6 Обработка матриц во Free Pascal

Матрица – это двумерный массив, каждый элемент которого имеет два индекса: номер строки и номер столбца.

Объявить двумерный массив (матрицу) можно так:

```
имя: array [индекс1_нач .. индекс1_кон,
          индекс2_нач .. индекс2_кон] of тип;
```

где тип определяет тип элементов массива, имя — имя матрицы, индекс1_нач .. индекс1_кон — диапазон изменения номеров строк, индекс2_нач .. индекс2_кон — диапазон изменения номеров столбцов матрицы.

Например,

```
var h: array [0..11,1..10] of real;
```

Описана матрица вещественных чисел h, состоящая из двенадцати строк и десяти столбцов (строки нумеруются от 0 до 11, столбцы — от 1 до 10).

Существует ещё один способ описать матрицы, для этого надо создать новый тип данных:

```
type
новый_тип=array [индекс1_нач .. индекс1_кон]
                                        of тип;
var
имя: array [индекс2_нач .. индекс2_кон]
                                    of новый_тип;
```

или

```
type
новый_тип=array [список_диапазонов] of тип;
var
    имя: новый_тип;
```

Например:

```
type
 massiv=array [1..30] of integer;
 matrica=array [0..15,0..13] of real;
var
 a, b: array [1..10] of massiv;
 c:matrica;
```

В данном случае в матрицах `a` и `b` есть 10 строк и 30 столбцов, а `c` – матрица, в которой есть шестнадцать строк и четырнадцать столбцов.

Для обращения к элементу матрицы необходимо указать ее имя и в квадратных скобках через запятую номер строки и номер столбца:

```
имя[номер_строки, номер_столбца]
```

или так

```
имя[номер_строки] [номер_столбца]
```

Например, `h[2,4]`[67] – элемент матрицы `h`, находящийся в строке под номером два и столбце под номером четыре.

Для обработки всех элементов матрицы необходимо использовать два цикла. Если обрабатываем матрицу построчно, то во внешнем цикле последовательно перебираем строки от первой до последней, затем во внутреннем — перебираем все (первый, второй, третий и т. д.) элементы текущей строки. При обработке элементов матрицы по столбцам, внешний цикл будет по столбцам, внутренний — по строкам.

На рис. 6.1 представлена блок-схема алгоритма обработки матрицы по строкам, на рис. 6.2 — по столбцам. Здесь `i` — номер строки, `j` — номер столбца, `N` — количество строк, `M` — количество столбцов матрицы `A`.

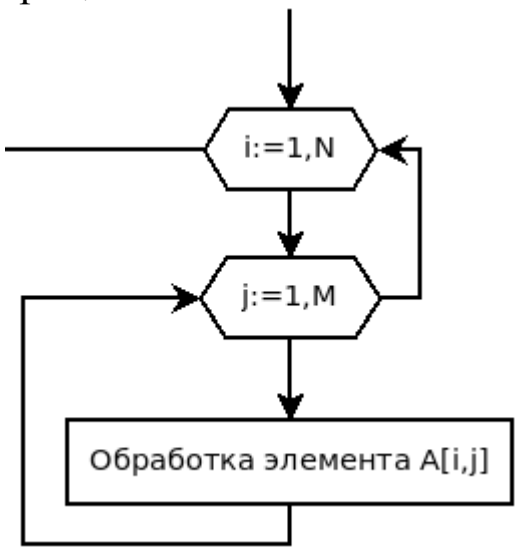

Рисунок 6.1: Построчная обработка матрицы

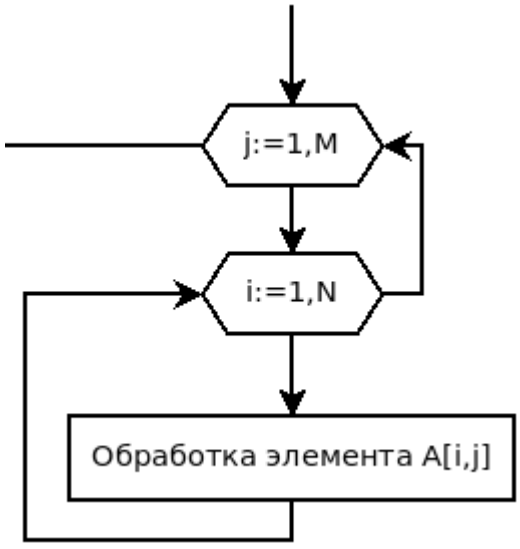

Рисунок 6.2: Алгоритм обработки матрицы по столбцам

67 или h[2][4]

6.1 Ввод-вывод матриц

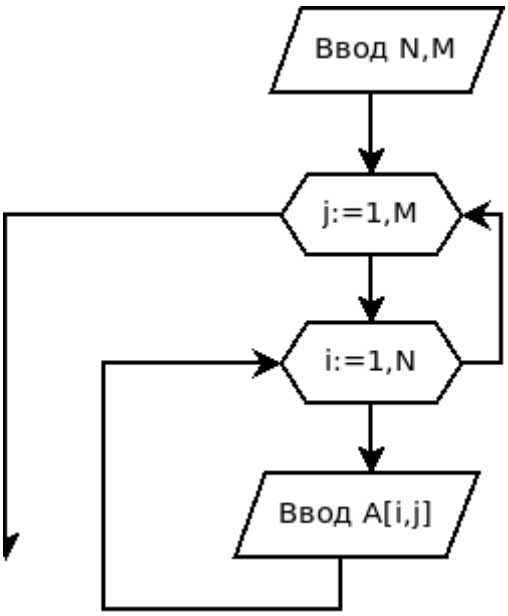

Рисунок 6.3: Блок-схема ввода элементов матрицы

Матрицы, как и массивы, нужно вводить (выводить) поэлементно. Вначале следует ввести размеры матрицы, а затем уже в двойном цикле вводить элементы. Блок-схема ввода элементов матрицы изображена на рис. 6.3.

Вывод можно осуществлять по строкам или по столбцам, но лучше, если элементы располагаются построчно, например,

```
 2   3  13  35
 5  26  76  37
52  61  79  17
```

Алгоритм построчного вывода элементов матрицы приведен на рис. 6.4.

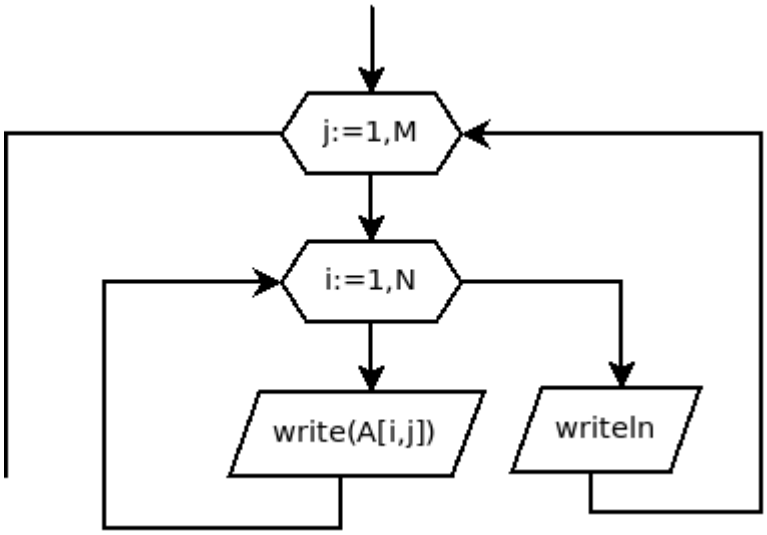

Рисунок 6.4: Построчный вывод матрицы

Об описании матриц на языке Free Pascal было рассказано в разделе 5.2 пятой главы, обращение к элементу $A_{i,j}$ матрицы можно осуществить с помощью конструкции `A[i,j]` или `A[i][j]`.

Рассмотрим реализацию ввода-вывода матриц в консольных приложениях. Для организации построчного ввода матрицы в двойном цикле по строкам и столбцам можно использовать оператор `read`.

```
for i:=1 to N do
for j:=1 to m do
    read(A[i,j]);
```

В этом случае элементы каждой строки матрицы можно разделять символами пробела или табуляции и только в конце строки нажимать **Enter**.

ЗАДАЧА 6.1. Написать консольное приложение ввода матрицы вещественных чисел и вывода ее на экран дисплея.

Ниже приведен пример программы ввода - вывода матрицы.

```
var
a: array [1..20,1..20] of real;
i,j,n,m: integer;
begin
{Ввод размеров матрицы}
writeln('Введите количество строк и столбцов
                                        матрицы A');
readln(n,m);
{ Ввод элементов матрицы. }
writeln('Введите матрицу');
for i:=1 to n do
for j:=1 to m do
 read(A[i,j]);
{ Вывод элементов матрицы. }
writeln ('матрица A ' );
for i:=1 to n do
begin
 for j:=1 to m do
     write(a[i,j]:8:3, ' ');   {Печать строки.}
 writeln    {Переход на новую строку.}
end;
```

На рис. 6.5 представлены результаты работы программы.

Ввод матрицы можно также организовать с помощью следующего цикла. Авторы предлагают читателю самостоятельно разобраться, в чем будет отличие ввода матрицы в этом случае.

```
for i:=1 to N do
 for j:=1 to m do
 begin
     write('A(',i,',',j,')=');
     readln(A[i,j])
 end;
```

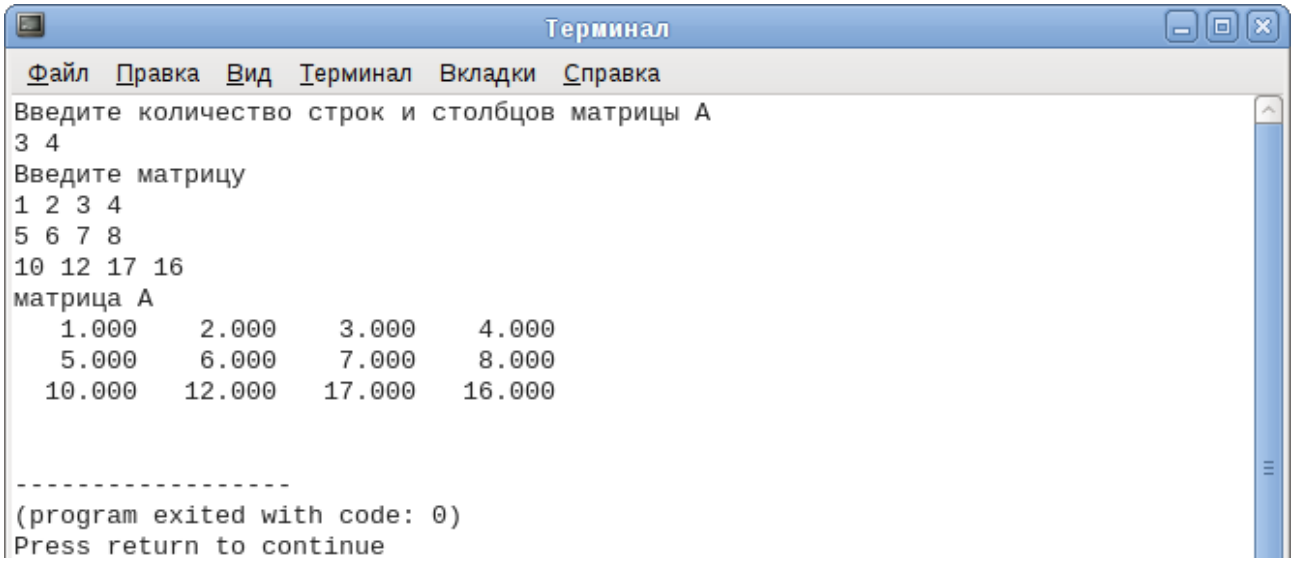

Рисунок 6.5: Результаты работы программы решения задачи 6.1

Для ввода-вывода матриц можно использовать компонент типа `TStringGrid`, с которым мы познакомились в пятой главе.

В качестве примера рассмотрим следующую задачу.

ЗАДАЧА 6.2. Составить программу транспонирования[68] матрицы A.

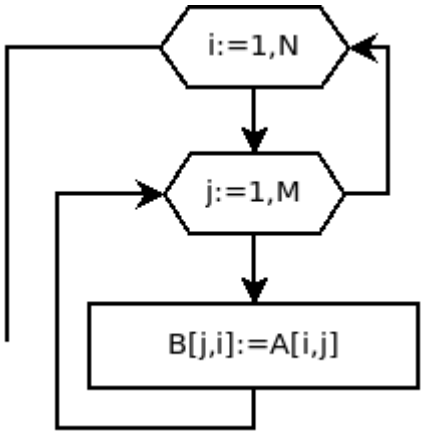

Рисунок 6.6: Блок-схема транспонирования матрицы A

Блок-схема транспонирования матрицы приведена на рис. 6.6. При транспонировании матрицы A(N,M) получается матрица B(M,N).

Рассмотрим частный случай транспонирования матрицы фиксированного размера `A(4,3)`. На форме разместим метки `Label1` и `Label2` со свойствами `Caption – Заданная матрица A` и `Транспонированная матрица B`, два компонента типа `TStringGrid`, изменив их свойства, так как показано в табл. 6.1, и кнопку `Транспонирование матрицы`.

Таблица 6.1: Свойства компонентов `StringGrid1`, `StringGrid2`.

Свойство	StringGrid1	StringGrid2	Описание свойства
Top	30	30	Расстояние от верхней границы таблицы до верхнего края формы

68 Транспонированная матрица — матрица, полученная из исходной матрицы A(N,M) заменой строк на столбцы.

Свойство	StringGrid1	StringGrid2	Описание свойства
`Left`	15	240	Расстояние от левой границы таблицы до левого края формы
`Height`	130	130	Высота таблицы
`Width`	200	200	Ширина таблицы
`ColCount`	4	5	Количество столбцов
`RowCount`	5	4	Количество строк
`DefaultColWidth`	30	30	Ширина столбца
`DefaultRowHeight`	20	20	Высота строки
`Options.goEditing`	true	false	Возможность редактирования таблицы

Окно формы приложения представлено на рис. 6.7.

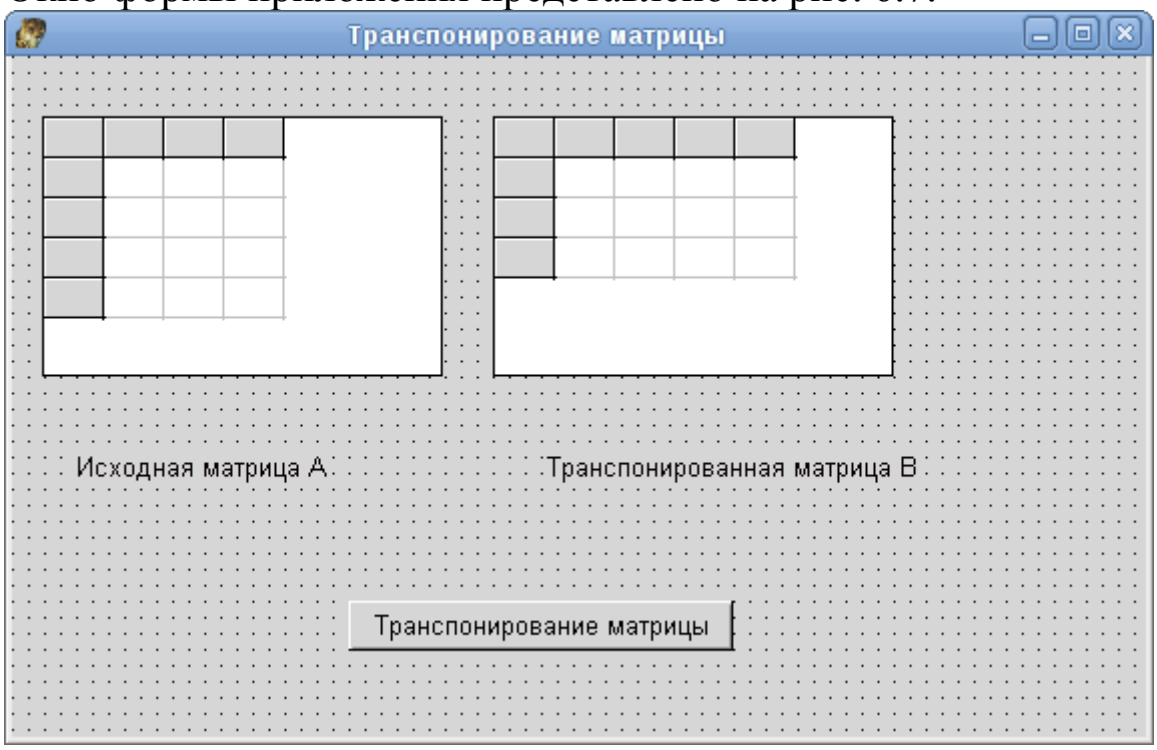

Рисунок 6.7: Форма приложения транспонирования матрицы

Ниже приведен текст подпрограммы с комментариями, которая будет выполняться, если пользователь щелкнет по кнопке `Транспонирование матрицы`.

```
procedure TForm1.Button1Click(Sender: TObject);
const n=4;m=3;  //Размерность матрицы A(n,m).
var
i,j:byte;               //Индексы матрицы:
//i – строки, j – столбцы.
//Исходная матрица.
A:array [1..n,1..m] of integer;
//Транспонированная матрица.
B:array [1..m,1..n] of integer;
Begin
//Исходные данные считываются
//из ячеек таблицы на форме,  и их значения
//записываются в двумерный массив A.
//Цикл по номерам строк.
for i:=1 to n do
//Цикл по номерам столбцов.
for j:=1 to m do
//Считывание элементов матрицы
//A из компонента StringGrid1.
 A[i,j]:=StrToInt(StringGrid1.Cells[j,i]);
//Формирование транспонированной матрицы B,
//см. блок-схему на рис. 6.6.
//Цикл по номерам строк.
for i:=1 to n do
//Цикл по номерам столбцов.
for j:=1 to m do
B[j,i]:=A[i,j];
//Элементы матрицы B выводятся
//в ячейки таблицы на форме.
//Цикл по номерам строк.
for i:=1 to n do
//Цикл по номерам столбцов.
for j:=1 to m do
//Обращение к элементам матрицы
//происходит по столбцам.
StringGrid2.Cells[i,j]:=IntToStr(B[j,i]);
end;
```

Результаты работы программы представлены на рис. 6.8.

Рисунок 6.8: Результаты работы программы транспонирования матрицы A(3,4)

Для демонстрации ввода-вывода матриц с помощью компонента типа `TStringGrid` мы рассмотрели работу с матрицами фиксированного размера `A(4,3)` и `B(3,4)`. Теперь рассмотрим общий случай решения задачи транспонирования матрицы `A(N,M)`.

Расположим на форме следующие компоненты:

- метку `label1` с надписью «`Введите размерность матрицы`»;
- метку `label2` с надписью «`N=`»;
- метку `label3` с надписью «`M=`»;
- метку `label4` с надписью «`Исходная матрица A`»;
- метку `label5` с надписью «`Преобразованная матрица B`»;
- поле ввода `Edit1` для ввода числа `N`;
- поле ввода `Edit2` для ввода числа `M`;
- компонент `StringGrid1` для ввода исходной матрицы `A`;
- компонент `StringGrid2` для хранения транспонированной матрицы `B`;
- кнопку `Button1` с надписью «`Ввод`» для ввода размеров матрицы `A`;

- кнопку `Button2` с надписью «`Очистить`» для очистки содержимого матриц;
- кнопку `Button3` с надписью «`Транспонирование`» для решения задачи транспонирования матрицы `A`;
- кнопку `Button4` с надписью «`Выход из программы`» для завершения работы программы.

Можно разместить компоненты на форме таким образом, как показано на рис. 6.9.

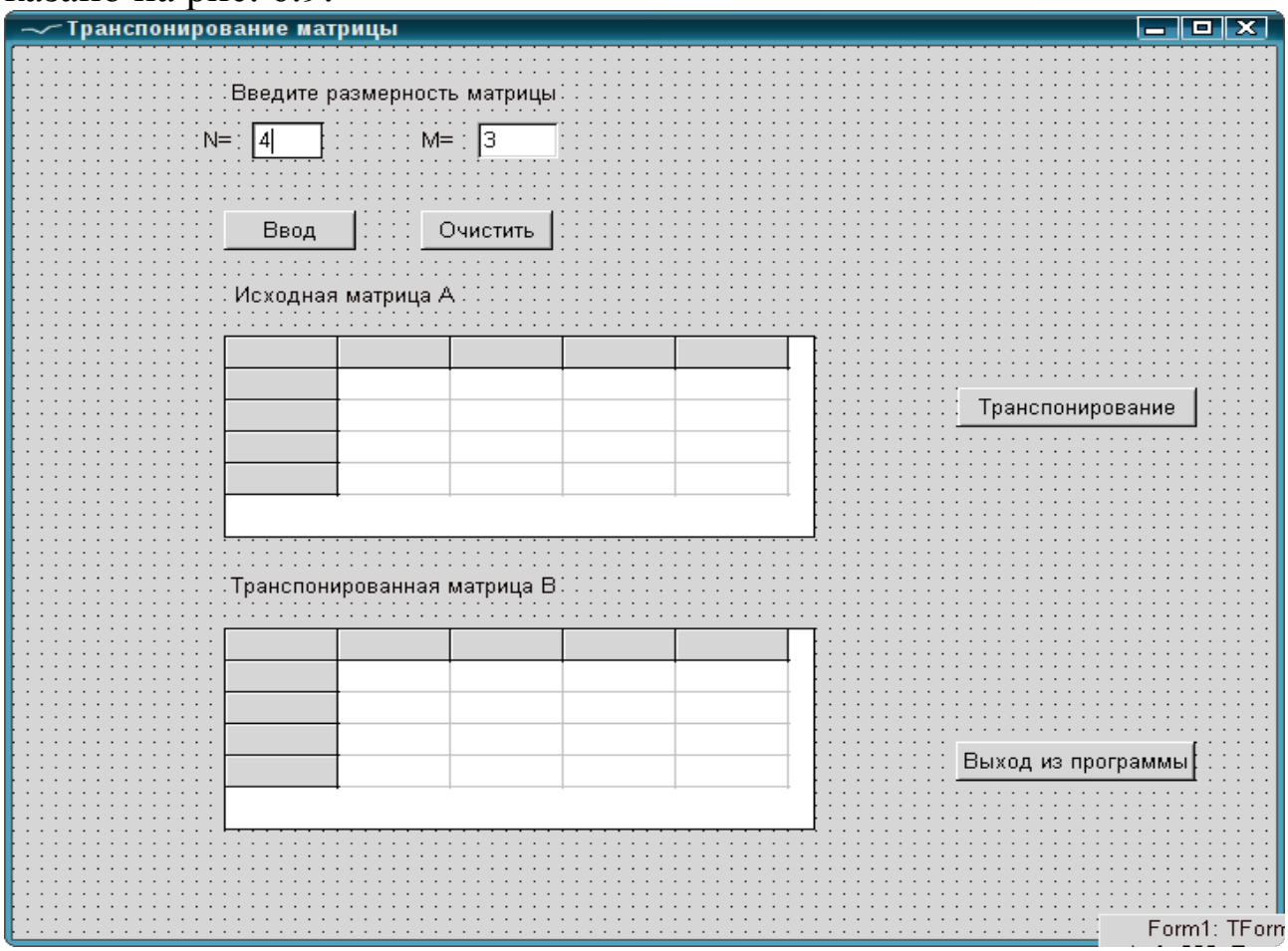

Рисунок 6.9: Окно формы решения задачи транспонирования матрицы A(N,M)

Установим свойство видимости `Visible` в `False` у компонентов метки `label4` и `label5`, `StringGrid1`, `StringGrid2`, кнопки `Button2` и `Button3`. После этого при старте программы будут видны только компоненты, отвечающие за ввод размеров матрицы, и кнопка `Выход из программы` (см. рис. 6.10)[69]. Матрицы `A` и `B`, их размеры `N`, `M` объявим глобально.

69 Свойству Caption формы присвоено значение **Транспонирование матрицы**.

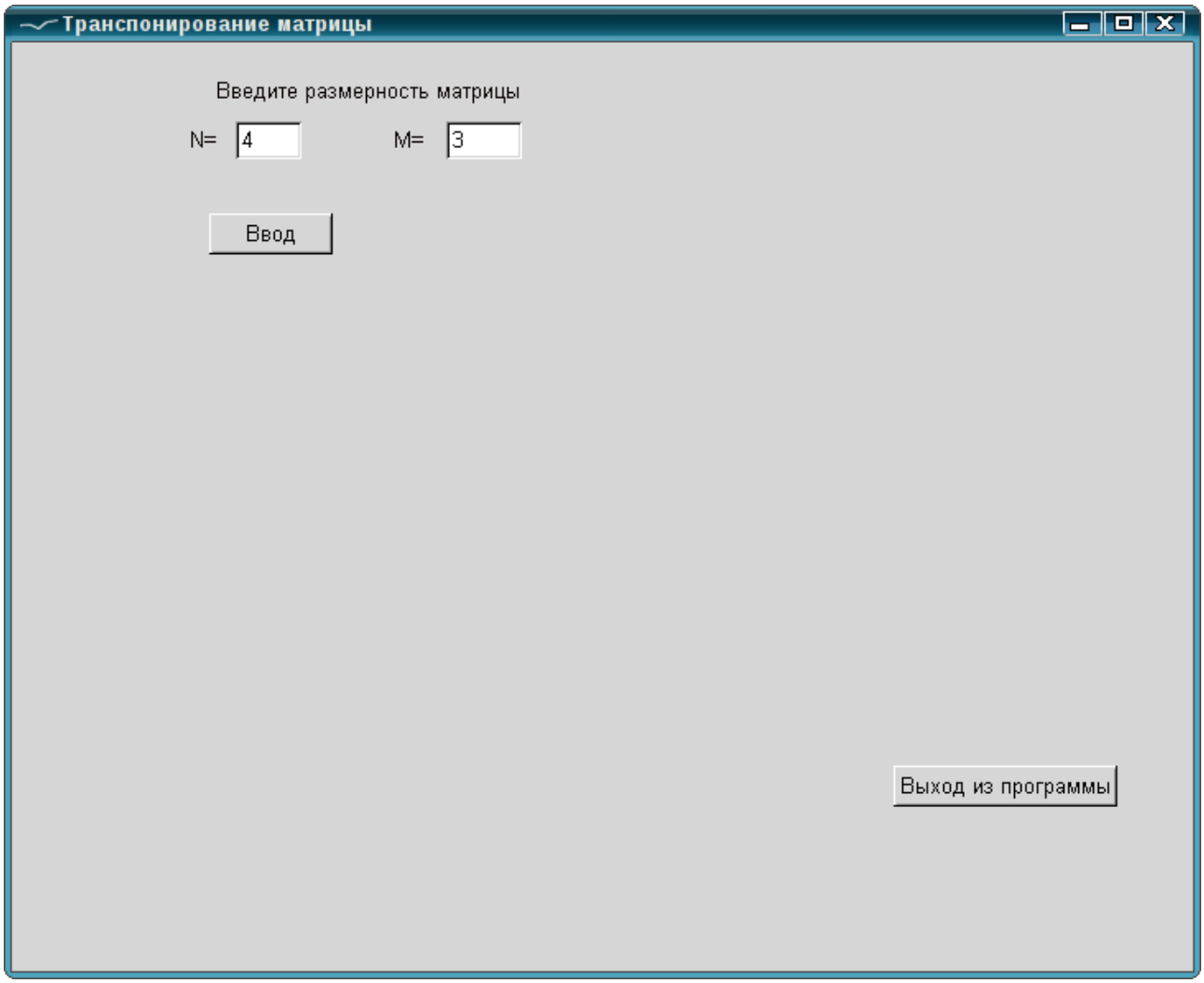

Рисунок 6.10: Стартовое окно программы транспонирования матрицы A(N,M)

```
type
     { TForm1 }
{Описание формы}
 TForm1 = class(TForm)
     Button1: TButton;
     Button2: TButton;
     Button3: TButton;
     Button4: TButton;
     Edit1: TEdit;
     Edit2: TEdit;
     Label1: TLabel;
     Label2: TLabel;
     Label3: TLabel;
     Label4: TLabel;
```

```
    Label5: TLabel;
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
 private
 { private declarations }
 public
 { public declarations }
end;
var
{Матрицы A,B}
    A,B:array[1..25,1..25] of integer;
{и их размеры}
    N,M:integer;
    Form1: TForm1;
```

Обработчик кнопки `Выход из программы` стандартен и представлен ниже.

```
procedure TForm1.Button4Click(Sender: TObject);
begin
 Close;
end;
```

Теперь напишем обработчик кнопки `Ввод`, который должен вводить и проверять корректность введения размеров матрицы, устанавливать свойства компонентов `StringGrid1` и `StringGrid2` (количество строк и столбцов), делать видимым компонент `StringGrid1`, кнопку `Транспонирование`, невидимыми компоненты, отвечающие за ввод размеров матрицы (метки `label1`, `label2`, `label3`, поля ввода `Edit1` и `Edit2`, кнопку `Ввод`).

```
procedure TForm1.Button1Click(Sender: TObject);
var i:byte;kod_n,kod_m,kod:integer;
begin
//Ввод размерности матрицы.
//Символьная информация преобразовывается
//в числовую и записывается в
 Val(Edit1.Text,N,kod_m); //переменную M
 Val(Edit2.Text,M,kod_n); //и переменную N.
//Если преобразование прошло успешно и
//введенные размеры удовлетворяют описанию
```

```
   //матриц A и B,
    if  (kod_n=0)  and  (kod_m=0)  and  (N>0)  and
(N<26) and (M>0) and (M< 26) then
   //то
    begin
   //визуализируется первая матрица,
       StringGrid1.Visible:=true;
       //соответствующая ей надпись,
       Label4.Visible:=true;
       Button2.Visible:=true;//кнопки Очистить
       //и транспонирование.
       Button3.Visible:=true;
       with StringGrid1 do
       begin
   //Определяем число строк (RowCount)
   //и столбцов(ColCount) в
   //компоненте StringGrid1.
           ColCount:=M+1;
           RowCount:=N+1;
   //и нумеруются строки и столбцы матрицы
           for i:=1 to RowCount-1 do
               Cells[0,i]:=IntToStr(i);
           for i:=1 to ColCount-1 do
               Cells[i,0]:=IntToStr(i);
       end;
   StringGrid2.ColCount:=N+1;
   StringGrid2.RowCount:=M+1;
   end
    else
    begin
   //При некорректном вводе выдается
   //соответствующее сообщение.
    MessageDlg('Размеры матрицы введены неверно!',
                          MtInformation,[mbOk],0);
   //Устанавливаются стартовые параметры
   //в поля ввода.
    Edit1.Text:='4';
    Edit2.Text:='3';
```

```
end;
end;
```

Теперь напишем обработчик кнопки Транспонирование. При щелчке по этой кнопке становится видимым компонент StrigGrid2, предназначенный для хранения транспонированной матрицы B, соответствующая ему надпись (label5), формируется матрица B. Матрица B выводится в компонент StringGrid2. Кнопка Ввод становится невидимой. Текст обработчика приведен ниже.

```
procedure TForm1.Button2Click(Sender: TObject);
var i,j:integer;
begin
//визуализируется вторая матрица,
//соответствующая ей надпись,
StringGrid2.Visible:=true;
label5.Visible:=true;
for i:=1 to N do       //Цикл по номерам строк.
for j:=1 to M do   //Цикл по номерам столбцов.
//Считывание элементов матрицы A
//из компонента StringGrid1.
A[i,j]:=StrToInt(StringGrid1.Cells[j,i]);
with StringGrid2 do
begin
for i:=1 to RowCount-1 do  //нумеруются строки
Cells[0,i]:=IntToStr(i);
//и столбцы компонента StringGrid2, в
//котором отображается матрица B.
for i:=1 to ColCount-1 do
Cells[i,0]:=IntToStr(i);
end;
//Формирование транспонированной матрицы B.
for i:=1 to N do       //Цикл по номерам строк.
for j:=1 to M do   //Цикл по номерам столбцов.
B[j,i]:=A[i,j];
//Элементы матрицы B выводятся
//в ячейки таблицы на форме.
for i:=1 to n do       //Цикл по номерам строк.
for j:=1 to m do   //Цикл по номерам столбцов.
//Обращение к элементам матрицы
```

```
//происходит по столбцам.
StringGrid2.Cells[i,j]:=IntToStr(B[j,i]);
Buuton1.Visible:=False; end;
```

Осталось написать обработчик события, которое произойдет при нажатии на кнопку Очистить. При щелчке по этой кнопке должны выполниться следующие действия:

- очистка содержимого компонентов StringGrid1, StringGrid2;
- компоненты StringGrid1, StringGrid2 и соответствующие им метки labe4 и label5, а также кнопки Транспонировать и Очистить должны стать невидимыми;
- становится видимой кнопка Ввод;
- в поля ввода записываются начальные значения размеров матрицы (N=4, M=3).

Текст обработчика кнопки Очистить с комментариями приведен ниже:

```
procedure TForm1.Button3Click(Sender: TObject);
var i,j:integer;
begin
//Очистка компонента StringGrid1.
 with StringGrid1 do
     for i:=1 to RowCount-1 do
     for j:=1 to ColCount-1 do
         Cells[j,i]:='';
//Очистка компонента StringGrid2.
 with StringGrid2 do
     for i:=1 to RowCount-1 do
     for j:=1 to ColCount-1 do
     Cells[j,i]:='';
//Делаем невидимыми компоненты
//StringGrid1, StringGrid2,
//labe4, label5.
 StringGrid1.Visible:=False;
 StringGrid2.Visible:=False;
 label4.Visible:=False;
 label5.Visible:=False;
//Делаем невидимыми кнопки «Транспонировать»
```

```
//и «Очистить».
 Button2.Visible:=False;
 Button3.Visible:=False;
//Делаем видимой кнопку «Ввод».
 Button1.Visible:=True;
//Запись начальных значений
//размеров матрицы (N=4, M=3).
 Edit1.Text:='4';
 Edit2.Text:='3';
end;
```

Получили работающую программу для транспонирования матрицы. На рис. 6.11 представлены результаты транспонирования матрицы `A(2,4)`.

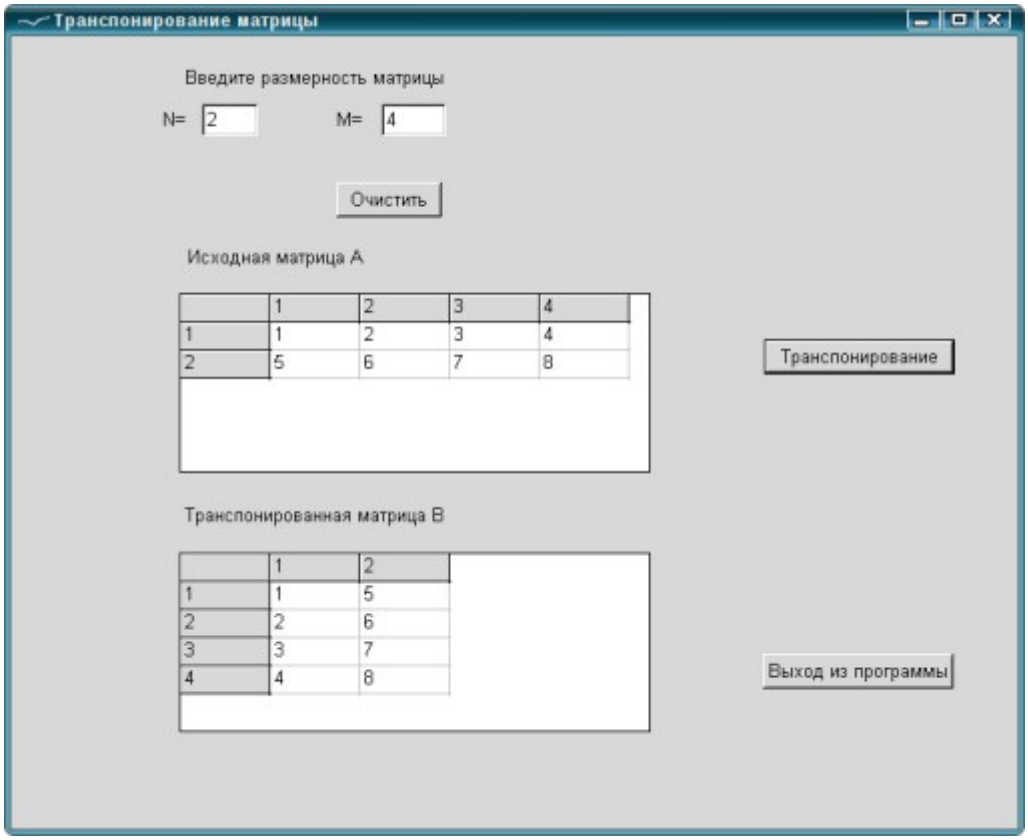

Рисунок 6.11: Транспонирование матрицы A(2,4)

Обратите внимание на использование оператора присоединения

```
with имя_компонента do оператор;
```

который упрощает доступ к свойствам компонента. Внутри оператора `With` имя компонента для обращения к его свойствам можно не использовать.

Например, для очистки элементов матрицы `A` вместо операторов

```
for i:=1 to StringGrid1.RowCount-1 do
    for j:=1 to StringGrid1.ColCount-1 do
        StringGrid1.Cells[j,i]:='';
```

был использован оператор

```
with StringGrid1 do
 for i:=1 to RowCount-1 do
    for j:=1 to ColCount-1 do
                Cells[j,i]:='';
```

Рассмотрим несколько задач обработки матриц. Для их решения напомним читателю некоторые свойства матриц (рис. 6.12):

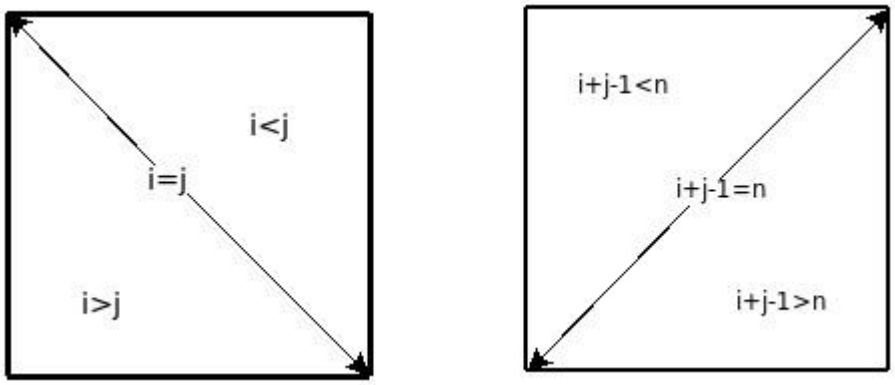

Рисунок 6.12: Свойства элементов матрицы

- если номер строки элемента совпадает с номером столбца (`i=j`), это означает что элемент лежит на главной диагонали матрицы;
- если номер строки превышает номер столбца (`i>j`), то элемент находится ниже главной диагонали;
- если номер столбца больше номера строки (`i<j`), то элемент находится выше главной диагонали;
- элемент лежит на побочной диагонали, если его индексы удовлетворяют равенству `i+j-1=n`;
- неравенство `i+j-1<n` характерно для элемента, находящегося выше побочной диагонали;
- соответственно, элементу, лежащему ниже побочной диагонали, соответствует выражение `i+j-1>n`.

6.2 Алгоритмы и программы работы с матрицами

Рассмотри несколько примеров решения задач обработки матриц.

ЗАДАЧА 6.3. Найти сумму элементов матрицы, лежащих выше главной диагонали (см. рис. 6.13).

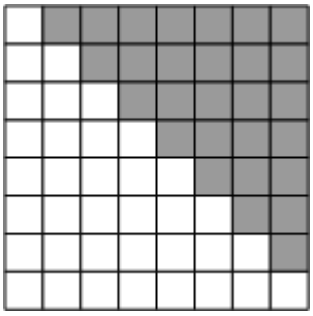

Рисунок 6.13: Иллюстрация к задаче 6.3

Рассмотрим два алгоритма решения данной задачи. *Первый алгоритм* (см. рис. 6.14) построен следующим образом: вначале переменная S для накапливания суммы обнуляется (`S:=0`). Затем с помощью двух циклов (первый по строкам, второй по столбцам) перебираются все элементы матрицы, но накапливание суммы происходит только в том случае, если этот элемент находится выше главной диагонали (если выполняется свойство `i<j`).

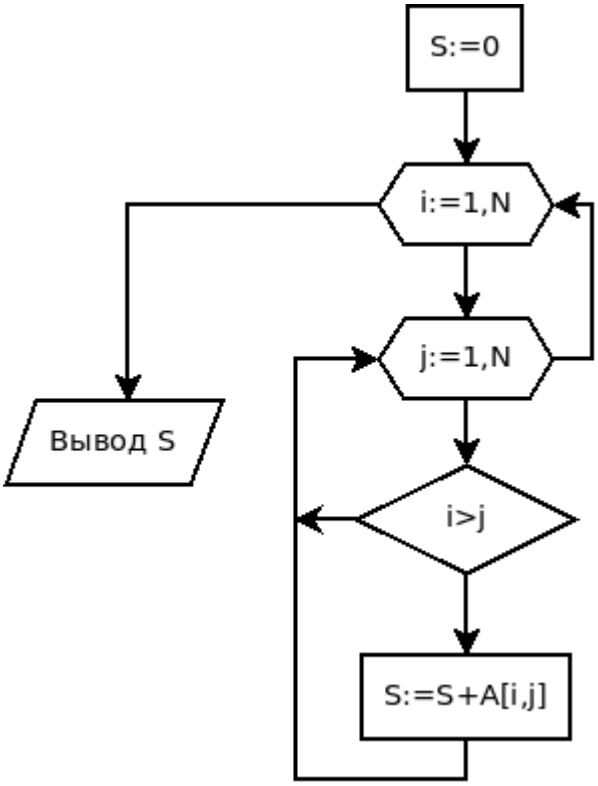

Рисунок 6.14: Блок-схема задачи 6.3 (алгоритм 1)

Текст консольного приложения с комментариями:

```
var
a:array [1..15,1..10]
                  of real;
i,j,n,m: integer;
s: real;

begin
write('Введите размеры');
writeln('матрицы');
write('n - количество');
writeln('строк: ');
readln (n);
write('m — количество');
writeln('столбцов: ');
readln (m);
writeln('Матрица A:');

for i:=1 to n do
    for j:=1 to m do
        read(a[i,j]);
s:=0;
for i:=1 to n do
    for j:=1 to m do
   {Если элемент лежит выше главной диагонали,то}
    if j>i then
        s:=s+a[i,j];{ накапливаем сумму. }
```

```
writeln('матрица A');
for i:=1 to n do
begin
 for j:=1 to m do
{Здесь формат важен,
особенно общая ширина поля!}
    write(a[i,j]:8:3,' ');
 writeln
end;
writeln('сумма элементов  матрицы', s:8:3);
end.
```

Результаты работы программы представлены на рис. 6.15.

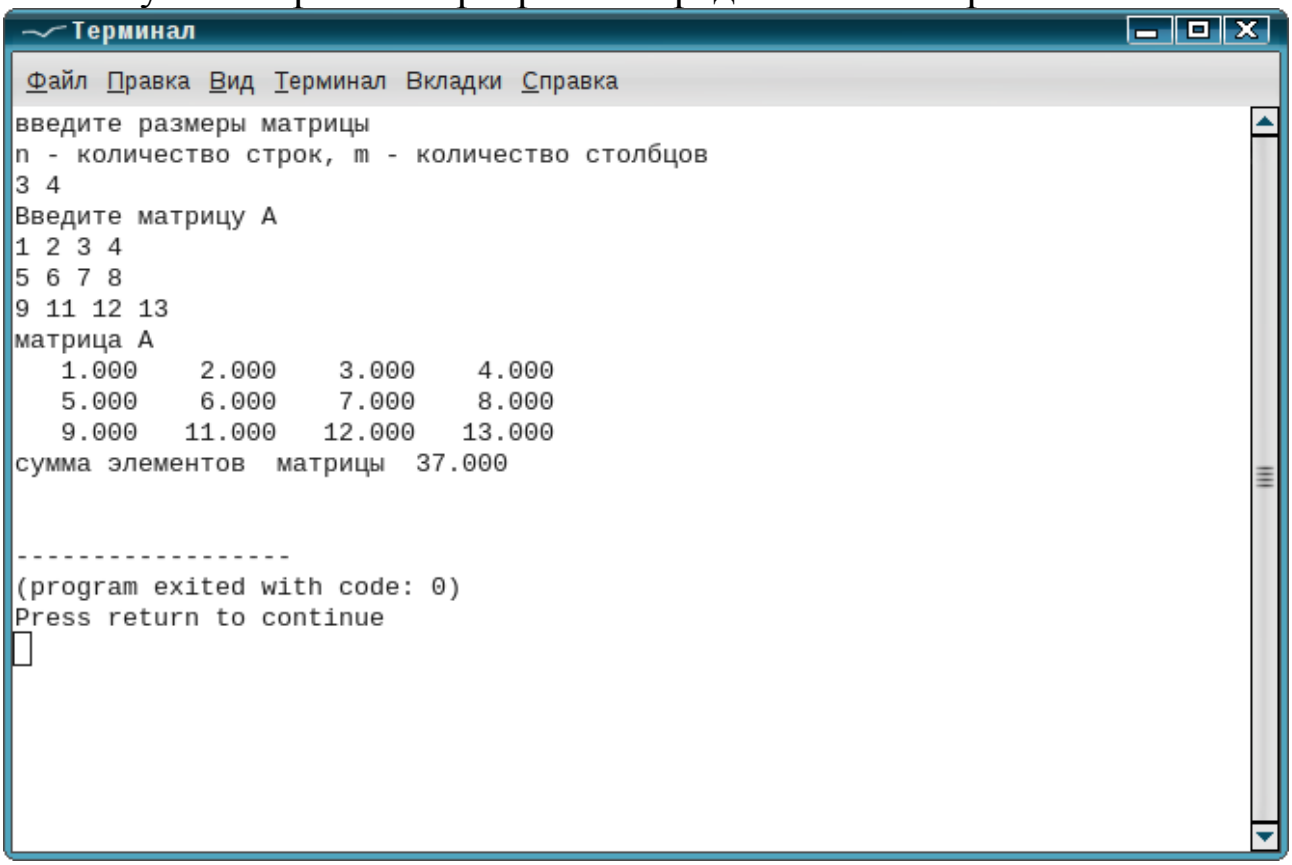

Рисунок 6.15: Результаты работы программы решения задачи 6.3

Второй алгоритм решения этой задачи представлен на рис. 6.16.

В нем проверка условия `i<j` не выполняется, но, тем не менее, в нем так же суммируются элементы матрицы, находящиеся выше главной диагонали. Для пояснения функционирования алгоритма обратимся к рисунку 6.13.

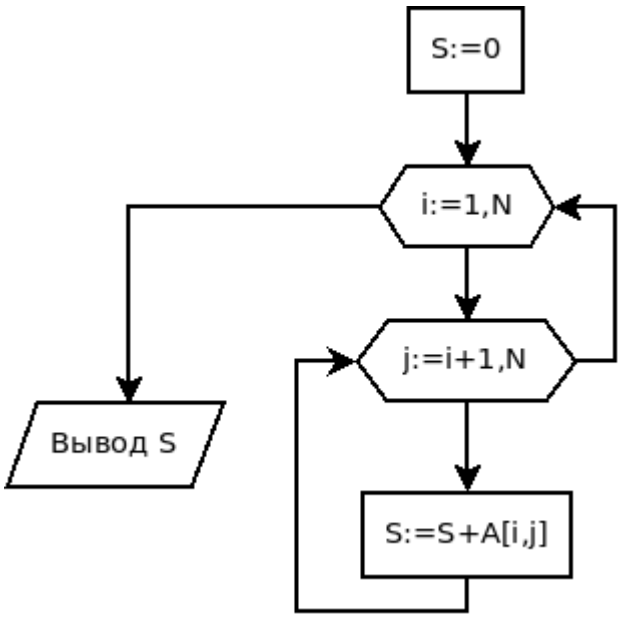

Рисунок 6.16: Блок-схема задачи 6.3 (алгоритм 2)

В первой строке заданной матрицы необходимо сложить все элементы, начиная со второго. Во второй – все, начиная с третьего, в `i`–й строке процесс суммирования начнется с (`i+1`)–го элемента и так далее. Таким образом, первый цикл работает от `1` до `N`, а второй от `i+1` до `M`.

Предлагаем читателю самостоятельно составить программу, соответствующую описанному алгоритму.

ЗАДАЧА 6.4. Вычислить количество положительных элементов квадратной матрицы A, расположенных по ее периметру и на диагоналях.

В квадратной матрице число строк равно числу столбцов. Прежде чем приступить к решению задачи, рассмотрим рисунок 6.17, на котором изображена схема диагоналей квадратных матриц различной размерности.

Из рисунка видно, что нет необходимости рассматривать все элементы матрицы. Достаточно рассмотреть элементы, расположенные в первой и последней строках, в первом и последнем столбцах, а также на диагоналях квадратной матрицы. Все эти элементы отмечены на рис. 6.17, причем черным цветом выделены элементы, которые принадлежат строкам, столбцам и диагоналям. Например, элемент $A_{1,1}$ принадлежит первой строке, первому столбцу и главной диагонали матрицы, элемент $A_{N,N}$ находится в последней строке, последнем столбце и принадлежит главной диагонали. Кроме того, если `N` – число нечетное (на рисунке 6.17 эта матрица расположена слева), то существует элемент с индексом (`N div 2 +1, N div 2 +1`), который находится на пересечении главной и побочной диагоналей. При четном значении `N` (матрица справа на рис. 6.17) диагонали не пересекаются.

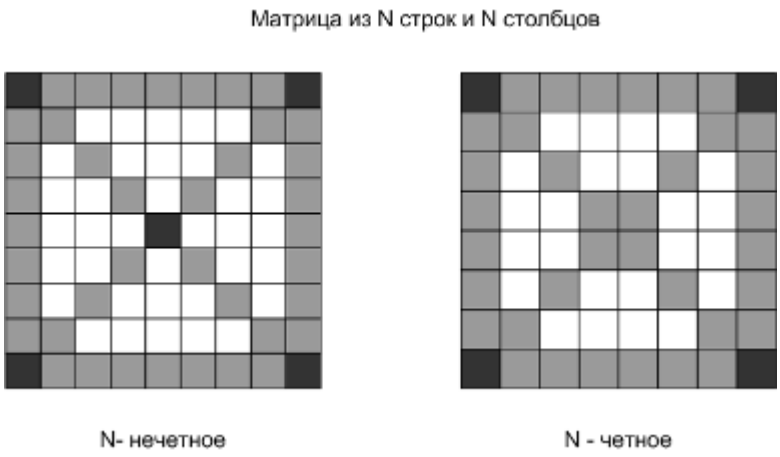

Рисунок 6.17: Рисунок к условию задачи 6.4

Рассмотрим алгоритм решения задачи. Для обращения к элементам главной диагонали вспомним, что номера строк этих элементов всегда равны номерам столбцов. Поэтому, если параметр `i` изменяется циклически от `1` до `N`, то $A_{i,i}$ — элементы расположенные на главной диагонали. Воспользовавшись свойством, характерным для элементов побочной диагонали, получим:

`i+j-1=N → j=N-i+1,`

следовательно, для строк `i=1,2,…,N` элемент $A_{i,N-i+1}$ — элемент побочной диагонали. Элементы, находящиеся по периметру матрицы, записываются следующим образом: $A_{1,i}$ — элементы, расположенные в первой строке (`i=1,2,…,N`), $A_{N,i}$ — элементы, расположенные в последней строке (`i=1,2,…,N`) и соответственно $A_{i,1}$ — элементы, расположенные в первом столбце (`i=1,2,…,N`), $A_{i,N}$ — в последнем столбце (`i=1,2,…,N`).

Алгоритм обработки построим следующим образом, сначала обработаем элементы, расположенные на диагоналях квадратной матрицы. Для этого необходимо в каждой строке (`i=1,2,…,N`) проверять знак элементов $A_{i,i}$ и $A_{i,N-i+1}$.

```
for i:=1 to N do
 begin
     if (a[i,i]>0) then     k:=k+1;
     if a[i,N-i+1]>0 then   k:=k+1;
 end;
```

Так как при проверке диагональных элементов матрицы угловые элементы были учтены, то при обработке элементов, расположенных по периметру матрицы, угловые элементы учитывать не нужно.

Поэтому надо перебрать элементы со второго до предпоследнего в первой и последней строках, в первом и последнем столбцах.

```
for i:=2 to N-1 do
 begin
{ Если элемент находится в первой строке. }
     if (a[1,i]>0) then        k:=k+1;
{ Если элемент находится в последней строке. }
     if (a[N,i]>0) then        k:=k+1;
{ Если элемент находится в первом столбце. }
     if (a[i,1]>0) then        k:=k+1;
{ Если элемент находится в последнем столбце. }
     if (a[i,N]>0) then        k:=k+1;
end;
```

Затем надо проверить, не был ли элемент, находящийся на пересечении диагоналей, подсчитан дважды. Это могло произойти только в том случае, если `N` – нечетно и элемент, расположенный на пересечении диагоналей[70], положителен.

```
if (N mod 2 <>0) and (a[n div 2+1,n div 2+1]>0)
then k:=k-1;
```

Ниже приведен полный текст консольного приложения решения задачи 6.4 с комментариями. На рис. 6.18 представлены результаты работы программы решения задачи 6.4.

```
var
     a:array [1..10,1..10] of integer;
     i,j,N,k:integer;
begin
     write('N=');
     readln(N);
//Ввод исходной матрицы.
 writeln('Введите матрицу A');
 for i:=1 to N do
     for j:=1 to N do
         read(a[i,j]);
//Вывод исходной матрицы.
 writeln('Была введена матрица A:');
```

70 На пересечении диагоналей находится элемент с индексами (`N div 2 +1, N div 2 +1`)

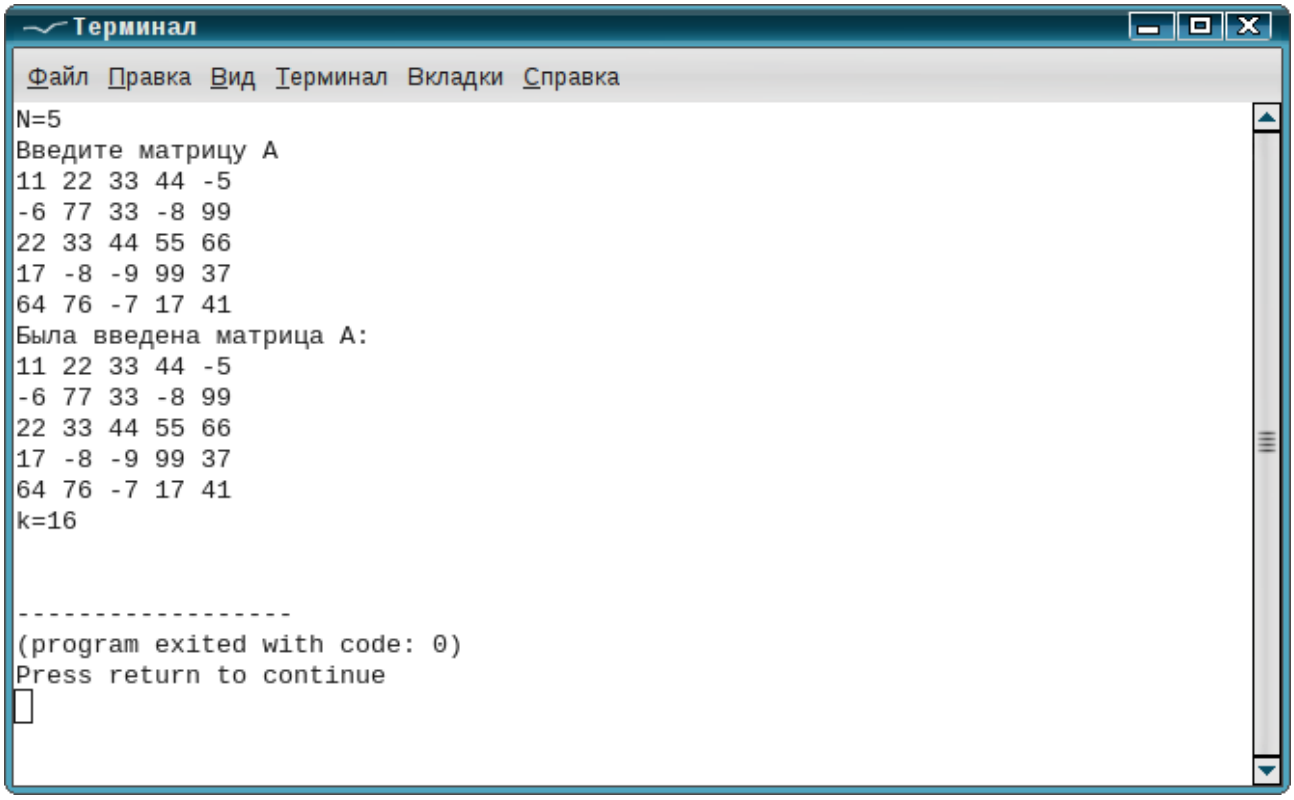

Рисунок 6.18: Результаты решения задачи 6.4

```
 for i:=1 to N do
 begin
     for j:=1 to N do
         write(a[i,j],' ');
     writeln;
 end;
 k:=0;
//Обработка элементов, расположенных
//на диагоналях матрицы.
 for i:=1 to N do
 begin
     if (a[i,i]>0) then    k:=k+1;
     if a[i,N-i+1]>0 then k:=k+1;
end;
//Обработка элементов, расположенных
//по периметру матрицы.
 for i:=2 to N-1 do
 begin
     if (a[1,i]>0) then k:=k+1;
     if (a[N,i]>0) then k:=k+1;
     if (a[i,1]>0) then k:=k+1;
```

```
        if (a[i,N]>0) then k:=k+1;
    end;
    { Если элемент, находящийся на пересечении диа-
гоналей, подсчитан дважды,  то уменьшить вычислен-
ное значение k на один. }
     if (n mod 2<>0) and (a[N div 2+1,N div 2+1]>0)
        then
                k:=k-1;
     writeln('k=',k);
    end.
```

ЗАДАЧА 6.5. Проверить, является ли заданная квадратная матрица единичной.

Единичной называют матрицу, у которой элементы главной диагонали – единицы, а все остальные – нули. Например,

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Решать задачу будем так. Предположим, что матрица единичная, (`pr:=true`) и попытаемся доказать обратное. В двойном цикле по строкам (`i:=1,2,...,N`) и по столбцам (`j:=1,2,...,N`) перебираем все элементы матрицы. Если диагональный элемент ($i=j$) не равен единице или элемент, расположенный вне диагонали ($i\neq j$), не равен нулю[71], то в логическую переменную `pr` записываем значение `false` и прекращаем проверку (аварийно покидаем цикл). После цикла проверяем значение `pr`, если переменная `pr` попрежнему равна `true`, то матрица единична, иначе она такой не является.

```
    var a:array[1..10,1..10] of real;
     i,j,n:integer;
     pr:boolean;
    begin
     writeln('Введите размер матрицы');
     readln(n);
     writeln('Введите матрицу');
     for i:=1 to n do
```

71 Воспользовавшись логическими операциями **and** и **or,** это сложное условие можно записать так *if ((i=j) and (a[i,j]<>1)) or ((i<>j) and (a[i,j]<>0)) then*

```
        for j:=1 to n do
        read(a[i,j]);
    {Предположим, что матрица единичная,
     и присвоим логической переменной значение
     истина. Если значение этой переменной при
     выходе из цикла не изменится, это будет
     означать, что матрица единичная.}
     pr:=true;
     for i:=1 to n do
        for j:=1 to n do
        if ((i=j) and (a[i,j]<>1)) or ((i<>j)
                          and (a[i,j]<>0))  then
    {Если элемент лежит на главной диагонали и не
равен единице или элемент лежит вне главной диаго-
нали и не равен нулю , то }
    begin
    {логической переменной присвоить значение ложь,
    это будет означать, что матрица не единичная.}
            pr:=false;
    { выйти из цикла. }
            break;
        end;
    {Проверка значения логической переменной
     и печать результата.}
    if pr then
        writeln('Матрица единичная')
    else
        writeln('Матрица не является единичной');
    end.
```

ЗАДАЧА 6.6. Преобразовать исходную матрицу так, чтобы последний элемент каждого столбца был заменен разностью минимального и максимального элемента в этом же столбце.

Для решения данной задачи необходимо в каждом столбце найти максимальный и минимальный элементы, после чего в последний элемент столбца записать их разность. Блок-схема алгоритма решения приведена на рис. 6.19.

Ниже приведен текст консольного приложения с комментариями.

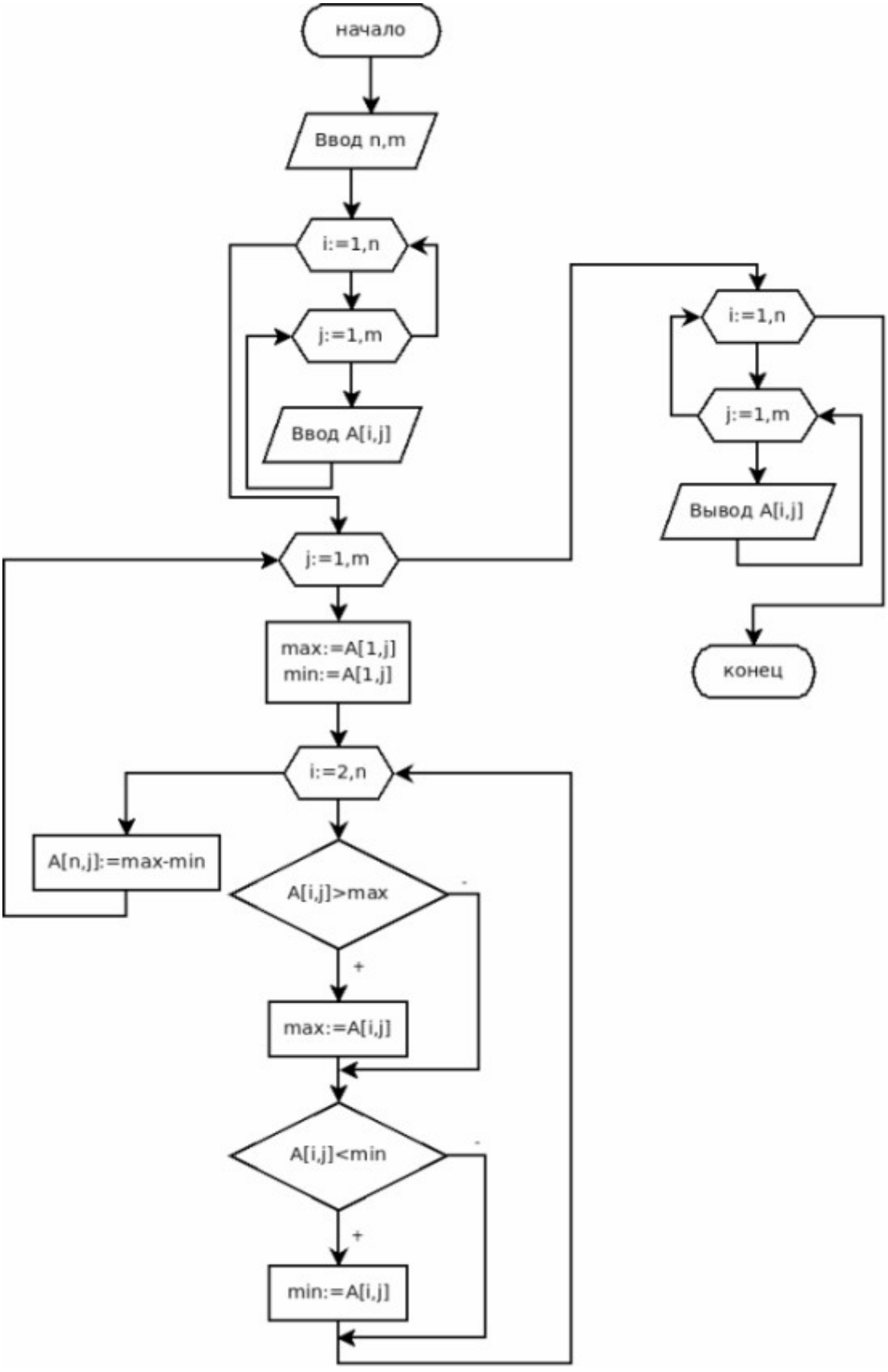

Рисунок 6.19: Блок-схема алгоритма решения задачи 6.6

```
    var a:array[1..25,1..25] of real;
    i,j,n,m:integer;
    max,min:real;
    begin
   {Ввод размеров матрицы.}
    writeln('Введите размеры матрицы');
    readln(n,m);
   {Ввод исходной матрицы.}
    writeln('Введите матрицу');
    for i:=1 to n do
        for j:=1 to m do
            read(a[i,j]);
   {Последовательно перебираем все столбцы матрицы.}
    for j:=1 to m do
    begin
   {Максимальным и минимальным объявляем первый элемент текущего (j-го) столбца матрицы.}
        max:=a[1,j];
        min:=a[1,j];
   {Последовательно перебираем все элементы в текущем (j-м) столбце матрицы.}
        for i:=2 to n do
        begin
   {Если текущий элемент больше максимального, то его и объявляем максимальным.}
            if a[i,j]>max then max:=a[i,j];
   {Если текущий элемент меньше минимального, то его и объявляем минимальным.}
            if a[i,j]<min then min:=a[i,j];
        end;
   {В последний элемент столбца записываем разность между максимальным и минимальным элементами столбца.}
        a[n,j]:=max-min;
    end;
   {Вывод преобразованной матрицы.}
    writeln('Преобразованная матрица');
```

```
    for i:=1 to n do
        begin
            for j:=1 to m do
                write(a[i,j]:7:3,' ');
            writeln;
        end;
end.
```

Теперь давайте создадим визуальное приложение, реализующее рассмотренный алгоритм. За основу возьмем форму (рис. 6.9) и проект транспонирования матрицы `A(N,M)`, разработанные для задачи 6.2. Окно формы несколько изменим. Кнопку `Транспонирование` переименуем в `Преобразование матрицы`, а свойство `Caption` метки `label5` установим `Преобразованная матрица A`. Кроме того изменим свойство `Caption` формы. Окно измененной формы представлено на рис. 6.20.

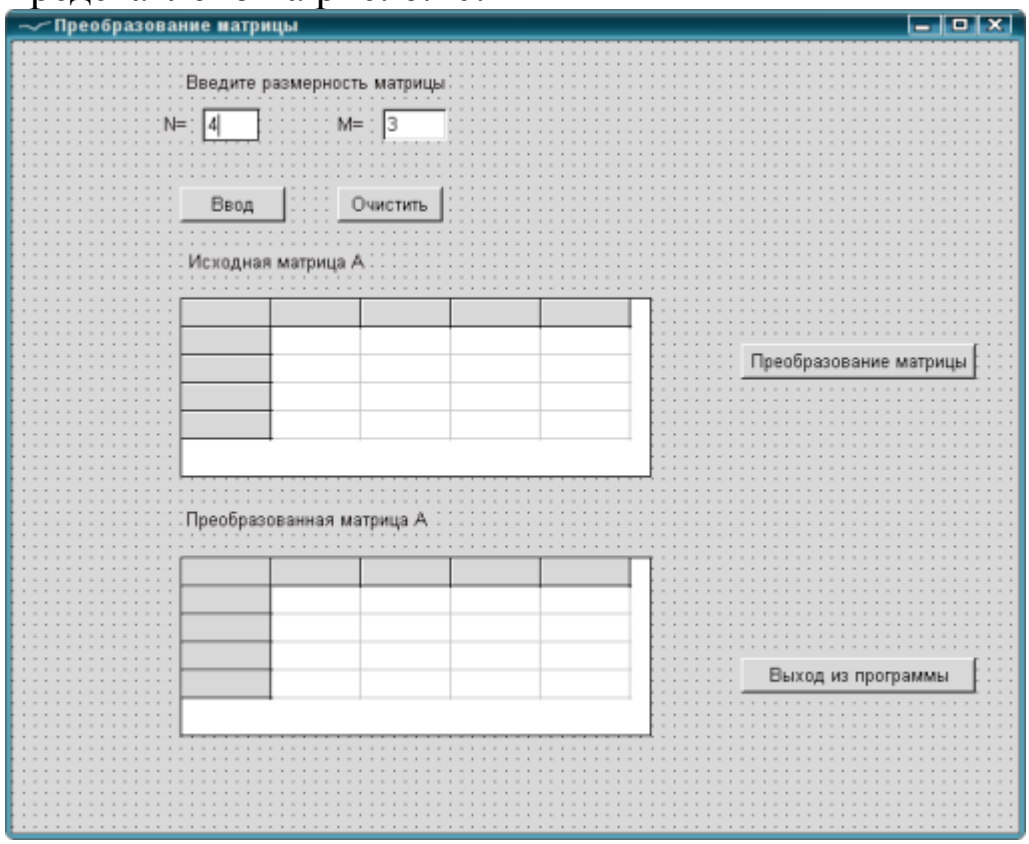

Рисунок 6.20: Окно формы решения задачи 6.6

Обработчики кнопок `Ввод`, `Очистить`, `Выход из программы` изменятся мало. Рассмотрим алгоритм работы обработчика кнопки `Преобразование матрицы`. В этом обработчике будет считывание матрицы из компонента `StringGrid1`, преобразование матрицы `A`

по алгоритму, представленному на рис. 6.19, вывод преобразованной матрицы в компонент StringGrid2.

Текст модуля визуального приложения решения задачи 6.6 с комментариями приведен ниже.

```
unit Unit1;
{$mode objfpc}{$H+}
interface
uses
 Classes,    SysUtils,    LResources,    Forms,
Controls, Graphics, Dialogs, StdCtrls,Grids;
//Описание формы
type
 { TForm1 }
 TForm1 = class(TForm)
 Button1: TButton;
 Button2: TButton;
 Button3: TButton;
 Button4: TButton;
 Edit1: TEdit;
 Edit2: TEdit;
 Label1: TLabel;
 Label2: TLabel;
 Label3: TLabel;
 Label4: TLabel;
 Label5: TLabel;
 StringGrid1: TStringGrid;
 StringGrid2: TStringGrid;
 procedure Button1Click(Sender: TObject);
 procedure Button2Click(Sender: TObject);
 procedure Button3Click(Sender: TObject);
 procedure Button4Click(Sender: TObject);
private
 { private declarations }
public
 { public declarations }
 end;
var
 A:array[1..25,1..25] of real;
```

```
    N,M:integer;
    Form1: TForm1;
   implementation
   { TForm1 }
   //Обработчик кнопки «Выход из программы».
   procedure TForm1.Button4Click(Sender: TObject);
   begin
    Close;
   end;
   //Обработчик первой кнопки — кнопки
   //ввода размерности матрицы.
   procedure TForm1.Button1Click(Sender: TObject);
   var i:byte;kod_n,kod_m,kod:integer;
   begin
   //Ввод размерности матрицы.
   //Символьная информация преобразовывается
   //в числовую и записывается в переменные N и M.
    Val(Edit1.Text,N,kod_m);
    Val(Edit2.Text,M,kod_n);
    //Если преобразование прошло успешно
   //и введенные размеры удовлетворяют
   //писанию матриц A и B,
    if  (kod_n=0)  and  (kod_m=0)  and  (N>0)  and
(N<26) and (M>0) and (M< 26) then
   //то
    begin
   //визуализируется первая матрица,
        StringGrid1.Visible:=true;
   //соответствующая ей надпись,
        Label4.Visible:=true;
   //кнопки Очистить
        Button2.Visible:=true;
   //и преобразование матрицы.
        Button3.Visible:=true;
        with StringGrid1 do
        begin
            ColCount:=M+1;
            RowCount:=N+1;
```

```
    //и нумеруются строки и
            for i:=1 to RowCount-1 do
                Cells[0,i]:=IntToStr(i);
    //столбцы первой таблицы.
            for i:=1 to ColCount-1 do
                Cells[i,0]:=IntToStr(i);
        end;
        StringGrid2.ColCount:=M+1;
        StringGrid2.RowCount:=N+1;
     end
     else
     begin
    //При некорректном вводе выдается
    //соответствующее сообщение.
        MessageDlg('Размеры матрицы введены не
верно!',MtInformation,[mbOk],0);
    //Устанавливаются стартовые параметры
    //в поля ввода.
        Edit1.Text:='4';
        Edit2.Text:='3';
     end; end;
    //Обработчик кнопки «Преобразование матрицы».
    procedure TForm1.Button2Click(Sender: TObject);
    var i,j:integer;
     max,min:real;
    begin
     StringGrid2.Visible:=true;
     label5.Visible:=true;
     for i:=1 to N do       //Цикл по номерам строк.
     for j:=1 to M do   //Цикл по номерам столбцов.
    //Считывание элементов матрицы A
    //из компонента StringGrid1.
     A[i,j]:=StrToFloat(StringGrid1.Cells[j,i]);
     with StringGrid2 do
     begin
        for i:=1 to RowCount-1 do //Нумеруются
            Cells[0,i]:=IntToStr(i);  //строки и
                          //столбцы 2-й матрицы.
```

```
        for i:=1 to ColCount-1 do
            Cells[i,0]:=IntToStr(i);
    end;
   //Решение задачи 6.6.
    for j:=1 to m do
    begin
   {Максимальным и минимальным объявляем первый
элемент текущего (j-го) столбца матрицы.}
        max:=a[1,j];
        min:=a[1,j];
   {Последовательно перебираем все элементы в те-
кущем (j-м) столбце матрицы.}
        for i:=2 to n do
        begin
   {Если текущий элемент больше максимального, то
его и объявляем максимальным.}
            if a[i,j]>max then max:=a[i,j];
   {Если текущий элемент меньше минимального, то
его и объявляем минимальным.}
            if a[i,j]<min then min:=a[i,j];
        end;
   {В последний элемент столбца записываем раз-
ность между максимальным и минимальным элементами
столбца.}
    a[n,j]:=max-min;
    end;
   //Элементы преобразованной матрицы A выводятся
в ячейки
   //таблицы на форме.
    for i:=1 to N do       //Цикл по номерам строк.
    for j:=1 to M do  //Цикл по номерам столбцов.
//Запись элемента преобразованной матрицы A в
ячейку StringGrid2.

    StringGrid2.Cells[j,i]:=FloatToStr(A[i,j]);
   //Делаем первую кнопку невидимой.
    Button1.Visible:=False;
   end;
```

```
//Обработчик кнопки «Очистить».
procedure TForm1.Button3Click(Sender: TObject);
 var i,j:integer;
begin
//Очистка компонента StringGrid1.
 with StringGrid1 do
     for i:=1 to RowCount-1 do
         for j:=1 to ColCount-1 do
             Cells[j,i]:='';
//Очистка компонента StringGrid2.
 with StringGrid2 do
     for i:=1 to RowCount-1 do
         for j:=1 to ColCount-1 do
             Cells[j,i]:='';
//Делаем невидимыми компоненты
//StringGrid1, StringGrid2,labe4, label5.
 StringGrid1.Visible:=False;
 StringGrid2.Visible:=False;
 label4.Visible:=False;
 label5.Visible:=False;
//Делаем невидимыми кнопки
//«Преобразование матрицы» и «Очистить».
 Button2.Visible:=False;
 Button3.Visible:=False;
//Делаем видимой кнопку «Ввод».
 Button1.Visible:=True;
//Запись начальных значений размеров
//матрицы (N=4, M=3).
 Edit1.Text:='4';
 Edit2.Text:='3';
end;
initialization
 {$I unit1.lrs}
end.
```

На рис. 6.21 представлено окно с результатами решения задачи.

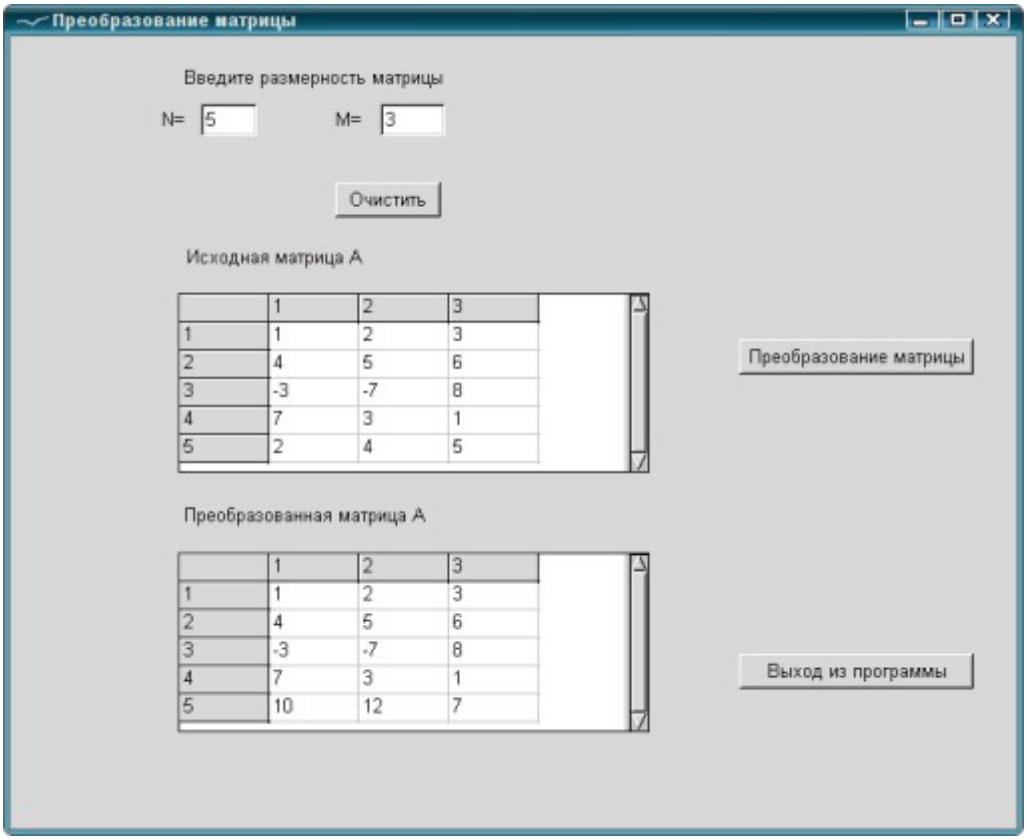

Рисунок 6.21: Результаты решения задачи 6.6

ЗАДАЧА 6.7. Поменять местами n-й и r-й столбцы матрицы A(K,M).

Задача сводится к обмену n-го и r-го элементов во всех строках матрицы. Блок-схема приведена на рис. 6.22.

Ниже приведен листинг консольного приложения с комментариями.

Результаты работы программы приведены на рис. 6.23.

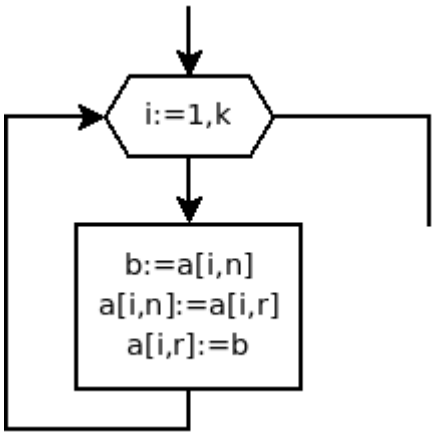

Рисунок 6.22: Блок-схема алгоритма решения задачи 6.7

```
type matrica=
  array [1..15,1..15] of real;
var
a:matrica; b:real;
i,j,k,m,n,r:byte;
begin
   //Ввод размеров матрицы.
   write ('k=');readln(k);
   write ('m=');readln(m);
   //Ввод матрицы A.
```

```
writeln('Матрица A');
for i:=1 to k do
 for j:=1 to m do read(a[i,j]);
//Ввод номеров столбцов матрицы,
//подлежащих обмену.
repeat
 write('n=');readln(n);
 write('r=');readln(r);
{Ввод считается верным, если n и r
не больше m и не равны друг другу.}
until (n<=m) and (r<=m) and (n<>r);
```

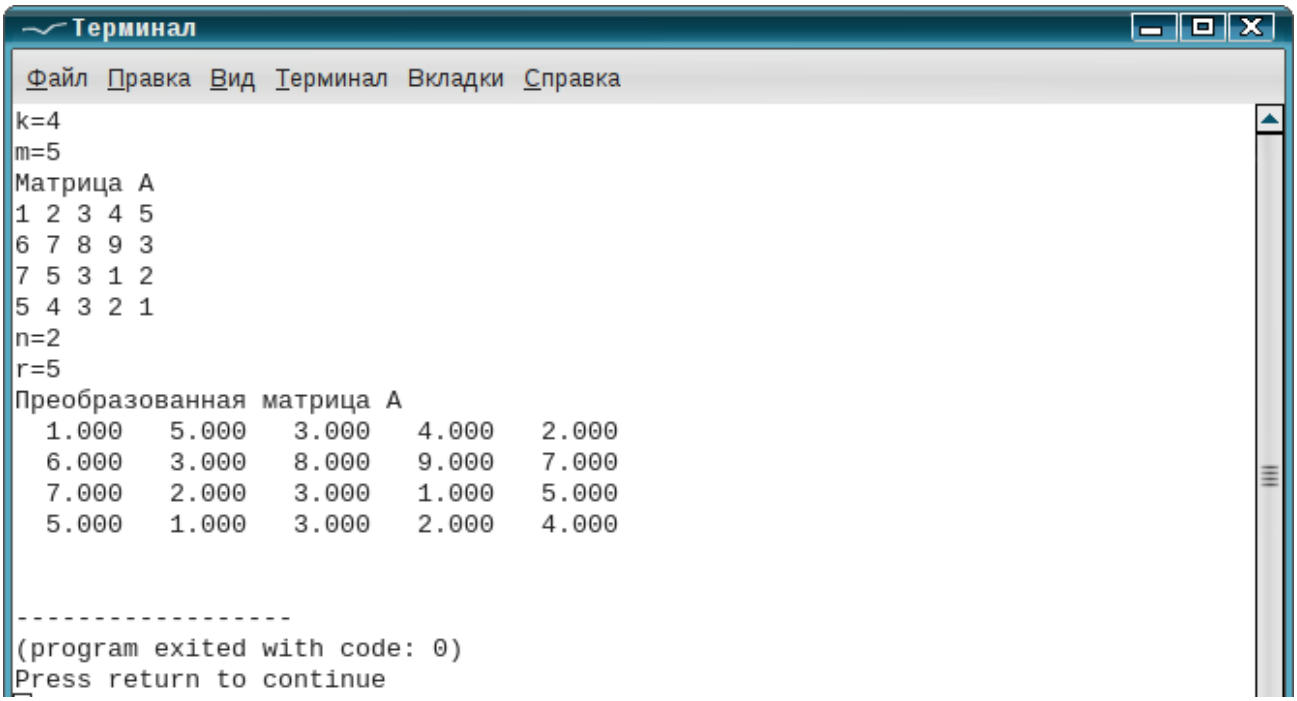

Рисунок 6.23: Результаты решения задачи 6.7

```
{Элементы столбца с номером r заменить
элементами столбца с номером n.}
for i:=1 to k do
begin
 b:=a[i,n];
 a[i,n]:=a[i,r];
 a[i,r]:=b
end;
writeln('Преобразованная матрица A');
for i:=1 to k do
begin
 for j:=1 to m do
```

```
        write(a[i,j]:7:3,' ');
     writeln;
   end;
end.
```

ЗАДАЧА 6.8. Преобразовать матрицу A(m,n) так, чтобы строки с нечетными индексами были упорядочены по убыванию, с четными – по возрастанию.

Каждая строка матрицы является одномерным массивом. Поэтому для упорядочивания строки или столбца можно использовать обычные алгоритмы сортировки массивов. При решении задачи необходимо последовательно просматривать все строки матрицы, если номер строки нечетен, то сортируем строку методом пузырька по убыванию, иначе — по возрастанию.

Блок-схема этого алгоритма представлена на рис. 6.24. Ниже представлено консольное приложение решения этой задачи с комментариями.

На рис. 6.25 приведены результаты работы программы.

```
var
 a:array [1..15,1..15] of real;
 j,i,k,m,n:byte;
b:real;
begin
//Ввод размеров матрицы.
 writeln('введите m и n');
 readln(m,n);
//Ввод матрицы.
 writeln('Матрица A');
 for i:=1 to m do
     for j:=1 to n do
         read(a[i,j]);
//Преобразование матрицы.
for i:=1 to m do
if (i mod 2)=0 then
{ Если номер строки четный, то }
begin
{ упорядочить ее элементы по возрастанию. }
```

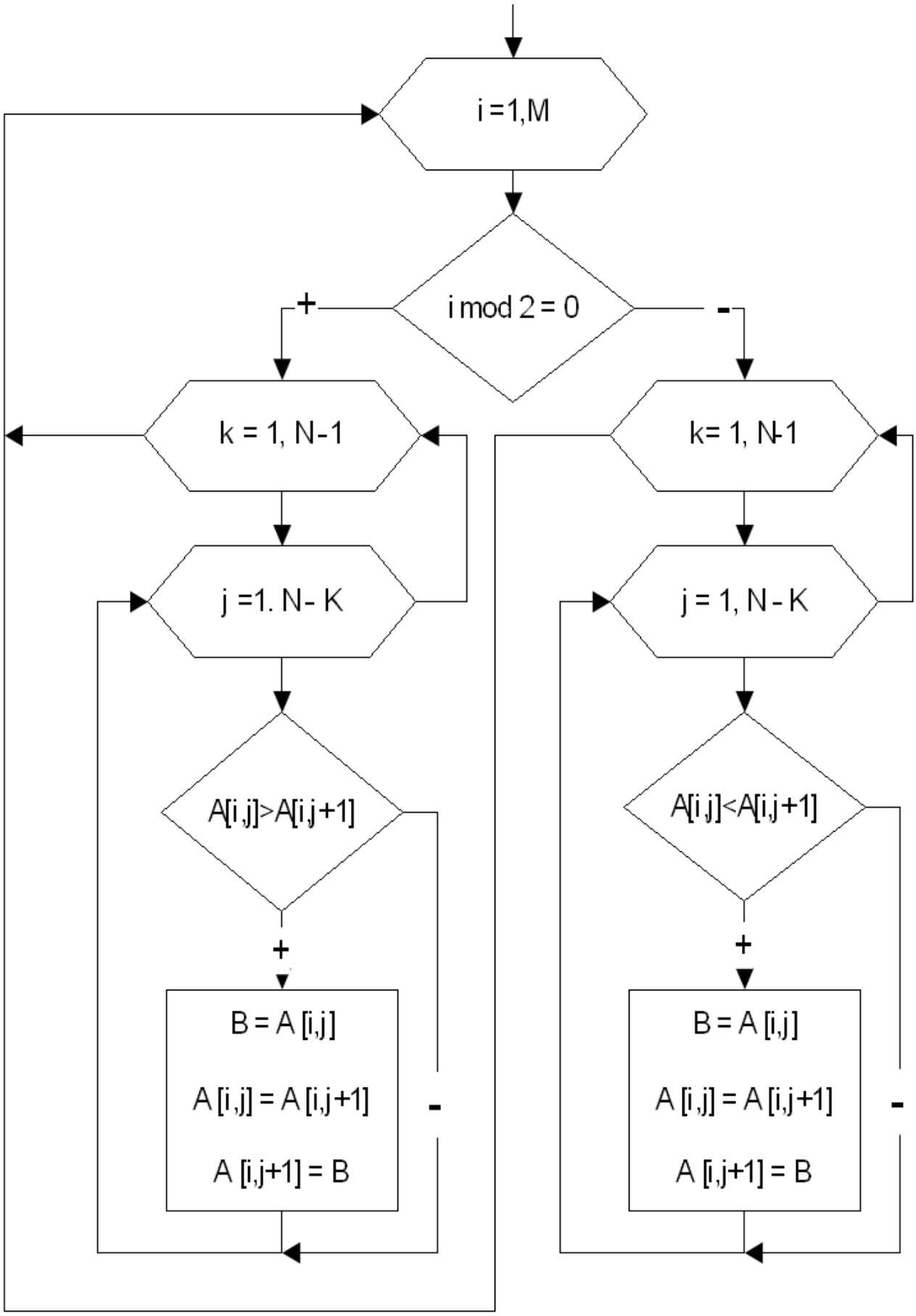

Рисунок 6.24: Блок-схема алгоритма решения задачи 6.8

```
{Упорядочивание строки матрицы методом
пузырька по возрастанию.}
     for k:=1 to n-1 do
         for j:=1 to n-k do
             if  a[i,j] > a[i,j+1] then
             begin
                 b:=a[i,j];
                 a[i,j]:=a[i,j+1];
                 a[i,j+1]:=b;
             end
end
else{ Если номер строки нечетный,
     то упорядочить ее элементы по убыванию.}
```

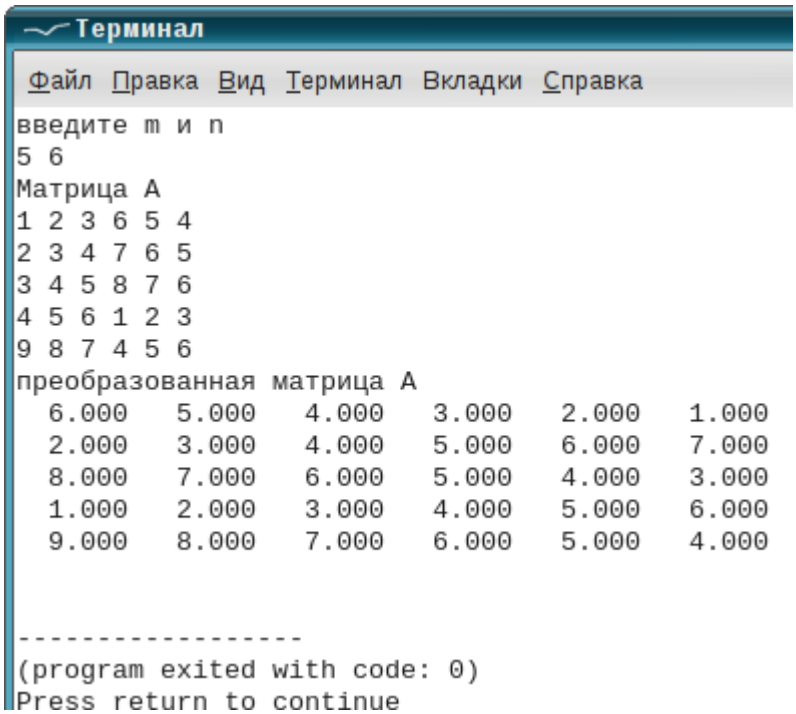

Рисунок 6.25: Результаты решения задачи 6.8

```
{Упорядочивание строки матрицы методом
пузырька по убыванию.}
 for k:=1 to n-1 do
     for j:=1 to n-k do
         if a[i,j] < a[i,j+1] then
         begin
             b:=a[i,j];
             a[i,j]:=a[i,j+1];
             a[i,j+1]:=b;
         end;
```

```
//Вывод преобразованной матрицы.
writeln('преобразованная матрица A');
for i:=1 to m do
begin
 for j:=1 to n do
     write (a[i,j]:7:3,' ');
writeln
end
end.
```

ЗАДАЧА 6.9. Задана матрица целых положительных чисел A(n,m). Сформировать вектор P(n), в который записать сумму простых чисел каждой строки матрицы в четверичной системе счисления, если в строке нет простых чисел, в соответствующий элемент массива записать число 0.

Для решения этой задачи нам понадобятся функции: проверки, является ли число простым, и перевода целого числа в четверичную систему счисления. Функция проверки, является ли число простым, подробно рассматривалась в пятой главе при решении задачи 5.7. Поэтому здесь просто приведем ее текст.

```
function prostoe(N:integer):boolean;
var i:integer;  pr:boolean;
begin
if N<1 then pr:=false
else
begin
 pr:=true;
 for i:=2 to N div 2 do
     if (N mod i = 0) then
     begin
         pr:=false;
         break;
     end;
end;
prostoe:=pr;
end;
```

Кроме того в пятой главе мы рассматривали алгоритм (рис. 5.45) и функцию перевода

```
function  perevod(N:real;P:word;kvo:word):real;
```

вещественного числа в *p*-чную систему счисления (задача 5.10). Нужная нам функция перевода целого числа в четверичную систему счисления является частным случаем рассмотренной ранее функции perevod. Ниже приведен текст функции perevod4 , которая переводит целое положительное число в четверичную систему счисления.

```
   {Функция перевода целого числа N в четверичную
систему счисления.}
   function perevod4(N:word):word;
    var
        s1,i ,q, ost: word;
   begin
   {В переменной s1 мы будем собирать число в чет-
веричной системе счисления.}
    s1:=0;
   {В переменной q будем последовательно хранить
степени десяти, вначале туда записываем 1 — десять
в 0 степени, а затем последовательно в цикле будем
умножать q на 10.}
   q:=1;
   {Перевод целого числа. Пока число не станет
равным 0.}
    while (N<>0) do
    begin
   {Вычисляем ost — очередной разряд числа, как
остаток от деления N на 4 (основание системы счис-
ления).}
        ost:=N mod 4;
   {Очередной разряд числа умножаем на 10 в степе-
ни i и добавляем к формируемому числу s1.}
        s1:=s1+ost*q;
   {Уменьшаем число N в 4 раза путем целочисленно-
го деления на 4.}
        N1:=N1 div 4;
   {Формируем следующую степень десятки.}
        q:=q*10;
    end;
   {Возвращаем число в четверичной системе счисле-
ния.}
```

```
 perevod:=s1;
end;
```

В каждой строке надо найти сумму простых чисел, а затем полученное число перевести в четверичную систему счисления. Поэтому необходимо для каждой строки (`i:=1,2,..,n`) выполнить следующее: обнулить сумму S (`S:=0`), организовать цикл по элементам строки (`j:=1,2,...,m`), внутри которого проверять, является ли текущий элемент $A_{i,j}$ простым и, если является, добавлять его к сумме `S`. После выхода из цикла по `j` необходимо проверить, были ли в строке с номером `i` простые числа (`S>0`), и, если были, перевести `S` в четверичную систему счисления и сформировать соответствующий элемент массива `P` (`P[i]:=perevod4(S)`).

Блок-схема алгоритма приведена на рис. 6.26.

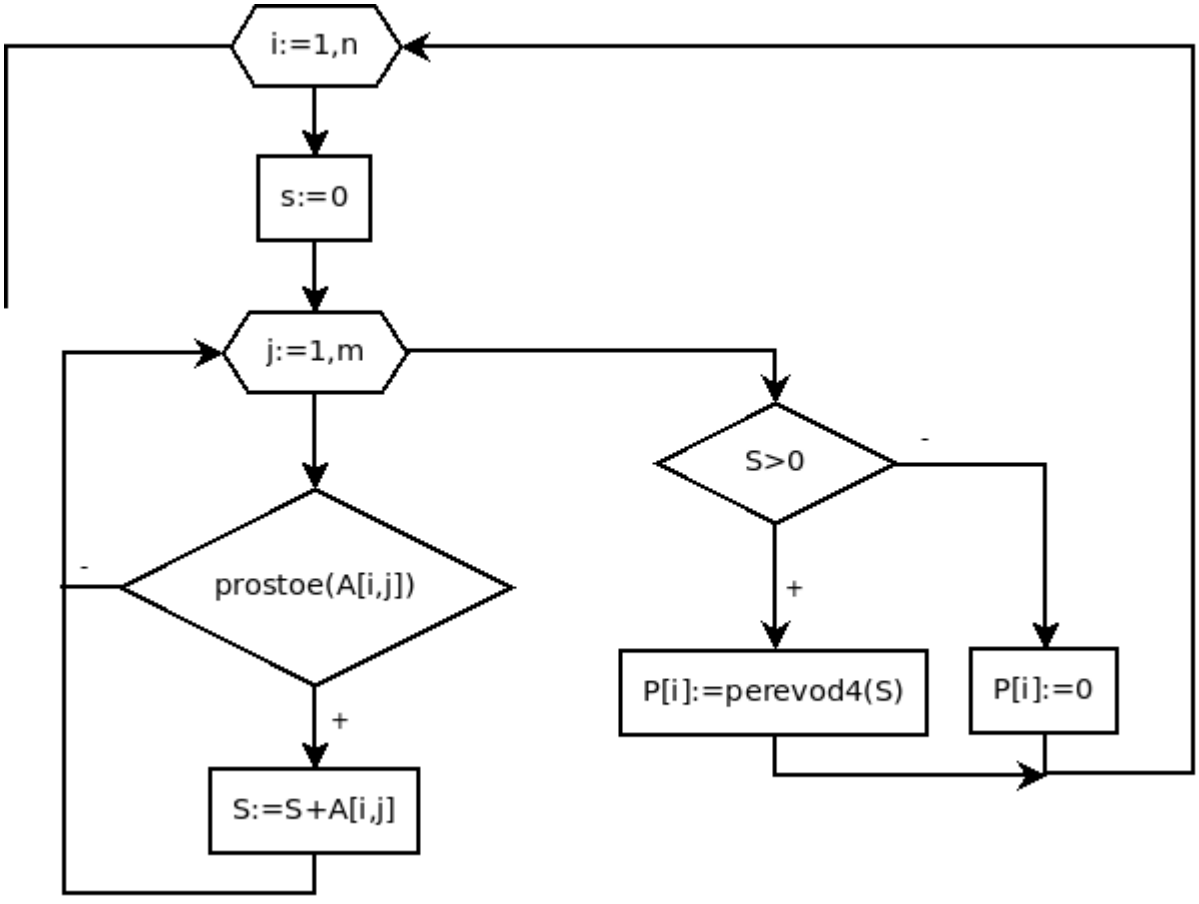

Рисунок 6.26: Блок-схема алгоритма решения задачи 6.9

Полный текст консольного приложения:

```
function prostoe(N:integer):boolean;
var i:integer;  pr:boolean;
begin
if N<1 then pr:=false
```

```
    else
    begin
     pr:=true;
     for i:=2 to N div 2 do
         if (N mod i = 0) then
         begin
             pr:=false; break;
         end;
    end;
    prostoe:=pr;
    end;
    function perevod4(N:word):word;
     var s1,q, ost: word;
    begin
    {В переменной s1 мы будем собирать число в чет-
веричной системе счисления.}
     s1:=0;
    {В переменной q будем последовательно хранить
степени десяти, вначале туда записываем 1 — десять
в 0 степени, а затем последовательно в цикле будем
умножать q на 10.}
    q:=1;
    {Перевод целого числа.
     Пока число не станет равным 0.}
     while (N<>0) do
     begin
    {Вычисляем ost — очередной разряд числа, как
остаток от деления N на 4 (основание системы счис-
ления).}
         ost:=N mod 4;
    {Очередной разряд числа умножаем на 10 в степе-
ни i и добавляем к формируемому числу s1.}
         s1:=s1+ost*q;
    {Уменьшаем число N в 4 раза путем целочисленно-
го деления на 4.}
         N:=N div 4;
    {Формируем следующую степень десятки.}
         q:=q*10;
```

```
    end;
   {Возвращаем число в 4-ной системе счисления.}
    perevod4:=s1;
   end;
   var S,i,j,n,m: word;
    a:array[1..25,1..25] of word;
    p:array[1..25] of word;
   begin
   //Ввод размеров матрицы.
    writeln('Введите размеры матрицы');
    readln(n,m);
    writeln('Введите матрицу A');{Ввод матрицы.}
    for i:=1 to n do
        for j:=1 to m do read(A[i,j]);
   {Последовательно перебираем все строки матрицы
для формирования суммы простых чисел каждой стро-
ки.}
    for i:=1 to n do
    begin
        S:=0;{Вначале сумма равна нулю.}
   {Перебираем все элементы в i-й строке матрицы.}
        for j:=1 to m do
   {Если очередной элемент в i-й строке матрицы —
простое число, то добавляем его к сумме.}
            if prostoe(A[i,j]) then
                        s:=s+A[i,j];
   {Если в строке были простые числа, то их сумму
переводим в четверичную систему счисления и запи-
сываем в p[i].}
        if s>0 then p[i]:=perevod4(s)
   {Если в строке не было простых чисел, то
p[i]:=0.}
        else p[i]:=0;
    end;
   //Вывод сформированного массива P.
    writeln('Массив P');
    for i:=1 to n do  write(P[i],' ');
    writeln;   end.
```

Результаты работы программы представлены на рис. 6.27.

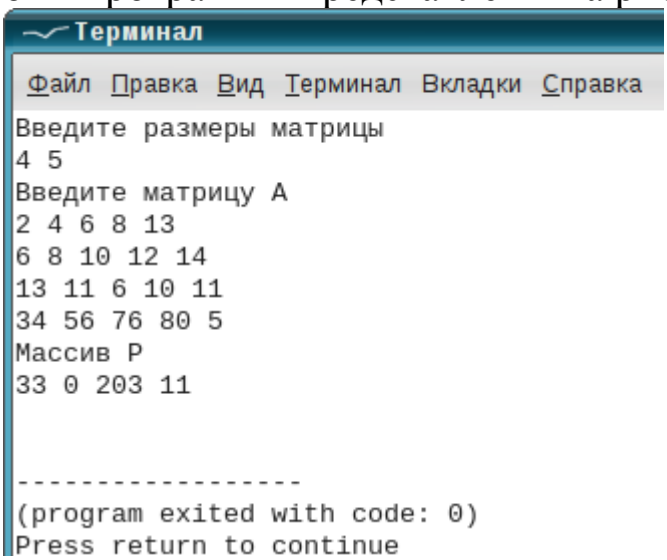

Рисунок 6.27: Результаты работы программы решения задачи 6.9

ЗАДАЧА 6.10. Написать программу умножения двух матриц A(N,M) и B(M,L).

Напомним некоторые сведения из курса математики.

Умножать можно только матрицы, у которых количество столбцов в первой матрице совпадает с количеством строк во второй матрице.

Матрица-произведение имеет столько строк, сколько было в первой матрице и столько столбцов, сколько было во второй. Таким образом, при умножении матрицы `A(N,M)` на матрицу `B(M,L)` получается матрица `C(N,L)`. Каждый элемент матрицы $C_{i,j}$ является скалярным произведением `i`-й строки матрицы `A` и `j`-го столбца матрицы `B`. В общем виде формула для нахождения элемента $C_{i,j}$ матрицы имеет вид:

$$C_{i,j} = \sum_{k=1}^{M} A_{ik} B_{kj} \text{ , где } \texttt{i = 1,N} \text{ и } \texttt{j = 1,L.} \qquad (6.1)$$

Рассмотрим более подробно формирование матрицы `C(3,2)` как произведение матриц `A(3,3)` и `B(3,2)`.

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \\ C_{31} & C_{32} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} \\ a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32} \end{pmatrix}.$$

Следует помнить, что $A \cdot B \neq B \cdot A$.

Блок-схема, реализующая расчет каждого элемента матрицы C по

формуле (6.1), приведена на рис. 6.28. Далее приведен текст программы умножения двух матриц с комментариями. Результат работы представлен на рис. 6.29.

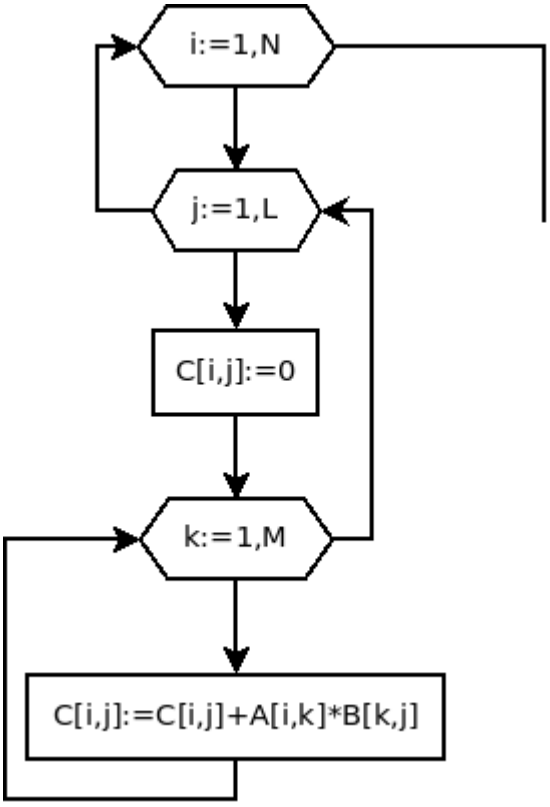

Рисунок 6.28: Блок-схема умножения двух матриц

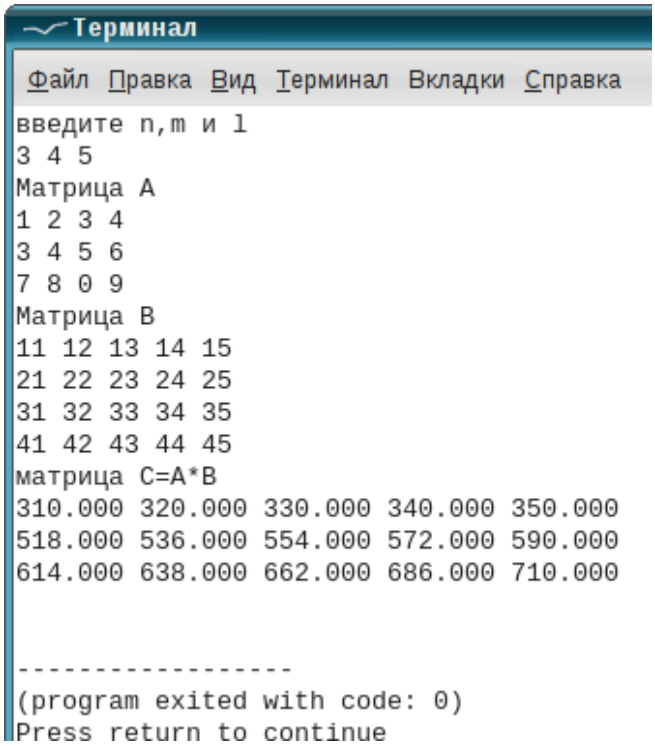

Рисунок 6.29: Результаты работы программы умножения матриц

```
//Умножение двух матриц.
type matrica=array [1..15,1..15] of real;
var  a,b,c:matrica; i,j,M,N,L,k:byte;
begin
 //Ввод размеров матриц.
 writeln('введите n,m и l'); readln(N, M, L);
 writeln('Матрица A');//Ввод матрицы A.
 for i:=1 to N do
     for j:=1 to M do read(a[i,j]);
 writeln('Матрица B');//Ввод матрицы B.
 for i:=1 to M do
     for j:=1 to L do read(b[i,j]);
 for i:=1 to N do //Формирование матрицы C.
     for j:=1 to L do
     begin
```

```
{В C[i,j] хранится результат скалярного произве-
дения i-й строки на j-й столбец.}
            c[i,j]:=0;
            for k:=1 to M do
                c[i,j]:=c[i,j]+a[i,k]*b[k,j];
        end;
    writeln('матрица C=A*B');//Вывод матрицы C=AB.
    for i:=1 to N do
    begin
        for j:=1 to L do
                write(c[i,j]:7:3,' ');
        writeln;
    end;
   end.
```

ЗАДАЧА 6.11 В матрице натуральных чисел A(N,M) найти строки, где находится максимальное из простых чисел. Элементы в них упорядочить по возрастанию. Если в матрице нет простых чисел, то оставить ее без изменений.

Перед решением задачи отметим некоторые ее особенности[72]. В матрице может не быть простых чисел, максимальных значений может быть несколько, и при этом некоторые из них будут находиться в одной строке.

При решении задачи нам понадобятся следующие подпрограммы:

1. Функция `Prostoe`, которая проверяет, является ли число P типа `word` простым. Она возвращает значение `true`, если число `P` – простое и `false` – в противном случае. Заголовок функции имеет вид

```
Function Prostoe (P:word):Boolean;
```

2. Процедура `Udal`, которая из массива чисел `X` удаляет значения, встречающиеся более одного раза. У процедуры два параметра: массив `X` и его размер `N`, оба – параметры переменные. Заголовок процедуры имеет вид:

```
Procedure Udal(var X:massiv; var N:word);
```

Перед описанием процедуры следует описать тип данных massiv (например, `massiv = array [1..200] of word`). Блок-схема процедуры `Udal` представлена на рис. 6.30.

72 Авторы рекомендуют читателям внимательно изучить этот пример, в нем сконцентрированы практически все основные моменты, рассмотренные нами до сих пор.

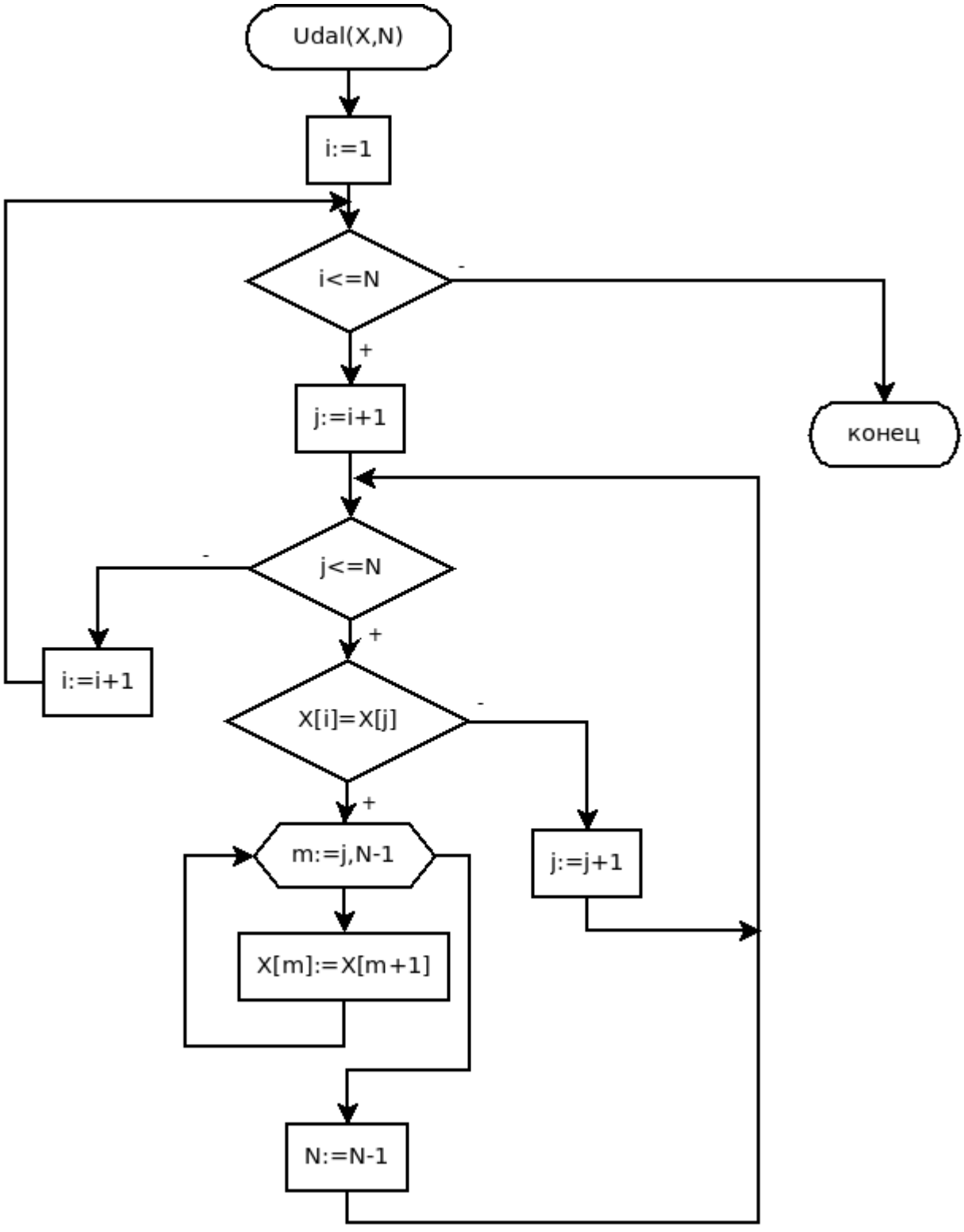

Рисунок 6.30: Блок-схема процедуры Udal

Удаление повторяющихся элементов происходит следующим образом. Просматриваются все элементы, начиная с первого, `i`-й элемент сравнивается со всеми последующими . Если $x_i = x_j$, то встретился повторяющийся элемент, и мы удаляем из массива элемент с но-

мером `j`. Алгоритм удаления был подробно рассмотрен в пятой главе.

3. Функция `Nalichie` возвращает `true`, если число `a` присутствует в массиве `b`, и `false` – в противном случае. Заголовок процедуры имеет вид:

```
Function Nalichie(a:word; b:massiv; N:word);
```

Блок-схема функции представлена на рис. 6.31.

4. Процедура `Vozr` упорядочения массива `x` по возрастанию.

Алгоритмы упорядочения рассматривались в пятой главе. Здесь авторами использовался алгоритм сортировки методом пузырька.

У процедуры `Vozr` два параметра: массив `x` (параметр-переменная) и его размер `N` (параметр-значение). Заголовок процедуры имеет вид:

```
Procedure Vozr(var x:massiv;
                              N:word);
```

Рассмотрим более подробно алгоритм решения задачи 6.11, который приведен на рис. 6.32-6.33. После ввода матрицы (блоки 1-4) предполагаем, что простых чисел нет.

В логическую переменную `Pr` записываем `false`, как только встретится простое число, в переменную Pr запишем `true`.

Количество максимальных значений среди простых чисел равно 0 (`k:=0`) (блок 5).

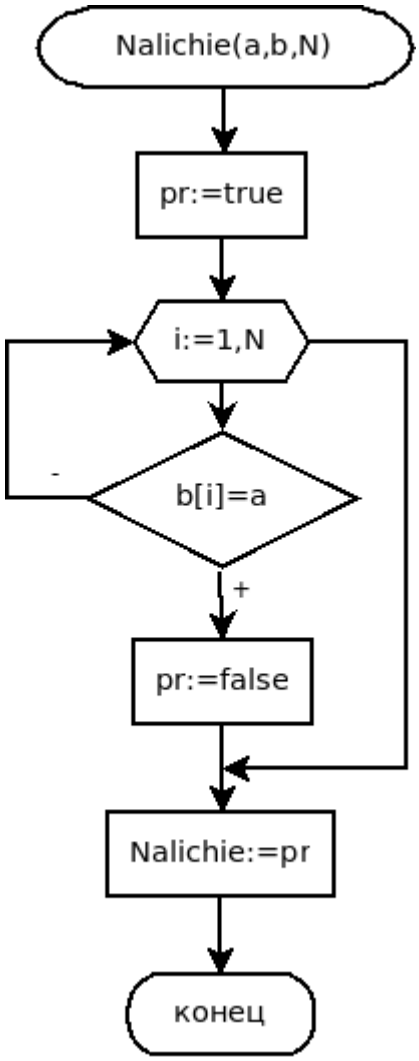

Рисунок 6.31: Блок-схема функции Nalichie

Для проверки, является ли простым числом каждый элемент матрицы, обращаемся к функции `Prostoe`. Если число простое (блок 8), проверяем, первое ли это простое число в матрице (блок 9).

Если это первое простое число, то переписываем его в переменную `max`, в переменную `k` записываем число `1` (количество максимумов равно `1`), номер строки, в которой находится максимум, записы-

ваем в массив mas под номером k[73]. В связи с тем что в матрице есть простые числа, в переменную Pr записываем true (блок 10).

Если это не первое простое число, сравниваем A_{ij} с переменной max. Если $A_{ij} > max$ (блок 11), то в переменную max запишем A_{ij}, в переменную k запишем 1 (есть один максимум), в mas[k] записываем i – номер строки, где находится максимальный элемент (блок 12). Если $A_{ij} = max$ (блок 13), то встретилось число, равное переменной max. В этом случае значение k увеличиваем на 1 и в mas[k] записываем номер строки, где находится элемент, равный max.

В результате двойного цикла обработки всех элементов матрицы (блоки 6-14) в переменной max будет храниться максимальное из простых чисел, в переменной k – количество максимумов, в массиве mas из k элементов будут храниться номера строк, где находятся максимальные значения среди простых чисел матрицы. В переменной Pr хранится true, если в матрице есть простые числа, false – в противном случае.

Если в матрице нет простых чисел (блок 15), выводим соответствующее сообщение (блок 16), в противном случае – с помощью процедуры Udal (блок 17) удаляем из массива mas элементы, встречающиеся более одного раза[74]. Затем просматриваем все строки матрицы (цикл начинается блоком 18), если номер этой строки присутствует в массиве mas (блок 19), то переписываем текущую строку матрицы в массив b (блоки 20-21) и обращаемся к процедуре упорядочивания массива по возрастанию Vozr (блок 22).

Упорядоченный массив b переписываем в i-ю строку матрицы A (блоки 23-24). На последнем этапе выводим на экран матрицу A после преобразования (блоки 25-27).

Ниже приведен листинг всей программы с подробными комментариями.

73 В массиве mas будут храниться номера строк, где находится максимум.

74 Если некоторые максимальные элементы находятся в одной строке, то в массиве mas есть повторяющиеся элементы.

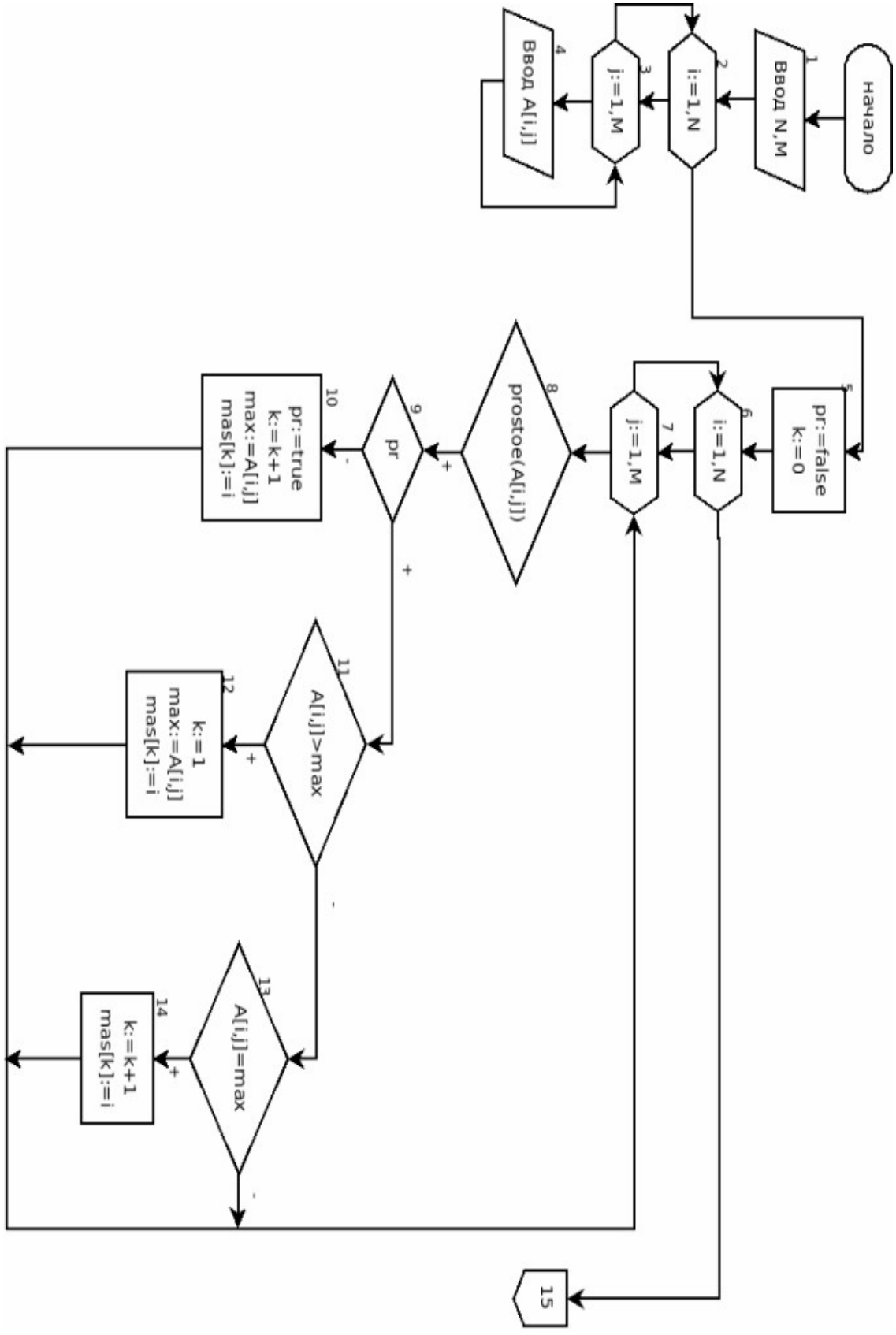

Рисунок 6.32: Блок-схема решение задачи 6.11 (начало)

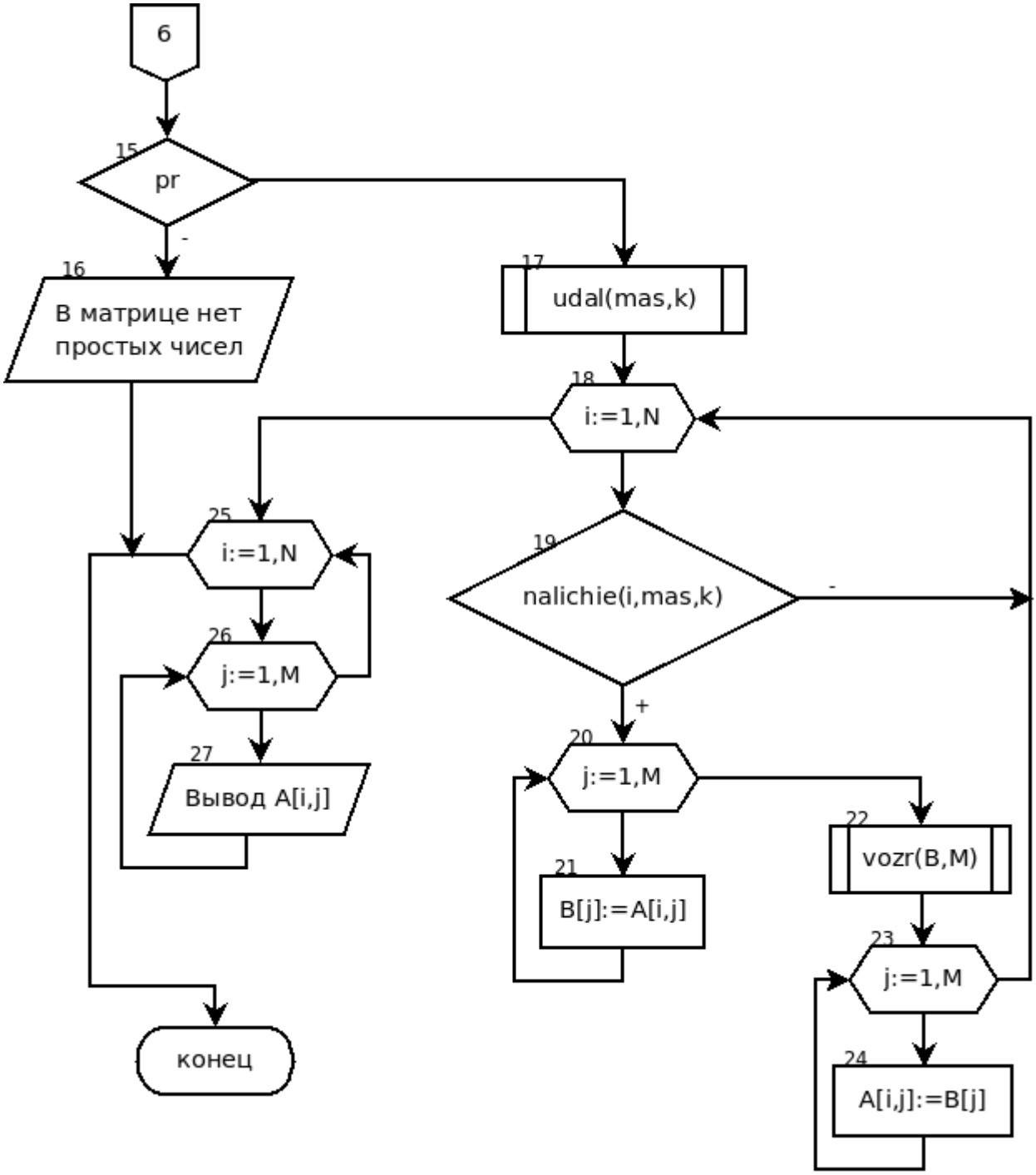

Рисунок 6.33: Блок-схема решения задачи 6.11 (продолжение)

```
{Тип данных massiv будет использоваться
при описании процедур. }
type massiv=array[1..200] of word;
{ Функция prostoe проверяет, является ли
число N простым (true) или нет (false). }
function prostoe(N:word):boolean;
var pr:boolean;i:word;
begin
```

```
    if N>0 then
    begin
   {Предполагаем, что число N – простое (pr=true)}
        pr:=true;
   {Проверяем, делится ли число N на какое-либо
    из чисел от 2 до N/2.}
        for i:=2 to n div 2 do
   { Если встречается число i,
   на которое делится N, то}
        if N mod i =0 then
        begin
   { число N не является простым (pr=false) и }
            pr:=false;
   { выходим из цикла. }
            break;
        end
    end
    else pr:=false;
   { Имени функции присваиваем значение переменной
pr. }
    prostoe:=pr;
   end;
   {Процедура udal удаляет из массива x элементы,
которые встречаются более одного раза. X, N яв-
ляются параметрами-переменными, так как эти значе-
ния возвращаются в головную программу при вызове
процедуры udal.}
   procedure udal(var x:massiv; var n:word);
   var i,j,m:word;
   begin
    i:=1;
   {Просматриваем все элементы, начиная с первого
i=1,2,…,N;i-й элемент сравниваем с последующими
j=i+1,i+2,…,n. }
    while(i<=n) do
    begin
        j:=i+1;
        while(j<=N) do
```

```
   { Если x[i] равно x[j], то встретился повторяю-
щийся элемент}
        if x[i]=x[j] then
        begin
   {и удаляем его (x[j]) из массива. }
            for m:=j to N-1 do x[m]:=x[m+1];
   { После удаления элемента количество элементов
уменьшаем на 1, при этом не переходим к следующему
элементу, так как после удаления под j-м номером
находится уже другой элемент. }
           N:=N-1;
        End
   { Если x[i] не равно x[j], то переходим к сле-
дующему элементу. }
        else j:=j+1;
        i:=i+1;
    end;
   end;
   {Функция nalichie возвращает true, если число a
встречается в массиве b, false – в противном слу-
чае. }
   function
nalichie(a:word;b:massiv;n:word):boolean;
   var
    pr:boolean;
    i:word;
   begin
   {Предполагаем, что в массиве b не встречается
значение a, в pr=false}
    pr:=false;
   { Перебираем все элементы массива. }
    for i:=1 to N do
   { Если очередной элемент массива b равен значе-
нию a, то в pr записываем true}
    if b[i]=a then
    begin
        pr:=true;
   { и выходим из цикла. }
```

```
        break
    end;
   { Имени функции присваиваем значение переменной
pr. }
    nalichie:=pr
   end;
   {Процедура vozr упорядочивает массив x по воз-
растанию. }
   procedure vozr(var x:massiv; n:word);
   { X является параметром-переменной, именно мас-
сив и возвращается в головную программу при вызове
процедуры vozr. }
   var i, j, b :word;
   begin
    for i:=1 to N-1 do
        for j:=1 to N-i do
            if x[j]>x[j+1] then
            begin
                b:=x[j];
                x[j]:=x[j+1];
                x[j+1]:=b;
            end
   end;
   //Начинается основная программа
   var
    N, M, i, j, k, max :word;
    A:array[1..20,1..20] of word;
    pr, L: boolean;
    mas, b:massiv;
   begin
   { Вводим элементы матрицы A. }
    write('N=');readln(N);
    write('M=');readln(M);
   writeln(' Matrica A');
    for i:=1 to N do
    for j:=1 to M do
        read(A[i,j]);
   { Предполагаем, что в матрице нет простых чи-
```

```
сел. }
    Pr:=false;
   { Количество элементов, равных максимальному,
равно 0. }
    k:=0;
   { Перебираем все элементы в матрице. }
    for i:=1 to N do
    for j:=1 to M do
    begin
   { Обращаемся к функции, которая проверяет, яв-
ляется ли число A[I,j] простым. }
        L:=Prostoe(A[i,j]);
   { Если число простое и }
        if L then
   { если простое число встретилось первый раз,}
           if not Pr then
           begin
   { записывем в pr true,}
              Pr:=true;
   { увеличиваем количество максимумов на 1, можно
было просто написать k:=1}
              k:=k+1;
   { Это число записываем в переменную max, это
первое простое число, и предполагаем, что оно мак-
симальное. }
              max:=A[i,j];
   {B mas[k] записываем номер строки, где хранится
число A[I,j]}
              mas[k]:=i
           end
           else
   { Если A[i,j] не первое простое число, то срав-
ниваем max и текущее простое значение матрицы A. }
              if A[i,j]>max then
   { Если A[I,j]> max, то }
              Begin
   { количество максимумов равно 1, т.к. встретил-
ся наибольший в данный момент элемент. }
```

```
                    k:=1;
   { в переменную max записываем A[i,j], }
                    max:=A[i,j];
   { в mas[k] записываем номер строки, где хранит-
ся число A[I,j]}
                    mas[k]:=i
                end
                else
   { Если A[i,j]=max (встретился элемент, равный
максимуму), то }
   if A[i,j]=max then
                    begin
   { количество максимумов увеличиваем на 1. }
                        k:=k+1;
   { в mas[k] записываем номер строки, где хранит-
ся число A[I,j]}
                        mas[k]:=i
                    end
    end;
   {Если в pr осталось значение false,то выводим
сообщение, что в матрице нет простых чисел, }
    if not Pr then writeln('В матрице A нет про-
стых чисел')
    else
    begin
   { иначе удаляем из массива mas, номера строк,
где хранятся максимумы, повторяющиеся элементы. }
        Udal(mas,k);
   {Перебираем все строки матрицы. }
        for i:=1 to N do
        begin
            L:=Nalichie(i,mas,k);
   {Если номер строки присутствует в массиве mas,}
            if L then
            begin
   {то переписываем строку в массив b}
                for j:=1 to M do
                    b[j]:=A[i,j];
```

```
    {упорядочиваем массив b по возрастанию. }
                Vozr(b,M);
    { упорядоченный массив записываем на место i-й
строки матрицы A. }
                for j:=1 to M do
                A[i,j]:=b[j];
            end
        end;
        writeln('Преобразованная матрица A');
        for i:=1 to N do
        begin
            for j:=1 to M do write(A[i,j],' ');
            writeln;
        end
     end
    end.
```

Результаты работы программы приведены на рис. 6.34.

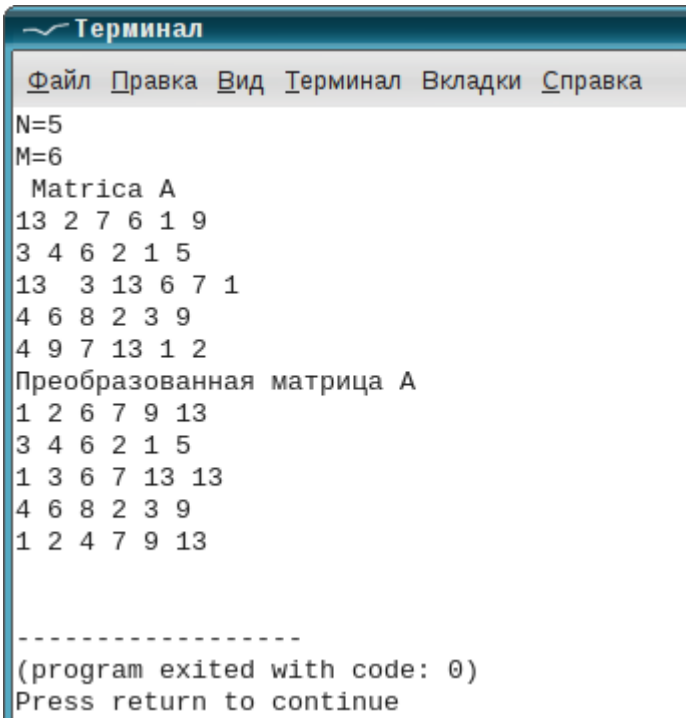

Рисунок 6.34: Результаты решения задачи 6.11

Авторы рекомендуют читателю по рассмотренным алгоритмам и консольным приложениям задач 6.7-6.11 разработать визуальные приложения, аналогичные тем, которые были разработаны для задач 6.2, 6.6.

6.3 Динамические матрицы

Понятие динамического массива можно распространить и на матрицы. Динамическая матрица представляет собой массив указателей, каждый из которых адресует одну строку (или один столбец).

Рассмотрим описание динамической матрицы. Пусть есть типы данных `massiv` и указатель на него `din_massiv`.

```
type massiv=array [1..1000] of real;
din_massiv=^massiv;
```

Динамическая матрица X будет представлять собой массив указателей.

```
Var X: array[1..100] of din_massiv;
```

Работать с матрицей надо следующим образом.

1. Определить ее размеры (пусть `N` – число строк, `M` – число столбцов).

2. Выделить память под матрицу.

```
for i:=1 to N do
getmem(X[i],M*sizeof(real));
```

Каждый элемент статического массива `X[i]` – указатель на динамический массив, состоящий из M элементов типа `real`. В статическом массиве `X` находится `N` указателей.

3. Для обращения к элементу динамической матрицы, расположенному в `i`-й строке и `j`-м столбце, следует использовать конструкцию языка Турбо Паскаль `X[i]^[j]`.

4. После завершения работы с матрицей необходимо освободить память.

```
for i:=1 to N do
     freemem(b[i],M*sizeof(real));
```

Рассмотрим работу с динамической матрицей на следующем примере.

ЗАДАЧА 6.12. В каждой строке матрицы вещественных чисел B(N,M) упорядочить по возрастанию элементы, расположенные между максимальным и минимальным значением.

Алгоритмы упорядочивания рассматривались в пятой главе, основные принципы работы с матрицами – в предыдущих параграфах текущей главы, поэтому в комментариях к тексту программы основное внимание уделено особенностям работы с динамическими матрицами.

```
   { Описываем тип данных massiv как массив 1000
вещественных чисел. }
   type massiv=array [1..1000] of real;
   { Указатель на массив. }
   din_massiv=^massiv;
   { Тип данных matrica – статический массив ука-
зателей, каждые элемент которого является адресом
массива вещественных чисел.}
   matrica=array [1..100] of din_massiv;
   var
   Nmax,Nmin,i,j,n,m,k:word;
   { Описана динамическая матрица b. }
   b:matrica;
   a,max,min:real;
   begin
   { Вводим число строк N и число столбцов M. }
   write('N=');readln(N);
   write('M=');readln(M);
   { Выделяем память под матрицу вещественных чи-
сел размером N на M.}
   for i:=1 to N do
   getmem(b[i],M*sizeof(real));
   { Вводим Матрицу B. }
   writeln('Matrica B');
   for i:=1 to N do
   for j:=1 to M do
   read(b[i]^[j]);
   {В каждой строке находим максимальный, мини-
мальный элементы и их номера, и элементы, располо-
женные между ними упорядочиваем методом
пузырька. }
   for i:=1 to N do
   begin
   { Поиск минимального, максимального элементов в
i-й строке матрицы и их номеров.}
   max:=b[i]^[1];
   Nmax:=1;
   min:=b[i]^[1];
```

```
   Nmin:=1;
   for j:=2 to M do
   begin
   if b[i]^[j]>max then
   begin
   max:=b[i]^[j];
   nmax:=j
   end;
   if b[i]^[j]<min then
   begin
   min:=b[i]^[j];
   nmin:=j
   end;
   end;
   { Если минимальный расположен позже максимального, nmin и nmax меняем местами}
   if nmax<nmin then
   begin
   j:=nmax;
   nmax:=nmin;
   nmin:=j;
   end;
   {В i-той строке упорядочиваем элементы, расположенные между nmin и nmax, методом пузырька. }
   j:=1;
   while nmax-1-j>=nmin+1 do
   begin
   for k:=nmin+1 to nmax-1 -j do
   if b[i]^[k]>b[i]^[k+1] then
   begin
   a:=b[i]^[k];
   b[i]^[k]:=b[i]^[k+1];
   b[i]^[k+1]:=a;
   end;
   j:=j+1;
   end;
   end;
   { Выводим преобразованную матрицу. }
```

```
writeln('Упорядоченная матрица B');
for i:=1 to N do
begin
for j:=1 to M do
write(b[i]^[j]:6:2, ' ');
writeln
end;
{ Освобождаем память. }
for i:=1 to N do
freemem(b[i],M*sizeof(real));
end.
```

Результаты работы программы представлены на рис. 6.35.

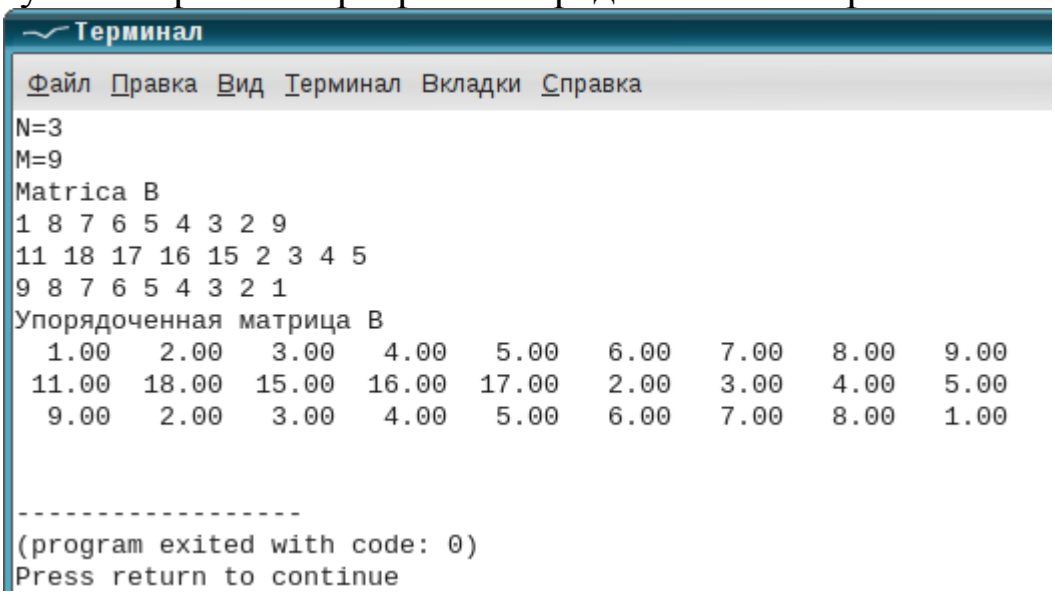

Рисунок 6.35: Результаты решения задачи 6.12

Динамическая матрица может быть достаточно большой, фактически ее размер ограничен только объемом свободной памяти.

6.4 Задачи для самостоятельного решения

1. Определить номера строки и столбца максимального простого числа прямоугольной матрицы `A(n,m)`. Подсчитать количество нулевых элементов матрицы и напечатать их индексы.

2. Найти среднее геометрическое значение элементов квадратной матрицы `X(n,n)`, находящихся по периметру этой матрицы и на ее диагоналях, если это возможно. Если среднее геометрическое вычислить невозможно, то поменять местами максимальный и минимальный элементы матрицы.

3. Сформировать вектор D, каждый элемент которого представляет собой среднее арифметическое значение элементов строк матрицы C(k,m), и вектор G – любой его компонент должен быть равен произведению элементов соответствующего столбца матрицы C.

4. Задана матрица A(n,m), в каждом столбце которой максимальный элемент необходимо заменить произведением отрицательных элементов этого же столбца.

5. Задана матрица A(n,n). Определить максимальный элемент среди элементов матрицы, расположенных выше главной диагонали, и минимальный элемент среди тех, что находятся ниже побочной диагонали. После этого выполнить сортировку каждого столбца матрицы по возрастанию.

6. Заменить строку матрицы P(n,m) с минимальной суммой элементов на строку, где находится максимальный элемент матрицы.

7. Переместить максимальный элемент матрицы F(k,p) в правый верхний угол, а минимальный элемент – в левый нижний.

8. Проверить, является ли матрица A(n,n) диагональной (все элементы нули, кроме главной диагонали), единичной (все элементы нули, на главной диагонали только единицы) или нулевой (все элементы нули).

9. Сформировать из некоторой матрицы A(n,n) верхнетреугольную матрицу B(n,n) (все элементы ниже главной диагонали нулевые), нижнетреугольную матрицу C(n,n) (все элементы выше главной диагонали нулевые) и диагональную матрицу D(n,n) (все элементы нули, кроме главной диагонали) .

10. Заданы матрицы A(m,n) и B(n,m). Найти матрицу $C=(A\cdot B)^4$.

11. Проверить, является ли матрица B(n,n) обратной к A(n,n). Произведением матриц A и B в этом случае должна быть единичная матрица.

12. Определить количество простых чисел, расположенных вне диагоналей матрицы B(n,n).

13. Проверить, лежит ли на главной диагонали максимальный отрицательный элемент матрицы A(n,n).

14. Переписать простые числа из матрицы A в массив B. Массив упорядочить по убыванию.

15. Переписать положительные числа из матрицы целых чисел A в массив B. Из массива B удалить числа, в двоичном представлении которых единиц больше, чем нулей.

16. Даны четыре квадратные матрицы A(n,n), B(n,n), C(n,n), D(n,n), в которых хранятся целые числа. Найти матрицу, в которой находится максимальное простое число.

17. Заданы четыре квадратные матрицы A(n,n), B(n,n), C(n,n), D(n,n), в которых хранятся целые числа. Найти матрицы, в которых на диагоналях есть простые числа.

18. Заданы три прямоугольные матрицы A(n,m), B(r,p), C(k,q). Найти матрицы, в которых по периметру расположены только отрицательные числа.

19. Проверить, лежит ли на побочной диагонали минимальный положительный элемент матрицы A(n,n).

20. Заданы матрицы D(n,n), A(m,n) и B(n,m). Найти матрицу C=(B·A). Проверить, является ли матрица C(n,n) обратной к D(n,n). Произведением матриц C и D в этом случае должна быть единичная матрица.

21. Заданы четыре квадратные матрицы A(n,n), B(n,n), C(n,n), D(n,n), в которых хранятся целые числа. Найти, в какой из матриц на побочной диагонали есть числа, состоящие из восьмерок.

22. Заменить столбец матрицы P(n,m) с максимальной суммой элементов на столбец, где находится максимальное число, состоящее из единиц.

23. Заданы четыре квадратные матрицы A(n,n), B(n,n), C(n,n), D(n,n), в которых хранятся целые числа. Определить, есть ли среди них матрицы, в которых на побочной диагонали находятся только числа, состоящие из единиц и двоек.

24. Переписать простые числа из матрицы целых чисел A в массив B. Из массива B удалить числа, расположенные между максимальным и минимальным элементами.

25. В матрице целых чисел A(n,n) упорядочить те строки, в которых диагональные элементы не содержат семерок.