

Лекция 6. ОБРАБОТКА МАТРИЦ В C++

В C++ определены и двумерные массивы (матрицы), например матрица A, состоящая из 4 строк и 5 столбцов..

$$A = \begin{pmatrix} 6 & -9 & 7 & 13 & 19 \\ 5 & 8 & 3 & 8 & 22 \\ 3 & 7 & 88 & 33 & 71 \\ 55 & 77 & 88 & 37 & 61 \end{pmatrix}$$

Двумерный массив (матрицу) можно объявить так:

тип имя_переменной [n] [m];

где **n** – количество строк в матрице(строки нумеруются от **0** до **n-1**),

m – количество столбцов (столбцы нумеруются от **0** до **m-1**).

Например,

```
int x[10][15];
```

Описана матрица x, состоящая из 10 строк и 15 столбцов (строки нумеруются от 0 до 9, столбцы от 0 до 14).

Для обращения к элементу матрицы необходимо указать ее имя, и в квадратных скобках номер строки, а затем в квадратных скобках – номер столбца. Например, `x[2][4]` – элемент матрицы x, находящийся в третьей строке и пятом столбце.

В C++ можно описать многомерные массивы, которые можно объявить с помощью оператора следующей структуры:

тип имя_переменной [n1][n2]...[nk];

6.1. Блок-схемы основных алгоритмов обработки матриц

Блок-схема ввода матрицы представлена на рис. 6.1

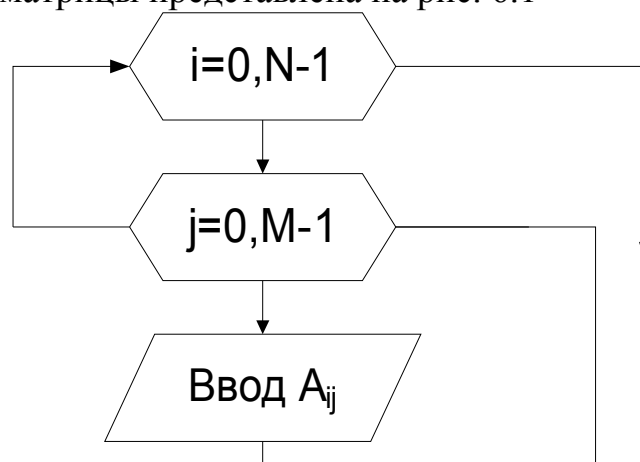


Рис 6.1. Ввод матрицы

С использованием функций `printf` и `scanf` ввод матрицы можно реализовать следующим образом.

```
printf("N="); scanf("%d", &N);  
printf("M="); scanf("%d", &M);  
printf("\n Vvedite A \n");  
for(i=0; i<N; i++)
```

```

for (j=0; j<M; j++)
{
    scanf ("%f", &b);
    a[i][j]=b;
}

```

С использованием cin и cout ввод матрицы можно реализовать следующим образом.

```

cout<<"N="; cin>>N;
cout<<"M="; cin>>M;
cout<<"\n Vvedite A \n";
for (i=0; i<N; i++)
    for (j=0; j<M; j++)
        cin>>a[i][j];

```

Блок-схема вывода матрицы представлена на рис. 6.2

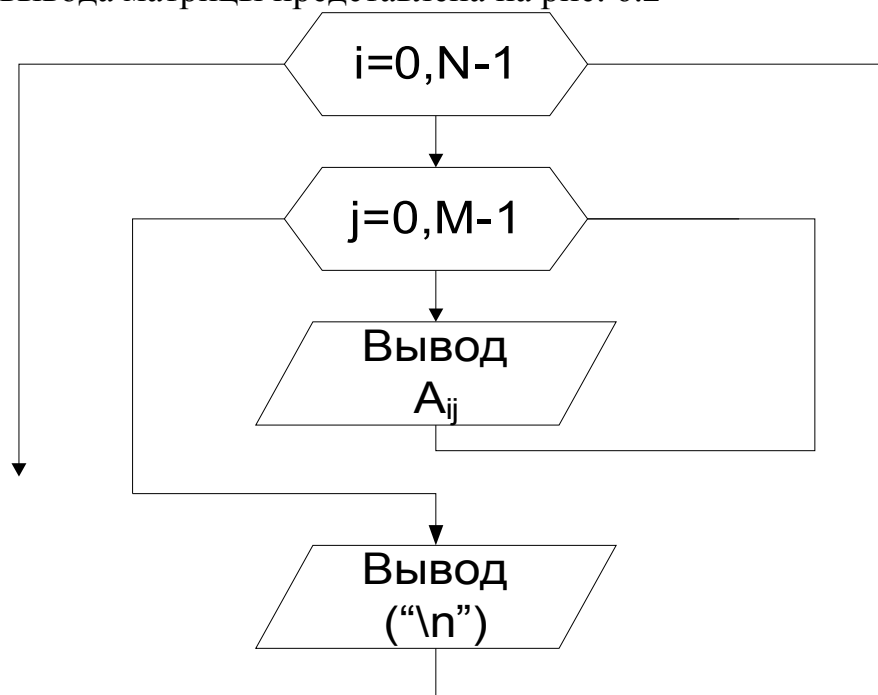


Рис 6.2. Вывод матрицы

С использованием функций printf и scanf построчный вывод матрицы можно реализовать следующим образом.

```

printf("\n Matrica A\n");
for (i=0; i<N; i++)
{for (j=0; j<M; j++)
    printf("%f\t", a[i][j]);
printf("\n");}

```

С использованием cin и cout построчный вывод матрицы можно реализовать следующим образом.

```

cout<<"\n Matrica A\n";
for (i=0; i<N; cout<<endl, i++)
for (j=0; j<M; j++)
    cout<<"\t"<<a[i][j]);

```

На рис. 6.3 представлена программа ввода-вывода матрицы и результаты ее работы.

```
main.cc x
1
2 #include <iostream>
3 using namespace std;
4 int main()
5 {
6     int i, j, N, M, a[20][20];
7     cout<<"N=";
8     cin>>N;
9     cout<<"M=";
10    cin>>M;
11    cout<<"Input Matrix A"<<endl;
12    for (i=0; i<N; i++)
13        for (j=0; j<M; j++)
14            cin>>a[i][j];
15    cout<<"Matrix A"<<endl;
16    for (i=0; i<N; i++)
17    {
18        for (j=0; j<M; j++)
19            cout<<a[i][j]<<"\t";
20        cout<<endl;
21    }
22    return 0;
23 }
24
```

Документы ▾ Задачи ▾

Терминал

```
N=4
M=5
Input Matrix A
1 2 3 4 5
6 7 8 9 0
-11 -12 -13 -14 -15
7 5 4 3 32
Matrix A
1      2      3      4      5
6      7      8      9      0
-11    -12    -13    -14    -15
7      5      4      3      32
```

Рис 6.3. Результаты работы программы ввода матрицы и ее вывода

Вначале в качестве максимального элемента запоминается первый элемент матрицы $A[0][0]$, номер строки $i_{max}=0$, $j_{max}=0$. Затем организуется цикл прохода по матрице, где каждый элемент сравнивается со значением max , и если

оказывается больше, то его значение запоминается как новое значение max, и также запоминаются его индексы. Блок-схема алгоритма представлена на рис. 6.4.

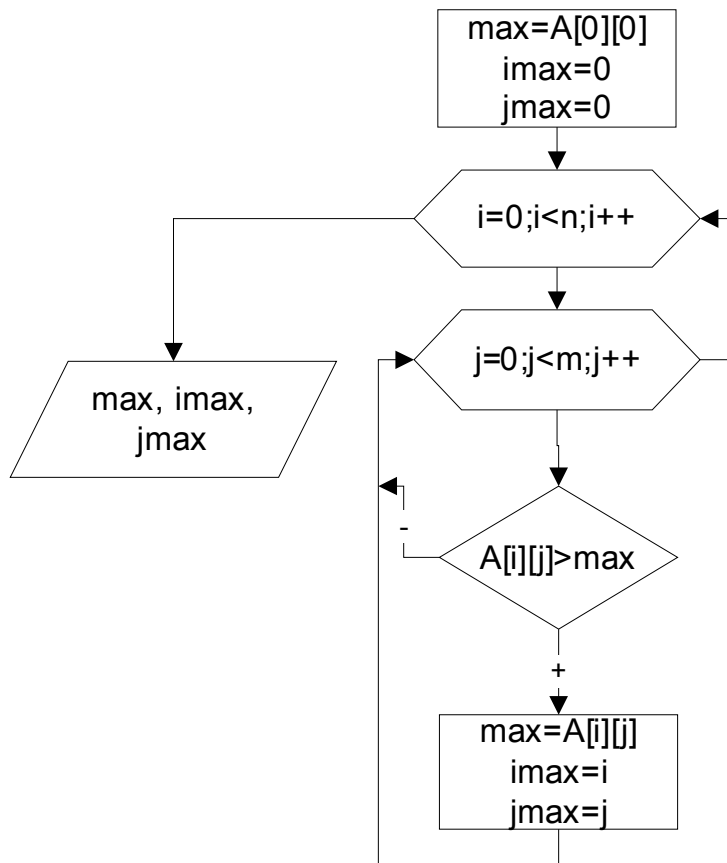


Рис 6.4. Поиск максимального элемента матрицы и его индексов

Программа поиска максимального элемента и его номера представлена ниже.

```

float a[20][20];
int i,j,n,m, imax, jmax;
float max;
// ввод матрицы
...
max=a[0][0];
imax=0; jmax=0;
for(i=0;i<n;i++)
    for(j=0;j<m;j++)
        if (a[i][j]<max)
        {
            max=a[i][j];
            imax=i;
            jmax=j;
        }
printf("\nmax=%f\n", max);
printf("\nimax=%d\n", imax);
printf("\njmax=%d\n", jmax);
  
```

Перед рассмотрением других алгоритмов, давайте вспомним некоторые свойства матриц (см. рис. 6.5):

- 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 *квадратной матрицы*, если его индексы удовлетворяют равенству $i + j + 1 = n$;
- неравенство $i + j + 1 < n$ характерно для элемента находящегося выше побочной диагонали *квадратной матрицы*;
- соответственно, элементу *квадратной матрицы*, лежащему ниже побочной диагонали соответствует выражение $i + j + 1 > n$.

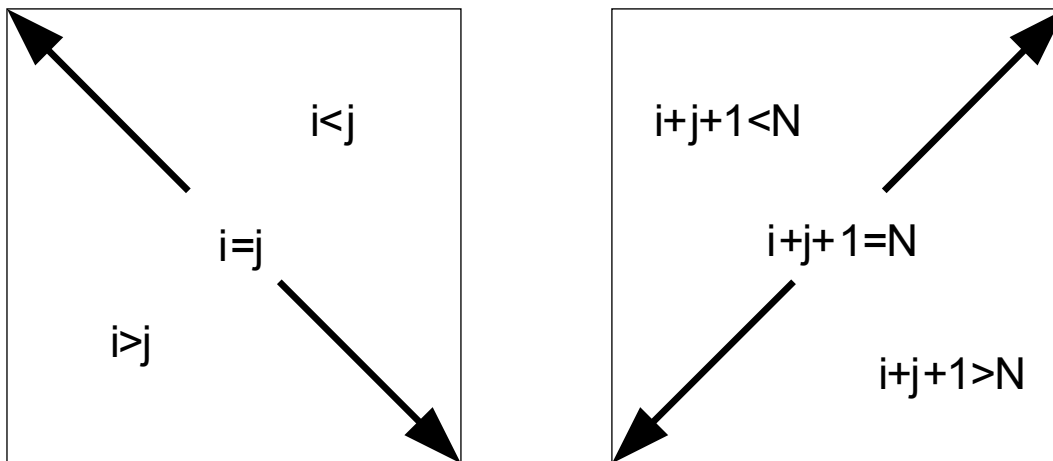


Рис 6.5. Соотношение индексов у квадратной матрицы

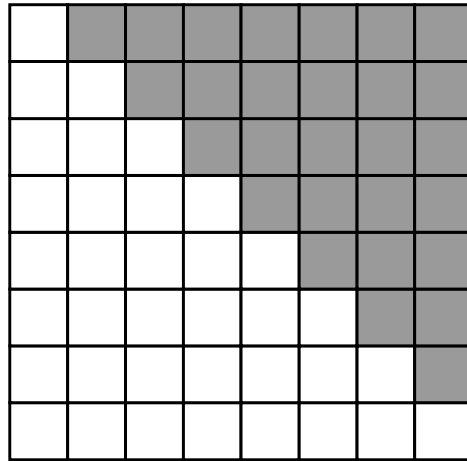
Вспомним из курса высшей математики некоторые типы квадратных матриц:

- матрица называется *единичной*, если все элементы нули, а на главной диагонали единицы;
- матрица называется *диагональной*, если все элементы нули, кроме главной диагонали;
- матрица *нулевая*, если все элементы нули;
- матрица называется *верхнетреугольной*, если все элементы ниже главной диагонали нули;
- матрица называется *нижнетреугольной*, если все элементы выше главной диагонали нули;
- матрица называется *симметричной*, если $A = A^T$.

Для проверки, что матрица В является обратной матрице А, нужно проверить условие $A \cdot B = E$ (E – единичная матрица).

Дальнейшие алгоритмы работы с матрицами рассмотрим на примерах решения задач.

ЗАДАЧА 6.1. Найти сумму элементов матрицы, лежащих выше главной диагонали.



Блок-схема решения задачи представлена на рис. 6.6.

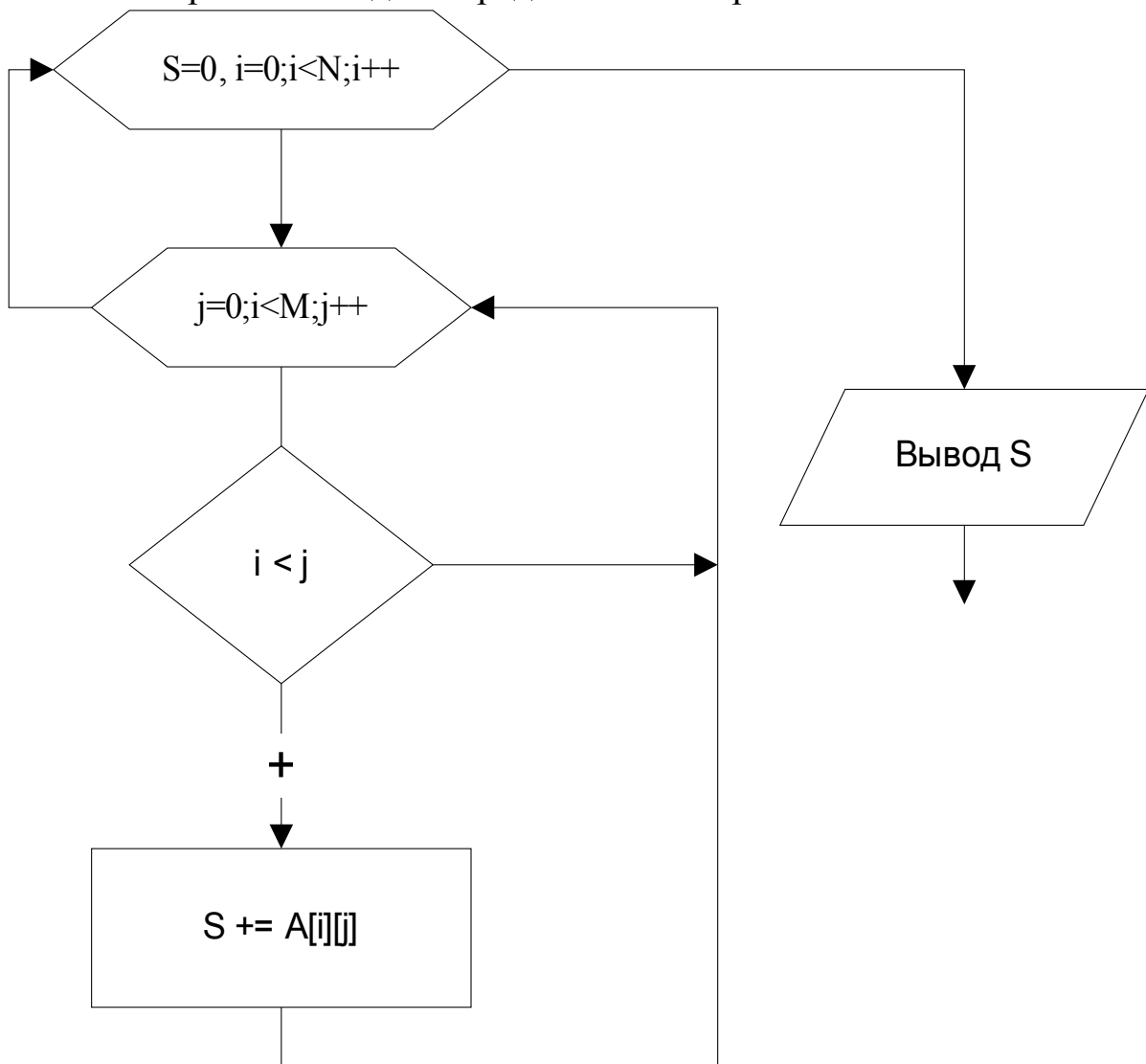


Рис.6.6. Блок-схема задачи 6.1.

```

int main()
{int S,i,j,N,M,a[20][20];
  cout<<"N=";cin>>N;
  cout<<"M=";cin>>M;
  cout<<"Input Matrix A"<<endl;

```

```

for (i=0; i<N; i++)
    for (j=0; j<M; j++)
        cin>>a[i][j];
for (S=i=0; i<N; i++)
    for (j=0; j<M; j++)
        if (j>i) S+=a[i][j];
cout<<"S="<<S<<endl; }

```

На рис.6.7 приведен второй алгоритм решения задачи.

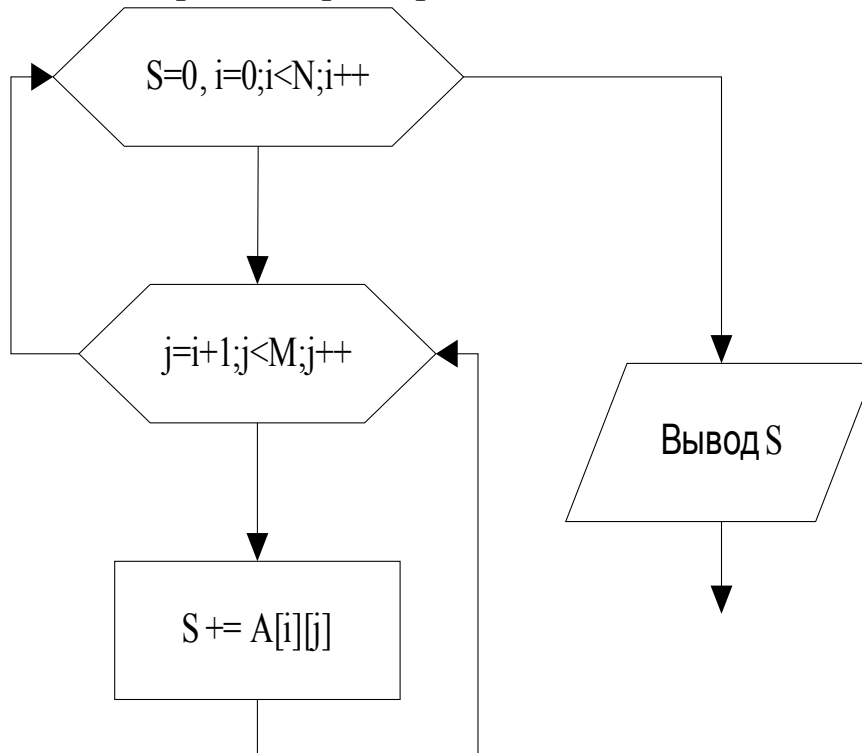
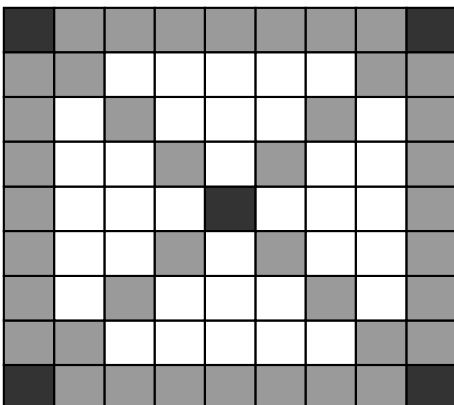


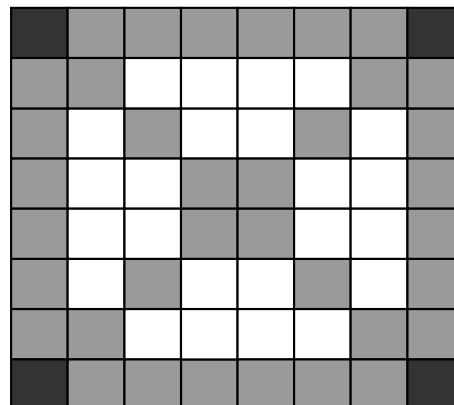
Рис 6.7. Второй вариант решения задачи 7.1

ЗАДАЧА 6.2. Вычислить количество положительных элементов квадратной матрицы, расположенных по ее периметру и на диагоналях. Блок-схема решения приведена на рис.6.7.

Матрица из N строк и N столбцов



N- нечетное



N - четное

Из соотношения индексов на побочной диагонали $i + j + 1 = n$ находим $j = n - i - 1$

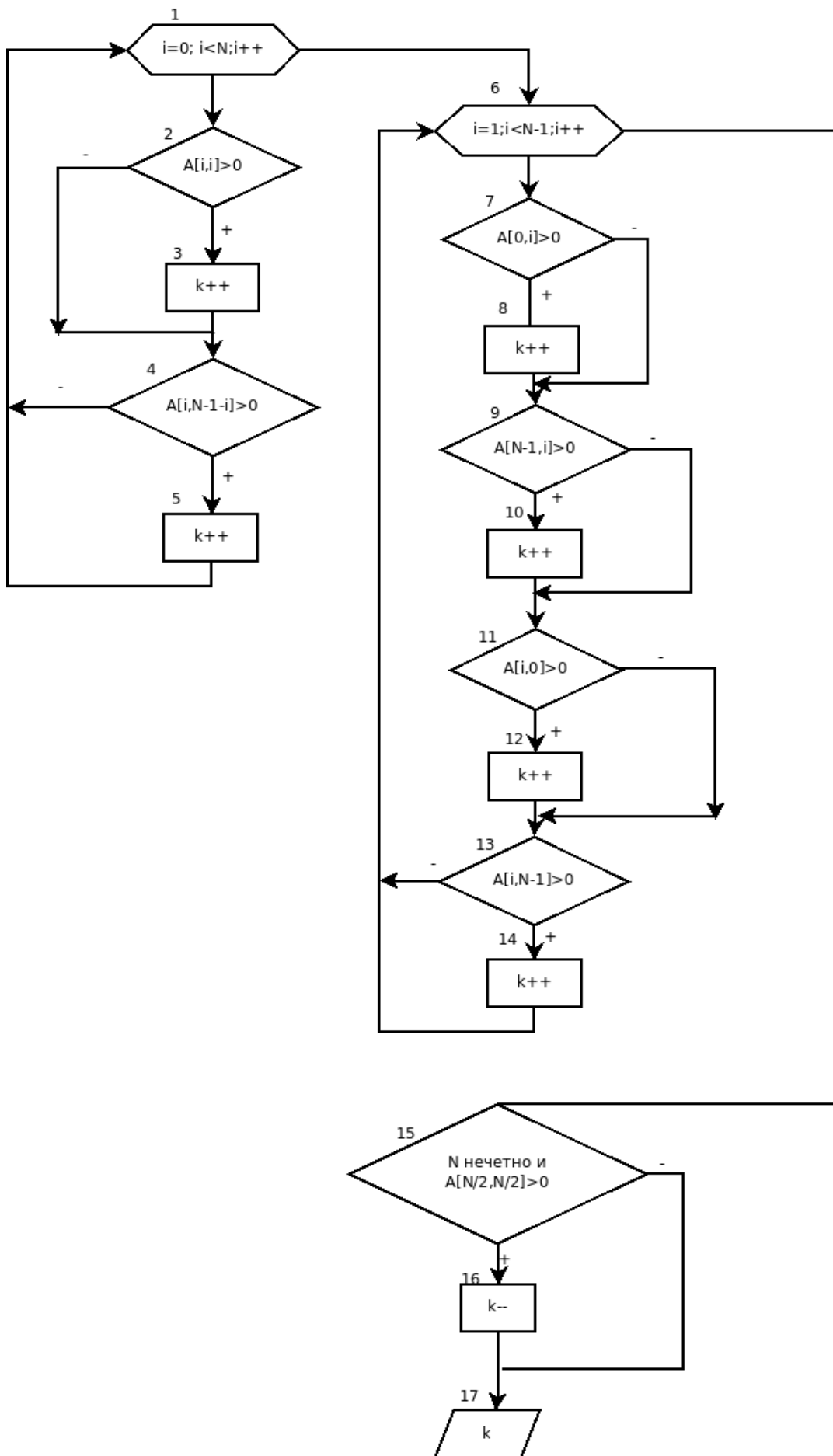


Рис 6.7. Блок-схема к примеру 6.2


```

#include <iostream>
using namespace std;
int main()
{int k,i,j,N,a[20][20];
cout<<"N=";cin>>N;
cout<<"Input Matrix A"<<endl;
for(i=0;i<N;i++)
for(j=0;j<N;j++)
cin>>a[i][j];
//цикл прохода по главной и побочной диагоналям
for(i=k=0;i<N;i++)
{ if(a[i][i]>0)k++;    if(a[i][N-i-1]>0)k++;}
//цикл прохода по первой и последней строкам матрицы,
//по первому и последнему столбцам,
//за исключением крайних элементов
for(i=1;i<N-1;i++)
{ if(a[0][i]>0)k++;    if(a[N-1][i]>0)k++;
  if(a[i][0]>0)k++;    if(a[i][N-1]>0)k++;}
//проверка, пересекаются ли диагонали, когда размерность
//матрицы - нечетное число
if ((N%2!=0)&&(a[N/2][N/2]>0))k--;
cout<<"k="<<k<<endl;}

```

ЗАДАЧА 6.3. Проверить, является ли заданная квадратная матрица единичной. Единичной называют матрицу, у которой элементы главной диагонали – единицы, а все остальные – нули.

```

#include <iostream>
using namespace std;
int main()
{int pr,i,j,N,a[20][20];
cout<<"N=";cin>>N;cout<<"Input Matrix A"<<endl;
for(i=0;i<N;i++)
for(j=0;j<N;j++)
cin>>a[i][j];
//Предполаем, что матрица – единичная, pr=1.
for(pr=1,i=0;i<N;i++)
for(j=0;j<N;j++)
//Если элемент лежит на главной диагонали и это не 1,
//или элемент вне главной диагонали и это не 0, то
// матрица не единичная, pr=0.
if (((i==j) && (a[i][j]!=1)) || ((i!=j) && (a[i][j]!=0)))
    { pr=0; break; }
if (pr) cout<<"Единичная матрица\n";
else cout<<"Матрица не является единичной\n";}

```

ЗАДАЧА 6.4. Поменять местами элементы главной и побочной диагонали матрицы $A(k,k)$. Алгоритм сводится к обмену элементов на главной и побочной диагоналях в каждой строке, его блок-схема приведена на рис.6.8.

```
#include <iostream>
using namespace std;
int main()
{
int i,j,k;
double b,a[20][20];
cout<<"k=";
cin>>k;
cout<<"Input Matrix A"<<endl;
for(i=0;i<k;i++)
    for(j=0;j<k;j++)
        cin>>a[i][j];
for(i=0;i<k;i++)
{
    b=a[i][i];
    a[i][i]=a[i][k-1-i];
    a[i][k-1-i]=b;}
cout<<"Output Matrix A"<<endl;
for(i=0;i<k;cout<<endl,i++)
    for(j=0;j<k;j++)
        cout<<a[i][j]<<"\t";
}
```

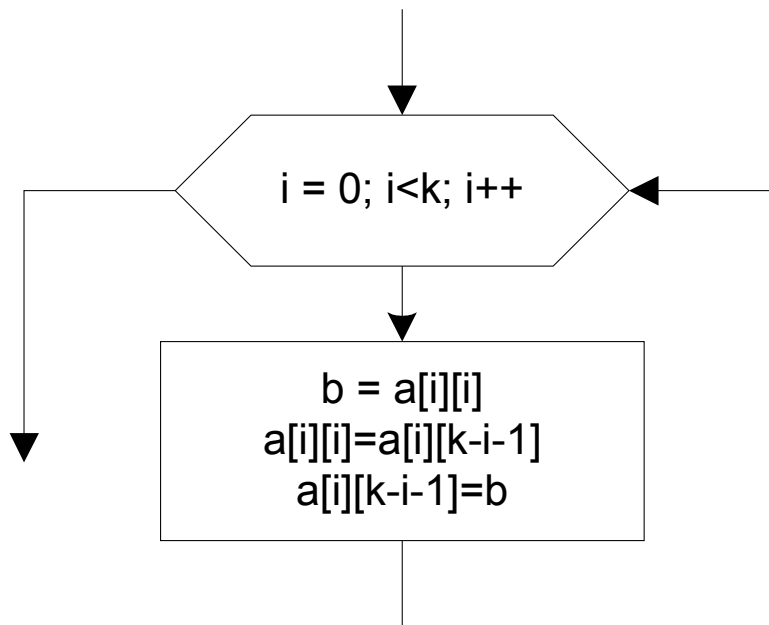


Рис 6.8. Блок-схема к примеру 6.4

ЗАДАЧА 6.5. Преобразовать исходную матрицу $A(N,M)$ так, чтобы нулевой элемент каждой строки был заменен средним арифметическим элементов этой

строки. Необходимо в каждой строке матрицы найти сумму элементов, разделить на количество элементов в строке и полученный результат записать в первый элемент строки. Блок-схема изображена на рис.6.9.

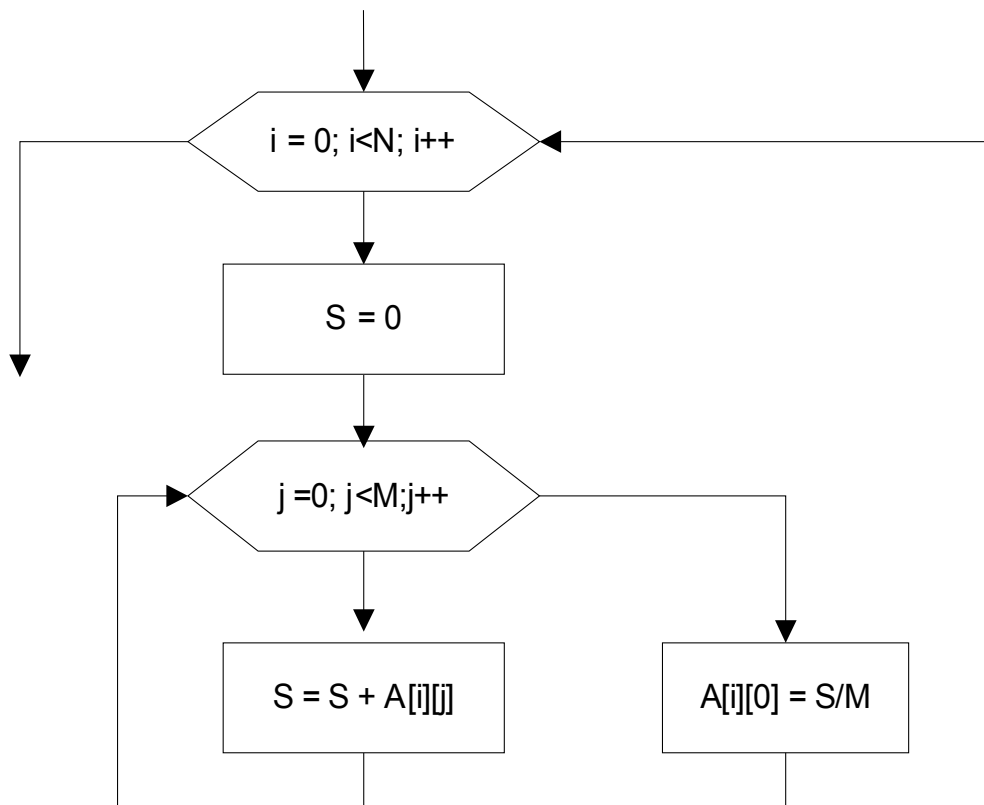


Рис 6.9. Блок-схема к примеру 6.5

```

#include <iostream>
using namespace std;
int main()
{int i, j, N, M;
double S, a[20][20];
cout<<"N=";    cin>>N;
cout<<"M=";    cin>>M;
cout<<"Input Matrix A"<<endl;
for (i=0; i<N; i++)
    for (j=0; j<M; j++)
        cin>>a[i][j];
for (i=0; i<N; a[i][0]=S/M, i++)
    for (S=j=0; j<M; j++)        S+=a[i][j];
cout<<"Output Matrix A"<<endl;
for (i=0; i<N; cout<<endl, i++)
    for (j=0; j<M; j++)
        cout<<a[i][j]<<"\t"; }
  
```

ЗАДАЧА 6.6. Преобразовать матрицу $A(m, n)$ так, чтобы строки с нечетными индексами были упорядочены по убыванию, с четными – по возрастанию. Перебираем все строки, если номер строки четный, то то упорядочиваем эту

строку по возрастанию методом пузырька, иначе - по убыванию методом пузырька. Блок-схема решения задачи приведена на рис.6.10.

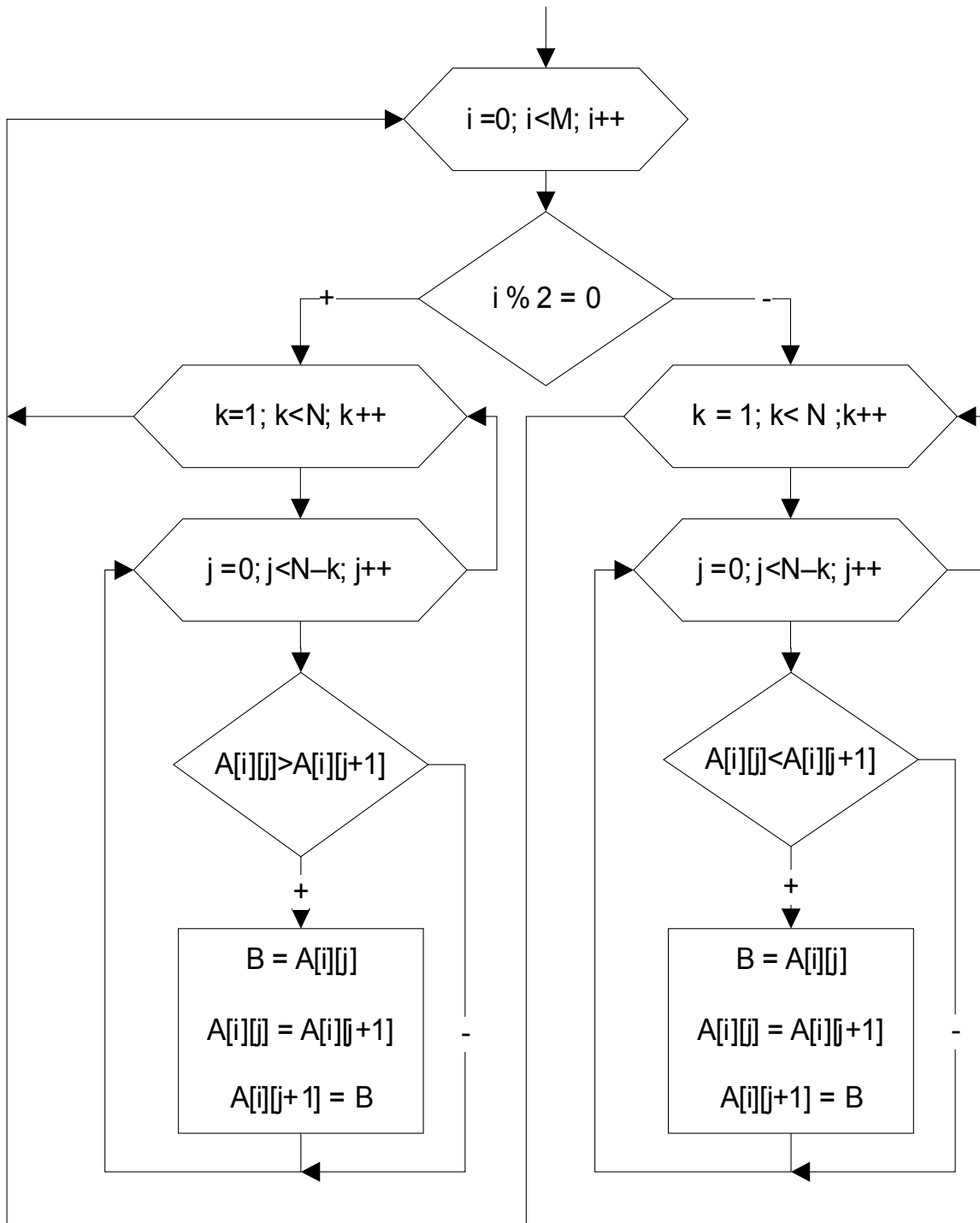


Рис 6.10. Блок-схема к примеру 6.6

```

#include <iostream>
using namespace std;
int main()
{int i, j, k, N, M;
double b, a[20][20];
cout<<"M=";cin>>M;
cout<<"N=";cin>>N;

```

```

cout<<"Input Matrix A"<<endl;
for(i=0;i<M;i++)
for(j=0;j<N;j++)
cin>>a[i][j];
for(i=0;i<M;i++)
//Проверка номера строки на четность
if(i%2==0)
{ //Упорядочение четной строки по возрастанию
for(k=1;k<N;k++)
for(j=0;j<N-k;j++)
if(a[i][j]>a[i][j+1])
{
b=a[i][j];
a[i][j]=a[i][j+1];
a[i][j+1]=b;
}
}
else //Упорядочение нечетной строки по убыванию
for(k=1;k<N;k++)
for(j=0;j<N-k;j++)
if(a[i][j]<a[i][j+1])
{
b=a[i][j];
a[i][j]=a[i][j+1];
a[i][j+1]=b;
}
cout<<"Output Matrix A"<<endl;
for(i=0;i<M;cout<<endl,i++)
for(j=0;j<N;j++)
cout<<a[i][j]<<"\t";
}

```

6.2. ДИНАМИЧЕСКИЕ МАТРИЦЫ

1-й способ работы с динамическими матрицами основан на работе с одинарным указателем. При работе с динамическими матрицами следует помнить, что выделенный участок памяти под матрицу $A(N, M)$ представляет собой участок памяти размером $N \times M$ элементов. Поэтому выделение памяти будет выглядеть следующим образом:

```
A=(тип *) calloc(n*m, sizeof(тип))
```

или

```
A=(тип *) malloc(n*m*sizeof(тип))
```

Для обращения к элементу $A_{i,j}$ необходимо, по номеру строки i и номеру столбца j вычислить номер этого элемента k в одномерном динамическом массиве. Учитывая, что в массиве элементы нумеруются с нуля $k=i \cdot M+j$.

Статический элемент матрицы $a[i][j]$ записывается как $*(a+i*m+j)$

2-й способ работы с динамическими матрицами основан на использовании двойного указателя – указателя на указатель.

```
float **a;
```

Это указатель на `float *`, или указатель на массив.

Выделение и очистка памяти в этом случае осуществляется следующим образом:

```
void main()
{
    int n,m;
    float **a;
    //Создали массив указателей в количестве n штук на float,
    // каждый //элемент массива, является адресом, в котором
    // хранится указатель на float.
    a=new float *[n];
    //Осталось определить значение этого указателя. Для этого
    //организуем цикл от 0 до n-1, в котором каждый указатель
    //будет адресовать участок памяти, в котором хранится m
    // элементов.
    for(i=0;i<n;i++)
        a[i]=new float(m);
    //Обращение к элементу матрицы в этом случае будет идти
    // стандартным образом a[i][j]
    //По окончании работы необходимо освободить память
    for(i=0;i<n;i++)
        delete a[i];
    delete [];
```

ЗАДАЧА 6.7. Задана матрица $A(N,M)$. Поменять местами ее максимальный и минимальный элементы. Блок-схема расчетной части задачи изображена на рис.6.11.

```
#include <iostream>
using namespace std;
//Решение задачи с использованием одномерного
// динамического массива
int main()
{int i,j,imax,jmax,imin,jmin,N,M;
double min,max,b,*a;
cout<<"N=";cin>>N;
cout<<"M=";cin>>M;
```

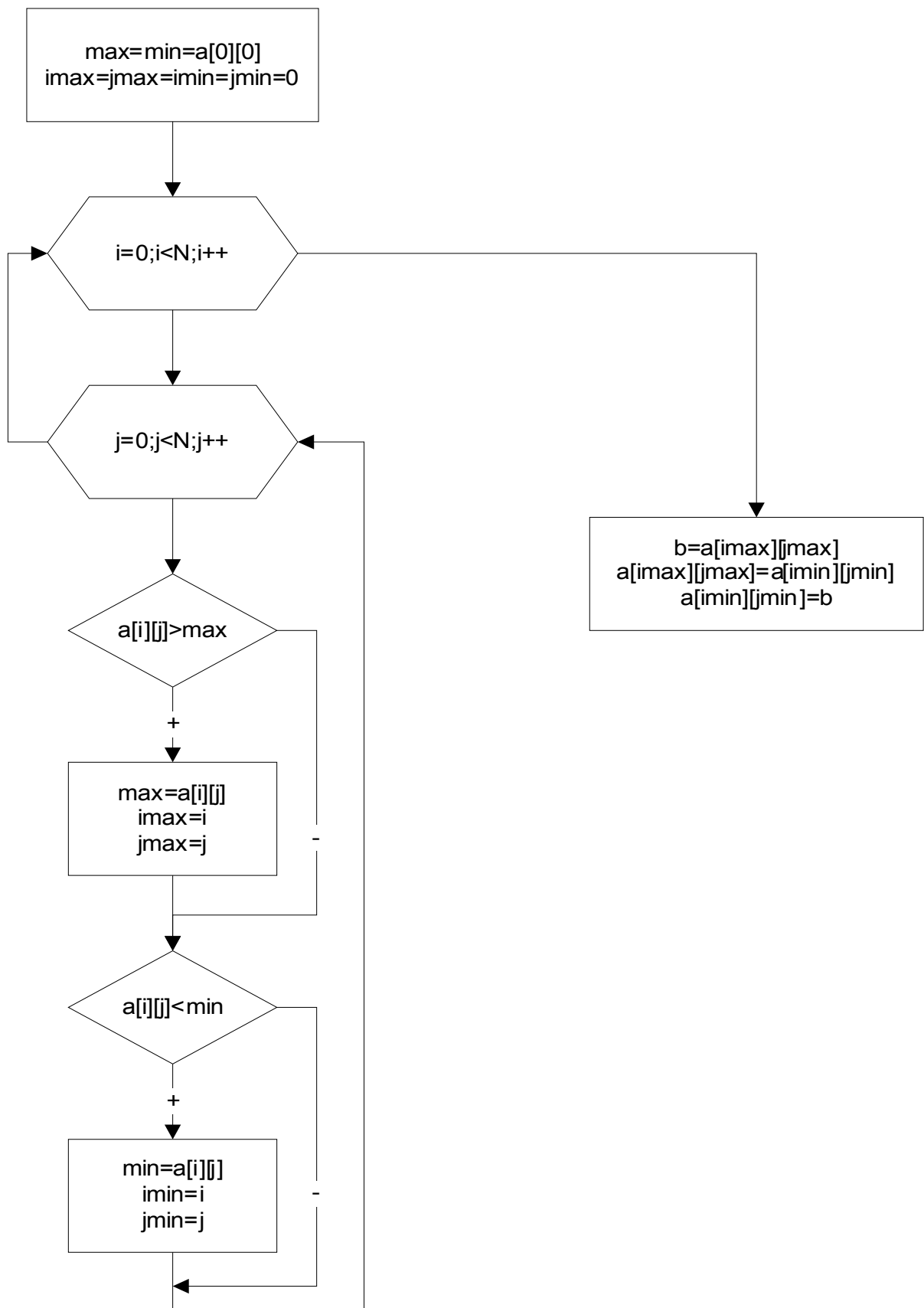


Рис 6.11. Блок-схема к задаче 6.7

```

A=(double *) calloc(n*m, sizeof(double));
cout<<"Input Matrix A"<<endl;
for(i=0;i<N;i++)

```

```

    for (j=0; j<M; j++)
        cin>>* (a+i*M+j);
for (max=min=*a, imax=jmax=imin=jmin=i=0; i<N; i++)
for (j=0; j<M; j++)
{   if ((*(a+i*M+j))>max)
        {max=*(a+i*M+j); imax=i; jmax=j;}
    if ((*(a+i*M+j))<min)
        {min=*(a+i*M+j); imin=i; jmin=j;}}
b=*(a+imax*M+jmax); *(a+imax*M+jmax)=*(a+imin*M+jmin);
*(a+imin*M+jmin)=b;
cout<<"Output Matrix A"<<endl;
for (i=0; i<N; cout<<endl, i++)
    for (j=0; j<M; j++)
        cout<<* (a+i*M+j)<<"\t"; }

```

ЗАДАЧА 6.8. Задана матрица $A(n, m)$. Сформировать вектор $P(m)$, в который записать номера строк максимальных элементов каждого столбца.

Алгоритм решения этой задачи следующий (рис. 6.12): для каждого столбца матрицы находим максимальный элемент и его номер, номер максимального элемента j -го столбца матрицы записываем в j -й элемент массива P .

```

#include <iostream>
using namespace std;
//Решение задачи с использованием двойного указателя.
int main()
{float max;float **a;int *p;int i, j, n, m, nmax;
cout<<"n=";      cin>>n;
cout<<"m=";      cin>>m;
a=new float *[n];
for (i=0; i<n; i++)
    a[i]=new float (m);
p=new int [m];
cout<<"Vvod matrici"<<endl;
for (i=0; i<n; i++)
    for (j=0; j<m; j++)
        cin>>a[i][j];
cout<<"Matrica"<<endl;
for (i=0; i<n; i++)
{   for (j=0; j<m; j++)
        cout<<a[i][j]<<"\t";
    cout<<endl;}
cout<<"Massiv P"<<endl;
for (j=0; j<m; j++)
{ //поиск максимального элемента в j-м столбце и его
//номера i
    max=a[0][j];

```



```

nmax=0;
for(i=1;i<n;i++)
if (a[i][j]>max)
{
max=a[i][j];
nmax=i;
}
//записываем найденный nmax в i-й элемент массива p
p[j]=nmax;
cout<<p[j]<<"\t";
}
cout<<endl;
delete [] a;
return 0;
}

```

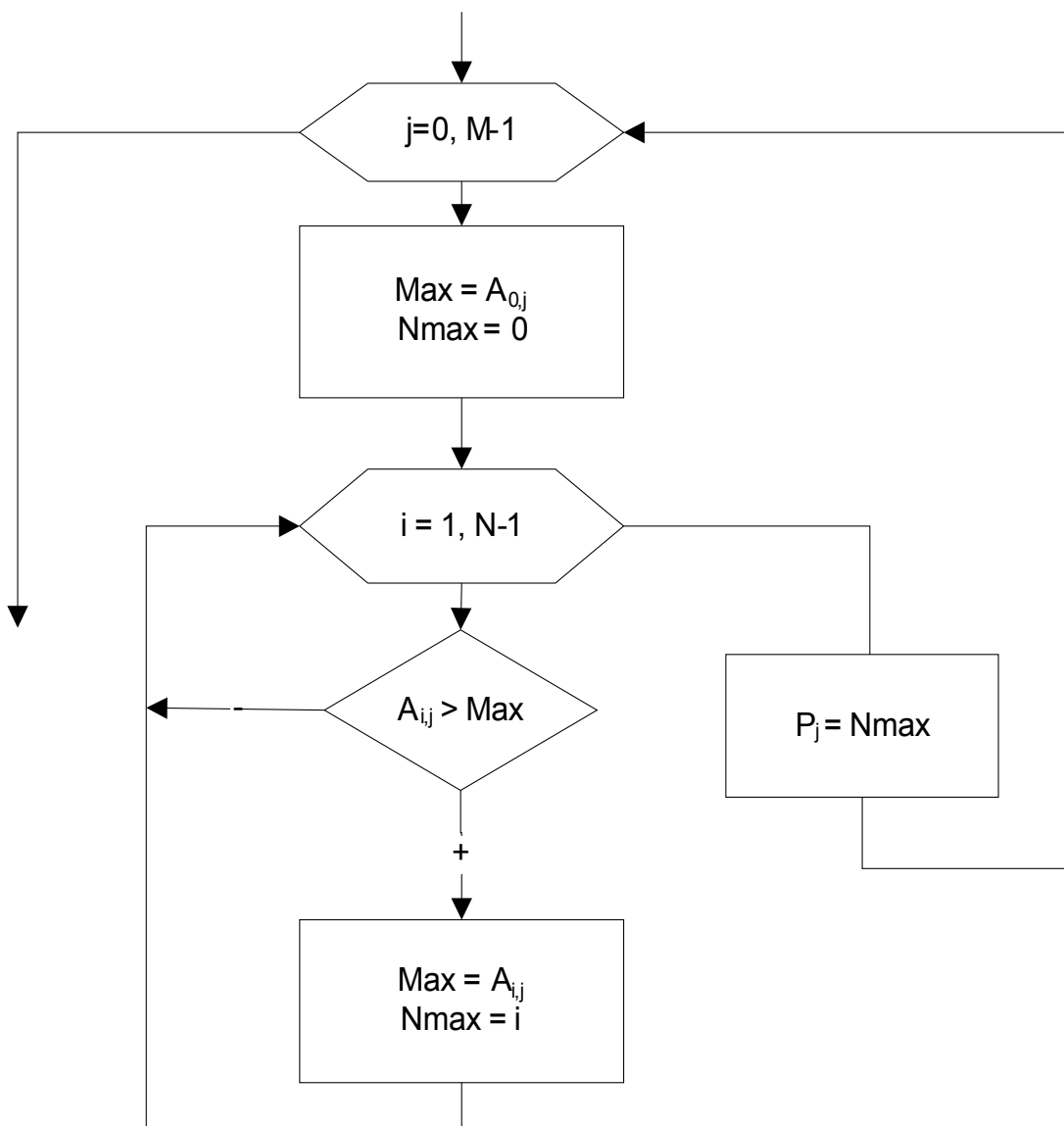


Рис 6.12. Блок-схема к примеру 6.8

ЗАДАЧА 6.9. Написать программу умножения двух матриц вещественных чисел $A(N,M)$ и $B(M,L)$.

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{10} \\ b_{20} & b_{21} \end{pmatrix} = \begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{10} \\ c_{20} & c_{21} \end{pmatrix}$$

$$\begin{pmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} \end{pmatrix}$$

В общем виде формула для нахождения элемента C_{ij} матрицы имеет вид:

$$C_{i,j} = \sum_{k=0}^{M-1} A_{ik} B_{kj}, \quad i = 0, N-1 \quad j = 0, L-1.$$

Нужно обратить внимание, что перемножать матрицы можно только в том случае, если количество строк левой матрицы совпадает с количеством столбцов правой. Кроме того $A \times B \neq B \times A$. Блок-схема перемножения двух матриц приведена на рис. 6.13

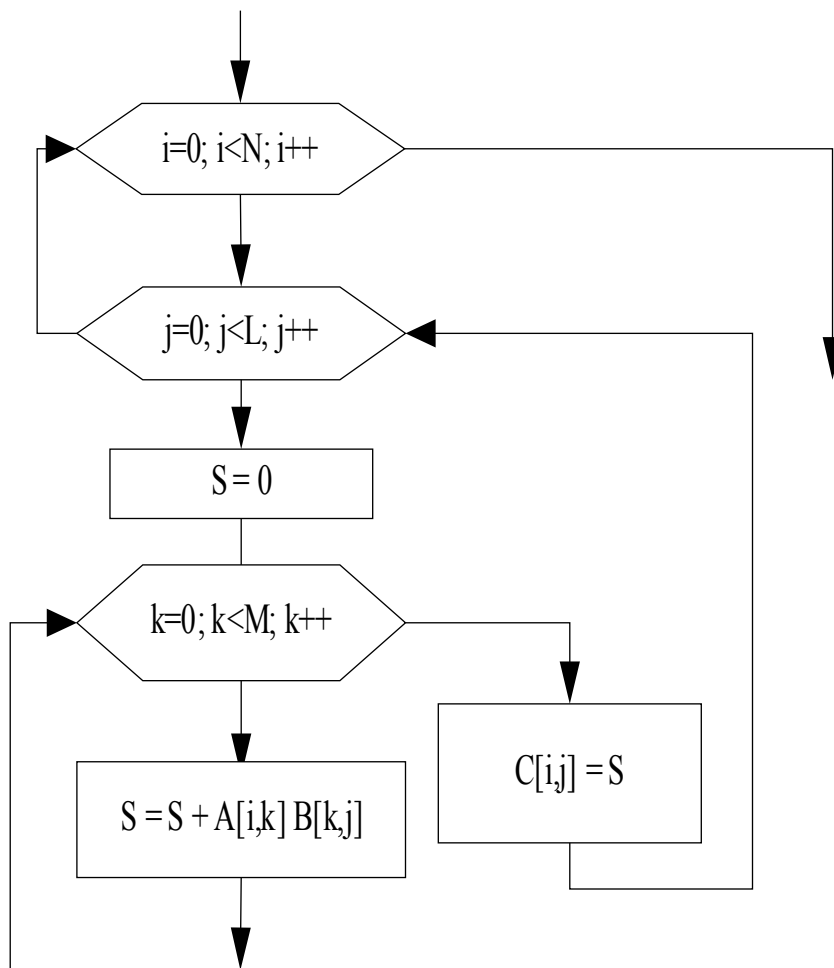


Рис 6.13. Блок-схема перемножения двух матриц

```

int main()
{int i, j, k, N, L, M;
double **a, **b, **c;

```

```

cout<<"N=";          cin>>N;
cout<<"M=";          cin>>M;
cout<<"L=";          cin>>L;
a=new double *[N];
for(i=0;i<N;i++)
    a[i]=new double[M];
b=new double *[M];
for(i=0;i<M;i++)
    b[i]=new double[L];
c=new double *[N];
for(i=0;i<N;i++)
    c[i]=new double[L];
cout<<"Input Matrix A"<<endl;
for(i=0;i<N;i++)
    for(j=0;j<M;j++)
        cin>>a[i][j];
cout<<"Input Matrix B"<<endl;
for(i=0;i<M;i++)
    for(j=0;j<L;j++)
        cin>>b[i][j];
for(i=0;i<N;i++)
    for(j=0;j<L;j++)
        for(c[i][j]=0,k=0;k<M;k++)
            c[i][j]+=a[i][k]*b[k][j];
cout<<"Matrix C"<<endl;
for(i=0;i<N;cout<<endl,i++)
    for(j=0;j<L;j++)
        cout<<c[i][j]<<"\t";
for(i=0;i<N;i++)        delete [] a[i];
delete [] a;
for(i=0;i<M;i++)        delete [] b[i];
delete [] b;
for(i=0;i<N;i++)        delete [] c[i];
delete [] c;}

```

Лекция 7. Решение задач линейной алгебры с использованием динамических матриц и функций

Рассмотрим решение следующих задач линейной алгебры:

- решение систем линейных алгебраических уравнений;
- вычисление обратной матрицы;
- вычисление определителя.

7.1. Решение систем линейных алгебраических уравнений методом Гаусса

ЗАДАЧА 7.1. Решить систему линейных алгебраических уравнений (7.1).

$$\begin{aligned}
 a_{00}x_0 + a_{01}x_1 + \dots + a_{0n-1}x_{n-1} &= b_0 \\
 a_{10}x_0 + a_{11}x_1 + \dots + a_{1n-1}x_{n-1} &= b_1 \\
 &\dots \\
 a_{n-10}x_0 + a_{n-11}x_1 + \dots + a_{n-1n-1}x_{n-1} &= b_{n-1}
 \end{aligned} \tag{7.1}$$

Через матрицу A обозначим матрицу коэффициентов системы, через вектор B – вектор свободных членов, X – вектор неизвестных:

$$A = \begin{pmatrix} a_{00} & a_{01} & \dots & a_{0n-1} \\ a_{10} & a_{11} & \dots & a_{1n-1} \\ \dots & \dots & \dots & \dots \\ a_{n-10} & a_{n-11} & \dots & a_{n-1n-1} \end{pmatrix} \quad b = \begin{pmatrix} b_0 \\ b_1 \\ \dots \\ b_{n-1} \end{pmatrix} \quad x = \begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{n-1} \end{pmatrix}$$

$$Ax = b.$$

Первый этап - *прямой ход метода Гаусса*, заключается в приведении расширенной матрицы (7.2) к *треугольному виду* (7.3). Это означает, что все элементы матрицы (7.2) ниже главной диагонали должны быть равны нулю.

$$A' = \begin{pmatrix} a_{00} & a_{01} & \dots & a_{0n-1} & b_0 \\ a_{10} & a_{11} & \dots & a_{1n-1} & b_1 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n-10} & a_{n-11} & \dots & a_{n-1n-1} & b_{n-1} \end{pmatrix} \tag{7.2}$$

$$A' = \begin{pmatrix} a_{00} & a_{01} & \dots & a_{0n-1} & b_0 \\ 0 & a_{11} & \dots & a_{1n-1} & b_1 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{n-1n-1} & b_{n-1} \end{pmatrix} \tag{7.3}$$

Для формирования нулевого столбца матрицы (7.3) необходимо из каждой строки (начиная со первой) вычесть нулевую, умноженную на некоторое число M .

В общем виде можно записать так:

$$1\text{-я строка} = 1\text{-я строка} - M \times 0\text{-я строка}$$

$$2\text{-я строка} = 2\text{-я строка} - M \times 0\text{-я строка}$$

...

$$i\text{-я строка} = i\text{-я строка} - M \times 0\text{-я строка}$$

...

$$n-1\text{-я строка} = n-1\text{-я строка} - M \times 0\text{-я строка}$$

Преобразование элементов первой строки будет происходить по формулам:

$$a_{10} = a_{10} - Ma_{00} \qquad a_{11} = a_{11} - Ma_{01} \dots$$

$$a_{1i} = a_{1i} - Ma_{0i} \qquad a_{1n-1} = a_{1n-1} - Ma_{0n-1} \dots$$

$$b_1 = b_1 - Mb_0$$

$$a_{10} - Ma_{00} = 0$$

$$M = \frac{a_{10}}{a_{00}}$$

Элементы второй строки и коэффициент M можно рассчитать аналогично:

$$a_{20} = a_{20} - Ma_{00} \qquad a_{21} = a_{21} - Ma_{01} \dots$$

$$a_{2i} = a_{2i} - Ma_{0i} \qquad a_{2n-1} = a_{2n-1} - Ma_{0n-1} \dots$$

$$b_2 = b_2 - Mb_0$$

$$a_{20} - Ma_{00} = 0$$

$$M = \frac{a_{20}}{a_{00}}$$

Таким образом, преобразование элементов i -й строки будет происходить следующим образом:

$$a_{i0} = a_{i0} - Ma_{00} \qquad a_{il} = a_{il} - Ma_{0l} \dots$$

$$a_{in-1} = a_{in-1} - Ma_{0n-1} \qquad b_i = b_i - Mb_0$$

$$a_{i0} - Ma_{00} = 0$$

Коэффициент M для i -й строки будет равен:

$$M = \frac{a_{ik}}{a_{kk}}$$

Далее необходимо повторить описанный выше алгоритм для следующих столбцов матрицы (7.2), причем начинать преобразовывать первый столбец со

второго элемента, второй столбец – с третьего и т.д. В результате система уравнений примет вид (7.4). Алгоритм этого процесса изображен на рис. 7.1.

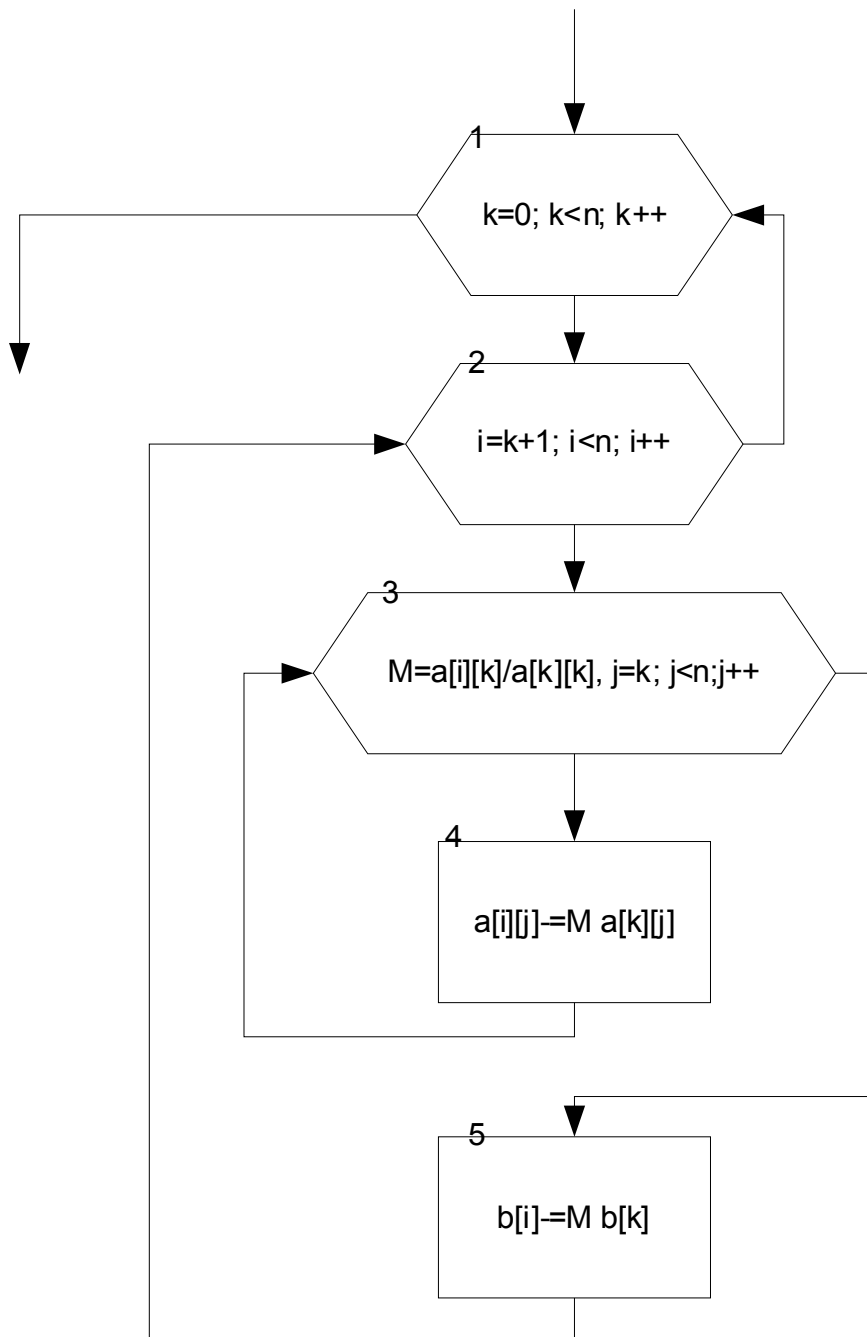


Рис 7.1. Блок-схема алгоритма преобразования расширенной матрицы к треугольному виду

$$\begin{aligned}
 a_{00} x_0 + a_{01} x_1 + a_{02} x_2 + \dots + a_{0n-1} x_{n-1} &= b_0 \\
 a_{11} x_1 + a_{12} x_2 + \dots + a_{1n-1} x_{n-1} &= b_1 \\
 a_{22} x_2 + \dots + a_{2n-1} x_{n-1} &= b_2 \\
 &\dots \\
 a_{n-1n-1} x_{n-1} &= b_{n-1}
 \end{aligned}
 \tag{7.4}$$

Если в матрице (7.2) на главной диагонали встретится элемент a_{kk} , равный нулю, то расчет коэффициента M для k -й строки будет невозможен. Избежать деления на ноль можно, избавившись от нулевых элементов на главной диагонали. Для этого перед обнулением элементов в k -м столбце необходимо найти в нем максимальный по модулю элемент (среди расположенных ниже a_{kk}), запомнить номер строки, в которой он находится, и поменять ее местами с k -й строкой. Алгоритм, отображающий эти преобразования, приведен на рис. 7.2.

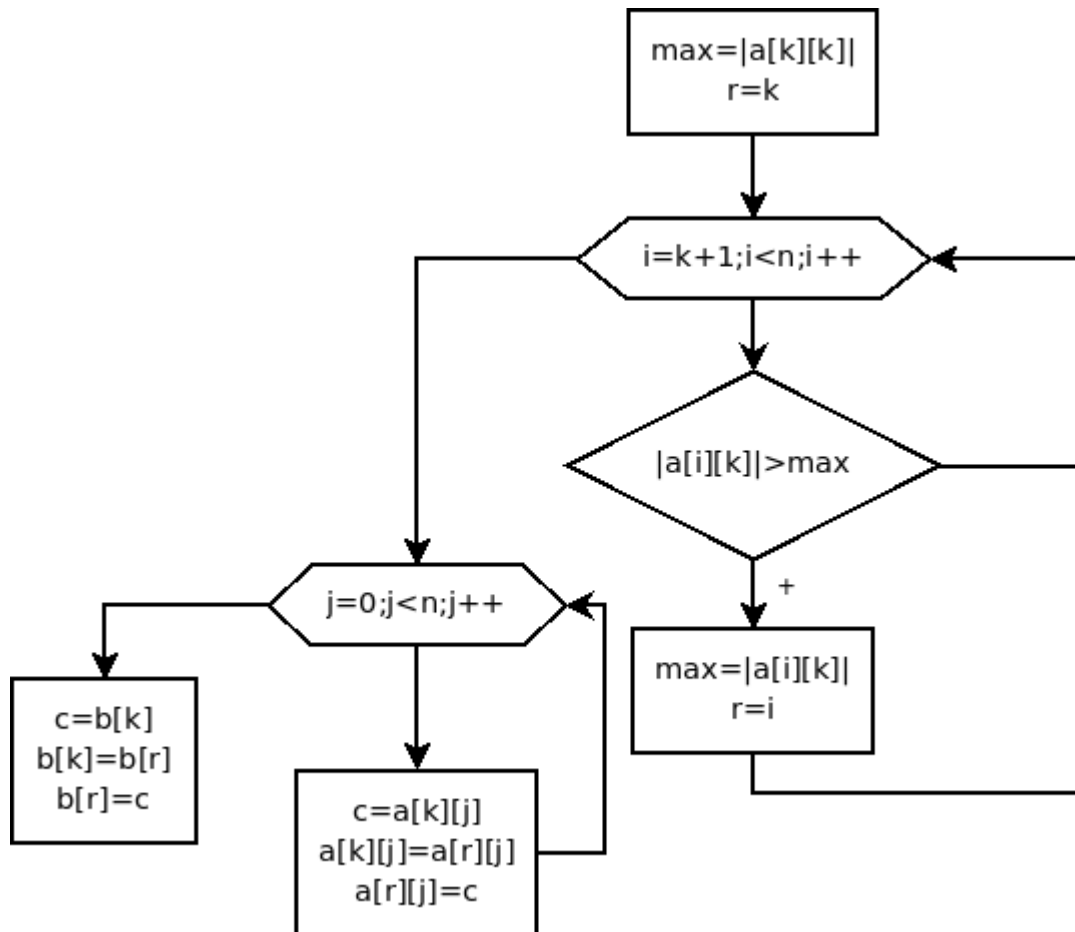


Рис 7.2. Блок-схема алгоритма перестановки строк расширенной матрицы. Решение системы (7.4) называют *обратным ходом метода Гаусса*.

Последнее $(n-1)$ -е уравнение системы (7.4) имеет вид:

$$a_{n-1n-1} x_{n-1} = b_{n-1} .$$

Если $a_{n-1n-1} \neq 0$, то $x_{n-1} = \frac{b_{n-1}}{a_{n-1n-1}}$.

В случае, если $a_{n-1n-1} = 0$ и $b_{n-1} = 0$, система имеет бесконечное множество решений.

При $a_{n-1n-1} = 0$ и $b_{n-1} \neq 0$ система решений не имеет.

Предпоследнее $(n-2)$ -е уравнение системы (7.4) имеет вид

$$a_{n-2n-2} x_{n-2} + a_{n-2n-1} x_{n-1} = b_{n-2} .$$

Откуда

$$x_{n-2} = \frac{b_{n-2} - a_{n-2n-1} x_{n-1}}{a_{n-2n-2}} .$$

Следующее $(n-3)$ -е уравнение системы (7.4) будет выглядеть так:

$$a_{n-3n-3} x_{n-3} + a_{n-3n-2} x_{n-2} + a_{n-3n-1} x_{n-1} = b_{n-3} ,$$

и откуда находим:

$$x_{n-3} = \frac{b_{n-3} - a_{n-3n-2} x_{n-2} - a_{n-3n-1} x_{n-1}}{a_{n-3n-3}}$$

Отсюда имеем:

$$x_{n-3} = b_{n-3} - \frac{(a_{n-3n-2} x_{n-2} + a_{n-2n-1} x_{n-1})}{a_{n-3n-3}} = \frac{b_{n-3} - \sum_{j=n-2}^{n-1} a_{n-3j} x_j}{a_{n-3n-3}} .$$

Таким образом, формула для вычисления i -го значения x будет иметь вид:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n-1} a_{ij} x_j}{a_{ii}} \quad i=n-1, \dots, 0$$

Алгоритм, реализующий обратный ход метода Гаусса, представлен в виде блок-схемы на рис. 7.3.

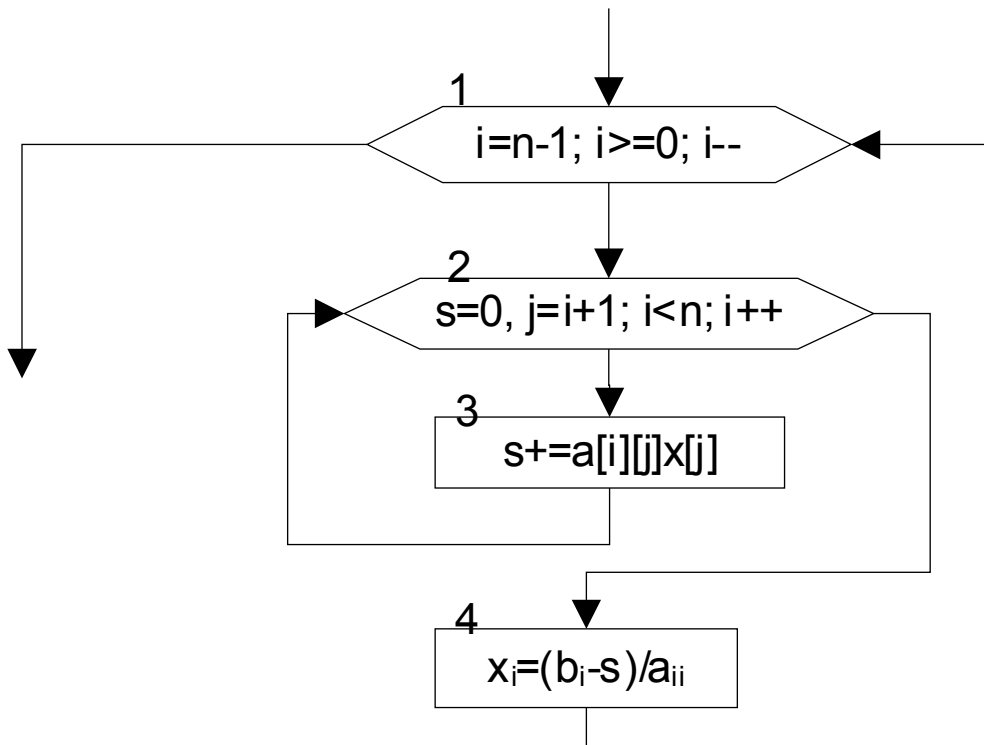


Рис 7.3. Блок-схема обратного хода метода Гаусса
 На рис.7.4 изображена общая блок-схема метода Гаусса.

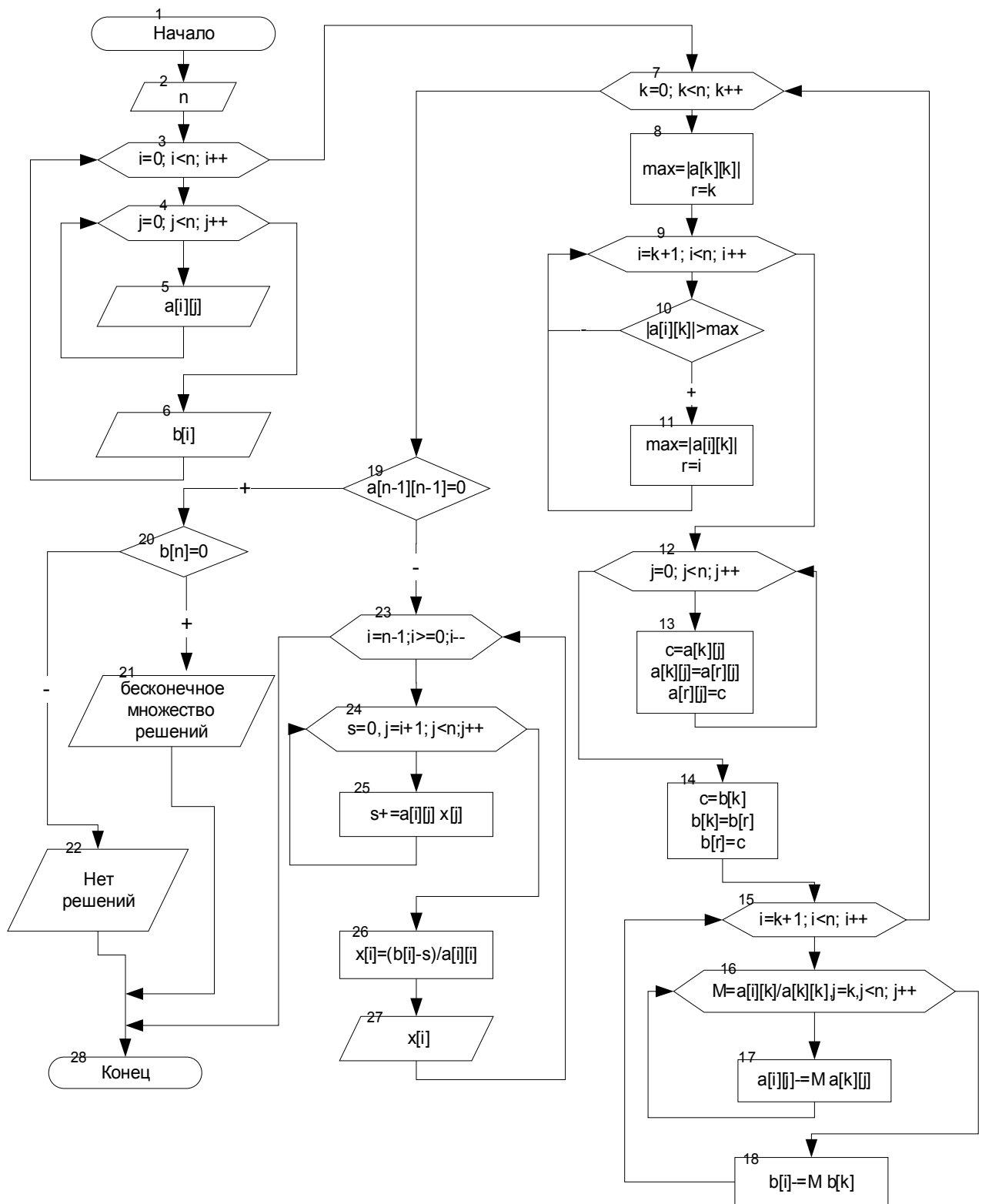


Рис 7.4. Блок-схема метода Гаусса

Теперь алгоритм решения СЛАУ, представленный на рис. 7.4 разобьем на главную функцию main() и функцию решения СЛАУ методом Гаусса. В функции main() вводятся исходные данные, затем вызывается функция SLAU решения системы линейных алгебраических уравнений и выводится вектор решения. Функция SLAU предназначена для решения системы линейных алгебраических уравнений методом Гаусса.

При написании функции следует учитывать следующее: в методе Гаусса изменяются матрица коэффициентов и вектор правых частей. Поэтому, для того чтобы их не испортить, в функции SLAU матрицу коэффициентов и вектор правых частей необходимо скопировать во внутренние (рабочие) переменные, и в функции обрабатывать внутренние переменные-копии.

Функция SLAU возвращает значение 0, если решение найдено, 1 – если система имеет бесконечное множество решений, 2 – если система не имеет решений.

```
int SLAU(double **matrica_a,int n,double *massiv_b,
double *x)
{
    int i,j,k,r;
    double c,M,max,s, **a, *b;
    a=new double *[n];
    for(i=0;i<n;i++)
    a[i]=new double[n];
    b=new double [n];
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            a[i][j]=matrica_a[i][j];
    for(i=0;i<n;i++)
        b[i]=massiv_b[i];
    for(k=0;k<n;k++)
    {
        max=fabs(a[k][k]);
        r=k;
        for(i=k+1;i<n;i++)
            if (fabs(a[i][k])>max)
            {
                max=fabs(a[i][k]);
                r=i;
            }
        for(j=0;j<n;j++)
        {
            c=a[k][j];    a[k][j]=a[r][j];
            a[r][j]=c;
        }
        c=b[k];b[k]=b[r];b[r]=c;
        for(i=k+1;i<n;i++)
        {
            for(M=a[i][k]/a[k][k],j=k;j<n;j++)
                a[i][j]-=M*a[k][j];
            b[i]-=M*b[k];
        }
    }
    if (a[n-1][n-1]==0)
```

```

        if(b[n-1]==0)
            return -1;
        else return -2;
else
{
    for(i=n-1;i>=0;i--)
    {
        for(s=0,j=i+1;j<n;j++)
            s+=a[i][j]*x[j];
        x[i]=(b[i]-s)/a[i][i];
    }
    return 0;}
for(i=0;i<n;i++)
    delete [] a[i];
delete [] a;
delete [] b;
}
int main()
{int result,i,j,N;
double **a, *b, *x;
cout<<"N=";          cin>>N;
a=new double *[N];
for(i=0;i<N;i++)
    a[i]=new double[N];
b=new double [N];
x=new double [N];
cout<<"Input Matrix A"<<endl;
for(i=0;i<N;i++)
    for(j=0;j<N;j++)
        cin>>a[i][j];
cout<<"Input massiv B"<<endl;
for(i=0;i<N;i++)
    cin>>b[i];
result=SLAU(a,N,b,x);
if (result==0)
{ cout<<"Massiv X"<<endl;
for(i=0;i<N;i++)
    cout<<x[i]<<"\t";
cout<<endl;
}
else if (result==-1)
    cout<<"Great number of Solution";
else if (result==-2)
    cout<<"No solution";
for(i=0;i<N;i++)

```

```

delete [] a[i];
delete [] a; delete [] b; delete [] x; }

```

7.2. Вычисление обратной матрицы методом Гаусса

ЗАДАЧА 7.2. Найти обратную матрицу к квадратной матрицы $A(N,N)$.

$$A = \begin{vmatrix} a_{00} & a_{01} & \dots & a_{0n-1} \\ a_{10} & a_{11} & \dots & a_{1n-1} \\ a_{20} & a_{21} & \dots & a_{2n-1} \\ \dots & \dots & \dots & \dots \\ a_{n-10} & a_{n-11} & \dots & a_{n-1n-1} \end{vmatrix} \quad (7.5)$$

Найти матрицу A^{-1} :

$$A^{-1} = \begin{vmatrix} y_{00} & y_{01} & \dots & y_{0n-1} \\ y_{10} & y_{11} & \dots & y_{1n-1} \\ \dots & \dots & \dots & \dots \\ y_{n-10} & y_{n-11} & \dots & y_{n-1n-1} \end{vmatrix} \quad (7.6)$$

Матрица (7.6) будет обратной к матрице (7.5), если выполняется соотношение

$$A \cdot A^{-1} = E,$$

где E – это единичная матрица, или подробнее:

$$\begin{vmatrix} a_{00} & a_{01} & \dots & a_{0n-1} \\ a_{10} & a_{11} & \dots & a_{1n-1} \\ a_{20} & a_{21} & \dots & a_{2n-1} \\ \dots & \dots & \dots & \dots \\ a_{n-10} & a_{n-11} & \dots & a_{n-1n-1} \end{vmatrix} \times \begin{vmatrix} y_{00} & y_{01} & \dots & y_{0n-1} \\ y_{10} & y_{11} & \dots & y_{1n-1} \\ y_{20} & y_{21} & \dots & y_{2n-1} \\ \dots & \dots & \dots & \dots \\ y_{n-10} & y_{n-11} & \dots & y_{n-1n-1} \end{vmatrix} = \begin{vmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 1 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{vmatrix}$$

Умножение матрицы (7.5) на нулевой столбец матрицы (7.6) даст нулевой столбец единичной матрицы:

$$\begin{aligned} a_{00}y_{00} + a_{01}y_{10} + \dots + a_{0n-1}y_{n-10} &= 1 \\ a_{10}y_{00} + a_{11}y_{10} + \dots + a_{1n-1}y_{n-10} &= 0 \\ &\dots \\ a_{n-10}y_{00} + a_{n-11}y_{10} + \dots + a_{n-1n-1}y_{n-10} &= 0 \end{aligned}$$

Система, полученная в результате умножения матрицы (7.5) на i -й столбец матрицы (7.6), будет выглядеть так:

$$\begin{aligned} a_{00}y_{0i} + a_{01}y_{1i} + \dots + a_{0n-1}y_{n-1i} &= 0 \\ a_{10}y_{0i} + a_{11}y_{1i} + \dots + a_{1n-1}y_{n-1i} &= 0 \\ &\dots \\ a_{i0}y_{0i} + a_{i1}y_{1i} + \dots + a_{i-1}y_{n-1i} &= 1 \\ &\dots \\ a_{n-10}y_{0i} + a_{n-11}y_{1i} + \dots + a_{n-1n-1}y_{n-1i} &= 0 \end{aligned}$$

А n -я система будет иметь вид:

$$a_{00}y_{0n-1} + a_{01}y_{1n-1} + \dots + a_{0n-1}y_{n-1n-1} = 0$$

$$a_{10}y_{0n-1} + a_{11}y_{1n-1} + \dots + a_{1n-1}y_{n-1n-1} = 0$$

...

$$a_{n-10}y_{0n-1} + a_{n-11}y_{1n-1} + \dots + a_{n-1n-1}y_{n-1n-1} = 1$$

Алгоритм нахождения обратной матрицы представлен в виде блок-схемы на рис. 7.5. Блоки 2–5 отражают формирование столбца единичной матрицы. Если условие 3 выполняется и элемент находится на главной диагонали, то он равен единице, все остальные элементы нулевые. В блоке 6 происходит вызов подпрограммы для решения системы уравнений методом Гаусса. В качестве параметров в эту подпрограмму передается исходная матрица A , сформированный в пунктах 2–5 вектор свободных коэффициентов B , размерность системы n . Вектор X будет решением i -ой системы уравнений и, следовательно, i -ым столбцом искомой матрицы Y .

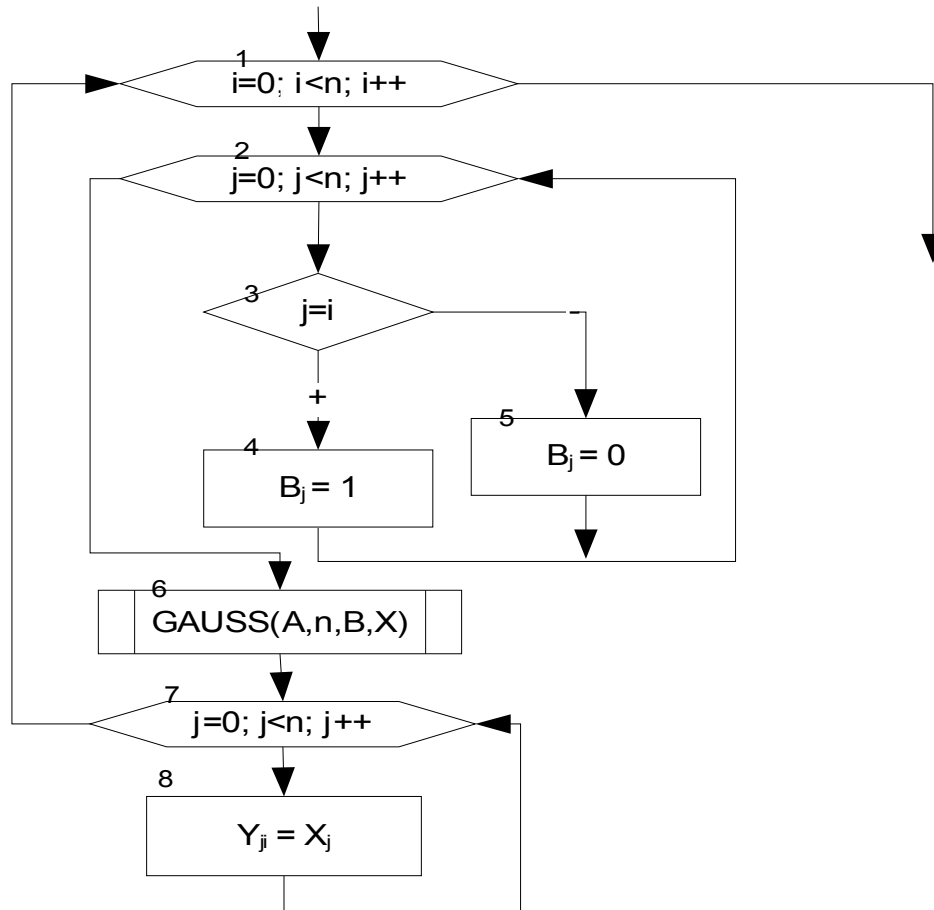


Рис 7.5. Блок-схема нахождения обратной матрицы

```

int SLAU(double **matrica_a, int n, double *massiv_b,
double *x)
{
    int i,j,k,r;
    double c,M,max,s, **a, *b;
    a=new double *[n];
    for(i=0;i<n;i++) a[i]=new double[n];
  
```

```

b=new double [n];
for(i=0;i<n;i++)
    for(j=0;j<n;j++)
        a[i][j]=matrica_a[i][j];
for(i=0;i<n;i++)
    b[i]=massiv_b[i];
for(k=0;k<n;k++)
{
    max=fabs(a[k][k]);
    r=k;
    for(i=k+1;i<n;i++)
        if (fabs(a[i][k])>max)
            { max=fabs(a[i][k]);
              r=i; }
    for(j=0;j<n;j++)
    { c=a[k][j];
      a[k][j]=a[r][j];
      a[r][j]=c; }
    c=b[k];    b[k]=b[r];    b[r]=c;
    for(i=k+1;i<n;i++)
    { for(M=a[i][k]/a[k][k],j=k;j<n;j++)
      a[i][j]-=M*a[k][j];
      b[i]-=M*b[k];
    }
}
if (a[n-1][n-1]==0)
    if(b[n-1]==0)    return -1;
    else return -2;
else
{for(i=n-1;i>=0;i--)
    {for(s=0,j=i+1;j<n;j++)
      s+=a[i][j]*x[j];
      x[i]=(b[i]-s)/a[i][i];
    }return 0; }
for(i=0;i<n;i++)
delete [] a[i];
delete [] a;
delete [] b;
}
int INVERSE(double **a, int n, double **y)
{ int i,j,res;
  double *b, *x;
  b=new double [n];
  x=new double [n];
  for(i=0;i<n;i++)
    {for(j=0;j<n;j++)

```

```

        if (j==i)    b[j]=1; else    b[j]=0;
            res=SLAU(a,n,b,x);
            if (res!=0) break;
                else for(j=0;j<n;j++)
                    y[j][i]=x[j];
    }
delete [] x;
delete [] b;
    if (res!=0)    return -1;
    else    return 0;}

int main()
{ int result,i,j,N;
double **a, **b;
cout<<"N=";    cin>>N;
a=new double *[N];
for(i=0;i<N;i++)
    a[i]=new double[N];
b=new double *[N];
for(i=0;i<N;i++)
    b[i]=new double[N];
cout<<"Input Matrix A"<<endl;
for(i=0;i<N;i++)
    for(j=0;j<N;j++)
        cin>>a[i][j];
result=INVERSE(a,N,b);
if (result==0)
{ cout<<"Inverse matrix"<<endl;
for(i=0;i<N;cout<<endl,i++)
    for(j=0;j<N;j++)
        cout<<b[i][j]<<"\t";
}
else cout<<"No Inverse matrix"<<endl;
for(i=0;i<N;i++)
    delete [] a[i];
delete [] a;
for(i=0;i<N;i++)
    delete [] b[i];
delete [] b;
}

```

7.3. Вычисление определителя методом Гаусса

ЗАДАЧА 7.3. Найти определитель квадратной матрицы $A(N,N)$.

Для верхнетреугольной матрицы определитель вычисляется как произведение элементов , лежащих на главной диагонали $det A = \sum_{i=0}^{n-1} a_{ii}$.

Поэтому матрицу вначале преобразуем к верхнетреугольному виду, а затем найдем произведение элементов на главной диагонали.

Преобразование матрицы (7.2) к виду (7.3) можно осуществить с помощью прямого хода метода Гаусса. Алгоритм вычисления определителя матрицы, изображенный в виде блок-схемы на рис. 7.6, представляет собой алгоритм прямого хода метода Гаусса, в процессе выполнения которого проводится перестановка строк матрицы.

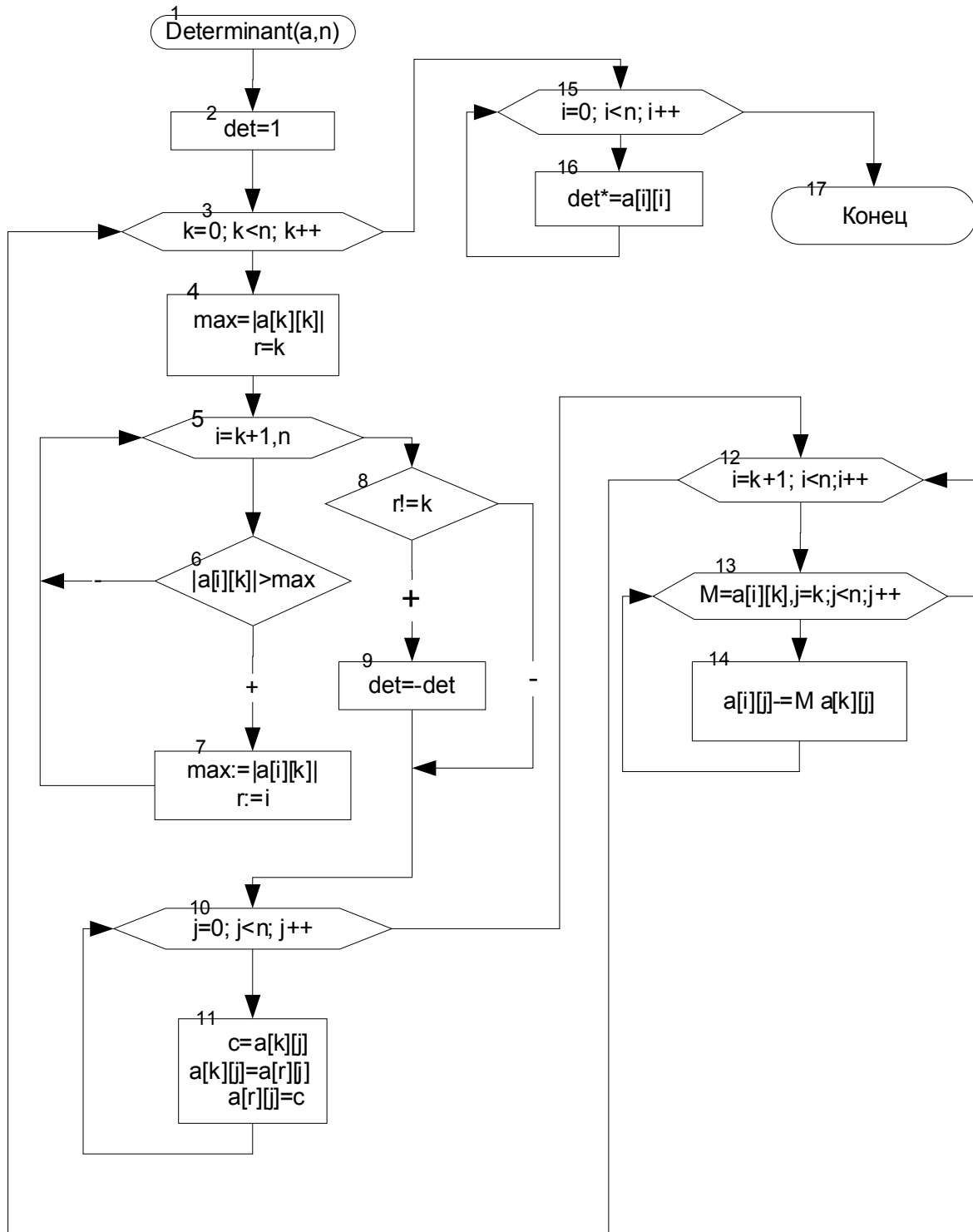


Рис 7.6. Блок-схема вычисления определителя квадратной матрицы

Если строки в матрице поменять местами, то определитель матрицы меняет знак на противоположный. В блок-схеме момент смены знака отражен в блоках 8–9. В блоке 8 определяется, будут ли строки меняться местами, и если ответ утвердительный, то в блоке 9 происходит смена знака определителя. В блоках 15–16 выполняется непосредственное вычисление определителя путем перемножения диагональных элементов преобразованной матрицы.

```
double determinant(double **matrica_a, int n)
{ int i,j,k,r; double c,M,max,s,det=1, **a;
  a=new double *[n];
  for(i=0;i<n;i++) a[i]=new double[n];
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
      a[i][j]=matrica_a[i][j];
  for(k=0;k<n;k++)
  {
    max=fabs(a[k][k]);      r=k;
    for(i=k+1;i<n;i++)
      if (fabs(a[i][k])>max)
        { max=fabs(a[i][k]);  r=i;    }
    if (r!=k) det=-det;
    for(j=0;j<n;j++)
      {c=a[k][j]; a[k][j]=a[r][j];    a[r][j]=c;    }
    for(i=k+1;i<n;i++)
      for(M=a[i][k]/a[k][k],j=k;j<n;j++)
        a[i][j]-=M*a[k][j];
  }
  for(i=0;i<n;i++)
    det*=a[i][i];
  return det;
}
int main()
{
  int result,i,j,N;
  double **a, b;
  cout<<"N=";      cin>>N;
  a=new double *[N];
  for(i=0;i<N;i++) a[i]=new double[N];
  cout<<"Input Matrix A"<<endl;
  for(i=0;i<N;i++)
    for(j=0;j<N;j++)
      cin>>a[i][j];
  cout<<"determinant="<<determinant(a,N)<<endl;
  for(i=0;i<N;i++)
    delete [] a[i];
  delete [] a;}
```