

Текущую дату в формате TDateTime возвращает функция

```
function Date:TDateString;
```

Текущее значение даты и времени можно определить с помощью вызова функции

```
function  Now:TDateTime;
```

С помощью функции

```
function DayofWeek(date:TDateTime):Integer;
```

можно узнать день недели, соответствующий дате date, при этом значение 1 отведено воскресенью, а значение 7 — субботе.

Функция

```
Function IsLeapYear(Year:Word):Boolean;
```

возвращает значение **True**, если год **Year** високосный, и **False** — в ином случае.

Пример 8.1.1.

```
program Project8_1_1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
Var
   d:TDateTime; i:Integer;
   Year,Month,Day,Hour,Min,Sec,MSec:Word;
begin
  d:=StrToDate('2.4.2003');  Writeln(d);
  d:=StrToDate('2.4.1812');  Writeln(d);
  d:=StrToTime('21:53:13'); Writeln(d);
  d:=StrToDateTime('2.4.2003 21:53:13');
  Writeln(d);
  d:=EncodeTime(21,53,13,00); Writeln(d);
  d:=EncodeDate(2003,4,2);   Writeln(d);
  i:=DayofWeek(d);
  Writeln('Day of Week=',i);
  d:=Date;
  Writeln('Today is ',DateToStr(d));
  d:=Time; Writeln('Time=',TimeToStr(d));
  d:=Now;
  Writeln('Today is ',DateTimeToStr(d));
  DecodeDate(d,Year,Month,Day);
  Writeln(Day,'.',Month,'.',Year);
  DecodeTime(d,Hour,Min,Sec,MSec);
  Writeln(Hour,':',Min,':',Sec);
  Readln;
End.
```

8.2. Задачи

8.2.1. Вычислить количество полных лет между двумя заданными датами.

8.2.2. Написать функцию, возвращающую количество дней, оставшихся до конца года.

8.2.3. Написать функцию, возвращающую количество дней, прошедших с начала года.

8.2.4. Вводится дата. Составить программу, выводящую предыдущую (вчерашнюю) и следующую (завтрашнюю) даты.

8.2.5. Найти день недели для заданной даты, используя формулу (см. [17, стр. 110])

$$W = d + \left[\frac{1}{5}(13m - 1)\right] + Y + \left[\frac{1}{4}Y\right] + \left[\frac{1}{4}C\right] - (2C) \bmod 7,$$

где C — число полных прошедших столетий (например, для 2003 года значение C равно 20), Y — номер года в столетии (например, для 2003 года значение Y равно 3), m — номер месяца, а d — день. Значение W, равное 0, соответствует воскресенью, 1 — понедельнику и так далее.

8.2.6. Вычислить количество пятниц, выпадающих на 13-е число, в интервале между двумя заданными датами.

8.2.7. Составить программу для пересчета дат старого стиля на новый стиль и обратно. Для пересчета даты J юлианского календаря (старый стиль) в дату G григорианского календаря (новый стиль) необходимо к J прибавить число дней N, определяемых по формуле (см., напр., [19, стр. 21], [25, стр. 75])

$$N = C - \left[\frac{C}{4}\right] - 2,$$

где C — число полных прошедших столетий. При пересчете дат следует учитывать, что количество високосных лет в юлианском и григорианском календарях не совпадает: три года в 400 лет в юлианском календаре високосные, а в григорианском простые. В григорианском календаре из списка високосных исключаются те годы, у которых первые две цифры не делятся на 4, а последние две являются нулями (в юлианском такие годы високосные). После реформы календаря в 1582 году такими годами были 1700, 1800 и 1900, а следующим будет 2100 год.

8.2.8. Составить программу для вычисления даты Пасхи для заданного года с помощью формул Гаусса (см., напр. [14, стр. 214], [19, стр. 23). При расчете христианской католической Пасхи нужно найти величины a, b, c, как остатки от деления номера года J на 19, 4 и 7. Затем вычислить значение d, как остаток от деления величины $19a + x$ на 30, где x равно 22, если $1582 \le J \le 1699$, x равно 23, если $1700 \le J \le 1899$ и x равно 24, если $1900 \le J \le 2099$. Далее следует выбрать y согласно правилу: $y = 2$, если $1582 \le J \le 1699$; $y = 3$, если $1700 \le J \le 1799$; $y = 4$, если $1800 \le J \le 1899$ и $y = 5$, если $1900 \le J \le 2099$. После этого найти значение e как остаток от деления величины $2b + 4c + 6d + y$ на 7. Если $d + e \le 10$, то Пасха будет $22 + d + e$ марта, иначе $d + e - 9$ апреля. При вычислении православной Пасхи всегда $x = 15$, $y = 6$ и результат получается в датах по старому стилю.

Литература

1. Абрамов С.А., Гнездилова Г.Г., Капустина Е.Н., Селюн М.И. *Задачи по программированию*. — М.: Наука. Гл. ред. физ.-мат. лит., 1988. — 224 с.
2. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. *Численные методы* — М.: Наука. Гл. ред. физ.-мат. лит., 1987. — 600 с.
3. Брудно А.Л., Каплан Л.И. *Московские олимпиады по программированию*. — М.: Наука. Гл. ред. физ.-мат. лит., 1990. — 208 с.
4. Вирт Н. *Систематическое программирование. Введение*: Пер. с англ. — М.: Мир, 1977. — 184 с.
5. Вирт Н. *Алгоритмы + структуры данных = программы*: Пер. с англ. — М.: Мир, 1985. — 406 с.
6. Вирт Н. *Алгоритмы и структуры данных*: Пер. с англ. — М.: Мир, 1989. — 360 с.
7. Вьюкова Н.И., Галатенко В.А., Ходулев А.Б. *Систематический подход к программированию*: — М.: Наука. Гл. ред. физ.–мат. лит., 1988. — 208 с.
8. Гудман С., Хидетниеми С. *Введение в разработку и анализ алгоритмов*: Пер. с англ. — М.: Мир, 1981. — 368 с.
9. Демидович Б.П., Марон И.А. *Основы вычислительной математики.* — М.: Физматгиз, 1963. — 660 с.
10. Карпов Б. *Delphi: специальный справочник.* — СПб.: Питер, 2002. — 688 с.
11. Кнут Д. *Искусство программирования для ЭВМ. Т.1. Основные алгоритмы*: Пер. с англ. — М.: Мир, 1976. — 735 с.; перераб. издание: М.: Изд. Дом “Вильямс”, 2000. —820 с.
12. Кнут Д. *Искусство программирования для ЭВМ. Т.2. Получисленные алгоритмы*: Пер. с англ. — М.: Мир, 1978. — 725 с.; перераб. издание: М.: Изд. Дом “Вильямс”, 2000. —712 с.
13. Кнут Д. *Искусство программирования для ЭВМ. Т.3. Сортировка и поиск*: Пер. с англ. — М.: Мир, 1978. — 846 с.; перераб. издание: М.: Изд. Дом “Вильямс”, 2000. —822 с.
14. Куликов С. *Нить времен: Малая энциклопедия календаря.* — М.: Наука. Гл. ред. физ.-мат. лит., 1991. — 288 с.
15. Липский В. *Комбинаторика для программистов*: Пер. с англ. — М.: Мир, 1988. — 213 с.
16. Нивергельт Ю., Фаррар Дж., Рейнгольд Э. *Машинный подход к решению математических задач*: Пер. с англ. — М.: Мир, 1977. — 352 с.

17. Оре О. *Приглашение в теорию чисел*: Пер. с англ. — М.: Наука. Гл. ред. физ.-мат. лит., 1980. — 128 с.
18. Плещинский Н.Б. *Объектное программирование в Delphi.* Учебное пособие. — Казань: Изд-во КМО, 1999. — 86 с.
19. *Работа с Turbo Professional. Программирование операций с датой и временем.* (Сост. Липачёв Е.К., Насибулин В.Г.) — Казань, 1995. — 38 с.
20. Рейнгольд Э., Нивергельт Ю., Део Н. *Комбинаторные алгоритмы. Теория и практика.* — М.: Мир, 1980. — 476 с.
21. Уэзерелл Ч. *Этюды для программистов*: Пер. с англ. — М.: Мир, 1982. — 288 с.
22. Фаронов В.В. *Delphi 3. Учебный курс.* —- М.: Нолидж, 1998. — 400 с.
23. Фаронов В.В. *Delphi 4. Учебный курс.* — М.: Нолидж, 1998. — 464 с.
24. Форсайт Дж., Малькольм М., Моулер К. *Машинные методы математических вычислений*: Пер. с англ. — М.: Мир, 1980. — 280 с.
25. Хренов Л.С., Голуб И.Я. *Время и календарь*. — М.: Наука. Гл. ред. физ.-мат. лит., 1989. — 128 с.
26. Bailey D.H., Borwein J.M., Borwein P.B. and Plouffe S. *The Ques for Pi* // The Mathematical Intelligencer, June, 1996 (www.cecm.sfu.ca/personal/pborwein/).
27. Reitwiesner G.W. *An ENIAC Determination of π and e to more than 2000 Decimal Places* // Mathematical Tables and Other Aids to Computation, Vol. 4, pp. 11 – 15, 1950 (www.jstor.org).
28. Shanks D., Wrench J.W. *Calculation of π to 100,000 Decimals* // Mathematics of Computation, Vol 16, pp. 76 – 99, 1962 (www.jstor.org).

Е.К. Липачёв

ВВЕДЕНИЕ В КОМПЬЮТЕРНЫЕ НАУКИ.
ОСНОВНЫЕ АЛГОРИТМЫ

Редактор И.Г. Кондратьева

Подписано в печать 16.10.2003. Форм. 60 x 84 1/16. Гарнитура «Таймс».
Печать офсетная. Усл. печ. л. 5,25. Уч.–изд. 5,5. Тираж 100. Заказ 294.

Издательство
«Казанский государственный университет им. В.И.Ульянова-Ленина»
420008, Казань, ул. Кремлевская, 18

Отпечатано в лаборатории оперативной полиграфии КГУ
420045, Казань, Кр.Позиция, 2а
Тел. 72-22-54