7 Обработка файлов средствами Free Pascal

В данной главе будут рассмотрены возможности языка FreePascal для работы с файлами. По ходу изложения материала читатель также познакомится с компонентами Lazarus, предназначенными для выбора файла.

Начнем со знакомства с типами файлов.

7.1 Типы файлов

Ввод данных с клавиатуры удобен при обработке небольших объемов информации. В случае обработки массивов, состоящих из сотен элементов, использовать ввод данных с клавиатуры нерационально. В подобных случаях данные удобно хранить в файлах, программа будет их считывать, обрабатывать, выводить результаты на экран или в файл. Рассмотрим, как это можно сделать.

С точки зрения программиста, все файлы можно разделить на три класса:

- типизированные;
- бестиповые;
- текстовые.

Файлы, состоящие из компонентов одного типа (целые, вещественные, массивы и т.д.), число которых заранее не определено и может быть любым, называются *типизированными*. Они заканчиваются специальным символом «*конец файла*», хранятся в двоичном виде, содержимое подобных файлов нельзя просмотреть обычным текстовым редактором, для просмотра подобных файлов нужно писать специальную программу.

В *бестиповых файлах* информация считывается и записывается блоками определенного размера. В подобных файлах хранятся данные любого вида и структуры.

Текстовые файлы состоят из любых символов. При записи информации в текстовый файл все данные преобразуются к символьному типу, в котором и хранятся. Просмотреть данные в подобном файле можно с помощью любого текстового редактора. Информация в текстовом файле хранится построчно. В конце каждой строки хранится специальный символ «*конец строки*». Конец самого файла обозначается символом «*конец файла*».

Для работы с файлами в программе следует описать файловую переменную. Для работы с *текстовым* файлом файловая переменная (например, *f*) описывается с помощью служебного слова *text*.

```
var f:text;
```

Для описания *типизированных*[75] файлов можно описать файловую переменную следующим образом:

```
var f:file of тип;
```
[76]

Бестиповый файл описывается с помощью служебного слова `file`.

Рассмотрим несколько примеров описания файловых переменных.

```
type
massiv=array[1..25]of real;
ff=file of real;
var
a:text; {Файловая переменная a для работы с текстовым файлом}
b:ff; {Файловая переменная f для работы с файлом вещественных чисел}
c:file of integer; {Файловая переменная c для работы с файлом целых чисел}
d:file of massiv; {Файловая переменная d предназначена для работы с типизированным файлом, элементами которого являются массивы из 25 вещественных чисел. }
```

Рассмотрим последовательно работу с файлами каждого типа.

7.2 Работа с типизированными файлами

Знакомство с методами обработки типизированных файлов начнем с подпрограмм, которые являются общими для всех типов файлов.

7.2.1 Процедура *AssignFile*

Для начала работы с файлом необходимо связать файловую переменную в программе с файлом на диске. Для этого используется про-

75 Типизированные файлы иногда называют компонентными.

76 Здесь **f** — имя файловой переменной, **тип** — тип компонентов файла, это могут быть как стандартные типы данных (например, real, integer и т.д.), так и типы, определенные пользователем.

цедура `AssignFile(f,s)`, где `f` – имя файловой переменной, а `s` – полное имя файла на диске (файл должен находиться в текущем каталоге при условии, что к нему специально не указывается путь).

Рассмотрим примеры использования `AssignFile` для различных операционных систем.

```
var
f:file of real;
begin
//Пример процедуры assign для ОС Windows.
AssignFile (f, 'd:\tp\tmp\abc.dat');
//Пример процедуры assign для ОС Linux.
AssignFile(f,'/home/pascal/6/pr1/abc.dat');
```

7.2.2 Процедуры reset, rewrite

После установления связи между файловой переменной и именем файла на диске нужно открыть файл, воспользовавшись процедурами `reset` или `rewrite`.

Процедура `reset(f)` (где `f` – имя файловой переменной) открывает файл, связанный с файловой переменной `f`, после чего становится доступным для чтения первый элемент, хранящийся в файле. Далее можно выполнять чтение и запись данных из файла.

Процедура `rewrite(f)` создает пустой файл (месторасположение файла на диске определяется процедурой `AssignFile`) для последующей записи в него данных.

Внимание!!! Если файл, связанный с файловой переменной *f*, существовал на диске, то вся информация в нем уничтожается.

7.2.3 Процедура CloseFile

Процедура `CloseFile(f)`, где `f` – имя файловой переменной, закрывает файл, который ранее был открыт процедурами `rewrite`, `reset`.

Процедуру `CloseFile(f)` следует *обязательно* использовать при закрытии файла, в который происходила запись данных.

Дело в том, что процедуры записи в файл не обращаются непосредственно к диску, они пишут информацию в специальный участок памяти, называемый буфером файла. После того как буфер заполнится, вся информация из него переносится в файл. При выполнении

процедуры `closefile` сначала происходит запись буфера файла на диск, и только потом файл закрывается. Если его не закрыть вручную, то закрытие произойдет автоматически при завершении работы программы. Однако при автоматическом закрытии файла информация из буфера файла *не переносится* на диск, и как следствие часть информации может пропасть.

Внимание!!! После записи информации в файл его обязательно закрывать с помощью процедуры `CloseFile`. Однако при чтении данных из файла нет необходимости в обязательном его закрытии.

7.2.4 Процедура rename

Переименование файла, связанного с файловой переменной `f`, осуществляется в то время, когда он закрыт, при помощи процедуры `rename(f,s)`, где `f` – файловая переменная, `s` – новое имя файла (строковая переменная).

7.2.5 Процедура erase

Удаление файла, связанного с переменной `f`, выполняется посредством процедуры `erase(f)`, в которой `f` также является именем файловой переменной. Для корректного выполнения этой операции файл должен быть закрыт.

7.2.6 Функция eof

Функция `eof(f)` (end of file), где `f` – имя файловой переменной, принимает значение «истина» (`true`), если достигнут конец файла, иначе – «ложь» (`false`). С помощью этой функции можно проверять, достигнут ли конец файла и можно ли считывать очередную порцию данных.

7.2.7 Чтение и запись данных в файл

Для записи данных в файл можно использовать процедуру `write`:

```
write(f, x1, x2,..., xn);
write(f, x);
```

здесь

`f` — имя файловой переменной,

`x`, `x1`, `x2`, ..., `xn` — имена переменных, значения из которых записываются в файл.

Тип компонентов файла обязательно должен совпадать с типом переменных. При выполнении процедуры `write` значения `x1`, `x2`, ..., `xn` последовательно записываются в файл (начиная с текущей позиции), связанный с файловой переменной `f`.

Для чтения информации из файла, связанного с файловой переменной `f`, можно воспользоваться процедурой `read`:

```
read(f, x1, x2, x3,..., xn);
read(f, x);
```

здесь

`f` — имя файловой переменной,

`x`, `x1`, `x2`, ..., `xn` — имена переменных, в которые считываются значения из файла.

Процедура `read` последовательно считывает компоненты из файла, связанного с файловой переменной `f`, в переменные `x1`, `x2`, ..., `xn`. При считывании очередного значения доступным становится следующее. Следует помнить, что процедура `read` не проверяет, достигнут ли конец файла. За этим нужно следить с помощью функции `eof`.

Для того чтобы записать данные в файл, необходимо выполнить следующее:

1. Описать файловую переменную.
2. Связать ее с физическим файлом (процедура `AssignFile`).
3. Открыть файл для записи (процедура `rewrite`).
4. Записать данные в файл (процедура `write`).
5. Обязательно закрыть файл (процедура `CloseFile`).

Рассмотрим создание компонентного файла на примере решения следующей несложной задачи.

ЗАДАЧА 7.1. Создать типизированный файл и записать туда `n` вещественных чисел.

Алгоритм решения задачи состоит из следующих этапов:

1. Открыть файл для записи с помощью оператора `rewrite`.
2. Ввести значение *n*.
3. В цикле (при *i* меняющемся от `1` до `n`) вводим очередное вещественное число *a*, которое сразу же записываем в файл с помощью процедуры `write`.
4. Закрываем файл с помощью процедуры `closefile`.

Текст консольного приложения, предназначенного для решения

данной задачи приведен ниже.

```
program pr1;
{$mode objfpc}{$H+}
uses
  Classes, SysUtils
  { you can add units after this };
 var f:file of real;
     i,n:integer;
     a:real;
begin
//Связываем файловую переменную с файлом
//на диске.
AssignFile(f,'/home/pascal/6/pr1/abc.dat');
//Открываем пустой файл для записи.
     rewrite(f);
//Определяем количество элементов в файле.
     write('n=');
     readln(n);
//В цикле вводим очередной элемент и
//записываем его в файл.
     for i:=1 to n do
     begin
         write('a=');
         readln(a);
         write(f,a);
     end;
//Закрываем файл.
//При записи данных в файл это делать
//обязательно.
     CloseFile(f)
end.
```

Рассмотрим следующую задачу

ЗАДАЧА 7.2. В папке */home/evgeniy/pascal/6/pr2/* находятся файлы вещественных чисел с расширением dat. В выбранном пользователем файле удалить все числа, меньшие среднего арифметического, расположенные между максимальным и минимальным элементами.

В задаче предполагается выбор пользователем файла для обра-

ботки. Для этого понадобится специальный компонент для выбора файла. Решение задачи 7.2 начнем со знакомства с компонентом `OpenDialog`  . Это компонент предназначен для создания стандартного диалогового окна выбора файла и расположен первым на странице Dialogs (см. рис. 7.1).

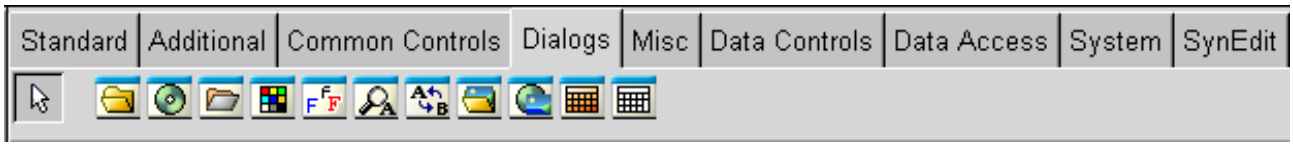

Рисунок 7.1: Компоненты страницы Dialogs

Среди основных свойств этого компонента можно выделить:

FileName: String — полное имя выбранного пользователем файла;

Filter: String — строка, которая возвращает фильтры отбираемых файлов; это свойство можно задать с помощью редактора фильтров или с помощью специальной строки. Для вызова редактора необходимо щелкнуть по кнопке  , после чего появится окно редактора фильтров (см. рис. 7.2).

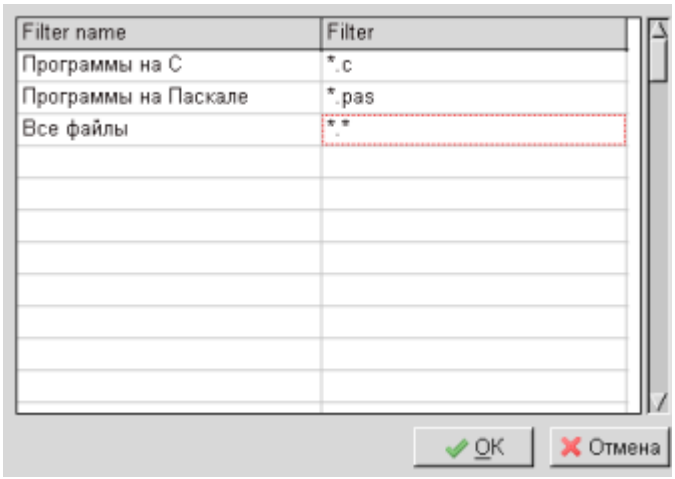

Рисунок 7.2: Редактор фильтров

Окно редактора фильтров состоит из двух столбцов: *Filter name* — имя фильтра (например, все файлы, программы на Паскале); *Filter* — соответствующая имени фильтра маска (фильтру «все файлы» соответствует маска «*.*», фильтру «программы на Паскале» — маска «*.pas».

Специальная строка состоит из последовательных имен фильтров и соответствующих им масок, разделенных символом «|». Специальная строка, соответствующая фильтру, представленному на рис.7.2, имеет вид.

```
OpenDialog1.Filter:=
    'Программы на C|*.c|Программы на Паскале|*.pas|Все файлы|*.*';
```

Первый фильтр в списке является фильтром по умолчанию. В примере на рис. 7.2 фильтр по умолчанию — «Программы на С». `InitialDialog` — имя каталога по умолчанию для выбора файлов.

`DefaultExt` — расширение, добавляемое к имени файла по умолчанию, если пользователь при ручном вводе имени файла не указал расширение.

Основным методом для компонента `OpenDialog` — является логическая функция `Execute`, которая приводит к открытию диалогового окна в соответствии со свойствами компонента. Функция `Execute` возвращает `true`, если пользователь выбрал файл каким-либо методом. Если пользователь нажал в диалоговом окне `Отмена` (`Cancel`), то метод `Execute` возвращает `false`. Имя выбранного файла возвращается в свойстве `FileName`.

Таким образом, вызов диалогового окна можно записать так.

```
Var
s:String;
begin
     if OpenDialog1.Execute then
     s:=OPenDialog1.FileName;
{Имя файла, выбранного пользователем, хранится в переменной s}.
end;
```

Разобравшись с компонентом для выбора файла, вернемся к задаче 7.2. Можно выделить следующие этапы ее решения.

1. Выбор файла.
2. Чтение данных из файла в массив вещественных чисел.
3. В массиве вещественных чисел удалить все числа, меньшие среднего арифметического, расположенные между максимальным и минимальным элементами.
4. Запись преобразованного массива в файл.

Разработку программы начнем с создания графического приложения (**Проект — Создать Проект — Приложение**).

На форме расположим следующие компоненты:

1. Две метки `Label1` и `Label2` для подписи.
2. `Memo1` — компонент, в котором будем хранить содержимое исходного файла.
3. `Memo2` — компонент, в котором хранится преобразованный файл.
4. `OpenDialog1` — компонент для выбора имени обрабатываемого файла.

5. Buttom1 — кнопка для запуска программы.
6. Buttom2 — кнопка для завершения программы.

Установим следующие свойства формы и компонентов (см. табл. 7.1 — 7.8).

Таблица 7.1: Свойства формы

Свойство	Caption	Name
Значение	Преобразование файла	Form_File

Таблица 7.2: Свойства метки label1

Свойство	Caption	Visible
Значение	Файл до преобразования	False

Таблица 7.3: Свойства метки label2

Свойство	Caption	Visible
Значение	Файл после преобразования	False

Таблица 7.4: Свойства компонента Memo1

Свойство	Lines	Visible
Значение	''	False

Таблица 7.5: Свойства компонента Memo2

Свойство	Lines	Visible
Значение	''	False

Таблица 7.6: Свойства компонента OpenDialog1

Свойство	InitDialog	DefaultExt
Значение	/home/evgeniy/pascal/6/pr2/	dat

Таблица 7.7: Свойства кнопки Button1

Свойство	Caption	Name	Visible
Значение	Преобразовать файл	Button_File	True

Таблица 7.8: Свойства кнопки Button2

Свойство	Caption	Name	Visible
Значение	Закрыть	Button_Close	False

Расположим компоненты на форме подобно показанному на рис. 7.3. При запуске программы на выполнение все компоненты, кроме кнопки «Преобразовать файл», будут невидимыми, свойство Visible установлено в False. Окно приложения при запуске пока-

зано на рис. 7.4.

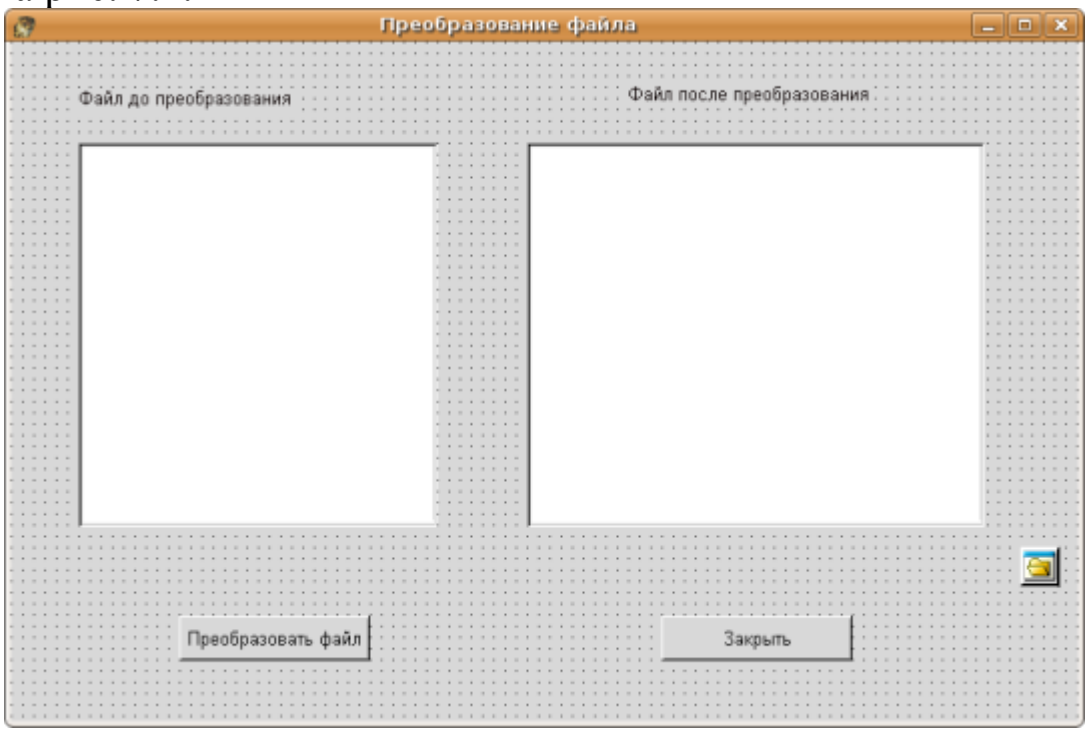

Рисунок 7.3: Форма с расположенными на ней компонентами

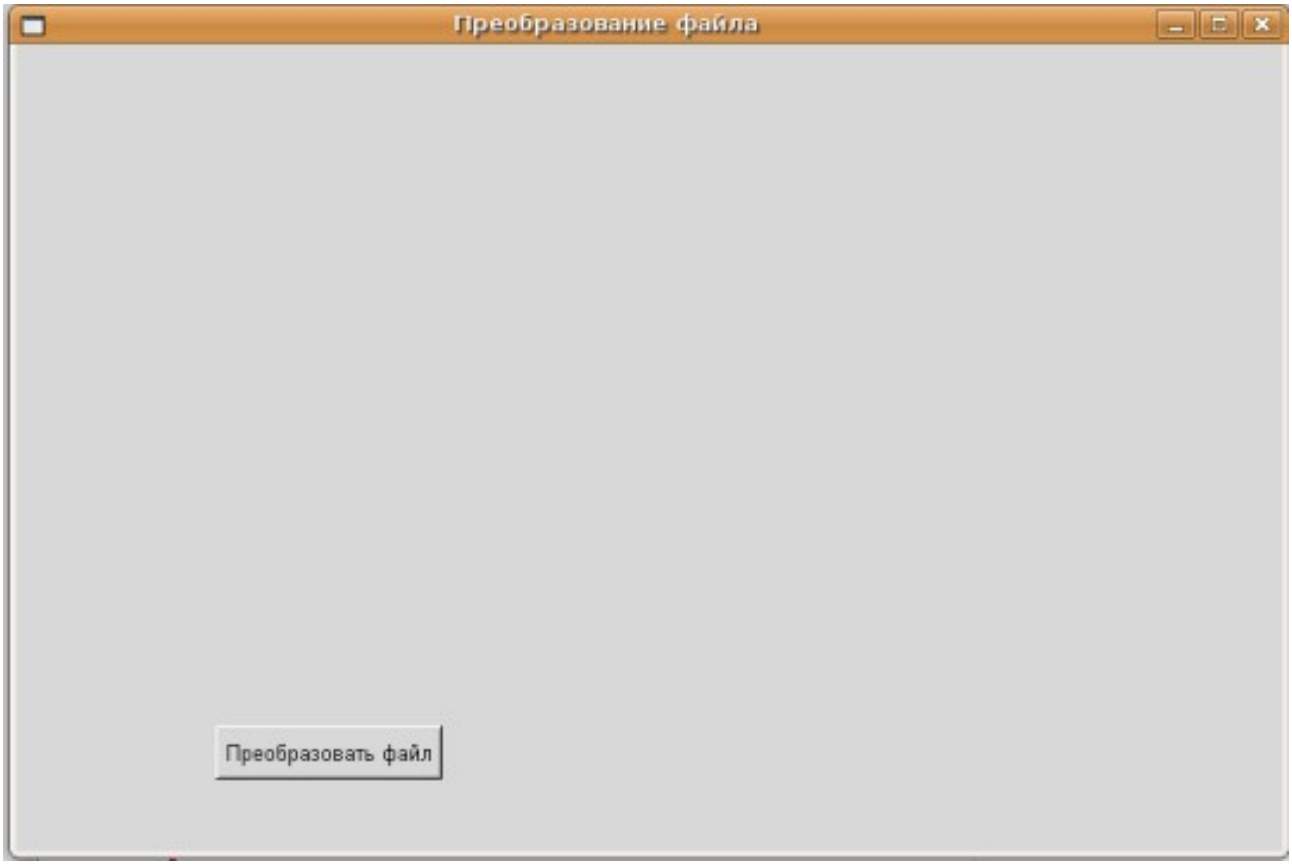

Рисунок 7.4: Приложение при запуске

Проектирование интерфейса приложения завершено. Осталось написать тексты обработчиков событий.

При создании формы установим свойство Filter компонента OpenDialog1. Поэтому подпрограмма FormCreate будет такой.

```
procedure TForm_File.FormCreate(Sender: TObject);
   begin
   OpenDialog1.Filter:=
   'Файлы вещественных чисел|*.dat|Все файлы|*.*';
   end;
```

При щелчке по кнопке «Преобразовать файл» должно происходить следующее:

1. Выбор файла из диалогового окна.
2. Считывание данных из файла в массив вещественных чисел.
3. Установка свойства Visible в True у компонентов label1 и Memo1.
4. Вывод содержимого массива в Memo1.
5. Преобразование массива.
6. Запись преобразованного массива в файл.
7. Установка свойства Visible в True у компонентов label2 и Memo2.
8. Вывод преобразованного массива в Memo2.
9. Установка свойства Visible в True у компонента Button_Close.

Ниже приведен текст обработки события Button_FileClick с подробными комментариями.

```
procedure Tform_File.Button_FileClick(
                                  Sender: TObject);
   var
   f:file of real;
   s:string;
   a:array[1..100] of real;
   nmax,nmin,i,j,n:integer;
   sum,max,min:real;
   begin
   //Открываем диалоговое окно для выбора файла.
    if opendialog1.execute then
    begin
   //Имя выбранного файла считываем в s.
       s:=OpenDialog1.FileName;
```

```
//Связываем файловую переменную
//с файлом на диске.
    AssignFile(f,s);
//Открываем файл для чтения.
    Reset(f);
//Обнуляем счетчик элементов массива.
    n:=0;
//Делаем видимыми первую
//метку и компонент Memo1.
    label1.Visible:=true;
    Memo1.Visible:=true;
//Переменная sum служит для
//нахождения суммы компонентов файла.
    Sum:=0;
//Пока не встретится конец файла,
//будем считывать очередной компонент.
    while not eof(f) do
    begin
//Индекс массива увеличиваем на 1.
        n:=n+1;
//Считываем очередной элемент из
//файла в массив.
        read(f,a[n]);
//Накапливаем сумму элементов массива.
        sum:=sum+a[n];
//Выводим очередной элемент в Memo1.
        Memo1.Lines.Add(FloatToStr(a[n]));
    end;
//Закрываем файл.
    CloseFile(f);
//Находим среднее арифметическое
//элементов массива.
    sum:=sum/n;
//Поиск максимального, минимального
//элементов в массиве и их индексов.
    max:=a[1];min:=max;nmin:=1;nmax:=1;
    for i:=2 to n do
    begin
```

```
        if a[i]>max then
        begin
            max:=a[i]; nmax:=i;
        end;
        if a[i]<min then
        begin
            min:=a[i]; nmin:=i;
        end;
    end;
//Если максимальный элемент расположен
//раньше минимального, то меняем местами
//nmin и nmax.
    if nmax<nmin then
    begin
        i:=nmax;
        nmax:=nmin;
        nmin:=i;
    end;
    i:=nmin+1;
//Цикл для удаления элементов,
//расположенных между максимальным
// и минимальным.
    while(i<nmax) do
    begin
//Если очередной элемент массива
//меньше среднего арифметического, то
        if a[i]<sum then
        begin
//удаляем его.
            for j:=i to n-1 do a[j]:=a[j+1];
//После удаления уменьшаем количество
//элементов и номер максимального на 1.
            n:=n-1;
            nmax:=nmax-1;
        end
//Если очередной элемент массива
//больше среднего арифметического, то
        else
```

```
//переходим к следующему.
            i:=i+1;
    end;
//Делаем видимыми вторую метку
//и компонент Memo2.
    label2.Visible:=true;
    Memo2.Visible:=true;
    rewrite(f);//Открываем файл для записи.
//Преобразованный массив записываем
//в файл и выводим в компонент Memo2.
    for i:=1 to n do
    begin
        write(f,a[i]);
        Memo2.Lines.Add(FloatToStr(a[i]));
    end;
//Закрываем файл, делаем видимой вторую кнопку.
    CloseFile(f);
    Button_Close.Visible:=True;
end;
end;
```

При щелчке по кнопке «**Закрыть**» программа должна завершать свою работу. Поэтому текст обработчика щелчка по кнопке будет очень простым.

```
procedure Tform_File.Button_CloseClick(
                                Sender: TObject);
begin
    Close;
end;
```

Проверим функционирование созданного приложения. После запуска программы появляется окно, подобное представленному на рис. 7.4. Щелкаем по кнопке `Преобразовать файл`, появляется окно, подобное представленному на рис. 7.5. После щелчка по кнопке OK окно программы становится похожим на представленное на рис. 7.6. Щелчок по кнопке `Закрыть` завершает работу приложения. Задача 7.2 решена.

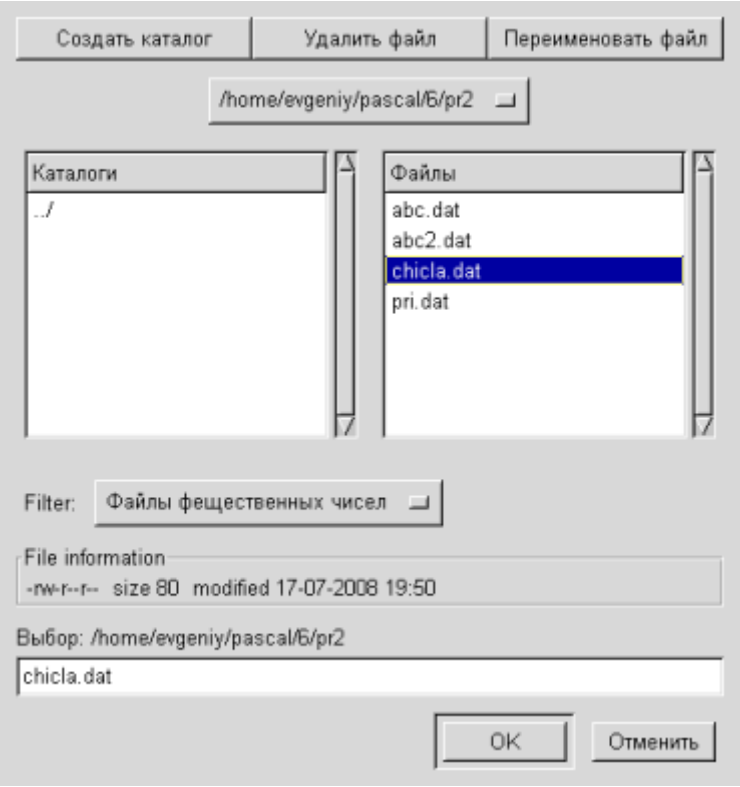

Рисунок 7.5: Диалоговое окно выбора файла

В связи с тем что файл — последовательная структура данных, при решении задачи пришлось содержимое файла целиком считывать в массив, проводить обработку данных в массиве и только затем записывать в файл. Такой метод не всегда удобен. Кроме того, при считывании данных из файла в массив мы ограничены описанием статического массива. Если в файле окажется больше элементов, чем при описании статического массива (в нашем случае >100), то программа не будет правильно работать.

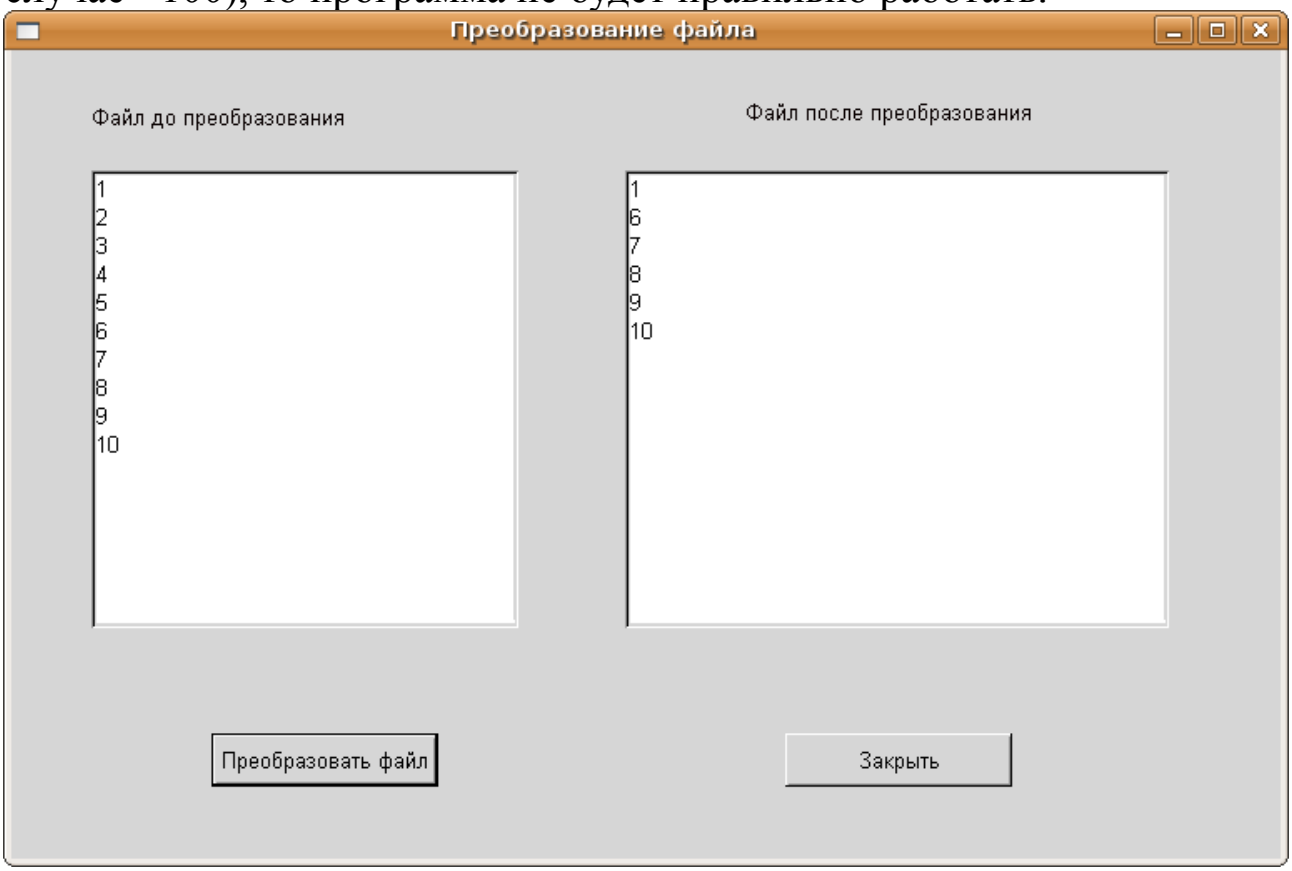

Рисунок 7.6: Окно с результатами работы программы

Использование динамического массива не решит эту проблему, потому что неизвестно количество элементов в массиве. Для решения этой программы в языке FreePascal реализован прямой доступ к файлу с помощью рассмотренных далее подпрограмм работы с файлами.

Одной из проблем при копировании данных из типизированного файла в массив является невозможность корректного выделения памяти под массив. При выделении памяти заранее неизвестно, сколько элементов находится в файле. Для решения этой задачи существует функция `filesize(f)` (`f`– файловая переменная), которая возвращает значение типа `longint` - число реальных компонентов в открытом файле, связанном с файловой переменной `f`. Для пустого файла функция возвращает число `0`.

Применим функцию `filesize` для переписывания вещественных чисел из файла в массив.

ЗАДАЧА 7.3. Переписать содержимое файла вещественных чисел в массив.

```
program Project1;
{$mode objfpc}{$H+}
uses  Classes, SysUtils
            { you can add units after this };
//Massiw - тип данных — массив
//из 1000 элементов.
type
    massiw=array[1..1000] of real;
var
f:file of real;
//a -указатель на массив;
a:^massiw;
n,i:word;
begin
//Связываем файловую переменную
//реальным файлом на диске.
assignfile(f,'/home/pascal/6/pr1/abc.dat');
//Открываем файл для чтения
reset(f);
//Записываем в переменную n - количество
```

```
//элементов в файле f.
n:=filesize(f);
writeln('в файле ',n, ' чисел');
//Выделяем память для динамического массива a
//из n вещественных чисел.
getmem(a,n*sizeof(real));
for i:=1 to n do
begin
//Считываем очередной элемент из файла
//в массив.
read(f,a^[i]);
//Вывод элемента на экран.
write(a^[i]:1:3, ' ')
end;
//Освобождаем память, выделенную
//для динамического массива.
freemem(a,n*sizeof(real));
//Закрываем файл.
closefile(f);
readln;
end.
```

Функция `filepos(f)` возвращает значение типа `longint` – текущую позицию в открытом файле, связанном с файловой переменной `f`. Сразу после открытия файла значение `filepos(f)` равно 0. После прочтения последнего компонента из файла значение `filepos(f)` совпадает со значением `filesize(f)`. Достижение конца файла можно проверить с помощью условия:

```
if filepos(f)=filesize(f) then
writeln('достигнут конца файла');
```

Процедура `seek(f,n)` устанавливает указатель в открытом файле, связанном с файловой переменной `f`, на компонент с номером `n` (нумерация компонентов идет от `0`). Затем значение компонента может быть считано.

Процедура `truncate(f)`, где `f` – имя файловой переменной, отсекает часть открытого файла, начиная с текущего компонента, и подтягивает на его место конец файла.

Использование процедур `seek` и `truncate` рассмотрим на при-

мере решения следующих двух несложных задач по обработке файлов.

ЗАДАЧА 7.4. В файле вещественных чисел */home/pascal/6/pr1/abc.dat* поменять максимальный и минимальный элементы.

Рассмотрим два варианта решения этой задачи.

В первом консольном варианте программы — после считывания компонентов файла в массив происходит поиск минимального и максимального элементов массива и их индексов. Затем максимальное и минимальное значения перезаписываются в файл.

```
program Project1;
{$mode objfpc}{$H+}
uses Classes, SysUtils
{ you can add units after this };
var f:file of real;
i, nmax, nmin :integer;
a:array[0..200] of real;
max,min:real;
begin
assignfile(f,'/home/pascal/6/pr1/abc.dat');
reset(f);
//Считываем компоненты файла в массив a.
for i:=0 to filesize(f)-1 do
begin
read(f,a[i]);
write(a[i]:1:2,' ');
end;
//Начальное присваивание максимального и
//минимального элементов массива и их индексов.
max:=a[0]; nmax:=0;
min:=a[0]; nmin:=0;
//Основной цикл для поиска максимального и
//минимального элементов массива и их индексов.
for i:=1 to filesize(f)-1 do
begin
if a[i]>max then
begin
max:=a[i];
nmax:=i
```

```
end;
if a[i]<min then
begin
min:=a[i];
nmin:=i
end;
end;
//Перезапись максимального и
//минимального значений в файл.
//Передвигаем указатель файла
//к максимальному элементу.
seek(f,nmax);
//Записываем на место максимального
//элемента минимальный.
write(f,min);
//Передвигаем указатель файла
//к минимального элементу.
seek(f,nmin);
//Записываем на место
//максимального элемента минимальный.
write(f,max);
//Обязательно закрываем файл.
closefile(f);readln;
end.
```

Вторую версию программы напишем как полноценное приложение. Использовать массив в программе не будем. Будет организован единственный цикл, в котором очередное значение считывается в переменную `a` и осуществляется поиск минимального и максимального элементов среди компонентов файла и их индексов. Затем происходит перезапись в файл максимального и минимального значений.

Разработку программы начнем с создания шаблона графического приложения (**Проект — Создать Проект — Приложение**).

На форме расположим следующие компоненты:

1. Две метки `Label1` и `Label2` для подписи.
2. `Edit1` — текстовое поле редактирования, в котором будем хранить содержимое исходного файла.
3. `Edit2` — текстовое поле редактирования, в котором хранится файл после преобразования.

4. `OpenDialog1` — компонент для выбора имени обрабатываемого файла.
5. `Button1` — кнопка для запуска программы.
6. `Button2` — кнопка для завершения программы.

Расположим компоненты на форме примерно так, как показано на рис. 7.7.

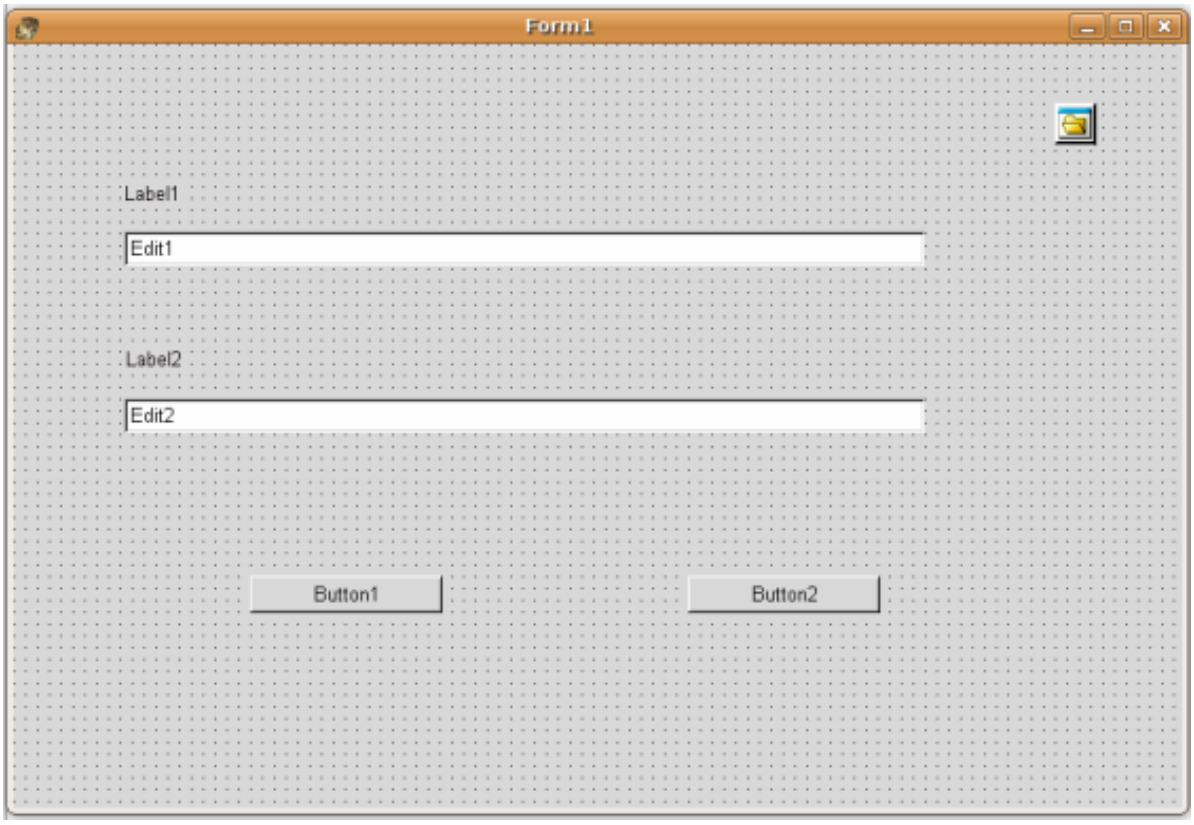

Рисунок 7.7: Окно проекта с компонентами

Основные свойства компонентов будем устанавливать программно при создании формы. Поэтому подпрограмма `FormCreate` будет такой.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Form1.Caption:=
           'Обмен максимального и
           минимального элементов в файле';
    label1.Caption:='Исходный файл';
    label1.Visible:=False;
    label2.Caption:='Файл после
                              преобразования';
```

```
    label2.Visible:=False;
    Edit1.Text:='';
    Edit1.Visible:=false;
    Edit2.Text:='';
    Edit2.Visible:=false;
    OpenDialog1.Filter:=
'Файлы вещественных чисел|*.dat|Все файлы|*.*';
OpenDialog1.InitialDir:='/home/pascal/6/pr1';
    Button1.Caption:='Преобразовать файл';
    Button2.Caption:='Выход';
    Button2.Visible:=False;
end;
```

Авторы надеются, что читателям, дочитавшим книгу до этого момента, текст процедуры `TForm1.FormCreate` понятен без пояснений.

При запуске программы окно формы будет иметь вид, представленный на рис. 7.8.

Основные действия программа будет осуществлять при щелчке по кнопке **Преобразовать файл**, текст соответствующей подпрограммы с комментариями приведен ниже.

```
procedure TForm1.Button1Click(Sender: TObject);
var
f:file of real;
i,nmax,nmin:integer;
a,max,min:real;
s1,s:string;

begin
//Открываем диалоговое окно для выбора файла.
if OpenDialog1.Execute then
begin
//В переменную s записываем полное имя файла.
    s:=OpenDialog1.FileName;
//Связываем файловую переменную
//с файлом на диске.
    assignfile(f,s);
```

Рисунок 7.8: Окно программы решения задачи 7.4 при запуске

```
//Открываем файл в режиме чтения.
    reset(f);
//Делаем видимыми первую метку и поле ввода.
    label1.Visible:=true;
    Edit1.Visible:=true;
//В строке s1 будем формировать
//содержимое файла.
    S1:='';
//Цикл для последовательного чтения
//всех элементов из файла
// вещественных чисел.
    for i:=0 to filesize(f)-1 do
    begin
//Считывание очередного элемента
//массива в переменную a.
        read(f,a);
        if i=0 then
//Начальное присваивание максимального
//и минимального значений и их индексов.
        begin
            max:=a;
```

```
            nmax:=i;
            min:=a;
            nmin:=i;
        end
        else
        begin
//Сравнение текущего значения с
//максимальным (минимальным).
            if max<a then
            begin
                max:=a;
                nmax:=i
            end;
            if min>a then
            begin
                min:=a;
                nmin:=i
            end
        end;
//Добавление очередного значения
//к выходной строке s1.
        s1:=s1+FloatToStr(a)+' ';
    end;
//Вывод содержимого файла
//в первое текстовое поле.
    Edit1.Text:=s1;
//Запрет изменения текстового поля.
    Edit1.ReadOnly:=true;
//Перезапись максимального
//и минимального значений в файл.
//Передвигаем указатель файла
//к максимальному элементу.
    seek(f,nmax);
//Записываем на место
//максимального элемента минимальный.
    write(f,min);
//Передвигаем указатель
//файла к максимальному элементу.
```

```
        seek(f,nmin);
    //Записываем на место
    //минимального элемента максимальный.
        write(f,max);
    //Обязательно закрываем файл.
        closefile(f);
        reset(f);
    //Делаем видимыми вторую метку и поле ввода.
        label2.Visible:=true;
        Edit2.Visible:=true;
    //Считываем данные из преобразованного файла
        s1:='';
        for i:=0 to filesize(f)-1 do
        begin
            read(f,a);
    //Добавление очередного значения
    //из преобразованного файла к строке s1.
            s1:=s1+FloatToStr(a)+' ';
        end;
    //Вывод содержимого преобразованного
    //файла во 2-е текстовое поле.
        Edit2.Text:=s1;
    //Запрет изменения текстового поля.
        Edit2.ReadOnly:=true;
    //Делаем видимой вторую кнопку.
        Button2.Visible:=True;
        CloseFile(f);
        end;
    end;
```

При щелчке по кнопке **Выход** программа должна завершать свою работу. Поэтому текст обработчика щелчка по кнопке будет очень простым.

```
procedure Tform_File.Button2Click(Sender:
                                                TObject);
    begin
        Close;
    end;
```

При щелчке по кнопке `Преобразовать файл`, появляется окно

выбора файла, подобное представленному на рис. 7.5. После выбора файла в файле происходит обмен максимального и минимального элементов и вывод содержимого файла до и после преобразования (см. рис. 7.9).При щелчке по кнопке `Выход` работа программы завершится.

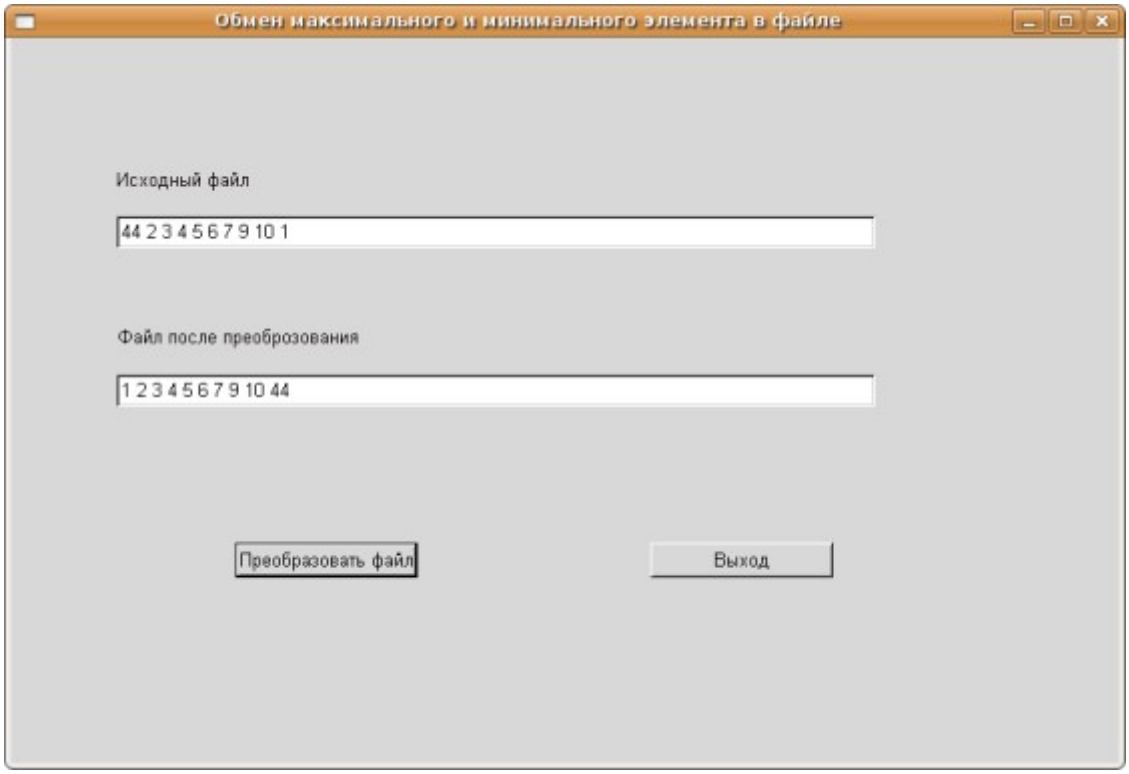

Рисунок 7.9: Окно программы решения задачи 7.4 после преобразования файла

При решении задачи 7.4 была использована процедура `seek`, с помощью которой стало возможным изменение данных в файле непосредственно на диске, без считывания в массив. Однако следует помнить, что большое количество отдельных обращений к файлу занимает больше времени, чем однократное обращение к файлу для обработки большого объема данных.

ЗАДАЧА 7.5. Задан файл вещественных чисел abc.dat. Удалить из него максимальный и минимальный элементы.

Рассмотрим две программы, решающие эту задачу. Алгоритм первой состоит в следующем: считываем компоненты файла в массив, в котором находим максимальный и минимальный элементы и их номера. Открываем файл для записи и вносим в него все элементы, за исключением максимального и минимального.

```
program Project1;
```

```
{$mode objfpc}{$H+}
uses  Classes, SysUtils
{ you can add units after this };
var
f:file of real;
max,min:real;
j, i,nmax,nmin:integer;
a:array [1..300] of real;
begin
assignfile(f,'/home/pascal/6/pr1/abc.dat');
reset(f);
//Запоминаем в переменной j
//количество компонентов в файле.
j:=filesize(f);
//Считываем компоненты файла в массив a.
for i:=1 to j do read(f,a[i]);
for i:=1 to j do write(a[i]:1:2,' ');
writeln;
closefile(f);
//Открываем файл для записи.
rewrite(f);
//Начальное присваивание максимального и
//минимального элементов массива и его индекса.
max:=a[1];min:=a[1];
nmax:=1;nmin:=1;
//Основной цикл для поиска максимального и
//минимального элементов массива и его индекса.
for i:=2 to j do
begin
     if a[i]>max then
     begin
         max:=a[i];
         nmax:=i
     end;
     if a[i]<min then
     begin
         min:=a[i];
         nmin:=i;
```

```
        end;
    end;
    //Перезапись элементов массива в файл,
    //за исключением элементов с
    //номерами nmax и nmin.
    writeln('Преобразованный файл');
    for i:=1 to j do
        if (i<>nmax)and (i<>nmin) then
        begin
            write(f,a[i]);
    //Вывод записанного в файл элемента на экран.
            write(a[i]:1:2,' ');
        end;
    closefile(f);
    readln;
    end.
```

Вторая программа работает следующим образом. Находим максимальный и минимальный элементы и их номера среди компонентов файла. Если nmin>nmax, то меняем содержимое переменных nmin и nmax. Далее элементы, лежащие между минимальным и максимальным (формально между элементами с номерами nmin и nmax), сдвигаем на один порядок влево. Тем самым мы убираем элемент с номером nmin. После этого все компоненты, лежащие после элемента с номером nmax, сдвинем на два порядка влево. Этим мы сотрем максимальный элемент. Затем два последних компонента в файле необходимо удалить.

```
program Project1;
{$mode objfpc}{$H+}
uses  Classes, SysUtils
{ you can add units after this };
var
f:file of real;
a:real;
max,min:real;
i,nmax,nmin:integer;
begin
assign(f,'/home/pascal/6/pr1/abc.dat');
reset (f);
```

```
//Поиск максимального и
//минимального элементов в файле и их индексов.
writeln('Исходный файл');
for i:=0 to filesize(f)-1 do
begin
     read(f,a);
     write(a:1:2,' ');
     if i=0 then
     begin
         max:=a;
         nmax:=i;
         min:=a;
         nmin:=i
     end
     else
     begin
         if a>max then
         begin
             max:=a;
             nmax:=i;
         end;
         if a<min then
         begin
             min:=a;
             nmin:=i;
         end
     end
end;
writeln;
//Сравниваем nmin и nmax.
if nmax<nmin then
begin
     i:=nmax;
     nmax:=nmin;
     nmin:=i
end;
// Сдвигаем элементы, лежащие между
//компонентами с номерами nmin и nmax,
```

```
//на один влево.
for i:=nmin to nmax-2 do
begin
     seek(f,i+1);
     read(f,a);
     seek(f,i);
     write(f,a);
end;
//Сдвигаем элементы, лежащие после компонента
//с номером nmax, на два влево.
for i:=nmax to filesize(f)-3do
begin
     seek(f,i+1);
     read(f,a);
     write(a:1:2);
     seek(f,i-1);
     write(f,a);
     write(a:1:2);
end;
//Отрезаем последние два компонента.
truncate(f);
closefile(f);
reset(f);
//Вывод преобразованного файла.
writeln('Преобразованный файл');
for i:=1 to filesize(f) do
begin
        read(f,a);
//Вывод записанного в файл элемента на экран.
        write(a:1:2,' ');
     end;
closefile(f);
end.
```

В результате программы из файла вещественных чисел удалено наибольшее и наименьшее число.

Кроме типизированных файлов, широкое применение при обработке числовых данных получили бестиповые файлы.

7.3 Бестиповые файлы в языке Free Pascal

Для хранения данных различного типа удобно использовать бестиповые файлы. При открытии бестиповых файлов следует использовать расширенный синтаксис процедур `reset` и `rewrite`.

```
Reset(var f: File; BuferSize:word);
Rewrite(var f: File; BuferSize:word);
```

Необязательный параметр `BufSize` определяет размер блока передачи данных – количество байт, считываемых или записываемых в файл данных за одно обращение к нему. Если этот параметр отсутствует, то используется значение, устанавливаемое по умолчанию (128) при открытии файла. Наибольшей гибкости можно достичь при размере блока 1 байт.

Для записи данных в бестиповый файл используется процедура `BlockWrite`:

```
BlockWrite(var f:file; var X; Count:word;
                          var WriteCount:word);
```

здесь `f` – имя файловой переменной, `X` – имя переменной, из которой данные записываются в файл, `Count` – количество блоков размером `BufSize`, записываемых в файл, необязательный параметр `WriteCount` определяет *реальное количество записанных в файл блоков*.

Процедура `BlockWrite` записывает `Count` блоков в файл, связанный с файловой переменной `f`[77] из переменной `X`. При корректной записи в файл возвращаемое процедурой `BlockWrite` значение `WriteCount` совпадает с `Count`.

При чтении данных из бестипового файла используется процедура `BlockRead`:

```
BlockRead(var f:file; var Y; Count:word;
                          var ReadCount:word);
```

где `f` – имя файловой переменной, `Y` – имя переменной, в которую считываются данные из файла, `Count` – количество блоков размером `BufSize`, считываемых из файла, необязательный параметр `ReadCount` определяет количество блоков размером `BufSize`, реально считанных из файла.

`BlockRead` считывает из файла, связанного с файловой пере-

77 Размер блока определяется при выполнении процедур Reset, Rewrite.

менной f, Count блоков в переменную Y. При корректном чтении из файла возвращаемое процедурой BlockRead значение ReadCount должно совпадать с Count.

Рассмотрим работу с бестиповыми файлами на примере решения следующих задач.

Задача 7.6. В бестиповом файле хранится массив вещественных чисел и количество элементов в нем. Удалить из файла максимальное вещественное число.

Считаем, что файл уже существует, ниже приведен текст консольного приложения создания файла.

```
program Project1;
{$mode objfpc}{$H+}
uses
    Classes, SysUtils
    { you can add units after this };
type
massiv=array[1..100000] of real;
var  i,N:word;
    f:file;
    x:^massiv;
begin
Assignfile(f,'/home/pascal/6/pr_6/new_file.dat');
//Открываем файл для записи,
//размер блока передачи данных 1 байт.
rewrite(f,1);
//Ввод числа N — количества целых
//положительных чисел в файле
write('N=');readln(N);
//Записываем в файл f целое число N,
//а точнее, 2 блока (sizeof(word)=2)
//по одному байту из переменной N.
BlockWrite(f,N,sizeof(word));
//Выделяем память для N элементов массива x
//вещественных чисел.
getmem(x,N*sizeof(real));
//Ввод массива.
writeln('Введите массив');
for i:=1 to N do
```

```
begin
     write('x(',i,')=');
     readln(x^[i]);
end;
for i:=1 to N do
// Записываем в файл f вещественное
//число x^[i], а точнее, 8  блоков
//(sizeof(real)=8) по одному байту
//из переменной x^[i].
BlockWrite(f,x^[i],sizeof(real));
//Запись массива в файл можно осуществить
//и без цикла с помощью следующего обращения
//к функции BlockRead —
//BlockWrite(f,x^,N*sizeof(real));
//Запись массива в бестиповый файл без
//цикла с помощью одного оператора
//BlockWrite(f,x^,N*sizeof(тип))
//кажется авторам предпочтительнее.
//Закрываем файл.
CloseFile(f);
//Освобождаем память.
freemem(x,N*sizeof(real));
readln;
end.
```

Разработку программы решения задачи 7.6 начнем с создания шаблона графического приложения (**Проект — Создать Проект — Приложение**).

На форме расположим следующие компоненты:

1. Две метки `Label1` и `Label2` для подписи.
2. `Edit1` — текстовое поле редактирования, в котором будем хранить содержимое исходного файла.
3. `Edit2` — текстовое поле редактирования, в котором хранится файл после преобразования.
4. `OpenDialog1` — компонент для выбора имени обрабатываемого файла.
5. `Buttom1` — кнопка для преобразования файла.

Расположим компоненты на форме примерно так, как показано на рис. 7.10.

Основные свойства компонентов будем устанавливать программно при создании формы. Программа решения задачи 7.6 будет осуществляться при щелчке по кнопке. Полный текст модуля с необходимыми комментариями.

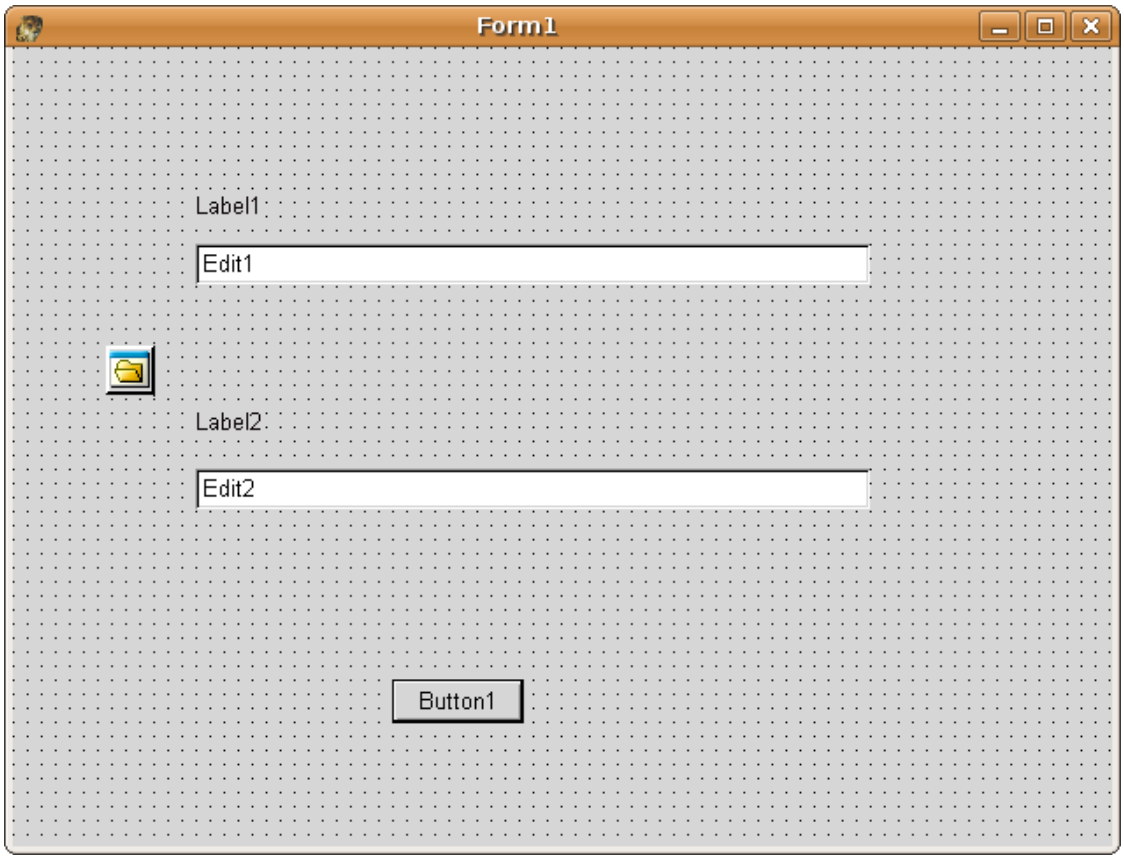

Рисунок 7.10: Окно формы с компонентами

```
   unit Unit1;
   {$mode objfpc}{$H+}
   interface
   uses
   Classes, SysUtils, LResources, Forms, Controls,
Graphics, Dialogs, StdCtrls;
   type
   { TForm1 }
   TForm1 = class(TForm)
   //На форме находятся:
   //Кнопка.
```

```
 Button1: Tbutton;
//Поле редактирования - для вывода
//содержимого исходного файла.
 Edit1: Tedit;
//Поле редактирования - для вывода
//содержимого файла после преобразования.
 Edit2: Tedit;
//Метки для подписи полей редактирования
 Label1: TLabel;
 Label2: Tlabel;
//Компонент для выбора файла
 OpenDialog1: TOpenDialog;
 procedure Button1Click(Sender: TObject);
 procedure FormCreate(Sender: TObject);
private
     private declarations }
public
    { public declarations }
end;
    massiv=array [1..50000] of real;
var
 Form1: TForm1;
 f:file;
implementation
{ TForm1 }

//Текст процедуры обработчика
//события «Щелчок по кнопке»;
procedure Tform1.Button1Click(Sender: TObject);
var
 x:^massiv;
 y:array[1..3000] of real;
 i,N:word;
 max:real;
 nmax:word;
 s:string;
begin
//Выбираем файл, который будем обрабатывать.
```

```
if OpenDialog1.Execute then
begin
//Имя обрабатываемого файла
//записываем в переменную s.
 s:=OpenDialog1.FileName;
//Связываем файловую переменную
//с именем файла, хранящимся в  переменной s.
 AssignFile(f,s);
//Открываем файл для чтения, размер
//блока передачи данных 1 байт.
 Reset(f,1);
//Считываем из файла f целое число (word) N ,
//а точнее, 2 блока (sizeof(word)=2) по одному
//байту из переменной N.
 BlockRead(f,N,sizeof(word));
//Выделяем память для N элементов массива x
//вещественных чисел.
 getmem(x,N*sizeof(real));
//Считываем из файла f N вещественных чисел
//в переменную x.
//Реально считываем N*sizeof(real)
//блоков по 1 байту.
//Заполняем массив X значениями,
//хранящимися в файле.
 BlockRead(f,x^,N*sizeof(real));
//Делаем видимыми первую метку
//и первый компонент Edit1.
 Label1.Visible:=true;
 Edit1.Visible:=true;
 for i:=1 to N do
//Добавление очередного значения из массива x
//к первому текстовому полю редактирования.
//В результате в первом текстом поле будет
//выведен исходный массив вещественных чисел.
Edit1.Text:=Edit1.Text+
        FloatToStrF(x^[i],ffFixed,5,2)+' ';
//Поиск максимального элемента и его номера.
 max:=x^[1];
```

```
 nmax:=1;
 for i:=2 to N do
 if x^[i]>max then
 begin
     max:=x^[i];
     nmax:=i;
 end;
//Удаление максимального элемента из массива.
 for i:=nmax to N-1 do
     x^[i]:=x^[i+1];
//Уменьшение количества элементов в массиве.
 N:=N-1;
//Закрываем файл.
 CloseFile(f);
 AssignFile(f,s);
//Открываем файл для записи.
 Rewrite(f,1);
//Записываем в файл f целое число N,
//а точнее, 2 блока
// (sizeof(word)=2) по одному байту из
//переменной N.
BlockWrite(f,N,sizeof(word));
BlockWrite(f,N,sizeof(word));
//Запись массива в файл с помощью
//обращения к функции BlockRead.
 BlockWrite(f,x^,N*sizeof(real));
//Закрываем файл.
 CloseFile(f);
//Освобождаем память.
 freemem(x,N*sizeof(word));
//Чтение данных из преобразованного файла
 AssignFile(f,s);
 Reset(f,1);
//Считываем из файла f целое число (word) N ,
//а точнее, 2 блока (sizeof(word)=2) по одному
//байту из переменной N.
 BlockRead(f,N,sizeof(word));
//Выделяем память для N элементов массива x
```

```
//вещественных чисел.
 getmem(x,N*sizeof(real));
//Считываем из файла f N вещественных чисел
//в переменную x. Реально считываем
//N*sizeof(real) блоков по 1 байту.
//Заполняем массив X значениями,
//хранящимися в файле.
 BlockRead(f,x^,N*sizeof(real));
//Делаем видимыми вторую метку
//и второй  компонент Edit2.
 Label2.Visible:=true;
 Edit2.Visible:=true;
//Содержимое массива x выводим
//в текстовое поле Edit2.
 for i:=1 to N do
Edit2.Text:=Edit2.Text+
            FloatToStrF(x^[i],ffFixed,5,2)+' ';
//Закрваем файл и освобождаем память.
 CloseFile(f);
 freemem(x,N*sizeof(real));
end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
//При создании формы устанавливаем
//свойства компонентов Edit1,
// Edit2, label1, label2 и кнопки Button1.
//Все компоненты,
//кроме кнопки, делаем невидимыми.
 Form1.Caption:='Работа с бестиповыми файлами';
 Label1.Width:=80;
 Label1.Caption:='Содержимое исходного файла';
 Edit1.Clear;
 Label2.Caption:=
        'Содержимое преобразованного файла';
 Label2.Width:=80;
 Edit2.Clear;
```

```
 Button1.Width:=150;
 Button1.Caption:='Преобразование файла';
 Label1.Visible:=false;
 Edit1.Visible:=false;
 Label2.Visible:=false;
 Edit2.Visible:=false;
end;
initialization
 {$I unit1.lrs}
end.
```

При запуске программы окно приложения будет таким, как показана на рис. 7.11. После щелчка по кнопке **Преобразовать файл** появится окно выбора файла (см. рис. 7.12). Окно программы после преобразования файла представлено на рис. 7.13.

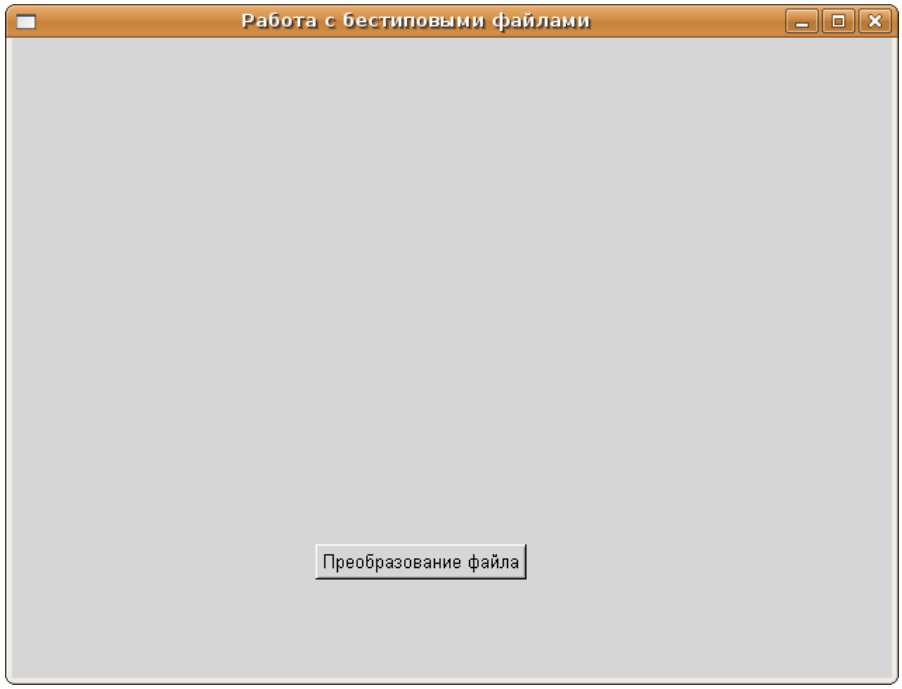

Рисунок 7.11: Окно программы решения задачи 7.6 при запуске

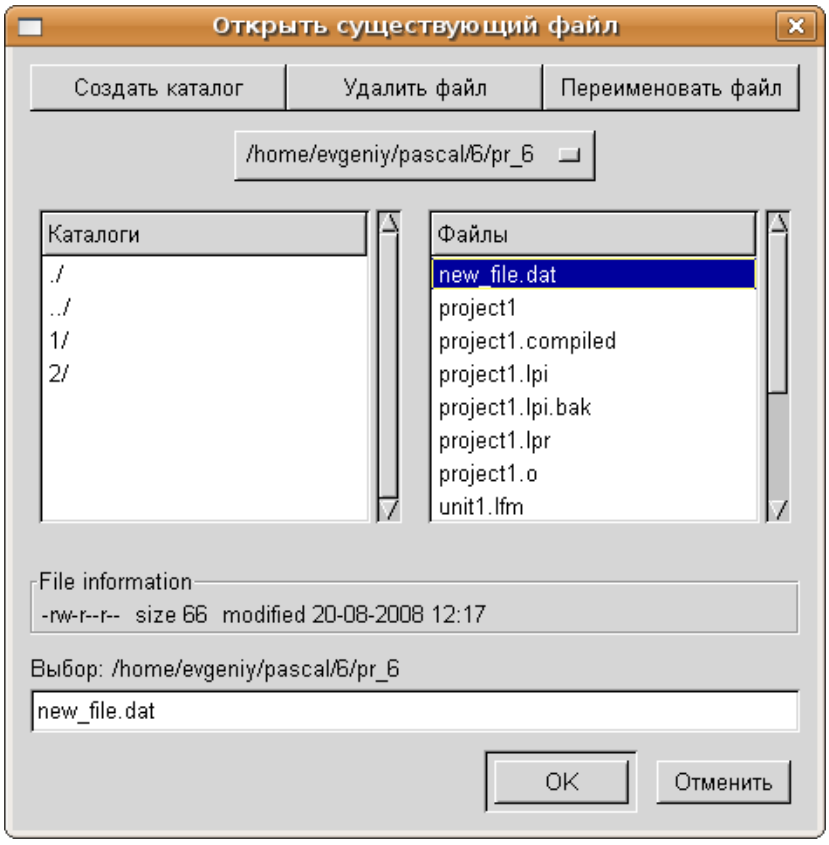

Рисунок 7.12: Окно выбора файла

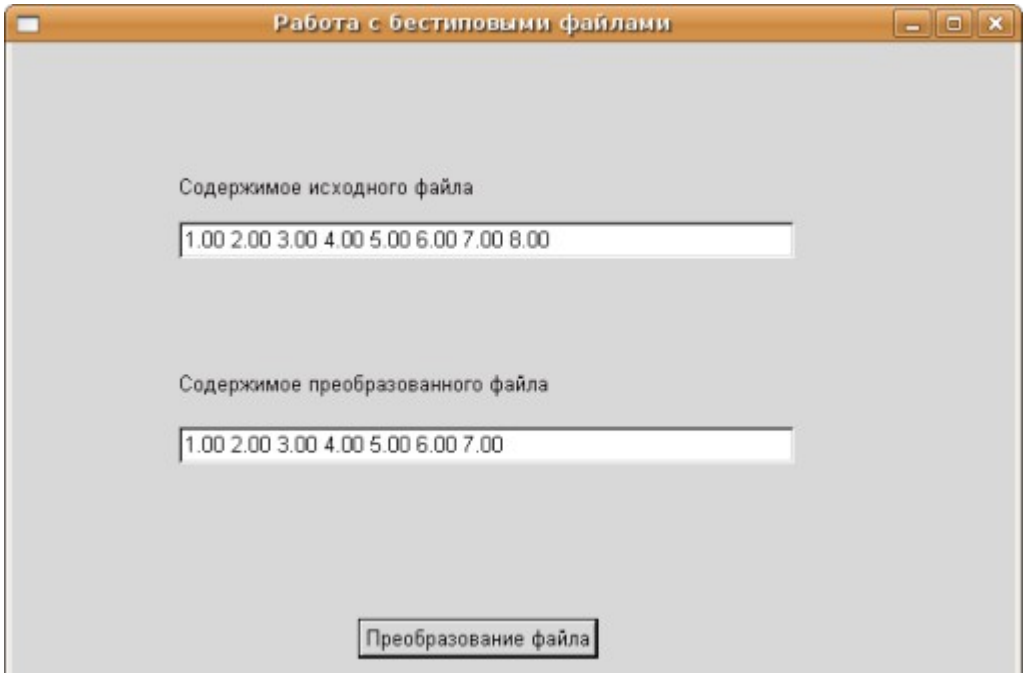

Рисунок 7.13: Окно работающей программы

Задача 7.7. В бестиповом файле хранятся матрицы вещественных чисел A(N,M), B(P,L) и их размеры. Умножить, если это возможно, A на B, результирующую матрицу $C = A \cdot B$ дописать в файл.

Решение задачи начнем с написания консольной программы создания файла. Для этого определимся со структурой файла. Пусть в нем хранится два значения типа `word N` и `M`, затем матрица вещественных чисел `A(N,M)`, потом `P` и `L` типа `word` и матрица `B(P,L)`.

Ниже приведен листинг подпрограммы создания бестипового файла с комментариями.

```
program Project1;
{$mode objfpc}{$H+}
uses
Classes, SysUtils
{ you can add units after this };
var
 f:file;
 i,j,n,m,l,p:word;
 a,b:array[1..20,1..20] of real;
begin
 Assign(f,'Prim.dat');
//Открываем файл для записи, размер
//блока передачи данных 1 байт.
 rewrite(f,1);
 write('N='); readln(N);
 write('M='); readln(M);
//Записываем в файл f целое число N,
//а точнее, 2 блока
//(sizeof(word)=2) по одному байту
//из переменной N.
 BlockWrite(f,N,sizeof(word));
//Записываем в файл f целое число M,
//а точнее, 2 блока (sizeof(word)=2)
//по одному байту из переменной M.
 BlockWrite(f,M,sizeof(word));
writeln('Matrica A');
 for i:=1 to N do
 for j:=1 to M do
 begin
//Вводим очередной элемент матрицы.
    read(A[i,j]);
//Записываем в файл f вещественное
```

```
//число A[i,j], а точнее, sizeof(real) блоков
//по одному байту из переменной A[i,j].
    BlockWrite(f,A[i,j],sizeof(real));
 end;
 write('L='); readln(L);
 write('P='); readln(P);
//Записываем в файл 2 блока по одному байту
//из переменной L.
 BlockWrite(f,L,sizeof(word));
//Записываем в файл 2 блока по одному
//байту из переменной P.
 BlockWrite(f,P,sizeof(word));
 writeln('Matrica B');
 for i:=1 to L do
 for j:=1 to P do
 begin
    read(B[i,j]);
//Записываем в файл sizeof(real) блоков
//по одному байту из переменной B[i,j].
    BlockWrite(f,B[i,j],sizeof(real));
 end;
//Закрываем файл.
 close(f);
end.
```

Теперь напишем программу, которая из бестипового файла считает матрицы A, B, их размеры, вычислит матрицу C как произведение A на B и допишет матрицу C в исходный файл.

```
program Project1;
{$mode objfpc}{$H+}
uses
Classes, SysUtils
{ you can add units after this };
var
 f:file;
 k,i,j,n,m,l,p:word;
 a,b,c:array[1..20,1..20] of real;
begin
//Связываем файловую переменную с файлом
```

```
//на диске.
 Assign(f,'Pr_7_6.dat');
//Открываем файл и устанавливаем размер
//блока в 1 байт.
 reset(f,1);
//Считываем в переменную N из файла f 2
//байта (sizeof(word)=2
//блоков по одному байту).
 BlockRead(f,N,sizeof(word));
//Считываем в переменную M из файла f 2 байта
//( sizeof(word)=2 блоков по одному байту).
 BlockRead(f,M,sizeof(word));
 for i:=1 to N do
 for j:=1 to M do
//Считываем в переменную A[i,j] из
//файла f sizeof(real) байт
//( sizeof(real) блоков по одному байту).
     BlockRead(f,A[i,j],sizeof(real));
//Считываем в переменную L из файла f 2 байта.
 BlockRead(f,L,sizeof(word));
//Считываем в переменную P из файла f 2 байта.
 BlockRead(f,P,sizeof(word));
 for i:=1 to L do
 for j:=1 to P do
//Считываем в переменную B[i,j] из
//файла f sizeof(real) байт.
     BlockRead(f,B[i,j],sizeof(real));
//Вывод матрицы A на экран.
 writeln('Matrica A');
 for i:=1 to N do
 begin
     for j:=1 to M do
         write(A[i,j]:1:2,' ');
     writeln
 end;
//Вывод матрицы B на экран.
 writeln('Matrica B');
 for i:=1 to L do
```

```
begin
    for j:=1 to P do
        write(B[i,j]:1:2,' ');
    writeln
end;
//Проверяем, возможно ли умножение матриц,
//если да,
if M=L then
begin
//то умножаем матрицу A на B
    for i:=1 to N do
        for j:=1 to P do
        begin
            c[i,j]:=0;
            for k:=1 to M do
                c[i,j]:=c[i,j]+a[i,k]*b[k,j]
        end;
//Вывод матрицы C.
    writeln('Matrica C=A*B');
    for i:=1 to N do
    begin
        for j:=1 to P do
            write(C[i,j]:1:2,' ');
        writeln
    end;
//Дописываем в файл f целое число N,  а точнее,
//2 блока по одному байту из переменной N.
    BlockWrite(f,N,sizeof(word));
//Дописываем в файл f целое число P, а точнее,
//2 блока по одному байту из переменной P.
    BlockWrite(f,P,sizeof(word));
    for i:=1 to N do
        for j:=1 to P do
//Записываем в файл f вещественное число
//C[i,j], а точнее, sizeof(real) блоков по
//одному байту из переменной C[i,j].
    BlockWrite(f,c[i,j],sizeof(real));
end
```

```
 else
     writeln('Умножение невозможно');
//Закрываем файл.
 close(f);
end.
```

7.4 Обработка текстовых файлов в языке Free Pascal

При работе с текстовыми файлами следует учесть следующее:

1. Действие процедур `reset`, `rewrite`, `close`, `rename`, `erase` и функции `eof` аналогично их действию при работе с компонентными (типизированными) файлами.

2. Процедуры `seek`, `truncate` и функция `filepos` не работают с текстовыми файлами.

3. При работе с текстовыми файлами можно пользоваться процедурой `append(f)`, где `f` – имя файловой переменной, которая служит для специального открытия файлов для дозаписи. Она применима только к уже физически существующим файлам, открывает и готовит их для добавления информации в конец файла.

4. Запись и чтение в текстовый файл осуществляются с помощью процедур `write`, `writeln`, `read`, `readln` следующей структуры:

```
read(f, x1, x2, x3, ..., xn);
read(f, x);
readln(f, x1, x2, x3, ..., xn);
readln(f, x);
write(f, x1, x2, x3, ..., xn);
write(f, x);
writeln(f, x1, x2, x3, ..., xn);
writeln(f, x);
```

В этих операторах `f` — файловая переменная. В операторах чтения (`read`, `readln`) `x, x1, x2, x3, ..., xn` — переменные, в которые происходит чтение из файла. В операторах записи `write`, `writeln` `x, x1, x2, x3, ..., xn` — переменные или константы, информация из которых записывается в файл.

Есть ряд особенностей при работе операторов `write`, `writeln`,

`read`, `readln` с текстовыми файлами. Имена переменных могут быть целого, вещественного, символьного и строкового типа. Перед записью данных в текстовый файл с помощью процедуры `write` происходит их преобразование в тип `string`. Действие оператора `writeln` отличается тем, что записывает в текстовый файл символ конца строки.

При чтении данных из текстового файла с помощью процедур `read`, `readln` происходит преобразование из строкового типа к нужному типу данных. Если преобразование невозможно, то генерируется код ошибки, значение которого можно узнать, обратившись к функции `IOResult`. Компилятор FreePascal позволяет генерировать код программы в двух режимах: с проверкой корректности ввода-вывода и без нее.

В программу может быть включен ключ режима компиляции. Кроме того, предусмотрен перевод контроля ошибок ввода-вывода из одного состояния в другое:

{$I+} – режим проверки ошибок ввода-вывода включен;

{$I-} – режим проверки ошибок ввода-вывода отключен.

По умолчанию, как правило, действует режим {$I+}. Можно многократно включать и выключать режимы, создавая области с контролем ввода и без него. Все ключи компиляции описаны в приложении.

При включенном режиме проверки ошибка ввода-вывода будет фатальной, программа прервется, выдав номер ошибки.

Если убрать режим проверки, то при возникновении ошибки ввода-вывода программа не будет останавливаться, а продолжит работу со следующего оператора. Результат операции ввода-вывода будет не определен.

Для опроса кода ошибки лучше пользоваться специальной функцией `IOResult`, но необходимо помнить, что опросить ее можно только один раз после каждой операции ввода или вывода: она обнуляет свое значение при каждом вызове. `IOResult` возвращает целое число, соответствующее коду последней ошибки ввода-вывода. Если `IOResult=0`, то при вводе-выводе ошибок не было, иначе `IOResult` возвращает код ошибки. Некоторые коды ошибок приведены в табл. 7.9.

Таблица 7.9: Коды ошибок

Код ошибки	Описание
2	файл не найден
3	путь не найден
4	слишком много открытых файлов
5	отказано в доступе
12	неверный режим доступа
15	неправильный номер диска
16	нельзя удалять текущую директорию
100	ошибка при чтении с диска
101	ошибка при записи на диск
102	не применена процедура Assign
103	файл не открыт
104	файл не открыт для ввода
105	файл не открыт для вывода
106	неверный номер
150	диск защищён от записи

Рассмотрим несколько практических примеров обработки ошибок ввода-вывода:

1. При открытии проверить, существует ли заданный файл и возможно ли чтение данных из него.

```
assign (f, 'abc.dat');
{$I-}
reset(f);
{$I+}
if IOResult<>0 then
writeln ('файл не найден или не читается')
else
begin
read(f,...);
...
close(f);
end;
```

2. Проверить, является ли вводимое с клавиатуры число целым.

```
var i:integer;
begin
{$I-}
repeat
write('введите целое число i');
readln(i);
until (IOResult=0);
{$I+}
{ Этот цикл повторяется до тех пор,
пока не будет введено целое число}
end.
```

При работе с текстовым файлом необходимо помнить специальные правила чтения значений переменных:

- если вводятся числовые значения, то два числа считаются разделенными, если между ними есть хотя бы один пробел, или символ табуляции, или символ конца строки;
- при вводе строк начало текущей строки идет сразу за последним введенным до этого символом. Вводится количество символов, равное объявленной длине строки. Если при чтении встретился символ «конец строки», то работа с этой строкой заканчивается. Сам символ конца строки является разделителем и в переменную никогда не считывается;
- процедура `readln` считывает значения текущей строки файла, курсор переводится в новую строку файла, и дальнейший ввод осуществляется с нее.

В качестве примера работы с текстовыми файлами рассмотрим следующую задачу.

ЗАДАЧА 7.8. В текстовом файле **abc.txt** находятся матрицы `A(N,M)` и `B(N,M)` и их размеры. Найти матрицу C=A+B, которую дописать в файл **abc.txt**.

Сначала создадим текстовый файл **abc.txt** следующей структуры: в первой строке через пробел хранятся размеры матрицы (числа N и M), затем построчно хранятся матрицы `A` и `B`.

На рис. 7.14 приведен пример файла **abc.txt**, в котором хранятся матрицы A(4,5) и B(4,5).

Текст консольного приложения решения задачи 7.8 с комментариями приведен ниже.

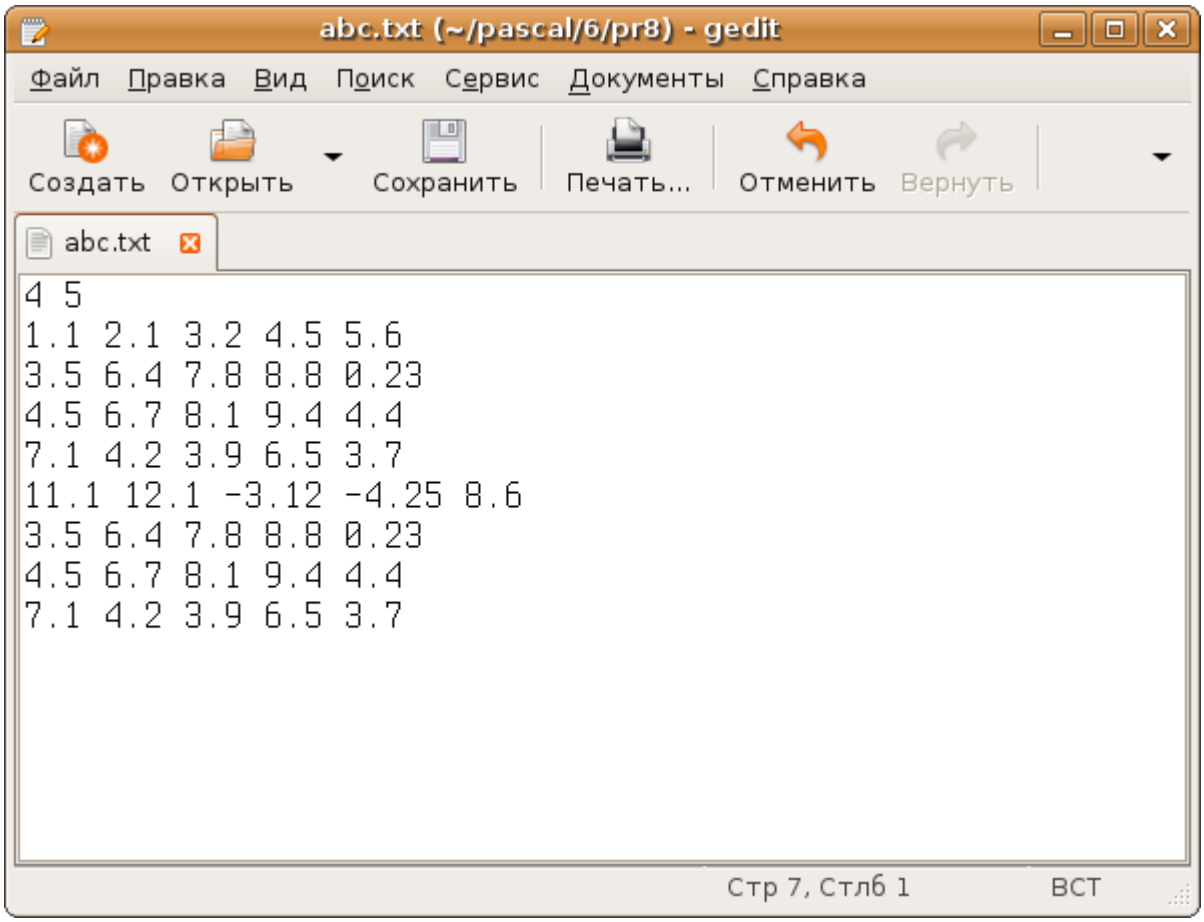

Рисунок 7.14: Содержимое файла ***abc.txt***

```
program project1;
{$mode objfpc}{$H+}
uses
 Classes, SysUtils
 { you can add units after this };
var
 f:Text;
i,j,N,M:word;
 a,b,c:array[1..1000,1..1000] of real;
begin
//Связываем файловую переменную f
//с файлом на диске.
 AssignFile(f,'abc.txt');
//Открываем файл в режиме чтения.
 Reset(f);
//Считываем из первой строки файла abc.txt
```

```
//значения N и M.
 Read(f,N,M);
//Последовательно считываем элементы
//матрицы A из файла.
 for i:=1 to N do
 for j:=1 to M do
 read(f,a[i,j]);
//Последовательно считываем элементы
//матрицы B из файла.
 for i:=1 to N do
 for j:=1 to M do
 read(f,b[i,j]);
//Формируем матрицу C=A+B.
 for i:=1 to N do
 for j:=1 to M do
 c[i,j]:=a[i,j]+b[i][j];
//Закрываем файл f.
 CloseFile(f);
//Открываем файл в режиме дозаписи.
 Append(f);
//Дозапись матрицы C в файл.
 for i:=1 to N do
 begin
     for j:=1 to M do
//Дописываем очередной элемент
//матрицы и пробел в текстовый файл.
     write(f,c[i,j]:1:2,' ');
//По окончании вывода строки матрицы,
//переходим на новую строку в текстовом файле.
     writeln(f);
 end;
//Закрываем файл.
 CloseFile(f);
end.
```

После работы программы файл **abc.txt** будет примерно таким, как показано на рис. 7.15.

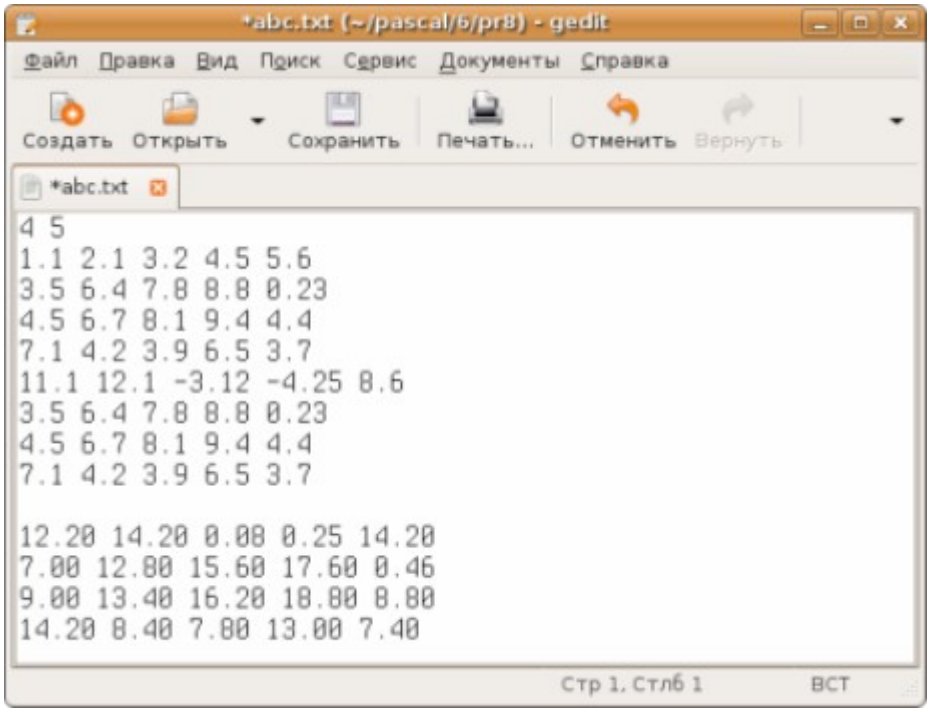

Рисунок 7.15: Файл abc.txt после дозаписи матрицы C

7.5 Задачи для самостоятельного решения

Во всех заданиях составить две программы. Первая должна формировать типизированный файл. Вторая – считать данные из этого файла, выполнить соответствующие вычисления и записать их результаты в текстовый файл.

1. Создать типизированный файл, куда записать `n` целых чисел. Из исходного файла сформировать массивы четных и нечетных чисел. Определить наибольший отрицательный компонент файла и наименьший положительный.

2. Создать типизированный файл, куда записать `n` целых чисел. На основе исходного файла создать массив утроенных четных чисел. Упорядочить его по убыванию элементов.

3. Создать типизированный файл, куда записать `n` целых чисел. Сформировать массив положительных чисел, делящихся на семь без остатка, используя элементы исходного файла. Упорядочить массив по возрастанию элементов.

4. Создать типизированный файл, куда записать `n` вещественных чисел. Из компонентов исходного файла сформировать массивы, из чисел, больших 10 и меньших двух. Вычислить количество нулевых компонентов файла.

5. Создать типизированный файл, куда записать n целых чисел. Из файла создать массив, элементы которого являются простыми числами и расположены после максимального элемента.

6. Создать типизированный файл, куда записать n целых чисел. Из файла целых чисел сформировать массив, записав в него только четные компоненты, находящиеся до минимального элемента.

7. Создать типизированный файл, куда записать n вещественных чисел. Сделать массив из элементов исходного файла, внеся в него числа, превосходящие среднее значение среди положительных значений файла.

8. Создать типизированный файл, куда записать n целых чисел. Из исходного файла сформировать массив, записав в него числа, расположенные в файле до максимального элемента и после минимального.

9. Создать типизированный файл, куда записать n целых чисел. Массив создать из исходного файла. Внести в него простые и совершенные числа, расположенные в файле между минимальным и максимальным элементами.

10. Создать типизированный файл, куда записать n целых чисел. Из исходного файла сформировать массив, в котором вначале расположить четные, а затем нечетные числа. Определить номера наибольшего нечетного и наименьшего четного компонентов.

11. Создать типизированный файл, куда записать n целых чисел. В файле поменять местами минимальный среди положительных элементов и третий по счету простой элемент.

12. Создать типизированный файл, куда записать n целых чисел. Из файла переписать все простые, расположенные после максимального элемента в новый файл.

13. Создать типизированный файл, куда записать n целых чисел. Найти среднее арифметическое среди положительных чисел, расположенных до второго простого числа.

14. Создать типизированный файл, куда записать n целых чисел. Поменять местами последнее совершенное и третье отрицательное числа в файле.

15. Создать типизированный файл, куда записать n целых чисел. Все совершенные и простые числа из исходного файла записать в массив, который упорядочить по возрастанию.

16. Создать типизированный файл, куда записать n целых чисел. Последнюю группу расположенных подряд положительных чисел из исходного файла переписать в текстовый файл.

17. Создать типизированный файл, куда записать n целых чисел. Найти в нем группу подряд расположенных простых элементов наибольшей длины.

18. Создать типизированный файл, куда записать n целых чисел. Из исходного файла сформировать массивы простых и отрицательных чисел. Определить наименьшее простое число в файле и наибольшее совершенное.

19. Создать типизированный файл, куда записать n целых чисел. Из файла создать массив, элементы которого не являются простыми числами и расположены до максимального значения файла.

20. Создать типизированный файл, куда записать n целых чисел. Из файла целых чисел сформировать массив, записав в него только кратные 5 и 7 значения, находящиеся после максимального элемента файла.

21. Создать типизированный файл, куда записать n вещественных чисел. Сделать массив из элементов исходного файла, внеся в него числа, превосходящие среднее значение среди положительных значений файла.

22. Создать типизированный файл, куда записать n вещественных чисел. Поменять местами последнее отрицательное число в файле с четвертым по счету числом.

23. Создать типизированный файл, куда записать n вещественных чисел. Найти сумму третьей группы подряд расположенных отрицательных элементов.

24. Создать типизированный файл, куда записать n целых чисел. Удалить из него четвертую группу, состоящую из подряд расположенных простых чисел.

25. Создать типизированный файл, куда записать n целых чисел. Найти разность между суммой простых чисел, находящихся в файле, и максимальным отрицательным значением файла.