Кафедра конструирования и производства радиоаппаратуры

УТВЕРЖДАЮ
Заведующий кафедрой КИПР
_____В.Н. ТАТАРИНОВ
“___” _____2012 г.

Типовые приемы программирования

Лабораторная работа по дисциплинам **«**Информатика**»** для студентов специальностей 211000.62 «Конструирование и технология электронных средств» (бакалавриат) и 162107.65 «Информатика и информационные технологии» (специалитет)

Разработчик:
Доцент кафедры КИПР
_____**Ю.П. Кобрин**

Томск 2012

СОДЕРЖАНИЕ

1 Цели работы

✓ Совершенствование навыков работы в системе ***Borland Pascal*** и освоение характерных приемов программирования.

✓ Закрепление навыков практической работы в интегрированной среде ***Borland Pascal***.

2 Порядок выполнения работы

1) Перед выполнением этой работы следует ознакомиться с приёмами программирования типовых задач (раздел 4 данной работы), а также, при необходимости, с дополнительной литературой [1] [2] [3] [4] [5] [6] [7] [8].

2) Получить у преподавателя индивидуальное задание. Разработать программу в соответствии с индивидуальным заданием.

3) Войти в свой личный каталог, загрузить и настроить систему ***Borland Pascal 7.0***.

4) Ввести шаблон программы, сделанный на предыдущих занятиях. Сделать его копию, записав под новым именем с помощью команды ***Save as*** ...

5) Ввести разработанную Вами программу, корректируя и дополняя свой шаблон.

6) Овладеть методикой поиска причин и исправления ошибок программы «по шагам» по тестовому примеру. Добиться, чтобы программа дала правильные результаты.

7) Оформить отчет по лабораторной работе и защитить его у преподавателя.

3 Защита отчета

Отчет должен быть выполнен в соответствии с [16] и состоять из следующих разделов:

✓ Тема и цель работы.

✓ Индивидуальное задание.

✓ Текст программы (в электронном виде), тестовые исходные данные и предполагаемые результаты тестирования.

✓ Результаты выполнения программы и выводы.

При защите отчета по работе для получения зачета студент должен:

✓ уметь отвечать на контрольные вопросы;

✓ обосновать структуру выбранного алгоритма и показать его работоспособность;

✓ уметь пояснять работу программы;

✓ продемонстрировать навыки работы в среде ***Borland Pascal***.

4 Характерные приемы программирования

4.1 Общие соображения

Опытные программисты знают, что практически любую программу можно в большинстве случаев сформировать из отдельных элементарных типовых подзадач, решение которых хорошо известно. И таких стандартных задач не так уж и много. Очевидно, чтобы научиться программировать, нужно овладеть способами решения подобных элементарных проблем.

4.2 Сохранение результатов

Во многих программах результатом вычислений является значение простой переменной, для записи которого в памяти компьютера выделяется одна ячейка. Если в процессе вычислений значение переменной изменяется, то после окончания работы остается лишь последнее значение результата. Чтобы записать все вычисленные значения, нужно выделить для их хранения необходимое количество ячеек памяти (массив), а текущий результат обозначить переменной с индексом (*рис. 4.1*).

> ***Массив*** – это ограниченная упорядоченная последовательность однотипных данных, имеющая общее имя. Для доступа к нужному компоненту массива используется его порядковый номер – индекс.

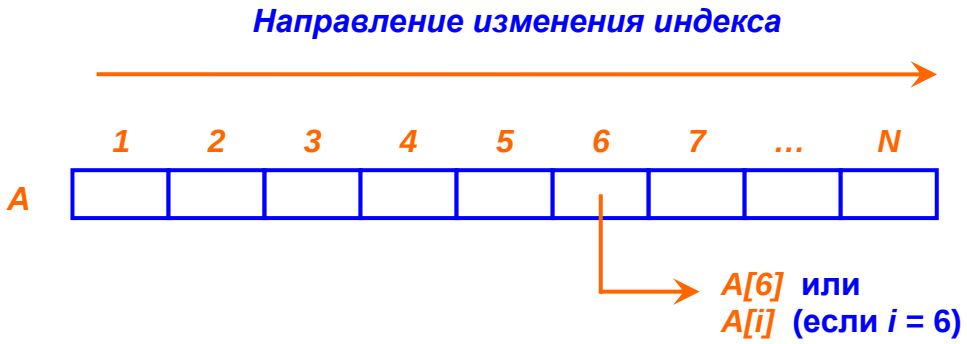


Рисунок 4.1 - Представление одномерных массивов (векторов) в компьютере

К необходимости применения массивов мы приходим всякий раз, когда требуется связать и использовать целый ряд родственных величин. Например, результаты многократных замеров напряжения на электрорадиоэлементе при изменении тока через него при снятии вольтамперной характеристики удобно рассматривать как совокупность вещественных чисел, объединенных в один сложный объект - массив измеренных напряжений.

При описании массива необходимо указать общее число входящих в массив компонентов и тип этих компонентов. Например:

```
var  {объявления массивов}
   A: array [1..10] of Real;      {массив A состоит из 10 компонентов типа Real}
   B: array [0..50] of Char;      {массив B состоит из 51 компонентов типа Char}
   C: array [-3..4] of Boolean;   {массив C состоит из 8 элементов типа Boolean}
```

Как видим, при описании массива используются зарезервированные слова ***array*** и ***of*** (массив, из). За словом ***array*** в квадратных скобках указывается тип-диапазон, с помощью которого компилятор определяет общее число компонентов массива. Тип-диапазон задается левой и правой границами изменения индекса массива. За словом ***of*** указывается тип компонентов, образующих массив. компонентами массивов могут быть как простые переменные любых типов[1], так и переменные составных типов (массивов, строк, записей, объектов и т.д.).

[1] Кроме ***LongInt*** и типов-диапазонов с использованием ***LongInt***

Доступ к каждому компоненту массива в программе осуществляется с помощью индекса - целого числа (точнее, выражения порядкового типа), служащего своеобразным именем компонента в массиве (если левая граница типа-диапазона равна 1, индекс компонента совпадает с его порядковым номером). При упоминании в программе любого компонента массива сразу за именем массива должен следовать индекс компонента в квадратных скобках, например:

```
var
   A: array [1..10] of Integer;   {массив A состоит из 10 элементов типа Integer}
   B: array [0..40] of Char;      {массив B состоит из 41 элемента типа Char}
   C: array [-2..2] of Boolean; {массив C состоит из 5 элементов типа Boolean}
   K: Integer;                      {индекс элементов массивов}
begin
   ...
   B[27] := 'Ж';                  {27 элементу массива B присваивается бука Ж}
   C[-2] := A[1] > A[2];          {-2 элементу массива C присваивается результат
                                   проверки условия A[1] > A[2]}
  for k : = 1 to 10 do  {обнуление массива A с помощью цикла for}
     A[k] := 0;

end.
```

В правильно составленной программе индекс не должен выходить за пределы, определенные типом-диапазоном. Например, можно использовать элементы *A[1], B[38], C[0],* но нельзя *A[0]* или *C[38]* (определение массивов см. выше). ***Borland Pascal*** может контролировать использование индексов в программе на этапе компиляции и на этапе счета программы, если установлена соответствующая опция.

Пример 1.

Вычислить и запомнить значения функции $z_i = \sqrt{\left(x_i^2 + 1\right)/i}$, где x_i – элементы массива $x_1, x_2, .., x_N$ $(N \le 100)$.

Решение.

```
Program Vector;   {Сохранение результатов с помощью массива }
Uses      {подключаем к программе стандартный модуль CRT }
  CRT;    {с процедурами TextColor, TextBackGround, ClrScr}

Const     {раздел описаний констант}
  Nmax = 100;        {максимальное число элементов массива}

Type   {раздел описаний нестандартных типов данных}
     {создаем новый тип данных – массив из Nmax вещественных элементов}
    tVector = array [1..Nmax] of real;

Var     {раздел объявления переменных}
  x,                    {массив аргументов}
  z : tVector;          {массив результатов}
  N,              {фактическое количество элементов в массиве}
  i:  integer; {индекс элемента в массиве}
```

```
begin          {начало основного блока программы Vector}
{Подготовка экрана дисплея}
  TextColor(Yellow);              {желтый цвет выводимых на экран символов}
  TextBackGround(Blue); {голубой цвет фона экрана}
  ClrScr;                         {очищаем экран от предыдущих результатов}

{Вывод назначения программы}
  writeln(' Таблица значений функции z[i] = ((x[i]*x[i] + 1) / i)^0.5');

{Диалог при вводе исходных данных}
  write('Фактическое число элементов в массиве (N <= ', Nmax, ') N : ');
  readln(N);
  for i := 1 to N do   {Цикл ввода исходных данных}
    begin      {начало тела цикла для ввода исходных данных }
  write('x[ ', i, '] : '); readln(x[i])    {введем  и запомним очередной аргумент}
    end;       {конец тела цикла для ввода исходных данных }

 {Цикл расчета функции и печати результатов в виде таблицы}
 writeln;      {вывод пустой строки перед заголовком таблицы}
 writeln('x':10, 'z':12);       {вывод заголовка таблицы}
 for i := 1 to N do    {Цикл расчета таблицы}
    begin      {начало тела цикла}
       z[i] := sqrt((sqr(x[i]) + 1) / i);          {запомним очередное значение функции}
  writeln(x[i]:10:2, z[i]:12:4)          {вывод строки таблицы}
    end;       {конец тела цикла}

end.    {конец основного блока программы Vector}
```

4.3 Нахождение сумм и произведений

Чтобы вычислить сумму значений некоторой функции $y = f(x)$ при различных значениях аргумента организуется цикл, в котором предусматривается не только вычисление значений функции, но и накопление суммы путем прибавления полученных слагаемых к сумме всех предыдущих слагаемых. Используется рекуррентная формула

$$Summa_i := Summa_{i-1} + y_i,$$

где $i = 1..N$ – номер цикла, $y_i = f(x_i)$, причем $Summa_0 = 0$.

С помощью этой формулы мы будем последовательно получать все промежуточные суммы:

$$Summa_1 = Summa_0 + f(x_1);$$

$$Summa_2 = Summa_0 + f(x_1) + f(x_2);$$

$$\ldots$$

$$Summa_2 = Summa_0 + f(x_1) + f(x_2) + \ldots + f(x_N).$$

Так как запоминать значения всех слагаемых и промежуточных сумм обычно не нужно, то в качестве $Summa$ и y достаточно использовать простые переменные. В этом случае накопление суммы можно вести в цикле по формуле

$$Summa := Summa + y,$$

где к старому значению суммы добавляется очередное вычисленное значение функции, после чего прежнее значение суммы замещается новым.

Предварительно, перед началом цикла следует присвоить формуле

Summa := 0,

и тогда после первого выполнения цикла ***Summa := y.***

Подобным же образом следует формировать и произведение конечного числа сомножителей. Необходимо лишь в цикле применять формулу ***P := P*y***, и ввести начальное значение произведения ***P := 1.***

Пример 2.

Ввести последовательность чисел[2] $x_1, x_2, \ldots, x_N$. Найти их сумму $S = \sum_{i=1}^{N} x_i$ и произведение $P = \prod_{i=1}^{N} x_i$.

Решение.

Программа может иметь вид:

```
Program AddMultiply; {Нахождение сумм и произведений последовательности чисел}
Var     {раздел объявления переменных}
  N,    {количество чисел в сумме и произведении}
  i:  integer;                {счетчик вводимых чисел}
  x,                          {текущее значение элемента последовательности чисел}
  Summa,                      {сумма чисел}
  P: real;                    {произведение чисел}

begin   {начало основного блока программы AddMultiply}
  {Вывод назначения программы}
   writeln('Вычисление суммы и произведения N чисел');

{Подготовка к суммированию и перемножению последовательности чисел}
  Summa := 0;                 {Начальное значение суммы чисел}
  P := 1;                     {Начальное значение произведения чисел}
  {Ввод последовательности чисел}
  write('Количество чисел N : '); readln(N); {Диалог при вводе N}
  writeln('Введите числа: ');     {Вывод приглашения ко вводу чисел}

for i := 1 to N do {Цикл ввода исходных данных, расчета сумм и произведений}
     begin                    {начало составного оператора для For i}
        write('x[', i, '] : '); readln(x);       {введем очередное число}
        Summa := Summa + x; {добавляем  его к предыдущему значению суммы}
        P := P * x                {умножаем его на предыдущее произведение}
  end; {конец составного оператора для For i }
  {Вывод результатов вычисления суммы и произведения чисел}
  writeln(' Сумма чисел  Summa = ', Summa:10:3);
  writeln('Произведение чисел P = ', P:10:3)
end.    {конец программы AddMultiply}
```

[2] Эта последовательность чисел может быть задана и массивом.

4.4 Вычисление бесконечных рядов

Для вычисления степеней, факториалов и суммы членов ряда используются рассмотренные ранее приемы накопления суммы и произведения. Так как заранее неизвестно, при суммировании скольких членов ряда будет достигнута требуемая точность, используются итерационные циклы. Выход из цикла достигается по условию достижения очередного члена ряда заданной точности ε.

Пример 3.

Вычислить с точностью до члена ряда, меньшего ε, сумму членов ряда

$$z = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!} = 1 + \sum_{n=1}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

Решение.

В целях уменьшения времени на вычисление значения очередного члена ряда целесообразно применить рекуррентные формулы, позволяющие вычислять его на основе знания предыдущего члена ряда.

```
Program Rjad;                { Вычисление бесконечных рядов }

Var     {раздел объявления переменных}
  n: integer; {номер члена ряда}
  u,                         {значение текущего члена ряда}
  z,                         {результат - сумма членов ряда}
  x,                         {аргумент ряда}
  eps: real;                 {погрешность вычисления суммы членов ряда}

begin                        {начало программы Rjad}

{Вывод назначения программы}
  writeln(' Вычисление суммы членов бесконечного ряда');

{Ввод исходных данных}
  write('Введите аргумент ряда   x : '); readln(x);
  write('Ведите погрешность    eps : '); readln(eps);

 {Подготовка к вычислению очередного члена ряда и нахождению их суммы}
  z  := 1;                   {Начальное значение суммы членов ряда}
  n := 1;                    {Номер члена ряда}
  u := 1;                    {Значение предыдущего члена ряда (начальное)}

{Цикл вычисления очередного члена ряда и нахождения их суммы}
  repeat   {Заголовок итерационного цикла}
      {вычисляем очередной член ряда - через предыдущий}
      u := - u * x * x  /(2* n * (2 * n - 1));
      z := z + u;        {добавляем к предыдущей сумме рассчитанный член ряда}
      n := n + 1;        {устанавливаем номер очередного члена ряда}
  until (abs(u) <= eps);       {проверка условия окончания итерационного цикла}

  {Вывод результатов вычисления суммы и произведения чисел}
  writeln(' Сумма членов ряда  z = ', z:10:3);
end.    {конец программы Rjad}
```

4.5 Поиск минимума и максимума методом общего поиска

Задача отыскания экстремумов ***целевой функции*** в условиях ограничений является весьма актуальной при оптимизации РЭС.

> ***Целевая функция*** (функция качества) – количественно выражает оптимизируемый показатель качества РЭС (масса, габариты, стоимость, коэффициент полезного действия и т.п.) в зависимости от варьируемых *проектных параметров*.

Нахождение наибольшего и наименьшего значения функции ***y = f(x)*** проще всего выполнить в цикле. При этом исходный интервал неопределенности $x_н$ .. $x_к$ (*рис. 4.2*) делится на *N* равных частей с последующим вычислением текущих значений целевой функции в полученных узлах сетки. Каждое из полученных значений целевой функции сравнивается с наибольшим или наименьшим из всех предыдущих значений функции. Если текущее значение функции окажется больше (меньше) наибольшего (наименьшего) из предыдущих значений функции, то его надо считать новым наибольшим (наименьшим) значением. В противном случае наибольшее (наименьшее) значение остается прежним.

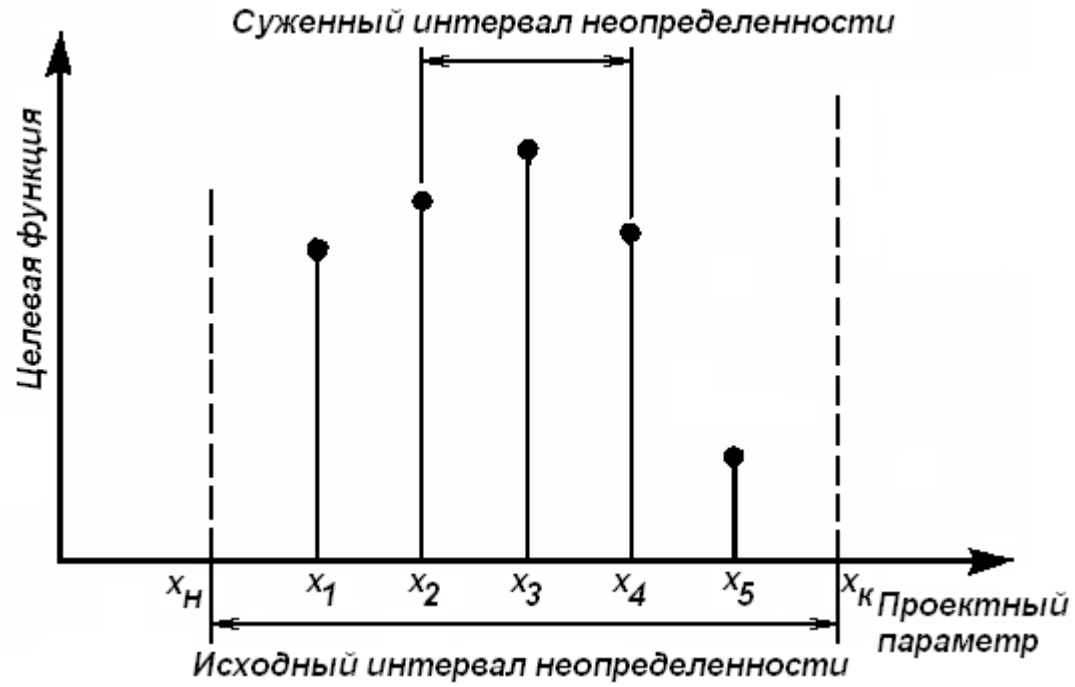


Рисунок 4.2 - Метод общего поиска

Сказанное можно описать математической формулой:

$$y_{\max} = \begin{cases} y_i, \ если \ y_i > y_{\max}; \\ y_{\max}, \ если \ y_i \le y_{\max} \end{cases} (i = 1,2,\ldots,n).$$

Аналогично, для наименьшего значения:

$$y_{\min} = \begin{cases} y_i, \ если \ y_i < y_{\min}; \\ y_{\min}, \ если \ y_i \ge y_{\min} \end{cases} (i = 1,2,\ldots,n).$$

До выполнения первого цикла в качестве начального значения y_{max} следует взять какое-либо значение функции или число, заведомо меньше любого значения функции (например, -10^{10}). Аналогично, для поиска y_{min} нужно взять число заведомо больше любого значения функции (например, $+10^{10}$).

Можно также в качестве начального значения y_{max} или y_{min} использовать первое вычисленное значение функции.

В теории оптимизации рассмотренный метод поиска экстремумов целевой функции называется *методом общего поиска*. Несмотря на низкую эффективность, метод общего поиска имеет важное достоинство – используемая целевая функция может иметь много экстремумов. Очевидно, точность поиска экстремума будет тем выше, чем меньше шаг расчета функции.

Пример 4.

Найти наибольшее и наименьшее значение напряжения, заданного целевой функцией $u = U_m \cdot e^{-b \cdot t} \cdot \sin(\varpi \cdot t + \phi)$, при изменении $t = 0 .. t_{max}$ с шагом Δt.

Решение.

```
Program MinMax;  {Программа поиска минимума и максимума }

Var   {раздел объявления переменных}
  u,                          {мгновенное напряжение}
  uMin, uMax,                 {минимальное и максимальное напряжения}
  Um,                         {амплитуда напряжения}
  b,                          {коэффициент затухания}
  Omega,                      {круговая частота}
  Psi,                        {сдвиг фазы}
  t,                          {текущее время}
  tMin, tMax,                 {минимальное и максимальное расчетное время}
  Delta_t: real;              {шаг изменения времени}

begin   {начало основного блока программы MinMax }

  {Вывод назначения программы}
  writeln('Поиск минимума и максимума напряжения');
  writeln('u := Um * exp(-b * t) * sin(Omega * t + Psi)');

  {Ввод исходных данных}
  writeln('Введите исходные данные:');
  write('Амплитуда напряжения, В          Um : '); readln(Um);
  write('Коэффициент затухания              b : '); readln(b);
  write('Круговая частота, рад/сек      Omega : '); readln(Omega);
  write('Сдвиг фазы, рад                  Psi : '); readln(Psi);
  write('Минимальное расчетное время   tMin : '); readln(tMin);
  write('Максимальное расчетное время tMax : '); readln(tMax);
  write('Шаг изменения времени         Delta_t : '); readln(Delta_t);

{Подготовка к поиску в цикле минимума и максимума}
   uMin := 1e10;              {начальное значение минимума напряжения}
  uMax := -1e10;              {начальное значение максимума напряжения}
  t := tMin;                  {начальное значение текущего времени}
  {Цикл поиска минимума и максимума}
repeat  {Заголовок цикла}
     {расчет очередного значения мгновенного напряжения}
```

```
    u := Um * exp(-b * t) * sin(Omega * t + Psi);
    if u <  uMin
       then uMin := u;                    {запомним значение u, меньшее предыдущих}
   if u >  uMax
      then uMax := u;                     {запомним значение u, большее предыдущих}
   t := t + Delta_t;            {новое текущее время}
 until (t > tMax);              {проверка условия окончания цикла}

{Вывод результатов}
 writeln('Минимальное напряжение   uMin = ', uMin:10:3);
 writeln('Максимальное напряжение uMax = ', uMax10:3)

end.    {конец программы MinMax}
```

4.6 Уточнение корней нелинейных уравнений

Для многих алгебраических и трансцендентных уравнений аналитическим способом корни найти не удается. В этом случае используют приближенные *итерационные методы* (методы последовательных приближений), которые состоят из двух этапов:

1) ***отделение корней***. Устанавливается интервал изоляции ***i***-го корня $[a_i, b_i]$, (функция на концах интервала должна иметь разные знаки). Это может быть сделано, например, методом «прогонки» (см. *пример 1*) или аналитическим путем. Находится грубое исходное приближение корня $x_0 = (a_i + b_i) / 2$;

2) ***уточнение корней***. Приближенные значения корней итерационно уточняются до заданной точности.

Существует огромное число методов приближенного нахождения корней нелинейных алгебраических уравнений [12 - 14]. Простейшим из них является классический *метод последовательных приближений.*

Пусть требуется решить нелинейное уравнение, заданное в общей форме

$$f(x) = 0. \qquad (1)$$

Перепишем его в виде

$$x = \varphi(x). \qquad (2)$$

Это можно сделать либо выразив явно x из уравнения (1), либо добавив по x к обеим частям (1), предварительно умножив их на некоторый коэффициент λ, значение которого можно изменять для обеспечения сходимости метода:

$$x = x + \lambda \cdot f(x) = \varphi(x) \qquad (3)$$

Подставив начальное приближение x_0 в (2) получим

$$x_1 = \varphi(x_0).$$

Далее

$$x_2 = \varphi(x_1)$$ и т. д.

Продолжая этот процесс, получим:

$$X_{n+1} = \varphi(x_n), \text{ где } n = 0, 1, 2, \ldots . \quad (4)$$

Если в результате такого итерационного процесса получаются значения, все ближе сходящиеся к корню уравнения, то говорят, что метод итераций *сходится*. В противном случае метод итераций расходится.

Из теории известно, что метод итераций сходится, если выполняется условие

$$|\varphi'(x)| < 1. \quad (5)$$

Подставляя в (5) выражение (3), получим, что для обеспечения сходимости нужно, чтобы коэффициент λ был в границах

$$-2 < \lambda \cdot \varphi'(x) < 0 \quad \text{или} \quad -2 / \varphi'(x) < \lambda < 0. \quad (6)$$

Итерационный процесс следует закончить, если выполнится условие

$$|xn+1 - xn| < \varepsilon, \quad (7)$$

где ε - допустимая абсолютная погрешность определения корня.

Для проверки необходимости окончания цикла часто также используют условие

$$\left|\frac{x_{n+1} - x_n}{x_n}\right| \le \bar{\varepsilon}, \quad (8)$$

где $\bar{\varepsilon}$ - относительная погрешность определения корня.

Пример 5.

Найти корни уравнения $f(x) = 2x - 3 \ln x - 3 = 0$ с точностью $\varepsilon = 10^{-6}$.

Решение.

Это уравнение имеет два корня: $x^{(1)}$ на интервале **[0.5, 0.6]** и $x^{(2)}$ на интервале **[3.2, 3.3]**. Используя (3) получим итерационную формулу

$$x = x + \lambda f(x) = x = x + \lambda (2x - 3 \ln x - 3).$$

Условие сходимости обеспечивается при $2 < \lambda (2 - 3 / x) < 0$.

Подставляя границы изоляции корня $x^{(1)}$ при $x = 0.5$ имеем $0.5 > \lambda > 0$, а при $x = 0.6$ имеем $2/3 > \lambda > 0$. Окончательно: $0.5 > \lambda > 0$. Выбираем $\lambda = 0.1$.. В качестве начального приближения первого корня можно принять $x_0 = (0.5 + 0.6) / 2 = 0.55$.

Аналогично следует поступить и для второго корня

Чтобы исключить возможность «зацикливания», в программе предусмотрен аварийный выход после *nMax* итераций.

Программа может иметь вид:

```
Program Iteraz;  {Программа уточнения корней уравнения}
Var   {раздел объявления переменных}
  x, x1,                    {значения корня уравнения на двух соседних итерационных
                             циклах}
  Lambda,                   {коэффициент сходимости }
  eps: real;                {абсолютная погрешность вычислений}
  nMax,                     {максимально-допустимое число итераций}
  n: integer;               {счетчик итерационных циклов}

begin   {начало основного блока программы Iteraz }

{Вывод назначения программы}
  writeln('Уточнение методом простой итерации корней');
  writeln('нелинейного уравнения f(x) = 2x - 3 ln x – 3 = 0');

  {Ввод исходных данных}
  writeln('Введите исходные данные:');
  write('Приближенное значение корня       x: '); readln(x);
  write('Погрешность вычислений           eps: '); readln(eps);
  write('Коэффициент сходимости    Lambda: '); readln(Lambda);
  write('Максимальное число итераций nMax: '); readln(nMax);

  writeln; {Пропуск строки перед печатью результатов}

  {Итерационное уточнение корня уравнения}
  for n := 1 to nMax do          {Заголовок цикла}
     begin
        {расчет очередного уточненного значения корня}
        x1  :=  x + Lambda * (2*x – 3 * ln( x) –3);

    {Вывод уточненного значения корня, если достигнута абсолютная точность eps}
        if abs(x1 - x) <= eps
          then begin  {Печать результата}
                    write('Уточненное значение корня  x = ', x1);
                     writeln('  получено  после n = ', n,  ' итераций');
                    Halt {Конец расчетов, остановка}
                  end;
        x := x1 {подготовка к очередному циклу}
     end;

{Аварийная печать}
  writeln('За  nMax = ', nMax, ' циклов достигнута точность Delta = ', x1 - x);
  writeln('Последнее значение корня  x = ', x1);

end.    {конец программы Iteraz }
```

4.7 Вложенные циклы

Любой цикл, содержащий внутри себя один или несколько циклов, называется *вложенным*. Цикл, охватывающий другие циклы, называется *внешним*, а прочие циклы – *внутренними*. Правила организации как для внешнего, так и внутреннего цикла такие же, как и для простого цикла. Параметры вложенных циклов одновременно не изменяются. При всяком значении параметра внешнего цикла параметр внутреннего цикла принимает по очереди все свои значения.

Пример 6.

Вычислить суммы положительных элементов каждой строки таблицы $A_{10\times8}$.

Решение.

Для представления таблиц в компьютерах используют двумерные массивы – матрицы (*рис. 4.3*).

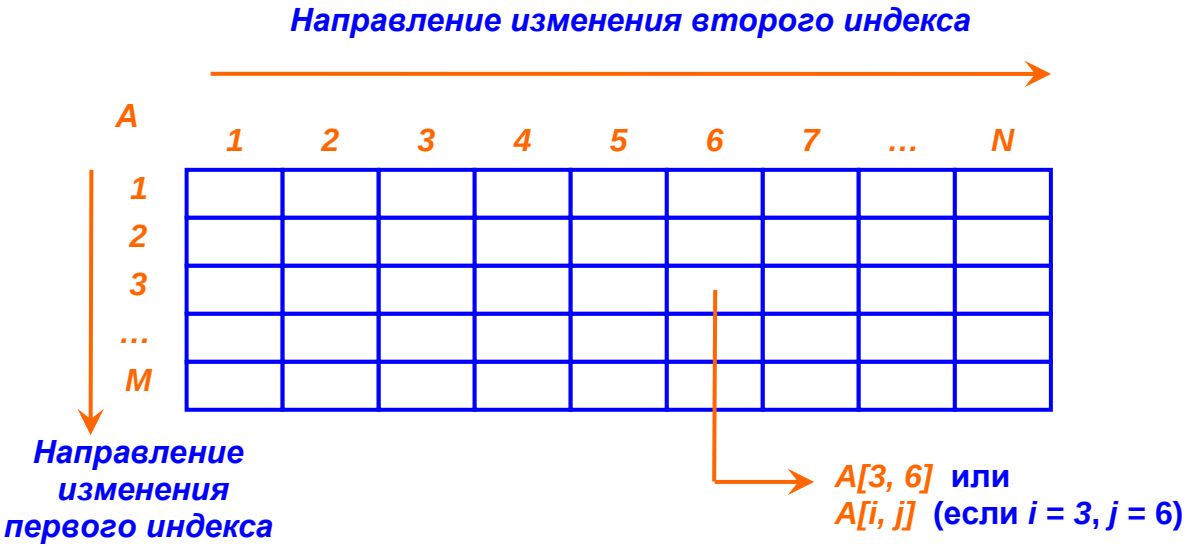


Рисунок 4.3 - Представление двумерных массивов (таблиц) в компьютере

Например, может быть создан тип

```
Type
  tMatriza = array [0..5, -2..2] of Byte; {Матрица 6×5 для целых чисел 0..255}
```

Глубина вложенности массивов - произвольная, поэтому количество элементов в списке индексных типов (размерность массива) не ограничено. Тем не менее, суммарная длина внутреннего представления любого массива не может быть больше 65520 байт. В памяти ПК элементы массива следуют друг за другом так, что *при переходе от младших адресов к старшим наиболее быстро меняется самый правый индекс массива*.

Если, например,

```
var
  A : array[1. .2, 1. .2] of Byte;
begin
  A[1, 1] := 1;
  A[2, 1] := 2;
  A[1, 2] := 3;
  A[2, 2] := 4;
end.
```

то в памяти последовательно друг за другом будут расположены байты со значениями ***1, 3, 2, 4 .***

Чтобы вычислить суммы положительных элементов каждой строки матрицы[3] необходимо организовать цикл перебора строк (изменяем индекс строки ***i = 1 .. N***). Внутри этого цикла следует организовать цикл перебора и суммирования положительных элементов строки (меняем индекс столбца ***j = 1 .. M***). Так как таких сумм столько же, сколько и строк, то для их хранения сумм положительных эле-

[3] Учебный вариант программы. Строго говоря, более «простодушный» вариант – вообще обойтись без запоминания элементов матрицы и массива сумм.

ментов строк предусмотрим одномерный массив. Очевидно, что размерность его равна количеству строк.

Программа может иметь вид:

```
Program ZiklZikl;                {Вложенные циклы}

Const {раздел объявления констант}
   Nmax = 20; {максимальное число строк в матрице}
   Mmax = 20; {максимальное число столбцов в матрице}
Type {раздел объявления нестандартных типов данных}
   tStolb = array [1..NMax] of real;  {массив длиной NMax - столбец матрицы}
   tMatr = array [1..MMax] of tStolb;       {массив длиной Mmax из столбцов матрицы}

Var     {раздел объявления переменных}
  A: tMatr;                      {прямоугольная матрица}
  Summa: tStolb; {массив сумм строк}
  N, M,                          {фактические количества строк и столбцов в матрице A}
  i, j: integer;                 {текущие номер строки и столбца}

begin                            {начало основного блока программы ZiklZikl }

  {Вывод назначения программы}
 writeln('Расчет сумм положительных элементов');
   writeln('в строках прямоугольной матрицы A[N * M]');
   writeln; {Пропуск строки}

  writeln('Введите исходные данные:');
  write('Число строк в матрице      (не более ', NMax, ')  N: '); readln(N);
  write('Число столбцов в матрице (не более ', MMax, ') M: '); readln(M);
  writeln(Введите элементы матрицы:);
  for i := 1 to N do               {Заголовок внешнего цикла}
    begin  {Начало тела внешнего цикла}
      Summa[i] := 0;               {Начальное значение суммы i-й строки}
      for j := 1 to M do           {Заголовок внутреннего цикла}
        begin  {Начало тела внутреннего цикла}
          {Диалог при вводе элемента матрицы A[i, j,
          стоящего на пересечении I – строки и j – го столбца}
          write('A[', i, ',', j, '] : '); readln(A[i, j]);
          if A[i, j] > 0   {если элемент положительный,}
            then Summa[i] := Summa[i] + A[i, j]; {то добавляем его к сумме предыдущих
                                 элементов строки}
        end  { Конец внутреннего цикла}
    end;  {Конец внешнего цикла}

  writeln; {Пропуск строки перед печатью результатов}
  for i := 1 to N do  {цикл вывода сумм элементов строк}
    writeln('Сумма положительных элементов ', i, '-й строки = ', Summa[i]);

end.    {конец программы ZiklZikl}
```

4.8 Сортировка элементов последовательности

При решении многих задач часто необходимо упорядочивать элементы последовательности по возрастанию или убыванию значений. Подпрограммы сортировки и поиска используются фактически во всех программах, работающих с базами данных, а также в компиляторах, интерпретаторах и в операционных системах.

Пример 7.

Упорядочить элементы массива $A = (a_1, a_2. \ldots, a_N)$, расположив их в порядке возрастания в том же массиве.

Решение.

Для решения задачи сортировки существует много различных алгоритмов. Для конкретного алгоритма большое значение имеет скорость сортировки. Скорость, с которой массив может быть упорядочен, прямо зависит от числа сравнений и числа необходимых операций обмена, причем операции обмена занимают большее время.

Рассмотрим один из простейших (не самый быстрый) – метод «пузырька», идея которого состоит в следующем.

Среди элементов последовательности выбирается наименьший (самый «легкий») элемент и ставится («всплывает») на первое место. При этом элемент, стоявший на первом месте, ставится на место наименьшего элемента - эти элементы *меняются местами*. Вслед за тем среди оставшихся элементов находится наименьший элемент и ставится на второе место и т.д. Процесс повторяется до тех пор, пока среди двух последних элементов последовательности в последний раз не будет выбран наименьший элемент и поставлен на предпоследнее место.

Программа может иметь вид:

```
Program Sort; {Сортировка элементов последовательности по возрастанию}

Const {раздел объявления констант}
  Nmax = 100; {максимальное число элементов в последовательности}

Var  {раздел объявления переменных}
  A: array[1..NMax] of real;  {массив с элементами Real длиной NMax}
  N,        {фактические количество  элементов в массиве A}
  I, k: integer;                    {переменные для организации циклов}
  R: Real;                          {вспомогательная переменная}

begin  {начало основного блока программы Sort }
     {Вывод назначения программы}
   writeln('Сортировка элементов массива по возрастанию');
   writeln; {Пропуск строки}

   writeln('Введите исходные данные:');
   write('Число элементов в последовательности   (не более ', NMax, ')  N: ');
   readln(N);
   writeln('Введите элементы последовательности:');
   for i := 1 to N do {Цикл ввода элементов последовательности}
      begin
         {Диалог при вводе элемента последовательности}
         write('A[', i, , '] : '); readln(A[i])
      end;
```

```
  writeln; {Пропуск строки перед печатью исходной последовательности}
  writeln('Неупорядоченная исходная последовательность элементов:');
 for i := 1 to N do {цикл вывода значений элементов неотсортированного массива}
    write(A[i]:5:2, ',  '); {выводим элементы друг за другом, в той же строке}
 writeln; {Переход на начало строки перед печатью результатов}

{Внешний цикл – перестановок элементов последовательности}
for i := 1 to N -1 do
   {Внутренний цикл - поиска наименьшего элемента}
   for k := i +1 to N do {просматриваем оставшиеся элементы A[k], начиная с k > i}
      if A[k] < A[i]  {Если элемент A[k] меньше A[i]-го, }
       then begin  {то меняем местами элементы A[k] и A[i] }
              R := A[k];  A[k] := A[i];  A[i] ;= R
            end;

writeln; {Пропуск строки перед печатью результатов}
writeln('Массив, отсортированный по возрастанию значений элементов:');
for i := 1 to N do  {цикл вывода элементов отсортированного массива}
   write(A[i]:5:2, ',  '); {выводим элементы друг за другом, в той же строке}

end.    {конец программы Sort }
```

4.9 Способы присвоения значений элементам массивов

В рассмотренных выше примерах были рассмотрен основной способ присвоения значений элементам массивов – ввод с помощью оператора ***Read***. Этот способ утомителен, требует повышенной концентрации внимания (особенно при вводе длинных последовательностей данных).

Нередко приходится в массив многократно вводить одну и ту же последовательность данных. Например, для тестового примера или для неизменно используемых в программе таблично заданных функциональных зависимостей. Облегчить решение подобных проблем можно путем инициализации массивов (присвоения начальных значений всем компонентам массивов) с помощью *типизированных констант.* Напомним, что типизированные константы приобретают указанные в их объявлениях значения лишь один раз: к моменту начала работы программы (на этапе компиляции). В ходе выполнения программы им можно присваивать другие значения, т.е. фактически они представляют собой *переменные с начальными значениями.*

Пример 8.

Проинициализировать значения одномерного массива ***A[10]***.

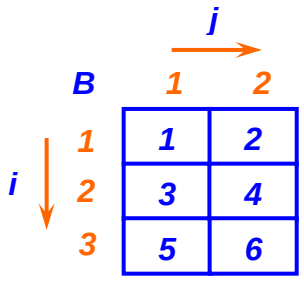
	1	2	3	4	5	6	7	8	9	10
A	0	2.1	4.0	5.65	6.1	6.7	7.2	8.0	8.7	9.3

Решение

```
type
  tVector10 = Array[1..10] of Real;
Const
    A: tVector10 = ( 0, 2.1, 4, 5.65, 6.1, 6.7, 7.2, 8, 8.7, 9.3 );
```

Пример 9.

Проинициализировать значения двумерного массива ***B[3×2]***



Решение

При инициализации двумерных массивов значения компонент каждого из входящих в него одномерных массивов записываются в скобках:

```
type
  tMatr3x2 = Array[1..3, 1..2] of Integer;
const
   B: tMatr3x2 = ( (1, 2), (3, 4), (5, 6) );
```

Нередко при проведении тестирования необходимо ввести несколько вариантов случайных последовательностей данных, причем важен лишь диапазон значений случайных чисел. В ***Borland Pascal*** имеются функции генерации как вещественных, так и целых псевдослучайных чисел (*табл. 4.1*). Причем, чтобы последовательности данных не повторялись, целесообразно перед использованием этих функций проинициализировать встроенный генератор псевдослучайных чисел случайным значением (текущим системным временем). Для этого нужно воспользоваться процедурой ***Randomize***.

Таблица 4.1 - Функции генерации псевдослучайных чисел

Заголовок функции	Назначение	Тип результата	Примеры использования
Random: real	Генерирует значение случайного числа из диапазона ***0 .. 1***	Вещественный	***Random⇒ 0.4876*** (случайное число)
Random(Range: word): word	Генерирует значение целого случайного числа из диапазона ***0 .. Range***	Целый	***Random(1000)*** ⇒ ***745*** (случайное число)

Пример 10.

Заполнить одномерный массива ***A[1000]*** целыми псевдослучайными числами в диапазоне ***±250.***

Решение.

Диапазон случайных чисел для этой задачи ***Diapason = 250 – (-250) = 500.*** Функция ***Random(500)*** будет генерировать псевдослучайные числа от ***0*** до ***500***. Для смещения диапазона в отрицательную область нужно от каждого случайного числа вычесть ***250***.

```
const
    N = 1000;                  {Количество компонентов в массиве}
    Diapason = 500;            {Диапазон значений случайных чисел}
    Diap2 = Diapason div 2;    {Смещение диапазона значений случайных чисел}

var
  A: array [1..N] of Integer;  {Массив чисел}
  i : Integer;                 {Индекс элемента массива}

…
{Наполняем в цикле  массив случайными числами и выводим их на экран:}
   for i := 1 to N do
      begin {Начало тела цикла}
         A[i] := random(Diapason) – Diap2;
          Write('A[',  i, '] = ',  A[i]:5)
      end;
 writeln; {Переход на начало строки}
…
```

Пример 11.

Заполнить двумерную таблицу ***A[5×8]*** вещественными псевдослучайными числами в диапазоне ***±50.***

Решение.

Диапазон случайных чисел для этой задачи ***Diapason = 50 – (-50) = 100.*** Функция ***Random*** будет генерировать псевдослучайные числа от ***0*** до ***1***. Для масштабирования их необходимо умножить на ***Diapason***. Для смещения диапазона в отрицательную область нужно от каждого случайного числа вычесть ***50***.

```
const
    N = 5;                     {Количество строк в таблице}
    M = 8;                     {Количество столбцов в таблице}
    Diapason = 100;            {Диапазон значений случайных чисел}
    Diap2 = Diapason div 2;    {Смещение диапазона значений случайных чисел}
var
  A: array [1..N, 1..M] of real; {Двумерный массив чисел}
  I, j : Integer;                {Индекс строк и столбцов элемента массива}
…
{Наполняем массив случайными числами и выводим их на экран:}
   for i := 1 to N do {внешний цикл – перебора строк}
     begin
        for j := 1 to N do {внутренний цикл – перебор столбцов}
          begin
             A[i, j] := random*Diapason – Diap2;
             Write(A[I, j]:8:2) {Выводим очередной элемент строки таблицы}
          end;
        writeln; {Переход на начало следующей строки}
   end;
…
```

5 Рекомендации по выполнению задания

При разработке программы предусмотреть:

✓ многократный ввод данных при исполнении программы, т.е. повторный счет при других исходных данных;

✓ простейший диалог при вводе данных типа «запрос-ответ»;

✓ вывод результатов в удобном для пользователя виде;

✓ использовать для отладки программы методику поиска причин и исправления ошибок «по шагам» по тестовому примеру.

6 Варианты индивидуальных заданий

1. Даны массивы с координатами ***N*** элементов (***x_i, y_i),*** расстояния между которыми рассчитываются по формуле $d_{i.j} = |x_i - x_j| + |y_i - y_j|$. Найти элементы, расстояние между которыми минимально.

2. Даны массивы с координатами ***N*** элементов (***x_i, y_i),*** расстояния между которыми рассчитываются по формуле $d_{i.j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. Найти элементы, расстояние между которыми максимально.

3. Даны массивы с координатами ***N*** элементов (***x_i, y_i),*** расстояния между которыми рассчитываются по формуле $d_{i.j} = |x_i - x_j| + |y_i - y_j|$. Переставить местами элементы, расстояние между которыми максимально.

4. Даны массивы с координатами ***N*** элементов (***x_i, y_i),*** расстояния между которыми рассчитываются по формуле $d_{i.j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. Переставить местами элементы, расстояние между которыми минимально.

5. Найти номер строки с минимальной суммой четных элементов целочисленной матрицы $A_{N\times M}$

6. Найти номер столбца с максимальной суммой нечетных элементов целочисленной матрицы $A_{N\times M}$.

7. Найти столбец с максимальным произведением положительных (ненулевых) элементов в столбце матрицы $A_{N\times M}$.

8. Найти строку с минимальным произведением отрицательных (ненулевых) элементов в строке матрицы $A_{N\times M}$

9. Вычислить сумму отрицательных элементов матрицы $A_{N\times M}$., расположенных над главной диагональю.

10. Даны матрица $A_{N\times M}$, состоящая из латинских букв. Отсортировать каждую строку в алфавитном порядке.

11. Даны матрица $A_{N\times M}$, состоящая из вещественных чисел. Расставить строки таким образом, чтобы элементы в первом столбце были упорядочены по убыванию.

12. Вычислить произведение положительных (ненулевых) элементов матрицы $A_{N\times M}$ расположенных под главной диагональю.

13. Определить и запомнить количества положительных и отрицательных элементов каждой строки матрицы $A_{N\times M}$.

14. Вычислить наибольшие значения функции $y_i = 3 \cdot e^{b_i x - 5x^2}$, если b_i задано массивом ***B*** = (***b_1, b_2, …, b_8***). Аргумент -2 < ***x*** < 2 меняется с шагом 0.1. Все $y_{\max}$ запомнить в массиве ***C*** = ***(c_1, c_2, …, c_8)***.

15. Определить количества положительных и количество отрицательных элементов кратных трем в каждом столбце матрицы $A_{N\times M}$..

16. Определить суммы произведений положительных и отрицательных элементов каждой строки матрицы $A_{N\times M}$.

17. Определить положение максимального элемента в строке матрицы $A_{N\times M}$, имеющей минимальную сумму положительных элементов.

18. Найти положение минимального элемента в столбце матрицы $A_{N\times M}$., имеющим максимальную сумму положительных элементов.

19. Упорядочить строки в матрицы $A_{N\times M}$ по возрастанию сумм элементов строк.

20. Упорядочить столбцы в матрицы $A_{N\times M}$ по убыванию сумм элементов столбцов.

21. Найти максимальный элемент в столбце с минимальной суммой элементов столбцов матрицы $A_{N\times M}$ и рассчитать произведение ненулевых сумм столбцов.

22. Найти сумму и произведение наибольших неотрицательных ненулевых элементов каждой строки матрицы $A_{N\times M}$.

23. Найти сумму и произведение наименьших положительных ненулевых элементов каждого столбца матрицы $A_{N\times M}$.

24. Перемножить матрицы $A_{N\times M}$ и $B_{M\times L}$. Элементы результирующей матрицы С вычислять по формуле $c_{i,k} = \sum_{j=1}^{M} a_{i,j} \cdot b_{j,k}$.

7 Список литературы

1. **Рапаков Г.Г., Ржеуцкая С.Ю.** Программирование *на языке Pascal.* — СПб. : БХВ-Петербург, 2004. - 480 с.

2. **Фаронов , В.В.** Программирование на персональных ЭВМ в среде Турбо-Паскаль. :, . - М. : Изд-во МГТУ, 1990.-580 с.

3. **Павловская , Т.А.** Паскаль. Программирование на языке высокого уровня: Учебник для вузов. — СПб. : Питер, 2007. — 393 с.

4. **Фаронов , В.В.** Турбо Паскаль 7.0. Начальный курс. Учебное пособие. -М. : ОМД Групп, 2003. - 616 с.

5. **Рютген Т., Франкен Г.** *Турбо Паскаль 7.0 .* — К. : Торгово-издательское бюро BHV, 1996-448 с.

6. **Попов, В.Б.** *Паскаль и Дельфи. Самоучитель.* — СПб. : Питер, 2004. — 544 с.

7. **Д. Мак-Кракен, У. Дорн.** *Численные методы и программирование на ФОРТРАНЕ.* – М. : МИР, 1977. – 583 с.

8. **Мудров, А.Е. .** Численные методы для ПЭВМ на языках Бейсик, Фортран и Паскаль. - Томск : МП "РАСКО", 1991. - 272 с.