Лекция 7 - продолжение

Формирование и обработка двумерных массивов

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.

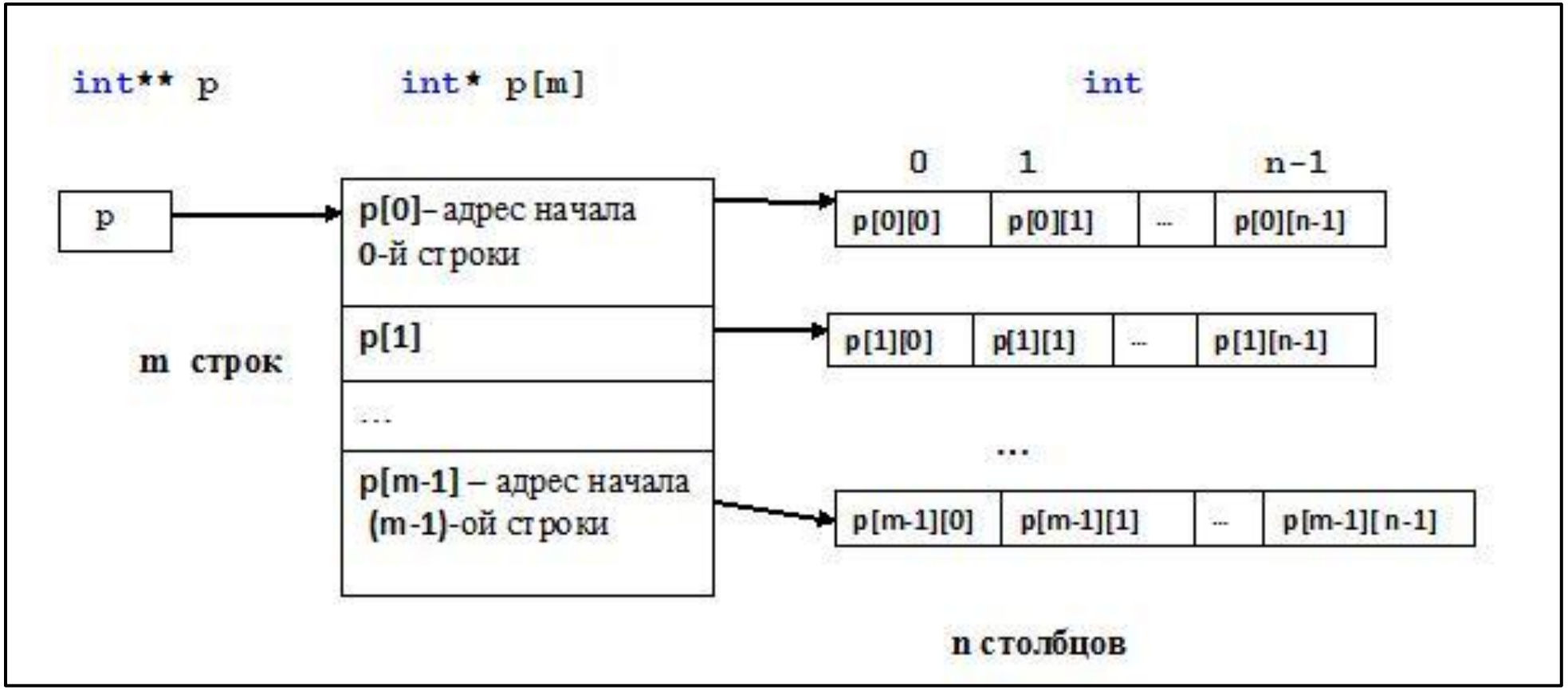
```
// создание динамического двумерного массива
int m,n;                    // число строк и столбцов матрицы
...
// вычисляются значения  m и n
...
int** p=new int*[m];//выделение памяти под массив
                    //из m указателей на строки матрицы
for (int i=0;i<m;i++)
    p[i]=new int[n];//динамически 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**.

7.6 Динамические двумерные массив

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. Также надо понимать, что 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 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, показан на рис.13, 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.

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 представлен на рис. 14. 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

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 массив

```
// 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
}
```

7.7 Типовые процедуры формирования и вывода двумерных 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
}
```

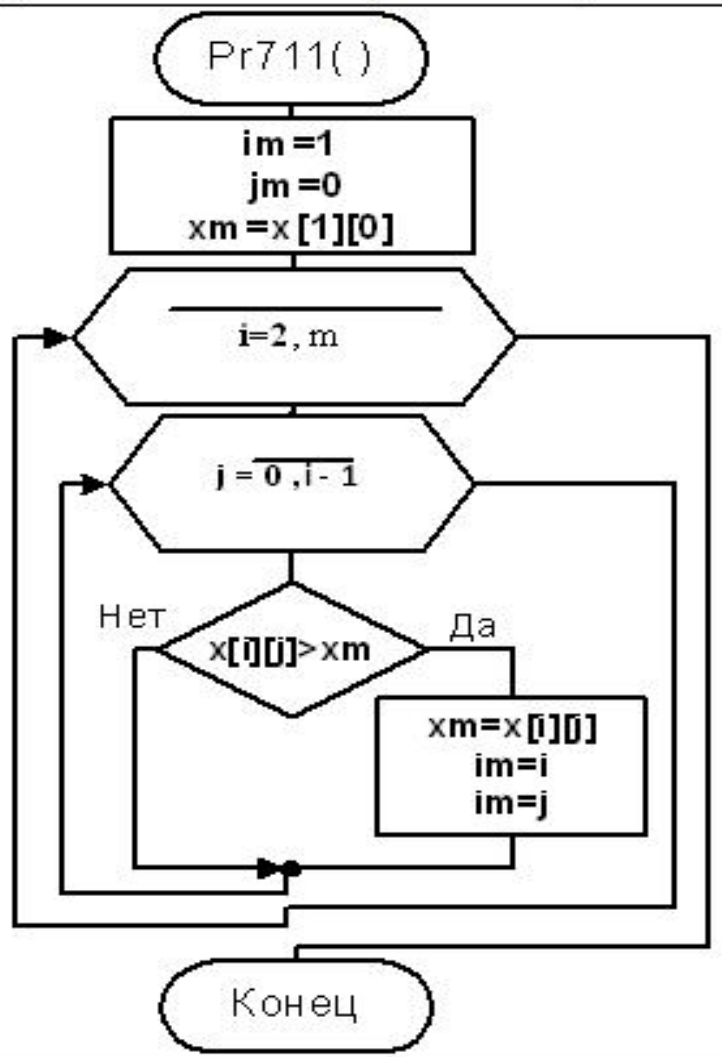
7.7 Типовые процедуры формирования и вывода двумерных массив

Далее создадим процедуру формирования одномерного массива согласно условию задачи. В этой процедуре теперь только один цикл по количеству строк матрицы. В теле цикла вызывается дополнительная функция **quantity**, которой в качестве фактических параметров передается указатель на **i**-ую строку матрицы и количество элементов в строке (т.е. количество столбцов **n**). Результат работы функции присваивается переменной **k**, а затем и очередному элементу формируемого одномерного массива.

```
// 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

Схема алгоритма и программный код процедуры приведены на рис.18. 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.

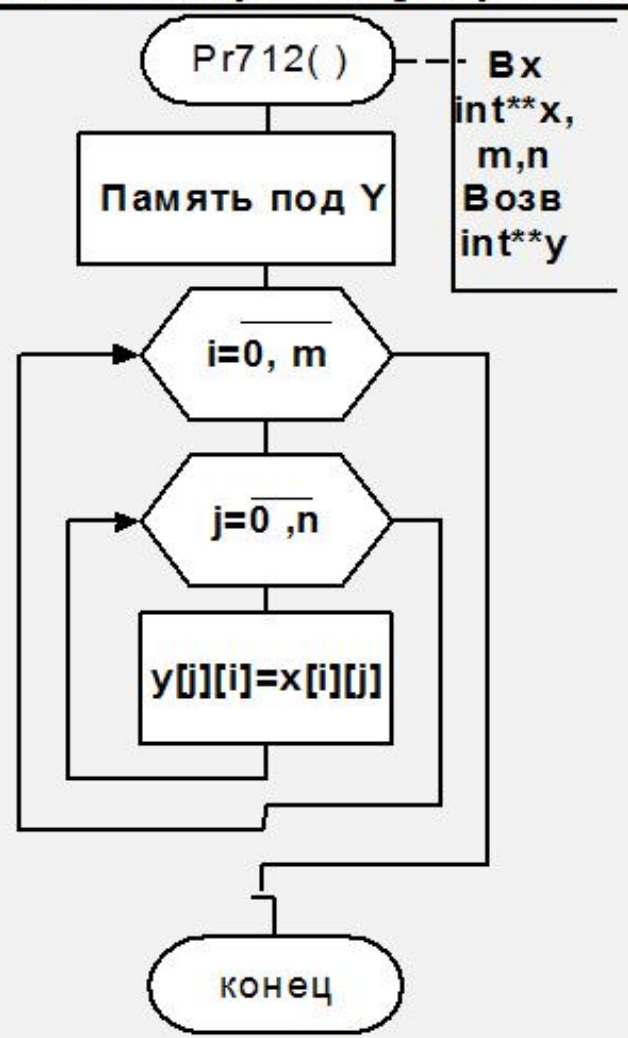
Вначале задаются начальные значения переменных, в которых будут сформированы результаты работы процедуры. Так как в строке с номером **0** нет элементов, расположенных под главной диагональю, то за начальное значение наибольшего элемента принимается единственный элемент **1**–й строки, находящийся под 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.**

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 массив

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**.

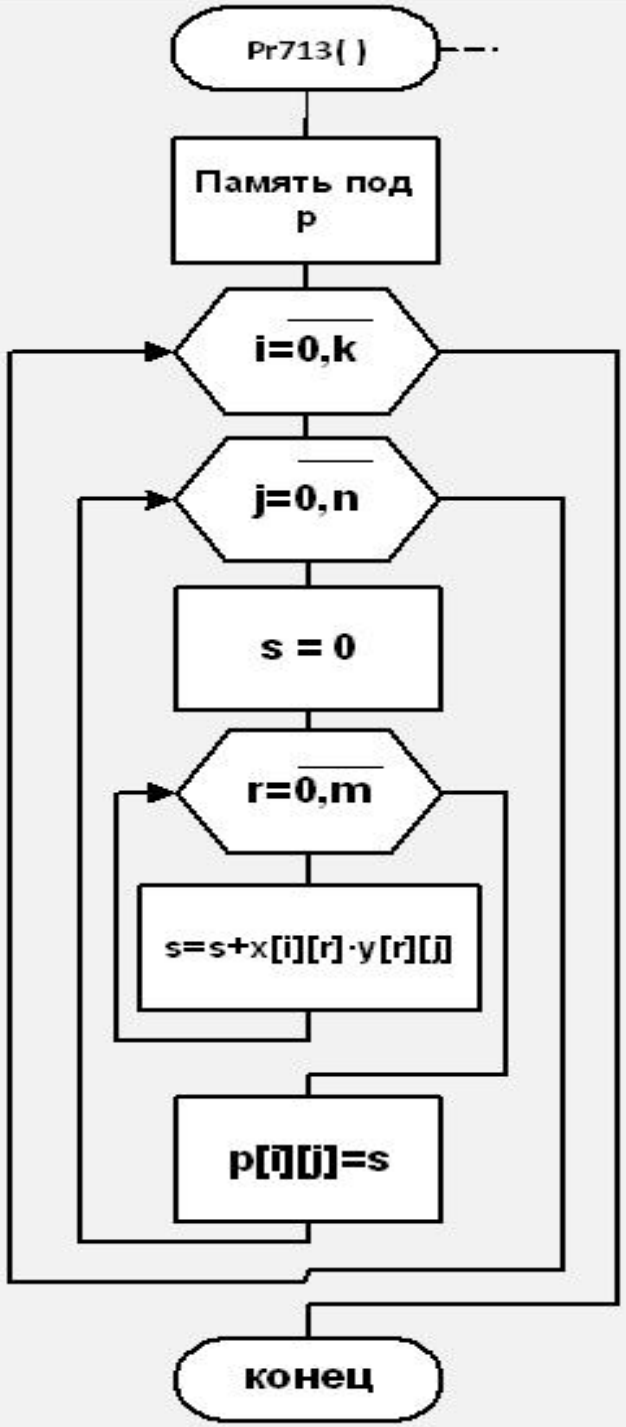
Пример 7.13. Разработать процедуру, которая вычисляет произведение матрицы `X(k, m)` на матрицу `Y(m, n)`.

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 массив

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).**

7.8 Примеры выполнения лабораторного задания

7.8.1 Первый пример выполнения лабораторного задания

1. Задание. Сформировать и вывести двумерный массив из натуральных случайных чисел. Создать новый массив, записав в него только те строки исходного массива, в которых есть хотя бы одно простое число. Если таких строк нет - то новый массив не создается.

7.8.1 Первый пример выполнения лабораторного задания

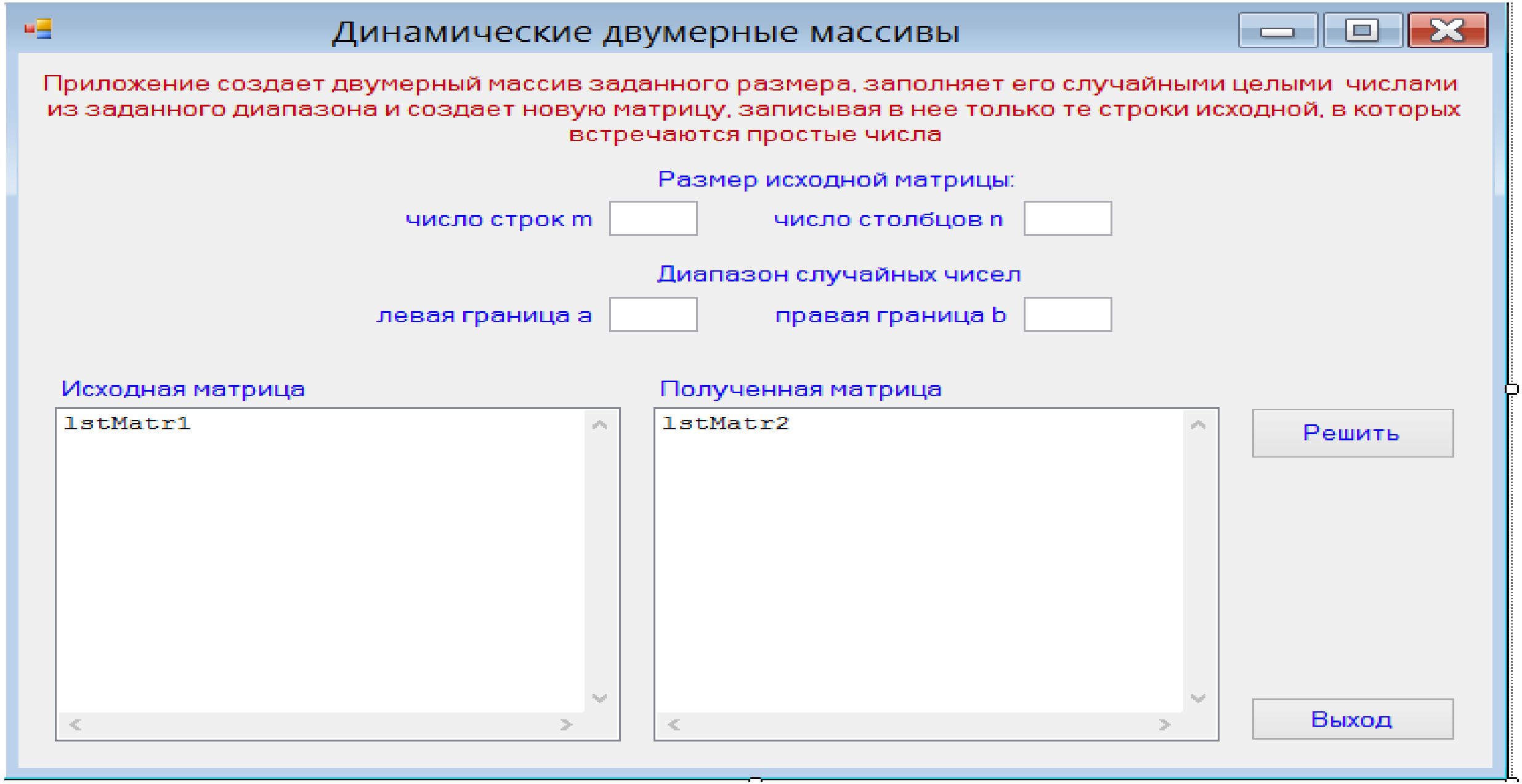
- **Формализация задачи.** Из условия задания следует, что его выполнение распадается на следующие этапы:
- ввод исходных данных (размеры исходного массива и границы диапазона случайных чисел);
- формирование исходного массива;
- вывод исходного массива;
- определение количества строк в исходном массиве, которые содержат хотя бы одно простое число, если таких строк нет, то новый массив создаваться не будет; очевидно, что этот этап необходимо разбить на части:
- проверка (анализ), имеется ли в строке матрицы хотя бы одно простое число;
- проверка, является ли натуральное число простым;
- формирование нового массива из строк исходного с простыми числами;
- вывод нового массива;
- освобождение памяти сформированных массивов.

7.8.1 Первый пример выполнения лабораторного задания

3.Программная реализация задания предполагает создание следующих процедур:

- функцию **GetInt** ввода целого числа с контролем непустого текстового поля и положительного значения – для ввода размеров исходного массива и границ диапазона натуральных случайных;
- функцию **input** для формирования двумерного массива из случайных натуральных чисел;
- функцию **output** для вывода двумерного массива целых чисел в заданный;
- функцию **simple,** которая проверяет, является ли ее формальный параметр простым числом;
- функцию **analys**, которая анализирует, есть ли в одной строке матрицы хотя бы одно простое число; так как одну строку матрицы можно рассматривать как одномерный массив, то можно использовать уже созданную для предыдущей лабораторной работы функцию проверки; функцию **analysMatr,** которая считает, сколько строк имеют хотя бы одно простое число;
- функцию **copy** для копирования одной строки исходной матрицы в строку другой матрицы – сводится к копированию элементов одного одномерного массива в другой;
- функцию **task,** которая формирует новый массив из тех строк исходного массива, в которых есть хотя бы одно простое число;
- функцию **del,** которая освобождает память, выделенную для динамической матрицы.

7.8.1 Первый пример выполнения лабораторного задания



7.8.1 Первый пример выполнения лабораторного задания

- В проект, помимо автоматически сгенерированных системой файлов, надо добавить 2 файла исходного кода: файл **GetPut.cpp** с функциями для ввода, вывода и освобождения памяти и файл **task.cpp** с остальными функциями, необходимыми для решения задачи**.**
- Программный код файла **GetPut.cpp**. приведен на рис. 22. В нем записаны следующие функции: **input** – для формирования динамического двумерного массива из натуральных случайных чисел; **GetInt** - для ввода целого числа с контролем непустого текстового поля и положительного значения; **output** – для вывода двумерного массива в заданный список формы; **del** - для освобождения памяти матрицы.
- Функция **del** имеет два параметра: указатель на двумерный массив и число строк в массиве. В цикле по числу строк освобождается память, выделенная элементам текущей строки, а после его завершения освобождается память, занятая массивом указателей на строки.
- Все остальные функции этого файла были рассмотрены ранее.

7.8.1 Первый пример выполнения лабораторного задания

```
// файл GetPut с функциями ввода, вывода и освобождения
// выделенной динамической памяти
#include "stdafx.h"

// Формирование массива из случайных целых чисел
int** input(int m, int n, int a, int b)
{
    int** matr = new int* [m];//память под m указателей на строки
    if (a > b)                        // если левая граница диапазона
                                      // случ. чисел введена некорректно,
        {int t=a; a=b; b=t;}          // то a и b меняются значениями,
                                      // чтобы a было меньше b
    Random^ rnd = gcnew Random;       // создание объекта типа Random.
     for (int i = 0; i < m; i++)      // цикл заполнения массива.
    {
        *(matr+i) = new int [n]; // память под n эл-тов в строке
        for (int j=0; j<n ;j++)
// заполнение элементов матрицы случайными значениями из [a, b].
        *(*(matr+i)+j) = rnd->Next(a,b);
    }
    return matr;       // возвращается указатель на созданную матрицу
}
```

7.8.1 Первый пример выполнения лабораторного задания

```
// определение функции ввода целого числа с контролем
//непустого текст. поля и положительного значения
bool GetInt(int& x,TextBox^ Tx,String^ s)
{
    if (Tx->Text->Length==0)//если текс поле пусто
    {
        MessageBox::Show(s,"Ошибка",MessageBoxButtons::OK,
                                MessageBoxIcon::Error);
        Tx->Focus();
        return false;// признак ошибки ввода
    }
    x= Convert::ToInt16(Tx->Text);
    if(x<1)
    {
        MessageBox::Show(s,"Ошибка",MessageBoxButtons::OK,
                                MessageBoxIcon::Error);
        Tx->Focus();
        return false;// признак ошибки ввода
    }
    return true;// успешный ввод
}
```

7.8.1 Первый пример выполнения лабораторного задания

```
// Функция выводит матрицу размера m x n в ListBox
void output(int** 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,-5:D2}",*(*(mas+i)+j));
        Lb->Items->Add(s);
    }
}

// Освобождение  памяти, выделенной для матрицы из m строк
void del(int** a, int m)
{
    for(int i=0; i<m; i++)
        delete[] *(a+i);
    delete[] a;
}
```

7.8.1 Первый пример выполнения лабораторного задания

```
// Файл task.cpp с функциями решения задачи
#include "stdafx.h"

// Функция определяет, является ли натуральное число простым
bool simple (int n)
{
    for(int i=2; i<=n/2; i++)
        if (n%i == 0) return false;
    return true;
}

// Функция проверяет, есть ли в одномерном массиве
// (т.е. в строке матрицы) хотя бы одно простое число
bool analys(int* a, int n)
{
    for (int i=0; i<n; i++)
        if (simple(*(a+i))) return true; // вызов функции проверки
                                         // простого числа
    return false;
}
```

7.8.1 Первый пример выполнения лабораторного задания

```
// Функция считает, сколько строк в матрице
// имеют хотя бы одно простое число
int analysMatr(int** a, int m, int n)
{
    int k=0;           // число строк, в которых есть простые числа
    for (int i=0; i<m; i++)          // цикл по строкам матрицы
        if (analys(*(a+i), n)) k++;  // вызов функции проверки
                                     // имеются ли в i-ой строке
                                     // матрицы простые числа
    return k;
}

// Функция копирования n элементов одномерного массива a в
//одномерный массив b, для копирования одной строки
// матрицы a в строку другой матрицы - b
void copy(int* b, int* a,  int n)
{
    for (int i=0; i<n; i++)
        *(b+i) = *(a+i);
}
```

7.8.1 Первый пример выполнения лабораторного задания

```
// Функция создания новой матрицы
int** task(int** matr, int m, int n, int& k )
{
    // Вызов функции анализа исходной матрицы
    k = analysMatr(matr ,m, n);      // k - число строк, в которых
                                     // есть простые числа
    if (!k) return 0;              // если k=0, то матрица не создается
    int** newMatr = new int*[k];     // память для k указателей
                                     // на строки новой матрицы
    k = 0;                           // счетчик строк новой матрицы
    for (int i=0; i<m; i++)        // проверяется каждая строка матрицы
        if (analys(*(matr+i),n))     // если в i-ой строке матрицы
        {                            // есть простое число
            *(newMatr+k) = new int[n]; // выделение памяти для
                                       // элементов этой строки,
            copy(*(newMatr+k), *(matr+i), n);   // копирование эл-тов
                                     // i-ой строки исходной матрицы
                                     // в k-ю строку новой матрицы
            k++;             // увеличение счетчика строк новой матрицы
        }
    return newMatr;        // возвращается указатель на новую матрицу
}
```

7.8.1 Первый пример выполнения лабораторного задания

- Следующая функция **analys** определяющая, имеется ли в одномерном массиве хотя бы одно простое число, также уже рассмотрена в предыдущей лабораторной работе №5. В данной работе мы используем ее для анализа одной строки двумерного массива, так как уже неоднократно отмечалось, что двумерный массив можно представить, как массив из одномерных массивов. Соответственно, одна строка двумерного массива – это одномерный массив.
- Функция **analysMatr** подсчитывает, сколько строк в матрице содержат хотя бы одно простое число. Функция имеет три параметра: указатель на двумерный массив, число строк массива **m** и число столбцов **n**. В теле функции объявляется и инициализируется нулем переменная **k**, для подсчета строк, содержащих простые числа. Далее организуется цикл по всем строкам матрицы. В цикле вызывается функция **analys** для проверки на наличие простого числа в одномерном массиве (т.е. в строке матрицы). Фактическими параметрами при вызове являются: указатель на **i**-ую строку матрицы и число элементов в строке – т.е. число столбцов **n**. Если функция вернула значение **true**, то в **i**-ой строке имеется простое число, и значение счетчика **k** таких строк увеличивается на 1. По окончании цикла, когда все строки матрицы проверены, это значение возвращается оператором **return** и функция **analysMatr** завершает свою работу.

7.8.1 Первый пример выполнения лабораторного задания

Следующая функция **copy** переписывает **n** элементов одномерного массива **a** в одномерный массив **b.** Функция имеет три параметра: указатель на исходный массив–источник, указатель на второй массив, в который копируются элементы исходного массива, и число копируемых элементов **n.** Переписывание этих элементов происходит в цикле по числу элементов **n.** Функция предназначена для копирования одной строки матрицы в строку другой матрицы.

7.8.1 Первый пример выполнения лабораторного задания

- Последняя функция в файле **task.cpp** – функция **task** создает новый динамический двумерный массив из тех строк исходного массива, в которых есть хотя бы одно простое число. Входными параметрами функции являются указатель на исходный двумерный массив **matr**, число строк **m** и число столбцов **n** исходного массива, выходным – счетчик строк нового двумерного массива (как параметр по ссылке), возвращаемым значением – указатель на новый массив. В теле функции, прежде всего вызывается функция **analysMatr**, которая определяет, сколько строк исходного массива содержат хотя бы одно простое число и записывает количество найденных строк в выходной параметр – переменную **k**. Если **k=0**, то новый массив не создается, а функция **task** оператором **return** возвращает ноль и прекращает свою работу. Если же значение **k** не равно нулю, выделяется динамическая память под массив из **k** указателей на строки для новой матрицы, адрес которой присваивается указателю **newMatr**. Затем, после обнуления счетчика строк **k** новой матрицы, организуется цикл по количеству строк исходной матрицы. В цикле для каждой **i-**ой строки исходной матрицы вызывается функция **analys.** Если функция **analys** вернула **true**, то выделяется память под **n** элементов для **k**-й строки новой матрицы, в которую с помощью функции **copy** копируется подходящая строка исходной матрицы и счетчик **k** строк новой матрицы увеличивается на 1.
- В конце работы функция **task** возвращает оператором **return** указатель **newMatr** на созданный новый двумерный массив, а через параметр по ссылке – число строк нового массива **k**.

7.8.1 Первый пример выполнения лабораторного задания

После того, как эти файлы исходного кода созданы и успешно откомпилированы, создается заголовочный файл **task.h** , в который из файлов исходного кода переносятся директивы **using** и записываются прототипы необходимых функций. Затем редактируется системный заголовочный файл **stdafx.h**, добавлением в него директивы компилятора для подключения созданного заголовочного файла **task.h.** На рисунке 24 приведен программный код созданного заголовочного файла **task.h** и системного заголовочного файла **stdafx.h**

7.8.1 Первый пример выполнения лабораторного задания

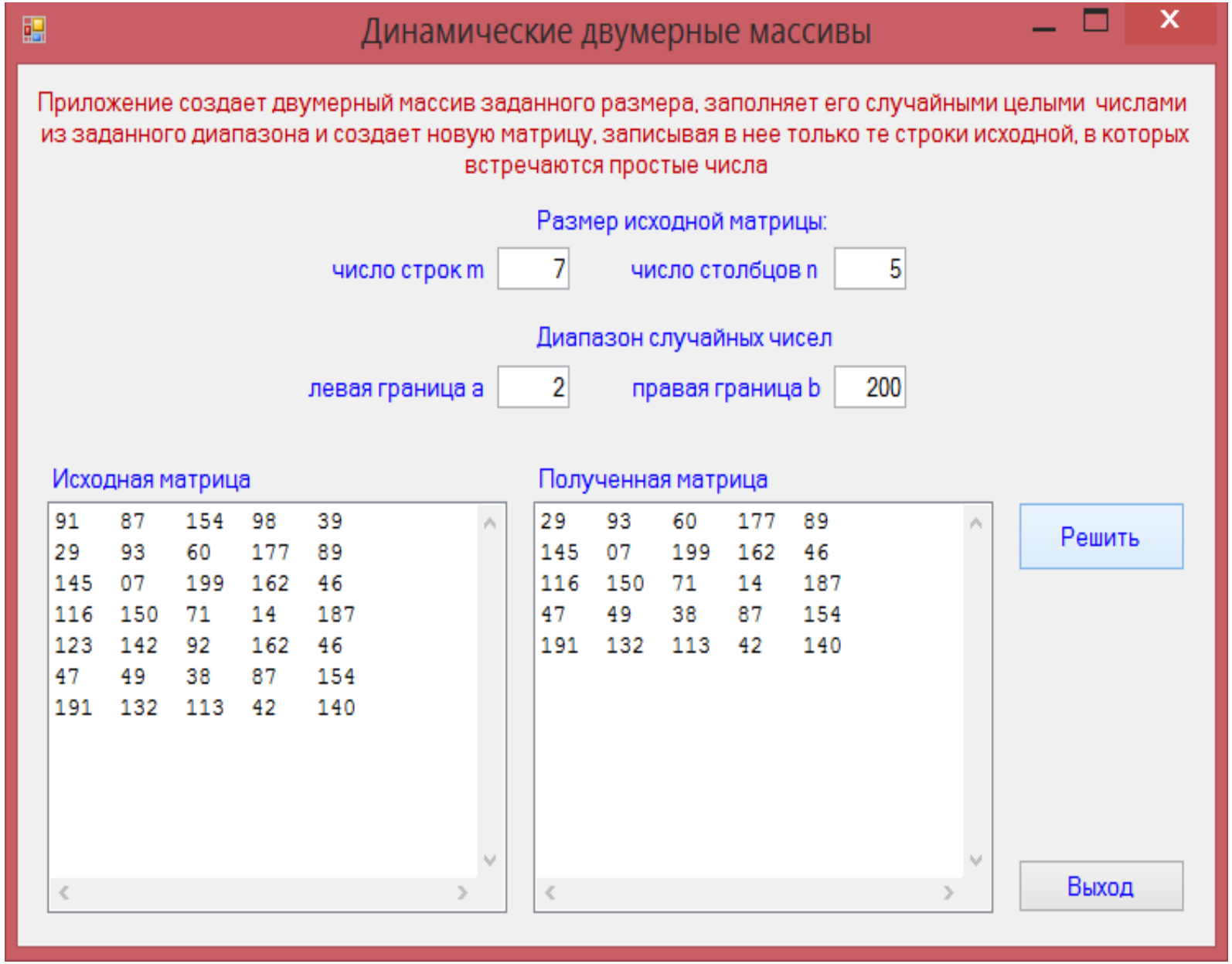
```
// заголовочный файл task.h
using namespace System;
using namespace System::Windows::Forms;

// прототипы функций
int** input(int m, int n, int a, int b);
bool GetInt(int&, TextBox^, String^);
void output(int** , int m, int n, ListBox^);
int** task(int** , int, int, int&);
void del(int** , int);

// файл stdafx.h:
#pragma once

#include "task.h"
```

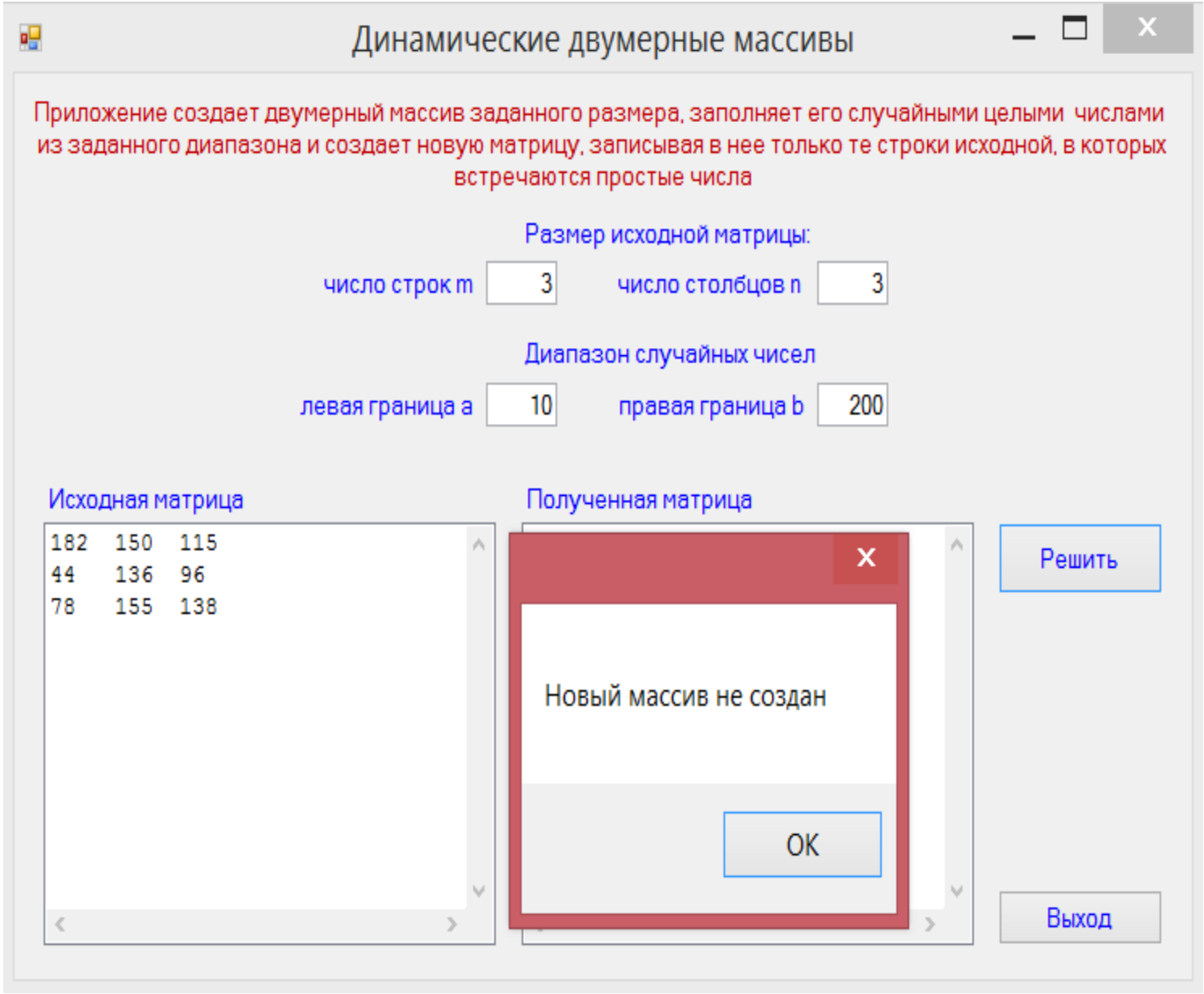
7.8.1 Первый пример выполнения лабораторного задания

```
// Событийная процедура нажатия на кнопку "Решить"
private: System::Void cmdTask_Click(System::Object^  sender,
                                    System::EventArgs^  e)
{
    lstMatr1->Items->Clear();    // очистка списков
    lstMatr2->Items->Clear();
    int m, n;                   // число строк и столбцов исходной матрицы
    int a, b;                   //диапазон случайных чисел
    if (!GetInt(m, txtM, "Введите число строк матрицы")) return;
    if (!GetInt(n, txtN, "Введите число столбцов матрицы")) return;
    if (!GetInt(a, txtA, "Введите левую границу диапазона")) return;
    if (!GetInt(b, txtB,"Введите правую границу диапазона")) return;
    int** matr = input(m,n,a,b);     //создание исходной матрицы
    output(matr,m,n,lstMatr1);       //вывод исходной матрицы
    int k;                           //число строк новой матрицы
    int** Matr2 = task(matr,m,n,k); //создание новой матрицы
    if (Matr2)                       // если новая матрица создана
    {
        output(Matr2, k, n, lstMatr2);  //вывод новой матрицы
        del(Matr2, k);          //освобождение памяти новой матрицы
    }
```

7.8.1 Первый пример выполнения лабораторного задания

```
    else
        MessageBox::Show("Новый массив не создан");
    del(matr,m);                    //освобождение памяти исходной матрицы
}

    // Событийная процедура нажатия на кнопку "Выход"
    private: System::Void cmdExit_Click(System::Object^  sender,
                                        System::EventArgs^  e)
    {
        this->Close();
    }
```


Динамические двумерные массивы
Приложение создает двумерный массив заданного размера, заполняет его случайными целыми числами из заданного диапазона и создает новую матрицу, записывая в нее только те строки исходной, в которых встречаются простые числа
Размер исходной матрицы:
число строк m
7
число столбцов n
5
Диапазон случайных чисел
левая граница a
2
правая граница b
200
Исходная матрица
91 87 154 98 39
29 93 60 177 89
145 07 199 162 46
116 150 71 14 187
123 142 92 162 46
47 49 38 87 154
191 132 113 42 140
Полученная матрица
29 93 60 177 89
145 07 199 162 46
116 150 71 14 187
47 49 38 87 154
191 132 113 42 140
Решить
Выход


Динамические двумерные массивы
Приложение создает двумерный массив заданного размера, заполняет его случайными целыми числами из заданного диапазона и создает новую матрицу, записывая в нее только те строки исходной, в которых встречаются простые числа
Размер исходной матрицы:
число строк m
3
число столбцов n
3
Диапазон случайных чисел
левая граница a
10
правая граница b
200
Исходная матрица
182 150 115
44 136 96
78 155 138
Полученная матрица
Новый массив не создан
OK
Решить
Выход

7.8.2 Второй пример выполнения лабораторного задания

1. **Задание**. Сформируйте и выведите двумерный вещественный массив **X** из случайных чисел. Вставьте столбец из единиц перед столбцом, содержащим минимальный элемент массива, и выведите полученный массив.

7.8.2 Второй пример выполнения лабораторного задания

1. Формализация задачи.

Из условия задания следует, что его выполнение

- распадается на следующие этапы:
- ввод исходных данных (размеры массива **X**);
- формирование массива **X**;
- вывод массива **X**;
- определение столбца с минимальным элементом массива;
- вставка столбца из единиц перед найденным столбцом;
- вывод массива **X** после вставки.

7.8.2 Второй пример выполнения лабораторного задания

Для программной реализации задания создадим следующие процедуры:

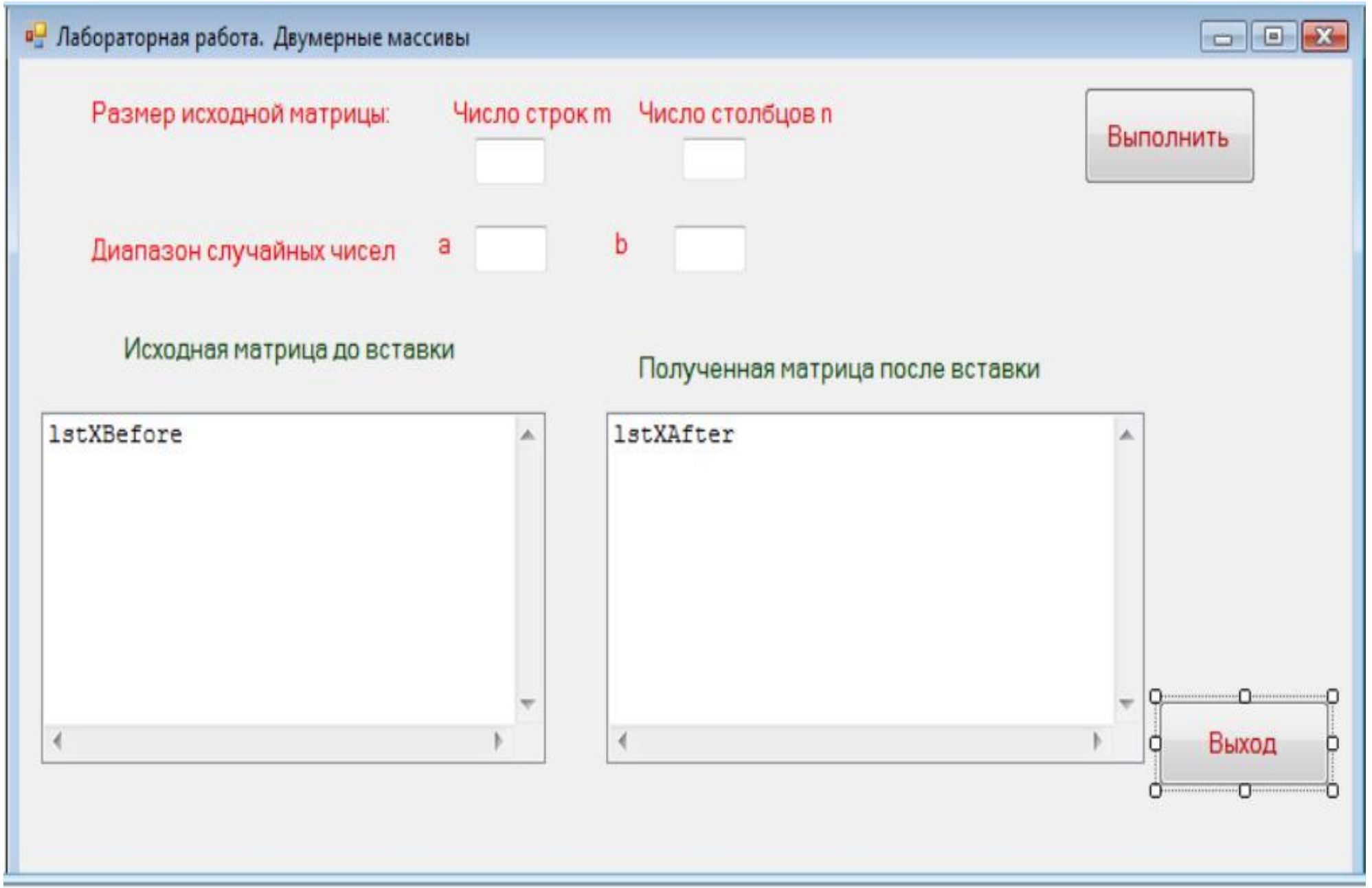
- **GetInt** – ввод целого числа с контролем непустого текстового поля и положительного значения для числа строк и столбцов массива;
- **GetFloat** – ввод вещественного числа для границ диапазона случайных чисел;
- **input** – формирование массива из случайных чисел (см. пример 7.7);
- **output** – вывод двумерного массива в заданный список формы (см. пример 7.8);
- **GetMinCol** – определение номера столбца массива, содержащего наименьший элемент;
- **InsertColumn** – вставка столбца с заданным значением всех его элементов перед столбцом с заданным номером.

7.8.2 Второй пример выполнения лабораторного задания

Ввод исходных данных и вызов указанных процедур будем производить в событийной процедуре по нажатию кнопки **Выполнить**.

Возможный вид формы проекта для организации интерфейса пользователя приведен на рисунке 27. В верхней части формы расположены текстовые поля для ввода исходных данных: количества строк массива (**txtM)**, количества столбцов массива (**txtN)** и границ диапазона случайных чисел (**txtA**) и (**txtB**). Ниже расположены два списка **ListBox:** список **lstXBefore** для вывода исходного массива **x** до вставки столбца и список **lstXAfter** – для вывода массива **x** после вставки. На форме также имеются две кнопки: кнопка **Выполнить (cmdTask)** для запуска программы на выполнение и кнопка **Завершить** (**cmdEnd**) для останова ее выполнения.

7.8.2 Второй пример выполнения лабораторного задания



7.8.2 Второй пример выполнения лабораторного задания

В проект, помимо автоматически сгенерированных системой файлов, надо добавить 2 файла исходного кода: файл **GetPut.cpp** с функциями для ввода, вывода и освобождения памяти и файл **task.cpp** с функциями **GetMinCol** и **InsertColumn.**

Программный код файла **GetPut.cpp**. приведен на рисунке 28. В нем записаны функции, которые мы уже рассматривали:

- **input** – для формирования динамического вещественного массива из случайных чисел;
- **GetInt** – для ввода числа строк и столбцов массива с контролем непустого текстового поля и положительного значения;
- **GetFloat** – для ввода границ диапазона случайных вещественных чисел с соответствующим контролем;
- **output** – для вывода двумерного массива в заданный список формы;
- **del** - для освобождения памяти матрицы.

7.8.2 Второй пример выполнения лабораторного задания

```
#include "stdafx.h"
// файл  GetPut функциями ввода, вывода и освобождения
// выделенной динамической памяти

// Формирование динамического массива размером m x n
// из случайных вещественных чисел [a, b]
float** input(int m,int n, float a,float b)
{
    float** matr=new float*[m];//память под m указателей
    if (a>b)                   //если левая граница диапазона
                               // случ. чисел введена некорректно,
    { float t=a; a=b; b=t;} //то a и b меняются значениями,
                             // чтобы a было меньше b
    Random^ rnd = gcnew Random;
    for (int i = 0; i < m; i++)   // Цикл заполнения массива
                        //случайными значениями с выделением памяти
    {
        *(matr+i)=new float [n];
        for (int j=0;j<n;j++)
            *(*(matr+i)+j) = a+(b-a) *rnd ->NextDouble();
    }
    return matr;// возвращается указатель на созданную матрицу
```

7.8.2 Второй пример выполнения лабораторного задания

```
// определение функции ввода целого числа с контролем
//непустого текст. поля и положительного значения
bool GetInt(int& x,TextBox^ Tx,String^ s)
{
    if (Tx->Text->Length==0)//если текс поле пусто
    {
        MessageBox::Show(s,"Ошибка",MessageBoxButtons::OK,
                         MessageBoxIcon::Error);
        Tx->Focus();
        return false;// признак ошибки ввода
    }
    x= Convert::ToInt16(Tx->Text);
    if(x<1)
    {
        MessageBox::Show(s,"Ошибка",MessageBoxButtons::OK,
                         MessageBoxIcon::Error);
        Tx->Focus();
        return false;// признак ошибки ввода
    }
    return true;// успешный ввод
}
```

7.8.2 Второй пример выполнения лабораторного задания

```
// определение функции ввода вещественного числа
//с контролем непустого текст. поля
bool GetFloat(float& x,TextBox^ Tx,String^ s)
{
    if (Tx->Text->Length==0)//если текс поле пусто
    {
        MessageBox::Show(s,"Ошибка",MessageBoxButtons::OK,
                                    MessageBoxIcon::Error);
        Tx->Focus();
        return false;// признак ошибки ввода
    }
    x= Convert::ToSingle(Tx->Text);
    return true;// успешный ввод
}
```

7.8.2 Второй пример выполнения лабораторного задания

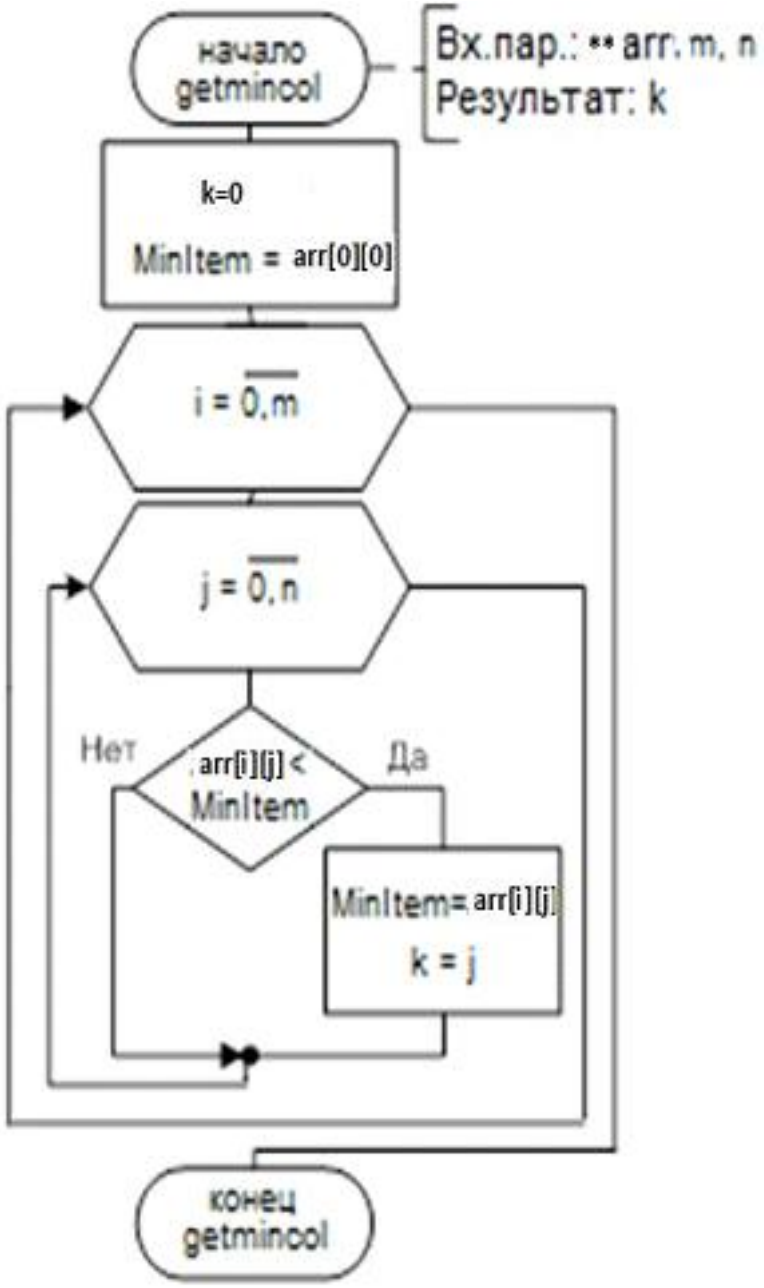
```
// функция выводит матрицу размера m x n в ListBox
void output(float** mas, int m, int n, ListBox^ Lb)
{
    Lb->Items->Clear();// очистка списка
    for (int i = 0; i < m; i++)
    {
        String^ s="";// строка для записи в ListBox
        for (int j=0;j<n;j++)
            s=s+String:: Format("{0,8:F2}",*(*(mas+i)+j));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

// освобождение  памяти, выделенной для матрицы из m строк
void del(float** a,int m)
{
    for(int i=0;i<m;i++)
        delete[] *(a+i);
    delete[] a;
}
```

7.8.2 Второй пример выполнения лабораторного задания

Основные две функции для решения задачи **GetMinCol** и **InsertColumn** записаны в файле **task.cpp**. Схемы алгоритмов и программные коды этих функций приведены на рисунках 29 и 30.

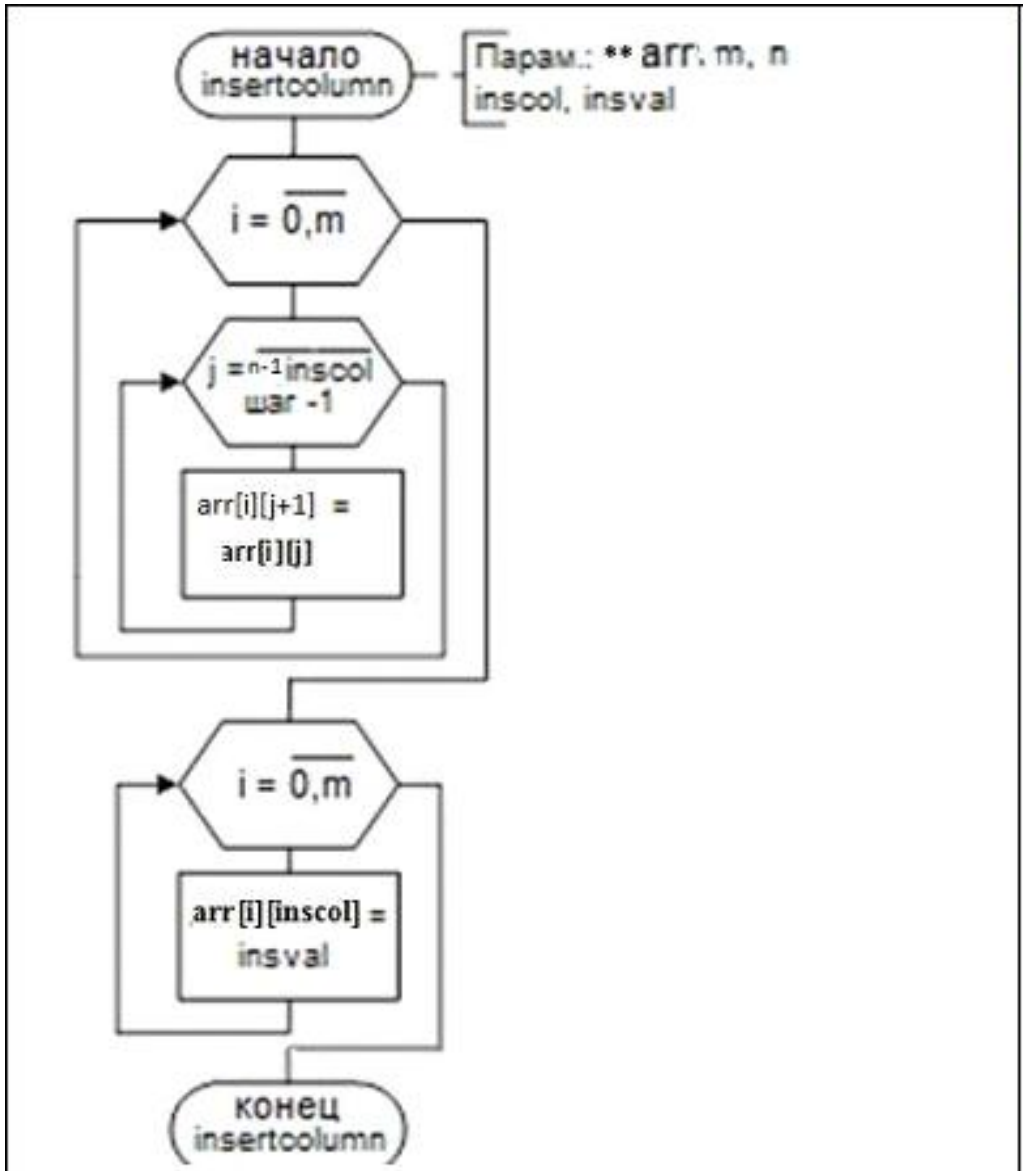
Процедура **GetMinCol** имеет три входных параметра – указатель на двумерный массив **Arr**, число строк **m** и число столбцов **n** этого массива. Возвращаемым результатом является переменная **k**, в которой после выполнения процедуры будет находиться номер столбца с наименьшим элементом массива. Переменная **MinItem** служит для хранения текущего наименьшего значения элементов массива. За начальное значение этой переменной принимается значение первого элемента массива **Arr[0][0],** и соответственно значение переменной **k**, в которой записывается номер столбца наименьшего элемента, устанавливается равным нулю.

7.8.2 Второй пример выполнения лабораторного задания




```
//Ф-ция находит номер столбца минимального
// элемента в двумерном массиве
int GetMinCol(float** arr, int m,int n)
{
    float MinItem=*(*(arr));//миним. эл-т
    int k=0;   // номер столбца миним. эл-та
    for(int i=0;i<m;i++)
        for(int j=0;j<n;j++)
            if(*(*(arr+i)+j)<MinItem)
            {
                MinItem=*(*(arr+i)+j);
                k=j;
            }
    return k;
}
```

7.8.2 Второй пример выполнения лабораторного задания




```
//ф-ция вставляет  столбец из единиц
// (значений insVal) перед столбцом
//  с номером insCol, содержащим минимальный
// элемент  двумерного массива
void insertColumn(float** arr, int m,int n,
                  int insCol, float insVal)
{
    for(int i=0;i<m;i++)
        for(int j=n-1;j>=insCol;j--)
            *(*(arr+i)+j+1)=*(*(arr+i)+j);//сдвиг
                // вправо для расширения массива
                //и освобождения позиции вставки
    for(int i=0;i<m;i++)
        *(*(arr+i)+insCol)=insVal; //вставка
}
```

7.8.2 Второй пример выполнения лабораторного задания

```
// заголовочный файл task.h
using namespace System;
using namespace System::Windows::Forms;

// прототипы функций
float** input(int m,int n, float a,float b);
bool GetInt(int& ,TextBox^ ,String^ );
bool GetFloat(float& ,TextBox^ ,String^ );
void output(float** mas, int m, int n, ListBox^ );
int GetMinCol(float** arr, int m,int n);
void insertColumn(float** arr, int m,int n, int insCol, float
insVal);
void del(float** a,int );


// файл stdafx.h:
#pragma once

// TODO.
#include "task.h"
```

7.8.2 Второй пример выполнения лабораторного задания

В событийной процедуре нажатия на кнопку «**Выполнить**» (**cmdTask_Click**) объявляются четыре переменные, которые с помощью соответствующих функций ввода с контролем непустого текстового поля получают значения числа строк **m** и столбцов массива **n** и **a, b** - границ диапазона случайных чисел. Так как в исходный массив вставкой будет добавляться еще один столбец, то необходимо выделить под массив память с одним «лишним» пока столбцом, поэтому введенная переменная **n**, увеличивается на **1**. затем последовательно вызывает процедуры:

- **input** для формирования динамического массива;
- **output** для вывода сформированного массива в список **lstXBefore,** причем так как один столбец в сформированном массиве лишний, то эта процедура вызывается с фактическим параметром для числа столбцов, равным **n-1**;
- **GetMinCol** для определения номера столбца с наименьшим элементом массива и присвоения найденного номера переменной **k**; эта процедура также вызывается с фактическим параметром для числа столбцов, равным **n-1;**
- **insertColumn** для вставки столбца перед столбцом с номером **k** и заполнения этого столбца значением **1**; также вызывается с фактическим параметром для числа столбцов, равным **n-1**;
- **output** для вывода измененного массива в список **lstXAfter**, вызывается с фактическим параметром для числа столбцов, равным **n**, так как после вставки в массив добавился один столбец.

7.8.2 Второй пример выполнения лабораторного задания

```
// Событийная процедура кнопки "Выполнить"
private: System::Void cmdTask_Click(System::Object^  sender,
                                    System::EventArgs^  e)

{
    lstXBefore->Items->Clear();   //очиска списков
    lstXAfter->Items->Clear();
    int m,n;              //число строк и столбцов исходной матрицы
    float a,b;            //диапазон случайных чисел
    if(!GetInt(m,txtM,"Введите число строк матрицы"))return;
    if(!GetInt(n,txtN,"Введите число столбцов матрицы"))return;
    if(!GetFloat(a,txtA,"Введите левую границу [a,b] "))return;
    if(!GetFloat(b,txtB,"Введите правую границу [a,b]"))return;
    n++;                  //увеличение числа столбцов, чтобы отвести
                          // необходимый размер памяти под массив
    float** x=input(m,n,a,b);//ввод исходного массива с запасом
                                  // памяти под лишний столбец
    output(x,m,n-1,lstXBefore);//вывод массива с меньшим числом
                                  // столбцов
    int k=GetMinCol(x,m,n-1); //определение номера столбца
                                  // с минимальным элементом
    lstXBefore->Items->Add("Номер столбца с мин. эл-том равен "
                    +k.ToString());//вывод найденного номера
```

7.8.2 Второй пример выполнения лабораторного задания

```
    insertColumn(x,m,n-1,k,1);// вставка столбца из единиц
                                //перед k-ым столбцом
    output(x,m,n,lstXAfter);   //вывод результирующей матрицы
    del(x,m);                 //вызов функции освобождение памяти
}

// Событийная процедура кнопки "Выход"
private: System::Void cmdEnd_Click(System::Object^  sender,
                                   System::EventArgs^  e)
{
    this->Close();
}
```

7.8.2 Второй пример выполнения лабораторного задания