Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра вычислительных методов и программирования

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ OBJECT PASCAL В СРЕДЕ DELPHI

Лабораторный практикум по курсу
«Основы алгоритмизации и программирования»
для студентов всех специальностей заочной формы обучения

В 2-х частях
Часть 2

Минск БГУИР 2009

УДК 681.3.06 (075.8)
ББК 32.973-018 я73
О-75

Авторы:
А. В. Аксенчик, И. Н. Коренская, А. А. Навроцкий, В. П. Шестакович

Основы программирования на языке Object Pascal в среде DELPHI : О-75 В 2 ч. Ч. 2 : лаб. практикум по курсу «Основы алгоритмизации и программирования» для студ. всех спец. заоч. формы обуч. / А. В. Аксенчик [и др.]. – Минск : БГУИР, 2009. – 52 с. ил.
ISBN 978-985-488-359-5 (ч. 2)

В лабораторном практикуме даны краткие теоретические сведения по основам программирования на языке Object Pascal в среде DELPHI, рассмотрены простейшие алгоритмы. В конце каждой темы приведены индивидуальные задания.

УДК 681.3.06 (075.8)
ББК 32.973-018 я73

Часть 1 издана в БГУИР в 2006 г. Авторы А. А. Бурцев, А. А. Навроцкий, В. П. Шестакович.

ISBN 978-985-488-359-5 (ч. 2)
ISBN 978-985-488-360-1

СОДЕРЖАНИЕ

ТЕМА 1. УКАЗАТЕЛИ И ИХ ИСПОЛЬЗОВАНИЕ ПРИ РАБОТЕ С ДИНАМИЧЕСКИМИ МАССИВАМИ

Цель работы: изучить способы работы с динамическими массивами данных.

1.1. Статическое и динамическое распределение оперативной памяти

Все команды и данные программы во время ее выполнения размещаются в определенных ячейках оперативной памяти. Часть данных записывается в ячейки памяти еще на этапе компиляции, и в процессе работы программы их адреса не изменяются относительно начала программы. Такое размещение данных и команд называется статическим, и соответствующие этим данным переменные называются ***статическими переменными***.

В языке Pascal возможна организация ***динамического размещения данных***, при которой под некоторые данные и подпрограммы память выделяется непосредственно по мере надобности во время их выполнения. После решения требуемой задачи память освобождается для других данных. Соответствующие таким данным переменные называются ***динамическими переменными***.

1.2. Понятие указателя

Для организации динамического распределения памяти используются переменные специального типа – указатели, которые обеспечивают работу непосредственно с адресами ячеек памяти. Под каждую переменную типа указатель отводится ячейка объемом **4** байта, в которую можно поместить адрес любой переменной.

Указатели объявляются следующим образом:

Var список_указателей : ^тип; // *типизированные указатели*

или **список_указателей : Pointer;** // *нетипизированные указатели*

Например:

```
type Vec = Array [1..20] of Integer;
Str = String[20];
Var a,b:^Extended;
    k:^Byte;
    i:^Vec;
    S1,S2:^Str;
    p,q:Pointer;
```

где `p,q` – ***нетипизированные указатели***;

`a,b,i,k,S1,S2` – ***типизированные указатели*** содержат адрес, с которого начинают размещаться данные, *например*, в ячейках, начиная с адреса **a,** размещается переменная типа `Extended`.

Значениями указателей являются адреса переменных, размещенных в памяти. Значение одного указателя можно передать другому при помощи оператора присваивания, *например*:

```
p:=q;
a:=b;
```

Необходимо помнить, что в операторе присваивания типизированные указатели должны быть одинакового типа.

Используя нетипизированные указатели, можно передать адрес между указателями разного типа, *например*:

```
p:=i;     k:=p;
```

С типизированными указателями можно работать как с обычными переменными, *например*:

```
S^:='Иванов';  //по адресу S размещается строка 'Иванов'
i^[11]:=88;
k^:=25;
m:=i^[9]+k^;   //m – статическая переменная целого типа
```

Указатели одинакового типа можно сравнивать на равенство **=** и неравенство **< >**, *например*:

```
if a=b Then ...
```

или

```
if a<>b Then ...
```

Сравнение и передача адреса указателям возможна только после присвоения им конкретных значений (адресов).

Адрес указателю можно присвоить с помощью специальной функции **Addr (<переменная>)**.

Пример:

```
p:=Addr(m);
a:=Addr(n);
```

где `m, n` – статические переменные.

Удаление адреса указателя осуществляется с помощью специальной функции **Nil,** *например*: `p:=Nil;  a:=Nil`; при этом довольно часто в программах применяется проверка условия:

```
if p<>Nil Then …
```

1.3. Динамическое распределение памяти

Вся свободная от программ память компьютера представляет собой массив байтов, называемый ***кучей***. При необходимости использования программой дополнительной памяти применяются процедуры `New`(); или `GetMem`();

Формат процедуры `New`();**:**

New (<типизированный указатель>);

Например:

```
New(a);
```

Данная процедура находит в куче свободный участок памяти, размер которого позволяет разместить тип данных **a,** и присваивает указателю **a** значение адреса первого байта этого участка. Далее участок памяти закрепляется за программой, и с ним можно работать через созданную в программе переменную **a^**. Такие переменные называются **динамическими**. После того как необходимость работы с этой переменной отпадет, данный участок памяти освобождается процедурой

Dispose(<типизированный указатель>);

При работе как с типизированными, так и с нетипизированными указателями аналогичные действия выполняют процедуры

GetMem (<имя указателя>, <размер области памяти>);
FreeMem (<имя указателя>, <размер области памяти>);

Для определения размера выделяемой памяти используется функция **SizeOf (<имя или тип переменной>)**, возвращающая количество занимаемых байт.

Необходимо помнить, что память под указатель выделяется 8-байтными порциями, поэтому возможна нежелательная фрагментация.

1.4. Организация динамических массивов

Обычно динамическое выделение и освобождение памяти используется при работе с массивами данных.

С помощью процедур `GetMem`(); и `FreeMem`(); можно выделять и освобождать память, отводимую под **массивы с изменяемым размером – динамические массивы**. Для этого надо определить тип указателя на массив с минимальным размером, а затем выделить необходимое количество памяти под элементы массива. *Например*:

```
Type Mas=Array[1..2] of <тип элементов>;
Var a:^Mas;                 //объявляется указатель на массив a
    mt:Word;
     …
//вычисляется необходимое количество байтов для размещения одного элемента
 mt:=SizeOf(<тип элемента>);
  GetMem(a,mt*n);       //выделяется память под n элементов массива
  For i:=1 To n Do
    a^[i]:=i*i+3;       //вычисляются значения элементов массива
  …
  FreeMem(a,mt*n);      //освобождается память
```

При работе с динамическими массивами необходимо отключать проверку выхода индекса за пределы массива.

Начиная с версии Delphi 4, в Object Pascal введены динамические массивы, не требующие указания границ массивов, *например*:

```
Var a:Array of Extended;            //объявление одномерного a и
    b:Array of Array of Integer;//двумерного b динамических массивов
```

Выделение памяти и задание размерности массива осуществляется путем ***инициализации*** массива процедурой **SetLength (<имя динамического массива>, <длина массива>);** В многомерных массивах сначала указывается длина первого измерения, затем второго, третьего и т.д. Так как нижняя граница индекса массива будет равна **0,** то верхняя граница индекса соответствует значению <длина массива>**-1.**

Приведем *пример* инициализации одномерного динамического массива `a` и двумерного динамического массива `b`:

```
SetLength(a,n);   //инициализация одномерного динамического массива a,
                  //состоящего из n элементов типа Extended
```

```
SetLength(b,n);   //инициализация двумерного динамического массива b,
For i:=0 To n-1 Do            //размерностью n строк и m столбцов
        SetLength(b[i],m);
```

Так как длина каждой строки задается отдельным оператором, то она может быть разной, *например*:

```
SetLength(b,n);
For i:=0 To n-1 Do
  SetLength(b[i],i+1); //выделение памяти под треугольную матрицу
```

или:

```
    SetLength(b,3);
    SetLength(b[0],8);      //длина 1-й строки равна 8
    SetLength(b[1],2);      //длина 2-й строки равна 2
    SetLength(b[2],4);      //длина 3-й строки равна 4
```

Так как имя динамического массива является указателем, то после завершения работы с массивом надо освободить память, используя процедуру

Finalize(<имя динамического массива>);

или оператор **<имя динамического массива> := Nil;**

Процедуру `SetLength()` в процессе выполнения программы можно вызывать произвольное количество раз. Каждый вызов приводит к изменению длины массива, причем **содержимое массива сохраняется**. Если при вызове `SetLength()` длина массива увеличивается, то добавленные элементы заполняются произвольными значениями, так называемым *мусором*. Если длина массива уменьшается, то содержимое отброшенных элементов теряется. Для работы с динамическими массивами Object Pascal можно использовать функции **Low()**, **High()**, **Copy()**.

Функции **Low(<имя динамического массива>)** и **High(<имя динамического массива>)** возвращают наименьшее и наибольшее значения индекса динамического массива, т.е. `0` и `длина-1` соответственно. Для **пустого** массива возвращаемое функцией **High()** значение равно `-1`. Функция **Copy()** возвращает заданную часть массива и имеет вид

Copy (<имя динамического массива>, <начальное значение индекса>,
<количество копируемых элементов>);

1.5. Компонент TBitBtn

Компонент `TBitBtn` расположен на странице палитры компонентов `Additional` и представляет собой разновидность стандартной кнопки `TButton`. Его отличительная особенность – наличие растрового изображения на поверхности кнопки, которое определяется свойством `Clyph`. Компонент имеет свойство `Kind`, которое задает одну из `11` стандартных разновидностей кнопок. Кнопка `bkClose` закрывает главное окно и завершает работу программы.

1.6. Пример написания программы

Задание: дан массив, состоящий из символов. Преобразовать его по следующему правилу: поместить цифры в начало массива, а за ними – остальные символы. При этом необходимо сохранить взаимное расположение символов в массиве.

Результат выполнения программы приведен на рис. 1.1.

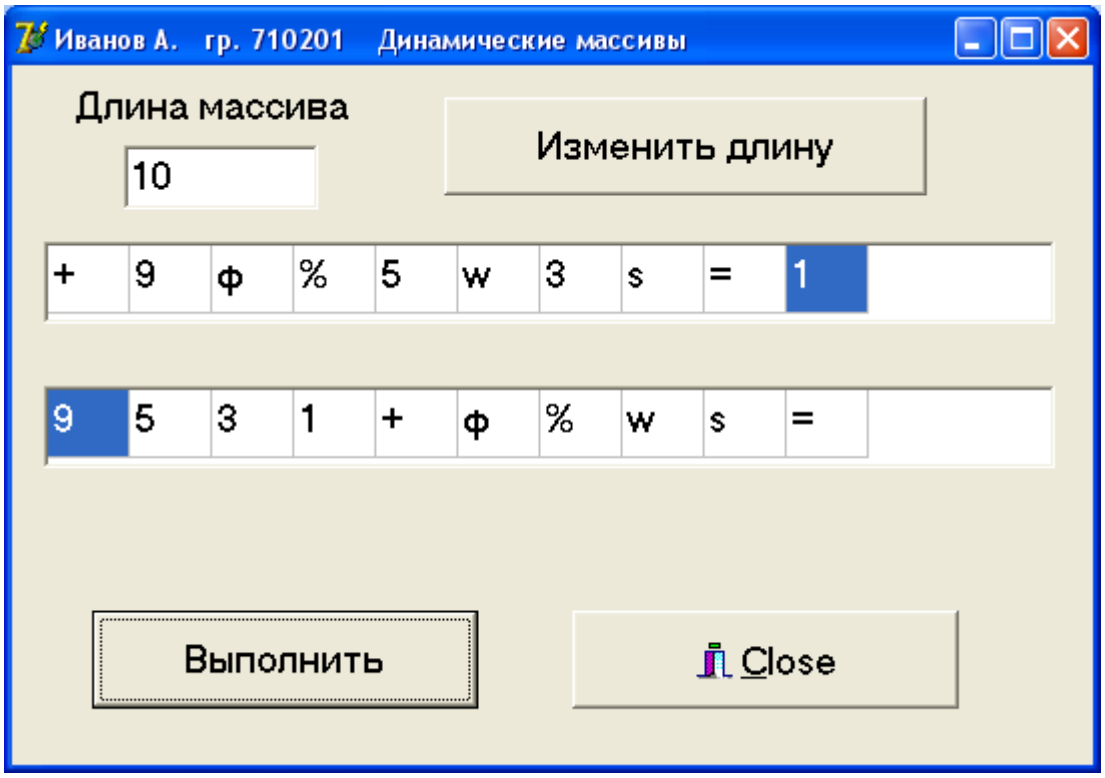


Рис. 1.1. Результат выполнения программы

Код программы имеет вид:

```
unit Unit1;
interface
uses Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls, Grids, But-
tons;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    BitBtn1: TBitBtn;
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
    Label1: TLabel;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
```

```
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

Var
  Form1:TForm1;
  n,k,i:Integer;
implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
Begin
  n:=3;
  Edit1.Text:=IntToStr(n);
  StringGrid1.ColCount:=n;
  StringGrid2.ColCount:=n;
End;

procedure TForm1.Button1Click(Sender: TObject);
Begin                              //измение длины массива
  n:=StrToInt(Edit1.Text);
  StringGrid1.ColCount:=n;
  StringGrid2.ColCount:=n;
End;

procedure TForm1.Button2Click(Sender: TObject);
   Var a,b : Array of Char;    //преобразование массива
       k,i : Integer;
Begin
  SetLength(a,n);              / выделение памяти под исходный a
  SetLength(b,n);              //и результирующий b массивы
  For k:=0 to n-1 Do           //считывание значений массива a
     a[k]:=StringGrid1.Cells[k,0][1]; //из StringGrid1
  i:=0;
  For k:=0 To n-1 do
     if a[k] in ['0'..'9'] Then Begin
                                  b[i]:=a[k];
                                  Inc(i);
                                End;
```

```
  For k:=0 To n-1 Do
     if Not(a[k] in ['0'..'9']) Then Begin
                                        b[i]:=a[k];
                                        Inc(i);
                                     End;
  For k:=0 To n-1 Do          //вывод массива b в StringGrid2
     StringGrid2.Cells[k,0]:=b[k];
   a:=Nil;  b:=Nil;           //освобождение памяти
End;

End.
```

1.7. Индивидуальные задания

По указанию преподавателя выберите вариант задания. Организуйте динамическое выделение памяти под массив. Введите данные из компонента `StrinGrid1`. Организуйте вывод результата в зависимости от варианта задания в компонент `StringGrid2` или в `Edit1`.

1. Дан массив, состоящий из символов. Расположить его элементы в обратном порядке.

2. Дан массив, состоящий из символов. Преобразовать его по следующему правилу: поместить латинские буквы в начало массива, а за ними – остальные символы. При этом необходимо сохранить взаимное расположение символов в массиве.

3. Дан массив, состоящий из символов. Вывести на экран цифру, наиболее часто встречающуюся в этом массиве.

4. Дан массив, состоящий из символов. Определить количество различных элементов массива (повторяющиеся элементы считать один раз).

5. Дан массив, состоящий из символов. Элементы массива циклически сдвинуть на k позиций влево.

6. Дан массив, состоящий из символов. Элементы массива циклически сдвинуть на n позиций вправо.

7. Дан массив, состоящий из чисел. Преобразовать его по следующему правилу: поместить отрицательные элементы в начало массива, а за ними – остальные значения. При этом необходимо сохранить взаимное расположение символов в массиве.

8. Элементы каждого из массивов X и Y упорядочены по возрастанию. Не используя сортировку, объединить элементы заданных массивов в массив Z так, чтобы они снова оказались упоряченными по возрастанию.

9. Дан массив, состоящий из символов. Определить, симметричен ли он, т.е. читается ли он одинаково слева направо и справа налево.

10. Даны два массива. Найти наименьшее значение среди тех элементов первого массива, которые не встречаются во втором массиве.

11. Дан массив, состоящий из символов. Заменить в нем строчные латинские буквы прописными.

12. Дан массив, состоящий из строчных латинских букв. Вывести на экран в алфавитном порядке все неповторяющиеся буквы.

13. Дан массив, состоящий из символов. Удалить из него повторные вхождения каждого символа.

14. Дан массив, состоящий из чисел. Удалить из него четные значения.

15. Дан массив, состоящий из чисел. Удалить из него отрицательные значения.

Контрольные вопросы и задания

1. Дайте определение указателя, динамического массива.
2. Какие бывают указатели? Приведите примеры объявления указателей.
3. Перечислите операции, допустимые над указателями.
4. Перечислите процедуры выделения и освобождения динамической памяти.
5. Приведите примеры выделения и освобождения памяти для одномерного и двумерного массивов.
6. Приведите пример динамического массива, не требующего указания границ.
7. Для чего применяются функции Low(), High(), Copy()?

ТЕМА 2. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ МНОЖЕСТВ

Цель работы: изучить правила составления программ с использованием данных типа «множество».

2.1. Краткие теоретические сведения

В математике *множество* – это некоторый набор или совокупность однотипных элементов. Над множествами допустимы следующие операции:

- **объединение множеств: C = A ∪ B**, где множество `C` содержит элементы, принадлежащие или `A`, или `B`, или `A` и `B` одновременно;
- **пересечение множеств: C = A ∩ B,** где множество C содержит элементы, принадлежащие и A, и B одновременно;
- **разность двух множеств: C = A \ B,** где множество C содержит элементы, принадлежащие A и не принадлежащие B.

Пример: `{1,2,3} ∪ {3,2,4}={1,2,3,4};`
`{1,2,3} ∩ {3,2,4}={2,3};`
`{1,2,3} \ {3,2,4}={1}.`

В языке Pascal под *множеством* понимают ограниченный, неупорядоченный набор различных элементов одного типа. В отличие от элементов массива элементы множества неупорядочены, поэтому следующие множества одинаковы: `{1,2,3,4},{4,3,2,1},{4,2,3,1}` и т.д. При этом можно рабо-

тать как с множествами-константами, так и с переменными типа «множество». Для задания множеств-констант в языке Pascal используется конструктор множества, представляющий собой квадратные скобки, в которые заключаются элементы множества: *например*, `[1,2,3,4],['a','c','w']`. Запись [] обозначает *пустое множество*, т.е. это множество, не содержащее ни одного элемента.

Множество-константу можно описать следующим образом:

Const имя=[<список констант>];

Например, `Const s=[1,3,5,7,9];`

Приведем *пример* инициализации типизированных констант:

```
Const  m1:Set of 0..9 =[0,2,4,6,8];
       m2:Set of Char=['a','+','3'];
```

Приведем формат описания переменных типа «множество»:

Type имя_типа=Set of <базовый тип элементов>;

Var имя_множества: имя_типа;

или

Var имя_множества: Set of <базовый тип элементов>;

где *базовый тип* – это тип элементов, входящих во множество. В качестве базового типа можно использовать любой порядковый тип. Так как количество элементов множества не должно превышать **256**, то в качестве целого типа можно использовать типы `Byte`, `ShortInt`, но нельзя - `Word`, `Integer`, `LongInt`.

Под *мощностью множества* понимается количество его элементов. Мощность пустого множества равна нулю.

Примеры:

```
Var a:Set of 'a'..'z';   //множество строчных латинских букв
    b:Set of Char;       //множество символов
    c:Set of Byte;       //множество чисел от 0 до 255
      …
    a:=['w','z'];          //задано множество из двух букв
    b:=['+','2','ф','z'];//задано множество из 4 символов
    c:=[32,77,3,18,43];  //задано множество из 5 чисел
```

Новые множества можно формировать, применяя операции над множествами. В Pascal имеются следующие операции над множествами:

- операция **объединения** множеств: **+** ;
- операция **пересечения** множеств: ***** ;
- **разность** двух множеств: **-** ;
- **сравнения: =, <>, >=, <=**;
- **проверки принадлежности: in**;
- **присваивания: :=** .

Поясним каждую из этих операций. Пусть имеются два однотипных множества `a` и `b`, а также переменная `x`:

```
a:=[ω2,ω4,ω1,ω5];          b:=[ω4,ω1,ω3];
```

Равенство и неравенство множеств. Множества равны между собой (выражение a=b – «истинно») тогда и только тогда, когда a и b содержат одни и те же элементы. Если a и b отличаются хотя бы одним элементом, то a и b не равны между собой (выражение a<>b – «истинно»).

Включение множества. Выражения a<=b или b>=a принимают значение «истинно», когда все элементы a являются также элементами b, и значение «ложно» – в противном случае.

Проверка принадлежности. Операция «проверка принадлежности» заключается в следующем: выражение x in a принимает значение «истинно», если значение переменной x принадлежит множеству a, и значение «ложно» – в противном случае. Тип переменной x должен быть таким же, как и базовый тип элементов множества a.

Присваивание значения. Оператор a:=b; означает, что переменной типа «множество» a присваивается текущее значение множества b.

Объединение множеств. Операция объединения множеств заключается в том, что множество a+b будет содержать элементы, которые принадлежат или a, или b, или обоим множествам одновременно:

d:=a+b; [ω1,ω2,ω3,ω4,ω5]

Пересечение множеств. Операция пересечения множеств заключается в том, что множество a*b будет содержать элементы, которые принадлежат и a и b одновременно: f:=a*b; [ω1,ω4]

Разность множеств. Разность множеств заключается в том, что множество a-b будет содержать элементы множества a, не входящие во множество b:

e:=a-b; [ω2,ω5]

В отличие от массивов к элементам множества нет прямого доступа по их номерам-индексам. Поэтому ввод элементов множества осуществляется с использованием операций объединения, а при выводе применяется операция принадлежности in .

При работе с множествами также используются следующие процедуры:

***Include (s, i)*;** – добавляет в множество s элемент i;

Exclude (s, i); – исключает из множества s элемент i.

Элемент i должен быть базового типа. Эти операции выполняются значительно быстрее, чем их эквиваленты: s:=s+[i]; s:=s-[i];

Использование множеств в ряде случаев позволяет в более компактном виде записать проверку условия, *например*, вместо оператора:

```
if (k=5) or (k=1) or (k=8) or (k=12) Then …
```

записать `if k in [5,1,8,12] Then …`

2.2. Пример написания программы

Задание: сформировать множество символов и выделить из него подмножество латинских букв.

Результат выполнения программы приведен на рис. 2.1. На форме расположены строка ввода `Edit1`, две кнопки `Button1` «`Добавить во множество`» и `Button2` «`Вывести`», два окна `Memo1`, `Memo2` с поясняющими надписями и кнопка `BitBtn1` «`Close`». После запуска программы в обработчике `FormCreate(Sender:TObject)` задается пустое множество `s:=[];`. Множество символов `s` (обработчик `Button1Click(Sender:TObject)`) формируется последовательно: при нажатии кнопки «Добавить во множество» из строки ввода `Edit1` добавляется в `s` очередной символ. При нажатии кнопки «Вывести» в обработчике `Button2Click(Sender:TObject)` из множества `s` при помощи операции «объединение множеств» выделяется подмножество латинских букв `r:=s*['a'..'z','A'..'Z'];`. Так как множество является ***неупорядоченной*** совокупностью данных одного типа, то для вывода результата используется цикл по всем допустимым символам множества с проверкой принадлежности текущего символа множеству `r`:

```
For c:=#0 To #255 Do
        if c in r Then Memo2.Lines.Add(c);
```

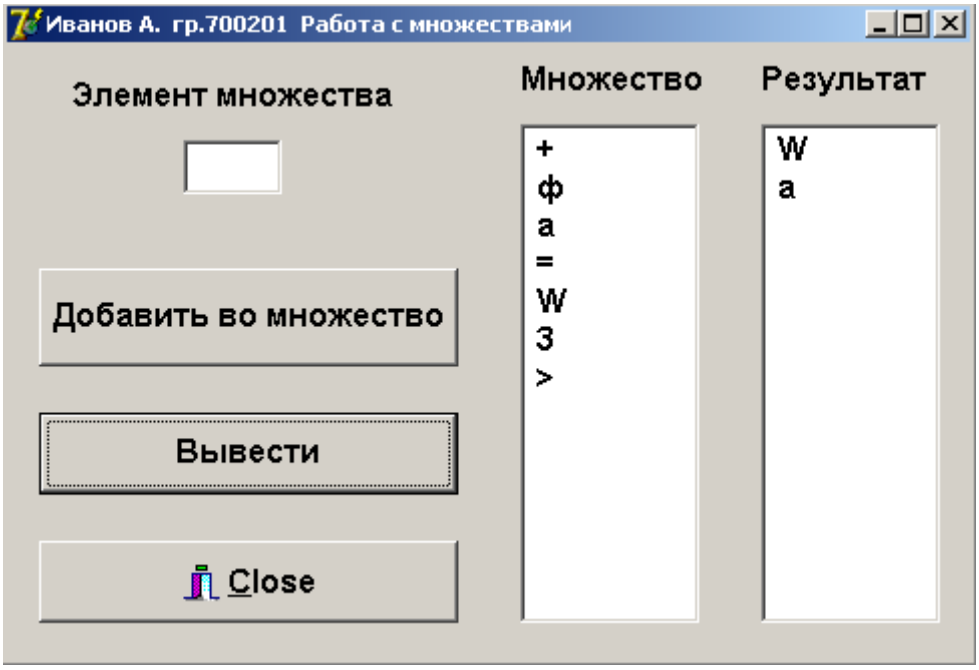


Рис. 2.1. Результат выполнения программы

Код программы имеет вид:

```
unit Unit1;
interface
uses Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, Buttons, StdCtrls;
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Label1: TLabel;
```

```
    Button1: TButton;
    Memo1: TMemo;
    Memo2: TMemo;
    BitBtn1: TBitBtn;
    Button2: TButton;
    Label2: TLabel;
    Label3: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  s,r:Set of Char;
  c:Char;
implementation
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
Begin
  s:=[];                    //задание пустого множества
  Memo1.Clear;    Memo2.Clear;
End;

procedure TForm1.Button1Click(Sender: TObject);
Begin                       //формирование множества
   c:=Edit1.Text[1];        //чтение символа
   Include(s,c);            //добавление символа во множество
   Memo1.Lines.Add(c);      //вывод символа в Memo1
   Edit1.Clear;             //очистка Edit1
   Edit1.SetFocus;          //установка фокуса в Edit1
end;

procedure TForm1.Button2Click(Sender: TObject);
begin              //выделение подмножества латинских букв и их  вывод
   Memo2.Clear;
   r:=s*['a'..'z','A'..'Z'];
   For c:=#0 to #255 Do //вывод результирующего множества
```

```
      if c in r Then Memo2.Lines.Add(c);
End;
End.
```

2.3. Индивидуальные задания

По указанию преподавателя выберите вариант задания. Введите данные из компонента `Edit1`. Организуйте вывод исходного множества и результата соответственно в компоненты `Memo1` и `Memo2`.

1. Сформировать множество символов и выделить из него подмножество гласных латинских букв.
2. Сформировать множество символов и выделить из него подмножество прописных латинских букв.
3. Сформировать множество символов и выделить из него подмножество строчных латинских букв.
4. Сформировать множество символов и выделить из него подмножество цифр.
5. Сформировать множество символов и выделить из него подмножество четных цифр.
6. Сформировать множество символов и выделить из него подмножество нечетных цифр.
7. Сформировать множество целых чисел и выделить из него подмножество чисел, кратных 3.
8. Сформировать множество целых чисел и выделить из него подмножество чисел, кратных 7.
9. Сформировать множество символов и выделить из него подмножество знаков препинания.
10. Сформировать множество символов и выделить из него подмножество скобок.
11. Сформировать множество символов и выделить из него подмножество знаков арифметических операций.
12. Сформировать множество символов и выделить из него подмножество латинских букв от `'k'` до `'w'`.
13. Сформировать множество символов и выделить из него подмножество латинских букв от `'b'` до `'f'`.
14. Сформировать множество символов и выделить из него подмножество знаков препинания и скобок.
15. Сформировать множество символов и выделить из него подмножество знаков препинания и арифметических операций.

Контрольные вопросы и задания

1. Дайте определение множества.
2. Приведите примеры описания переменных типа «множество».
3. Как задать множество-константу?
4. Как осуществляется ввод/вывод множеств-переменных?
5. Перечислите операции, допустимые над данными типа «множество».

ТЕМА 3. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ СТРОК

Цель работы: изучить приемы работы с данными строкового типа, приобрести навыки работы со строковыми процедурами и функциями Pascal, изучить правила перевода чисел из одной системы счисления в другую.

3.1. Краткие теоретические сведения

Строка – это последовательность символов кодовой таблицы ASCII. При использовании в выражениях строка-константа заключается в апострофы. Переменную строкового типа можно задать через описание типа в разделе определения типов или непосредственно в разделе описания переменных.

3.2. Описание переменных строкового типа

Короткие строки применялись в Pascal, вплоть до версии Delphi 1. Их длина ограничивалась **255** символами. В последующих версиях Delphi они стали описываться следующим образом:

Var sk : String [N]; //*N≤255*

или **Var sk : ShortString;** //*короткая строка максимальной длины*

ее эквивалент: **Var sk : String [255];**

Длинные строки. В силу того что короткие строки ограничены по длине (≤255), в них нельзя записать текст большой длины, что бывает неудобно. В последних версиях Delphi введены так называемые длинные строки. Для их описания используется ключевое слово String, *например*:

```
Var sd:String;
```

Длинные строки являются динамическими переменными (указателями), память для которых (до 2 Гб) выделяется по мере надобности на этапе выполнения программы.

Широкие строки. Они отличаются от длинных строк только тем, что каждый символ строки кодируется не одним, а двумя байтами памяти (ANSI кодировка позволяет работать только с 256 символами, что явно недостаточно, *например*, при работе с китайским или японским алфавитом). Для описания «широких строк» используется ключевое слово WideString, *например*:

```
Var ss: WideString;
```

Нуль-терминальные строки. Строковые переменные этого типа представляют собой цепочки символов, ограниченные символом конца строки #0. Для их описания используется ключевое слово PChar, *например*:

```
Var sc:PChar;
```

Необходимость в нуль-терминальных строках возникает только при прямом обращении к API-функциям ОС. При работе с компонентами Delphi в основном используются более удобные короткие и длинные строки.

Примеры описания строк:

```
Type Str=String[30];
```

```
Var  C1,C2:Str;
```
или
```
Var C3,C4:String[25];
```

Объем памяти, требуемый для размещения строки, равен ее максимальной длине плюс 1 байт, в котором запоминается длина данной строки.

Допустимо использование **типизированных констант строкового типа**, *например*:

```
Const  S1:String[5]='БГУИР';
       S2:String[4]=#107#116#102#44; {ktf.}
       S3:String[10]='Да';
```

Знак # означает, что символ представляется его ASCII кодом.

При работе со строковой переменной к отдельным символам строки можно обратиться с помощью индексов, указанных в квадратных скобках: S1[2],S2[3]. При этом символ с нулевым индексом S1[0] содержит значение, равное количеству символов в строке S1.

Выражения, в которых операндами служат строковые данные, называются *строковыми*. Они состоят из строковых констант, переменных, строковых функций и знаков операций. Если длина строкового выражения превышает максимальную длину строковой переменной, то все лишние символы справа отбрасываются.

Над строковыми данными допустимы операции: *присваивания* (:=), *сцепления* (+) и *отношения* (=,<>,>,<,>=,<=).

Операции отношения имеют приоритет более низкий, чем операция сцепления, т.е. вначале всегда выполняются все операции сцепления, если они присутствуют, и лишь потом реализуются операции отношения. Сравнение строк с помощью операций отношения производится слева направо до первого несовпадающего символа. Строка считается больше, если в ней первый несовпадающий символ имеет больший номер в кодовой таблице (см. прил. 2).

Тип результата выполнения операций отношения над строковыми операндами всегда будет логическим (Boolean) и принимает значение True, если выражение истинно, и False, если выражение ложно.

Строки считаются равными, если они полностью совпадают по текущей, а не по объявленной длине и содержат одни и те же символы.

3.3. Встроенные стандартные процедуры для обработки строк

Delete (Var S:String; Poz, L:Integer); – видоизменяет строку S, стирая L символов, начиная с символа с номером Poz.

Например:
```
S:='строка';      Delete(S,2,4);    //S='са'
```

После удаления подстроки ее оставшиеся части как бы склеиваются. Если Poz=0 или превышает длину строки S или L=0, то строка не изменится. При значении L, большем длины строки, будет удалена подстрока, начиная от Poz и

до конца строки. Такой прием применяют для «подрезания» строк до заданной величины.

Insert (Sv:String; Var S:String; Poz:Integer); – вставляет подстроку `Sv` в строку `S`, начиная с позиции `Poz`.

Например:

```
S:='Начало-конец';
Insert('середина-',S,8);  {Имеем S='Начало–середина–конец'}
```

3.4. Встроенные стандартные функции для обработки строк

Length (S : String):Byte – возвращает текущую длину строки `S`. Результат имеет целочисленный тип.

Concat (S1, S2,…, SN : String):String – выполняет слияние строк `S1, S2,…,SN` в том порядке, в каком они указаны в списке параметров.

Например: `Sum:=Concat(S1,S2,S3);`

Если сумма длин строк в `Concat` превысит объявленную длину строки, стоящей в левой части оператора присваивания, то лишние символы будут отсекаться. Следует помнить, что вместо `Concat` можно применять операцию сцепления. *Например,*

```
Sum:=S1+S2+S3;
```

Copy (S:String; Poz, L:Length):String – выделяет из строки `S` последовательность из `L` символов, начиная с позиции `Poz`. Если `Poz>Length(S)`, то функция вернет пустую строку, а если `L` больше, чем число символов от `Poz` до конца строки `S`, то вернется остаток строки `S` от `Poz` до ее конца.

Например: *'са'*

```
Sum:=Copy('ABC***123',4,3);    {Sum='***'}
Sum:=Copy('ABC',4,3);          {Sum=''}
Sum:=Copy('ABC***123',4,11);   {Sum='***123'}
```

Pos(S1, S:String):Byte – возвращает номер **первого** встретившегося символа в строке `S`, с которого начинается включение в `S` подстроки `S1`. Если `S` не содержит в себе `S1`, то функция вернет `0`. Недостатком функции `Poz` является то, что она возвращает позицию первого вхождения `S1` в `S`, т.е. вызов

```
Var P:Byte;
    .   .   .
    P:=Poz('abc','Nom abcabcabcfcd');
```

завершит свою работу, вернув значение 5, хотя есть еще и 8, и 11.

Pred(C:Char):Char – выдает предшествующий `C` символ.

Succ(C:Char):Char – выдает последующий за `C` символ.

Chr(X:Byte):Char – возвращает символ кодовой таблицы, код которого равен `x`.

Ord(C:Char):Byte – возвращает число, равное коду символа `C`.

3.5. Системы счисления

Под позиционной системой счисления понимают способ записи чисел с помощью цифр, при котором значение цифры определяется ее положением (порядком) в записи числа. Число `R` в `p`-ичной системе счисления можно представить в развернутом виде:

$R = a_n a_{n-1} \ldots a_1 a_0, a_{-1} \ldots a_{-k} = a_n p^n + a_{n-1} p^{n-1} + \ldots + a_1 p^1 + a_0 p^0 + a_{-1} p^{-1} + \ldots + a_{-k} p^{-k}$,

где a_i – цифры, p – основание системы счисления. Количество цифр равно p. Для записи цифр в общем случае может быть использован любой набор p символов. Обычно для $p \le 10$ используют символы 0…9, для $p > 10$ добавляют буквы латинского алфавита *A,B,C,D,E,F*, которым в десятичной системе соответствуют числа 10,11,12,13,14,15. *Например,*

$R = (D3,E)_{15} = D \cdot 15^1 + 3 \cdot 15^0 + E \cdot 15^{-1} = 13 \cdot 15^1 + 3 \cdot 15^0 + 14 \cdot 15^{-1} = (198,93)_{10}$;

$R = (124,5)_8 = 1 \cdot 8^2 + 2 \cdot 8^1 + 4 \cdot 8^0 + 5 \cdot 8^{-1} = (84,625)_{10}$.

В ПК обычно используются системы счисления с основанием, равным степени двойки: двоичная, восьмеричная и шестнадцатеричная. Имеются процессоры, реализующие троичную систему счисления. Для удобства пользователей ввод, вывод и операции над числами в компьютере производятся в десятичной системе счисления. Перевод целой и дробной частей числа из десятичной системы в другую осуществляется по разным правилам. При переводе целой части она делится на основание новой системы счисления, остаток представляет очередную цифру $a_0, a_1, \ldots, a_n$, а частное снова делится на основание. Процесс повторяется до тех пор, пока частное не станет равным нулю. Заметим, что цифры получаются в порядке, обратном порядку следования их в записи числа. При переводе дробной части она умножается на основание системы счисления. Целая часть полученного числа представляет очередную цифру $a_{-1}, a_{-2}, \ldots, a_{-k}$, а дробная часть опять умножается на основание системы счисления. Расчеты выполняют до получения заданной точности числа.

3.6. Пример написания программы

Задание: дана строка символов, состоящая из слов. Слова в строке могут быть разделены следующими символами: `' ','.',',',':',';','(',')','?','!','[',']','{','}'`. Выделить слова из строки и вывести их в алфавитном порядке.

Результат выполнения программы приведен на рис. 3.1. На форме расположены: компонент `ComboBox1`, в который вводятся данные, окно вывода `Memo1` для отображения результатов и кнопка `BitBtn1` «`Close`». Ввод строки заканчивается нажатием клавиши `Enter`. Завершение работы программы происходит при нажатии на кнопку `Close`.

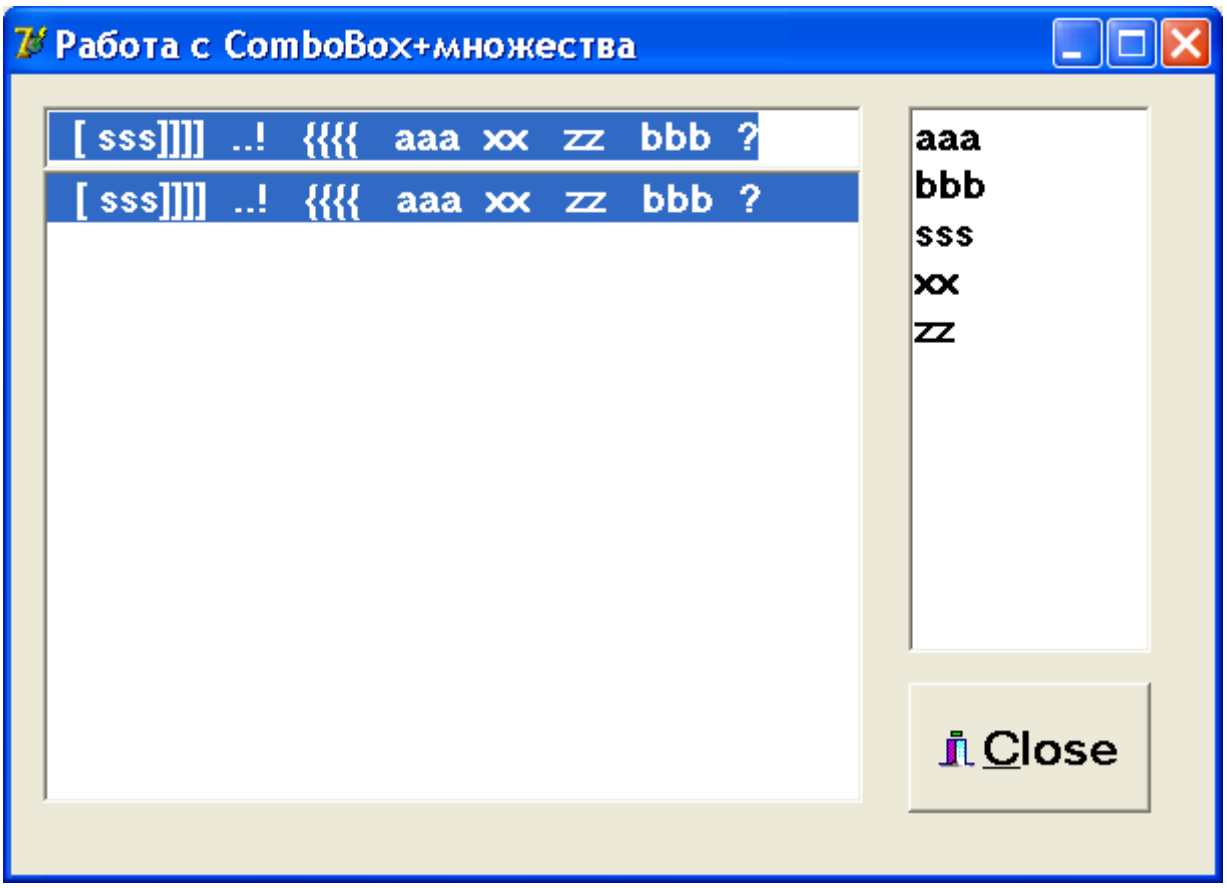


Рис. 3.1. Результат выполнения программы

Код программы имеет вид:

```
unit Unit1;
interface
uses Windows,Messages,SysUtils,Variants,Classes,
Graphics,Controls,Forms,Dialogs,StdCtrls,Buttons;
type
   TForm1 = class(TForm)
   ComboBox1: TComboBox;
   BitBtn1: TBitBtn;
   Memo1: TMemo;
   procedure FormActivate(Sender: TObject);
   procedure ComboBox1KeyPress(Sender: TObject; var
Key: Char);
    procedure ComboBox1Click(Sender: TObject);
        private
        { Private declarations }
    public
        { Public declarations }
  end;

var  Form1: TForm1;
```

```
implementation

{$R *.dfm}

procedure TForm1.FormActivate(Sender:TObject);
Begin                          //обработка события активации формы
   ComboBox1.SetFocus; //передача фокуса ComboBox1
   Memo1.Clear;
End;

procedure TForm1.ComboBox1KeyPress(Sender:TObject;
                                   var Key:Char);
Begin                 //обработка события нажатия левой клавиши мыши
    if Key=#13 Then   //проверка нажатия клавиши Enter
      Begin
        ComboBox1.Items.Add(ComboBox1.Text);   //строка из окна
                          //редактирования заносится в список выбора
        ComboBox1.Text:='';          //очистка окна редактирования
      End;
End;

procedure TForm1.ComboBox1Click(Sender: TObject);
Const c:Set of Char=[' ','.',',',':',';','(',')','?','!',
'[',']','{','}'];                    //список разделителей слов
Var s:String;
    n,k,i:Integer;
    a:Array[1..100] of String;
Begin
    s:=ComboBox1.Text;               //запись информации в строку s
    Memo1.Clear;
    s:=s+' ';
    n:=0;                            //счетчик количества слов
    While s<>'' Do
       Begin                         //удаление начальных разделителей
        While (s<>'') And (s[1] in c) Do Delete(s,1,1);
        if s<>'' Then
          Begin
           For k:=1 To Length(s) Do   //выделение слова
            If s[k] in c Then Break;
           n:=n+1;
           a[n]:=Copy(s,1,k-1);       //формирование массива слов
           Delete(s,1,k);    //удаление выделенного слова из строки
          End;   //if
```

```
        End;      //While
    For i:=1 To n-1 Do        //сортировка по алфавиту
        For k:=1 To n-i Do
             If a[k]>a[k+1] Then
                 Begin
                   s:=a[k];
                   a[k]:=a[k+1];
                   a[k+1]:=s;
                 End;
    For k:=1 To n Do
        Memo1.Lines.Add(a[k]);
End;
End.
```

3.7. Индивидуальные задания

По указанию преподавателя выберите вариант задания. Введите данные из компонента `ComboBox1`. Организуйте вывод результатов в окно вывода `Memo1`.

1. Дана строка символов, состоящая из слов, разделенных пробелами. Найти количество слов с пятью символами.
2. Дана строка, представляющая собой запись числа в четырнадцатеричной системе счисления. Преобразовать ее в строку, представляющую собой запись числа в десятеричной системе счисления.
3. Дана строка, представляющая собой запись числа в десятеричной системе счисления. Преобразовать ее в строку, представляющую собой запись числа в восьмеричной системе счисления.
4. Дана строка, представляющая собой запись числа в восьмеричной системе счисления. Преобразовать ее в строку, представляющую собой запись числа в двоичной системе счисления.
5. Дана строка символов, состоящая из произвольных десятичных чисел, разделенных пробелами. Вывести на экран числа этой строки в порядке возрастания их значений.
6. Дана строка, состоящая из слов, разделенных пробелами. Найти и вывести на экран самое короткое слово.
7. Дана строка, состоящая из слов, отделенных друг от друга одним или несколькими разделителями (пробелы, точки, запятые, скобки и пр. (см. п. 3.5)). Подсчитать количество символов в самом длинном слове.
8. Дана строка, состоящая из слов, отделенных друг от друга одним или несколькими разделителями (пробелы, точки, запятые, скобки и пр. (см. п. 3.5)). Найти и вывести на экран слова с четным количеством символов.
9. Дана строка, состоящая из слов, отделенных друг от друга одним или несколькими разделителями (пробелы, точки, запятые, скобки и пр. (см. п. 3.5)). Подсчитать количество символов в самом коротком слове.

10. Дана строка символов, состоящая из произвольных десятичных чисел, разделенных пробелами. Вывести четные числа этой строки.

11. Дана строка, представляющая собой запись числа в двоичной системе счисления. Преобразовать ее в строку, представляющую собой запись числа в шестнадцатеричной системе счисления

12. Дана строка символов, состоящая из произвольного текста, слова разделены одним или несколькими пробелами. Вывести на экран порядковый номер слова, накрывающего `k`-ю позицию (если на `k`-ю позицию попадает пробел, то номер предыдущего слова).

13. Дана строка, состоящая из слов, отделенных друг от друга одним или несколькими разделителями (пробелы, точки, запятые, скобки и пр. (см. п. 3.5)). Вывести на экран порядковый номер слова минимальной длины.

14. Дана строка символов, состоящая из произвольного текста, слова разделены пробелами. Вывести на экран порядковый номер слова максимальной длины и номер позиции в строке, с которой оно начинается.

15. Дана строка символов, состоящая из произвольного текста, слова разделены пробелами. Вывести на экран порядковый номер слова минимальной длины и количество различных символов в нем.

Контрольные вопросы и задания

1. Как описываются строковые данные?
2. Чему равна максимальная длина строковой переменной?
3. Какие операции допустимы над строковыми данными? Назовите их приоритет.
4. Какие выражения называются строковыми?
5. Какие стандартные процедуры и функции применяются в Pascal для работы со строковыми данными?

ТЕМА 4. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ЗАПИСЕЙ И ФАЙЛОВ

Цель работы: изучить правила работы с компонентами `TOpenDialog` и `TSaveDialog`. Изучить правила работы с типами «запись» и «файл». Составить и отладить программу с использованием файлов.

4.1. Понятие записи

Удобство использования структурированных типов, когда под одним именем объединяется целый набор данных, пояснялось ранее на примерах массивов, множеств и строк. Указанные типы объединяют под одним именем одинаковые данные. В отличие от них записи позволяют объединить под одним именем разнотипные данные, что предоставляет более широкие возможности для составления программ.

Запись – это структура данных, в которой под одним именем объединены переменные, возможно, разного типа, называемые полями.

Пример описания записи:

```
Type
  Tzap=Record
      a,b,c:Typ_1;          //разделы фиксированной части записи
      e,f:Typ_2;
                ...
      z,x,y,u:Typ_n;
      Case Byte of
      1:(d:Typ_01;          //разделы  вариантной части записи
         g:Typ_02);
               ...
         m:(p,q:Typ_1m)
       End;
Var z1,z2:Tzap;
```

где z1, z2 – записи, a,b,c,e,f,z,x,y,u,d,g,p,q – поля записей. Все поля в одной записи должны иметь различные имена. Записи могут иметь фиксированную и вариантную части. Вариантная часть указывается в конце записи и начинается с оператора Case <перечисляемый тип> of. В данной части перед константой приводятся возможные варианты полей, заключаемые в скобки. Всем вариантам отводится одна и та же область памяти, объем которой равен **максимальному** из объемов вариантов полей. Возможны записи, имеющие только фиксированную часть (отсутствует вариантная часть) или имеющие только вариантную часть (отсутствует фиксированная часть).

Запись-константа задается следующим образом:

```
Type Point=Record
                x,y:Real;
             End;
Const w:Point=(x:1.5;y:8.4);
```

4.2. Операции над записями

Для однотипных записей допускается оператор присваивания:

```
z1:=z2;
```

При выполнении указанного оператора всем полям записи z1 присваиваются значения соответствующих полей записи z2.

К полю записи можно получить доступ через составное имя записи, состоящее из имени записи и имени поля, разделенные точкой, *например*:

```
Type
    Typ_1=Record
             a,b:Extended;
         End;
Var z1,z2:Typ_1;
    …
   z1.a:=25.86;
```

```
z2.b:=z1.a;
```

Если поле в свою очередь состоит из записи (вложенность записей), то для обращения к полю записи потребуется составное имя с двумя точками. *Например*, если `Typ_2` определяется как

```
Type
    Typ_2=Record
            e: Record re,im:Extended; End;
          End;
Var z1:Typ_2;
```

то полям `re` и `im` можно присвоить значения следующим образом:

```
z1.e.re:=5.1;
z1.e.im:=0.8;
```

Для упрощения доступа к полям записи применяется оператор присоединения:

With <имя записи> Do
Begin
<операции с полями записи>
End;

Например, присвоить значения полям `re` и `im` можно и так:

```
With z1.e Do
 Begin
    re:=5.1;
    im:=0.8;
 End;
```

То обстоятельство, что все варианты записи помещены в одном месте памяти, позволяет использовать эффект наложения друг на друга переменных различных типов.

Приведем *пример* наложения одномерного массива на двумерный:

```
Type
 Zap=Record
   Case Boolean Do
    True:(a:Array[1..4]of Integer);
    False:(b:Array[1..2,1..2] of Integer);
End;
Var z:Zap;
          ...
  For i:=1 To 4 Do z.a[i]:=i;
  WriteLn(z.b[2,1]);          //будет выведено число 3
```

4.3. Понятие файла

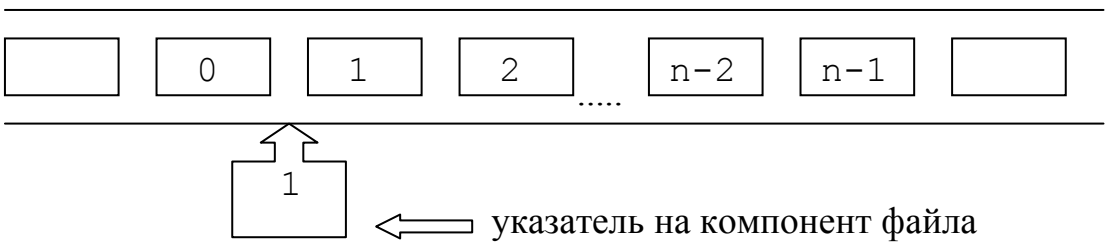
Файлом называется именованный участок внешней памяти. Файлы предназначены для размещения данных на внешних носителях и последующей работы с этими данными. В языке Pascal для организации и последующей работы с файлами предусмотрен специальный файловый тип переменных. Операциям

над переменными файлового типа соответствуют определенные действия над внешними носителями (дисками, принтерами и т.д.).

Переменная файлового типа, или коротко файл, в языке Pascal представляет последовательность однотипных компонент, соответствующих последовательности записей на внешнем носителе. В отличие от массива компоненты файла не имеют индекса и их количество заранее не оговаривается. Файловые переменные в Delphi описываются следующим образом:

```
Var
     ft1,ft2:File of<тип компонент>;   //типизированные файлы
     Lw,Lr:TextFile;                   //текстовые файлы
     f1,f2:File;                       //нетипизированные  файлы
```

Объясняя принцип работы с файлами, можно для наглядности считать, что каждый файл записан на некоторой магнитной ленте, как это показано на следующем рисунке:



n – количество записанных компонент.

Указатель определяет положение магнитной головки магнитофона, с помощью которой осуществляется покомпонентная запись или чтение информации. В начале файла записана информация о файле BOF (Begin of File), его имя, тип, длина и т.д., в конце файла помещается признак конца файла EOF (End of File). Если файл пуст, то BOF и EOF совмещены, а указатель установлен в ноль. Если файл не пуст, то указатель совмещен либо с началом некоторой компоненты и его значение равно номеру этой компоненты (нумерация начинается с нуля), либо указатель совмещен с признаком конца и его значение равно количеству компонент.

4.4. Операции над файлами

4.4.1. Типизированные файлы

Пусть f – имя типизированного файла, а переменные x,y,z имеют тот же тип, что и его компоненты.

```
Type
     Typ=<тип компонента файла f>;
     Var
        f:File of Typ;
        x,y,z:Typ;
```

Работа над файлами начинается с процедур открытия файла:

AssignFile (f, <имя файла> : String);

Reset (f); или **ReWrite (f);**

Процедура AssignFile(); устанавливает соответствие между файловой переменной f и <именем файла> на носителе. Процедуры Reset(f); и ReWrite(f); инициируют (подготавливают) файл для работы. Для создания нового файла с именем <имя файла> на носителе следует использовать оператор ReWrite(f); При работе с уже существующим файлом необходимо применять оператор Reset(f); После выполнения процедур открытия файла указатель всегда устанавливается в начало файла (на компоненту с номером 0). При открытии файла оператором ReWrite(f); вся информация, ранее записанная в файл, стирается и признак конца файла совмещается с его началом (файл пуст). Запись значений переменных в файл производится покомпонентно с помощью оператора

```
Write(<имя файловой переменной>,<список переменных>);
```

Например:

```
Write(f,x);
Write(f,y,z);
```

После записи каждой переменной в файл значение указателя увеличивается на единицу, что соответствует его перемещению к следующему компоненту файла. При добавлении переменной после последней записи в конец файла (указатель находится напротив признака конца файла), признак конца файла смещается на длину этой записи и количество компонентов в файле увеличивается на единицу.

Количество компонентов, записанных в файл, определяется функцией FileSize(f).

Чтение значений переменных из файла производится с помощью оператора Read(<имя файловой переменной>,<список переменных>); *Например*:

```
Read(f,x);
Read(f,y,z);
```

При чтении каждой переменной указатель увеличивается на единицу. Если производится попытка чтения при указателе, установленном на конец файла, то происходит останов программы из-за ошибки чтения из файла.

Определить конец файла позволяет функция EOF(f), возвращающая значение True, если достигнут конец файла, и False в противном случае. Приведем *пример* чтения из файла, если конец его еще не достигнут:

```
if Not EOF(f) Then Read(f,x);
```

Для чтения или записи заданного компонента файла указатель предварительно устанавливается на номер этого компонента при помощи процедуры **Seek(**<имя файловой переменной>**,<номер компонента>);**

Например:

```
Seek(f,2);
Read(f,x);
Write(f,y);
```

При выполнении этих операторов переменная x будет прочитана из компонента с номером 2 (третьего), а переменная y запишется в компонент с номером 3 (четвертый).

Текущее значение положения указателя определяется функцией

FilePos(<имя файловой переменной>).

После окончания работы с файлом его следует обязательно закрыть процедурой CloseFile(<имя файловой переменной>);

Иначе, в случае отсутствия данного оператора из-за специфики обмена данными с файлом, часть информации, помещенная в файл, может быть утеряна.

4.4.2. Текстовые файлы

Для удобства хранения текстовой информации, представленной последовательностью строк символов, в языке Pascal введены файловые переменные текстового типа (см. выше Lw, Lr) или просто текстовые файлы. Текстовые файлы можно представлять эквивалентным типизированным файлом, компонентами которого являются символы. Однако в текстовом файле последовательность символов разбита на строки различной длины, т.е. в конце каждой строки ставится специальный признак EOLN (End of LiNe), а в конце файла - признак конца файла EOF. При просмотре такого файла текстовым редактором на экране возникает естественная «книжная» страница текста, которую затем можно отредактировать. При работе с текстовым файлом следует помнить, что в отличие от типизированного после открытия такого файла процедурой Reset(Lr); разрешается только чтение из файла, *например*:

```
Read(Lr,a,b);
ReadLn(Lr,c);
ReadLn(Lr);
```

При открытии файла процедурой Rewrite(Lw); все записи файла стираются, и для файла становится возможна только операция записи. Для сохранения ранее записанной информации текстовый файл следует открывать процедурой Append(Lw); В этом случае указатель устанавливается на конец файла и последующие записи добавляются в конец файла.

Для записи в текстовый файл применяются операторы Write(<имя файловой переменной>,<список переменных>); и WriteLn(<имя файловой переменной>,<список переменных>); *например:*

```
Write(Lw,a,b);
WriteLn(Lw,c:8,'переменная d=',d:10:2);
```

Как видно из примеров, для текстовых файлов возможны модификации операторов ввода/вывода ReadLn/WriteLn, осуществляющие переход на следующую строку после чтения или записи указанных в операторах переменных. При этом оператор WriteLn(); добавляет после записи символ конца строки EOLN.

Заметим, что переменные a,b,c в приведенных примерах могут быть любого скалярного типа. При выводе информации в текстовый файл часто применяется форматирование, при котором происходит автоматическое преобразование чисел в их строковые представления. При вводе строковые представления чисел, разделенные пробелами, автоматически преобразуются в числовые данные в соответствии с типом переменных.

4.4.3. Нетипизированные файлы

Тип компонентов нетипизированного файла заранее не оговорен и представляется последовательностью байтов.

Такой файл описывается следующим образом:

Var <имя файловой переменной>**:File;**

Например, `Var f:File;`

Для открытия файла такого типа применяются процедуры:

Reset (f, Size);

или **ReWrite (f, Size);**

где **Size** – размер блока в байтах.

Нетипизированные файлы можно использовать как для обработки файлов любого типа, так и для организации высокоскоростного обмена между дисками и памятью.

4.5. Процедуры и функции работы с файлами

AssignFile (var F; FileName : String); – связывает файловую переменную F и файл с именем FileName.

Reset (Var F[: File; RecSize : Word]); – открывает существующий файл. При открытии нетипизированного файла RecSize задает размер элемента файла.

ReWrite(var F[: File; RecSize: Word]); – создает и открывает новый файл.

Append (Var F : TextFile); – открывает текстовой файл для дописывания данных в конец файла.

Read (F, v1[,v2,…vn]); – чтение значений переменных, начиная с текущей позиции для типизированных файлов и строк для текстовых.

Write (F, v1[,v2,…vn]); – запись значений переменных, начиная с текущей позиции для типизированных файлов и строк для текстовых.

CloseFile (F); – закрывает ранее открытый файл.

ReName (Var F; NewName : String); – переименовывает закрытый файл любого типа.

Erase(var F); – удаляет закрытый файл любого типа.

Seek(var F; NumRec: Longint); – для нетекстового файла устанавливает указатель на элемент с номером NumRec.

Truncate (Var F); – удаляет записи из файла, начиная с текущей позиции (обрезает файл).

IoResult : Integer – код результата последней операции ввода/вывода.

FilePos (Var F) : LongInt – для нетекстовых файлов возвращает номер текущей позиции (отсчет ведется от нуля).

FileSize (Var F) : LongInt – для нетекстовых файлов возвращает количество компонентов в файле.

EOLn (Var F : TextFile) : Boolean – возвращает `True`, если достигнут конец строки.

EOF (Var F) : Boolean – возвращает `True`, если достигнут конец файла.

SeekEOLN (Var F : TextFile) : Boolean – возвращает `True`, если пройден последний значимый символ в строке или файле, отличный от пробела или знака табуляции.

SeekEOF (Var F: TextFile) : Boolean – то же, что и `SeekEOLN`, но для всего файла.

4.6. Компоненты TOpenDialog и TSaveDialog

Компоненты `TOpenDialog` и `TSaveDialog` находятся на странице `Dialogs`. Все компоненты этой страницы являются невизуальными, т.е. не видны в момент работы программы. Поэтому их можно разместить в любом удобном месте формы. Оба рассматриваемых компонента имеют идентичные свойства и отличаются только внешним видом. После вызова компонента появляется диалоговое окно, с помощью которого выбирается имя файла и путь к нему. В случае успешного завершения диалога имя выбранного файла и маршрут поиска содержатся в свойстве `FileName`. Для фильтрации файлов, отображаемых в окне просмотра, используется свойство `Filter`, а для задания расширения файла, в случае, если оно не задано пользователем, – свойство `DefaultExt`. Для изменения заголовка диалогового окна используется свойство `Title`.

4.7. Настройка компонентов TOpenDialog и TSaveDialog

Для установки компонентов `TOpenDialog` и `TSaveDialog` на форму необходимо на странице `Dialogs` меню компонентов щелкнуть мышью соответственно по пиктограммам  или  и расположить их в любом свободном месте формы. Для настройки фильтра следует выбрать соответствующий компонент и дважды щелкнуть по правой части свойства `Filter` инспектора объектов. Появится окно `Filter Editor`, в левой части которого записывается текст, характеризующий соответствующий фильтр, а в правой части – маска. Для `OpenDialog1` установим значения маски, как показано на рис. 4.1. Формат `*.dat` означает, что будут видны все файлы с расширением `dat`, а формат `*.*` – все файлы (с любым именем и с любым расширением).

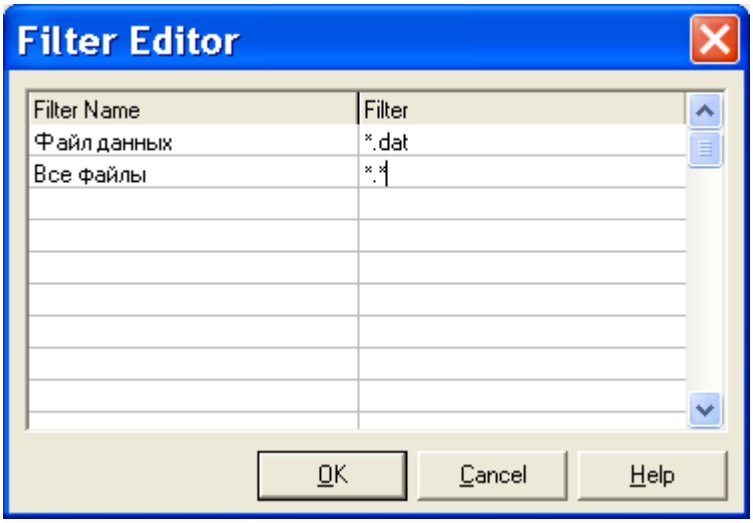


Рис. 4.1. Установка маски фильтра

Для того чтобы файл автоматически записывался, *например*, с расширением `dat`, в свойстве `DefaultExt` компонента `SaveDialog` необходимо указать маску `*.dat` (для текстового файла – `*.txt`).

4.8. Пример написания программы

Задание: составить и отладить программу, включающую режимы ***создания*** типизированного файла (каждая запись содержит фамилию студента и его оценки по физике, математике и химии), ***чтения*** ранее созданного файла, ***записи*** содержимого в текстовый файл и ***вывода*** списка студентов, не имеющих четверок.

Ниже на рисунках приведены результаты выполнения программы после нажатия кнопок «Читать» (рис. 4.2) и «Вывести» (рис. 4.3).

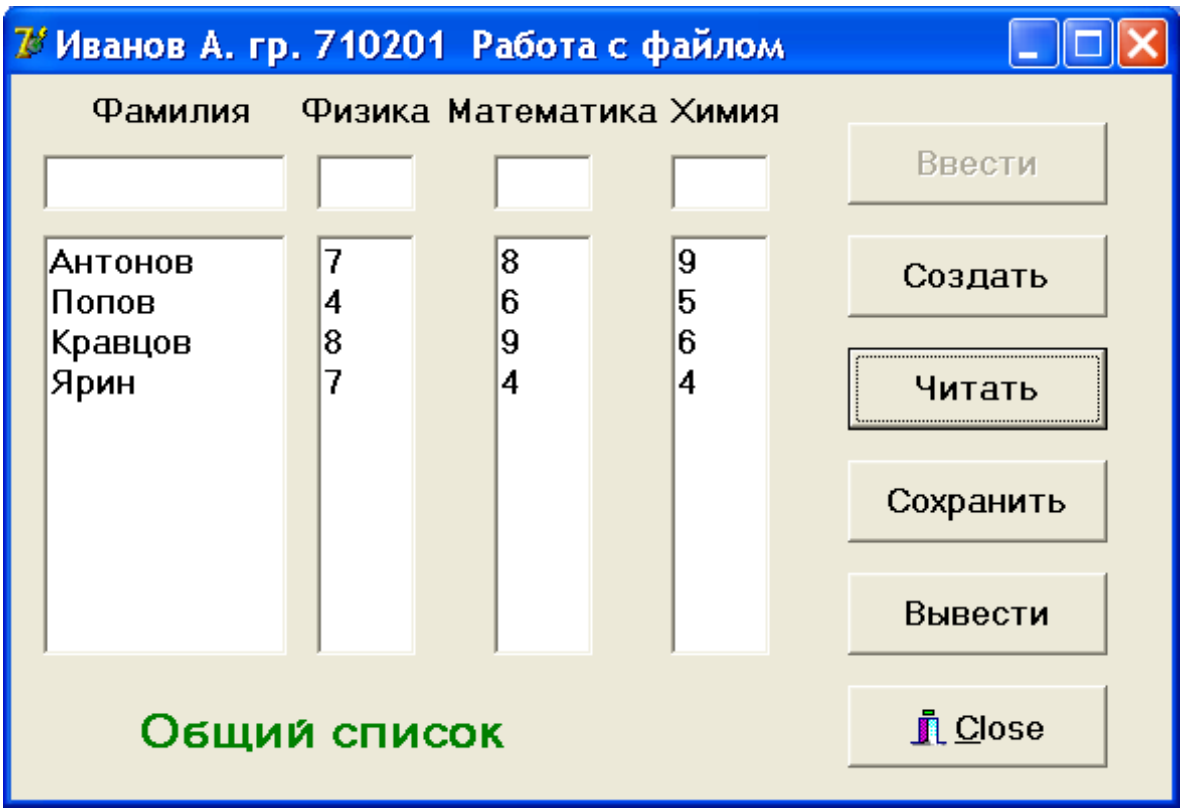


Рис. 4.2. Результат выполнения программы после нажатия кнопки «Читать»

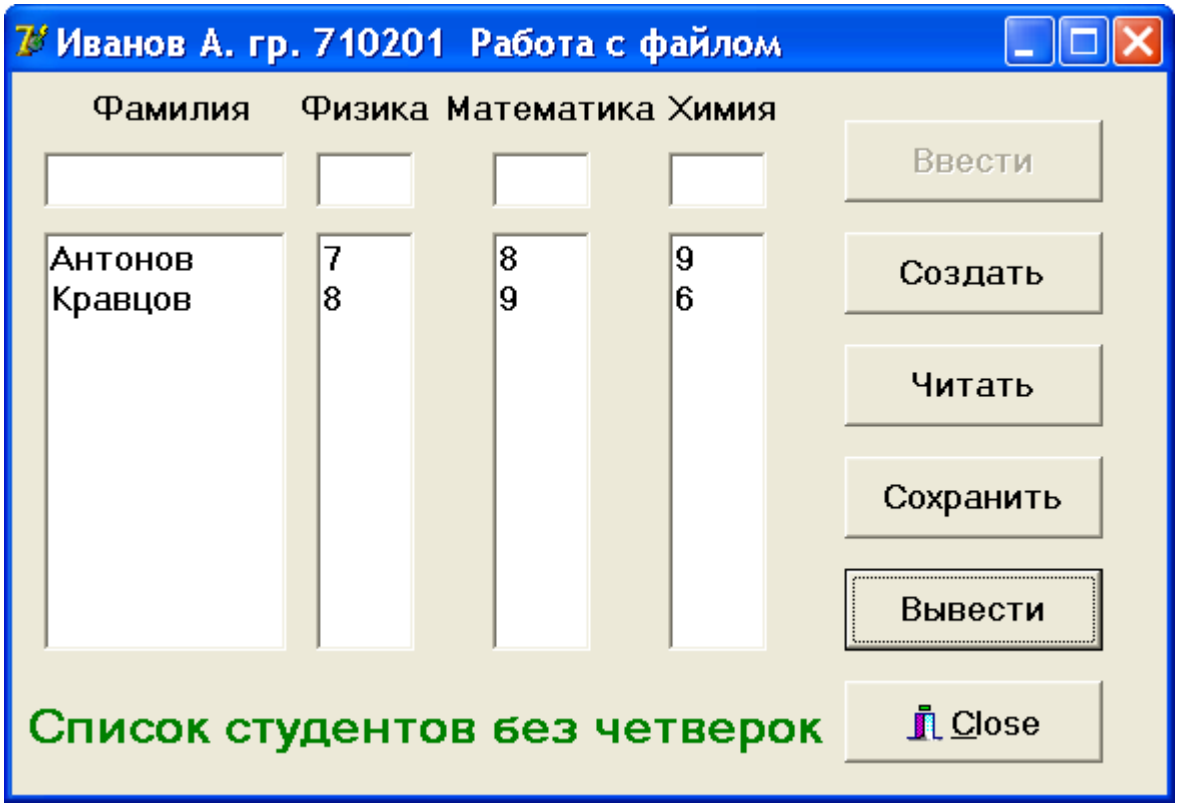


Рис. 4.3. Результат выполнения программы после нажатия кнопки «Вывести»

Код программы имеет вид:

```
unit Unit1;
interface
uses  Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms,Dialogs, StdCtrls, Buttons;
type
  TForm1 = class(TForm)
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    BitBtn1: TBitBtn;
    Memo1: TMemo;
    Memo2: TMemo;
    Memo3: TMemo;
    Memo4: TMemo;
    Label5: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

Type Stud=Record
       fam:String[20];
       oc:Array[1..3] of Byte;
```

```
     End;
Var
  Form1: TForm1;
  f:File of Stud;
  ft:TextFile;
  w:Stud;
  fname,fnamet:String;
  zak:Boolean;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
Begin
 Memo1.Clear; Memo2.Clear; Memo3.Clear; Memo4.Clear;
 Edit1.Clear; Edit2.Clear; Edit3.Clear; Edit4.Clear;
 Button1.Enabled:=False;
 Label5.Caption:= '';
 zak:=False;
End;

procedure TForm1.Button1Click(Sender: TObject);
Begin                    //ввод данных из формы и запись их в файл f
   w.fam:=Edit1.Text;
   w.oc[1]:=StrToInt(Edit2.Text);
   w.oc[2]:=StrToInt(Edit3.Text);
   w.oc[3]:=StrToInt(Edit4.Text);
   Write(f,w);
   Memo1.Lines.Add(w.fam);
   Memo2.Lines.Add(IntToStr(w.oc[1]));
   Memo3.Lines.Add(IntToStr(w.oc[2]));
   Memo4.Lines.Add(IntToStr(w.oc[3]));
   Edit1.Clear; Edit2.Clear; Edit3.Clear; Edit4.Clear;
End;

procedure TForm1.Button2Click(Sender: TObject);
Begin                    //создание типизированного файла f
  SaveDialog1.Title:='Создать файл';
  SaveDialog1.DefaultExt:='.dat';
  if SaveDialog1.Execute Then
      Begin
        fname:=SaveDialog1.FileName;
        AssignFile(f,fname);
```

```
        ReWrite(f);
      End;
  Button1.Enabled:=True;
  zak:=True;
  Label5.Caption:='        Общий список';
End;

procedure TForm1.Button3Click(Sender: TObject);
Begin                    //чтение данных из файла f и вывод их в Memo
  OpenDialog1.Title:='Открыть файл';
  if OpenDialog1.Execute Then
    Begin
      fname:=OpenDialog1.FileName;
      AssignFile(f,fname);
      Reset(f);
    End;
  Memo1.Clear; Memo2.Clear; Memo3.Clear; Memo4.Clear;
  While Not EOF(f) Do
     Begin
       Read(f,w);
       Memo1.Lines.Add(w.fam);
       Memo2.Lines.Add(IntToStr(w.oc[1]));
       Memo3.Lines.Add(IntToStr(w.oc[2]));
       Memo4.Lines.Add(IntToStr(w.oc[3]));
     End;
  CloseFile(f);
  Label5.Caption:='        Общий список';
End;

procedure TForm1.Button4Click(Sender: TObject);
Begin            //запись результатов в текстовый файл ft
  SaveDialog1.Title:='Сохранить в текстовом файле';
  SaveDialog1.DefaultExt:='.txt';
  if SaveDialog1.Execute Then
    Begin
      fnamet:=SaveDialog1.FileName;
      AssignFile(ft,fnamet);
      ReWrite(ft);
    End;
  WriteLn(ft,'        Общий список студентов');
  WriteLn(ft,'   Фамилия  Физика  Математика  Химия ');
  Reset(f);
  While Not EOF(f) Do
   Begin
```

```
      Read(f,w);
      With w Do
         WriteLn(ft,fam:10,oc[1]:6,oc[2]:10,oc[3]:9);
    End;
  CloseFile(f);
  CloseFile(ft);
End;

procedure TForm1.Button5Click(Sender: TObject);
 Var k,m:Integer;
Begin        //вывод в Memo сведений о студентах, не имеющих четверок
  Memo1.Clear; Memo2.Clear; Memo3.Clear; Memo4.Clear;
  Reset(f);
  While Not EOF(f) Do
   Begin
     Read(f,w);
     m:=0;
     For k:=1 To 3 Do
        if w.oc[k]=4 Then m:=1;
     if m=0 Then
       Begin
          Memo1.Lines.Add(w.fam);
          Memo2.Lines.Add(IntToStr(w.oc[1]));
          Memo3.Lines.Add(IntToStr(w.oc[2]));
          Memo4.Lines.Add(IntToStr(w.oc[3]));
       End;
  End;
 CloseFile(f);
 Label5.Caption:='Список студентов без четверок';
End;

procedure TForm1.BitBtn1Click(Sender: TObject);
Begin
  if zak Then CloseFile(f);
End;

End.
```

4.9. Индивидуальные задания

По указанию преподавателя выберите вариант задания. Предусмотрите режимы: создания типизированного файла, чтения ранее созданного файла, записи содержимого в текстовый файл и вывода записей текстового файла в компонент TMemo.

1. В магазине формируется список лиц, записавшихся на покупку товара. Каждая запись этого списка содержит фамилию, домашний адрес покупателя и дату постановки его на учет. Удалить из списка те повторяющиеся записи, у которых совпадают фамилия и домашний адрес покупателя.

2. Список товаров, имеющихся на складе, включает наименование товара, его количество и дату поступления товара на склад. Вывести в алфавитном порядке список товаров, хранящихся больше месяца.

3. Для получения места в общежитии формируется список студентов, содержащий фамилию студента, его средний балл и доход на члена семьи. Известно, что общежитие в первую очередь предоставляется тем, у кого доход на члена семьи меньше двух минимальных зарплат, а затем остальным студентам в порядке уменьшения их среднего балла. Вывести список очередности предоставления мест в общежитии.

4. В справочной автовокзала хранится расписание движения автобусов. Для каждого рейса указаны пункт назначения, время отправления и прибытия автобуса. Вывести информацию о рейсах, которыми можно воспользоваться для прибытия в пункт назначения раньше заданного времени.

5. Информация о сотрудниках фирмы включает фамилию, количество проработанных часов за месяц и почасовой тариф. Рабочее время свыше 144 часов считается сверхурочным и оплачивается в двойном размере. Вывести размер заработной платы каждого сотрудника фирмы, учитывая подоходный налог, который составляет 12 % от суммы заработка.

6. Информация об участниках спортивных соревнований содержит название команды, фамилию игрока и его возраст. Вывести информацию о самой молодой команде.

7. Для книг, хранящихся в библиотеке, указывается автор, название книги и год ее издания. Вывести отсортированный по фамилиям авторов список книг, изданных после заданного года.

8. Различные цехи завода выпускают продукцию нескольких наименований. Сведения о выпущенной продукции состоят из наименования продукции, ее количества и номера цеха, в котором она изготовлена. Для заданного цеха необходимо вывести по каждому наименованию продукцию в порядке убывания ее количества.

9. Информация о сотрудниках предприятия содержит фамилию, номер отдела и дату начала работы. Вывести списки сотрудников по отделам в порядке убывания их стажа.

10. Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит фамилию, адрес и три оценки. Определить количество абитуриентов, проживающих в г. Минске и сдавших вступительные экзамены со средним баллом не ниже 8. Вывести их фамилии в алфавитном порядке.

11. В справочной аэропорта хранится расписание вылета самолетов на следующие сутки. Для каждого рейса указаны номер рейса, пункт назначения и время вылета самолета. Вывести для заданного пункта назначения все номера рейсов и время вылета самолетов в порядке возрастания времени вылета.

12. У администратора железнодорожных касс хранится информация о свободных местах в поездах дальнего следования в следующем виде: пункт назначения, время отправления поезда и количество свободных мест в нем. Оргкомитет международной конференции обращается к администратору с просьбой зарезервировать *заданное* количество мест до *заданного* города со временем отправления поезда не позднее *заданного*. Вывести время отправления или сообщение о невозможности выполнения заказа в полном объеме.

13. Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит фамилию абитуриента и три оценки. Необходимо определить средний балл по университету и вывести список абитуриентов, сдавших вступительные экзамены выше среднего балла по университету. Первыми в списке должны идти студенты, получившие на экзаменах оценки не ниже 9.

14. В радиоателье хранятся квитанции о сданной в ремонт радиоаппаратуре. Каждая квитанция содержит наименование изделия (телевизор, радиоприемник и т. п.), дату приемки его в ремонт и состояние готовности заказа (выполнен, не выполнен). Вывести информацию о состоянии заказов по группам изделий.

15. На междугородной АТС информация о разговорах содержит: название города, время разговора, тариф и номер телефона абонента. Вывести по каждому городу общее время разговоров с ним и сумму.

Контрольные вопросы и задания

1. Дайте определение файла и файловой переменной.
2. Как описывают файловую переменную, текстовый файл?
3. Какими стандартными процедурами для работы с файлами располагает Pascal ?
4. Перечислите особенности текстовых файлов.

ТЕМА 5. ПРОГРАММИРОВАНИЕ С ОТОБРАЖЕНИЕМ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ

Цель работы: изучить возможности построения изображений с использованием класса `TCanvas` и графиков с помощью компонента `TChart`.

5.1. Создание изображений

Экран дисплея устроен таким образом, что любое изображение формируется из набора светящихся точек, получивших название **пиксел**. Нарисовать картинку в среде Delphi можно на многих компонентах (*например*, на форме, на `TPaintBox`). Однако наиболее удобно использовать компонент `TImage` (страница `Additional`). Для рисования используют класс `TCanvas`, который является свойством многих компонентов и представляет собой прямоугольный холст в виде матрицы из пикселей и набор инструментов для рисования на нем. Каждый пиксел имеет координату (x,y), где x – порядковый номер пиксела, начиная от левой границы холста, а y – порядковый номер пиксела, начиная от

верхней границы холста. Левый верхний угол холста имеет координату (0,0), а нижний правый (Image1.Width-1,Image1.Height-1).

Основные ***свойства*** класса `TCanvas`:

Property Pen : TPen – карандаш; имеет свойства: `Color` – цвет, `Width` – толщина, `Style` – стиль (`psSolid` – сплошной, `psDash` – штриховой, `psDot` - пунктирный, `psClear` – отсутствие линии и др.).

Property Brush : TBrush – кисть; имеет свойства: `Color` – цвет, `Style` – стиль (`bsSolid` – сплошной, `bsCross` – сетка, `bsClear` – отсутствие фона и др.). Данное свойство определяет фон заполнения замкнутых фигур.

Property Font : TFont – шрифт; имеет свойства: `Color` – цвет, `Size` – размер, `Style` – стиль (`fsBold` – жирный, `fsItalic` – курсив и др.).

Некоторые ***методы*** класса `TCanvas`:

Ellipse (X1, Y1, X2, Y2: Integer); – рисует эллипс в охватывающем прямоугольнике (`X1,Y1`), (`X2,Y2`) и заполняет внутреннее пространство эллипса текущей кистью.

LineTo (X, Y: Integer); – рисует линию от текущего положения пера до точки (`X,Y`).

MoveTo (X, Y: Integer); – перемещает карандаш в точку (`X,Y`) без вычерчивания линий.

Polygon (Points: Array of TPoint); – рисует многоугольник по точкам, заданным в массиве `Points`.

Например: `Canvas.Polygon([Point(x1,y1),Point(x2,y2), Point(x3,y3)]);`. Конечная точка соединяется с начальной, и многоугольник заполняется кистью. Для вычерчивания без заполнения используется метод `PolyLine`.

Rectangle (X1, У1, X2, У2: Integer); – рисует и заполняет прямоугольник (`X1,У1`), (`X2,У2`). Для вычерчивания без заполнения используется `FrameRect` или `PolyLine`.

TextOut (X, У : Integer; Const Text : String); – выводит текстовую строку `Text` так, чтобы левый верхний угол прямоугольника, охватывающего текст, располагался в точке (`X,У`).

5.2. Построение графиков с помощью компонента TChart

Обычно результаты расчетов представляются в виде графиков и диаграмм. Среда Delphi имеет мощный пакет стандартных программ вывода на экран и редактирования графической информации, который реализуется с помощью визуально отображаемого на форме компонента `TChart`. Построение графика (диаграммы) производится после вычисления таблицы значений функции `y=f(x)`. Полученная таблица передается с помощью метода `AddXY` в специальный двумерный массив `Chart`**`I`**`.SeriesList`**[k],** где **k** – номер графика (`0,1,2,...`). Компонент `TChart` осуществляет всю работу по отображению

графиков, переданных в объект Chart**I**.SeriesList[**k**]: отображает переданную таблицу в виде всевозможных графиков или диаграмм, рисует координатную сетку, строит и размечает оси, подписывает название графика и его осей. При необходимости с помощью встроенного редактора EditingChart компоненту TChart передаются данные о толщине, стиле и цвете линий, параметрах шрифта подписей, шагах разметки координатной сетки и другие настройки. В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам компонента TChart. Так, *например*, свойство Chart1.BottomAxis содержит значение максимального предела нижней оси графика. Перенести график в отчет можно через буфер обмена, используя процедуру Chart**I**.CopyToClipBoardMetaFile(True). Для изменения параметров компонента TChart необходимо дважды щелкнуть по нему мышью в окне формы. Появится окно редактирования EditingChat1 (рис. 5.1). Для создания нового объекта Series1 щелкнуть по кнопке Add на странице Series. В появившемся диалоговом окне TeeChart Gallery выбрать пиктограмму с надписью Line (график выводится в виде линий). Если не нужно отображать график в трехмерном виде, то надо отключить независимый переключатель 3D. После нажатия на кнопку OK появится новая серия с названием Series1. Для изменения названия графика следует нажать кнопку Title. Закладка Legend задает список обозначений диаграммы (ее можно убирать с экрана). Название графика вводится на странице Titles. Разметка осей меняется на странице Axis. Страница Series задает характеристики (цвет, толщина линий) для определенного графика. Нажимая различные кнопки меню, познакомьтесь с другими возможностями EditingChat.

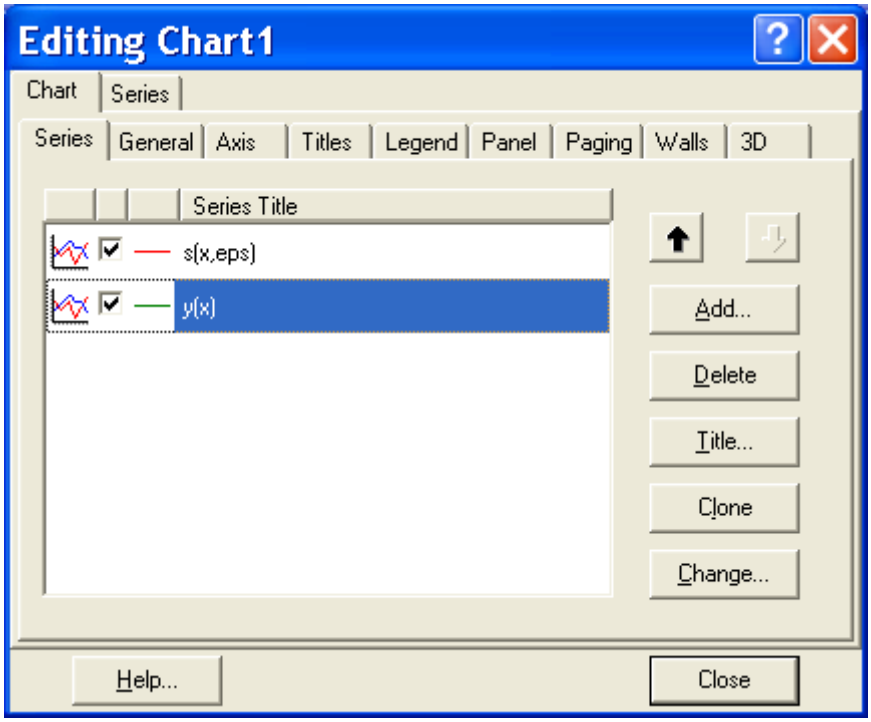


Рис. 5.1. Окно редактирования EditingChat1

5.3. Пример написания программы

Задание 1: построить графики функций `sin(x)` и `cos(x)` с помощью компонента `TChart`.

Результат выполнения программы приведен на рис. 5.2.

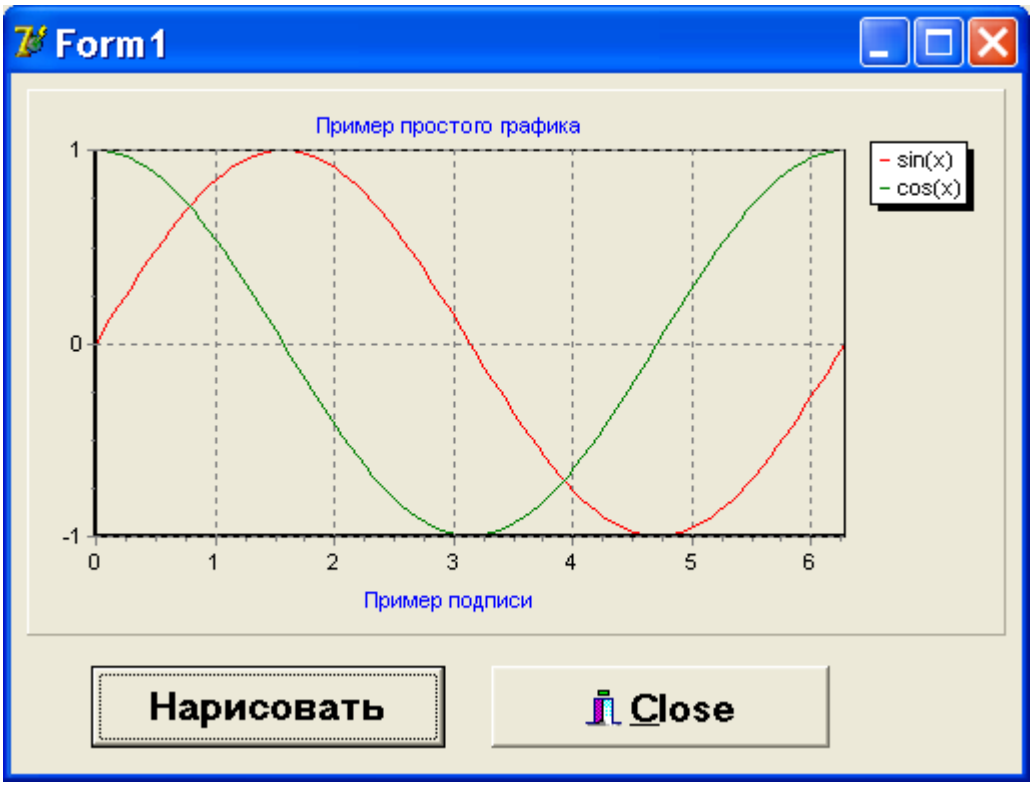


Рис. 5.2. Результат выполнения задания 1

Код программы имеет вид:

```
unit Unit1;
interface
uses Windows,Messages,SysUtils,Variants,Classes,
     Graphics, Controls,Forms,Dialogs,StdCtrls,TeEngine,
     Series,ExtCtrls,TeeProcs,Chart,Buttons;
type
  TForm1 = class(TForm)
    Chart1: TChart;
    Series1: TLineSeries;
    Series2: TLineSeries;
    Button1: TButton;
    BitBtn1: TBitBtn;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;
  f:File of Extended;
implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
  Var k:Integer;          //построение графиков функций sin(x) и cos(x)
      x,y,z:Extended;
Begin
 For k:=0 To 100 Do
  Begin
   x:=0.02*Pi*k;  y:=Sin(x);  z:=Cos(x);
   Series1.AddXY(x,y,'',clRed);
   Series2.AddXY(x,z,'',clGreen);
  End;
End;
End.
```

Задание 2: вывести на форму простейшие фигуры и текст.
Результат выполнения программы приведен на рис. 5.3.

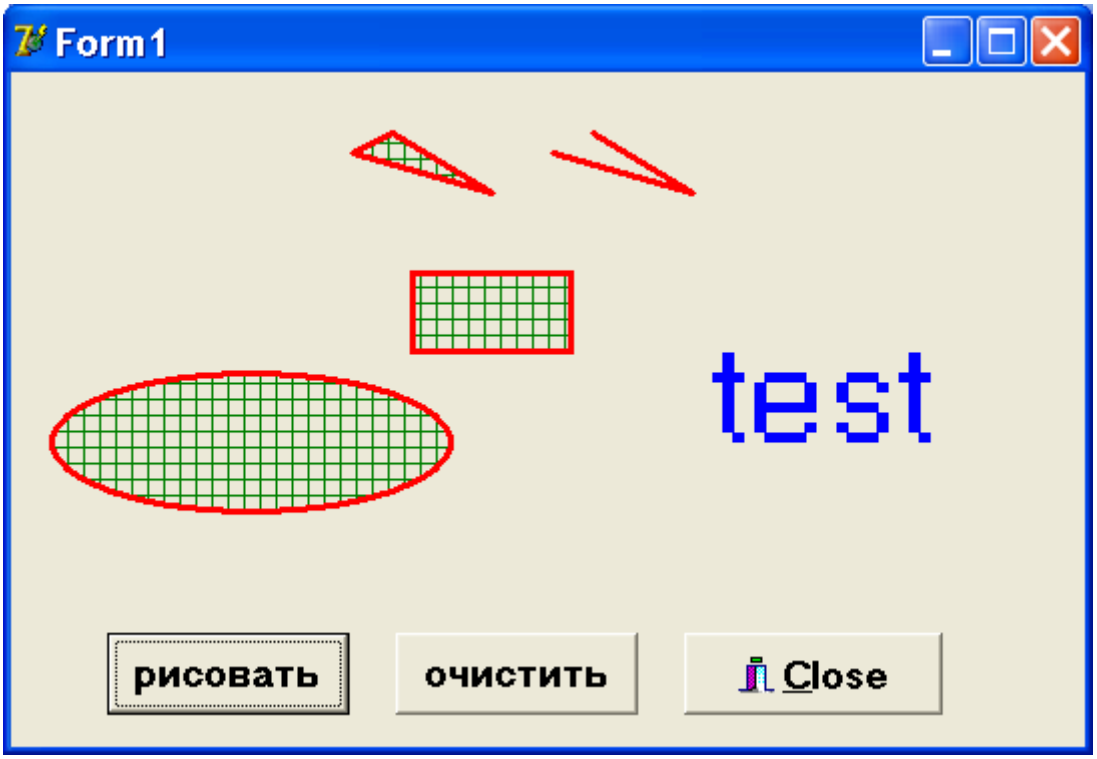


Рис. 5.3. Результат выполнения задания 2

Код программы имеет вид:

```
unit Unit1;
```

```
interface
uses
     Windows,Messages,SysUtils,Variants,Classes,Graphics,
     Controls,Forms,Dialogs,StdCtrls,Buttons;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    BitBtn1: TBitBtn;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
procedure TForm1.Button1Click(Sender:TObject);
Begin
 With Form1.Canvas Do
  Begin
   Pen.Width:=3;
   Pen.Color:=clRed;
   Brush.Style:=bsCross;
   Brush.Color:=clGreen;
   Polygon([Point(190,30),Point(240,60),point(170,40)]);
   Polyline([point(290,30),point(340,60),point(270,40)]);
   Rectangle(200,100,280,140);
   Ellipse(20,150,220,220);
   Font.Color:=clBlue;
   Font.Size:=50;
   TextOut(350,120,'test');
  End;
End;

procedure TForm1.Button2Click(Sender:TObject);
Begin           //очистка
    Refresh;
```

```
End;

End.
```

5.4. Индивидуальные задания

Постройте графики двух функций f(x) по вашему выбору. Получите таблицу данных для указанного интервала и шага таблицы. Ввод исходных данных организуйте из компонента Edit. Самостоятельно выберите удобные параметры настройки.

Используя функции класса TCanvas, нарисуйте геометрические фигуры, соответствующие указанному преподавателем варианту задания.

1. Даны три числа a,b,c. Необходимо определить, существует ли треугольник с такими длинами сторон.

2. Даны четыре числа a,b,c,d. Необходимо определить, существует ли четырехугольник с такими длинами сторон.

3. Отобразить взаимное расположение двух окружностей с радиусами R_1 и R_2 с центрами в точках (x_1, y_1), (x_2, y_2) соответственно.

4. Отобразить взаимное расположение окружности с радиусом R с центром в точке (x_0, y_0) и прямой, проходящей через точки с координатами (x_1, y_1) и (x_2, y_2) (пересекаются, касаются, не пересекаются).

5. Определить количество точек с целочисленными координатами, лежащих внутри окружности радиусом R с центром в точке (x_0, y_0).

6. Найти координаты точек пересечения двух окружностей с радиусами R_1 и R_2 с центрами в точках (x_1, y_1) и (x_2, y_2) соответственно.

7. Найти координаты точки, симметричной данной точке M, с координатами (x_1, y_1) относительно прямой Ax+By+C=0.

8. Даны две точки $M_1(x_1, y_1)$, $M_2(x_2, y_2)$ и прямая Ax+By+C=0. Необходимо найти на этой прямой такую точку $M_0(x_0, y_0)$, чтобы суммарное расстояние от нее до двух данных точек было минимально.

9. Даны три точки с координатами (x_1, y_1), (x_2, y_2), (x_3, y_3), которые являются вершинами некоторого прямоугольника со сторонами, параллельными осям координат. Найти координаты четвертой точки.

10. Даны координаты четырех точек (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4). Необходимо определить, образуют ли они выпуклый четырехугольник.

11. Даны координаты четырех точек (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4). Необходимо определить, какую из фигур они образуют: ромб, квадрат или трапецию.

12. Даны координаты двух вершин (x_1, y_1) и (x_2, y_2) некоторого квадрата. Необходимо найти возможные координаты других его вершин.

13. Даны координаты двух вершин (x_1, y_1) и (x_2, y_2) некоторого квадрата, которые расположены на диагонали, и точка (x_3, y_3). Необходимо определить, находится ли точка внутри квадрата.

14. Даны координаты трех вершин $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ треугольника. Необходимо найти координаты точки пересечения его медиан.

15. Даны координаты трех вершин $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ треугольника. Необходимо найти длины его высот.

Контрольные вопросы и задания

1. Какие компоненты применяются для создания изображений в Delphi?
2. Какой класс используется для рисования? Что он собой представляет?
3. Назовите свойства класса `TCanvas`.
4. Перечислите методы класса `TCanvas`.
5. Объясните процесс построения графиков (диаграмм) с помощью компонента `TChart`.

ЛИТЕРАТУРА

1. Архангельский, А. Я. Программирование в Delphi 7 /А. Я. Архангельский. – М. : ЗАО «Издательство БИНОМ», 2003.
2. Фаронов, В. В. Delphi 6: учебный курс /В. В. Фаронов. – М. : Издатель Молгачева С. В., 2001.
3. Брукшир, Дж. Г. Введение в компьютерные науки / Дж. Г. Брукшир. – СПб, Киев : Вильямс, 2001.
4. Программирование алгоритмов в среде Delphi : лаб. практикум. В 2 ч. Ч. 1 / А. К. Синицын [и др.]. – Минск : БГУИР, 2004.
5. Колосов, С. В. Программирование в среде Delphi : учеб. пособие / С. В. Колосов. – Минск : БГУИР, 2005.
6. Калиткин, Н. Н. Численные методы / Н. Н. Калиткин. – М. : Наука, 1978.
7. Бахвалов, Н. С. Численные методы / Н. С. Бахвалов. – М. : Наука, 1975.
8. Численные методы анализа / В. П. Демидович [и др.]. – М. : Физматгиз, 1963.
9. Волков, Е. А. Численные методы / Е. А. Волков. – М. : Наука, 1982.
10. Вычислительные методы высшей математики. Т.1 / В. И. Крылов [и др.]. – Минск : Выш. шк., 1972.
11. Крылов, В. И. Вычислительные методы высшей математики. Т.2 / В. И. Крылов [и др.]. – Минск : Выш. шк., 1975.
12. Форсайт, Дж. Машинные методы математических вычислений / Дж. Форсайт [и др.]. – М. : Мир, 1980.
13. Шуп, Т. Решение инженерных задач на ЭВМ / Т. Шуп. – М. : Мир, 1982.
14. Самарский, А. А. Введение в численные методы / А. А. Самарский. – М. : Наука, 1982.
15. Березин, И. С. Методы вычислений. Т.2 / И. С. Березин, Н. П. Жидков. – М. : Физматгиз, 1970.
16. Банди, Б. Методы оптимизации. Вводный курс / Б. Банди. – М. : Мир, 1989.

ПРИЛОЖЕНИЕ 1

Процедуры и функции преобразования строкового представления чисел

Для работы со строками применяются следующие процедуры и функции (в квадратных скобках указываются необязательные параметры).

Подпрограммы преобразования строк в другие типы	
Function **StrToFloat** (St: String) : Extended	Преобразует символы строки `St` в вещественное число; строка не должна содержать ведущих или ведомых пробелов
Function **StrToInt** (St: String) : Integer	Преобразует символы строки `St` в целое число; строка не должна содержать ведущих или ведомых пробелов
Procedure **Val** (St : String; Var X; Code : Integer);	Преобразует строку символов `St` во внутреннее представление целой или вещественной переменной `X`, которое определяется типом этой переменной; параметр `Code` равен нулю в случае успешного преобразования
Подпрограммы обратного преобразования	
Function **FloatToStr** (Value: Extended) : String	Преобразует вещественное значение `Value` в строку символов
Function **FloatToStrF** (Value : Extended; Format: TFloatFormat; Precision, Digits : Integer) : String	Преобразует вещественное значение `Value` в строку символов с учетом параметров `Precision` и `Digits` (см. пояснения ниже)
Procedure **Str** (X [:width [: Decimals]]; Var St : String);	Преобразует число `X` любого вещественного или целого типа в строку символов `St`; если присутствуют параметры `Width` и `Decimals`, то они задают формат преобразования

Правила использования параметров функции FloatToStrF	
Значение Format	**Описание**
`ffExponent`	Научная форма представления с множителем `eXX.Precision` задает общее количество десятичных цифр мантиссы; `Digits` – количество цифр в десятичном порядке `XX`
`ffFixed`	Формат с фиксированным положением разделителя целой и дробной частей; `Precision` задает общее количество десятичных цифр в представлении числа, `Digits` – количество цифр в дробной части. Число округляется с учетом первой отбрасываемой цифры: `3,14`

ffGeneral	Универсальный формат, использующий наиболее удобную для чтения форму представления вещественного числа; соответствует формату ffFixed, если количество цифр в целой части меньше или равно Precision и само число - больше или равно 10^{-5}, в противном случае соответствует формату ffExponent: 3,1416
ffNumber	Отличается от ffFixed использованием символа разделителя тысяч при выводе больших чисел (для русифицированной версии Windows таким разделителем является пробел)
ffCurrency	Денежный формат, соответствующий ffNumber, но в конце строки ставится символ денежной единицы (для русифицированной версии Windows-символы «р.»). Для Value=π*1000 получим: 3 141,60 р.

ПРИЛОЖЕНИЕ 2

Таблицы символов ASCII

Стандартная часть таблицы символов ASCII

КС	С	КС	С	КС	С	КС	С	КС	С	КС	С	КС	С	КС	С
0		16	►	32		48	0	64	@	80	P	96	`	112	p
1	☺	17	◄	33	!	49	1	65	A	81	Q	97	a	113	q
2	☻	18	↕	34	"	50	2	66	B	82	R	98	b	114	r
3	♥	19	‼	35	#	51	3	67	C	83	S	99	c	115	s
4	♦	20	¶	36	$	52	4	68	D	84	T	100	d	116	t
5	♣	21	§	37	%	53	5	69	E	85	U	101	e	117	u
6	♠	22	▬	38	&	54	6	70	F	86	V	102	f	118	v
7	•	23	↨	39	'	55	7	71	G	87	W	103	g	119	w
8	◘	24	↑	40	(	56	8	72	H	88	X	104	h	120	x
9	○	25	↓	41	)	57	9	73	I	89	Y	105	i	121	y
10	◙	26	→	42	*	58	:	74	J	90	Z	106	j	122	z
11	♂	27	←	43	+	59	;	75	K	91	[	107	k	123	{
12	♀	28	∟	44	,	60	<	76	L	92	\	108	l	124	|
13	♪	29	↔	45	-	61	=	77	M	93	]	109	m	125	}
14	♫	30	▲	46	.	62	>	78	N	94	^	110	n	126	~
15	☼	31	▼	47	/	63	?	79	O	95	_	111	o	127	⌂

Некоторые из вышеперечисленных символов имеют особый смысл. Так, *например*, символ с кодом 9 обозначает символ горизонтальной табуляции, символ с кодом 10 – символ перевода строки, символ с кодом 13 – символ возврата каретки.

Дополнительная часть таблицы символов

КС	С	КС	С	КС	С	КС	С	КС	С	КС	С	КС	С	КС	С
128	А	144	Р	160	а	176	▒	192	└	208	╨	224	р	240	Ё
129	Б	145	С	161	б	177	▒	193	┴	209	╤	225	с	241	ё
130	В	146	Т	162	в	178	▓	194	┬	210	╥	226	т	242	Є
131	Г	147	У	163	г	179	│	195	├	211	╙	227	у	243	є
132	Д	148	Ф	164	д	180	┤	196	─	212	╘	228	ф	244	Ї
133	Е	149	Х	165	е	181	╡	197	┼	213	╒	229	х	245	ї
134	Ж	150	Ц	166	ж	182	╢	198	╞	214	╓	230	ц	246	Ў
135	З	151	Ч	167	з	183	╖	199	╟	215	╫	231	ч	247	ў
136	И	152	Ш	168	и	184	╕	200	╚	216	╪	232	ш	248	°
137	Й	153	Щ	169	й	185	╣	201	╔	217	┘	233	щ	249	·
138	К	154	Ъ	170	к	186	║	202	╩	218	┌	234	ъ	250	·
139	Л	155	Ы	171	л	187	╗	203	╦	219	█	235	ы	251	√
140	М	156	Ь	172	м	188	╝	204	╠	220	▄	236	ь	252	№
141	Н	157	Э	173	н	189	╜	205	═	221	▌	237	э	253	¤
142	О	158	Ю	174	о	190	╛	206	╬	222	▐	238	ю	254	■
143	П	159	Я	175	п	191	┐	207	╧	223	▀	239	я	255	

Примечание. В таблицах обозначение **КС** означает «код символа», а **С** – «символ».

Св. план 2008, поз. 47

Учебное издание

Аксенчик Анатолий Владимирович
Коренская Ирина Николаевна
Навроцкий Анатолий Александрович
Шестакович Вячеслав Павлович

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ OBJECT PASCAL В СРЕДЕ DELPHI

Лабораторный практикум по курсу
«Основы алгоритмизации и программирования»
для студентов всех специальностей заочной формы обучения

В 2-х частях
Часть 2

Редактор М. В. Тезина
Корректор Е. Н. Батурчик

Подписано в печать 15.12.2008. Формат 60×84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 3,14.
Уч. изд. л. 2,7. Тираж 300 экз. Заказ 189.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6