5 Использование языка Free Pascal для обработки массивов

5.1 Общие сведения о массивах

В предыдущих главах мы рассматривали задачи, в которых использовались скалярные переменные. Однако при обработке однотипных данных (целочисленных значений, строк, дат и т.п.) оказывается удобным использовать массивы. Например, можно создать массив для хранения значений температуры в течение года. Вместо создания множества (365) переменных для хранения каждой температуры, например `temperature1`, `temperature2`, `temperature3`, ... `temperature365` можно использовать один массив с именем `temperature`, где каждому значению будет соответствовать порядковый номер (рис. 5.1).

№ элемента	1	2	3	4	...	364	365
temperature	-1.5	-3	-6.7	1		2	-3

Рисунок 5.1: Массив значений температур

Таким образом, можно дать следующее определение.

Массив – структурированный тип данных, состоящий из фиксированного числа элементов одного типа.

Массив, представленный на рисунке 5.2, имеет 7 элементов, каждый элемент сохраняет число вещественного типа. Элементы в массиве пронумерованы от 1 до 7. Такого рода массив, представляющий собой просто список данных одного и того же типа, называют простым, или одномерным массивом. Для доступа к данным, хранящимся в определенном элементе массива, необходимо указать имя массива и порядковый номер этого элемента, называемый индексом.

1-й элемент массива	2-й элемент массива	3-й элемент массива	4-й элемент массива	5-й элемент массива	6-й элемент массива	7-й элемент массива
-1.5	-3.913	13.672	-1.56	45.89	4.008	-3.61

Рисунок 5.2: Одномерный массив из 7 вещественных чисел

Если возникает необходимость хранения данных в виде таблиц, в формате строк и столбцов, то необходимо использовать многомерные массивы.

На рисунке 5.3 приведен пример массива, состоящего из трех строк и четырех столбцов. Это *двумерный массив*. Строки в нем мож-

но считать первым измерением, а столбцы — вторым. Для доступа к данным, хранящимся в этом массиве, необходимо указать имя массива и два индекса, первый должен соответствовать номеру строки, а второй — номеру столбца, в которых хранится необходимый элемент.

		Номера столбцов			
		1	**2**	**3**	**4**
Номера строк	**1**	6.3	4.3	-1.34	5.02
	2	1.1	4.7	8.12	8.5
	3	-2.4	-6.2	11.23	8.18

Рисунок 5.3: Двумерный числовой массив

После общего знакомства с понятием «массив», рассмотрим работу с массивами в языке Free Pascal.

5.2 Описание массивов

Для описания массива служат служебные слова `array of`. Описать массив можно двумя способами:

Ввести новый тип данных, а потом описать переменные нового типа. В этом случае формат оператора `type` следующий:

```
type
имя_типа = array [тип_индекса] of
                                    тип_компонентов;
```

В качестве **типа_индекса** следует использовать перечислимый тип. **Тип_компонентов** — это любой ранее определенный тип данных, например:

```
type
massiv=array[0..12] of real;
//Тип данных massiv из 13 элементов,
//элементы нумеруются от 0 до 12.
dabc=array[-3..6] of integer;
//Тип данных dabc из 10 элементов,
//элементы нумеруются от -3 до 6.
var
x,y:massiv;
z: dabc;
```

Можно не вводить новый тип, а просто описать переменную следующим образом:

```
var
переменная: array [тип_индекса] of
                                  тип_переменной;
```

Например:

```
var
z,x: array[1..25] of word;
//Массивы z и x из 25 значений типа word,
//элементы нумеруются от 1 до 25.
g:array[-3..7] of real;
//Массив g из 11 значений типа real,
//которые нумеруюся от -3 до 7.
```

Для описания массива можно использовать предварительно определенные константы:

```
const
n=10;
m=12;
var
a: array[1..n] of real;
b: array[0..m] of byte;
```

Константы должны быть определены до использования, так как массив не может быть переменной длины!

Двумерный массив (матрицу) можно описать, применив в качестве базового типа (типа компонентов) одномерный:

```
type
massiv=array[1..200] of real;
matrica=array[1..300] of massiv;
var
ab:matrica;
```

Такая же структура получается при использовании другой формы записи:

```
Type
matrica = array [1..300,1..200] of real;
var
ab:matrica;
```

или

```
var ab:array [1..300,1..200] of real;
```

При всех трех определениях мы получали матрицу вещественных чисел, состоящую из 300 строк и 200 столбцов.

Аналогично можно ввести трехмерный массив, или массив большего числа измерений:

```
type
abc=array [1..4,0..6,-7..8,3..11] of real;
var b:abc;
```

5.3 Операции над массивами

Для работы с массивом как с единым целым надо использовать имя массива (без указания индекса в квадратных скобках). Для доступа к элементу массива необходимо указать имя массива и в квадратных скобках порядковый номер элемента массива, например x[1], y[5], c[25], A[8].

В языке Free Pascal определена операция присваивания для массивов, идентичных по структуре (с одинаковыми типами индексов и компонентов). Например, если массивы `C` и `D` описаны как

```
var C,D: array [0..30] of real;
```

то можно записать оператор `C:=D;`. Такая операция сразу всем элементам массива `C` присвоит значения соответствующих им по номерам элементов массива `D`.

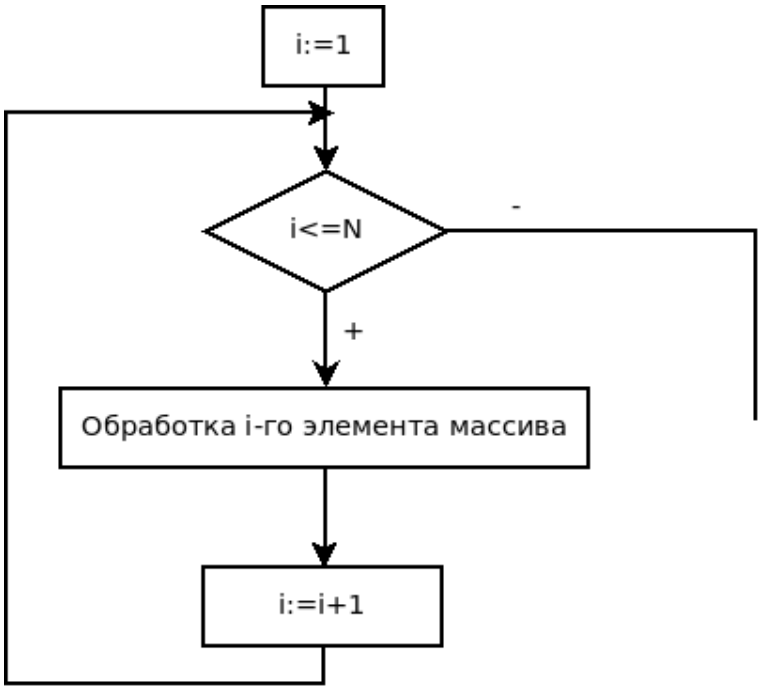

Рисунок 5.4: Блок-схема обработки элементов массива с использованием цикла с предусловием

Выполнение любой другой операции над массивом надо организовывать поэлементно, для чего необходимо организовать цикл, в котором последовательно обрабатывать элементы массива, сначала обрабатываем первый элемент массива, затем второй, третий, …, n-й (рис. 5.4-5.5).

Для обработки элементов массива удобно использовать цикл `for`.

Рисунок 5.5: Блок-схема обработки элементов массива с использованием цикла с for

5.4 Ввод-вывод элементов массива

Язык Free Pascal не имеет специальных средств ввода-вывода всего массива, поэтому данную операцию следует организовывать поэлементно.

При вводе массива необходимо последовательно вводить 1-й, 2-й, 3-й и т.д. элементы массива, аналогичным образом поступить и при выводе. Следовательно, как для ввода, так и для вывода необходимо организовать стандартный цикл обработки массива (рис. 5.6 - 5.7).

Для обращения к элементу массива необходимо указать имя массива и в квадратных скобках номер элемента, например `X[5]`, `b[2]` и т.д.

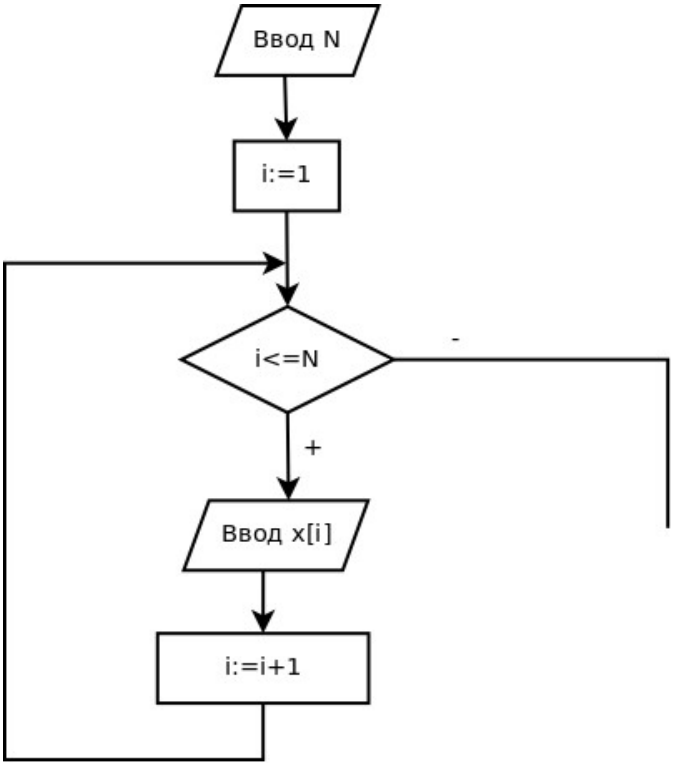

Рисунок 5.6: Алгоритм ввода массива X с использованием цикла с предусловием

5.4.1 Организация ввода-вывода

Реализуем эти алгоритмы в консольных приложениях.

```
/Ввод элементов массива X с помощью цикла while.
var x: array [1..100] of real;
    i,n: integer;
begin
```

```
    writeln ('введите размер массива'); readln(N);
    i:=1;
    while (i<=N) do
    begin
        write('x(',i,')= ');
        readln(x[i]);
        i:=i+1
    end;
end.
//Ввод элементов массива X с помощью цикла for.
var i,n: integer;
    x: array [1..100] of real;
begin
    readln(N);
    for i:=1 to N do
    begin
        write('x(',i,')= ');
        readln(x[i])
    end;
end.
```

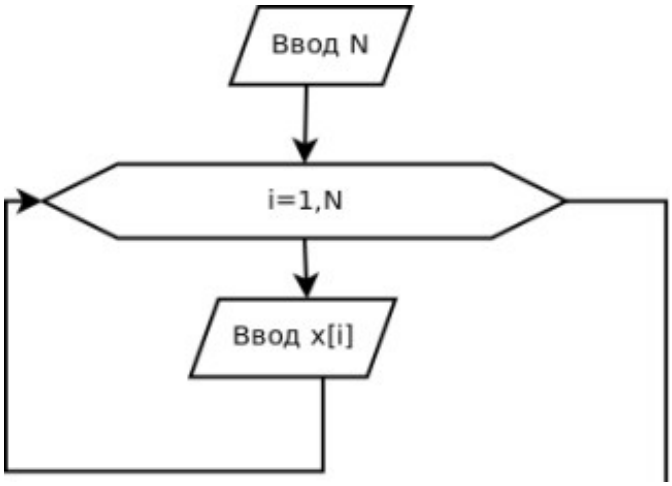

Рисунок 5.7: Алгоритм ввода массива X с использованием блока модификации

Цикл `for … do` удобнее использовать для обработки всего массива, и в дальнейшем при выполнении подобных операций с массивами мы чаще будем применять именно его.

Вывод массива организуется аналогично вводу, только вместо блока ввода элемента массива будет блок вывода.

Предлагаем читателю рассмотреть несколько вариантов вывода массива вещественных чисел

`a=(1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8),`

самостоятельно разобраться, чем они отличаются друг от друга, и решить, какой из вариантов удобнее в конкретной задаче.

```
//Вариант 1
for i: = 1 to n do
        write (a[i]:3:1);
```

Результат первого варианта вывода массива на экран представлен на рис. 5.8. Обратите внимание, что между числами в данном варианте отсутствует пробел.

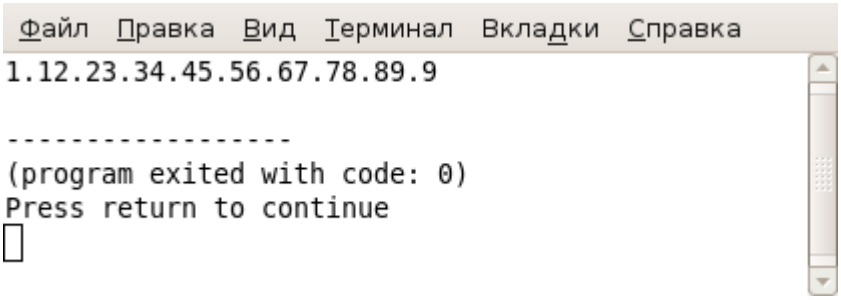

Рисунок 5.8: Результат первого варианта вывода массива

```
//Вариант 2
for i: = 1 to n do
            write (a[i]:6:2);
```

Результат второго варианта вывода массива на экран представлен на рис. 5.9

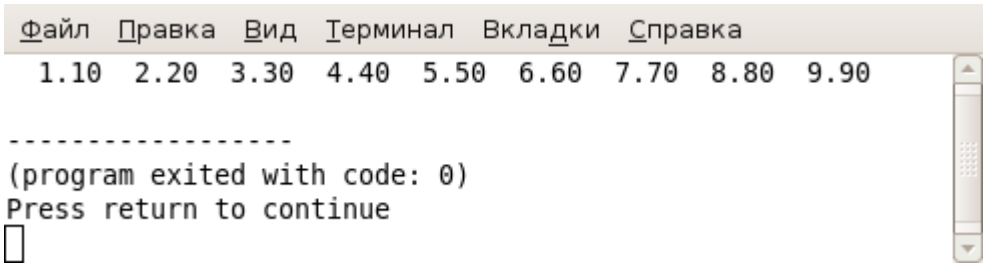

Рисунок 5.9: Результат второго варианта вывода массива

```
// Вариант 3
for i: = 1 to n do
            write (a[i]:3:1,' ');
```

Результат третьего варианта вывода массива на экран представлен на рис. 5.10.

```
// Вариант 4
writeln ('массив A');
for i:=1 to n do
            writeln(a[i]:6:2);
```

Результат четвертого варианта вывода массива на экран представлен на рис. 5.11.

```
// Вариант 5
 for i:=1 to n do
        write ('a(',i,')=',a[i]:3:1,' ');
```

Результат пятого варианта вывода массива на экран представлен на рис. 5.12.

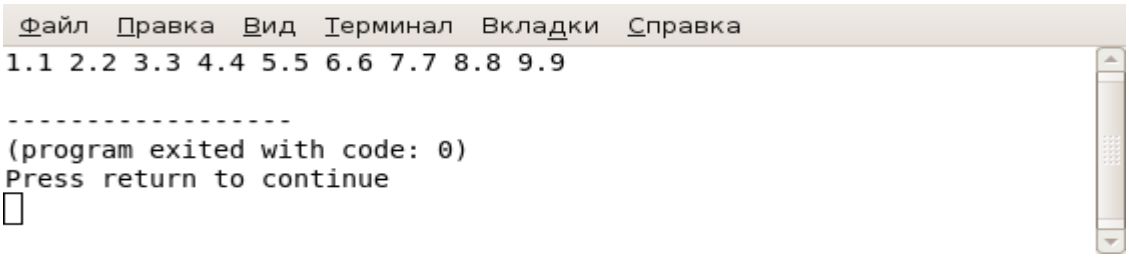

Рисунок 5.10: Результат третьего варианта вывода массива

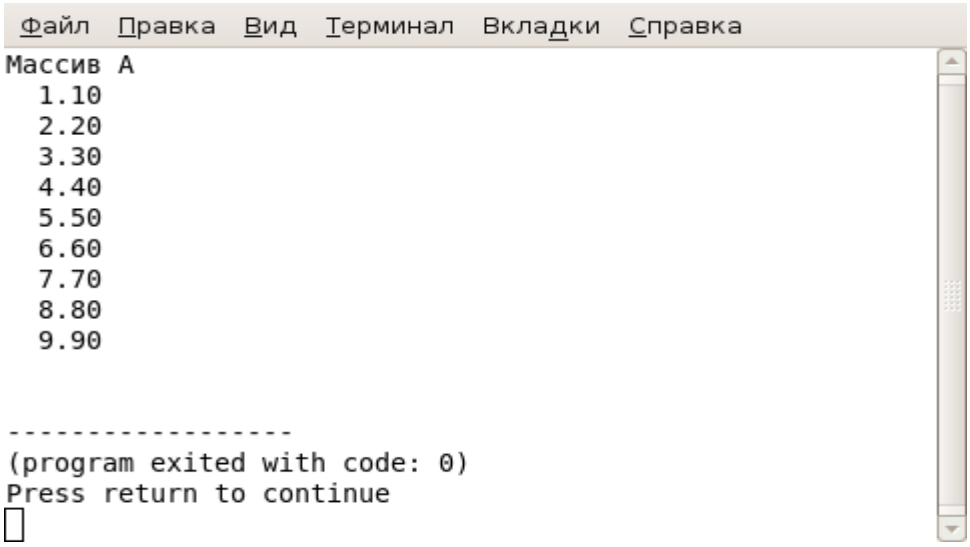

Рисунок 5.11: Результат четвертого варианта вывода массива

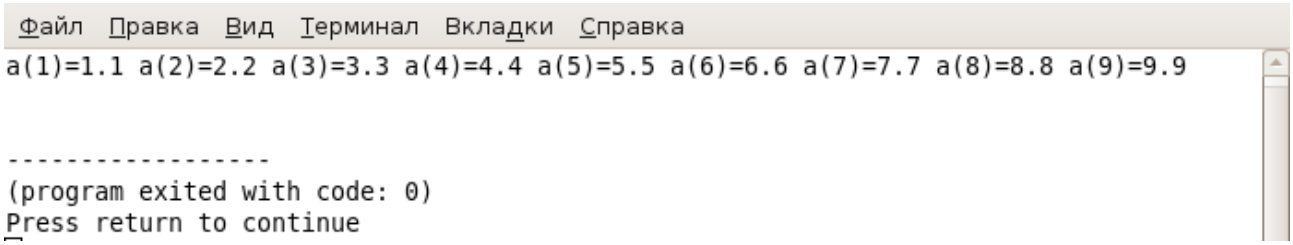

Рисунок 5.12: Результат пятого варианта вывода массива

```
// Вариант 6
for i:=1 to n do
    writeln ('a(',i, ')= ',a[i]:6:2);
```

Результат шестого варианта вывода массива на экран представлен на рис. 5.13.

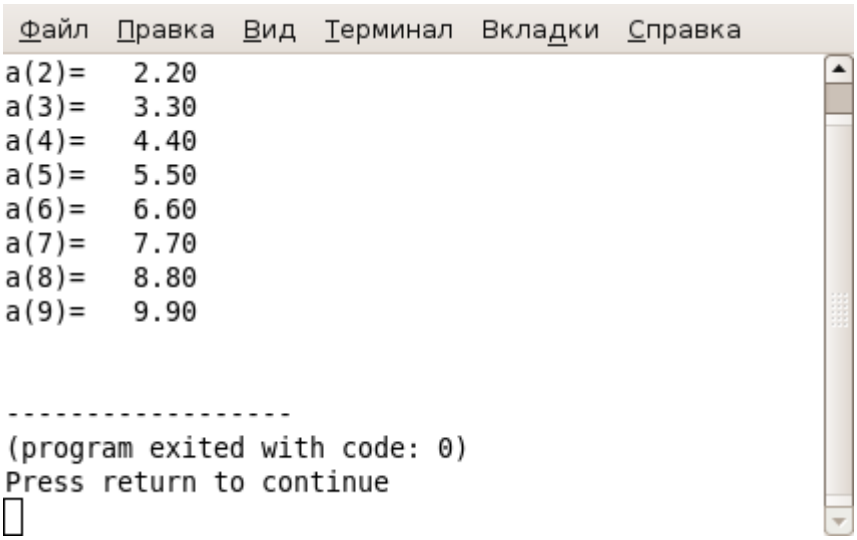

Рисунок 5.13: Результат шестого варианта вывода массива

5.4.2 Ввод-вывод данных в визуальных приложениях

Рассмотрим возможности организации ввода-вывода массивов в визуальных приложениях, для вывода массивов можно использовать стандартный компонент типа `TEdit`.

Рассмотрим эти возможности на примере стандартной задачи вывода массива вещественных чисел

`a=(1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8).`

Расположим на форме кнопку (компонент типа `TButton`) и компонент типа `TEdit` (см. рис. 5.14).

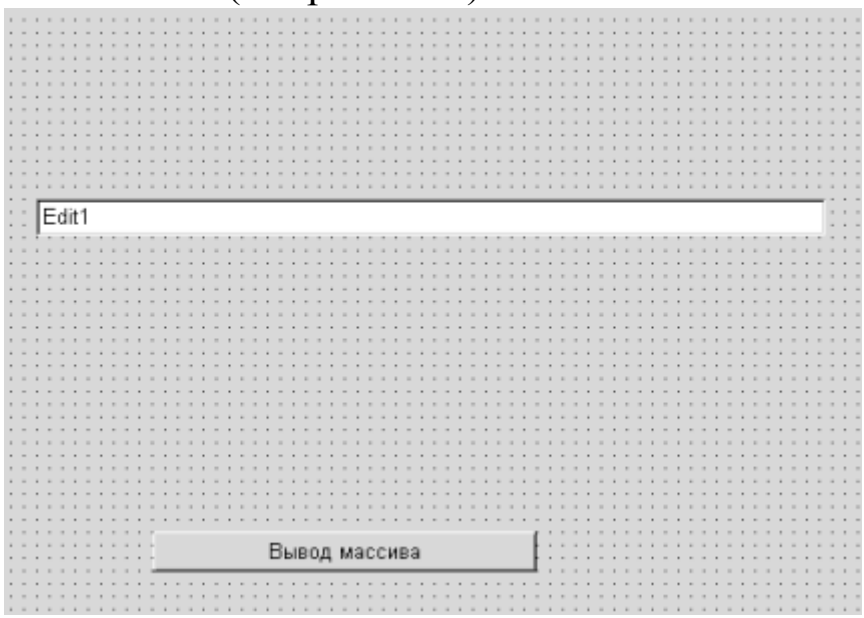

Рисунок 5.14: Форма для задачи вывода массива вещественных чисел

В табл. 5.1 - 5.2 приведены свойства компонентов типа TButton и TEdit.

Таблица 5.1: Свойства компонента Edit1

Свойство	Name1	Text	Height	Left	Top	Width	ReadOnly
Значение	Edit1	' '	23	103	184	492	True

Таблица 5.2: Свойства компонента Button1

Свойство	Name1	Caption	Height	Left	Top	Width
Значение	label1	Button1	25	177	392	239

Наше приложение по щелчку по кнопке «**Вывод массива**» будет выводить массив a в поле ввода Edit1. Алгоритм вывода массива заключается в следующем: каждое число переводится в строку с помощью функции FloatToStr, после чего полученная строка добавляется к полю вывода. Текст обработчика события Button1Click с комментариями приведен ниже.

```
procedure TForm1.Button1Click(Sender: TObject);
var
//Исходный массив a.
a:array [1..8] of
 real=(1.1,2.2,3.3,4.4, 5.5, 6.6, 7.7, 8.8);
 i:integer;
//В переменной n хранится
//количество элементов в массиве
//вещественных чисел.
 n:integer=8;   S:string='';
begin
 Edit1.Text:='';
//Цикл for для последовательной
//обработки всех элементов массива a.
     for i:=1 to n do
//Добавление в поле ввода Edit1
//строки, которая получена из
//элемента массива a[i].
 Edit1.Text:=Edit1.Text+FloatToStr(a[i])+' ';
end;
```

После щелчка по кнопке «**Вывод массива**» окно приложения станет подобным тому, как представлено на рис. 5.15.

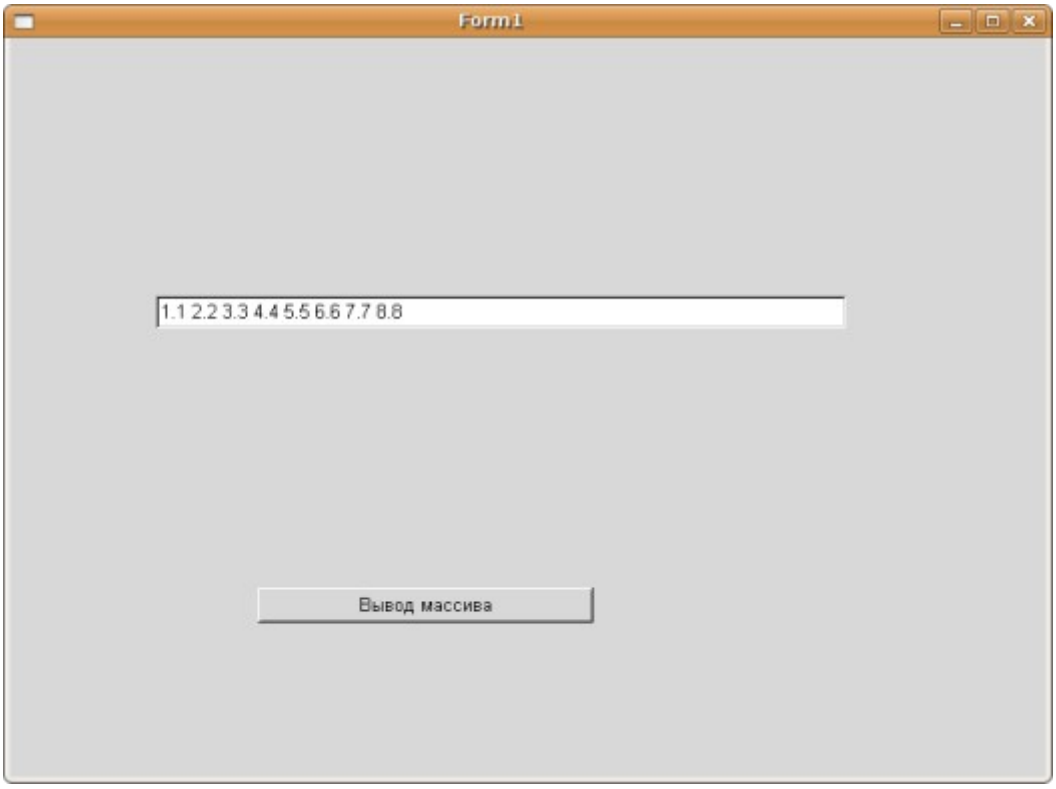

Рисунок 5.15: Вывод массива в поле ввода

Для ввода массивов в Lazarus в простейшем случае можно воспользоваться функцией `InputBox`. В качестве примера рассмотрим проект, в котором будет осуществляться ввод массива при щелчке по кнопке. На форме расположим единственную кнопку. Текст обработчик события `Button1Click` с комментариями приведен ниже.

```
procedure TForm1.Button1Click(Sender: TObject);
var i,n:byte; X:array [1..20] of real;
begin
//Количество элементов массива.
n:=StrToInt(InputBox('Ввод элементов
                     массива','n=','7'));
for i := 1 to n do     //Поэлементный ввод.
//Ввод очередного элемента массива
X[i]:=StrToFloat(InputBox('Ввод элементов
        массива','Введите '+IntToStr(i)+ '
                              элемент','0,00'));
end;
```

При щелчке по кнопке на экране появится окно для ввода размера массива (см. рис. 5.16). После корректного ввода размера массива последовательно будут появляться диалоговые окна для ввода очередного элемента, подобные представленному на рис. 5.17.

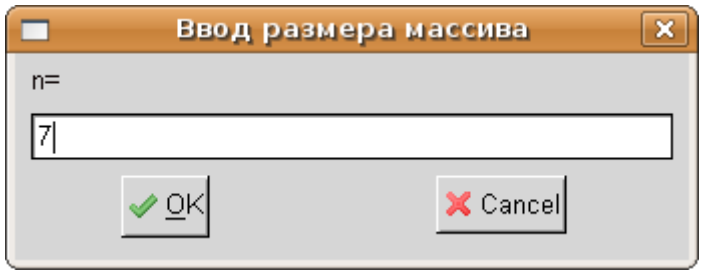

Рисунок 5.16: Ввод размера массива

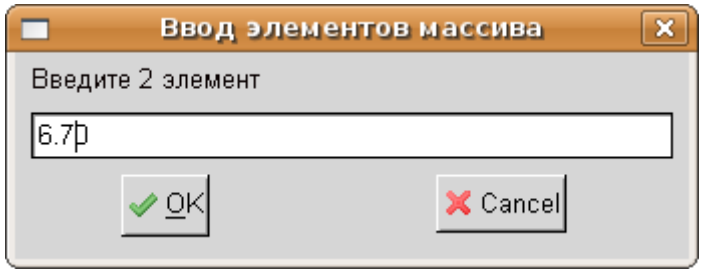

Рисунок 5.17: Ввод второго элемента массива

Для вывода массива с помощью диалогового окна можно применить функцию `MessageDlg`:

```
for i:= 1 to n do
 MessageDlg('X['+IntToStr(i)+']=
 '+FloatToStr(X[i]), MtInformation, [mbOk],0),
```

которая будет открывать отдельное окно для каждого элемента (см. рис. 5.18).

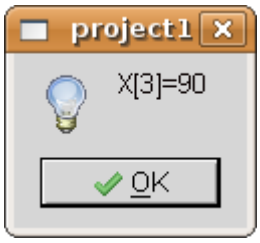

Рисунок 5.18: Вывод третьего элемента массива

Чтобы у пользователя была возможность просматривать элементы массива одновременно, можно из них сформировать строку, а затем вывести ее, например, на форму в виде метки или в окне сообщения:

```
var
i,n:byte;
X:array  [1..20]  of  real;
S:string;
begin
//В переменную строкового типа записывается
//пустая строка.
   S:='';
   for i:=1 to n do
```

```
//Выполняется слияние элементов массива,
//преобразованных в строки и символов пробела -
//результат строка, в которой элементы массива
//перечислены через пробел.
S:=S+FloatToStrF(X[i],ffFixed,5,2)+' ';
//Вывод строки на форму в виде метки.
Label2.Caption:=S;
//Аналогично строку можно вывести в виде
//сообщения, используя функцию
//MessageDlg(S,MtInformation,[mbOk],0);
```

Наиболее универсальным способом ввода-вывода как одномерных, так и двумерных массивов является компонент типа `TStringGrid` («таблица строк»). Познакомимся с этим компонентом подробнее.

На рис. 5.19 представлен проект формы, на которой расположен компонент типа `TStringGrid`[57] (таблица строк). Ячейки таблицы строк предназначены для чтения или редактирования данных. Этот компонент может служить как для ввода, так и для вывода массива.

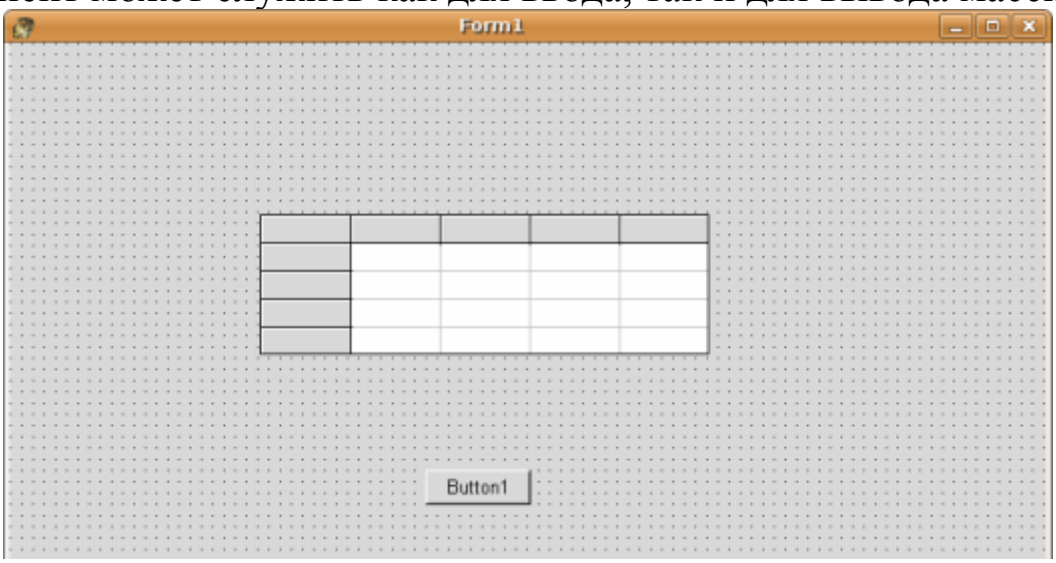

Рисунок 5.19: Форма с компонентом "таблица"

Основные свойства этого компонента представлены в таблице 5.3.

Таблица 5.3: Основные свойства компонента типа `TstringGrid`

Свойство	Описание
`Name`	Имя компонента
`ColCount`	Количество столбцов таблицы

57 Компонент типа `TStringGrid` расположен на странице `Additional`.

Свойство	Описание
RowCount	Количество строк таблицы
Cells	Двумерный массив, в котором хранится содержимое таблицы. Ячейка таблицы, находящаяся на пересечении столбца номер col и строки номер row, определяется элементом Cells [col,row]; строки в компоненте нумеруются от 0 до RowCount-1, (столбцы от 0 до ColCount-1)
FixedCols	Количество зафиксированных слева столбцов таблицы, которые выделяются цветом и при горизонтальной прокрутке таблицы остаются на месте
FixedRows	Количество зафиксированных сверху строк таблицы, которые выделяются цветом и при вертикальной прокрутке таблицы остаются на месте
ScrollBars	Параметр определяет наличие полос прокрутки, возможны следующие значения параметра: •ssNone — отсутствие полос прокурутки (пользователь может в этом случае перемещаться только с помощью курсора) •ssHorizontal, ssVertical или ssBoth — наличие горизонтальной, вертикальной или обеих полос прокрутки; •ssAutoHorizontal, ssAutoVertical или ssAutoBoth — появление горизонтальной, вертикальной или обеих полос прокрутки по мере необходимости.
Options.goEditing	Логическая переменная, которая определяет, может пользователь (True) или нет (False) редактировать содержимое ячеек таблицы
Options.goTab	Логическая переменная, которая разрешает (True) или запрещает (False) использование клавиши **Tab** для перемещения курсора в следующую ячейку таблицы
DefaultColWidth	Ширина столбцов таблицы
DefaultRowHeight	Высота строк таблицы

Свойство	Описание
GridLineWidth	Ширина разграничительных линий между ячейками таблицы
Left	Расстояние от таблицы до левой границы формы
Top	Расстояние от таблицы до верхней границы формы
DefaultColWidth	Ширина столбцов таблицы
DefaultRowHeight	Высота строк таблицы
Height	Высота компонента типа TStringGrid
Width	Ширина компонента типа TStringGrid
Font	Шрифт, которым отображается содержимое ячеек таблицы

Рассмотрим использование компонента для ввода-вывода массивов на примере программы, с помощью которой можно осуществить ввод массива из восьми вещественных чисел, а затем вывести его в обратном порядке.

Разместим на форме две метки, два компонента типа TStringGrid и одну кнопку. Свойства компонентов StringGrid1 и StringGrid2 можно задать такими же, как показано в табл. 5.4.

Таблица 5.4: Основные свойства компонента типа TStringGrid

Свойство	StringGrid1	StringGrid2	Описание
ColCount	8	8	Количество столбцов таблицы
RowCount	1	1	Количество строк таблицы
FixedCols	0	0	Количество зафиксированных слева столбцов таблицы
FixedRows	0	0	Количество зафиксированных сверху строк таблицы

Свойство	StringGrid1	StringGrid2	Описание
`Options.goEditing`	True	False	Логическая переменная, которая определяет возможность пользователю редактировать содержимое ячеек таблицы
`Left`	186	186	Расстояние от таблицы до левой границы формы
`Top`	72	216	Расстояние от таблицы до верхней границы формы
`Height`	24	24	Высота компонента
`Width`	518	518	Ширина компонента

Окно формы приложения ввода—вывода массива будет подобным представленному на рис. 5.20.

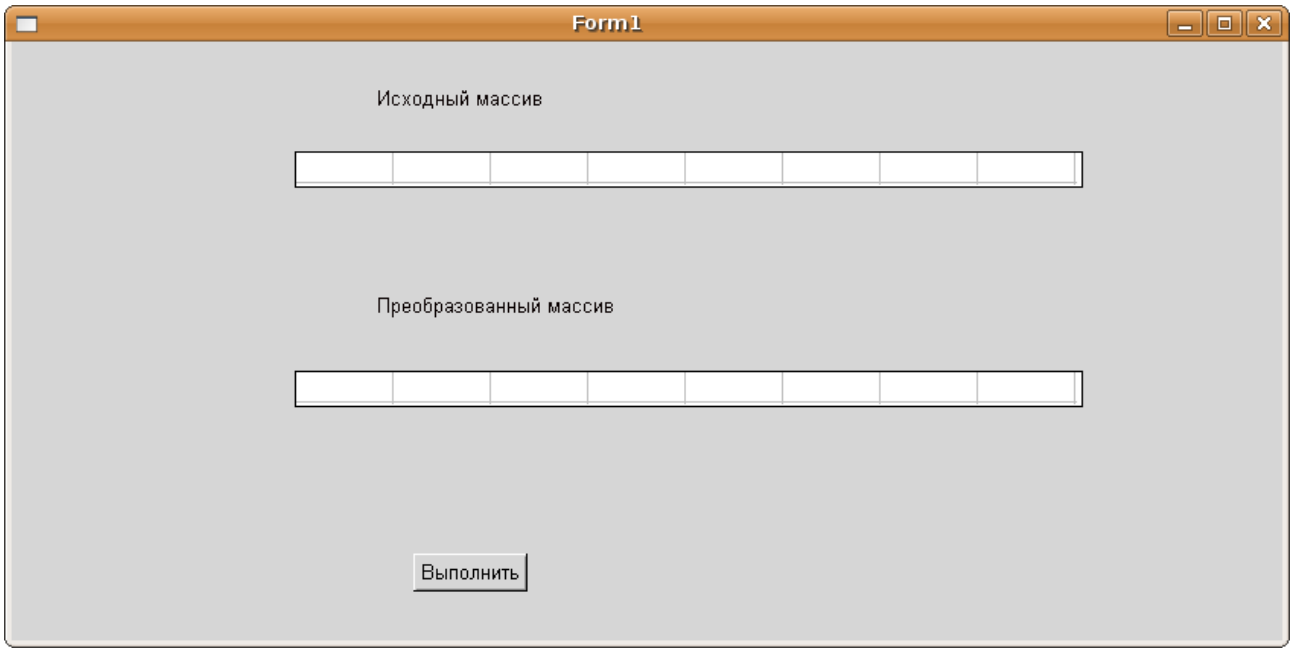

Рисунок 5.20: Окно формы приложения

В первую таблицу будем вводить элементы массива, во вторую выводить преобразованный массив. Щелчок по кнопке **Выполнить** вызовет следующую подпрограмму:

```
procedure Tform1.Button1Click(
                                Sender: TObject) ;
```

```
var  n,i:integer;   a:array [0..7] of real;
begin
for i:=0 to 7 do        //Ввод массива.
//Из поля таблицы считывается элемент,
//преобразовывается в число и
//присваивается элементу массива.
a[i]:=StrToFloat(StringGrid1.Cells[i,0]);
for i:=0 to 7 do        //Вывод массива.
//Элемент массива преобразовывается в строку
//и помещается в поле таблицы.
StringGrid2.Cells[i,0]:=
            FloatToStrF(a[7-i],ffFixed,5,2);
end;
```

При запуске программы на выполнение появляется окно приложения, подобное представленному на рис. 5.20, пользователь вводит исходный массив, щелкаем по кнопке **Выполнить**, после чего окно приложения принимает вид, подобный представленному на рис. 5.21.

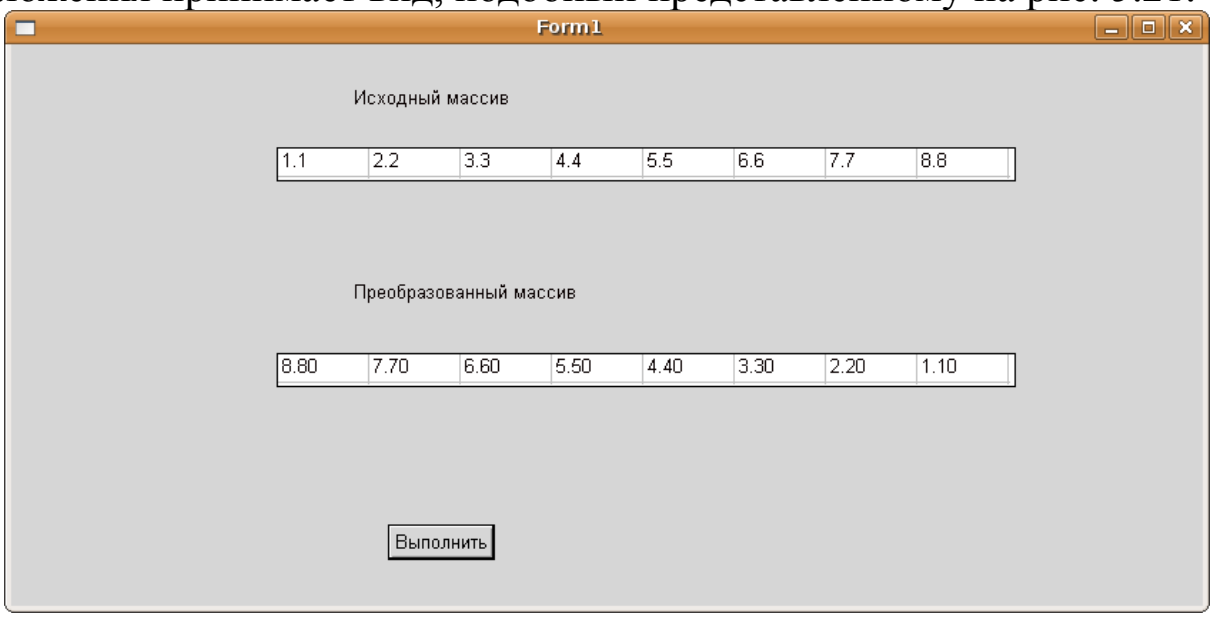

Рисунок 5.21: Окно программы ввода-вывода массива

В этом параграфе были рассмотрены различные способы ввода-вывода как в консольных, так и в визуальных приложениях. В дальнейшем мы будем использовать те из них, которые удобнее при решении конкретной задачи.

Теперь перейдем к рассмотрению основных алгоритмов обработки одномерных массивов, многие из которых аналогичны соответствующим алгоритмам обработки последовательностей (вычисление

суммы, произведения, поиск элементов по определенному признаку, выборки и т. д.). Отличие заключается в том, что в массиве одновременно доступны все его компоненты, поэтому становятся возможными более сложные действия с массивами (например, сортировка элементов массива, удаление и вставка элементов и т.д.).

5.5 Вычисление суммы и произведения элементов массива

Нахождение суммы и произведения элементов массива аналогично подобным алгоритмам нахождения суммы и произведения элементов последовательности.

Дан массив `X`, состоящий из `n` элементов. Найти *сумму элементов* этого массива. Переменной `S` присваивается значение, равное нулю, затем последовательно к переменной `S` добавляются элементы массива `X`.

Блок-схема алгоритма расчета суммы приведена на рис. 5.22. Соответствующий алгоритму фрагмент программы будет иметь вид:

```
s:=0;
for i:=1 to n do s:=s+x[i];
writeln('s=',s:7:3);
```

Найдем *произведение элементов* массива `X`. Решение задачи сводится к тому, что значение переменной `P`, в которую предварительно была записана единица, последовательно умножается на значение `i`–го элемента массива. Блок-схема алгоритма приведена на рис. 5.23.

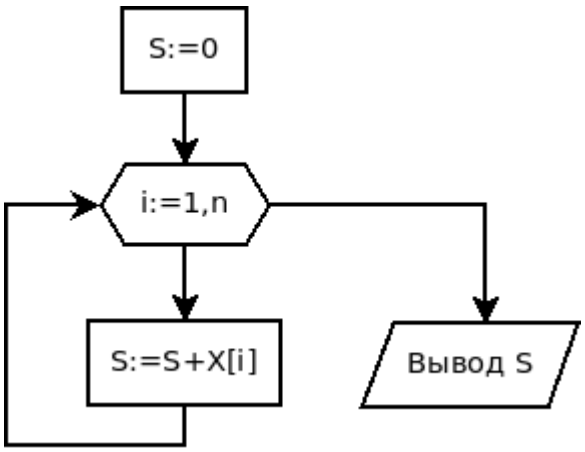

Рисунок 5.22: Алгоритм нахождения суммы элементов массива

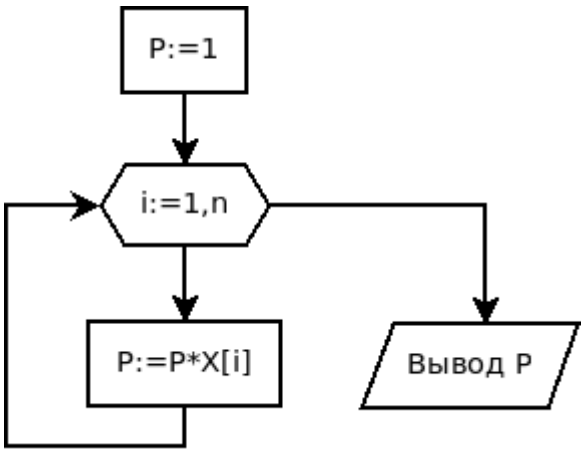

Рисунок 5.23: Алгоритм нахождения произведения элементов массива

Соответствующий фрагмент программы будет иметь вид:

```
p:=1;
for i:=1 to n do p:=p*x[i];
writeln('P=',P:7:3);
```

5.6 Поиск максимального элемента в массиве и его номера

Рассмотрим задачу поиска максимального элемента (Max) и его номера (Nmax) в массиве X, состоящем из n элементов.

Алгоритм решения задачи следующий. Предположим, что первый элемент массива является максимальным, и запишем его в переменную Max, а в Nmax – его номер (число 1).

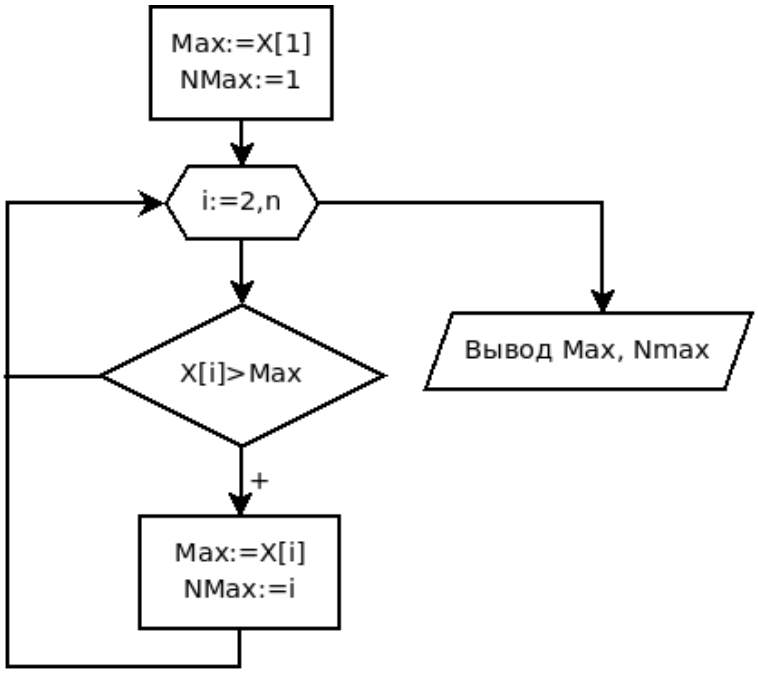

Рисунок 5.24: Алгоритм поиска максимального элемента массива и его номера

Затем все элементы, начиная со второго, сравниваем в цикле с максимальным. Если текущий элемент массива оказывается больше максимального, то записываем его в переменную Max, а в переменную Nmax – текущее значение индекса i. Процесс определения максимального элемента в массиве изображен при помощи блок-схемы на рис. 5.24. Соответствующий фрагмент программы имеет вид:

```
Max:=X[1]; Nmax:=1;
for i:=2 to n do
    if X[i]>Max then
    begin
        Max:=X[i];
        Nmax:=i;
   end;
write(' Max=',Max:1:3,' Nmax=',Nmax);
```

Алгоритм поиска минимального элемента в массиве будет отличаться от приведенного выше лишь тем, что в условном блоке и, соответственно, в конструкции `if` текста программы знак поменяется с > на <.

5.7 Сортировка элементов в массиве

Сортировка представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений. Например, массив `X` из `n` элементов будет отсортирован в порядке возрастания значений его элементов, если

`X[1] ≤ X[2] ≤… ≤ X[n],`

и в порядке убывания, если

`X[1] ≥ X[2] ≥… ≥ X[n].`

Многие алгоритмы сортировки основаны на том факте, что надо переставлять два элемента таким образом, чтобы после перестановки они были правильно расположены друг относительно друга. При сортировке по возрастанию после перестановки элемент с меньшим индексом должен быть не больше элемента с большим индексом[58]. Рассмотрим некоторые из алгоритмов.

5.7.1 Сортировка методом «пузырька»

Наиболее известным методом сортировки является сортировка массивов пузырьковым методом. Ее популярность объясняется запоминающимся названием[59] и простотой алгоритма. Сортировка методом пузырька основана на выполнении в цикле операций сравнения и при необходимости обмена соседних элементов. Рассмотрим алгоритм пузырьковой сортировки на примере *сортировки по возрастанию* более подробно.

Сравним первый элемент массива со вторым, если первый окажется больше второго, то поменяем их местами. Затем сравним второй с третьим, если второй больше третьего, то также поменяем их, далее сравниваем третий и четвертый, и если третий большего четвертого, меняем их местами. После трех этих сравнений самым большим элементом станет элемент с номером `4`. Если продолжить сравнения соседних элементов, сравнить четвертый с пятым, пятый с

58 При сортировке по убыванию после перестановки элемент с меньшим индексом должен быть не меньше элемента с большим индексом.

59 Название алгоритма происходит из-за подобия процессу движения пузырьков в резервуаре с водой, когда каждый пузырек находит свой собственный уровень.

шестым и т. д. до сравнения (n-1)–го и n–го элементов, то в результате этих действий самый большой элемент станет на последнее (n-е) место. Теперь повторим данный алгоритм сначала, с 1-го до n−1 элемента (последний, n-й элемент, рассматривать не будем, так как он уже занял свое место). После проведения данной операции самый большой элемент оставшейся части массива станет на свое (n−1)-е место. Так повторяем до тех пор, пока не упорядочим весь массив.

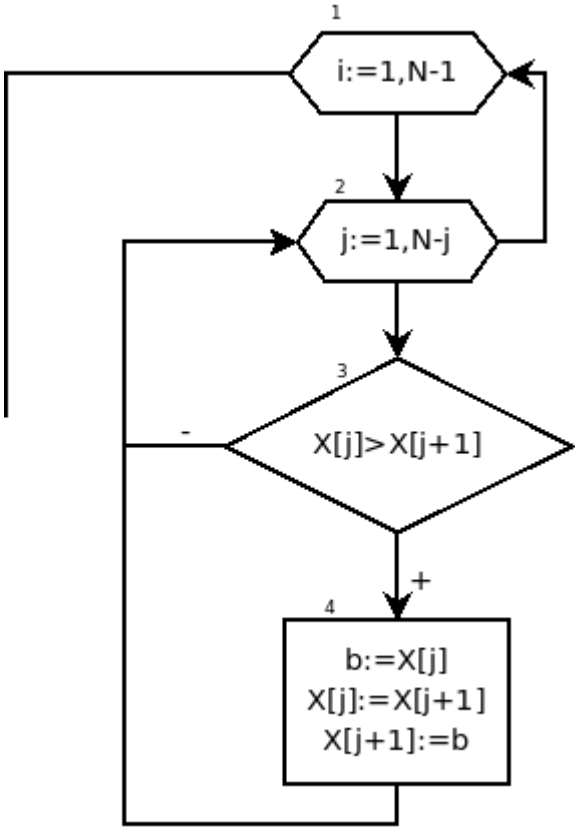

Рисунок 5.25: Алгоритм упорядочивания по возрастанию методом пузырька

В табл. 5.5 подробно показан процесс упорядочивания элементов в массиве.

Нетрудно заметить, что для преобразования массива, состоящего из n элементов, необходимо просмотреть его n−1 раз, каждый раз уменьшая диапазон просмотра на один элемент.

Блок–схема описанного алгоритма приведена на рис. 5.25.

Для обмена двух элементов в массиве (блок 4) используется буферная переменная b, в которой временно хранится значение элемента, подлежащего замене.

Таблица 5.5: Процесс упорядочивания элементов в массиве по возрастанию

Номер элемента	**1**	**2**	**3**	**4**	**5**
Исходный массив	7	3	5	4	2
Первый просмотр	3	5	4	2	7
Второй просмотр	3	4	2	5	7
Третий просмотр	3	2	4	5	7
Четвертый просмотр	2	3	4	5	7

Ниже приведен текст консольного приложения, предназначенного для упорядочивания массива по возрастанию методом пузырька.

На рис. 5.26 приведены результаты работы этой программы.

```
var  i,j,n: byte; b:real;
 X: array [1..100] of real;
begin
 writeln ('введите размер массива ');
 readln (n);
 for i:=1 to n do
 begin
     write('X[',i,']=');
     readln (X[i]);
 end;
 writeln ('массив X ');
 for i:=1 to n do
     write (x[i]:5:2,' ');
 writeln;
 for j:=1 to n-1 do
     for i:=1 to n-j do
         if X[i] > X[i+1] then
         { Если текущий элемент больше
           следующего, то }
         begin  { поменять их местами. }
             b:=X[i];   { Сохранить значение
                          текущего элемента. }
             X[i]:=X[i+1];{Заменить текущий
                          элемент следующим. }
             X[i+1]:=b; {Заменить следующий
                          элемент переменной b.}
         end;
 writeln('упорядоченный массив');
 for i:=1 to n do
     write (X[i]:5:2,' ');
 writeln;
end.
```

Для упорядочивания элементов в массиве по убыванию их значений необходимо при сравнении элементов массива заменить знак > на < (см. блок 3 на рис. 5.25).

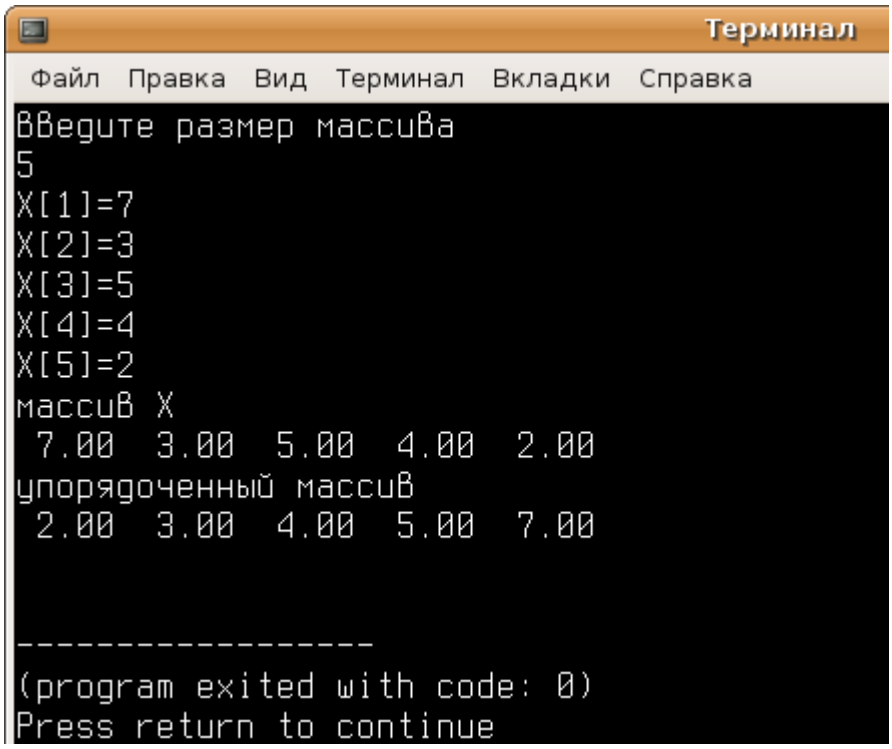

Рисунок 5.26: Результат программы упорядочивания массива по возрастанию

5.7.2 Сортировка выбором

Для сортировки элементов массива по возрастанию (убыванию) можно воспользоваться алгоритмом сортировки выбора максимального (минимального) элемента.

Алгоритм выбором приведен в виде блок-схемы на рис. 5.27. Найдем в массиве самый большой элемент (блоки 2–5) и поменяем его местами с последним элементом (блок 6). После этого максимальный элемент станет на свое место. Теперь надо повторять эти действия (блоки 2–6), уменьшив количество просматриваемых элементов на единицу (блок 7), до тех пор, пока количество рассматриваемых элементов не станет равным одному (блок 8). В связи с тем что мы на каждом шаге уменьшаем количество элементов на `1`, то, чтобы не потерять размер массива (`N`), необходимо в начале алгоритма переписать `N` в переменную `K` (блок 1) и уменьшать уже значение `K`.

При упорядочивании массива по убыванию необходимо перемещать минимальный элемент. Для чего в алгоритме (рис. 5.27) в блоке 4 достаточно знак > поменять на знак <.

Ниже приведен фрагмент программы упорядочения массива по возрастанию с использованием сортировки выбором максимального элемента.

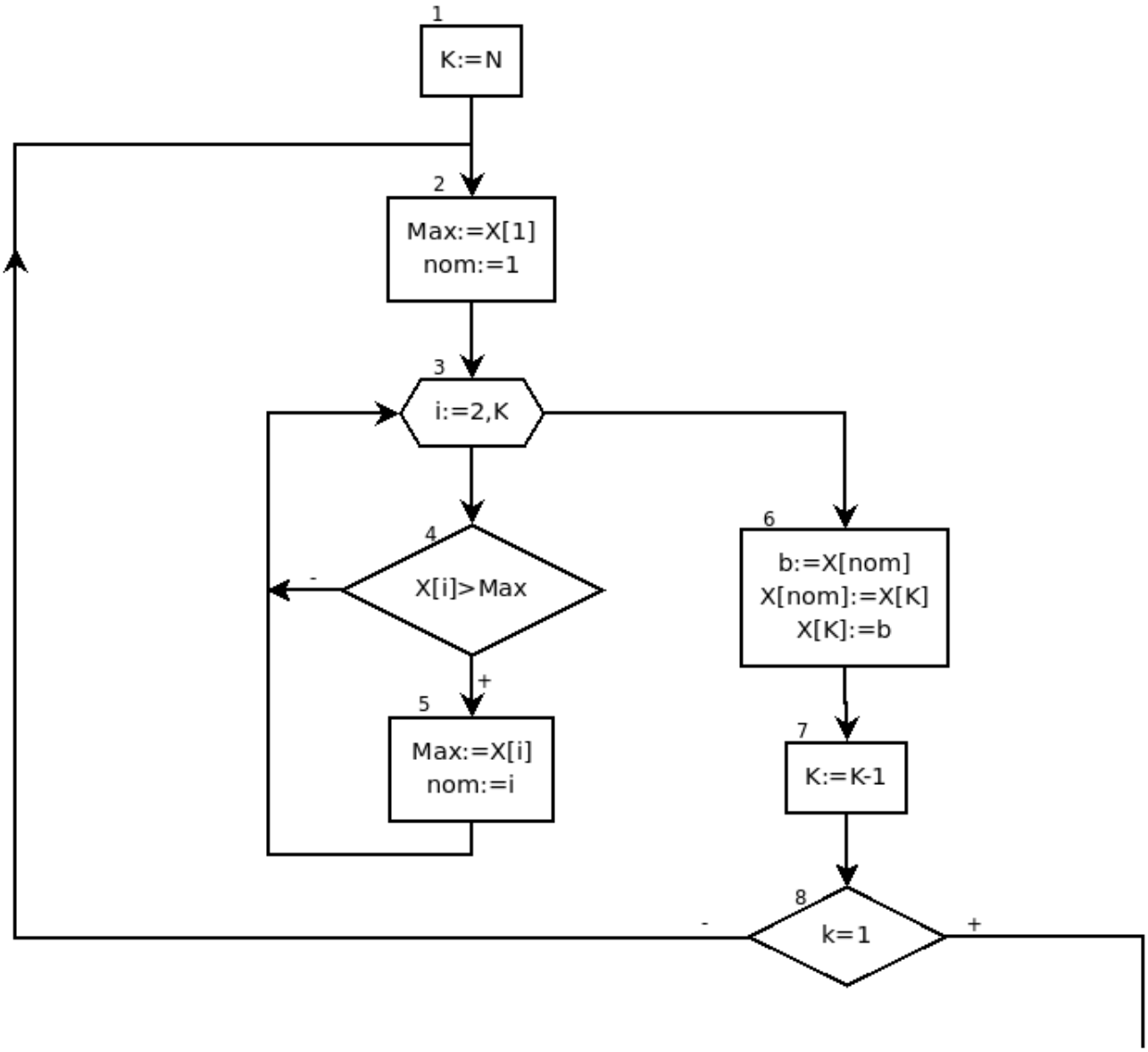

Рисунок 5.27: Сортировка массива по возрастанию выбором наибольшего элемента

```
//Сортировка выбором.
repeat
max:=x[1]; nom:=1;
for i:=2 to k do
 if max < X[i] then
 begin
     max:=X[i]; nom:=i;
 end;
 b:=x[nom]; x[nom]:=x[k];  x[k]:=b;
 k:=k-1;
until k=1;
```

5.8 Удаление элемента из массива

Знакомство с алгоритмом удаления элемента из массива начнем со следующей простой задачи. Необходимо удалить третий элемент из массива X, состоящего из 6 элементов. Алгоритм удаления третьего элемента заключается в том, что на место третьего элемента следует записать четвертый, на место четвертого — пятый, а на место пятого — шестой.

```
X[3]:=X[4]; X[4]:=X[5]; X[5]:=X[6];
```

Таким образом, все элементы с третьего по пятый надо переместить влево на один, на место i-го элемента нужно записать (i+1)-й. Блок-схема алгоритма представлена на рис. 5.28.

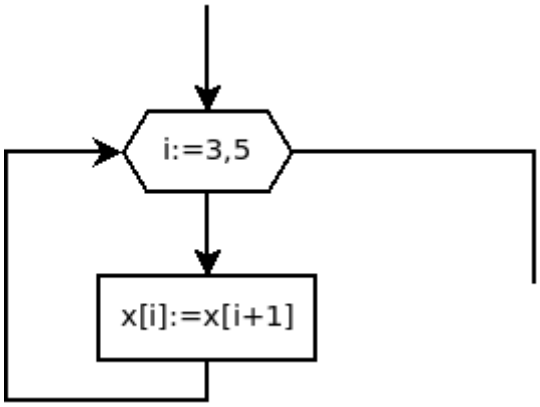

Рисунок 5.28: Алгоритм удаления 3-го элемента из массива

Теперь рассмотрим более общую задачу: необходимо удалить m-й элемент из массива X, состоящего из n элементов. Для этого достаточно записать (m+1)-й элемент на место элемента с номером m, (m+2)-й элемент – на место (m+1)-го и т.д., n-й элемент – на место (n−1)-го. Процесс удаления элемента из массива представлен на рис. 5.29.

Рисунок 5.29: Процесс удаления элемента из массива

Алгоритм удаления из массива X размерностью n элемента с номером m приведен на рис. 5.30.

После удаления элемента[60] из массива, изменится количество элементов в массиве (уменьшается на один) и у части элементов изменяется индекс. Если элемент удален, то на место него приходит следующий, передвигаться к которому (путем увеличения индекса на один) нет необходимости. Следующий элемент сам сдвинулся влево после удаления.

60 А фактически сдвига части массива на один элемент влево.

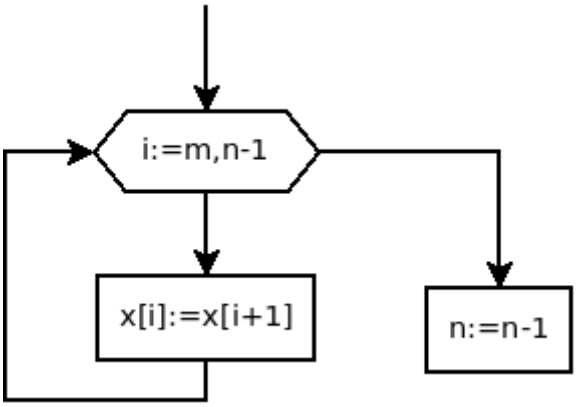

Рисунок 5.30: Алгоритм удаления m-го элемента из массива из n элементов

Если обрабатывается массив, в котором часть элементов удаляется, то после удаления элемента не надо переходить к следующему (при этом уменьшается количество элементов). В качестве примера рассмотрим следующую задачу.

ЗАДАЧА 5.1. Удалить из массива X(n) отрицательные элементы.

Алгоритм решения задачи довольно прост, перебираем все элементы массива, если элемент отрицателен, то удаляем его путем сдвига всех последующих на один влево. Единственное, о чем стоит помнить, что после удаления не надо переходить к следующему для последующей обработки, он сам сдвигается на место текущего. Блок-схема решения задачи 5.1 представлена на рис. 5.31. Ниже представлен текст программы с комментариями. Результаты работы программы представлены на рис. 5.32.

```
program upor_massiv;
var
 i,n,j:byte;
 X: array [1..100] of real;
 begin
 writeln ('введите размер массива ');
 readln (n);
 {Ввод массива.}
 for i:=1 to n do
 begin
     write('X[',i,']=');
     readln (X[i]);
 end;
writeln ('массив X ');
for i:=1 to n do
     write (x[i]:5:2,' ');
writeln;
```

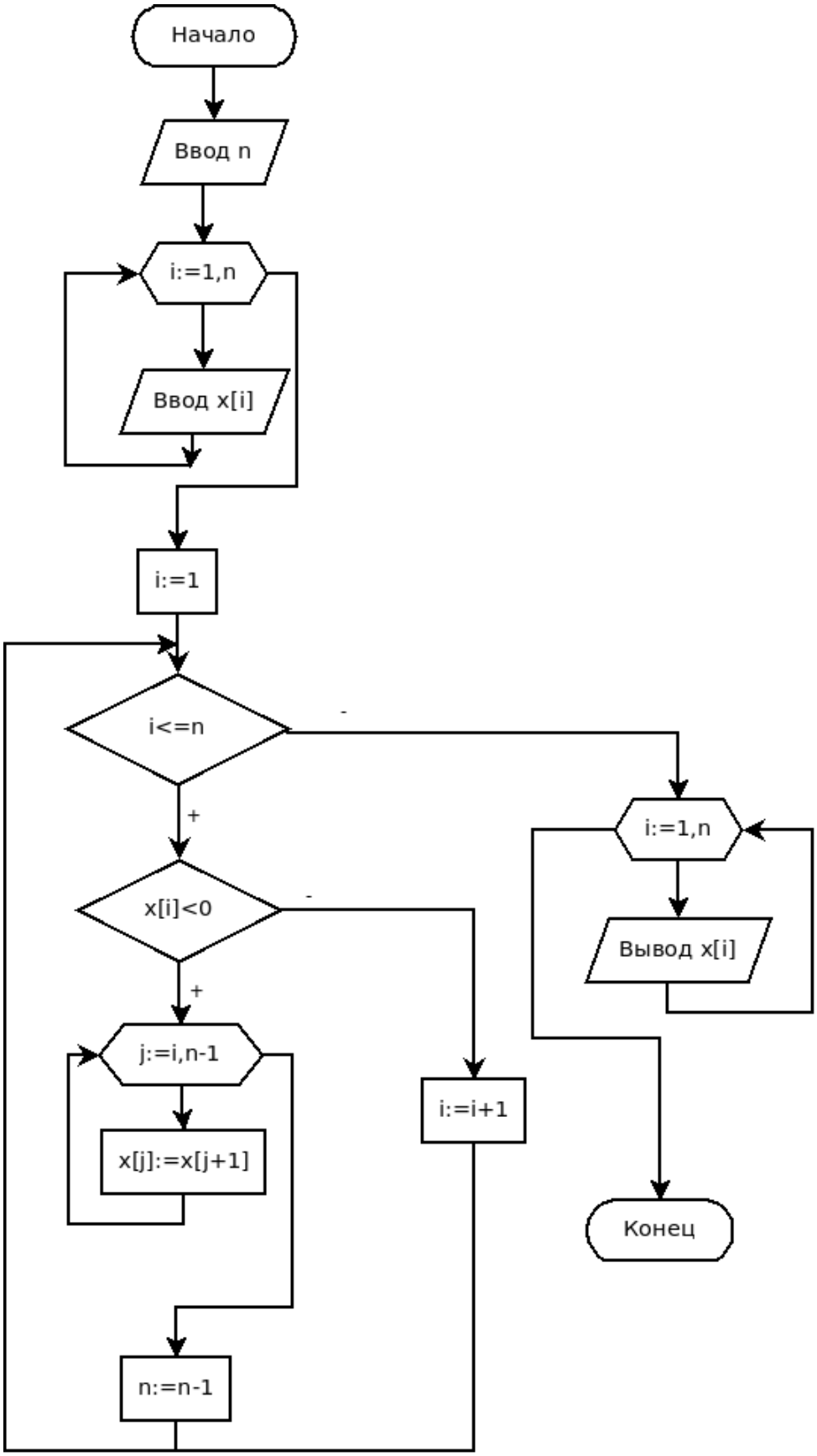

Рисунок 5.31: Блок-схема решения задачи 5.1

```
i:=1;
while(i<=n)do
{Если очередной элемент массива X[i]
отрицателен, то }
 if x[i]<0 then
 begin
{Удаляем элемент массива с номером i.}
     for j:=i to n-1 do x[j]:=x[j+1];
{Уменьшаем размер массива.}
{Не переходим к следующему элементу массива.}
     n:=n-1;
 end
else
{Если элемент не удалялся, то переходим к
следующему элементу массива.}
     i:=i+1;
writeln('Измененный массив:');
for i:=1 to n do write (X[i]:5:2,' ');
writeln;end.
```

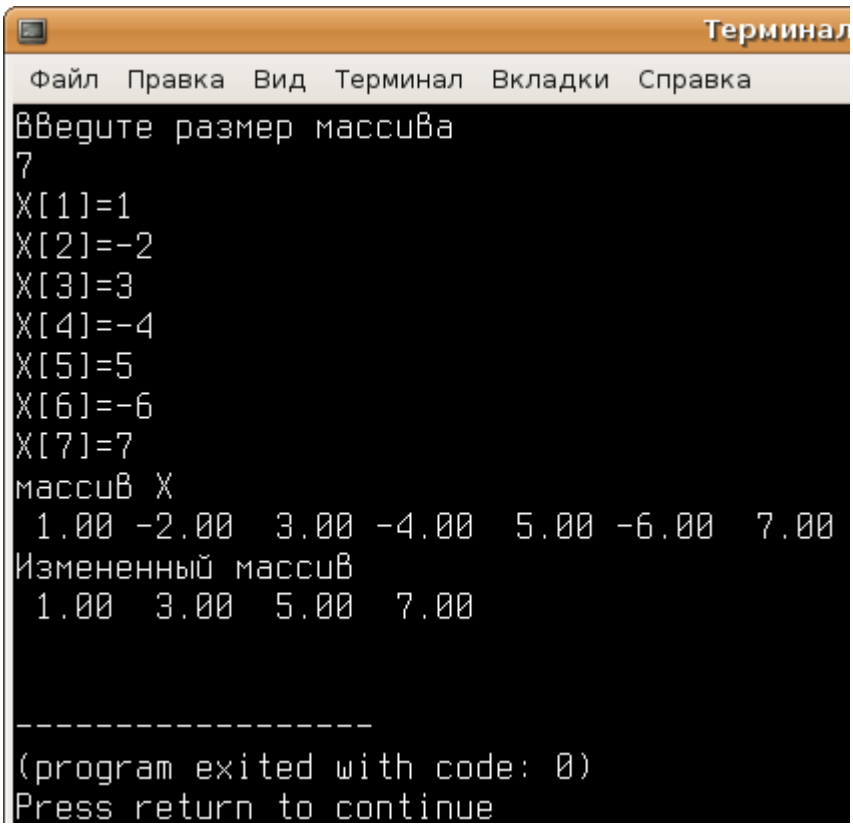

Рисунок 5.32: Результаты работы программы решения задачи 5.1

5.9 Вставка элемента в массив

Рассмотрим несложную задачу: вставить число `b` в массив `X(10)` между третьим и четвертым элементами.

Для решения этой задачи необходимо все элементы, начиная с четвертого, сдвинуть вправо на один элемент. А потом в четвертый элемент массива надо будет записать `b` (`X[4]:=b;`).

Но чтобы не потерять соседнее значение, сдвигать надо сначала десятый на один вправо, затем девятый, восьмой и т. д. до четвертого.

Блок-схема алгоритма вставки приведена на рис. 5.33.

В общем случае блок-схема вставки числа `b` в массив `X(N)` между элементами с номерами `m` и `m+1` представлена на рис. 5.34.

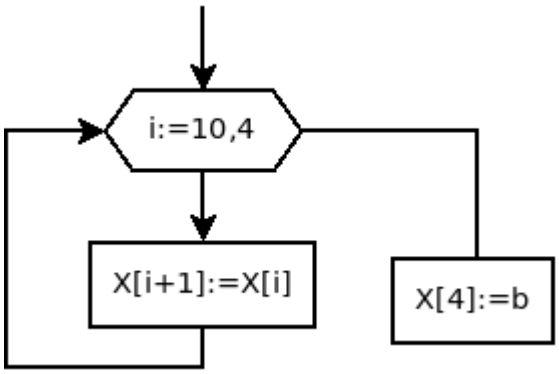

Рисунок 5.33: Вставка числа b между третьим и четвертым элементов массива X

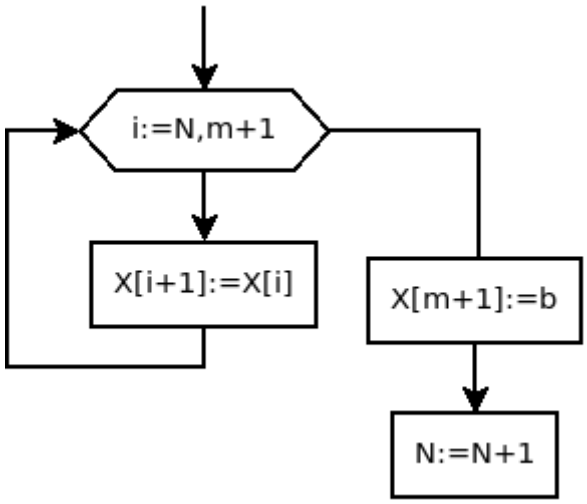

Рисунок 5.34: Вставка числа b в массив X

Ниже представлен фрагмент программы, реализующий этот алгоритм[61].

```
var i,n,m:byte; X: array [1..100] of real;
b:real;
begin
 writeln ('N='); readln (n);
 for i:=1 to n do
 begin
     write('X[',i,']='); readln (X[i]);
 end;
 writeln ('Массив X');
 for i:=1 to n do write (x[i]:5:2,' ');
```

61 При описании массива X необходимо предусмотреть достаточный размер для вставки одного элемента.

```
writeln;
writeln ('m='); readln (m);
writeln ('b='); readln(b);
for i:=n downto m+1 do
    x[i+1]:=x[i];
x[m+1]:=b;
n:=n+1;
writeln('Измененный массив');
for i:=1 to n do write (X[i]:5:2,' ');
writeln;
end.
```

5.10 Использование подпрограмм для работы с массивами

Рассмотрим, как можно передавать массивы в подпрограмму. Как известно (см. главу 4), чтобы объявить переменные в списке формальных параметров подпрограммы, необходимо указать их имена и типы. Однако типом любого параметра в списке может быть только стандартный или ранее объявленный тип. Поэтому для того чтобы передать в подпрограмму массив, необходимо вначале описать его тип[62], а затем объявлять процедуру:

```
type тип_массива= array [список_индексов] of тип;
procedure имя_поцедуры(имя_массива: тип_массива);
    …
```

Например:

```
type
vector=array [1..10] of byte;
matrica=array [1..3, 1..3] of real;
procedure proc(A:matrica;b:vector;var x:vector);
```

Понятно, что передача в подпрограмму строки вида

```
имя_переменной: string[длина_строки];
```

которая фактически является массивом[63], должна осуществляться аналогично:

```
type тип_строки= string [длина_строки];
procedure имя_поцедуры(имя_строки: тип_ строки);
    …
```

62 Тип данных массив, объявление массива см. в п.2.3.9.

63 Тип данных строка, объявление строки см. в п.2.3.9.

Например:

```
type
stroka_5=string[5];
stroka_10=string[10];
function fun(Str: stroka_5):stroka_10;
```

Массивы в подпрограмму можно передавать, используя понятие открытого массива. *Открытый массив* — это массив[64], при описании которого указывается тип составляющих его элементов, но не определяются границы изменения индексов:

```
имя_открытого_массива: array of array of … тип;
```

Например:

```
var
massiv_1: array of real;
massiv_2: array of array of char;
massiv_3: array of array of array of byte;
```

Распределение памяти и указание границ индексов по каждому измерению открытых массивов осуществляется в ходе выполнения программы с помощью функции `SetLength`:

```
SetLength(имя_массива, список_границ_индексов);
```

Для освобождения выделенной памяти нужно выполнить оператор:

```
имя_массива:=NIL;
```

Нижняя граница открытого массива (минимальное значение номера элемента) всегда равна нулю. *Верхняя граница* (максимальное значение номера элемента) возвращается стандартной функцией:

```
high(имя_массива)
```

Открытый массив можно использовать и при обычной обработке массивов в языке Free Pascal. Рассмотрим использование открытого массива на примере простейшей задачи нахождения суммы элементов массива.

```
var
//Описание открытого массива.
 x: array of real;
 s:real;
 i,n:integer;
Begin
write('n='); readln(n);
```

64 Тип данных массив, объявление массива, обращение к массиву см. в п.2.2.9.

```
//Выделяется память для размещения
//n вещественных значений:
SetLength(x,n);
for i:=0 to high(x) do read(x[i]);
s:=0;
for i:=0 to high(x) do s:=s+x[i];
writeln('сумма=',s:7:3);
x:=NIL;     //Освобождение памяти.
end.
```

Открытый массив может быть формальным параметром подпрограммы:

```
procedure   имя_поцедуры(имя_открытого_массива:
array of тип;);
```

Применение открытого массива в подпрограмме позволяет обрабатывать одномерные массивы произвольной длины:

```
//Процедура предназначена для вывода на экран
//сообщений о значениях элементов
//одномерного массива.
//Параметром подпрограммы является
//открытый массив целых чисел.
procedure outputArray(X:array of integer);
var i:byte;
begin
 //Элементы в открытом массиве
//пронумерованы от 0 до high(X).
 for i:=0 to high(X) do
 //Вывод сообщения:
//X[номер_элемента]=значение_элемента.
     writeln('X[',i,']=',X[i]);
end;
var
 A:array [1..10] of integer;
 C: array of integer;
 i:byte;
begin
 //Формирование одномерного массива A
//из 10 элементов.
 for i:=1 to 10 do A[i]:= 2*i+1;
```

```
 //Выделяется память для размещения
//3 целочисленных значений:
 SetLengTh(C,3);
 //Формирование одномерного массива C
//из 3 элементов.
 for i:=0 to 2 do C[i]:= 1-2*i;
 //Обращение к подпрограмме.
 outputArray(A);
 outputArray(C);
end.
```

Без использования открытых массивов процедуру `outputArray` пришлось бы записать так:

```
//Описание типа: массив целых чисел,
//пронумерованных от 0 до10.
type massiv=array [0..10] of integer;
//Процедура предназначена для вывода сообщений
//о значениях элементов одномерного массива.
procedure outputArray(X:massiv;nN,nK:byte);
//Параметры подпрограммы:
//1. Массив целых чисел X.
//2. Нижняя граница индекса nN.
//3. Верхняя граница индекса nK.
var i:byte;
begin
 //Элементы массива нумеруются от nN до nK.
 for i:=nN to nK do
     writeln('X[',i,']=',X[i]);
end;
```

5.11 Использование указателей для работы с динамическими массивами

Все объявленные в программе статические переменные, которые мы рассматривали до этого момента, размещаются в одной непрерывной области оперативной памяти, которая называется сегментом данных. Для работы с массивами большой размерности можно воспользоваться так называемой динамической памятью, которая выделяется программе после запуска программы на выполнение. Размер динамической памяти можно варьировать в широких пределах. По умолча-

нию этот размер определяется всей доступной памятью ПК.

Динамическое размещение данных осуществляется компилятором непосредственно в процессе выполнения программы. При динамическом размещении заранее неизвестно количество размещаемых данных. Кроме того, к ним нельзя обращаться по именам, как к статическим переменным.

Оперативная память ПК представляет собой совокупность элементарных ячеек для хранения информации – байтов, каждый из которых имеет собственный номер. Эти номера называются адресами, они позволяют обращаться к любому байту памяти.

5.11.1 Работа с динамическими переменными и указателями

Free Pascal имеет гибкое средство управления памятью – указатели.

Указатель – переменная, которая в качестве своего значения содержит адрес байта памяти. Указатель занимает 4 байта.

Как правило, указатель связывается с некоторым типом данных. В таком случае он называется типизированным. Для его объявления используется знак ^, который помещается перед соответствующим типом, например:

```
type massiv=array [1..2500] of real;
var a:^integer; b,c:^real; d:^massiv;
```

В языке Free Pascal можно объявлять указатель, не связывая его с конкретным типом данных. Для этого служит стандартный тип `pointer`, например:

```
var p,c,h: pointer;
```

Указатели такого рода будем называть *нетипизированными*. Поскольку нетипизированные указатели не связаны с конкретным типом, с их помощью удобно динамически размещать данные, структура и тип которых меняются в ходе работы программы.

Значениями указателей являются адреса переменных памяти, поэтому следовало ожидать, что значение одного из них можно передавать другому. На самом деле это не совсем так. Эта операция проводится только среди указателей, связанных с одними и теми же типами данных.

Например:

```
Var
p1, p2:^integer; p3:^real; pp:pointer;
```

В этом случае присваивание `p1:=p2;` допустимо, в то время как `p1:=p3;` запрещено, поскольку `p1` и `p3` указывают на разные типы данных. Это ограничение не распространяется на нетипизированные указатели, поэтому можно записать `pp:=p3; p1:=pp;` и достичь необходимого результата.

Вся динамическая память во Free Pascal представляет собой сплошной массив байтов, называемый кучей. Физически куча располагается за областью памяти, которую занимает тело программы.

Начало кучи хранится в стандартной переменной `heaporg`, конец – в переменной `heapend`. Текущая граница незанятой динамической памяти хранится в указателе `heapprt`.

Память под любую динамическую переменную выделяется процедурой `new`, параметром обращения к которой является типизированный указатель. В результате обращения последний принимает значение, соответствующее динамическому адресу, начиная с которого можно разместить данные, например:

```
var
i,j:^integer;
r:^real;
begin
new(i);
new(R);
new(j)
```

В результате выполнения первого оператора указатель `i` принимает значение, которое перед этим имел указатель кучи `heapprt`. Сам `heapprt` увеличивает свое значение на 4, так как длина внутреннего представления типа `integer`, связанного с указателем `i`, составляет 4 байта. Оператор `new(r)` вызывает еще одно смещение указателя `heapprt`, но уже на 8 байтов, потому что такова длина внутреннего представления типа `real`. Аналогичная процедура применяется и для переменной любого другого типа. После того как указатель стал определять конкретный физический байт памяти, по этому адресу можно разместить любое значение соответствующего типа, для чего сразу за указателем без каких-либо пробелов ставится значок `^`, например:

```
i^:=4+3;
j^:=17;
r^:=2 * pi;
```

Таким образом, значение, на которое указывает указатель, то есть собственно данные, размещенные в куче, обозначаются значком `^`. Значок `^` ставится сразу за указателем. Если после указателя значок `^` отсутствует, то имеется в виду адрес, по которому размещаются данные. Динамически размещенные данные (но не их адрес!) можно использовать для констант и переменных соответствующего типа в любом месте, где это допустимо, например:

```
r^:=sqr(r^)+sin(r^+i^)-2.3
```

Невозможен оператор

```
r:=sqr(r^)+i^;
```

так как указателю r нельзя присвоить значение вещественного типа.

Точно так же недопустим оператор

```
r^:=sqr(r);
```

поскольку значением указателя `r` является адрес, и его (в отличие от того значения, которое размещено по данному адресу) нельзя возводить в квадрат. Ошибочным будет и присваивание `r^:=i`, так как вещественным данным, на которые указывает `r^`, нельзя давать значение указателя (адрес). Динамическую память можно не только забирать из кучи, но и возвращать обратно. Для этого используется процедура `dispose(p)`, где `p` – указатель, который не изменяет значение указателя, а лишь возвращает в кучу память, ранее связанную с указателем.

При работе с указателями и динамической памятью необходимо самостоятельно следить за правильностью использования процедур `new`, `dispose` и работы с адресами и динамическими переменными, так как транслятор эти ошибки не контролирует. Ошибки этого класса могут привести к зависанию компьютера, а то и к более серьезным последствиям!

Другая возможность состоит в освобождении целого фрагмента кучи. С этой целью перед началом выделения динамической памяти текущее значение указателя `heapptr` запоминается в переменной-указателе с помощью процедуры `mark`. Теперь можно в любой момент освободить фрагмент кучи, начиная с того адреса, который запомнила процедура `mark`, и до конца динамической памяти. Для этого используется процедура `release`.

Процедура `mark` запоминает текущее указание кучи `heapptr`

(обращение `mark(ptr)`, где `ptr` – указатель любого типа, в котором будет возвращено текущее значение `heapptr`). Процедура `release(ptr)`, где `ptr` – указатель любого типа, освобождает участок кучи от адреса, хранящегося в указателе до конца кучи.

5.11.2 Работа с динамическими массивами с помощью процедур getmem и freemem

Для работы с указателями любого типа используются процедуры `getmem`, `freemem`. Процедура `getmem(p,size)`, где `p` – указатель, `size` – размер в байтах выделяемого фрагмента динамической памяти (`size` типа `word`), резервирует за указателем фрагмент динамической памяти требуемого размера.

Процедура `freemem(p,size)`, где `p` – указатель, `size` – размер в байтах освобождаемого фрагмента динамической памяти (`size` типа `word`), возвращает в кучу фрагмент динамической памяти, который был зарезервирован за указателем. При применении процедуры к уже освобожденному участку памяти возникает ошибка.

После рассмотрения основных принципов и процедур работы с указателями возникает вопрос: а зачем это нужно? В основном для того, чтобы работать с так называемыми динамическими массивами. Последние представляют собой массивы переменной длины, память под которые может выделяться (и изменяться) в процессе выполнения программы, как при каждом новом запуске программы, так и в разных её частях. Обращение к `i`-му элементу динамического массива `x` имеет вид `x^[i]`.

Рассмотрим процесс функционирования динамических массивов на примере решения следующей задачи.

ЗАДАЧА 5.2. Найти максимальный и минимальный элементы массива X(N).

Вспомним решение задачи традиционным способом.

```
Program din_mas1;
Var
 x:array [1..150] of real;
i,n:integer; max,min:real;
begin
 writeln('введите размер массива');
 readln (n);
for i:=1 to N do
```

```
begin
 write('x[',i,']='); readln(x[i]);
end;
max:=x[1]; min:=x[1];
for i:=2 to N do
begin
 if x[i] > max then max:=x[i];
 if x[i] < min  then  min:=x[i];
end;
 writeln('максимум=',max:1:4);
 writeln('минимум=',min:1:4);
end.
```

Теперь рассмотрим процесс решения задачи с использованием указателей. Распределение памяти проводим с помощью процедур `new-dispose` (программа din_mas2) или `getmem-freemem` (программа din_mas2).

```
 type massiw= array[1..150]of real;
 var
 x:^massiw;
 i,n:integer;
 max,min:real;
begin
{Выделяем память под динамический
массив из 150 вещественных чисел.}
 new(x);
 writeln('Введите размер массива');
 readln(n);
for i:=1 to N do
begin
 write('x(',i,')=');
 readln(x^[i]);
end;
max:=x^[1];min:=x^[1];
for i:=2 to N do
begin
 if x^[i] > max then max:=x^[i];
 if x^[i] < min then min:=x^[i];
end;
```

```
writeln('максимум=',max:1:4,
                    ' минимум=',min:1:4);
{ Освобождаем память. }
dispose(x);
end.
type
 massiw=array[1..150]of real;
var
 x:^massiw;
 i,n:integer;max,min:real;
begin
writeln('Введите размер массива');
readln(n);
{ Выделяем память под n элементов массива. }
getmem(x,n*sizeof(real));
for i:=1 to N do
begin
 write('x(',i,')=');
 readln(x^[i]);
end;
max:=x^[1];min:=x^[1];
for i:=2 to N do
begin
 if x^[i] > max then max:=x^[i];
 if x^[i] < min then min:=x^[i];
end;
writeln('максимум=',max:1:4,
                         ' минимум=',min:1:4);
{ Освобождаем память. }
freemem(x,n*sizeof(real));
end.
```

При работе с динамическими переменными необходимо соблюдать следующий порядок работы:

1. Описать указатели.
2. Выделить память под массив (функции `new` или `getmem`).
3. Обработать динамический массив.
4. Освободить память (функции `dispose` или `freemem`).

5.12 Примеры программ

ЗАДАЧА 5.3. Дан массив A, состоящий из k целых чисел. Записать все отрицательные элементы массива A в массив B.

Решение задачи заключается в следующем. Последовательно перебираются элементы массива A. Если среди них находятся отрицательные, то они записываются в массив B. На рисунке 5.35 видно, что

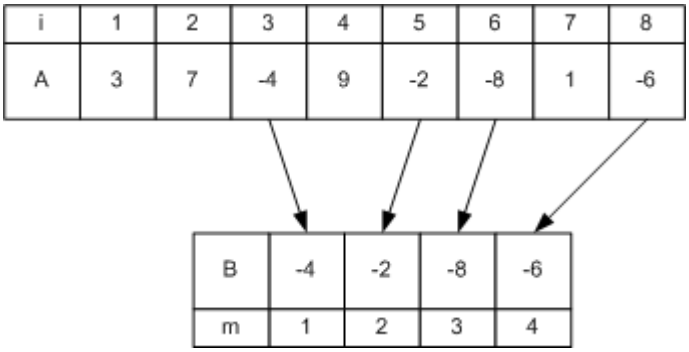

Рисунок 5.35: Процесс формирования массива B из отрицательных элементов массива A

первый отрицательный элемент хранится в массиве A под номером три, второй и третий — под номерами пять и шесть соответственно, а четвертый — под номером восемь. В массиве B этим элементам присваиваются номера *один*, *два*, *три* и *четыре*.

Поэтому для их формирования необходимо определить дополнительную переменную. В блок–схеме, приведенной на рисунке 5.36, роль такой переменной выполняет переменная m. В процессе формирования массива B в переменной m хранится номер сформированного элемента. Вначале в массиве B нет ни одного элемента, и поэтому m=0 (блок 2). В цикле (блок 3) последовательно перебираем все элементы A, если очередной элемент массива A отрицателен (блок 5), то переменная m увеличивается на единицу, а значение элемента массива A записывается в массив B под номером m (блок 6). В блоке 7 проверяем, были ли в массиве A отрицательные элементы и сформирован ли массив B. В результате этого алгоритма будет сформирован массив B отрицательных чисел, состоящий из k чисел.

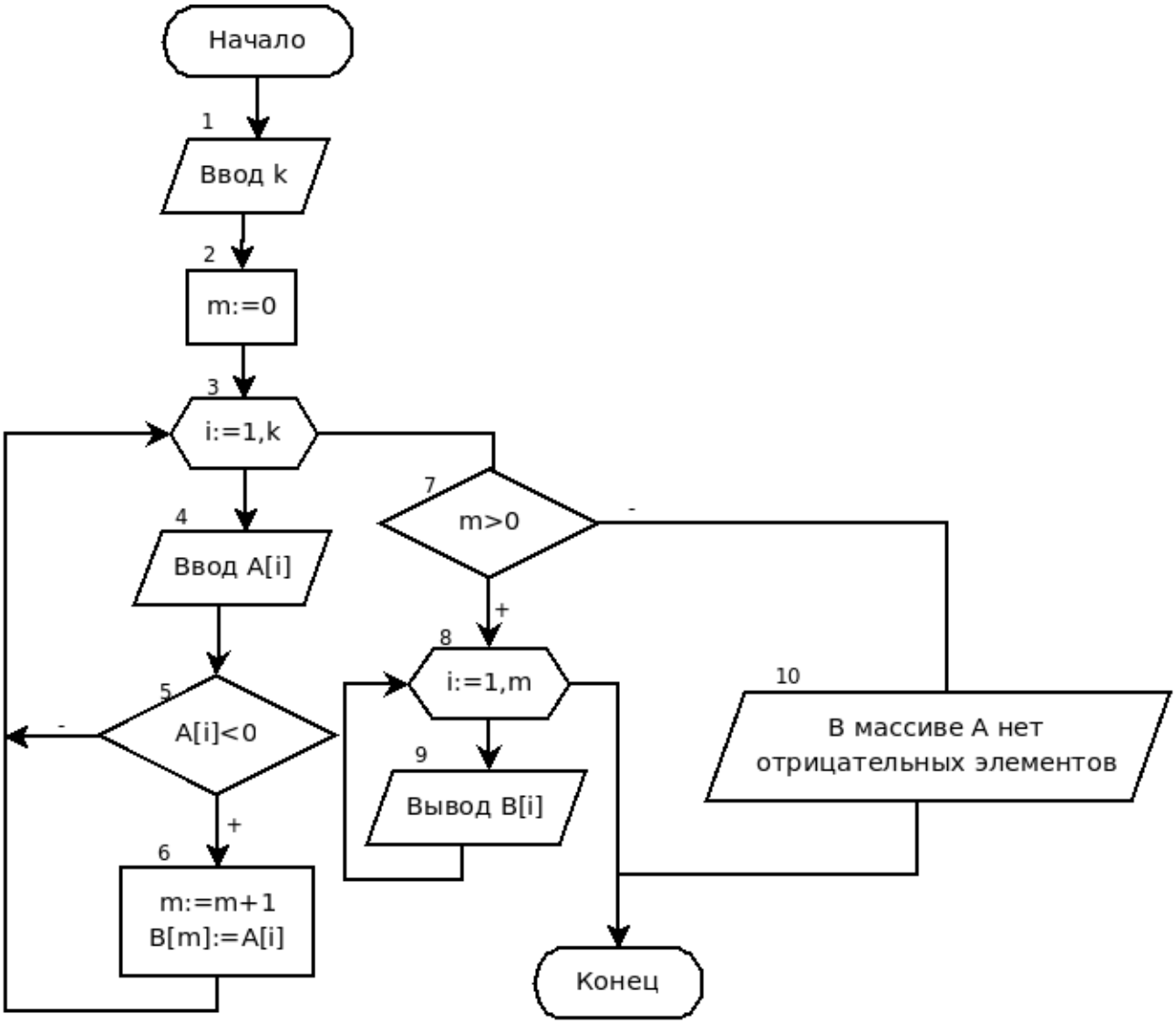

Рисунок 5.36: Блок-схема формирования массива B из отрицательных элементов массива A

Приведенная ниже программа реализует описанный алгоритм.

```
var a,b:array [1..200] of word; k,m,i:byte;
begin
 write('введите размерность массива к=');
 readln(k);
 m:=0;
 for i:=1 to k do
 begin
     write('A[',i,']=');readln(A[i]);
 if A[i]<0 then
 begin
     m:=m+1;
     B[m]:=A[i];
```

```
 end;
 end;
 if m>0 then
     for i:=1 to m do   write(B[i],' ')
 else
write('В массиве нет отрицательных элементов');
end.
```

ЗАДАЧА 5.4. Задан массив y из n целых чисел. Сформировать массив z таким образом, чтобы в начале шли отрицательные элементы массива y, затем — положительные и, наконец, — нулевые.

Алгоритм решения этой задачи основывается на алгоритме перезаписи элементов, удовлетворяющих какому-либо условию из одного массива в другой, который был подробно рассмотрен в предыдущей задаче. Блок-схема решения задачи 5.4 представлена на рис. 5.37.

Текст программы с комментариями приведен ниже.

```
var i,k,n: integer;
 y,z:array [1..50] of integer;
begin
 writeln('Введите n<=50'); readln (n);
for i:=1 to n do //Ввод массива y.
 begin
     write('y[',i,']='); readln(y[i]);
 end;
 k:=0;
//Перезапись отрицательных чисел
//из массива y в массив z.
 for i:=1 to n do
     if y[i] < 0 then
     begin
         k:=k+1; z[k]:=y[i];
     end;
//Перезапись положительных чисел
//из массива y в массив z.
 for i:=1 to n do
     if y[i] >0 then
     begin k:=k+1;
             z[k]:=y[i]
     end;
```

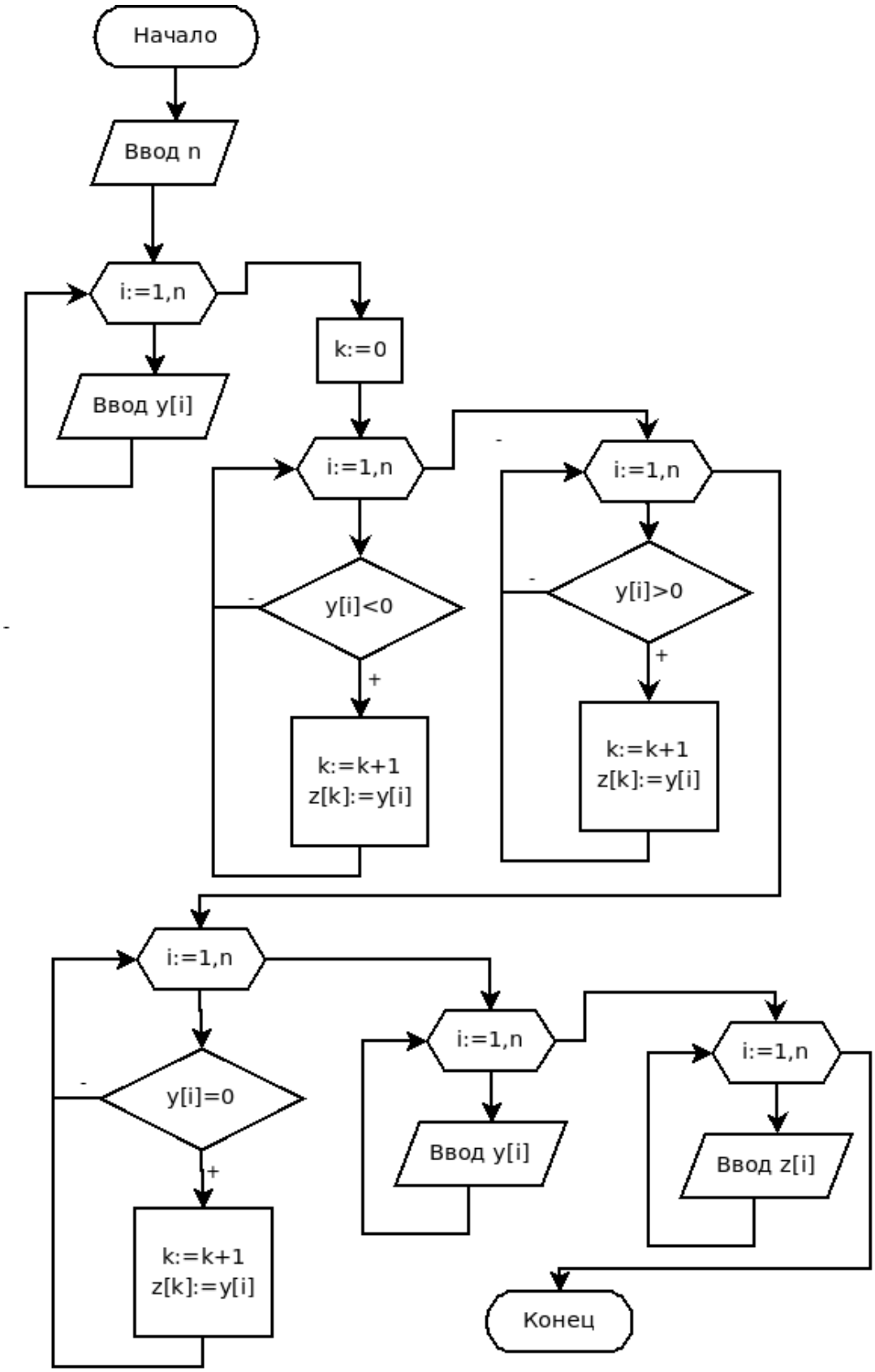

Рисунок 5.37: Блок схема решения задачи 5.4

```
//Перезапись нулевых чисел из массива
//y в массив z.
 for i:=1 to n do
     if y[i]=0 then
     begin
         k:=k+1;
         z[k]:= y[i];
     end;
// Вывод массива Y.
 writeln ('Массив Y:');
 for i:=1 to n do
         write(y[i],' ');
 writeln;
// Вывод массива Z.
 writeln ('Массив Z:');
 for i:=1 to n do
         write (z[i],' ');
 writeln;
end.
```

ЗАДАЧА 5.5. Переписать элементы в массиве X в обратном порядке.

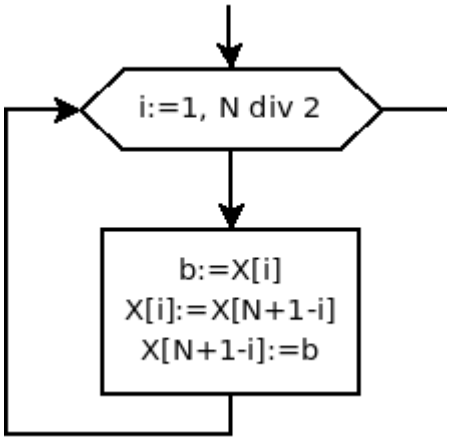

Рисунок 5.38: Фрагмент блок–схемы к задаче 5.5

Алгоритм решения задачи состоит в следующем: меняем местами 1-й и n-й элементы, затем 2-й и n−1-й элементы, и так до середины массива, элемент с номером i следует обменять с элементом n+1−i. Блок-схема обмена элементов в массиве представлена на рис. 5.38.

Ниже приведен текст консольного приложения задачи 5.5.

```
type
 massiv=array [1..100] of real;
var
 x:massiv;
 i,n:integer;
```

```
 b:real;
begin
//Ввод размера массива.
writeln ('Введите размер массива'); readln(n);
//Ввод массива.
for i:=1 to n do
begin
     write ('x[',i,']=');
     readln(x[i]);
end;
//Перебирается первая половина массива,
//и меняются элементы местами 1-й с n-м,
//2-й с (n-1), … i-й с (n+1-i)-м элементом.
for i:=1 to n div 2  do
begin
     b:=x[n+1-i];
     x[n+1-i]:=x[i];
     x[i]:=b;
end;
//Ввод преобразованного массива.
writeln('Преобразованный массив');
for i:=1 to n do
     write(x[i]:1:2,' ');
end.
```

ЗАДАЧА 5.6. Удалить из массива X, состоящего из n элементов, первые четыре нулевых элемента.

Вначале количество нулевых элементов равно нулю (`k=0`). Последовательно перебираем все элементы массива. Если встречается нулевой элемент, то количество нулевых элементов увеличиваем на `1` (`k:=k+1`). Если количество нулевых элементов меньше или равно `4`, то удаляем очередной нулевой элемент. Если встречаем пятый нулевой элемент (`k>4`), то аварийно выходим из цикла (дальнейшая обработка массива бесполезна).

Блок-схема представлена на рис. 5.39.

Текст программы с комментариями приведен ниже.

```
const n=20;
var
```

```
    X:array [1..n] of byte;
    k,i,j:integer;
begin
 for i:=1 to n do
    readln(X[i]);
k:=0;   {Количество нулевых элементов.}
j:=1;   {Номер элемента в массиве X.}
while j<=n do  {Пока не конец массива.}
begin
{Если нашли нулевой элемент, то}
    if x[j]=0 then
    begin
        k:=k+1; {посчитать его номер}
        if k>4 then
                break
    {Если k превышает 4, то выйти из цикла.}
    {Иначе удаляем j-й элемент из массива.}
        else
            for i:=j to n-k do
                X[i]:=X[i+1];
    end
{ Если встретился ненулевой элемент,
то просто переходим к следующему.}
else
    j:=j+1; {Если элемент ненулевой }
end;
{Вывод на печать измененного массива.}
{Количество элементов в массиве
 уменьшилось на k.}
for i:=1 to n-k do
        write(X[i],' ');
end.
```

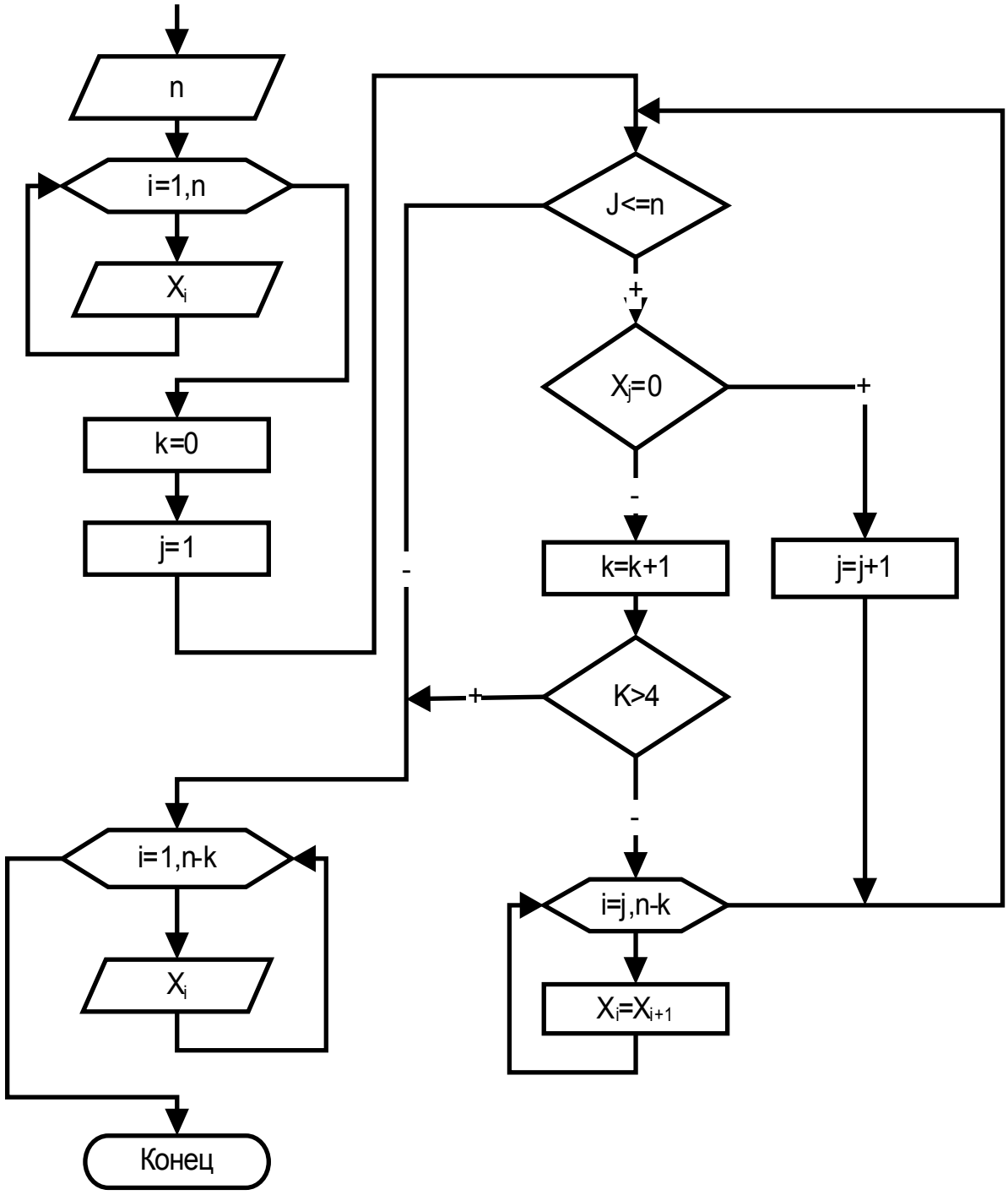

Рисунок 5.39: Алгоритм решения задачи 5.6

ЗАДАЧА 5.7. Найти сумму простых чисел целочисленного массива C(N).

Идея алгоритма состоит в следующем. Сначала сумма равна 0. Последовательно перебираем все элементы, если очередной элемент простой, то добавляем его к сумме. Для проверки, является ли число простым, напишем функцию `prostoe`, которая проверяет, является ли число простым.

Блок-схема этой функции представлена на рис.5.40.

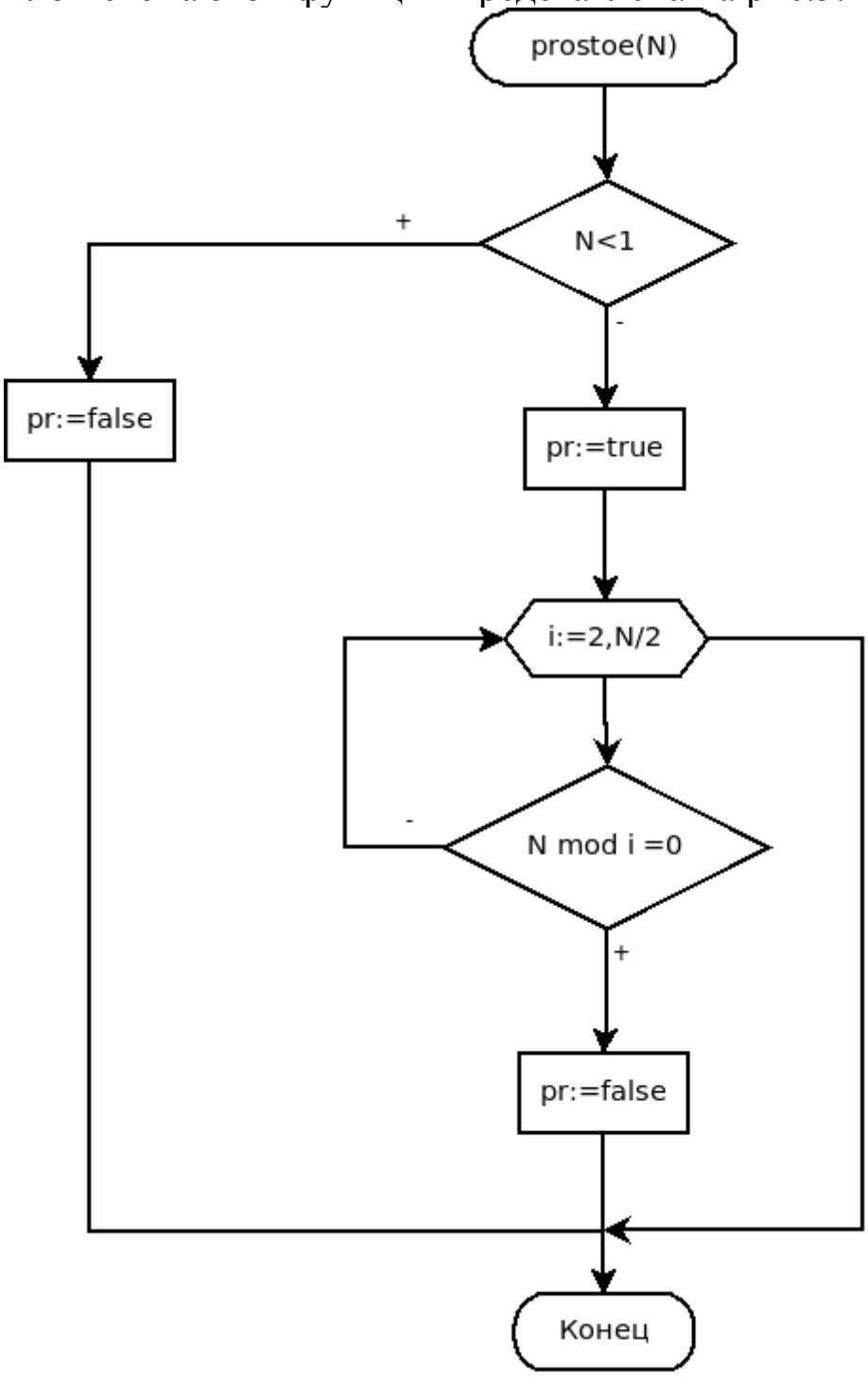

Рисунок 5.40: Блок-схема функции prostoe

Заголовок функции `Prostoe` имеет вид:

```
function Prostoe(N:integer):boolean;
```

Функция возвращает значение `true`, если число `N` является простым. В противном случае результатом функции является значение `false`. Блок–схема решения задачи 5.7 изображена на рис. 5.41.

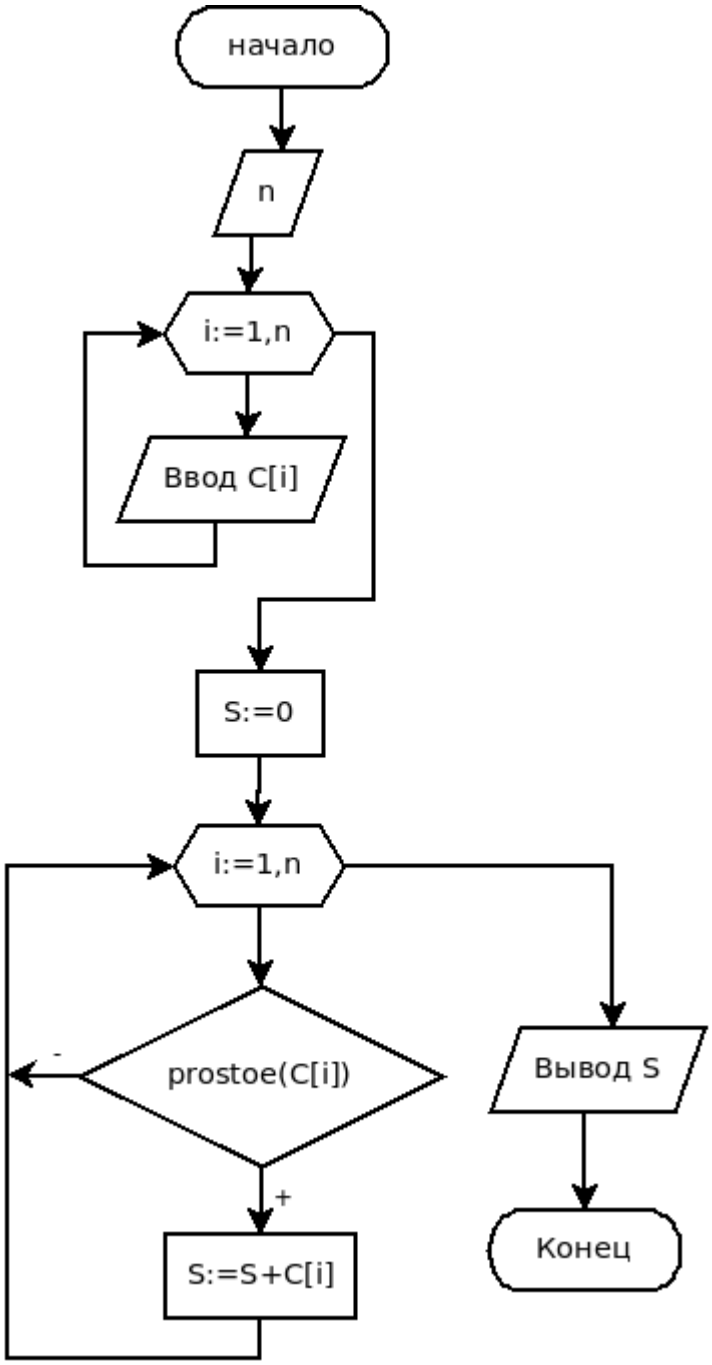

Рисунок 5.41: Блок-схема решения задачи 5.7

Далее приведен текст программы, реализующей этот алгоритм, с комментариями.

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

```
begin
{ Предположим, что текущее число простое. }
pr:=true;
for i:=2 to N div 2 do
 if (N mod i = 0) then
{ Если найдется хотя бы один делитель,
кроме единицы и самого числа, то }
 begin
{ изменить значение логической переменной }
    pr:=false;
{ и досрочно выйти из цикла. }
    break;
 end; end;
prostoe:=pr;
end;
{Основная программа}
var
 c:array [1..50] of word;
 i,n:byte; S:word;
begin
 write('Введите размерность массива n=');
 readln(n);
 for i:=1 to n do
 begin
    write('Введите ',i,'-й элемент массива ');
    readln(C[i]);
 end;
 S:=0;
 for i:=1 to n do
 { Если число простое, то накапливать сумму. }
 if prostoe(C[i]) then
    S:=S+C[i];
 {Вывод найденной суммы.}
 writeln('Сумма простых чисел массива S=',S);
end.
```

ЗАДАЧА 5.8. Определить, есть ли в заданном массиве серии элементов, состоящих из знакочередующихся чисел (рис. 5.42). Если есть, то вывести на экран количество таких серий.

Рисунок 5.42: Массив с тремя сериями знакочередующихся элементов

Идея алгоритма состоит в следующем. Последовательно в цикле от `1` до `n-1` сравниваются соседние элементы (первый и второй, второй и третий, … , предпоследний и последний). Если соседние элементы имеют разные знаки (их произведение отрицательно), то количество элементов в серии (переменная `k`) увеличиваем на один[65]. Если произведение соседних элементов в серии не отрицательно, то возможны два варианта: либо сейчас оборвалась серия из знакочередующихся элементов, либо очередные два элемента одного знака. Выбор из этих двух вариантов можно осуществить, сравнив переменную `k` с единицей. Если `k>1`, сейчас оборвалась серия из знакочередующихся элементов, и количество таких серий (`kol`) надо увеличить на 1.

После выхода из цикла проверяем, не было ли в конце массива серии из знакочередующихся элементов. Если такая серия был (`k>1`), то количество серий (`kol`) опять увеличиваем на 1. В завершение выводим количество серий из знакочередующихся элементов — переменную `kol`. Блок-схема решения задачи 5.8 представлена на рис. 5.43. Ниже приведен текст консольного приложения на языке Free Pascal.

```
var
 x:array[1..50] of real;
 n,i,k,kol:integer;
begin
 write('n=');
 readln(n);
 for i:=1 to n do
     read(x[i]);
{Так как минимальная серия состоит
из двух элементов, }
{ k присвоим значение 1. }
     k:=1; { Длина серии. }
```

65 Количество элементов в серии изначально равно нулю, так как после встречи в массиве первой пары знакочередующихся элементов в серии станет сразу два элемента, а все последующие идущие подряд элементы разного знака в серии будут увеличивать длину серии на один.

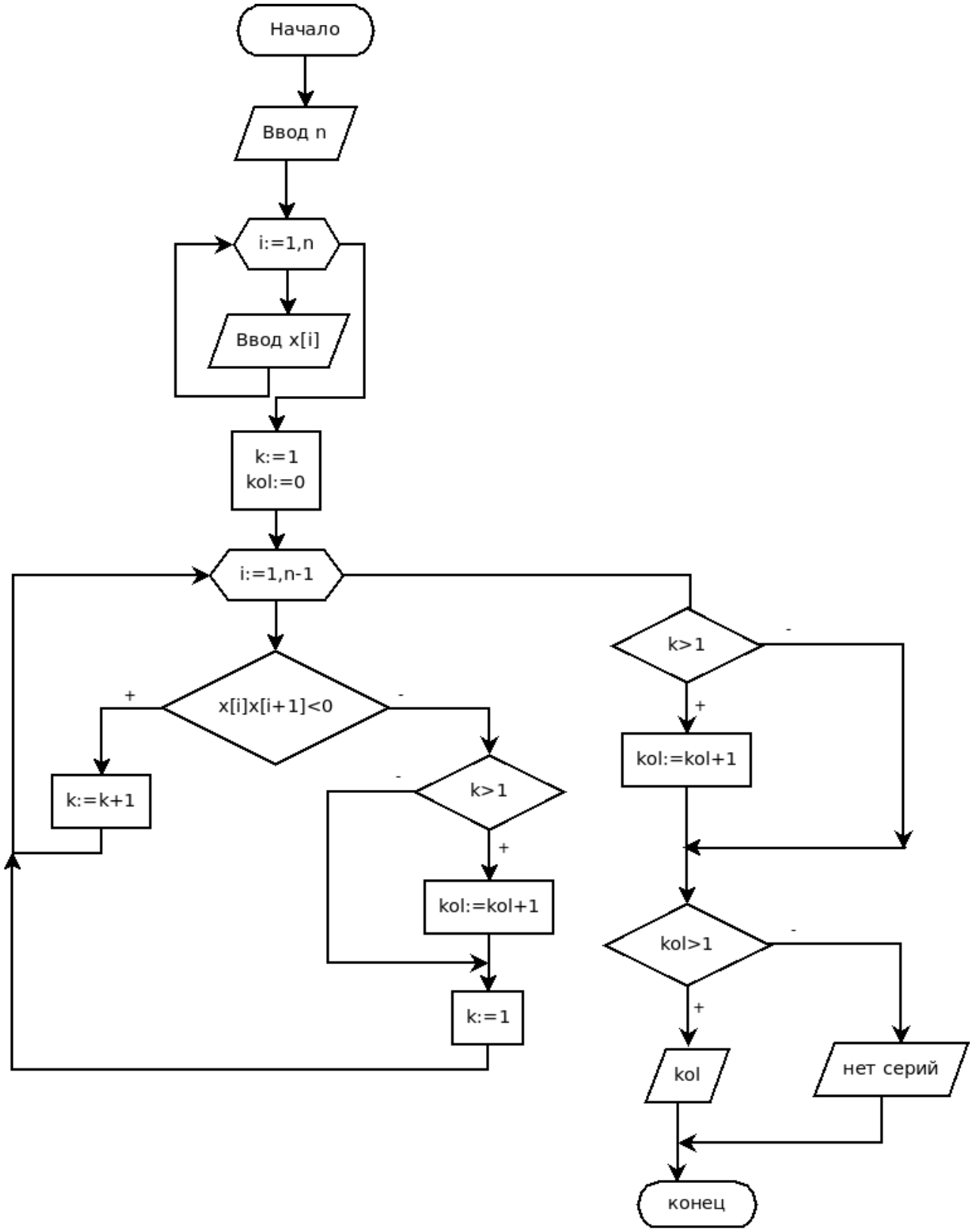

Рисунок 5.43: Блок–схема решения задачи 5.8

```
    kol:=0; { Количество серий в массиве. }
 for i:=1 to n-1 do
{ Если при умножении двух соседних элементов
```

```
 результат – отрицательное число,
то элементы имеют разный знак. }
     if x[i]*x[i+1]<0 then
     k:=k+1 { Показатель продолжения серии. }
     else
     begin
{ Если серия разорвалась, то увеличить счетчик
  количества серий. }
          if k>1 then
              kol:=kol+1;
{ Подготовить показатель продолжения серии }
{ к возможному появлению следующей серии. }
          k:=1;
     end;
{Проверка, не было ли серии в конце массива. }
 if k>1 then
{ Если да, увеличить счетчик еще на единицу. }
     kol:=kol+1;
     if kol>0 then
         write('Количество знакочередующихся
                                    серий=',kol)
     else
         write('Знакочередующихся серий нет')
end.
```

ЗАДАЧА 5.9. В заданном массиве найти самую длинную серию элементов, состоящую из единиц.

Для максимальной серии будем хранить ее длину (`max`), номер последнего элемента (`kon_max`).

Эта задача похожа на предыдущую, отличие заключается в том, что надо фиксировать не только тот факт, что серия кончилась, но и саму серию. Серия может характеризоваться двумя из трех параметров: первый элемент серии, последний элемент серии, длина серии. В связи с тем что мы фиксируем серию в момент ее окончания, в качестве параметров серии будем использовать последний элемент серии (`kon`) и ее длину (`k`).

Алгоритм решения этой задачи следующий. Вначале количество серий (`kol`) и ее длина (`k`) равны нулю. Перебираем последовательно

все элементы. Если текущий элемент равен 1, то количество элементов в серии[66] увеличиваем на 1. Если текущий элемент не равен 1, то возможны два варианта: либо сейчас оборвалась серия из единиц, либо встретился очереной не равный единице элемент. Выбор из этих двух вариантов можно осуществить, сравнив переменную k с единицей. Если k>1, сейчас оборвалась серия из единиц, и количество таких серий (kol) надо увеличить на 1, зафиксировать конец серии (kon:=i-1), длину серии (dlina:=k). После этого необходимо проверить, какая по счету серия. Если это первая серия (kol=1), то объявляем ее максимальной, в переменную max записываем длину текущей серии k, в переменную kon_max — kon (последний элемент текущей серии). Если это не первая серия (kol>1), то длину текущей серии (k) сравниваем с длиной серии максимальной длины (max). И если k>max, то текущую серию объявляем серией максимальной длины (max:=k; kon_max:=kon;). Если встретился не равный единице элемент, надо количество элементов в серии положить равным нулю (k:=0).

После выхода из цикла надо также проверить, не было ли в конце серии, состоящей из единиц. Если серия была в конце, то следует обработать ее так же как и серию, которая встретилась в цикле.

Блок-схема решения задачи приведена на рис. 5.44.

Ниже приведен текст консольного приложения решения задачи.

```
var x:array[1..50] of integer;
n,i,k,kol,kon,max,kon_max,dlina:integer;
begin
{Ввод размера массива.}
write('n=');
readln(n);
{Ввод массива}
writeln('Массив X');
for i:=1 to n do
read(x[i]);
{Начальное присваивание длины серии
и количества серий}
k:=0; { Длина серии. }
kol:=0; { Количество серий в массиве. }
```

66 Количество подряд идущих единиц.

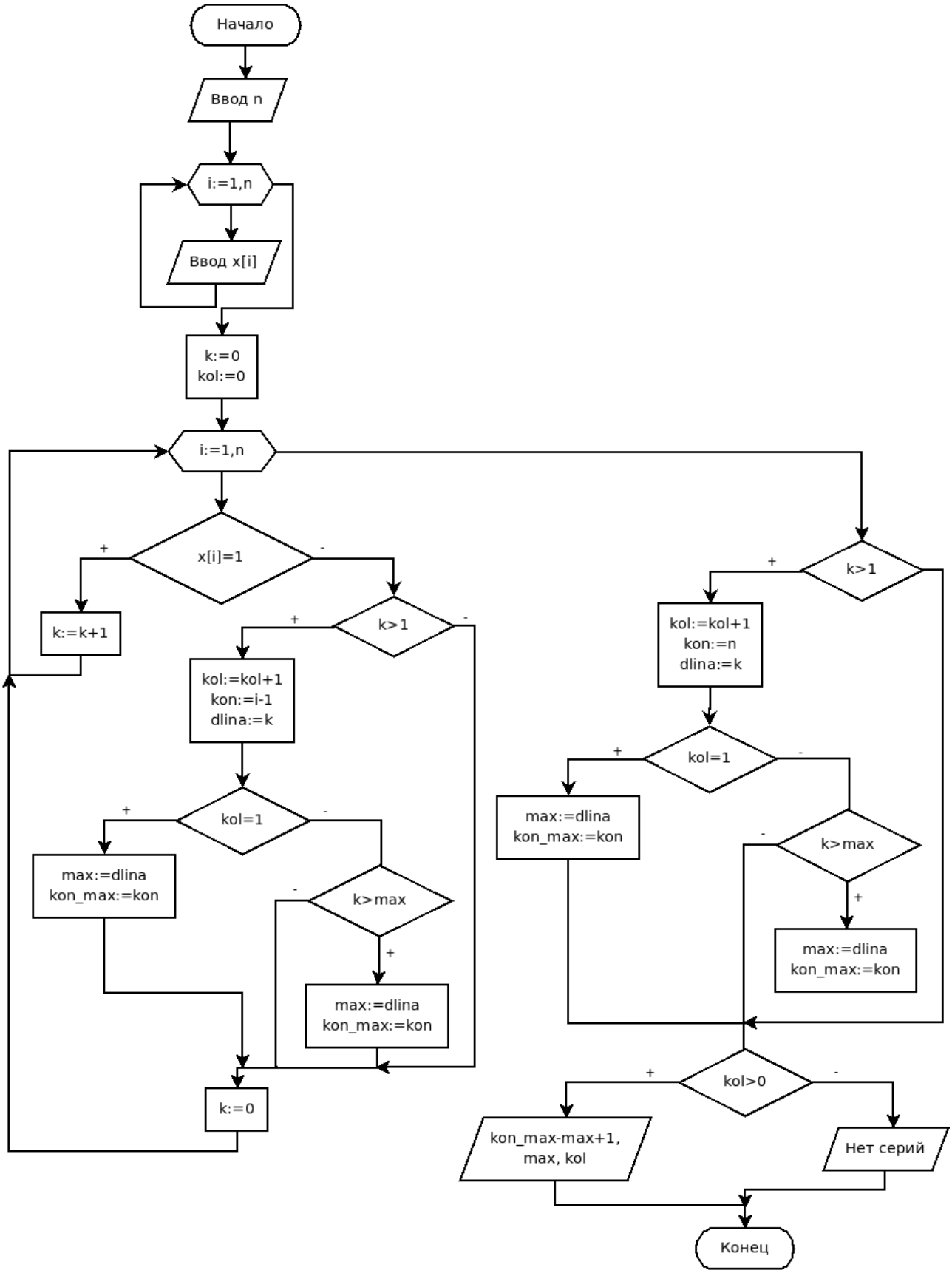

Рисунок 5.44: Блок-схема решения задачи 5.9

```
{Перебираем все элементы в массиве}
for i:=1 to n do
{Если текущий элемент равен 1, то }
if x[i]=1 then
{количество подряд идущих единиц
увеличить на 1.}
k:=k+1
else
{ Если текущий элемент не равен 1, то}
begin
{ проверяем, была ли серия до этого, k>1?}
 if k>1 then
{Если только что оборвалась серия, то}
 begin
{увеличиваем количество серий.}
    kol:=kol+1;
{Фиксируем тот факт, что на предыдущем
элементе серия закончилась,}
    kon:=i-1;
{Длина серии равна k.}
    dlina:=k;
{Если это первая серия,}
    if kol=1 then
{объявляем ее максимальной.}
    begin
{Длина максимальной серии единиц.}
        max:=dlina;
{Конец максимальной серии, состоящей из
 единиц, хранится в переменной kon_max.}
        kon_max:=kon;
    end
{Если это не первая серия,
состоящая из единиц,}
    else
{то ее длину сравниваем с длиной серии
с максимальным количеством единиц.}
        if k>max then
{Если длина текущей серии больше,}
```

```
        begin
{то объявляем ее максимальной.}
            max:=dlina; kon_max:=kon;
        end;
 end;
{Если текущий элемент массива не равен 1,
то количество подряд встречающихся единиц
начинаем считать сначала (k:=0).}
 k:=0;
end;
{Проверка, не было ли серии в конце массива. }
if k>1 then
{ Если да, увеличить счетчик еще на единицу. }
begin
    kol:=kol+1;
{Серия закончилась на последнем элементе}
    kon:=n; dlina:=k;
{Обработка последней серии так,
как это происходило в цикле.}
    if kol=1 then
    begin
        max:=dlina; kon_max:=kon;
    end
    else
        if k>max then
        begin
            max:=dlina; kon_max:=kon;
        end;
end;
{Если серии были, то}
 if kol>0 then
{вывод информации о серии с максимальным
количеством единиц.}
 begin
    writeln('Количество серий, состоящих из
                                единиц=',kol);
    writeln('Наибольшая серия начинается
    с номера ',kon_max-max+1,', заканчивается
```

```
    номером ', kon_max,', ее длина равна ', max)
    end {Вывод информации об отсутствии серий.}
  else writeln('Нет серий, состоящих из единиц')
  end.
```

ЗАДАЧА 5.10. Задан массив вещественных чисел. Перевести все элементы массива в *p*-ричную систему счисления.

Перед решением всей задачи давайте разберемся с алгоритмом перевода вещественного числа из десятеричной в другую систему счисления. Этот алгоритм можно разделить на следующие этапы:

1. Выделение целой и дробной частей числа.
2. Перевод целой части числа в другую систему счисления.
3. Перевод дробной части числа в другую систему счисления.
4. Объединение целой и дробной частей числа в новой системе счисления.

Алгоритм перевода целого числа в другую систему счисления.

Разделить нацело число на основание новой системы счисления. Получим остаток и частное. Остаток от деления будет младшим разрядом числа. Его необходимо будет умножить на 10 в нулевой степени. Если частное не равно нулю, то продолжим деление; новый остаток даст нам следующий разряд числа, который надо будет умножить на десять в первой степени, и т. д. Деление будем продолжать до тех пор, пока частное не станет равным 0. Особенностью алгоритма является то, что число формируется в обратном порядке от младшего разряда к старшему, что позволит в один проход собрать число в новой системе счисления.

Алгоритм перевода дробной части числа в новую систему счисления.

Умножить дробную часть числа на основание системы счисления. В полученном произведении выделить целую часть числа, это будет старший разряд числа, который необходимо умножить на 10^{-1}. Дробную часть опять умножить на основание системы счисления. В произведении целая часть будет очередным разрядом (его умножать надо будет на 10^{-2}), а дробную часть необходимо опять умножать на основание системы счисления до получения необходимого количества разрядов исходного числа.

Блок-схема функции перевода вещественного число N из 10-й системы счисления в другую систему представлена на рис. 5.45.

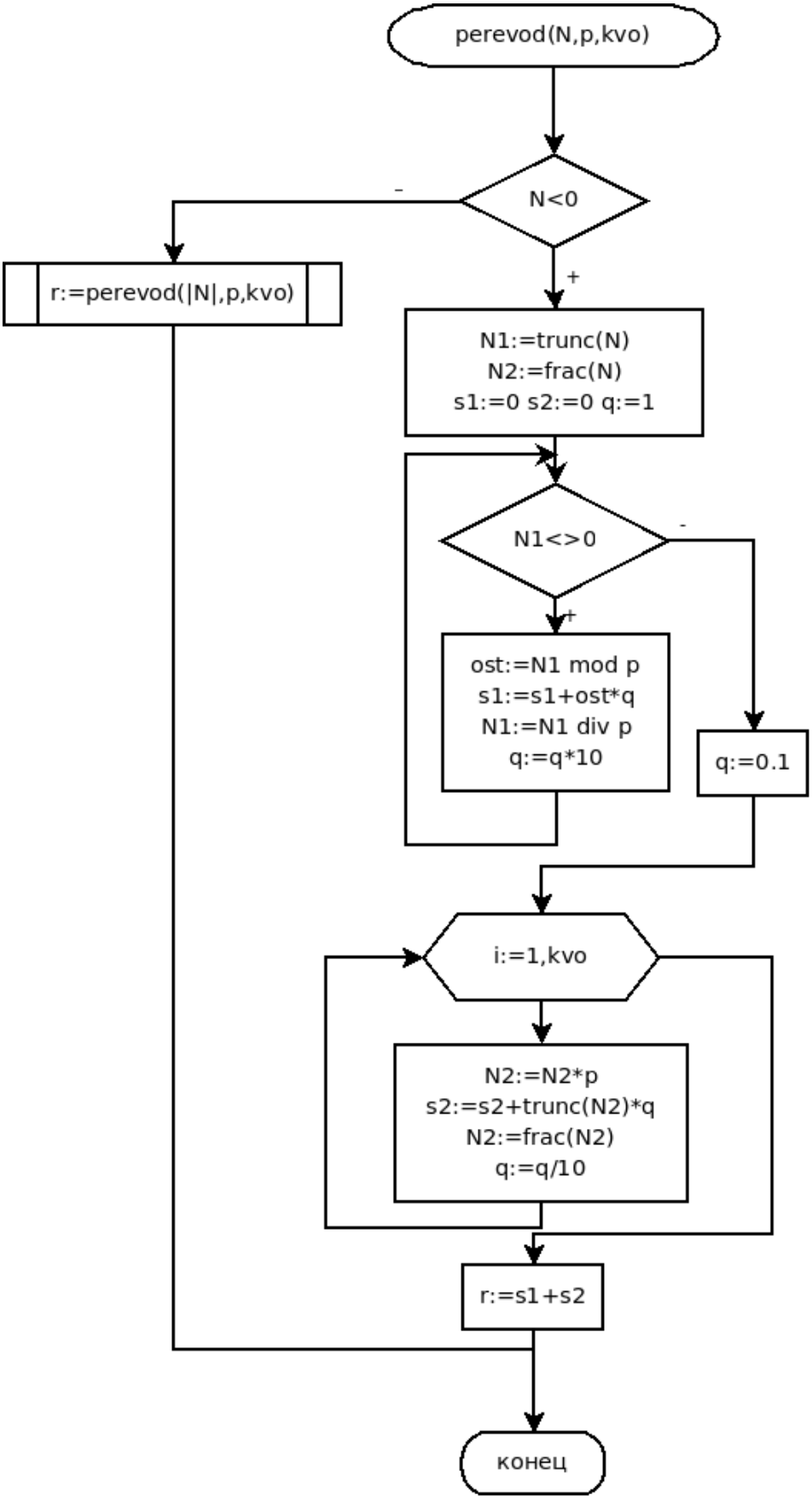

Рисунок 5.45: Блок-схема функции перевода вещественного числа в p-ричную систему счисления

Обратите внимание, как в блок-схеме и в функции реализовано возведение в степень. В связи с тем что при переводе целой части числа последовательно используются степени *10*, начиная *0*, для формирования степеней десяти, вводится переменная *q*, которая в начале равна *1*, а затем последовательно в цикле умножается на *10*. При переводе дробной части числа последовательно нужны отрицательные степени *10*: $10^{-1}, 10^{-2}, \ldots$. Поэтому при формировании дробной части числа переменная *q:=0.1*, которая последовательно в цикле делится на *10*.

Ниже приведен текст консольной программы решения задачи 5.10 с комментариями.

```
{ Функция перевода вещественного числа в p-рич-
ную систему счисления, входные параметры функции:
вещественное число N, основание системы счисления
— целое число p, kvo — количество разрядов в дроб-
ной части формируемого числа.}
   function perevod(N:real;P:word;kvo:word):real;
   var i ,N1, ost: word;
    s1, N2, r, s2:real;
    q:real;
   begin
   {Если исходное число отрицательно, то для его
перевода рекурсивно обращаемся к функции perevod,
передавая в качестве параметра модуль числа.}
   if N<0 then r:=-perevod(abs(N),P,kvo)
   else
   begin
   {Выделяем целую N1 и дробную N2 части веще-
ственного числа N.}
    N1:=trunc(N);N2:=frac(N);
    s1:=0;s2:=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 целой части числа. Пока число не ста-
нет равным 0.}
```

```
    while (N1<>0) do
    begin
   {Вычисляем ost — очередной разряд числа, как остаток от деления N1 на основание системы счисления.}
        ost:=N1 mod P;
   {Очередной разряд числа умножаем на 10 в степени i и добавляем к формируемому числу s1.}
        s1:=s1+ost*q;
{Уменьшаем число N1 в p раз путем целочисленного деления на p.}
        N1:=N1 div P;
   {Формируем следующую степень десятки.}
        q:=q*10;
    end;
   {В переменной q будем последовательно хранить отрицательные степени десяти, вначале туда записываем 0.1 — десять в минус первой степени, а затем последовательно в цикле будем делить q на 10.}
    q:=0.1;
    for i:=1 to kvo do
    begin
   {Умножаем дробную часть на 10.}
        N2:=N2*p;
   {Вычисляем очередной разряд числа как целую часть от умножения N2 на основание системы счисления. Очередной разряд числа умножаем на 10 в степени i и добавляем к формируемому числу s2.}
        s2:=s2+trunc(N2)*q;
   {Выделяем дробную часть от сформированного числа}
        N2:=frac(N2);
   {Формируем очередную отрицательную степень 10.}
        q:=q/10;
    end;
   {Суммируем целую и дробную части числа в p-ричной системе счисления.}
    r:=s1+s2;
```

```
    end;
     perevod:=r;
    end;
    var C:array[1..100] of real;
    p,i,n:word;
    begin
    {Ввод размера массива.}
     Write('n=');readln(n);
    {Ввод массива.}
     writeln('Массив C');
     for i:=1 to n do  read(C[i]);
    {Ввод системы счисления.}
     writeln('Введите основание
                        системы счисления');
    readln(p);
    {Перевод всех элементов массива в другую систе-
му счисления.}
     for i:=1 to n do
     c[i]:=perevod(C[i],p,5);
    {Вывод преобразованного массива.}
     writeln('Преобразованный массив C');
     for i:=1 to n do
     write(C[i]:1:5,' ');
    end.
```

Результаты работы программы представлены на рис. 5.46.

```
Файл  Правка  Вид  Терминал  Вкладки  Справка
n=10
Массив C
5.7 9.905 -78.946 76.894 -11.1111 11.11 0.89654 -0.1324 132.13 90.87
Введите основание системы счисления
5
Преобразованный массив C
10.32222 14.42303 -303.43311 301.42133 -21.02342 21.02333 0.42201 -0.03123 1012.03111 330.41333

------------------
(program exited with code: 0)
Press return to continue
```

Рисунок 5.46: Результаты работы консольной программы решения задачи 5.10

ЗАДАЧА 5.11. Из массива целых положительных чисел Y удалить 3 последних числа, цифры которых в восьмеричном представлении образуют убывающую последовательность.

Для решения этой задачи понадобится функция, которая будет проверять, образуют ли цифры числа в восьмеричном представлении убывающую последовательность цифр.

Заголовок этой функции будет иметь вид:

```
function vosem(N:word):boolean;
```

На вход функции `vosem` приходит целое десятичное число (формальный параметр `N`), функция возвращает `true`, если цифры числа в восьмеричном представлении образуют убывающую последовательность и `false` в противном случае.

При разработке алгоритма этой задачи следует помнить, что при переводе числа из десятеричной системы в восьмеричную разряды числа мы будем получать в обратном порядке. Значит, получаемые восьмеричные разряды наоборот должны формировать возрастающую последовательность цифр.

Текст функции с комментариями приведен ниже.

```
function vosem(N:word):boolean;
var pr:boolean;
tsifra,tsifra_st:word; i:integer;
begin
 i:=0;
{Предположим, что цифры числа N в восьмеричном представлении образуют убывающую последовательность.}
pr:=true;
{Пока число N не равно 0, }
 while N<>0 do
 begin
{Достаем очередной разряд числа в восьмеричной системе.}
     tsifra:=N mod 8;
{Уменьшаем число в 8 раз.}
     N:=N div 8; i:=i+1;
{Если разряд не первый}
     if i>1 then {и текущий разряд меньше или равен предыдущему, цифры в 8-м представлении числа N не образуют убывающую последовательность (pr:=false) и аварийно покидаем цикл.}
     if tsifra<=tsifra_st then
```

```
        begin
            pr:=false; break;
        end;
        tsifra_st:=tsifra;
    end;
    vosem:=pr;
end;
```

Алгоритм решения задачи будет следующий. Перебираем все числа в массиве в обратном порядке, проверяем, образуют ли цифры текущего элемента массива в восьмеричном представлении убывающую последовательность. Если образуют, то количество таких чисел (k) увеличиваем на 1. Если $k \le 3$, то удаляем текущий элемент массива.

Создадим визуальное приложение, предназначенное для решения задачи 5.11. Расположим на форме следующие компоненты: три кнопки, три метки, одно поле ввода и две таблицы строк. Расположим их примерно так, как показано на рис. 5.47.

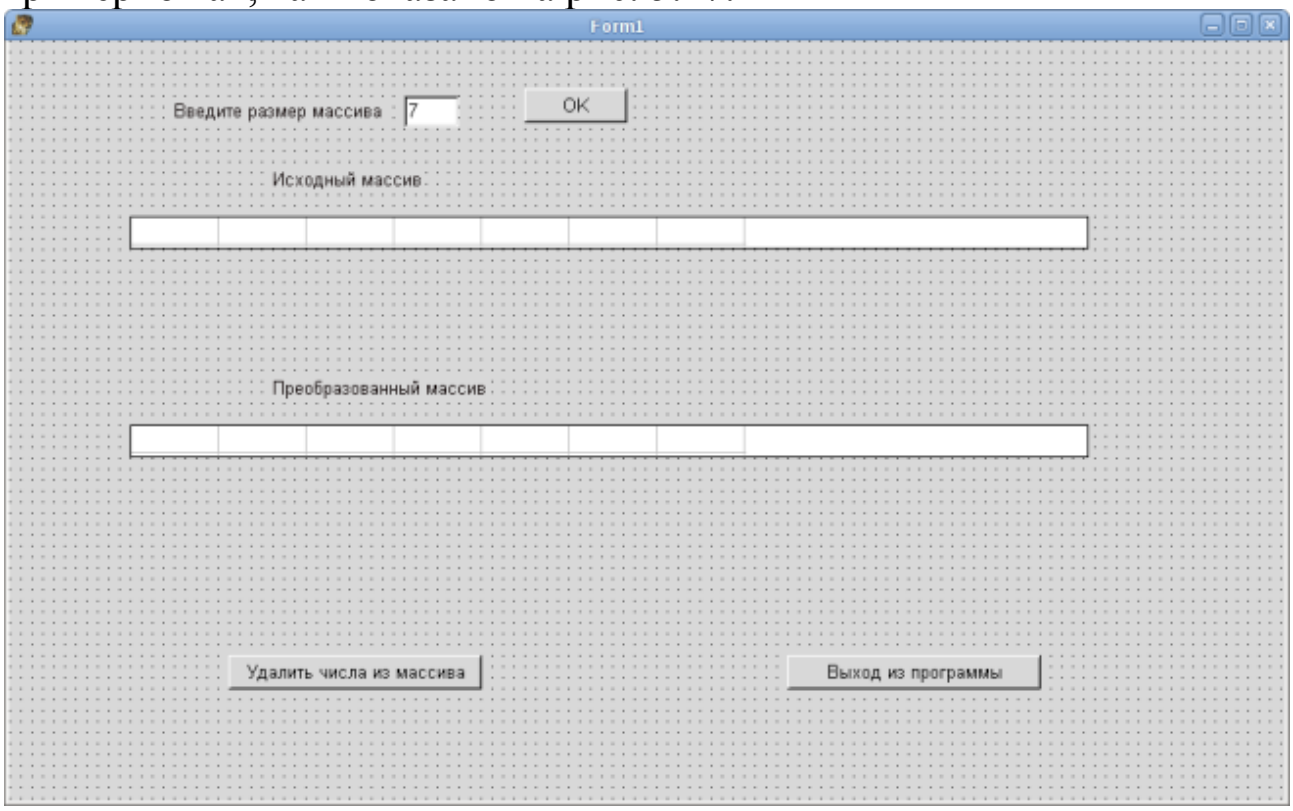

Рисунок 5.47: Форма для решения задачи 5.11

Свойства основных компонентов представлены в таблицах 5.6-5.7. При запуске приложения видимы метка Label1, поле для ввода размера массива Edit1 и кнопка Button1.

Таблица 5.6: Свойства меток, кнопок и текстового поля

Name	Caption (Text)	Width	Visible	Left	Top
label1	Введите размер массива	153	true	120	46
label2	Исходный массив	110	false	192	96
label3	Преобразованный массив	157	false	192	210
Edit1	7	40	true	288	40
Button1	OK	75	true	376	35
Button2	Удалить числа из файла	185	false	160	448
Button3	Выход из программы	185	false	568	448

Таблица 5.7: Свойства таблиц строк

Свойства	Таблица 1	Таблица 2
Name	StringGrid1	StringGrid2
ColCount	7	7
RowCount	1	1
Visible	false	false
FixedCols	0	0
FixedRows	0	0
Options.goEditing	true	true

При щелчке по кнопке OK (Button1) из поля ввода Edit1 считывается размер массива и становятся видимыми две другие кнопки, метка Label2, таблица строк StringGrid1 для ввода элементов массива. Метка Label1, поле для ввода размера массива Edit1 и кнопка Button1 становятся невидимыми. Окно приложения после щелчка по кнопке OK станет подобным представленному на рис. 5.48.

При щелчке по кнопке **Удалить числа из массива** происходят следующие действия:

- считывание массива из StringGrid1;
- удаление из массива трех последних чисел, цифры которых в восьмеричном представлении образуют убывающую последовательность;
- становятся видимыми метка Label3, таблица строк StringGrid2 для вывода элементов преобразованного массива.

Ниже приведен текст модуля с необходимыми комментариями. В результате работы программы решения задачи 5.11 окно приложения примет вид, представленный на рис. 5.49

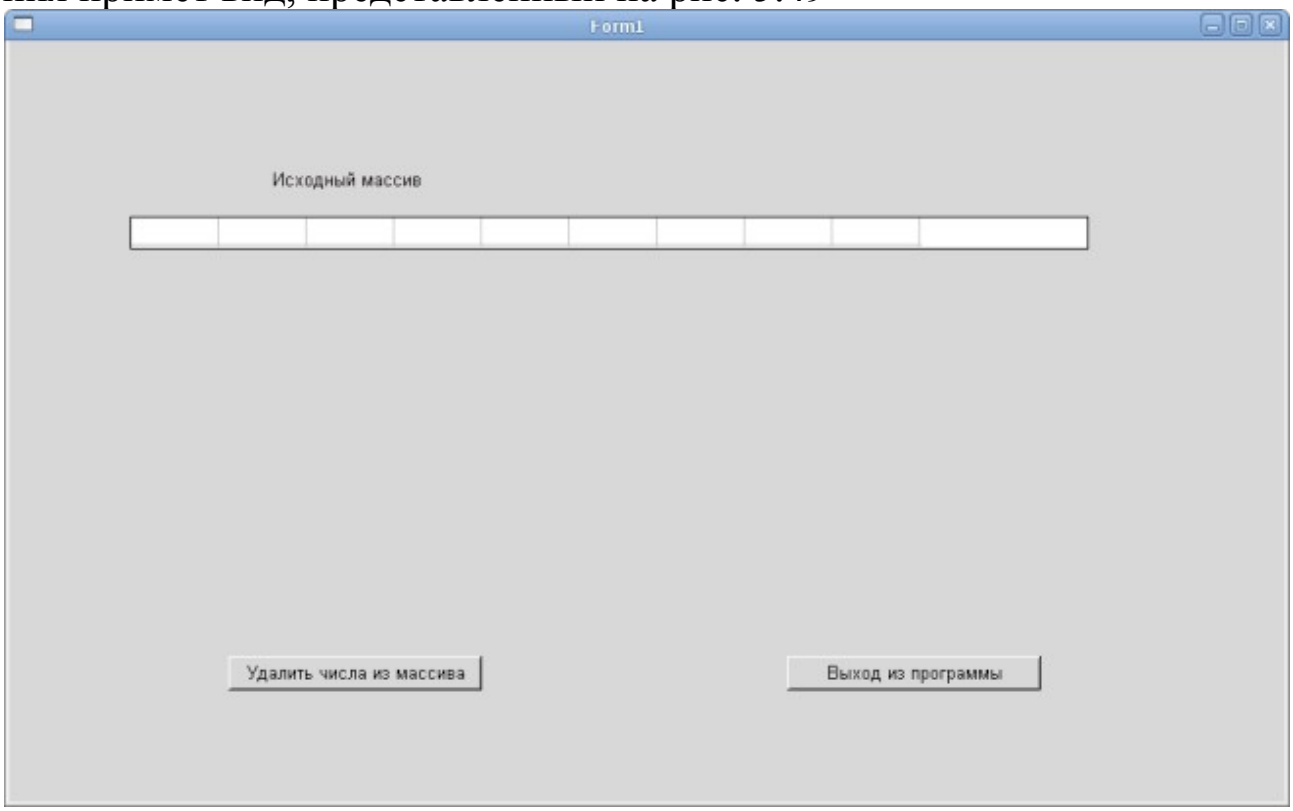

Рисунок 5.48: Окно приложения после щелчка по кнопке ОК

```
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;
    Edit1: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
```

```
        procedure Button1Click(Sender: TObject);
         procedure Button2Click(Sender: TObject);
        procedure Button3Click(Sender: TObject);
    private
        { private declarations }
    public
        { public declarations }
     end;
    type massiv=array[1..100] of word;
    var
     Form1: TForm1;
     N:word;
     X:massiv;
    implementation
    { TForm1 }
    {Функция vosem проверяет, образуют ли цифры
числа N в восьмеричном представлении убывающую по-
следовательность}
    function vosem(N:word):boolean;
    var pr:boolean;
     tsifra,tsifra_st:word;
     i:word;
    begin
     i:=0;
    {Предположим, что цифры числа N в восьмеричном
представлении образуют убывающую последователь-
ность.}
    pr:=true;
    {Пока число N не равно 0, }
     while N<>0 do
     begin
    {Достаем очередной разряд числа в восьмеричной
системе.}
        tsifra:=N mod 8;
    {Уменьшаем число в 8 раз.}
        N:=N div 8;
        i:=i+1;
    {Если разряд не первый}
```

```
        if i>1 then
   {И текущий разряд меньше или равен предыдущему,
цифры в восьмеричном представлении числа N не об-
разуют убывающую последовательность (pr:=false) и
аварийно покидаем цикл.}
        if tsifra<=tsifra_st then
        begin
            pr:=false;
            break;
        end;
        tsifra_st:=tsifra;
    end;
    vosem:=pr;
   end;
   {Функция удаления из массива X(N) элемента с
номером m.}
procedure udal(var X:Massiv; m:word;var N:word);
   var i:word;
   begin
    for i:=m to N-1 do
        x[i]:=x[i+1];
    N:=N-1;
   end;
   {Обработчик щелчка по кнопке OK.}
   procedure TForm1.Button1Click(Sender: TObject);
   begin
   {Считываем размер массива из поля ввода.}
    N:=StrToInt(Edit1.Text);
   {Делаем невидимыми первую метку, поле ввода и
кнопку OK.}
    Label1.Visible:=False;
    Edit1.Visible:=False;
    Button1.Visible:=False;
   {Делаем видимыми вторую метку, таблицу строк.}
    label2.Visible:=True;
    StringGrid1.Visible:=True;
   {Устанавливаем количество элементов
   в таблице строк.}
```

```
    StringGrid1.ColCount:=N;
   {Делаем видимыми вторую и третью кнопки.}
    Button2.Visible:=True;
    Button3.Visible:=True;
   end;
   {Обрабочик событий кнопки «Удалить числа из
массива».}
   procedure TForm1.Button2Click(Sender: TObject);
   var k,i:word;
   begin
   {Считываем массив из таблицы строк.}
    for i:=0 to N-1 do
    X[i+1]:=StrToInt(StringGrid1.Cells[i,0]);
   k:=0;
   {Перебираем все элементы массива в обратном по-
рядке.}
    for i:=N-1 downto 0 do
   {Если цифры очередного элемента массива в вось-
меричном представлении образуют убывающую последо-
вательность}
        if vosem(x[i]) then   begin
   {Увеличиваем счетчик таких чисел на 1.}
          k:=k+1;
   {Если это первое, второе или третье число,
удовлетворяющее условию, то удаляем его из масси-
ва.}
          if k<=3 then
              udal(x,i,N);   end;
   {Делаем видимыми третью кнопку и вторую таблицу
строк.}
    label3.Visible:=True;
    StringGrid2.Visible:=True;
    StringGrid2.ColCount:=N;
   {Вывод преобразованного массива.}
    for i:=0 to N-1 do
       StringGrid2.Cells[i,0]:=IntToStr(X[i+1]);
   end;
   {Обработчик кнопки закрытия окна.}
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin
 Close;
end;
initialization
 {$I unit1.lrs}
end.
```

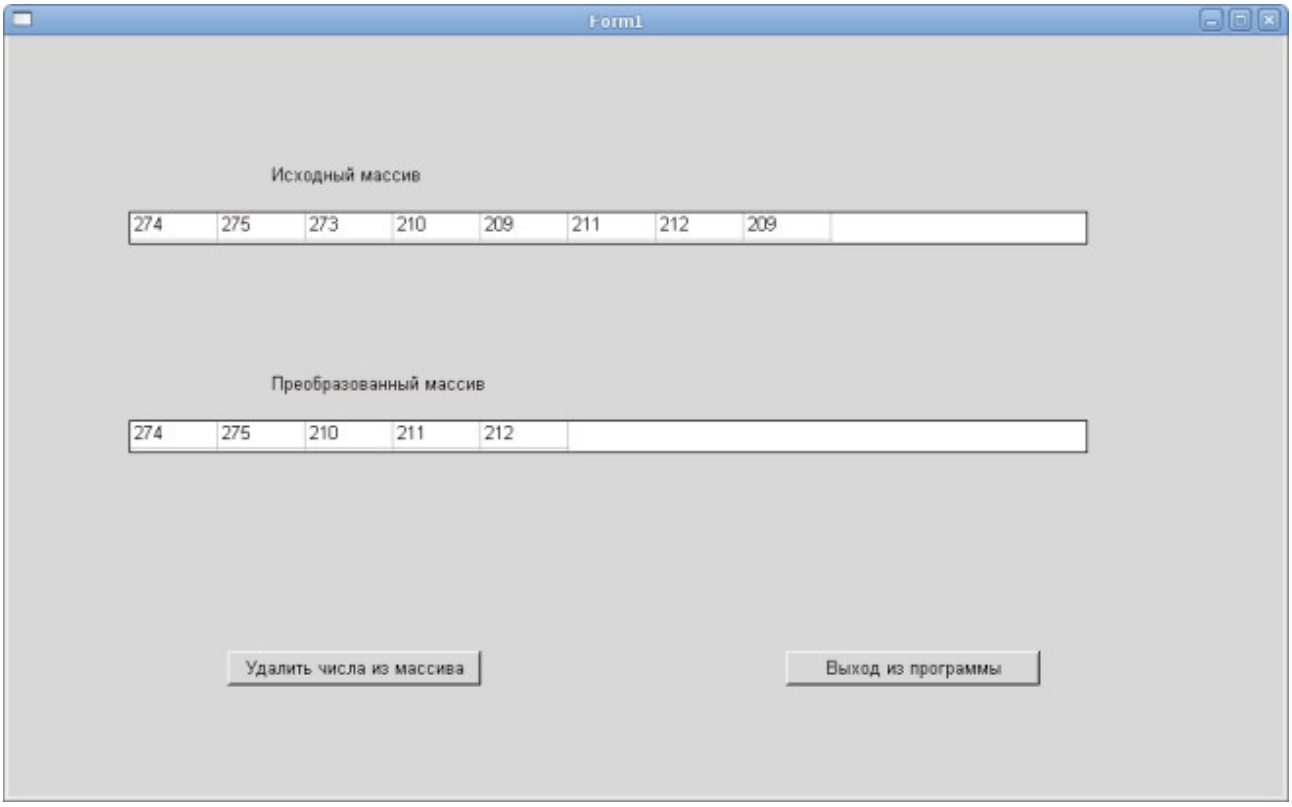

Рисунок 5.49: Окно с результатами решения задачи 5.11

5.13 Задачи для самостоятельного решения

1. Записать положительные элементы массива X подряд в массив Y. Вычислить сумму элементов массива X и произведение элементов массива Y. Из массива Y удалить элементы, расположенные между максимальным и минимальным элементами.

2. Сформировать массив B, записав в него элементы массива A с нечетными индексами. Вычислить среднее арифметическое элементов массива B и удалить из него максимальный, минимальный и пятый элементы.

3. Дан массив целых чисел X. Переписать пять первых положительных элементов массива и последних два простых элемента в массив Y. Найти максимальный отрицательный элемент массива X.

4. Записать элементы массива X, удовлетворяющие условию $1 \le x_i \le 2$, подряд в массив Y. Поменять местами максимальный и минимальный элементы в массиве Y.

5. Переписать элементы массива целых чисел X в обратном порядке в массив Y. Вычислить количество четных, нечетных и нулевых элементов массива Y.

6. Определить максимальный и минимальный элементы среди положительных нечетных элементов целочисленного массива X. Удалить из массива все нулевые элементы.

7. Переписать элементы целочисленного массива $X=(x_1, x_2, \ldots, x_{12})$ в массив $Y = (y_1, y_2, \ldots, y_{12})$, сдвинув элементы массива X вправо на три позиции. При этом три элемента с конца массива X перемещаются в начало: $(y_1, y_2, \ldots, y_{12}) = (x_{10}, x_{11}, x_{12}, x_1, x_2, \ldots, x_9)$. Определить номера максимального простого и минимального положительного элементов в массивах X и Y.

8. Записать элементы массива $X = (x_1, x_2, \ldots, x_{15})$ в массив $Y = (y_1, y_2, \ldots, y_{15})$, сдвинув элементы массива X влево на четыре позиции. При этом четыре элемента, стоящие в начале массива X, перемещаются в конец: $(y_1, y_2, \ldots, y_{15}) = (x_5, x_6, \ldots, x_{15}, x_1, x_2, x_3, x_4)$. Поменять местами минимальный и максимальный элементы массива Y.

9. В массиве X определить количество элементов меньших среднего арифметического значения. Удалить из массива положительные элементы, расположенные между максимальным и минимальным.

10. Вычислить среднее арифметическое элементов массива X, расположенных между его минимальным и максимальным значениями. Если минимальный элемент размещается в массиве раньше максимального, то упорядочить массив на данном промежутке по возрастанию его элементов.

11. Определить, содержит ли заданный массив группы элементов, расположенные в порядке возрастания их значений. Если да, то определить количество таких групп.

12. В заданном массиве целых чисел найти самую маленькую серию подряд стоящих нечетных элементов.

13. Удалить из массива целых чисел все простые числа, расположенные до максимального значения.

14. Удалить из массива предпоследнюю группу элементов, представляющих собой знакочередующийся ряд.

15. Задан массив целых положительных чисел X. Определить количество совершенных чисел в массиве. Удалить из массива последних два отрицательных числа. Сформировать массив Y, куда записать номера элементов массива X, являющихся простыми числами.

16. Переписать положительные элементы массива целых чисел X в обратном порядке в массив Y. Вычислить процент четных, нечетных и нулевых элементов массива Y. Перевести элементы массива Y в двоичную систему счисления.

17. Определить максимальный и минимальный элементы среди положительных четных элементов целочисленного массива X. Удалить из массива X совершенные числа, расположенные после максимального значения.

18. Заданы массивы вещественных чисел X и Y. Сформировать массив Z, куда записать положительные элементы массивов Y и Z в семеричной системе счисления. Определить номера максимального и минимального элементов в массиве Z.

19. Записать четные положительные элементы целочисленных массивов X и Y в массив Z. Поменять местами минимальный и максимальный элементы массива Z. Вывести элементы массива Z в четверичной системе счисления.

20. Из целочисленного массива X удалить все числа, превышающие среднее арифметическое простых элементов массива.

21. В массивах вещественных чисел X и Y записаны координаты точек на плоскости. Найти две точки, расстояние между которыми наименьшее.

22. Определить, содержит ли заданный массив вещественных чисел группы элементов, расположенные в порядке убывания их значений. Если да, то определить группу наименьшей длины.

23. В заданном массиве целых чисел найти самую большую серию подряд стоящих четных элементов.

24. Удалить из массива целых чисел все элементы, которые в пятеричном представлении не содержат нулей.

25. Из массивов вещественных чисел A и B сформировать массив C, записав в него элементы массивов A и B, которые не содержат «семерок» в восьмеричном представлении.