Лекция 7

Формирование и обработка двумерных массивов

7.1 Матрицы и двумерные массивы

7.1.1 Понятие матриц и двумерных массивов

При решении ряда математических задач широко используются матрицы. *Матрица* – это прямоугольная таблица, которая представляет собой совокупность *строк* и *столбцов*, на пересечении которых находятся элементы матрицы. В частности, матрицы применяются в математике для компактной записи математических моделей, например, систем линейных алгебраических или дифференциальных уравнений и др. В этом случае количество строк матрицы соответствует числу уравнений, а количество столбцов – количеству неизвестных. В результате решение систем уравнений сводится к алгебраическим операциям над матрицами.

7.1.1 Понятие матриц и двумерных массивов

Кроме того, при решении различных задач с использованием компьютера для хранения и обработки различных данных используются двумерные таблицы. Над элементами или совокупностью элементов этих таблиц часто необходимо производить поэлементные операции, например, умножение каждого элемента одной таблицы на соответствующий элемент другой.

В языках программирования высокого уровня, в том числе и в C++ для представления матриц и двумерных таблиц используются *двумерные массивы*.

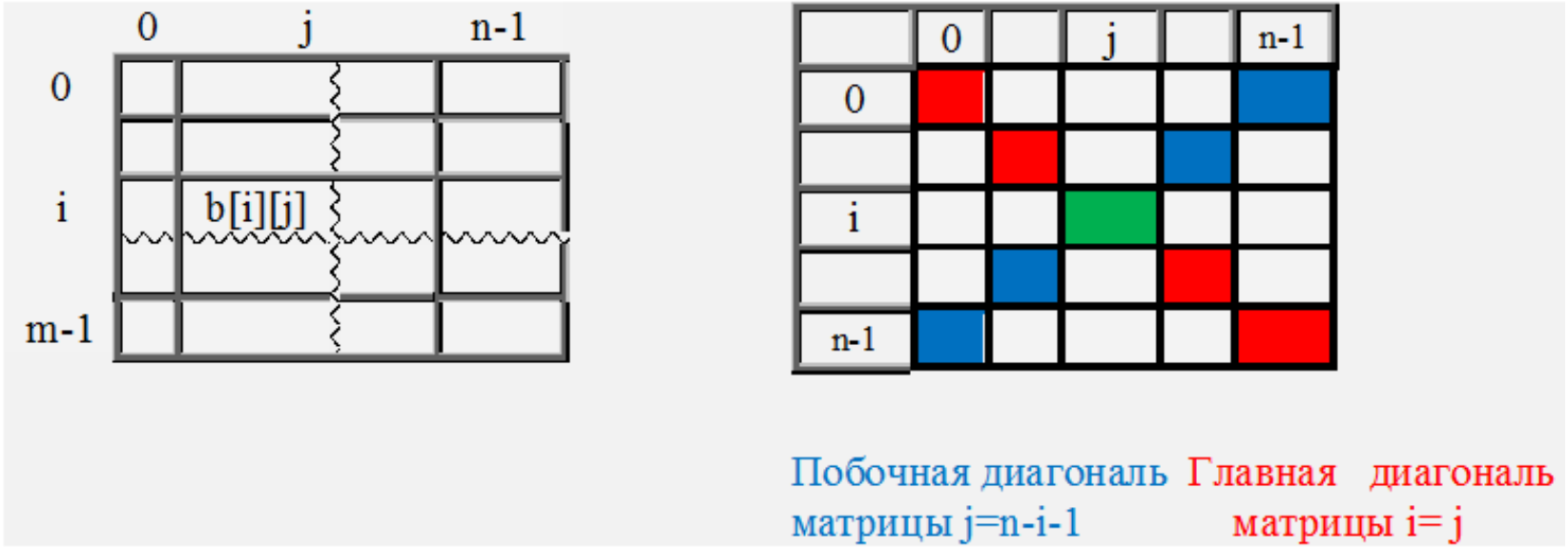
Двумерные массивы, в отличие от одномерных, имеют два измерения и стандартный доступ к их элементам осуществляется с помощью двух индексов: номера строки и номера столбца. В языке C++ начальное значением обоих индексов – 0.

7.1.1 Понятие матриц и двумерных массивов

Например, матрица 4×3 может быть представлена двумерным массивом из четырех строк с номерами от 0 до 3 и трех столбцов с номерами от 0 до 2, как на рис. Причем матрицы, как и соответствующие двумерные массивы могут быть как прямоугольные, так и квадратные.

b[0][0]	b[0][1]	b[0][2]
b[1][0]	b[1][1]	b[1][2]
b[2][0]	b[2][1]	b[2][2]
b[3][0]	b[3][1]	b[3][2]

Матрица прямоугольная **b[m][n]** - m≠n ***Матрица квадратная*** **A[m][n]** - n=m



7.1.1 Понятие матриц и двумерных массивов

Количество индексов (измерений) указывает на ***размерность*** массива. В приведенном примере размерность массива **b** равно 2**,** т.е. массив двумерный (имеет два измерения), а ***количество элементов*** массива **b** равно **12:** количество элементов по первому измерению – 4 (4 строки), количество измерений по второму измерению – 3 (3 столбца).

Однако C++ допускает использование массивов с размерностью больше 2.

7.1.2 Объявление и инициализация автоматических двумерных массивов

Точно так же, как и для одномерных массивов, при объявлении (описании) двумерного *автоматического* массива (расположенного в автоматической памяти, а не в куче) требуется указать размер каждого измерения массива (число строк и столбцов), причем эти **размеры** при объявлении массива также **должны быть константами**, например,

```
const int M=4,N=3;      // именованные константы
int b[M][N];            // объявление целого двумерного массива b с рис.
float matr[5][10];      // объявление вещественного двумерного массива
                        // с именем  matr, в котором 5 строк и
                        // в каждой строке по 10 вещественных чисел
```

Итак, двумерные массивы – это структура данных (организации данных) в языках программирования, представляющая (моделирующая) матрицы и прямоугольные таблицы.

7.1.2 Объявление и инициализация автоматических двумерных массивов

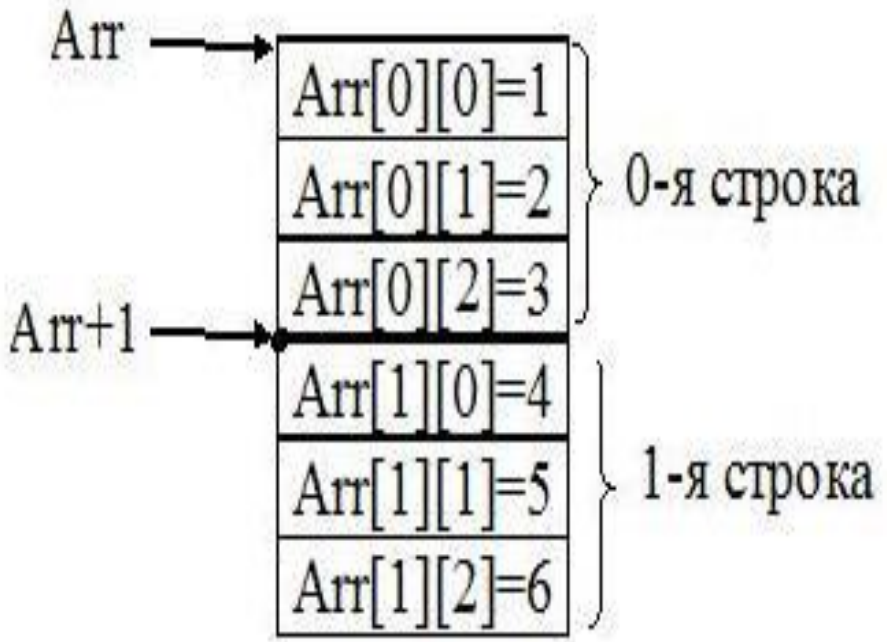
Двумерный массив, как и любая переменная должен иметь имя. Имя двумерного массива в C++ – это константный указатель на строку с нулевым индексом.

Вместе с объявлением двумерного массива также возможна и его инициализация списком значений, аналогично одномерному массиву. При инициализации двумерный массив представляется как одномерный массив со списком всех его значений, заключенных в фигурных скобках. При этом каждый элемент в фигурных скобках либо является строкой двумерного массива и заключается в свои фигурные скобки, (в этом случае левую размерность, т.е. число строк при описании можно не указывать), либо задается общий список элементов в том порядке, в котором элементы располагаются в памяти. Рассмотрим пример трех эквивалентных объявлений двумерного массива с инициализацией:

7.1.2 Объявление и инициализация автоматических двумерных массивов

```
int Arr[2][3] = {{1,2,3},{4,5,6}}; //массив из 2 строк и 3 столбц
int Arr[ ][3] = {{1,2,3},{4,5,6}}; // То же, кол-во строк
                                   //определяет компилятор
int Arr[2][3] = {1,2,3,4,5,6};     //То же, задан общим списком
```

Расположение в памяти



Интерпретация программиста

3 столбца

$Arr_{0,0}$=1	$Arr_{0,1}$=2	$Arr_{0,2}$=3
$Arr_{1,0}$=4	$Arr_{1,1}$=5	$Arr_{1,2}$=6

2 строки

7.2 Доступ к элементам двумерных массивов

Доступ к определенному элементу двумерного массива имеет следующий формат:

имя_массива`[`*выражение1*`][`*выражение2*`]`

где: *выражение1* (номер строки) и *выражение2* (номер столбца) – любые допустимые целочисленные выражения, в частности, константы или переменные. Например,

`b[0][1]` `Arr[i][j]` `a[0][k+1]`

Напомним, что первый (левый) индекс – это номер строки, а второй (правый) индекс – это номер столбца.

Так как имя двумерного массива компилятор интерпретирует, как константный указатель на начало массива, то обратиться к элементу массива `Arr[i][j]` можно также, используя имя массива как указатель, например,

`*(Arr[i]+j)` или `*(*(Arr+i)+j)` или `*(Arr+i)[j]`,

поскольку `Arr[i]` является адресом начала `i`-ой строки массива.

7.2 Доступ к элементам двумерных массивов

Низкоуровневый код, который сгенерирует компилятор, во всех трех показанных способах доступа к элементу будет одним и тем же. Для вычисления адреса произвольного `[i][j]`-го элемента двумерного массива компилятору нужно знать адрес начала массива, количество элементов в строке и размер элемента. Например:

```
const int M = 5, N = 6;   // именованные константы
float matr[M][N];         // описание двумерного массива
      // адрес элемента matr[i][j]будет вычисляться как:

//адрес_начала_массива+i*N*sizeof(float)+j*sizeof(float)
```

Обратите внимание, что в вычислении адреса участвует только младшая размерность массива (т.е. число столбцов), а старшая размерность (число строк) нигде не фигурирует! Важен только номер строки – `i`.

Замечание. Исходя из арифметики указателей, компилятор при вычислении выражения `(Arr+1)` получит адрес следующей строки массива, как показано ранее на рис.

7.3 Формирование и обработка двумерных массивов

Формирование и обработка двумерных массивов, как правило, производятся с использованием вложенных циклов, которые могут обеспечить перебор всех элементов массива или его подмножество. В некоторых задачах может иметь значение *порядок перебора* элементов массива: *по строкам* или *по столбцам*. Если внешний цикл будет организован по первому индексу, а внутренний цикл по второму индексу, то выполняется построчный перебор элементов двумерного массива. Если внешний цикл в качестве параметра использует второй индекс, а внутренний цикл – первый индекс, то элементы массива перебираются по столбцам.

```
//копирование элементов массива по строкам:
    for (int i=0;i<2;i++)
        for(int j=0;j<3;j++)
            Z[i][j]=Arr[i][j];

//копирование элементов массива по столбцам:
    for(int j=0;j<3;j++)
        for (int i=0;i<2;i++)
            Z[i][j]=Arr[i][j];
```

7.4 Использование двумерных массивов в качестве параметров

При использовании двумерных массивов в качестве формальных параметров процедуры требуется указать тип его элементов и, в квадратных скобках после его имени, количество элементов в каждом измерении. Например, для решения задачи копирования массива, представленного на рис. 3, можно было бы создать функцию с заголовком:

```
// объявление функции с параметрами - матрицами
void copy(int Arr[2][3], int Z[2][3])

// или так (без указания первого размера):
void copy(int Arr[ ][3], int Z[ ][3])
```

7.4 Использование двумерных массивов в качестве параметров

Функции неважно знать, сколько всего строк в матрице, ей важно знать, насколько велики эти строки. Тогда компилятор сможет вычислить, где находится каждый элемент.

Как мы уже рассмотрели, при вычислении адреса элемента двумерного массива значение левой размерности неважно.

Вспомните, что для параметров – одномерных массивов мы не указывали размер массива. Но всегда следует передавать размер каждого измерения массива через отдельные параметры.

Для того, чтобы вызвать функцию с фактическим параметром – двумерным массивом, такой *массив должен быть объявлен в вызывающей функции с точно таким же количеством строк и столбцов, как и формальный параметр* (иначе компилятор покажет ошибку несовпадения типов).

7.4 Использование двумерных массивов в качестве параметров

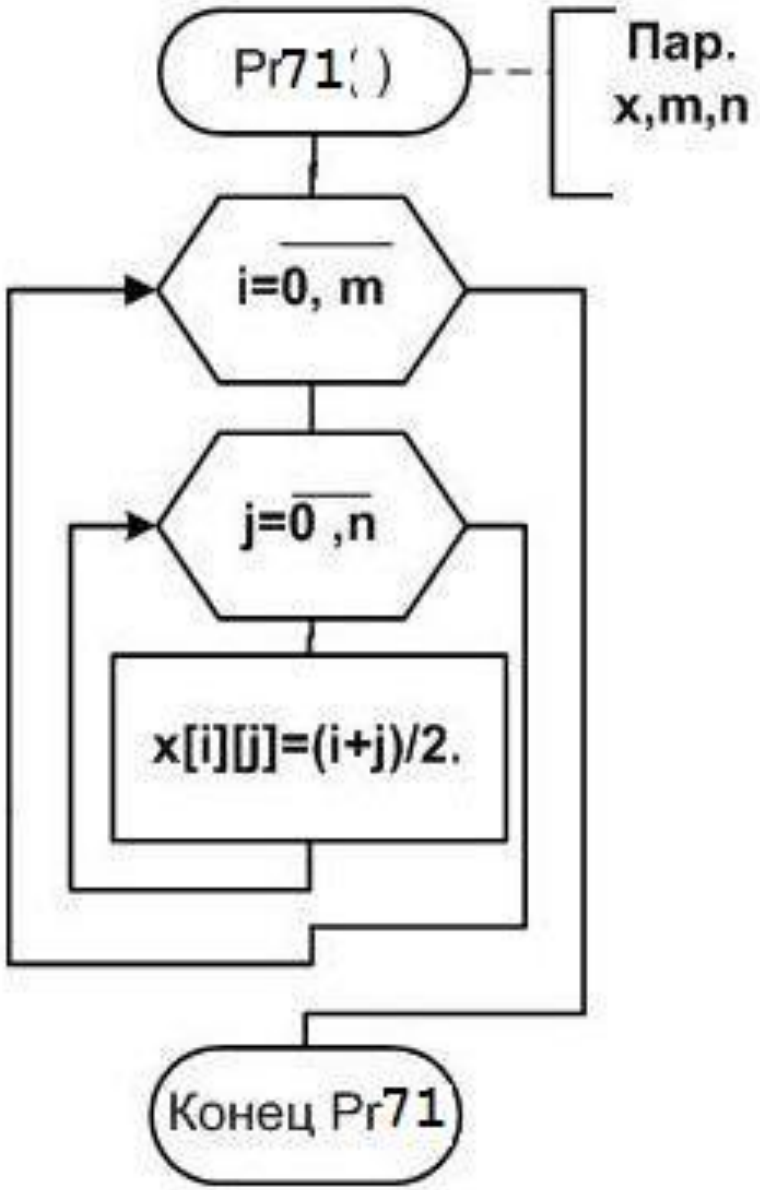
При объявлении двумерного массива можно выделить ему память с запасом. *При вызове функции* можно указать фактически используемые размеры каждого измерения через входные параметры, но *сам массив (как фактический параметр) обязан передаваться с тем же количеством столбцов, под которое ему выделялась память при объявлении* (количество строк не имеет значения, если первый размер не был указан в формальном параметре – массиве).

Формирование двумерных массивов может производиться теми же способами, что и в случае одномерных массивов: путем инициализации по заданному правилу, путем инициализации случайными числами.

Пример 7.1. Разработать процедуру, которая формирует двумерный вещественный массив **x,** содержащий **m** строк и **n** столбцов по правилу: **x[i][j] = (i+j)/2.**

Схема алгоритма и программный код процедуры приведены на далее. Функция имеет три параметра: входным и одновременно выходным параметром является массив **x,** которому была выделена память m×n. Также функция имеет еще два входных параметра, указывающие нужное число строк и число столбцов массива. Для формирования массива организуются вложенные циклы: внешний – по строкам и внутренний – по столбцам массива. В теле внутреннего цикла вычисляются значения элементов массива по заданному правилу.

7.4 Использование двумерных массивов в качестве параметров




```
// формирование двумерного массива по
//  заданному правилу
void Pr71(float x[][3], int m,int n)
{
    for(int i=0;i<m;i++)
        for(int j=0;j<n;j++)
            x[i][j]=(i+j)/2.0;
}

//фрагмент кода вызывающей функции
...
const int m=5,n=3;
float mas[m][n];
Pr71(mas,m,n); //корректный вызов
Pr71(mas,5,3); //корректный вызов
Pr71(mas,3,3); //корректный вызов
Pr71(mas,5,2); //Ошибка! Несовпадение типов
//по второму измерению(кол-ву столбцов)
```

Обратите внимание на примеры вызова функции **Pr71**, которые также приведены на слайде. Третий вызов этой функции **Pr71(mas,3,3);** корректен, хотя второй параметр, показывающий число строк матрицы не совпадает с числом строк матрицы при ее описании **(3 ≠ 5)**, но, как мы уже упоминали, это не имеет значения. Однако следует оговориться, что если бы мы захотели вызвать функцию следующим образом: **Pr71(mas,6,3);** то так как второй параметр превышает максимальное число строк, под которое массиву выделена память **(6>5)**, то возникнет ошибка времени выполнения, но компилятор эту ошибку не укажет!

Важное замечание. Так как для двумерного массива, являющегося формальным параметром, в заголовке функции надо обязательно указать размер только второго измерения, то возникает соблазн вместо конкретного значения использовать идентификатор константы или целой переменной. Т.е. написать заголовок следующим образом:

```
// вместо правильного заголовка функции
void Pr71(float x[][3],int m, int n)

void Pr71(float x[][n],int m, int n) // написать так нельзя
// n в квадратных скобках будет счиаться необъявленным
// идентификатором, и даже если описать n как глобальную
// константу, то будет другая ошибка – в квадратных скобках
// можно записать только константное выражение,
// т.е. конкретное число.
```

Пример 7.2. Разработать процедуру, которая формирует двумерный вещественный массив из случайных *вещественных* чисел заданного диапазона [a, b].

Мы уже рассматривали формирование одномерного массива из случайных вещественных чисел (лекция 4, пример 4.1). Программный код процедуры формирования двумерного массива отличается от упомянутого примера тем, что первым формальным параметром является двумерный массив, для которого необходимо указать размер второго измерения, и добавляется еще один входной параметр, показывающий фактический размер этого дополнительного, по сравнению с одномерным массивом, измерения. Кроме того, конечно, добавлен еще один внутренний регулярный цикл для перебора индексов столбцов массива.

На рис. приведен пример формирования двумерного массива, который в вызывающей процедуре должен быть описан так же, как и в предыдущем примере 7.1, т.е. содержащий три столбца.

```
// Формирование двумерного нединамического
// массива из случайных вещественных чисел
void input(float matr[][3],int m,int n, int a,int b)
{
    Random^ rnd = gcnew Random; // Создание объекта типа Random.
     for (int i = 0; i < m; i++)
        for (int j=0;j<n;j++)
         // Генерация случ. веществ. значений в диапазоне от a до b.
            matr[i][j] = a+(b-a) *rnd ->NextDouble();
}
```

Пример 7.3. Разработать процедуру вывода двумерного вещественного массива.

Отображение двумерных массивов на форме программы удобнее всего, как и одномерных, производить с помощью объектов типа **ListBox**. При относительно небольших размерах массива можно организовать построчный вывод массива: каждая строка выводится в отдельный элемент списка.

Рассмотрим пример процедуры построчного вывода вещественного двумерного массива в заданный список **ListBox**. Здесь, так же, как ранее, для вывода в **ListBox** создается объект типа **String^**; перед каждым входом во внутренний цикл этот объект типа **String^** очищается для записи в него элементов одной строки массива. Во внутреннем цикле по столбцам каждый элемент массива, отформатированный с двумя знаками после запятой, добавляется в объект типа **String^.** По окончании внутреннего цикла полученный объект типа **String^,** в котором теперь записаны отформатированные элементы одной строки массива, выводится в **ListBox.**

```
// функция выводит матрицу размера m x n в ListBox
void output(float mas[][3], int m, int n, ListBox^ Lb)
{
    Lb->Items->Clear(); // очистка списка
    for (int i = 0; i < m; i++)
    {
        String^ s="";// объект-строка для записи в ListBox
        for (int j=0;j<n;j++)
            s=s+String:: Format("{0,8:F2}",mas[i][j]); // добавление
                         //к объекту String^ отформатированного
                         //с двумя знаками после запятой элемента
                         // i-ой строки и j-го столбца
        Lb->Items->Add(s);
    }
}
```

Важное замечание. При создании объекта **ListBox** следует в конструкторе изменить значения его свойств **HorizontalScrollbar** и **ScrollAlwaysVisible** на **True** для того, чтобы при необходимости можно было бы с помощью полос прокрутки увидеть все элементы двумерного массива. Также полезно установить шрифт в списке **ListBox** (свойство **Font**), как **Courier New**

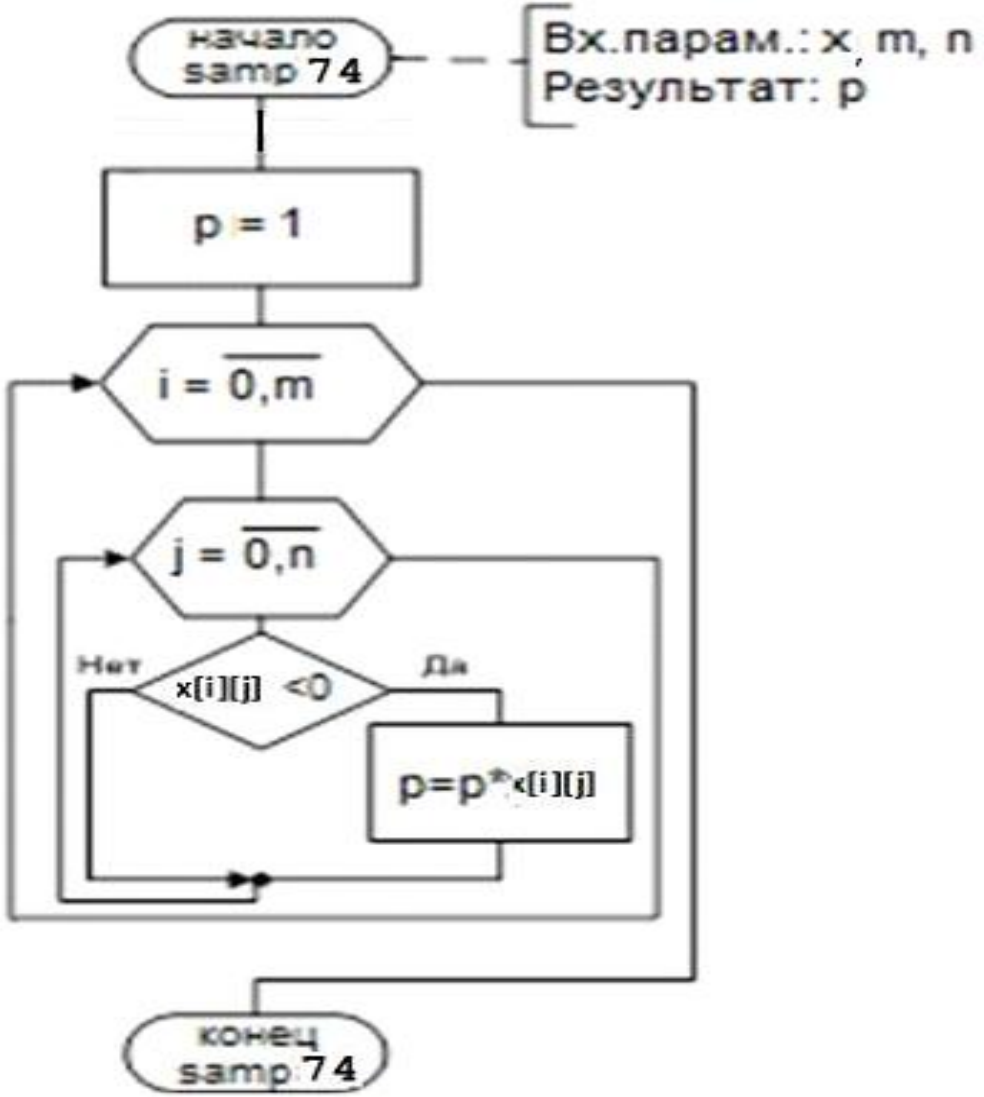
7.5 Типовые алгоритмы и примеры их реализации

7.5.1 Типовые алгоритмы

- формирование массива в соответствии с заданными правилами;
- формирование одномерного массива из двумерного в соответствии с заданными правилами;
- формирование нового двумерного массива из исходного двумерного массива в соответствии с заданными правилами;
- определение количества элементов массива при заданном условии;
- определение суммы (произведения) значений элементов массива при заданном условии;
- поиск наибольшего (наименьшего) элемента массива или его части и его индексов;
- перестановка элементов массива или его части по заданным правилам;
- перестановка заданных строк (столбцов) массива;
- транспонирование матрицы (замена строк столбцами и наоборот);
- алгебраическое умножение двух матриц (произведение двух матриц **A** и **B)**

7.5.2 Примеры реализации типовых алгоритмов формирования и обработки автоматических двумерных массивов

Пример 7.4. Разработать процедуру, которая вычисляет произведение отрицательных элементов заданного вещественного двумерного массива **x**



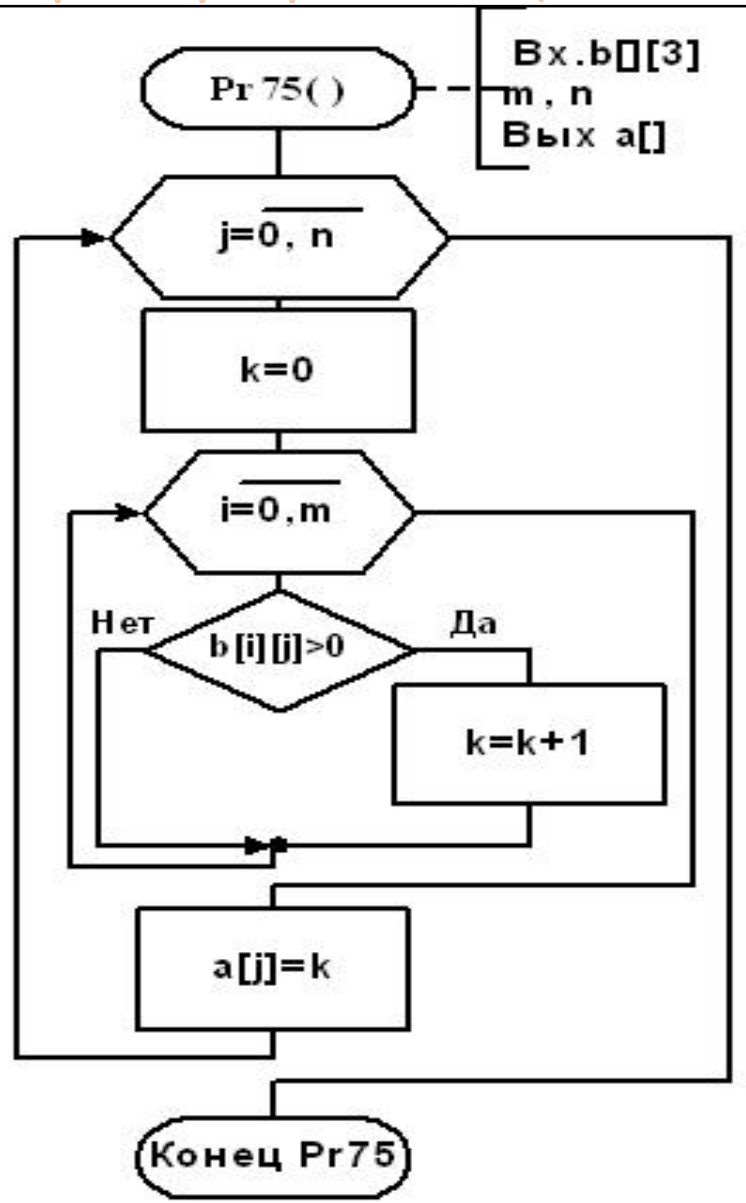

```
// функция вычисляет произведение
// отрицательных элементов матрицы
float samp74(float x[][3],int m,int n)
{
    float P=1;//искомое произведение
    for (int i = 0; i < m; i++)
        for (int j=0;j<n;j++)
            if(x[i][j]<0) P*=x[i][j];
    return P;
}
```

7.5.2 Примеры реализации типовых алгоритмов формирования и обработки автоматических двумерных массивов

Пример 7.5. Разработать процедуру, которая формирует целый одномерный массив **a**, каждый элемент которого представляет собой количество положительных элементов соответствующего столбца, заданного вещественного двумерного массива **b.**

Входными параметрами процедуры являются двумерный массив **b**, число его строк **m** и число столбцов **n,** выходным – одномерный массив **a**. Внешний цикл организован по столбцам массива **b**. В теле этого цикла сначала обнуляется счетчик **k** положительных элементов для очередного столбца. Затем во внутреннем цикле по строкам массива подсчитывается количество **k** положительных элементов в очередном столбце. По окончании внутреннего цикла подсчитанное количество записывается в очередной элемент массива **a**.

7.5.2 Примеры реализации типовых алгоритмов




```
// Функция вычисляет кол-во положит. элементов
// в каждом столбце матрицы b и записывает его
// в одномерный массив a
void Pr75(float b[][3],int m,int n,int a[])
{
    for (int j=0;j<n;j++)//внеш. цикл по столбцам
    {
        int k=0;//счетчик положительных элементов
                // в j-ом столбце
        for(int i=0;i<m;i++)
            if(b[i][j]>0) k++;
        a[j]=k;//запись счетчика в одномер.массив
    }
}

//фрагмент событийной процедуры:
...
const int m=10,n=3;
float b[m][n];//выделение памяти под m=10 строк
              // и n=3 столбца
input(b,5,3,-20,20); //заполнение только 5 строк
                     //и 3 столбцов
output(b,5,3,listBox3);//вывод 5 строк
                       //и 3 столбцов матрицы
int a[n];   // в массиве a число эл-тов равно
            //числу столбцов матрицы,
            //но памятьвыделяется константой n
Pr75(b,5,3,a);//вызов ф-ции формиров. массива a
output(a,3,listBox1); //вывод одномер. массива
...
```

7.5.2 Примеры реализации типовых алгоритмов формирования и обработки автоматических двумерных массивов

- Напомним, память под любой нединамический массив выделяется до выполнения проекта. В данном примере для двумерного массива **b** отведена память под 10 строк и 3 столбца. Причем можно было бы изменить значение константы **m=10** на другое число, но изменить **n=3** нельзя, т.к. в формальном параметре функции **Pr75** для младшего измерения (во вторых квадратных скобках) массива **b** указано именно число 3. Но вполне можно заполнить массив меньшим (но не большим!) числом строк и столбцов, чем отведенная для массива память. Это показывает третий оператор событийной процедуры, который вызывает рассмотренную уже нами в примере 7.2 функцию заполнения массива случайными вещественными числами. В данном случае случайными числами из отрезка **[-20,20]** заполняются только 5 строк и 3 столбца матрицы. В остальной незаполненной памяти матрицы остается «мусор».

7.5.2 Примеры реализации типовых алгоритмов формирования и обработки автоматических двумерных массивов

Далее вызывается функция примера 7.3 для вывода полученной матрицы в **ListBox** на форме, фактическими параметрами которой также обеспечивается вывод только 5-ти строк и 3-х столбцов.

Следующий оператор объявляет одномерный массив **a**, который требуется сформировать по условию задачи. Размер массива **a** совпадает с количеством столбцов исходного двумерного массива **b**, поэтому этот размер определяется той же константой **n=3**.

Далее вызывается сама функция формирования одномерного массива с фактическим числом столбцов 3. В последнем операторе вызывается функция вывода одномерного массива из лекции 4 (пример 4.4, рис.9).

7.5.2 Примеры реализации типовых алгоритмов формирования и обработки автоматических двумерных массивов

Как видно из рассмотренных примеров, процедура, в которой формальным параметром является многомерный автоматический (нединамический) массив, оказывается не очень-то универсальной, так как все размеры каждого измерения массива (кроме самого старшего) должны обязательно совпадать с соответствующими размерами в фактических параметрах. Т.е. массив в вызывающей функции должен быть объявлен с точно таким же количеством значений в каждом измерении, как записано в заголовке вызываемой функции. И, если вдруг появится необходимость увеличить размер массива, для которого уже написана функция, то требуется менять ее заголовок. Гораздо больше возможностей предоставляют процедуры, создаваемые для динамических массивов, к рассмотрению которых мы и перейдем.

7.6 Динамические двумерные массивы

В тех задачах, где размерности массива могут меняться во время выполнения программы, и для универсальности функций, работающих с многомерными массивами, можно использовать динамическое выделение памяти под массивы.

Имя двумерного массива компилятор интерпретирует, как константный указатель на нулевую строку. Поэтому для выделения памяти под двумерный массив нужно сначала выделить память под одномерный массив указателей, а затем каждому элементу этого массива присвоить адрес памяти, выделенной под строку двумерного массива.

Пример 7.6. Создание и освобождение динамического двумерного массива.

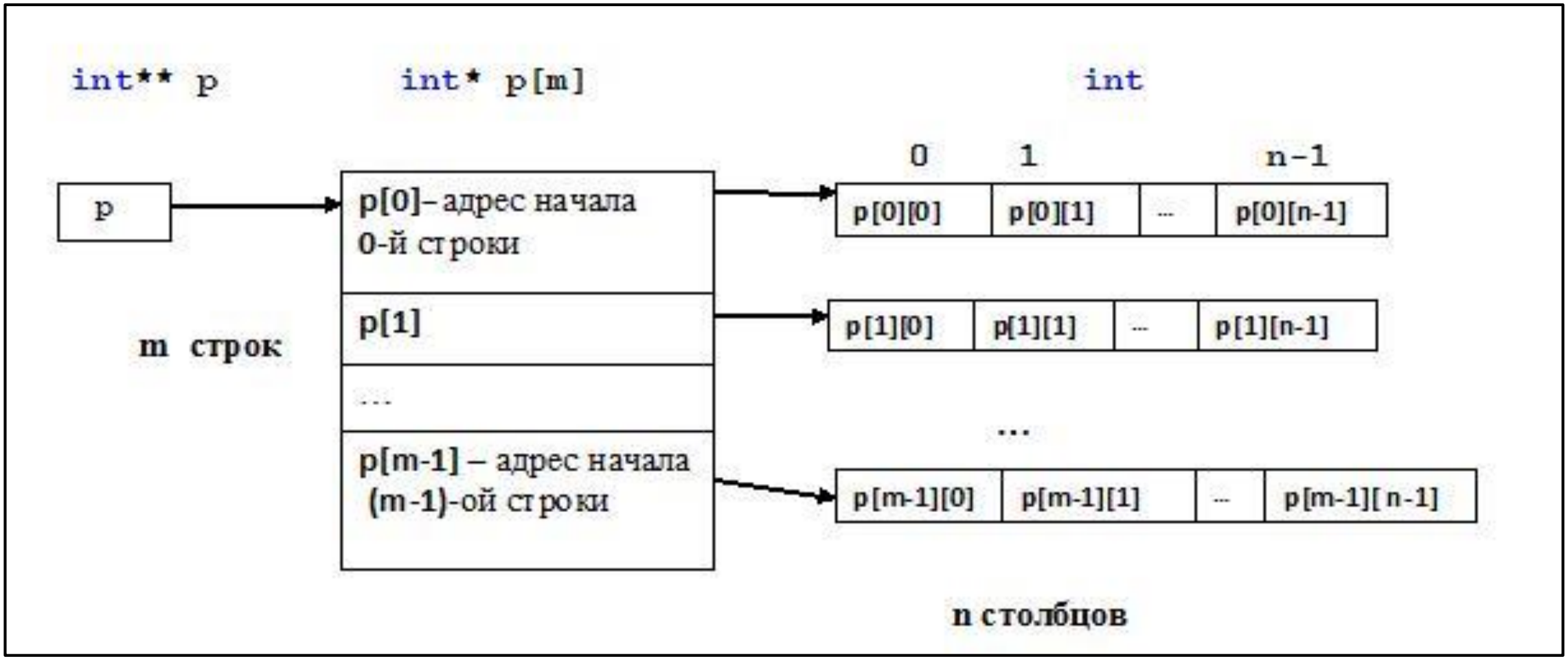
```
// создание динамического двумерного массива
int m,n;                    // число строк и столбцов матрицы
...
// вычисляются значения  m и n
...
int** p=new int*[m];//выделение памяти под массив
                    //из m указателей на строки матрицы
for (int i=0;i<m;i++)
    p[i]=new int[n];//динамически выделяется память под
                    //очередную строку из n целых элементов,
                    //а возвращаемый указатель запоминается
                    //в соответсвующем элементе массива
                    //указателей на строки
//Теперь можно заполнить выделенную под матрицу память:
for (int i = 0; i < m; i++)
    for (int j=0;j<n;j++)
        p[i][j]=i+j; //например, формируем значения элементов
...
//продолжаем работать с массивом

...
```

Здесь сначала объявляются и вычисляются переменные, в которых будут храниться необходимые числа строк (**m**) и столбцов (**n)** массива. Далее, после того, как число строк и столбцов определено, создается переменная **p** типа «указатель на указатель», выделяется память под массив указателей на строки матрицы (количество строк – **m**), и адрес начала этого массива указателей присваивается созданной переменной **p**.

Затем организуется цикл по числу строк матрицы для выделения памяти под каждую строку. В цикле каждому элементу массива указателей на строки присваивается адрес начала участка памяти, выделенного под строку двумерного массива (память выделяется под **n** целых значений – под **n** столбцов матрицы). Каждая строка состоит из **n** элементов типа **int**.

При таком подходе в памяти создается довольно сложная структура, с присущими динамическому выделению памяти, накладными расходами, зато минимизируются действия при изменении размерностей массива во время выполнения

При таком способе создания динамического двумерного массива, строки этого массива расположены в разных участках динамической памяти (кучи), а не занимают непрерывную линейную область памяти, как автоматические массивы.



Доступ к элементу такого массива на уровне языка C++ синтаксически выглядит так же, как и в случае автоматического массива, но на низком уровне компилятор, исходя из типа указателя, вычисляет адрес принципиально по-другому. Когда выполняется оператор вида

```
int x=p[i][j];
```

компилятор сначала извлекает адрес **i**-ой строки из массива указателей, а потом к этой базе добавляет смещение **j**-го элемента относительно начала **i**-ой строки. Таким образом, для доступа к элементу массива можно также использовать эквивалентную **p[i][j]** запись ***(*(p+i)+j)**, но адрес элемента **p[i][j]**будет вычисляться как:

```
p+i*sizeof(int*)+j*sizeof(int)
```

Рассмотрим, как при такой организации массива можно изменить размер массива, добавив к нему одну строку, а затем освободить захваченную динамическую память:

```
// Действия программиста при увеличении
   // количества строк в двумерном массиве
   ...
   m+=1;// увеличили число строк
   int** tmp=new int*[m]; //выделили новый блок памяти
                          //только под массив указателей на строки
                          //(сами строки трогать не будем!)
   for (int i = 0; i < (m-1); i++)
       tmp[i]=p[i];       //просто переписали в новый массив
                          //адреса строк
   delete[] p;            //теперь старый массив указателей
                          //нам не нужен - освобождаем память
   p=tmp;                 //перенаправили указатель p на новый массив
   p[m-1]=new int[n];  //выделили память для добавляемой строки
   //продолжаем пользоваться указателем p
```

```
...
// в конце работы необходимо освободить
// ВСЮ захваченную динамическую память.
// ОСВОБОЖДЕНИЕ динамической памяти матрицы
for (int i = 0; i < m; i++)
    delete[] p[i];   //в цикле освобождаем память,
                     //занятую i-ой строкой
delete[] p;          //освобождаем память,занятую
                     // массивом указателей на строки
p=0;                 //теперь этим указателем пользоваться
                    //нельзя, поэтому для безопасности обнуляем
...
```

Обратите внимание, что для освобождения памяти, выделенной под матрицу, недостаточно одного оператора **`delete[] p;`** так как такой оператор удалит массив указателей, но не строки массива, на которые они указывают. Для полного освобождения памяти сначала нужно в цикле удалить сами строки, а уже потом массив указателей.

7.7 Типовые процедуры формирования и вывода динамических двумерных массивов

При использовании двумерных динамических массивов в качестве формального параметра процедуры можно просто использовать указатель на начало массива.

Пример 7.7. Разработать процедуру, которая формирует динамический двумерный вещественный массив из случайных *вещественных* чисел заданного диапазона `[a,b]`.

Программный код функции формирования матрицы из случайных чисел и фрагмент событийной процедуры, которая вызывает разработанную функцию, представлен на рис. Функция имеет 4 входных целочисленных параметра: число строк `m`, число столбцов `n`, и `a`, `b` – границы диапазона случайных чисел и возвращаемое значение типа указатель на указатель на `float`.

7.7 Типовые процедуры формирования и вывода двумерных массивов

```
// формирование динамической матрицы из случайных веществ. чисел
float** input(int m,int n, int a,int b)
{
    float** matr=new float* [m]; //выделение памяти под m указателей
                                 //для строк матрицы
    Random^ rnd = gcnew Random;
     for (int i = 0; i < m; i++)   // Цикл заполнения массива.
    {
        *(matr+i)=new float [n];//выделение памяти под столбцы
                                //матрицы в каждой строке
        for (int j=0;j<n;j++)
            *(*(matr+i)+j) = a+(b-a) *rnd ->NextDouble();
    }
    return matr;// возвращается указатель на созданную матрицу
}
      //фрагмент событийной процедуры
      ...
      int m,n;  //число строк и столбцов исходной матрицы
      int a,b;  //диапазон случайных чисел
      //определение значений m, n, a, b любым способом
      ...
      float** matr=input(m,n,a,b); //вызов функции ввода для создания
                                   //динамич. матрицы из случайных чисел [a,b]
```

7.7 Типовые процедуры формирования и вывода двумерных массивов

Сначала в теле функции объявляется переменная **matr** типа «указатель на указатель на **float**» и выделяется память под массив указателей на **float**. Количество элементов в массиве указателей равно **m** – количеству строк в создаваемой матрице, и в них будут храниться адреса строк матрицы.

После создания объекта типа **Random** для формирования случайных чисел, организуется вложенный цикл. Во внешнем цикле динамически выделяется память для каждой строки матрицы (каждая строка состоит из **n** элементов типа **float**) – выделяется память под **n** вещественных чисел типа **float**, т.е. под столбцы в очередной строке. Адрес начала участка выделенной памяти запоминается в соответствующем элементе массива указателей.

Далее во внутреннем цикле из случайных чисел формируются значения **n** элементов текущей **i**-ой строки. После завершения внутреннего цикла происходит изменение значения параметра внешнего цикла **i** и переход к новой строке – все описанные действия повторяются для очередной строки.

По окончании внешнего цикла функция оператором **return** возвращает указатель на сформированную матрицу.

Пример 7.8. Разработать процедуру вывода динамического двумерного вещественного массива.

Программный код процедуры, организующей вывод динамического массива в список **ListBox** на форме отличается от процедуры вывода автоматического массива примера 7.3 формальным параметром – вместо массива с квадратными скобками, показывающими количество элементов в измерениях массива, в процедуру передается имя массива как указатель на указатель. Также, для доступа к элементам массива используется не индексирование (квадратные скобки с индексами элемента), а операция разыменования, как более эффективная, хотя, как мы уже говорили, оба способа доступа приводят к одинаковым результатам.

Пример 7.8. Разработать процедуру вывода динамического двумерного вещественного массива.

Программный код процедуры, организующей вывод динамического массива в список **ListBox** на форме, показан на рис., и отличается от процедуры вывода автоматического массива примера 7.3 формальным параметром – вместо массива с квадратными скобками, показывающими количество элементов в измерениях массива, в процедуру передается имя массива как указатель на указатель. Также, для доступа к элементам массива используется не индексирование (квадратные скобки с индексами элемента), а операция разыменования, как более эффективная, хотя, как мы уже говорили, оба способа доступа приводят к одинаковым результатам.

7.7 Типовые процедуры формирования и вывода двумерных массивов

```
// функция выводит динамическую матрицу размера m x n в ListBox
void output(float ** mas, int m, int n, ListBox^ Lb)
{
    Lb->Items->Clear();// очистка списка
    for (int i = 0; i < m; i++)
    {
        String^ s="";// строка для записи в ListBox
        for (int j=0;j<n;j++)
            s=s+String:: Format("{0,8:F2}",*(*(mas+i)+j));
        Lb->Items->Add(s);
    }
}
```

Пример вызова этой функции в событийной процедуре (продолжение фрагмента предыдущего примера на рис. 12):

```
output(matr,m,n,listBox1);
```

При вызове этой функции в качестве фактического параметра передается имя массива, которое, как мы уже многократно повторяли, является константным указателем на сам массив.

Пример 7.9. Решить задачу примера 7.5, используя динамические массивы. Требуется сформировать целый одномерный массив **a**, каждый элемент которого представляет собой количество положительных элементов соответствующего столбца заданного вещественного двумерного массива **b**, учитывая, что и исходный двумерный массив **b** и создаваемый одномерный массив **a** имеют динамическое распределение памяти.

Программный код процедуры отличается от примера 7.5 тем, что память под одномерный массив выделяется динамически в теле процедуры (по количеству столбцов матрицы **b**), и, соответственно, указатель на сформированный массив возвращается оператором **return**. Матрица **b,** являясь динамической, передается в процедуру как указатель.

7.7 Типовые процедуры формирования и вывода двумерных массивов

Пример 7.9. Решить задачу примера 7.5, используя динамические массивы. Требуется сформировать целый одномерный массив **a**, каждый элемент которого представляет собой количество положительных элементов соответствующего столбца заданного вещественного двумерного массива **b**, учитывая, что и исходный двумерный массив **b** и создаваемый одномерный массив **a** имеют динамическое распределение памяти.

Программный код процедуры представлен на рис. Он отличается от примера 7.5 тем, что память под одномерный массив выделяется динамически в теле процедуры (по количеству столбцов матрицы **b**), и, соответственно, указатель на сформированный массив возвращается оператором **return**. Матрица **b,** являясь динамической, передается в процедуру как указатель.

7.7 Типовые процедуры формирования и вывода двумерных массивов

```
// функция вычисляет кол-во положит. элементов
// в каждом столбце динамической матрицы b и записывает
// его в одномерный динамический массив a
int* Pr79_1(float** b,int m,int n)
{
    int* a=new int[n];      //выделение памяти под массив a
    for (int j=0;j<n;j++)   //внешний цикл по столбцам
    {
        int k=0;            //счетчик полож. эл-тов в j-ом столбце
        for(int i=0;i<m;i++)
            if(*(*(b+i)+j)>0) k++;
        *(a+j)=k;           //запись счетчика в одномерный массив
    }
    return a;               //возврат указателя на сформированный массив
}
```

Пример 7.10. Сформировать целый одномерный массив `a`, каждый элемент которого представляет собой количество положительных элементов соответствующей строки заданного вещественного двумерного массива `b`.

Для сравнения с предыдущим примером, приведем решение задачи формирования одномерного целочисленного массива, в который будем записывать количества положительных элементов каждой строки, а не каждого столбца динамического двумерного массива. Программный код этой функции отличается от кода предыдущей (примера 7.9) только тем, что внешний цикл теперь требуется организовать по строкам, а внутренний – по столбцам и, соответственно, память под одномерный массив выделять по числу строк, а не столбцов двумерного массива.

7.7 Типовые процедуры формирования и вывода двумерных массивов

```
// функция вычисляет кол-во положит. элементов
// в каждой строке динамической матрицы b и записывает
// его в одномерный динамический массив a
int* Pr710_1(float** b,int m,int n)
{
    int* a=new int[m];      //выделение памяти под массив a
    for (int i=0;i<m;i++)   //внешний цикл по строкам
    {
        int k=0;            //счетчик полож. эл-тов в i-ой строке
        for(int j=0;j<n;j++)
            if(*(*(b+i)+j)>0) k++;
        *(a+i)=k;           //запись счетчика в одномерный массив
    }
    return a;               //возврат указателя на сформированный массив
}
```

А теперь решим эту же задачу по–другому, и интерпретируем двумерный массив, как одномерный, состоящий из строк исходного двумерного массива. Поэтому можно подсчет числа положительных элементов каждой строки представить, как подсчет числа положительных элементов одномерного массива. Для этого создадим дополнительную функцию.

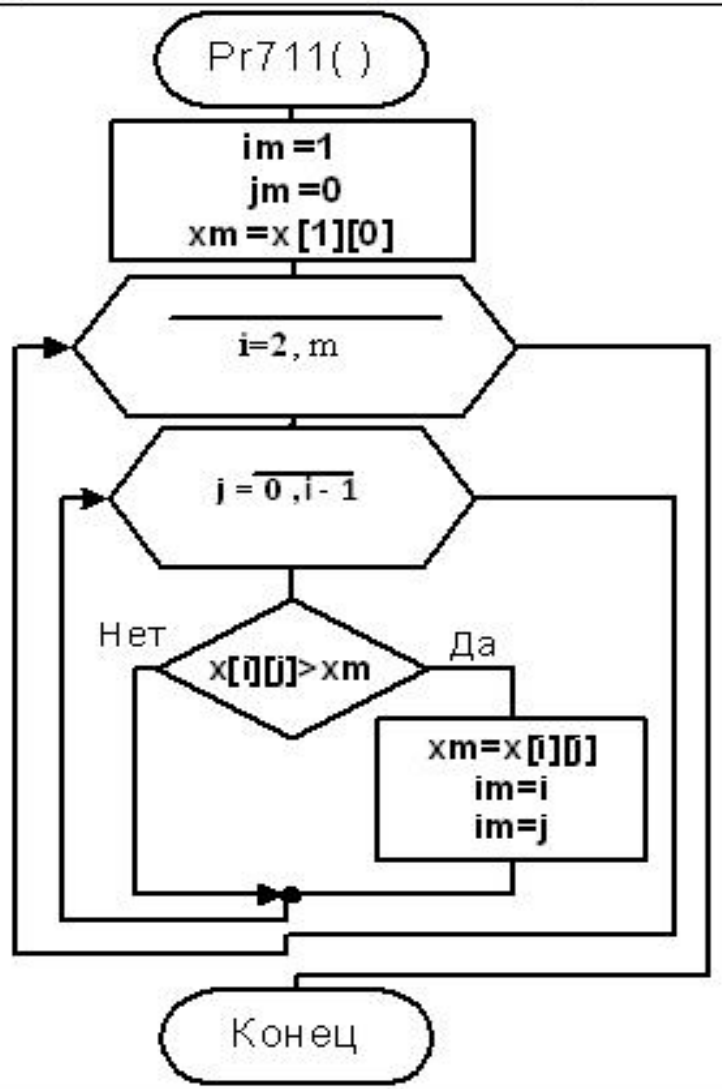
```
// функция вычисляет число положительных элементов в массиве
int quantity(float* x, int n)
{
    int k=0; //счетчик полож. эл-тов в массиве
    for (int i=0;i<n;i++)//цикл по всем эл-там
        if(*(x+i)>0) k++;
    return k;//возврат счетчика в вызывающую ф-цию
}
```

Продолжение решения примера 7.10 (2 способ). Теперь только один цикл по числу строк, в котором вызывается ф-ция quantity с двумя фактическими пар-рами: это указатель на *i*-ю строку матрицы и число элементов в строке (т.е. число столбцов)

```
// функция создает одномерный динамический массив, записывая в него
// количество положит. эл-тов в каждой строке динамической
// матрицы используя результат работы функции  quantity
int* Pr710_2(float** b,int m,int n)
{
    int* a=new int[m];//выделение памяти под массив a
    for(int i=0;i<m;i++)//цикл по строкам матрицы
    {
        int k=quantity(*(b+i),n);//вызов ф-ции подсчета
                                 //числа полож. эл-тов
                                 //для i-ой строки матрицы
        *(a+i)=k;                //запись результата работы ф-ции
                                 // в одномерный массив
    }
    return a; //возврат указателя на сформированный массив
}
```

7.7 Типовые процедуры формирования и вывода двумерных массив

Пример 7.11. Разработать процедуру, которая определяет значения наибольшего элемента двумерного вещественного массива и его индексов среди элементов, лежащих ниже главной диагонали.




```
// функция находит ниже главной диагонали
//максимальный элемент и его индексы
float Pr711(float** x, int m, int& im,
                int& jm)
{
    float xm=*(*(x+1)+0);//максимальный эл-т
    im=1;jm=0;              //его индексы
    for(int i=2;i<m;i++)
        for (int j=0;j<i;j++)
            if(*(*(x+i)+j)>xm)
            {
                xm=*(*(x+i)+j);
                im=i;
                jm=j;
            }
    return xm;
}
```

7.7 Типовые процедуры формирования и вывода двумерных массивов

Входными параметрами процедуры являются указатель на динамический двумерный массив **x**, и число его строк **m**, которое для квадратной матрицы совпадает с числом столбцов. Выходными параметрами являются переменные **im** и **jm** – номера строки и столбца, на пересечении которых находится наибольший элемент, а возвращаемым значением – значение вычисленной в функции переменной **xm** – наибольшего элемента массива среди расположенных под главной диагональю.

Вначале задаются начальные значения переменных, в которых будут сформированы результаты работы процедуры. Так как в строке с номером **0** нет элементов, расположенных под главной диагональю, то за начальное значение наибольшего элемента принимается единственный элемент **1**–й строки, находящийся под главной диагональю в столбце с номером **0** – **x[1][0].** В программе доступ к этому элементу, как и к остальным элементам массива, будем записывать через операцию разыменования ***(*(x+1)+0)** – заметим, что, конечно, это выражение можно было записать и без нуля, как ***(*(x+1))**. Соответственно, в переменных **im** и **jm** фиксируются его индексы **im = 1**, **jm = 0.**

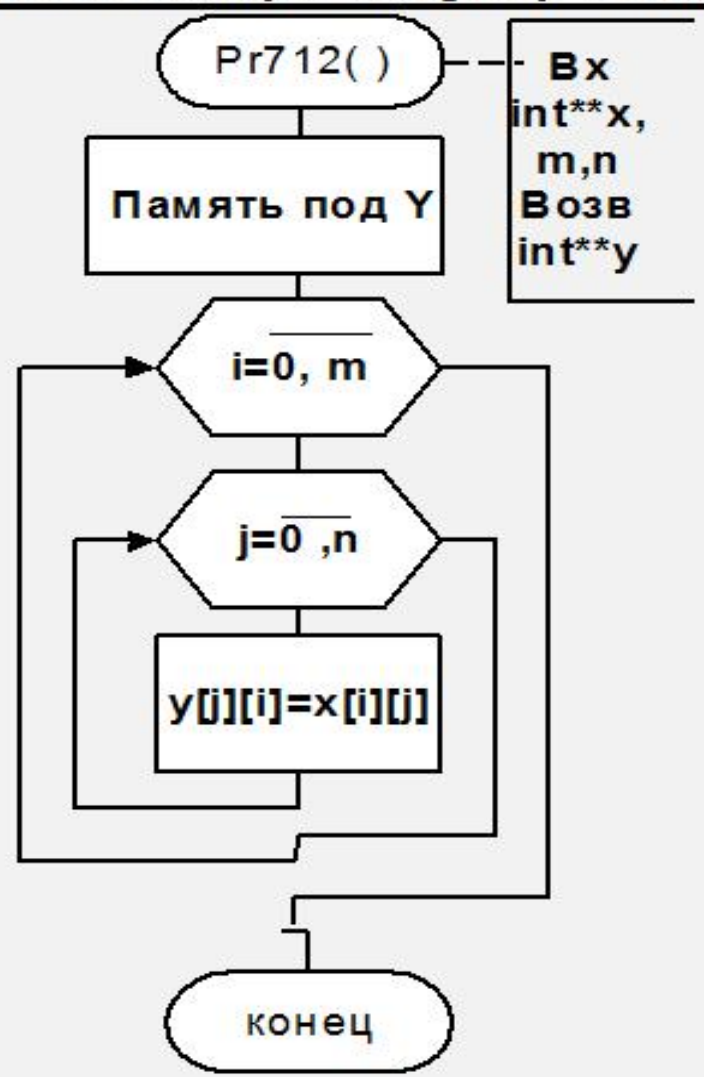
7.7 Типовые процедуры формирования и вывода двумерных массивов

Далее организуются вложенные циклы по строкам и столбцам массива для сравнения очередного элемента под главной диагональю с текущим наибольшим значением **xm**. Если очередной элемент **x[i][j]** больше **xm**, то в **xm** записывается этот текущий элемент, а в **im** и **jm** – его индексы. После завершения внешнего цикла в этих трех переменных окажутся искомые значения.

Так как в строке с номером **1** всего один элемент расположен под главной диагональю, то внешний цикл начинается со строки с номером **2**. Внутренний цикл в каждой **i**–й строке начинается со столбца с номером **0** и заканчивается столбцом с номером **i–1**, поскольку остальные элементы в этой строке находятся над главной диагональю или на ней.

7.7 Типовые процедуры формирования и вывода двумерных массивов

Пример 7.12. Разработать процедуру, которая транспонирует вещественную матрицу **x** в матрицу **y** .




```
// функция создает  динамический двумерный
// массив y, путем транспонирования
// из динамического двумерного массива x,
// содержащего  m строк и n столбцов
int** Pr712(int** x, int m,int n)
{
   //выделение памяти под массив y
   int** y=new int* [n]; //под n указателей
                         // на строки
   for(int i=0;i<n;i++)
      *(y+i)=new int [m];// под m эл-тов в
                         // каждой строке
   // транспонирование
   for(int i=0;i<m;i++)
      for(int j=0;j<n;j++)
         *(*(y+j)+i)=*(*(x+i)+j);
   return y;
}
```

7.7 Типовые процедуры формирования и вывода двумерных массивов

Входными параметрами процедуры являются указатель на двумерный массив **x** (исходная матрица), число строк **m** и число столбцов **n** исходной матрицы, возвращаемым значением – указатель на двумерный массив **y** (транспонированная матрица).

В начале процедуры распределяется память под результирующий массив **y**, причем верхние границы индексов меняются: **n** становится верхней границей числа строк, а **m** –верхней границей числа столбцов.

Далее организуются вложенные циклы, и в теле внутреннего цикла происходит присваивание каждому элементу массива **y** значения соответствующего элемента массива **x**, причем это соответствие устанавливается путем замены номера строки на номер столбца и наоборот.

Примечание: матрица **y** является транспонированной к матрице **x**, если строки матрицы **y** являются столбцами матрицы **x**, а столбцы матрицы **y** являются строками матрицы **x**. Отсюда следует, что количество строк матрицы **y** равно количеству столбцов матрицы **x**, а количество столбцов матрицы **y** равно количеству строк матрицы **x**.

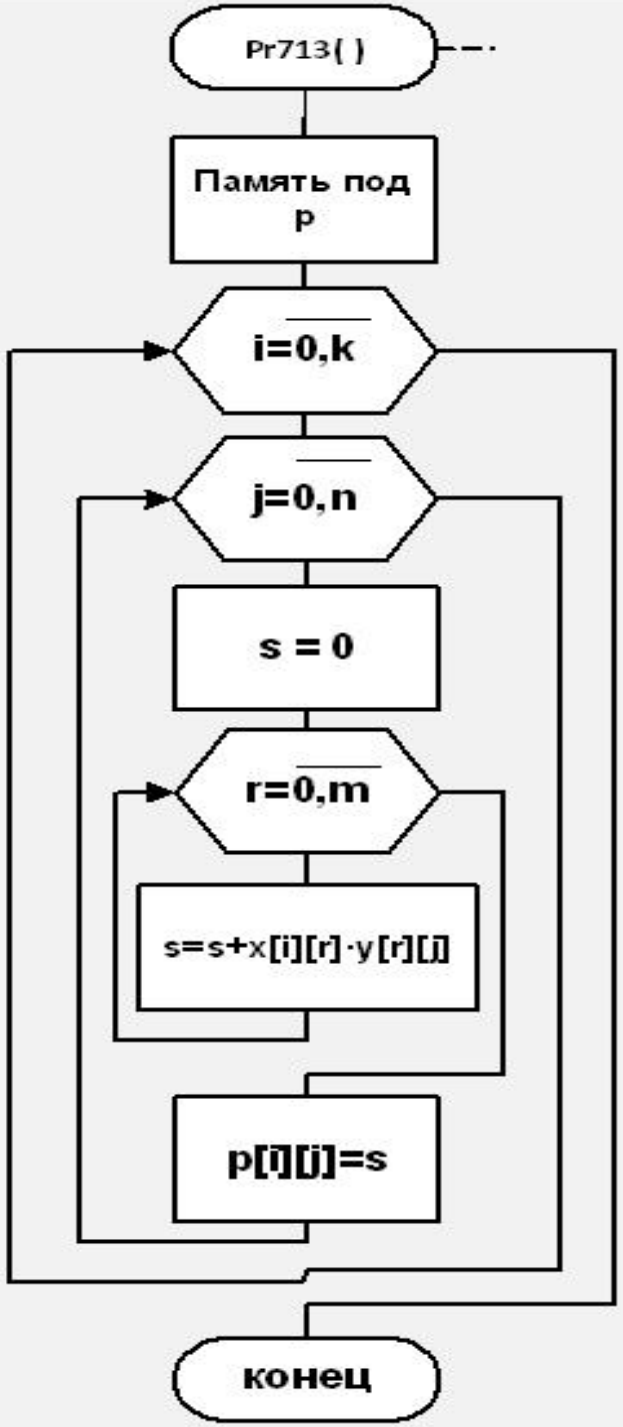
Пример 7.13. Разработать процедуру, которая вычисляет произведение матрицы `X(k, m)` на матрицу `Y(m, n)`.

Произведением вектора-строки `a(n)` на вектор-столбец `b(n)` с одинаковым числом элементов `n` называется сумма произведений соответствующих элементов этих векторов

$$\sum_{i=0}^{n} a_i \cdot b_i$$

Таким образом, произведение двух матриц можно записать следующим образом:

$$p_{ij} = \sum_{r=0}^{m} x_{ir} * y_{rj}, \text{где } i = \overline{0, k};\ j = \overline{0, n}$$




```
//Ф-ция вычисляет произведение матрицы
// x размера(k x n)на матрицу y размера(m x n)
// и получает матрицу p размера(k x m)
float**  Pr713(float** x, float** y,
               int k,int n,int m)
{
    //выделение памяти под массив y
    float** p=new float* [k];//под k указателей
                             // на строки
    for(int i=0;i<k;i++)
        *(p+i)=new float [n];// под n эл-тов в
                             // каждой строке
    // вычисление произведения в 3-х циклах:
    for(int i=0;i<k;i++)
        for(int j=0;j<n;j++)
        {
            //вычисление произведения
            // i-го вектора-строки x
            // на j-ый вектор-столбец y :
            float s=0;// сумма произведений
                      //элементов векторов
            for(int r=0;r<m;r++)
                s+=*(*(x+i)+r)* (*(*(y+r)+j));
            *(*(p+i)+j)= s;
        }
    return p;
}
```

7.7 Типовые процедуры формирования и вывода двумерных массивов

Входными параметрами процедуры являются указатели на двумерные массивы **x** и **y** (исходные матрицы) и их размеры **k, m** и **n**, возвращаемым значением – указатель на двумерный массив **p** (матрица–произведение). Предполагается, что условие равенства числа столбцов массива **x** и числа строк массива **y** выполняется и проверено в вызывающей процедуре.

В начале процедуры выделяется память под динамический массив–произведение **p**. Далее организуются вложенные циклы с параметрами **i** и **j** для формирования элементов массива **p.** В теле внутреннего цикла организован еще один вложенный цикл с параметром **r** для накопления в переменной **s** суммы произведений элементов исходных массивов в соответствии с приведенной выше формулой. Накопленная сумма присваивается очередному элементу результирующего массива.

Примечание: Операция умножения матрицы **X** на матрицу **Y** определена только для случая, когда число столбцов матрицы **X** равно числу строк матрицы **Y**.

Произведением матрицы **X(k,m)** на матрицу **Y(m,n)** называется матрица **P(k,n),** в которой элемент, стоящий на пересечении **i** -й строки и **j**-го столбца, равен произведению **i**-го вектора-строки матрицы **X(k,m)** на **j**-й вектор-столбец матрицы **Y(m,n).**