Оконное приложение C++

Матрица

Преподаватель:

Готовская Светлана Борисовна

Теория (вспомним)

Двумерный массив (матрица) — это совокупность пронумерованных однотипных данных; прямоугольная таблица, состоящая из N строк и M столбцов.

	столбец 0	столбец 1	столбец 2	столбец 3
строка 0	arr[0][0]	arr[0][1]	arr[0][2]	arr[0][3]
строка 1	arr[1][0]	arr[1][1]	arr[1][2]	arr[1][3]
строка 2	arr[2][0]	arr[2][1]	arr[2][2]	arr[2][3]

Матрица характеризуется именем, размерностью и размером.

Имя матрицы **образуется по общему правилу образования имен, т. е. представляет собой идентификатор, например, A, Bl, C_8 и т. д.**

Размерность матрицы **определяется числом индексов (у матрицы два индекса: первый показывает номер строки, а второй номер столбца, где элемент находится).**

Теория (вспомним)

Двумерный массив (матрица) — это совокупность пронумерованных однотипных данных; прямоугольная таблица, состоящая из N строк и M столбцов.

	столбец 0	столбец 1	столбец 2	столбец 3
строка 0	arr[0][0]	arr[0][1]	arr[0][2]	arr[0][3]
строка 1	arr[1][0]	arr[1][1]	arr[1][2]	arr[1][3]
строка 2	arr[2][0]	arr[2][1]	arr[2][2]	arr[2][3]

Матрица характеризуется именем, размерностью и размером.

Двухмерный массив (*матрица*) представляет собой список одномерных массивов.

Общая форма записи двухмерного массива:

тип имя_массива[размер1] [размер2];

В приведенной записи *размер1* означает количество строк двухмерного массива, а *размер2* - количество столбцов.

Для двухмерных массивов общий размер массива в байтах вычисляется по формуле:
всего байт = число строк * число столбцов * размер типа в байтах.
Для определения размера типа в байтах применяется функция sizeof()

	столбец 0	столбец 1	столбец 2	столбец 3
строка 0	arr[0][0]	arr[0][1]	arr[0][2]	arr[0][3]
строка 1	arr[1][0]	arr[1][1]	arr[1][2]	arr[1][3]
строка 2	arr[2][0]	arr[2][1]	arr[2][2]	arr[2][3]

j

i	0	1	2	3
0	4	7	5	6
1	1	8	4	7
2	2	5	9	1

a – имя матрицы

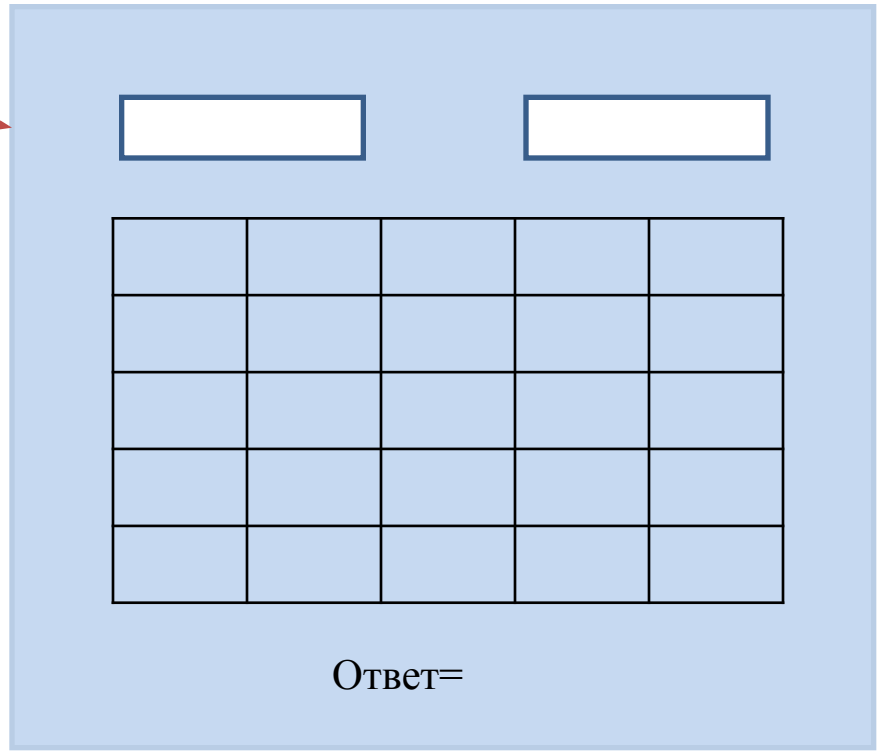
a[i][j] – элемент матрицы

i – индекс строки

j – индекс столбца

int a[3][4] – описание матрицы

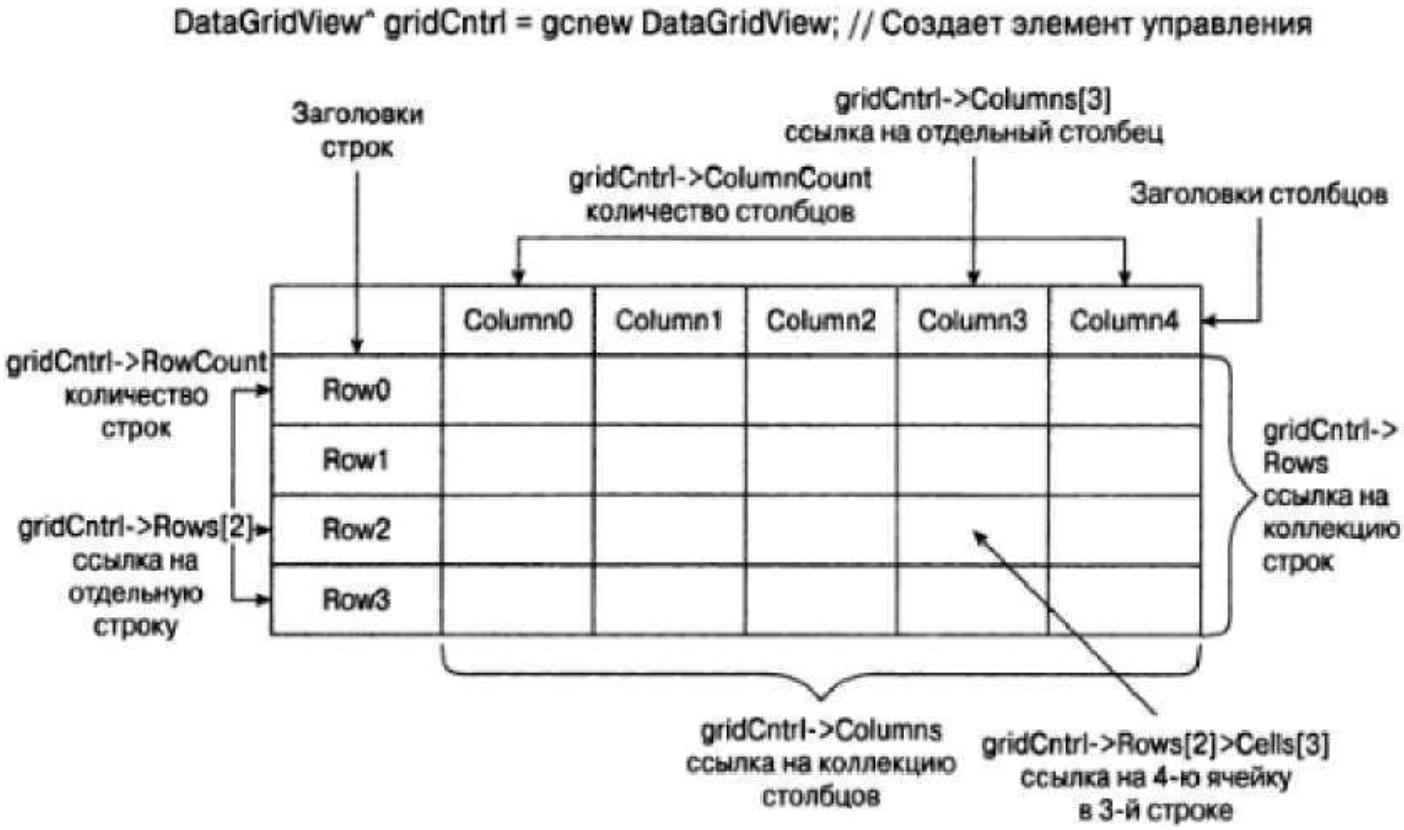
```
int main()
{
   setlocale(LC_ALL, "rus");
   int a[100][100];
   int N,M;
   cin >> N >> M;
    // Помещаем в массив значения
   for (int i = 0; i < N; i++)
   for (int j = 0; j < M; j++)
     {
       cin >> a[i][j];
      }
   // Выводим значения массива
   cout << "Вывод массива" << endl;
   for (int i = 0; i < N; i++)
   {
   for (int j = 0; j < M; j++)
        cout << a[i][j] << “\t";
   }
 // Решаем задачу
int s=0;
for (int i = 0; i < N; i++)
    for (int j = 0; j < M; j++)
    {
       s=+ a[i][j] ;
     }
cout << s;
system("pause");
return 0;
}
```



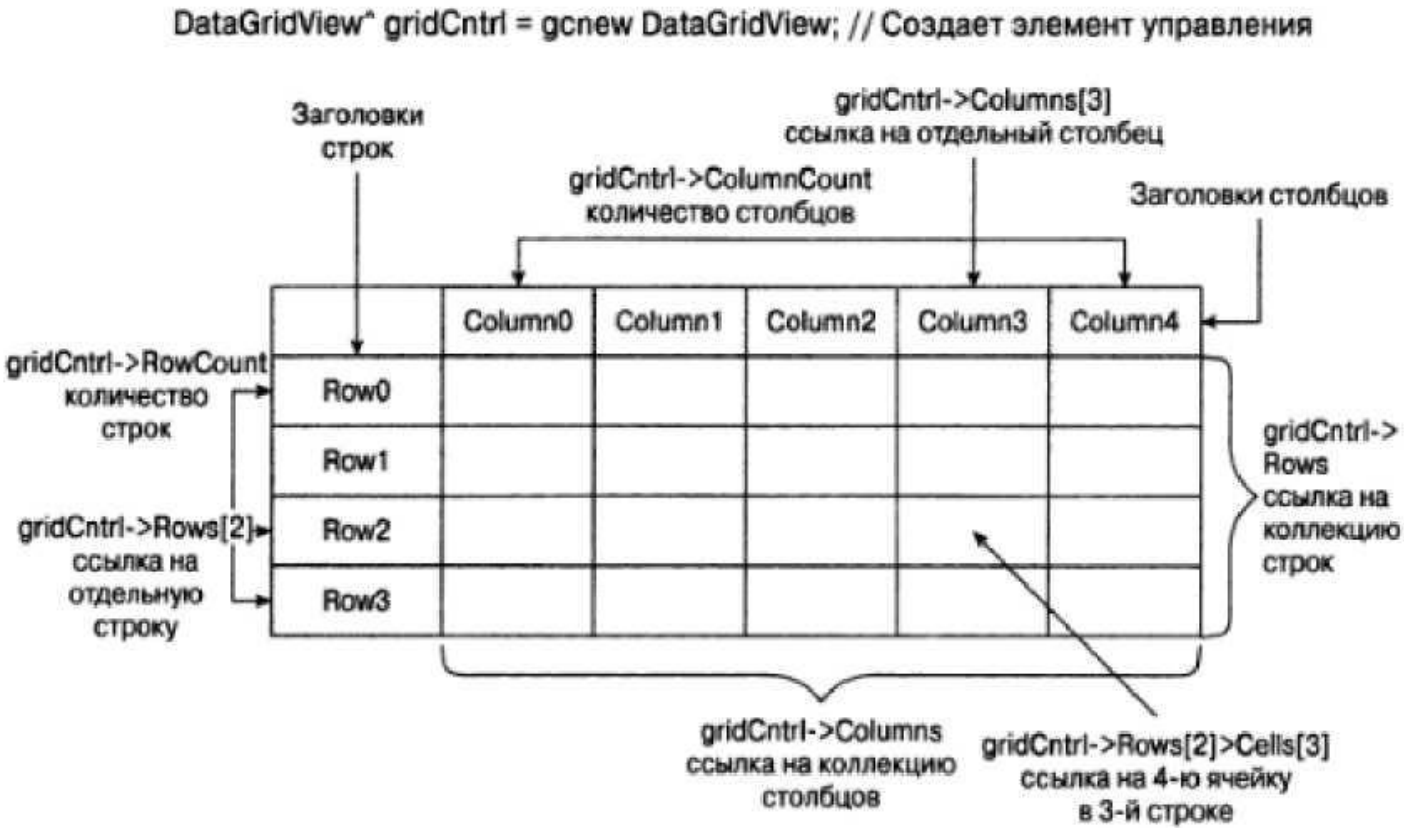
Использование элемента управления DataGridView для работы с матрицами

DataGridView - это элемент управления, который может отображать буквально любой вид данных в ячейках прямоугольной сетки.

Элемент управления DataGridView позволяет отображать и изменять прямоугольный массив данных из множества различных источников данных. Его можно использовать также для отображения практически любых данных, созданных непосредственно в программе. По своей сущности это сложный элемент управления, который обеспечивает огромную гибкость при его применении, и преимуществами множества его функциональных возможностей можно воспользоваться через множество свойств, функций и событий.



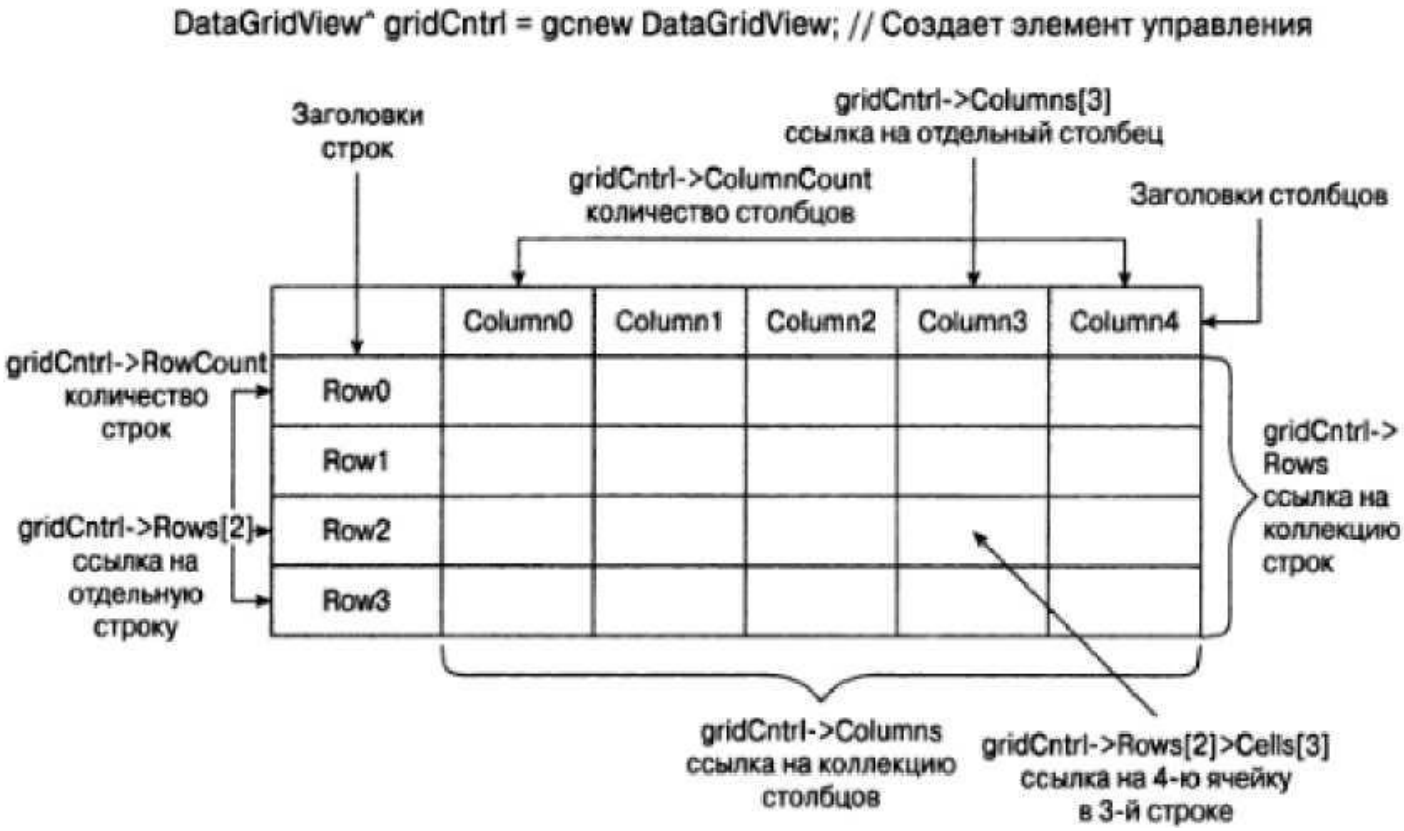
Данные элемента управления DataGridView отображаются в прямо угольном массиве ячеек, которые можно представить в виде коллекции строк или столбцов. *Каждый столбец ячеек имеет в верхней части ячейку заголовка*, которая, как правило, содержит идентифицирующий этот столбец текст, а *в начале каждой строки расположена ячейка заголовка строки*, как показано на рисунке.



Обращение к строкам и столбцам ячеек выполняется через свойства объекта DataGridView.

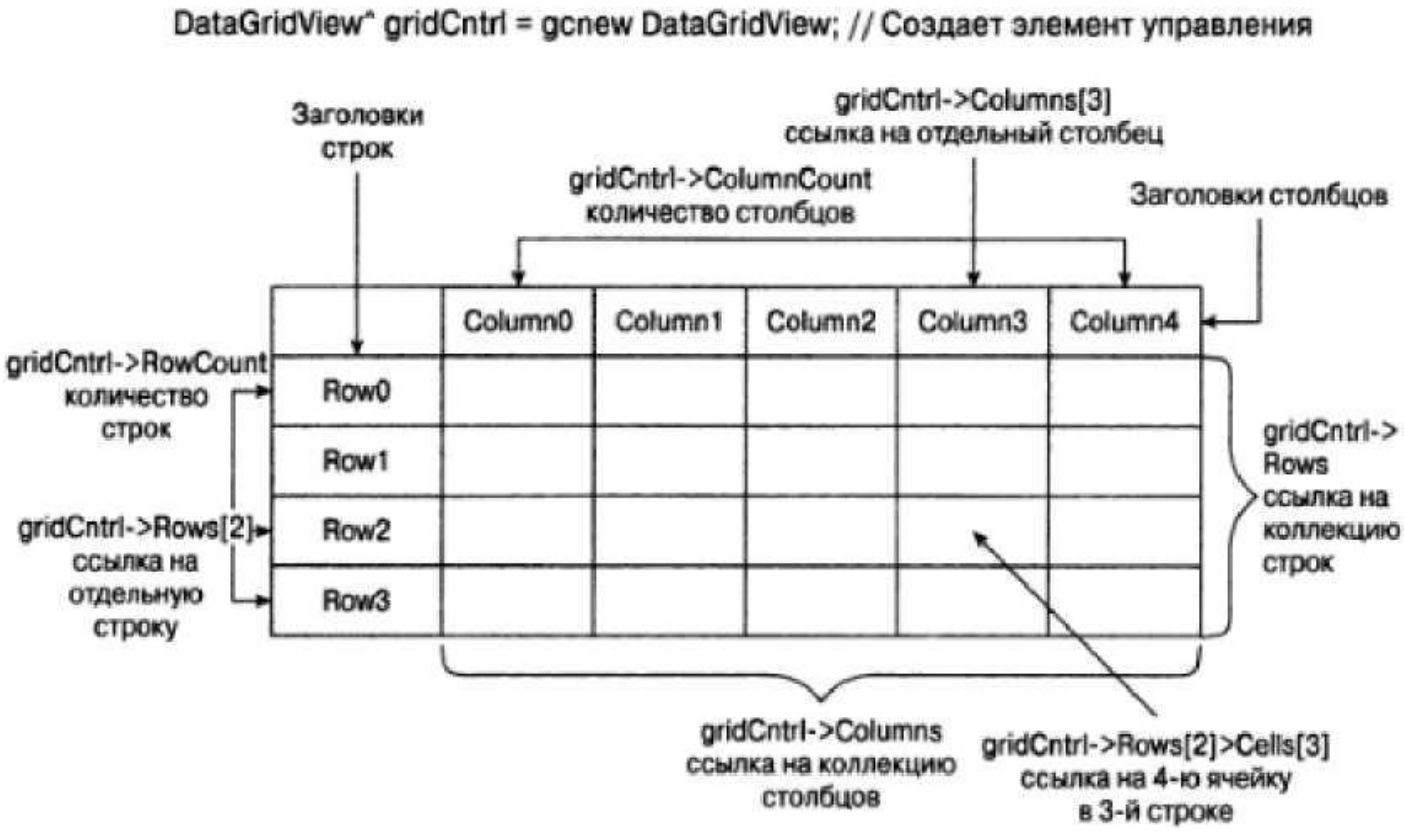
Свойство **Rows** возвращает значение типа DataGridRowCollection, представляющее коллекцию всех строк, а обращение к конкретной строке выполняется с помощью индекса, что можно видеть на рисунке.

Аналогично свойство **Columns** элемента управления возвращает значение типа DataGridviewColumnCollection, которое также можно индексировать для обращения к конкретному столбцу.



Индексация строк и столбцов осуществляется, начиная с нуля.

Свойство **Cells** объекта DataGridRowCollection представляет коллекцию, содержащую ячейки строки, и это свойство можно индексировать для получения доступа к конкретной ячейке в строке. Пример ссылки на четвертую ячейку в третьей строке приведен на рисунке.



Количество строк доступно как значение свойства **RowCount**, а свойство **ColumnCount** возвращает количество столбцов.

Вначале, когда элемент управления еще не связан с источником данных, он не будет содержать ни строк, ни столбцов. Количество столбцов и / или строк можно определить, устанавливая значения свойств элемента управления, но при его использовании для отображения данных из источника данных это действие выполняется автоматически.

Персональная настройка элемента управления DataGridView

(свойство элемента)

Свойства объекта **DataGridViewCellStyle**, влияющие на внешний вид ячейки

Свойство	Описание
BackColor	Это значение - объект System::Drawing::Color. который определяет цвет фона ячейки. В классе Color диапазон стандартных цветов определен в виде статических членов. Значение, используемое по умолчанию — Color:: Empty
ForeColor	Это значение - объект color, который определяет цвет изображения ячейки. Значение, используемое по умолчанию - Color::Empty
SelectionBackColor	Это значение - объект color, который определяет цвет фона выбранной ячейки. Значение, используемое по умолчанию - Color::Empty
SelectlonForeColor	Это значение - объект color, который определяет цвет изображения выбранной ячейки. Значение, используемое по умолчанию - Color :: Empty
Font	Это значение - объект System::Drawing::Font, определяющий шрифт, который должен использоваться для отображения текста е ячейке. Значение, используемое по умолчанию - null
Alignment	Это значение определяет выравнивание содержимого ячейки. Допустимые значения определены перечислением DataGridViewAlignnent, поэтому значение может быть любой из следующих констант; Bottomcenter, Bot- tomLeft, BottomRight, MiddleCenter, Middle- Left, MiddleRight, TopCenter, TopLeft, To- pRight. NotSet. Значение, используемое по умолчанию - NotSet
WrapMode	Это значение определяет переход текста в ячейке в следующую строку, если он слишком длинен, чтобы уместиться в ячейке. Допустимое значение - одна из констант, определенных перечислимым объектом DataGridViewTri- State: True, False, NotSet. Значение, используемое по умолчанию - NotSet
Padding	Это значение - Объект типа System::Windows:: Forms::Padding, который определяет пробел между со держимым и краем ячейки. Конструктор класса Padding требует передачи аргумента типа int, который представляет значение дополнения содержимого ячейки пробелами, измеренное в пикселях. Значение, используемое по умолчанию, соответствует отсутствию дополнения содержимого ячейки
Format	Это значение - строка формата, которая определяет способ форматирования содержимого строки. Это форматирование аналогично используемому в функции Console:: WriteLine(). Значение, используемое по умолчанию, - пустая строка

Персональная настройка элемента управления DataGridView

Нам необходимо, чтобы элемент управления размещался в фиксированной позиции в клиентской области формы. Это можно осуществить, устанавливая значение свойства **Dock**:

dataGridView->Dock = DockStyle::Fill;

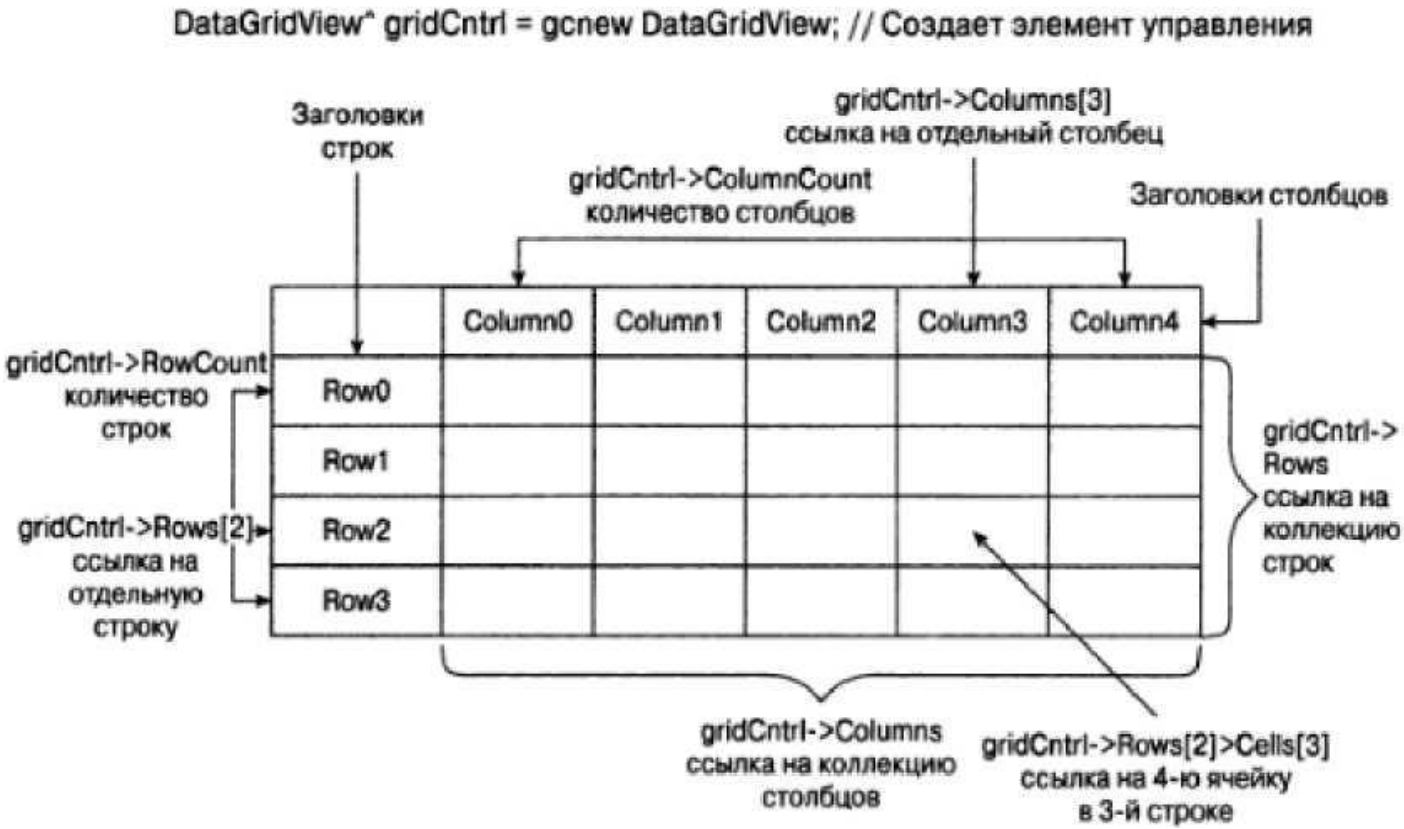
В качестве значения свойства Dock должна быть установлена одна из констант, определенных перечислением DockStyle. Другими допустимыми значениями этого свойства являются Top (Вверху), Bottom (Внизу), Left (Слева), Right (Справа) и None (Нет), которые указывают стороны элемента управления, привязанные к позиции размещения.

Привязку позиции элемента управления к клиентской области формы можно выполнять также посредством установки свойства **Anchor** элемента управления. Значение этого свойства указывает края элемента управления, которые должны быть привязаны к клиентской области формы. Значение является побитовой комбинацией констант, определенных перечислением AnchorStyles, и может принимать любые или все из значений Top, Bottom, Left и Right. Например, чтобы привязать верхнюю и левую стороны элемента управления, в качестве значения нужно было бы указать AnchorStyles ::Top & AnchorStyles::Left.

Определение свойства Anchor ведет к фиксации позиции элемента управления и его линеек прокрутки внутри контейнера заданного размера. Поэтому при изменении размера окна приложения элемент управления и его линейки прокрутки сохраняют свои размеры. Если установить значение свойства Dock так, как в предыдущем операторе, при изменении размера окна приложения отображаемая в нем часть элемента управления будет изменяться с соответствующим изменением линеек прокрутки. Поэтому теперь работать с приложением значительно удобнее.

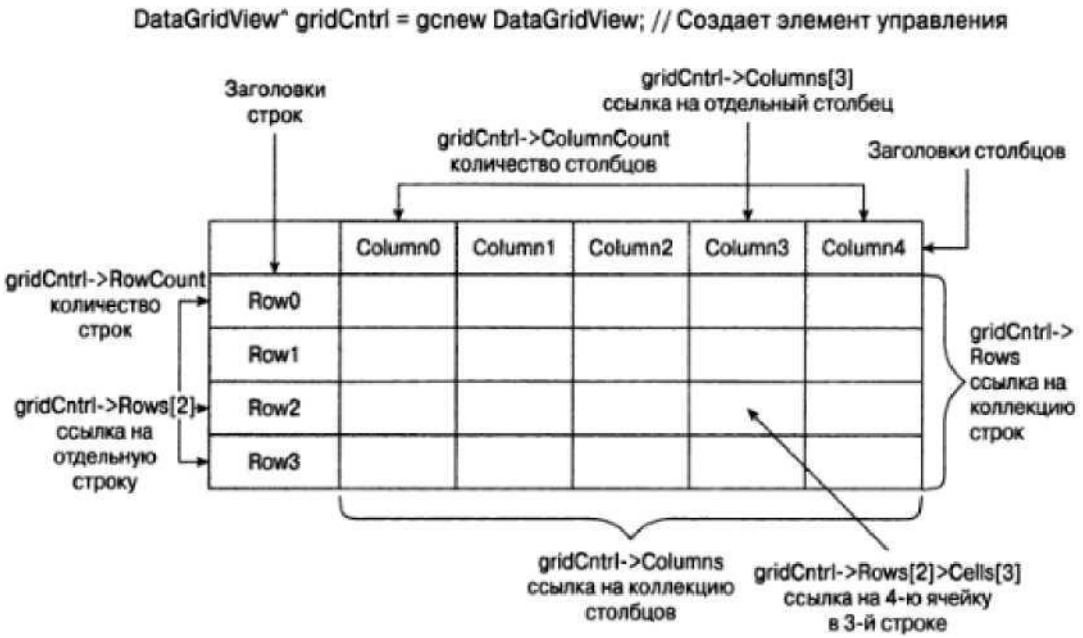
Нам требуется, чтобы ширина, столбцов автоматически изменялась, обеспечивая отображение полных строк данных в ячейках. Для этого можно вызывать функцию AutoResizeColumns (): dataGridview->AutoResizeColumns();

Этот оператор подбирает ширину всех столбцов в соответствии с текущим содержимым, в том числе в соответствии с содержимым ячеек заголовков. Обратите внимание, что эта настройка выполняется во время вызова функции, поэтому к моменту её вызова содержимое уже должно существовать. При последующем изменении содержимого ширина столбца не изменяется. Если требуется, чтобы ширина столбца автоматически изменялась при каждом изменении содержимого ячеек, для элемента управления



Элемент управления DataGridView применяется в трех различных режимах. Мы будет применять *«несвязанный режим».* В несвязанном режиме передача данных элементу управления выполняется вручную, как правило, с помощью функции Add () применительно к свойству Rows элемента управления. Этот режим следует использовать для отображения сравнительно небольших объемов данных.

В несвязанном режиме элемент управления DataGridView можно использовать для отображения в приложении любых данных, которые могут быть отображены в табличном виде. Потому этот инструмент очень удобен для отображения данных во множестве разнообразных приложений.



Следующие операторы создают элемент управления, обращение к которому выполняется посредством дескриптора DataGridView, а затем устанавливают количество столбцов, равное 3:

```
DataGridView^ dataGridView = dcnew DataGridView;  //создает элемент управления
DataGridView->ColumnCount = 3;                    // Устанавливает количество столбцов.
```

При желании столбцы в элементе управления можно пометить путем установки свойства Name (это имя элемента управления в коде!!!) для каждого столбца, указывая заголовки, идентифицирующие данные в каждом из них. Это можно выполнить так:

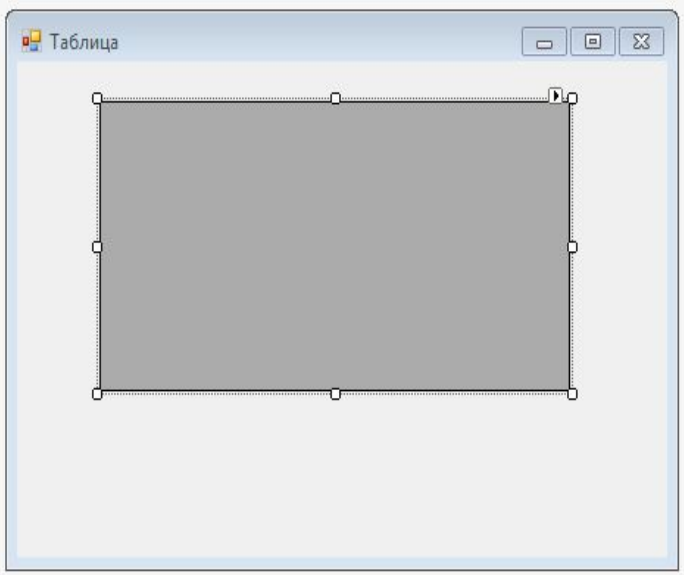
```
dataGridView->Columns[0]->Name="Name";
dataGrldView->Columns[l]->Name="Phone Number";
dataGridView->Columns[2]->Name="Address";
```

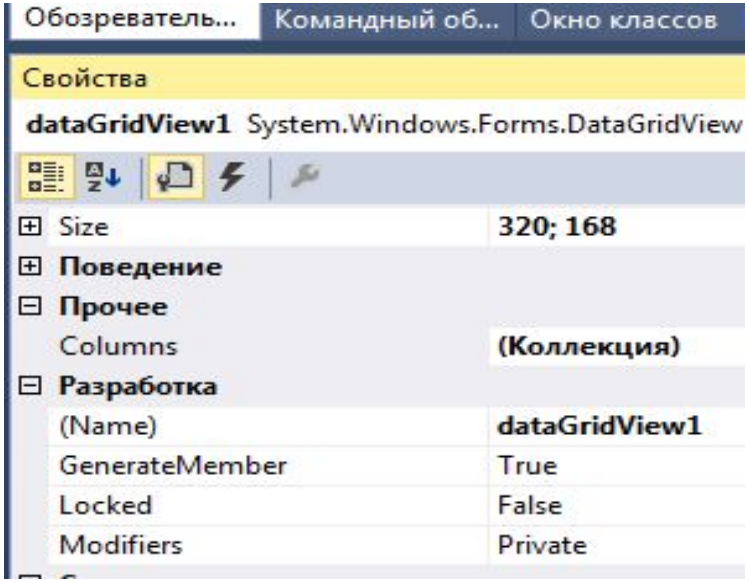
Свойства для добавления и удаления строк

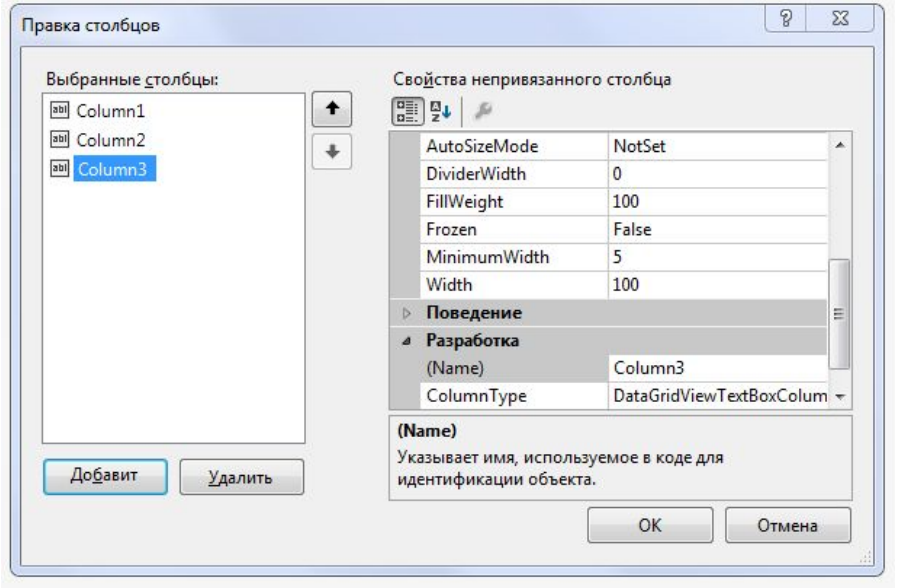
Функции	Описание
Add ()	Добавляет в коллекцию одну или более строк
Insert ()	Вставляет в коллекцию одну или более строк
Clear ()	Удаляет все строки коллекции
AddCopy()	Добавляет копию строки, указанную в аргументе
InsertCopy()	Вставляет копию строки, указанную первым аргументом, в позицию, которая задана вторым аргументом
Remove ()	Удаляет строку, указанную аргументом типа DataGridviewRow.
RemoveAt ()	Удаляет строку, указанную значением индекса, которое передано в качестве аргумента.

Задание 1

1) Создайте новый проект. Добавьте на форму визуальную компоненту DataGridView.
2) Запустите приложение
3) Выберите свойство Columns и нажмите на кнопку с тремя точками, расположенную справа. В окне «Правка столбцов» выбрать «Добавить», столбец (column). Таким образом добавьте три столбца.



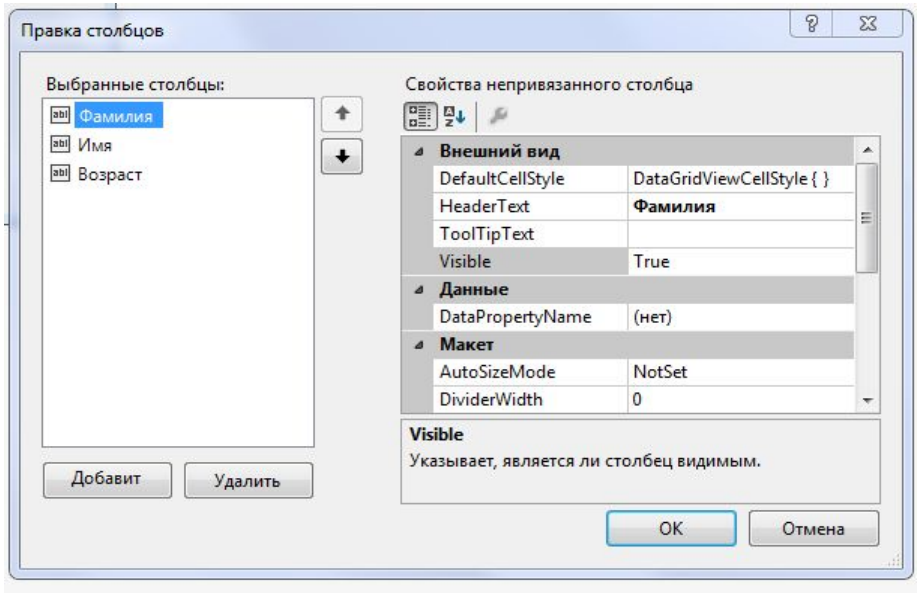


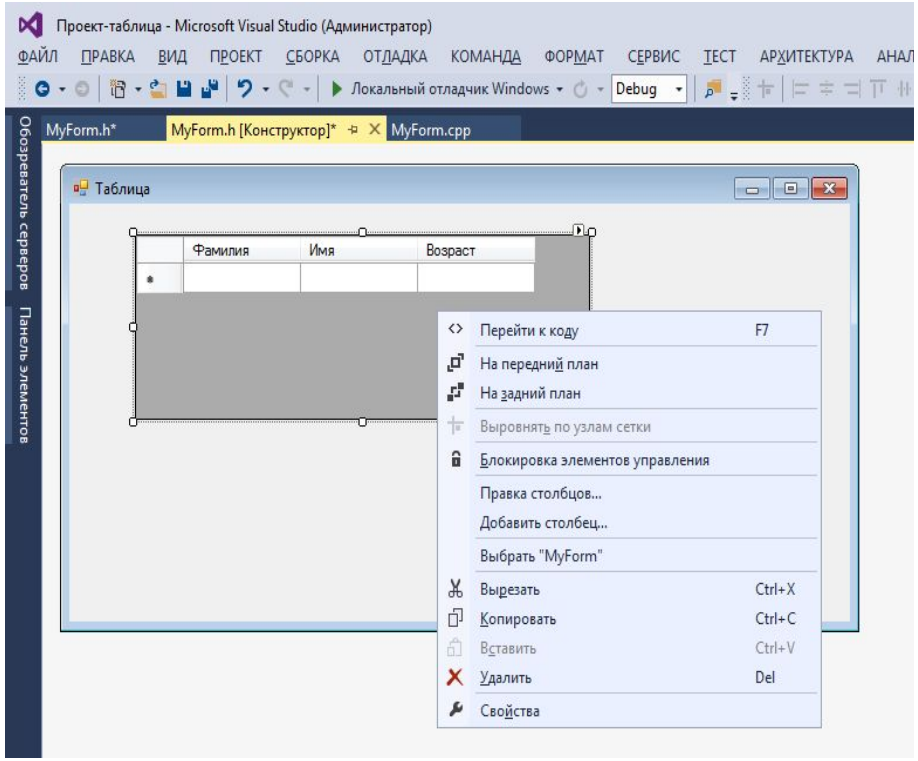


Задание 1

4) В свойстве HeaderText для каждого столбца пропишите их названия: Фамилия, Имя, Возраст. Если вы случайно вышли из данного окна, нажмите на компоненту (которая на форме) правую кнопку мышки и выберите «Правка столбцов». Если столбцы не отображаются, установите свойство видимости в True.

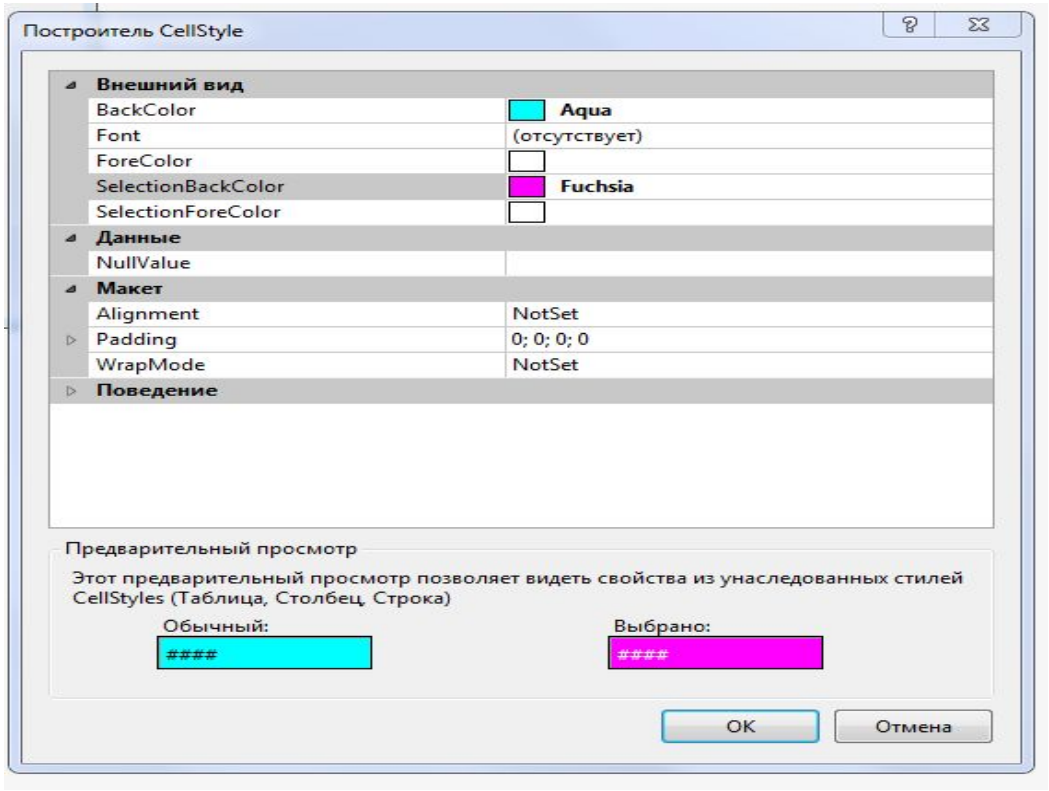
5) Запустите приложение. Проверьте его работу. Заполните таблицу информацией.





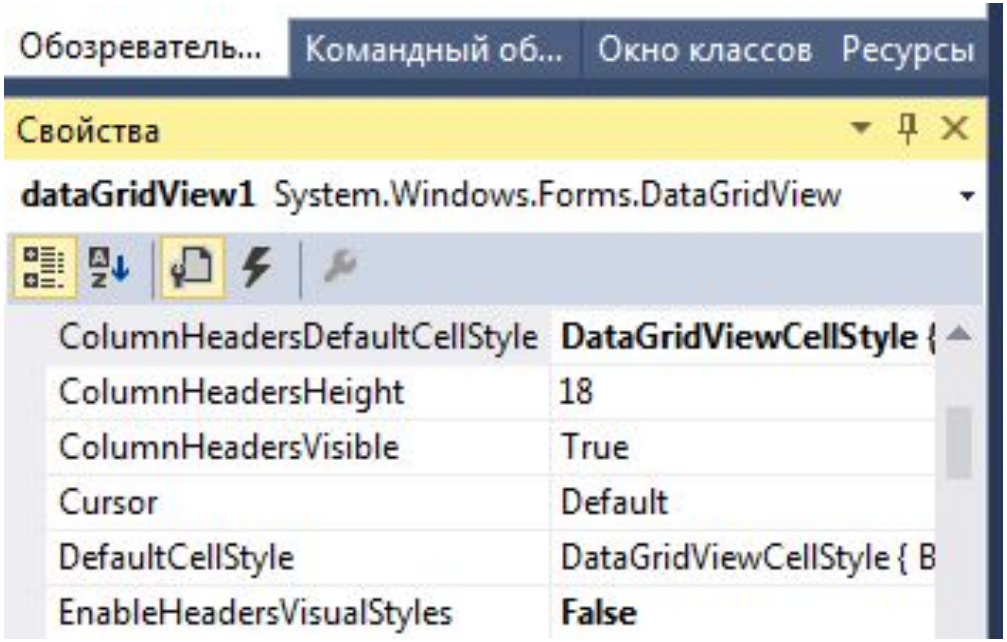
Задание 1

6) Внешний вид каждой ячейки определяется объектом типа DataGridViewCellStyle. Потренируйтесь работать с этим свойством.



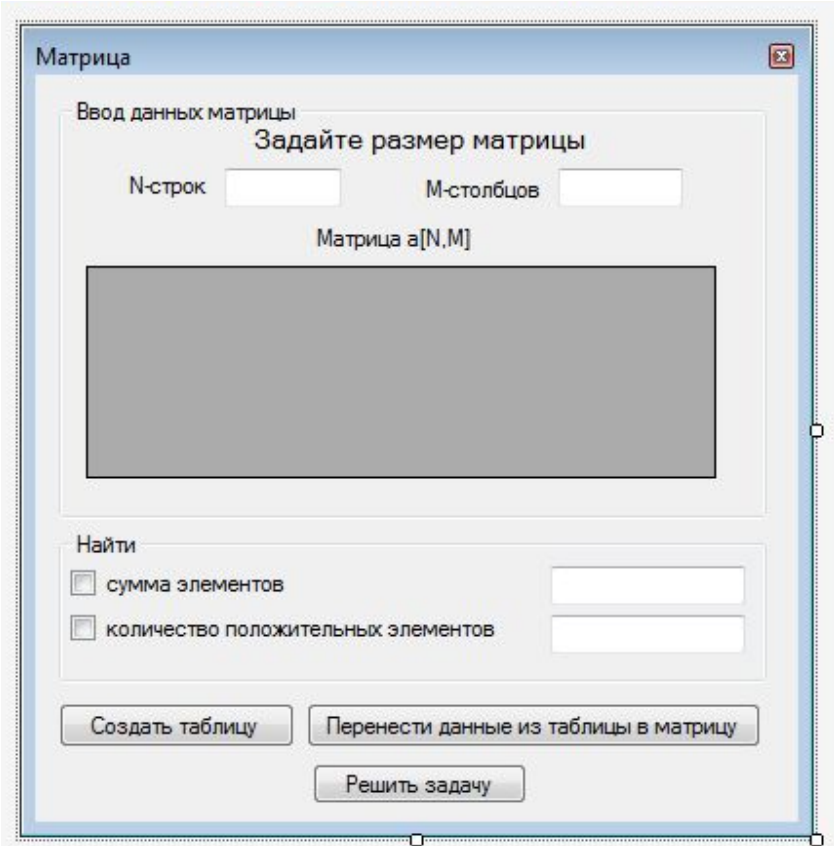
Задание 1

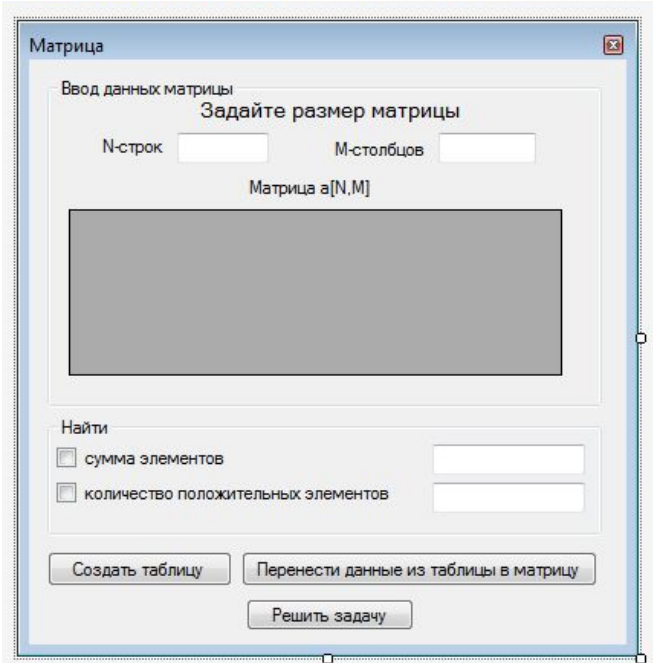
7) Проработай свойство настройки заголовков столбцов. *Если хотите самостоятельно* определить внешний вид заголовков столбцов, значение свойства EnableHeadersVisualStyles потребуется установить равным false.
Определите несколько свойств, определяющих внешний вид заголовков столбцов. Простой способ достижения этого - создание объекта DataGridViewCellStyle.



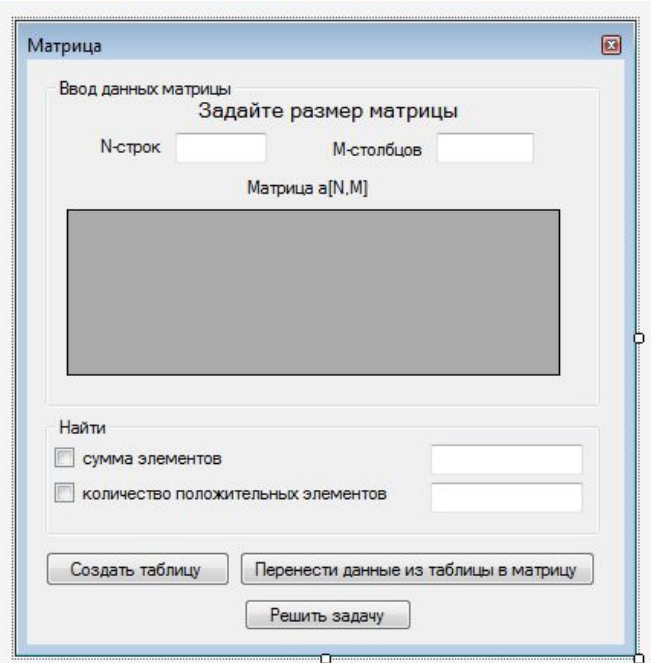
8) Запустите приложение. Проверьте его работу.
9) Установите другие свойства визуальной компоненты DataGridView. Запустите приложение. Проверьте его работу.

```
#pragma once
#include<math.h>
int m, n, s, kol;
int a[50][50];
```



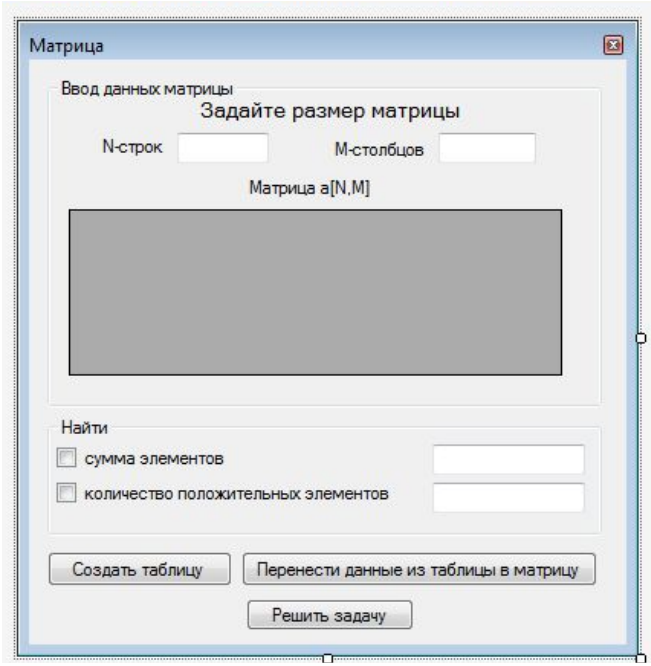



```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
{
       //Проверка, что не пустые компоненты textBoxl и textBox2
       if ((textBox1->Text != "") && (textBox2->Text != ""))
              {
              n = Convert::ToInt32(textBox1->Text);
              m = Convert::ToInt32(textBox2->Text);
                           //Чистка столбцов компонента DataGridView, если они не пусты
                           dataGridView1->Columns->Clear();
                           //Заполнение компонента DataGridView строками
                           dataGridView1->RowCount = n;
                           //Заполнение компонента DataGridView столбцами
                           dataGridView1->ColumnCount = m;
                            }
                           else
{MessageBox::Show("Заполните, пожалуйста, данные", "Ошибка ввода данных",
MessageBoxButtons::OK, MessageBoxIcon::Exclamation); }
                            }
```




```
private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e) {

            for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
            {
            a[i][j] = Convert::ToSingle(dataGridView1->Rows[i]->Cells[j]->Value);
            }
}
```




```
private: System::Void button3_Click(System::Object^  sender, System::EventArgs^  e) {
                    //переменные s и kol обнуляем
                     s = 0; kol = 0;
                     //Выполняем расчеты
                     for (int i = 0; i < n; i++)
                     for (int j = 0; j < m; j++)
                     {
                    s = s + a[i][j];
                    if (a[i][j] > 0)     { kol++; }
                     }
                     //Вывод данных нахождения суммы элементов матрицы
                     if (checkBox1->Checked == true) {
                    textBox3->Text = Convert::ToString(s);
                     }
                    //Вывод данных нахождения количества положительных элементов матрицы
                    if ((checkBox2->Checked == true) && (kol != 0)) {
                     textBox4->Text = Convert::ToString(kol);
                            }
                    else
      if (checkBox2->Checked == true) {textBox4->Text = Convert::ToString("нет элементов"); }
                    return;
}
```