# **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 ее элементы. 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.

При программировании задач с участием матриц для их хранения используются *двумерные массивы*. Двумерный массив, в отличие от одномерного, имеет два измерения и стандартный доступ к его элементам осуществляется с помощью двух индексов: номер строки и номер столбца. Начальное значением обоих индексов – 0. Например, двумерный массив из четырех строк с номерами от 0 до 3 и трех столбцов с номерами от 0 до 2 может быть представлен в виде матрицы следующим образом (рис.1):

|  |  |  |
| --- | --- | --- |
| **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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | j | | n-1 | | 0 |  |  |  |  | |  |  |  |  |  | | i |  | b[i][j] |  |  | |  |  |  |  |  | | m-1 |  |  |  |  | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 0 |  | j |  | n-1 | | 0 |  |  |  |  |  | |  |  |  |  |  |  | | i |  |  |  |  |  | |  |  |  |  |  |  | | n-1 |  |  |  |  |  |   Побочная диагональ Главная диагональ  матрицы j=n-i-1 матрицы i= j |

Рисунок 1 – Представление двумерного массива в виде матрицы

*Прямоугольная и квадратные матрицы*

Однако, следует иметь ввиду, что в памяти компьютера такой массив располагается в ***последовательных ячейках по строкам.***

***Количество индексов (измерений)*** указывает на ***размерность*** массива. В приведенном на рис.1 примере размерность массива **b** *–* 2**,** т.е. массив двумерный (имеет два измерения). В этом примере ***количество элементов*** массива **b** равно **12,** количество элементов по первому измерению – 4 (4 строки), а количество измерений по второму измерению – 3 (3 столбца). Однако С++ допускает использование массивов с размерностью больше 2.

В общем случае двумерные массивы используются для хранения любых типов данных.

### 7.1.2 Объявление и инициализация 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 (т.е. 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.1**  **float matr[5][10]; // объявление вещественного двумерного массива**  **// с именем matr, в котором 5 строк и**  **// в каждой строке по 10 вещественных чисел** |

Итак, двумерный массив – это способ организации данных как прямоугольной таблицы, содержащей несколько строк одинаковой длины. Таким образом, двумерный массив можно представить (как и делает компилятор), как одномерный, каждым элементом которого является строка (т.е. одномерный массив). Имя двумерного массива компилятор интерпретирует, как 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 значений в фигурных скобках. При этом каждый элемент в фигурных скобках либо является строкой двумерного массива (т.е. одномерным массивом) 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. Пример трех эквивалентных объявлений двумерного массива с инициализацией:

|  |
| --- |
| **int Arr[2][3] = {{1,2,3},{4,5,6}}; //массив из 2 строк и 3 столбцов**  **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** |

Еще раз напомним, что программист интерпретирует такой массив как прямоугольную таблицу из 2-х строк и 3-х столбцов, а на самом деле элементы такого массива располагаются в одномерной памяти ПК линейно и непрерывно (рис. 2):

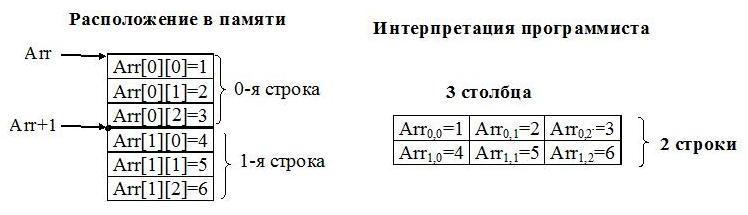
****

Рисунок 2 – Двумерный массив 2х3

## **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 **в***ыражение2* (номер столбца) – любые допустимые целочисленные выражения, в частности, константы или переменные. Для доступа к элементу двумерного массива надо указать оба его индекса. 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. 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 на рис. 2.

## **7.3 Формирование и обработка двумерных массивов**

Формирование и обработка двумерных массивов, как правило, производятся с ис­пользованием вложенных циклов с параметром, которые обеспечивают последовательный перебор всех элементов массива. В некоторых задачах может иметь значение *порядок перебора* элементов массива: *по строкам* или *по столбцам*. Если внешний цикл будет организован по первому индексу (по номеру строки), а внутренний цикл по второму индексу (по номеру столб­ца), 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.

На рис. 3 приведен пример копирования элементов описанного ранее на рис. 2 массива **Arr** (в котором 2 строки и 3 столбца) в массив **Z** по строкам и по столбцам.

|  |
| --- |
| матр копир |

Рисунок 3 – Копирование элементов двумерного массива 2х3 по строкам и по столбцам.

Выбор порядка перебора элементов массива, если он имеет значение, определяется конкретной решаемой задачей.

## **7.4 Использование автоматических двумерных массивов в качестве параметров процедур.**

При использовании двумерного массива в качестве формального параметра 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])** |

Так как двумерный массив – это массив одномерных массивов, то значение первой размерности можно не указывать. Функции неважно знать, сколько всего строк в матрице, ей важно знать, насколько велики эти строки (т.е. неважно, сколько всего одномерных массивов содержится в матрице, важен размер одномерных массивов). 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). При объявлении двумерного массива можно выделить ему память с запасом. *При вызове функции* можно указать фактические 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 и 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+i)/2.**

Схема алгоритма и программный код процедуры приведены на рис. 4. Функция имеет три параметра: входным и одновременно выходным параметром является массив **x,** которому была выделена память под три столбца. 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.

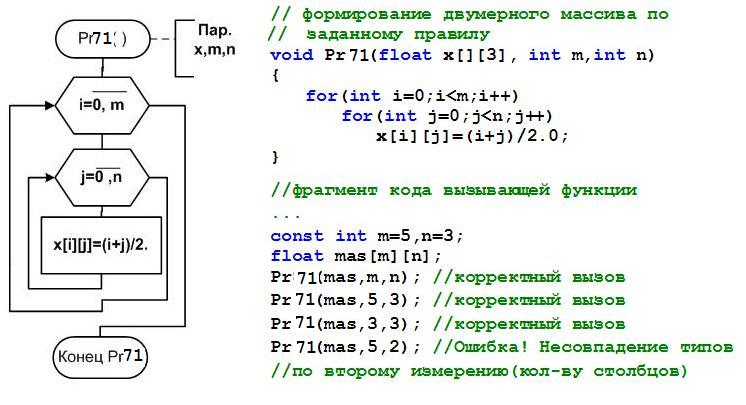


Рисунок 4 - Схема алгоритма и программный код функции формирования двумерного массива по заданному правилу для примера 7.1

Обратите внимание на примеры вызова функции **Pr71**, которые также приведены на рис. 4. 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 необходимо указать размер второго измерения (количество столбцов), то возникает соблазн (для якобы универсальности функции) 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 (рис. 5) 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. 5 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.

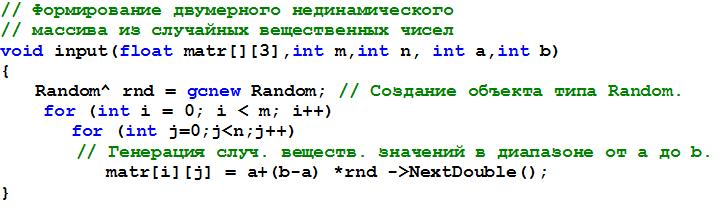


Рисунок 5 - Программный код функции формирования вещественного двумерного массива из случайных чисел диапазона **[a,b]**

**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.

На рис. 6 приведен 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.**

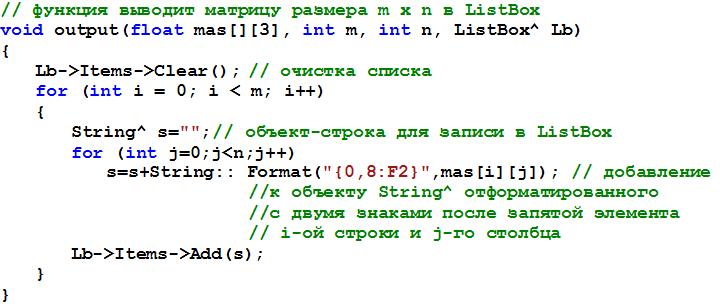


Рисунок 6 - Программный код функции вывода вещественного двумерного массива в список формы **ListBox**

**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 формирования и обработки двумерных массивов и примеры их реализации**

### 7.5.1 Типовые алгоритмы

К типовым алгоритмам формирования и обработки двумерных массивов можно отнести алгоритмы решения следующих задач:

* 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);
* перемножение двух матриц (произведением двух матриц **А** и **В**  называется матрица **С**, элемент которой, находящийся на пересечении **о**й строки и **j**-го столбца, равен сумме произведений элементов **i**-й строки матрицы **А** на соответствующие по порядку элементы **j**-го столбца матрицы **В**).

Мы рассмотрим примеры реализации некоторых из этих алгоритмов. При создании процедур, которые их реализуют, необходимо учитывать, что программный код может зависеть от типа памяти, которую занимает двумерный массив.

### 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.**

Схема алгоритма и программный код процедуры приведены на рис. 7. Входными параметрами функции являются двумерный массив **x**, а также переменные, указывающие число строк и столбцов массива. Результат работы процедуры формируется в возвращаемой переменной **p**. Перед началом вложенных циклов значение этой переменной устанавливается в **1**. В теле внутреннего цикла проверяется знак очередного элемента массива, и, если он отрицателен, то **p** домножается на этот элемент. По окончании внешнего цикла в переменной **p** накопится искомое произведение.

|  |  |
| --- | --- |
| матр пр74 | матр прим74пр |

Рисунок 7 – Схема алгоритма и программный код функции, вычисляющей произведение отрицательных элементов вещественного двумерного массива примера 7.4

**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.**

Схема алгоритма и программный код процедуры приведены на рис.8. 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**.

После кода функции **Pr75** приведен фрагмент кода событийной процедуры, иллюстрирующий правила и возможности работы с автоматическими двумерными массивами.

Как уже многократно повторялось, память под любой нединамический массив выделяется на этапе компиляции. 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».

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).

|  |  |
| --- | --- |
|  | матр прим75.jpg |

Рисунок 8 – Схема алгоритма и программный код функции примера 7.5

и фрагмент кода событийной процедуры

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.

Как уже было сказано, двумерный массив – это одномерный массив, состоящий из одномерных же массивов. Имя двумерного массива компилятор интерпретирует, как константный указатель на нулевую строку. Поэтому для выделения памяти под двумерный массив можно 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. На рис. 9 показан способ создания целочисленного динамического двумерного массива.

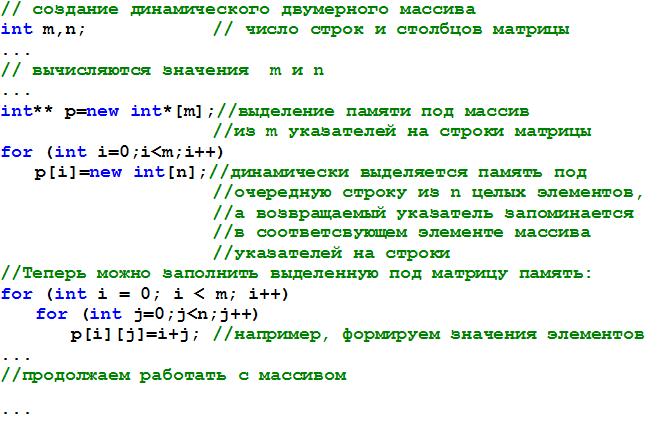


Рисунок 9 – Создание динамического двумерного массива

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 **р** типа «указатель на указатель», выделяется память под массив указателей на строки матрицы (количество строк - **m**), и адрес начала этого массива указателей присваивается созданной переменной **р**.

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 (рис. 10) со всеми, 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.

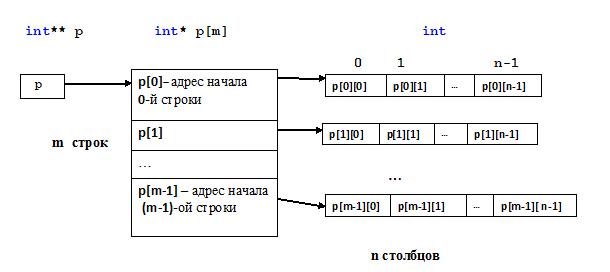


Рисунок 10– Выделение памяти под динамический двумерный массив

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),** ноадрес 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 (рис.11), а затем освободить захваченную динамическую память:

|  |
| --- |
| матр динам добав строки  **матр динам освоб** |

Рисунок 11– Добавление строки в динамический двумерный массив и освобождение памяти, выделенной динамическому массиву

*Обратите внимание*, что для освобождения памяти, выделенной под матрицу, недостаточно одного оператора **delete[] p;** так как такой оператор удалит массив указателей, но не объекты (строки массива), на которые они указывают. Для полного освобождения памяти сначала нужно в цикле удалить сами строки, а уже потом массив указателей, как показано на рисунке 11.

## **7.7 Типовые процедуры формирования и вывода динамических двумерных массивов**

При использовании двумерного динамического массива 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].**

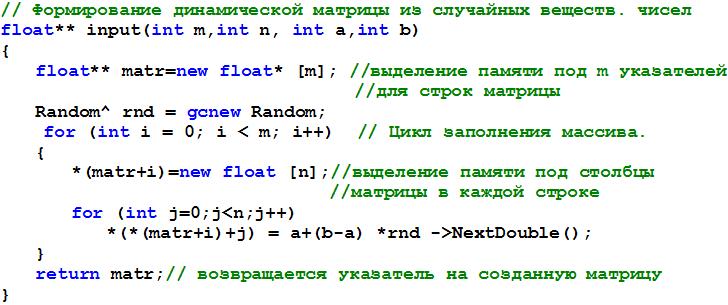
Программный код функции формирования матрицы из случайных чисел и фрагмент событийной процедуры, которая вызывает разработанную функцию, представлен на рис. 12. Функция имеет 4 входных целочисленных параметра: число строк **m**, число столбцов **n**, и **a, b** – границы диапазона случайных чисел и возвращаемое значение типа указатель на указатель на **float**.

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 матрицы.

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

****

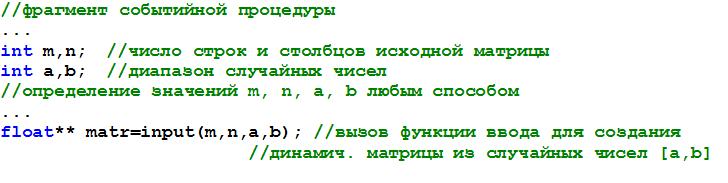
****

Рисунок 12 - Программный код функции формирования **динамического** вещественного двумерного массива из случайных чисел диапазона **[a,b]**

**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.

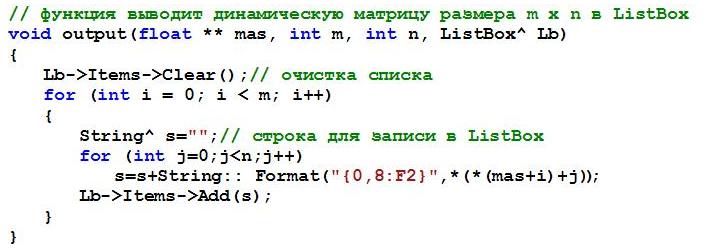
****

Рисунок 13 - Программный код функции вывода динамического вещественного двумерного массива в список **ListBox** на форме

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.

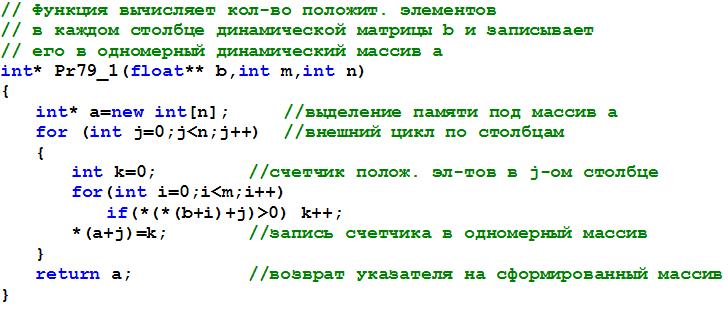


Рисунок 14 - Программный код функции создания динамического целочисленного массива, в который записываются количества положительных элементов каждого столбца динамического двумерного массива

**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 (рис. 15) и, соответственно, память под одномерный массив выделять по числу строк, а не столбцов двумерного массива.

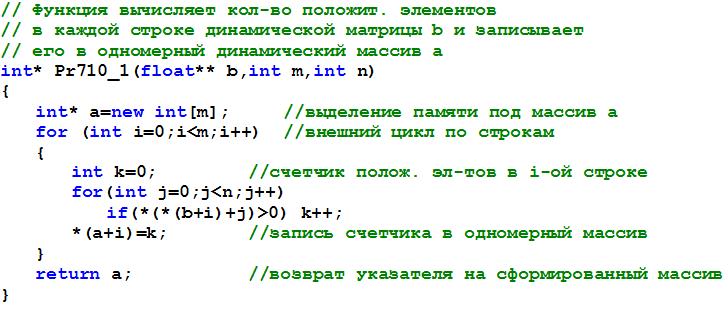


Рисунок 15 - Программный код функции создания динамического целочисленного массива, в который записываются количества положительных элементов каждой строки динамического двумерного массива (1 способ)

А теперь решим эту же задачу по–другому, имея в виду, что матрица – это массив одномерных массивов. Поэтому можно подсчет числа положительных элементов каждой строки представить, как подсчет числа положительных элементов одномерного массива. Создадим дополнительную функцию, которая подсчитывает число положительных элементов в одномерном массиве (рис. 16)

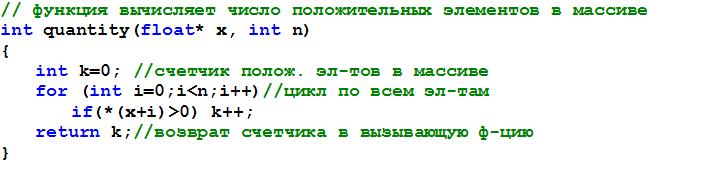


Рисунок 16 - Программный код функции вычисления положительных элементов в одномерном массиве вещественных чисел.

Теперь 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 (рис. 17).

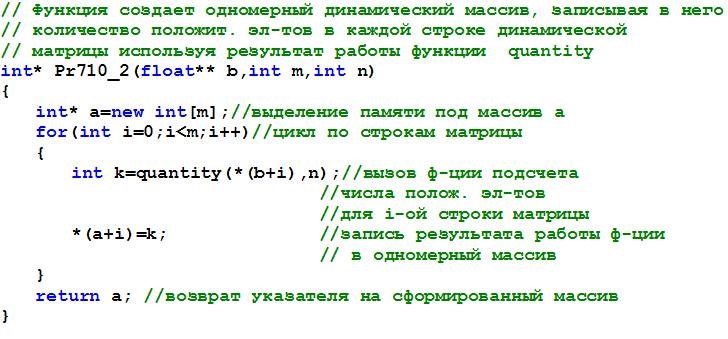


Рисунок 17 - Программный код функции создания динамического целочисленного массива, в который записываются количества положительных элементов каждой строки динамического двумерного массива (2 способ) с использованием дополнительной функции

Заметим, что решить подобным образом задачу примера 7.9, где требуется рассматривать не строки, а столбцы матрицы, нельзя.

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

Примечание: понятие главной диагонали определено для массивов с одинаковым количеством строк и столбцов (квадратных матриц); для всех элементов главной диагонали выполняется условие **i=j**, где **i** – номер строки, а **j** – номер столбца, как было показано на рис.1.

|  |  |
| --- | --- |
|  | матр дин пр 711 |

Рисунок 18 – Схема алгоритма и программный код функции

примера 7.11

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.

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

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**.

Схема алгоритма и программный код процедуры приведены на рис.19. 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).

|  |  |
| --- | --- |
|  | матр трансп |

Рисунок 19 – Схема алгоритма и программный код процедуры**Pr712**

Примера 7.12

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.

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

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).**

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



Таким образом, произведение двух матриц можно записать следующим образом:



Схема алгоритма и программный код процедуры приведены на рис.20.

|  |  |
| --- | --- |
|  | матр произвед |

Рисунок 20 – Схема алгоритма и программный код процедуры**Pr713,** вычисляющей произведение двух матриц Примера 7.13

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.

## **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.
2. **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.

Для программной реализации задания создадим следующие процедуры (некоторые из них мы уже создали в примере выполнения предыдущей лабораторной работы для динамических одномерных массивов):

* функцию **GetInt** ввода целого числа с контролем непустого текстового поля и положительного значения – для ввода размеров исходного массива и границ диапазона натуральных случайных чисел – уже была создана;
* функцию **input** для формирования двумерного массива из случайных натуральных чисел;
* функцию **output** для вывода двумерного массива целых чисел в заданный список (элемент управления **ListBox**) формы;
* функцию **simple,** которая проверяет, является ли ее формальный параметр простым числом – уже создана;
* функцию **analys**, которая анализирует, есть ли в одной строке матрицы хотя бы одно простое число; так как одну строку матрицыможно рассматривать как одномерный массив, то можно использовать уже созданную для предыдущей лабораторной работы функцию проверки наличия хотя бы одного простого числа в одномерном массиве;
* функцию **analysMatr,** которая считает, сколько строк в матрице 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,** котораяформирует новый массив из тех строк исходного массива, в которых есть хотя бы одно простое число;
* функцию **del,** которая освобождает память, выделенную для динамической матрицы.

Возможный вид формы проекта для организации интерфейса пользователя приведен на рисунке 21. В верхней части формы расположены текстовые поля для ввода исходных данных: количества строк массива (**txtM)** и количества столбцов массива (**txtN)** и границ диапазона случайных чисел(**txtA)** и(**txtB).** Ниже расположены два списка **ListBox:** список **lstMatr1** для вывода исходного массива и список **lstMatr2** – для вывода нового массива. На форме также имеются две кнопки: кнопка **Решить (cmdTask)** для запуска программы на выполнение, и кнопка **Выход (cmdExit)** для останова ее выполнения.

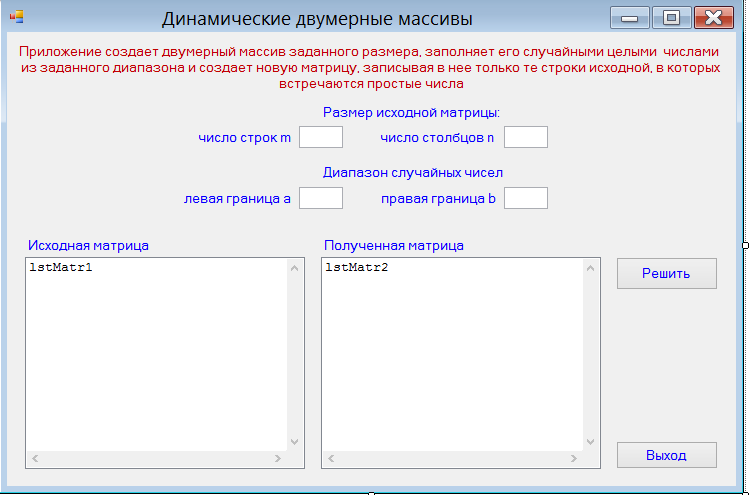


Рисунок 21 – Графический интерфейс проекта

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 рисунке 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.

|  |
| --- |
| лаб7 ввод1  ввод целого с контр  лаб7 вывод |

Рисунок 22 - Программный код файла **GetPut.cpp**.

Основные функции для решения задачи записаны в файле **task.cpp**. Программный код файла приведен на рисунке 23.

|  |
| --- |
| лаб7 task1  лаб7 task2  лаб7 task3 |

Рисунок 23 - Программный код файла **task.cpp**

Первая функция **simple** проверяет, является ли натуральное число простым числом, уже рассмотрена в предыдущих лабораторных работах.

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.

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.

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 массивиз **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**.

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**

|  |
| --- |
| лаб7 заг |

Рисунок 24 – Программные коды заголовочных файлов

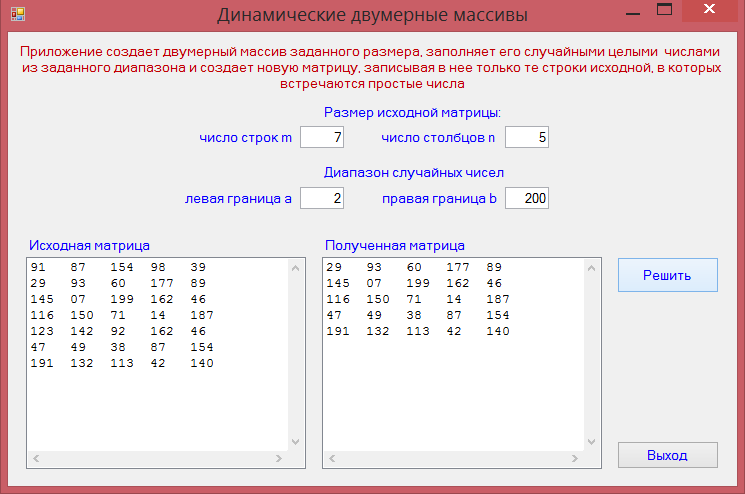
И, наконец, создается программный код событийных процедур. На рисунке 25 приведен код этих процедур, вставляемых в файл **Form1.h**после директивы **#pragma endregion**.

|  |
| --- |
| лаб7 соб1  лаб7 соб2  лаб7соб выход |

Рисунок 25 – Программный код событийных процедур в файле **Form1.h**

В событийной процедуре нажатия на кнопку **Решить** (**cmdTask\_Click**) объявляются четыре переменные, которые с помощью функции ввода **GetInt** с контролем непустого текстового поля получают значения числа строк **m** и столбцов массива **n** и **a, b** - границы диапазона случайных натуральных чисел. Затем вызывается функция **input** создания динамического двумерного массива и заполнения его случайными натуральными числами. После вывода исходного массива функцией **output** в предназначенный список на форме, вызывается функция **task** создания нового массива согласно условию задачи. Если результат работы функции **task** не равен нулю, то новый массив создан, и полученный массив выводится на форму функцией **output**, а затем вызывается функция **del** освобождения динамической памяти, которая выделялась новому массиву. Если же функция **task** вернула ноль, то новый массив не создан и выводится соответствующее сообщение. Последний оператор событийной процедуры – оператор вызова функции **del** для освобождения динамической памяти, которая выделялась исходному массиву.

На рис. 26 приведены результаты выполнения программы при различных исходных данных. Они подтверждают правильную работу используемых алгоритмов и соответствующих программ.



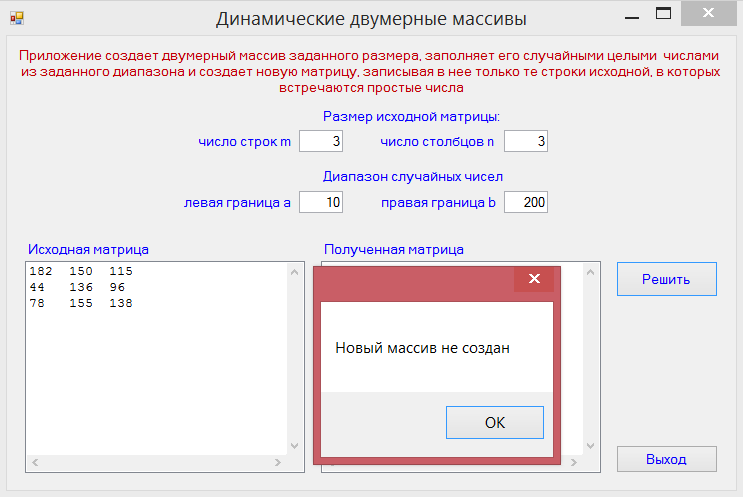


Рисунок 26 – Результаты выполнения проекта

### 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.

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.

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.

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.

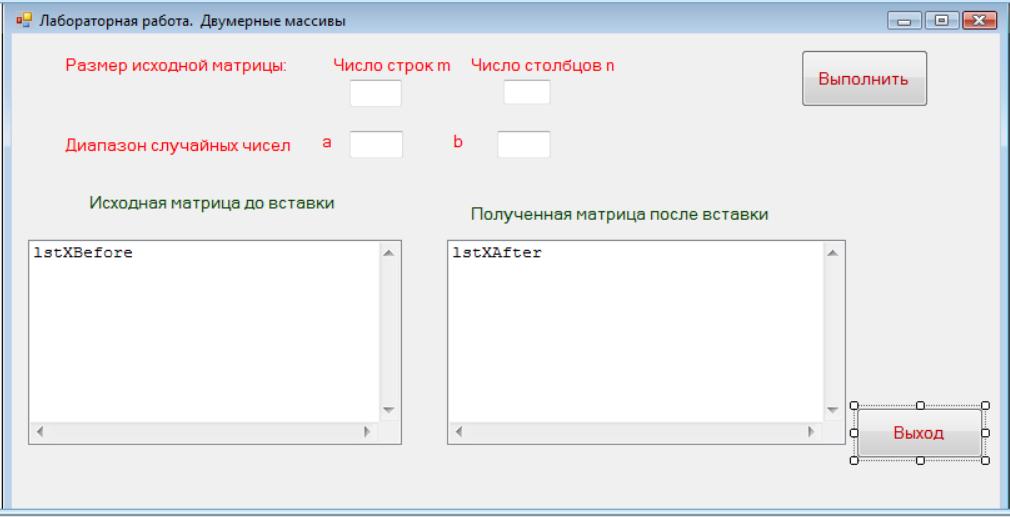


Рисунок 27 – Графический интерфейс проекта

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.

|  |
| --- |
| ввод матр  ввод целого с контр  ввод вещест с контр  вывод матр  матр освоб памяти |

Рисунок 28 - Программный код файла **GetPut.cpp**.

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.

Во вложенных циклах перебираются все элементы массива. Если значение очередного элемента, меньше текущего наименьшего **MinItem**, то это значение присваивается **MinItem**, а в переменной **k** фиксируется номер столбца этого элемента. По окончании внешнего цикла в переменной **k** будет находиться искомый номер столбца с наименьшим значением среди всех элементов массива. В коде процедуры – это значение возвращается оператором **return**.

|  |  |
| --- | --- |
| схема матр поисх мин | матр поиск мин прогр |

Рисунок 29 - Схема алгоритма и программный код функции **GetMinCol**

Процедура **InsertColumn** имеет 5 параметров. Входными параметрами процедуры являются: число строк **m** и число столбцов **n** двумерного массива, номер столбца **InsCol,** перед которым необходимо вставить новый столбец, и значение **InsVal**, которое необходимо присвоить всем элементам вставленного столбца. Двумерный массив **Arr** передается по указателю и является одновременно входным и выходным параметром процедуры.

В начале процедуры организуются вложенные циклы для сдвига столбцов массива, начиная со столбца с номером **InsCol**, вправо, чтобы освободить место для нового столбца. Внешний цикл организован по строкам массива. Во внутреннем цикле параметр **j** изменяется от **n-1** до **InsCol** с шагом **–1**, и следующему элементу в текущей строке массива присваивается соседнее слева значение (другими словами, текущий элемент присваивается соседнему справа).

После того, как процедура сдвига столбцов завершена, организуется цикл по строкам массива, в котором каждому элементу столбца с номером **InsCol** присваивается заданное значение **InsVal**.

|  |  |
| --- | --- |
| **схема матр вставка столбца** | **матр вставка столбца** |

Рисунок 30 - Схема алгоритма и программный код функции **InsertColumn**

После того, как файлы исходного кода**GetPut.cpp**и **task.cpp** 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\_din.h*.*** На рисунке 31 приведен программный код созданного заголовочного файла **task.h** и системного заголовочного файла **stdafx.h**

|  |
| --- |
| матр заг |

Рисунок 31 – Программные коды заголовочных файлов

И, наконец, создается программный код событийных процедур. На рисунке 32 приведен код этих процедур, вставляемых в файл **Form1.h**после директивы **#pragma endregion**.

|  |
| --- |
| матр соб1  матр соб2 |

Рисунок 32 – Программный код событийных процедур в файле **Form1.h**

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.

На рисунке 33 приведены результаты выполнения программы. Они подтверждают правильную работу используемых алгоритмов и соответствующих программ.

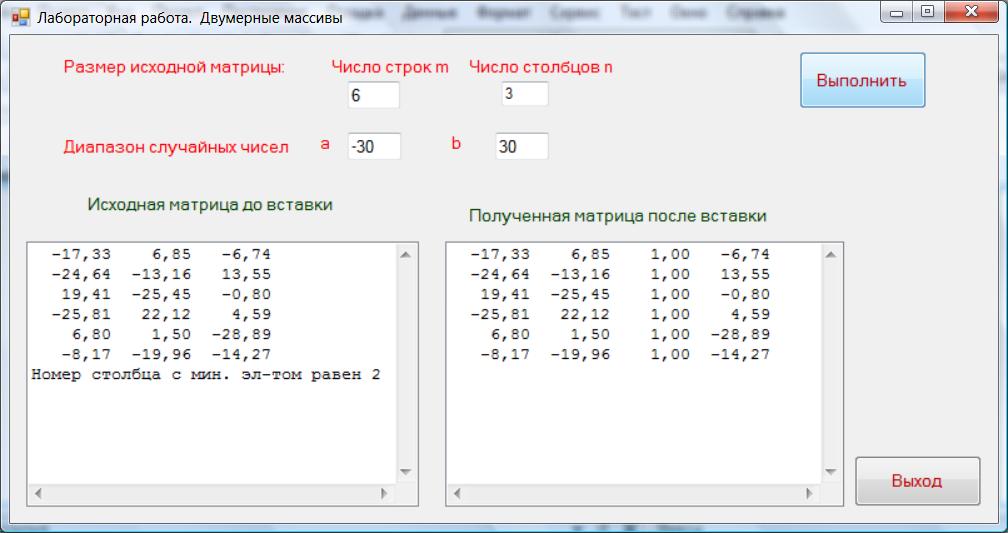


Рисунок 33 – Результаты выполнения проекта