Часть 1.Введение

Visual Studio – это интегрированная среда разработки программ, объединяющая текстовый редактор, компилятор, компоновщик и отладчик.

Visual Studio выполняет компиляцию и запуск программ в соответствии с проектом.

Проект – это структура данных, содержащая всю информацию, необходимую для компиляции исходных файлов программы и ее компоновки со стандартными библиотеками (например, библиотекой ввода/вывода).

Компиляция и компоновка исходных файлов называется сборкой проекта. В результате успешной сборки Visual Studio создает приложение (двоичный исполняемый файл программы).

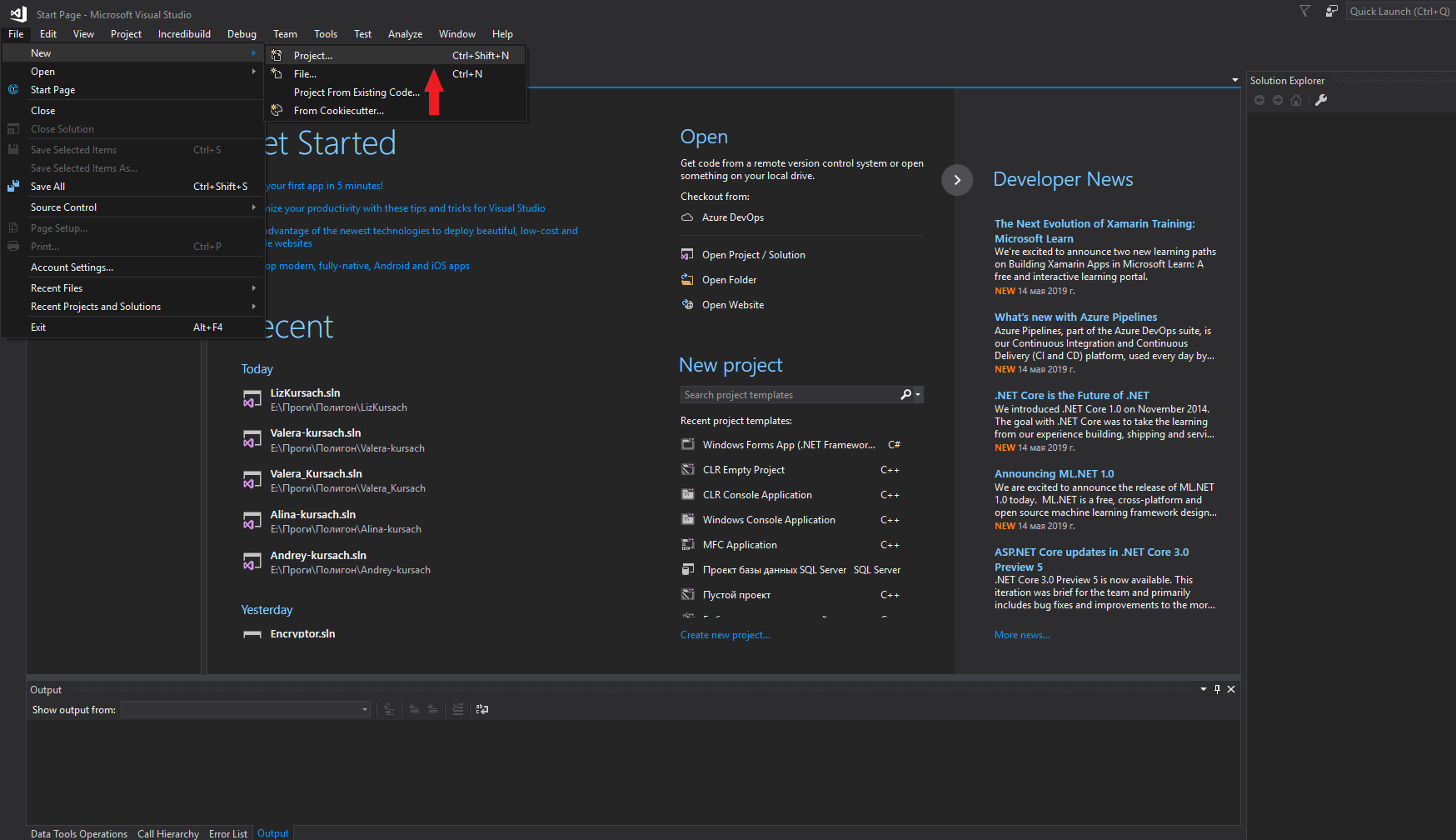
Создание нового проекта Проект состоит из набора файлов с исходным текстом программы (исходных файлов) и набора параметров, определяющих компиляцию и компоновку этих файлов в исполняемый файл приложения.

У проекта должно быть уникальное имя. Параметры проекта хранятся в файлах с расширениями ".DSW" и ".DSP" в папке проекта.

Далее подробно описаны действия по созданию проекта простого CLR приложения Visual C++.

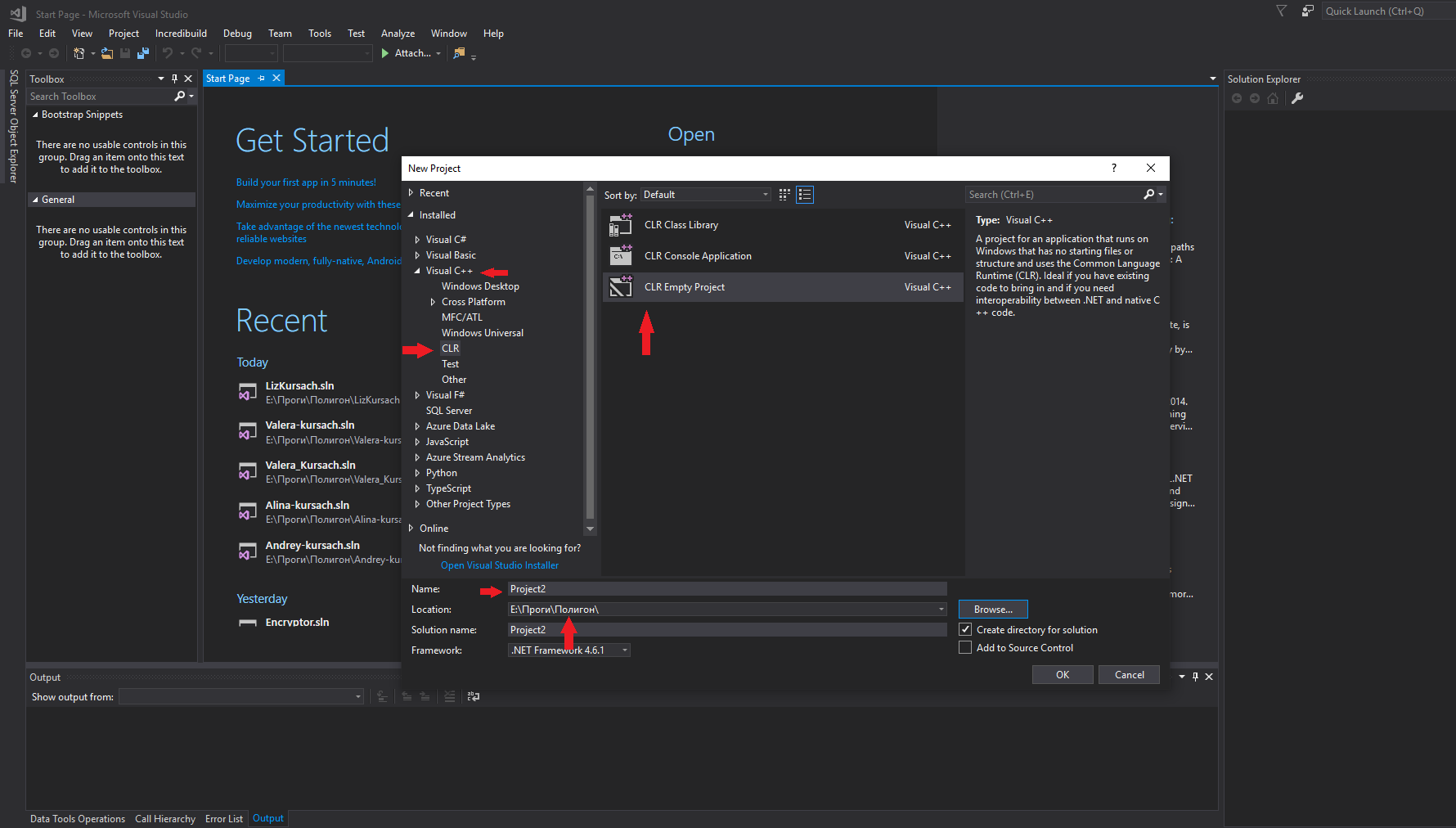
Запустите Visual Studio и проделайте перечисленные ниже действия.

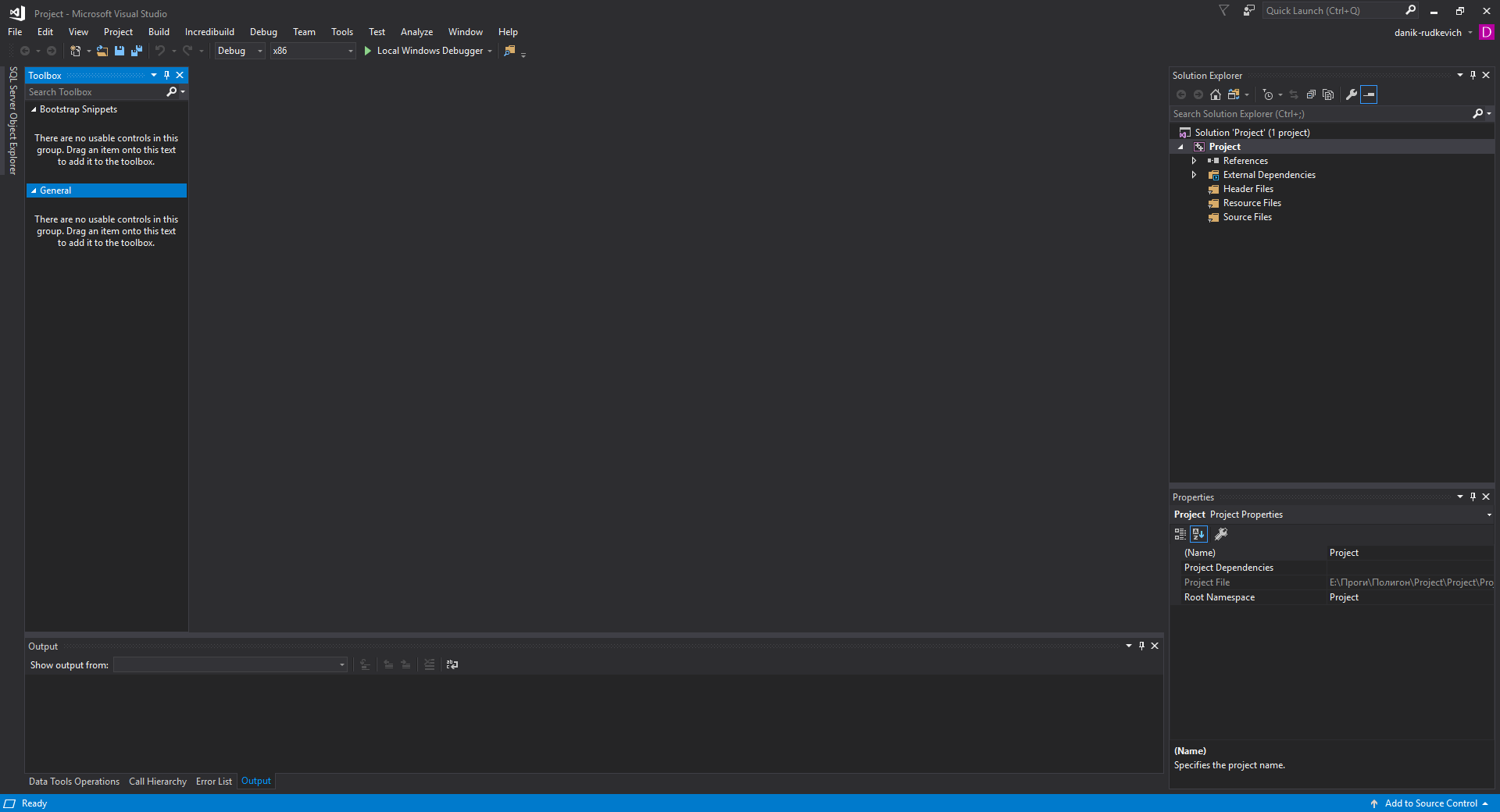
Создание проекта



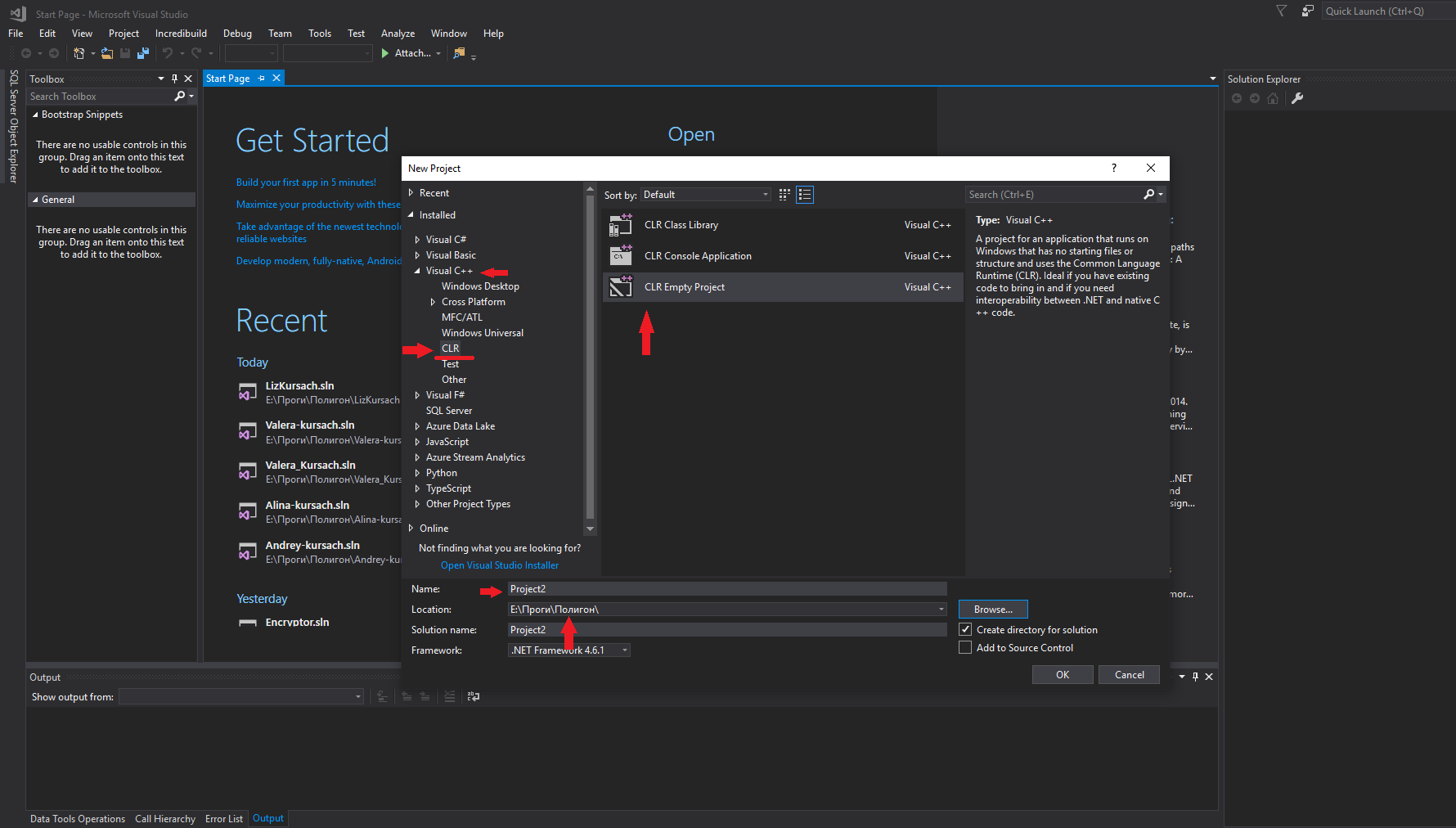
Выбираем пункт File->new->Project

Переходим на вкладку Visual c++ -> CLR -> CLR Empty Project



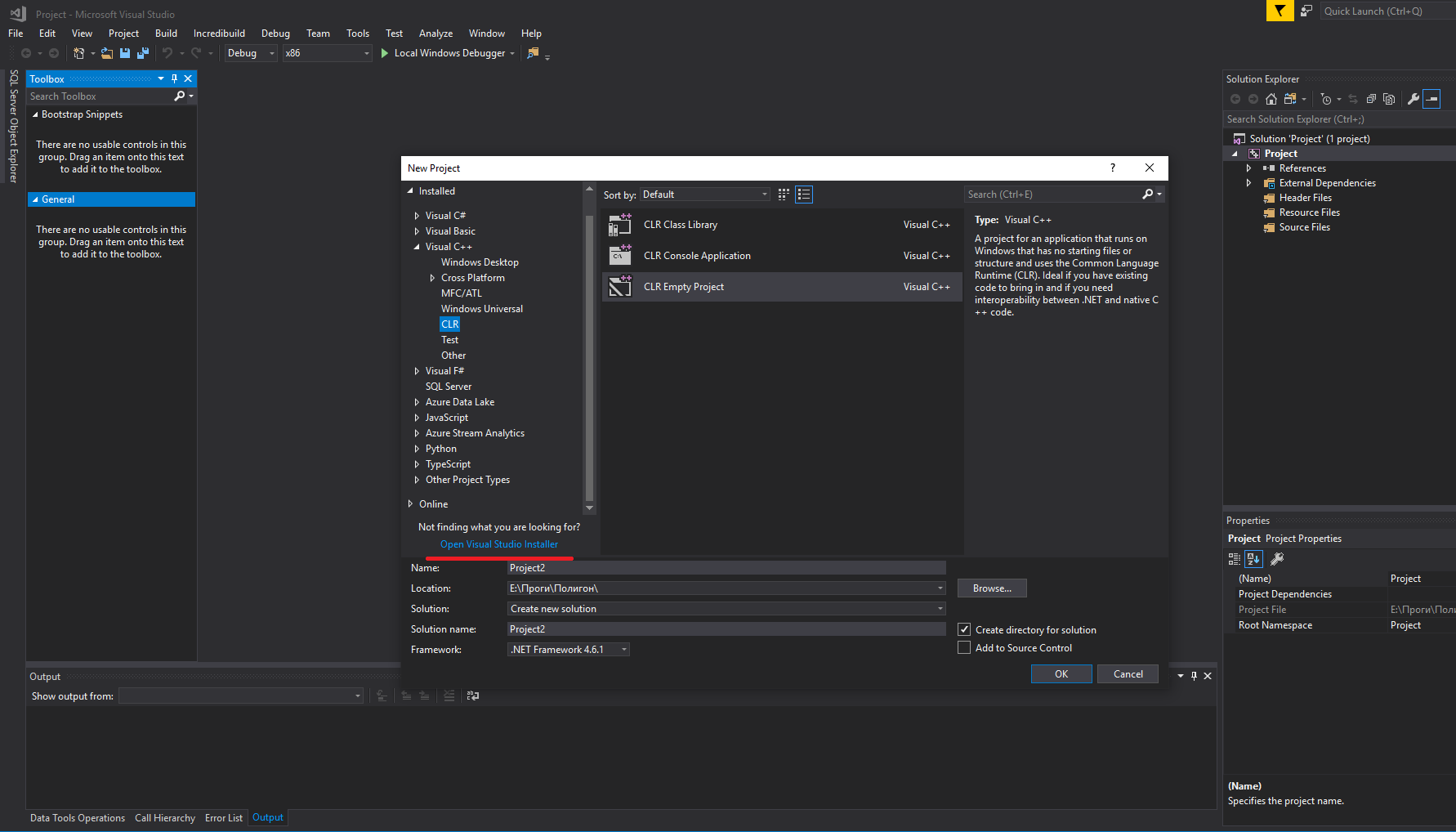


Что делать если у вас нет CLR?

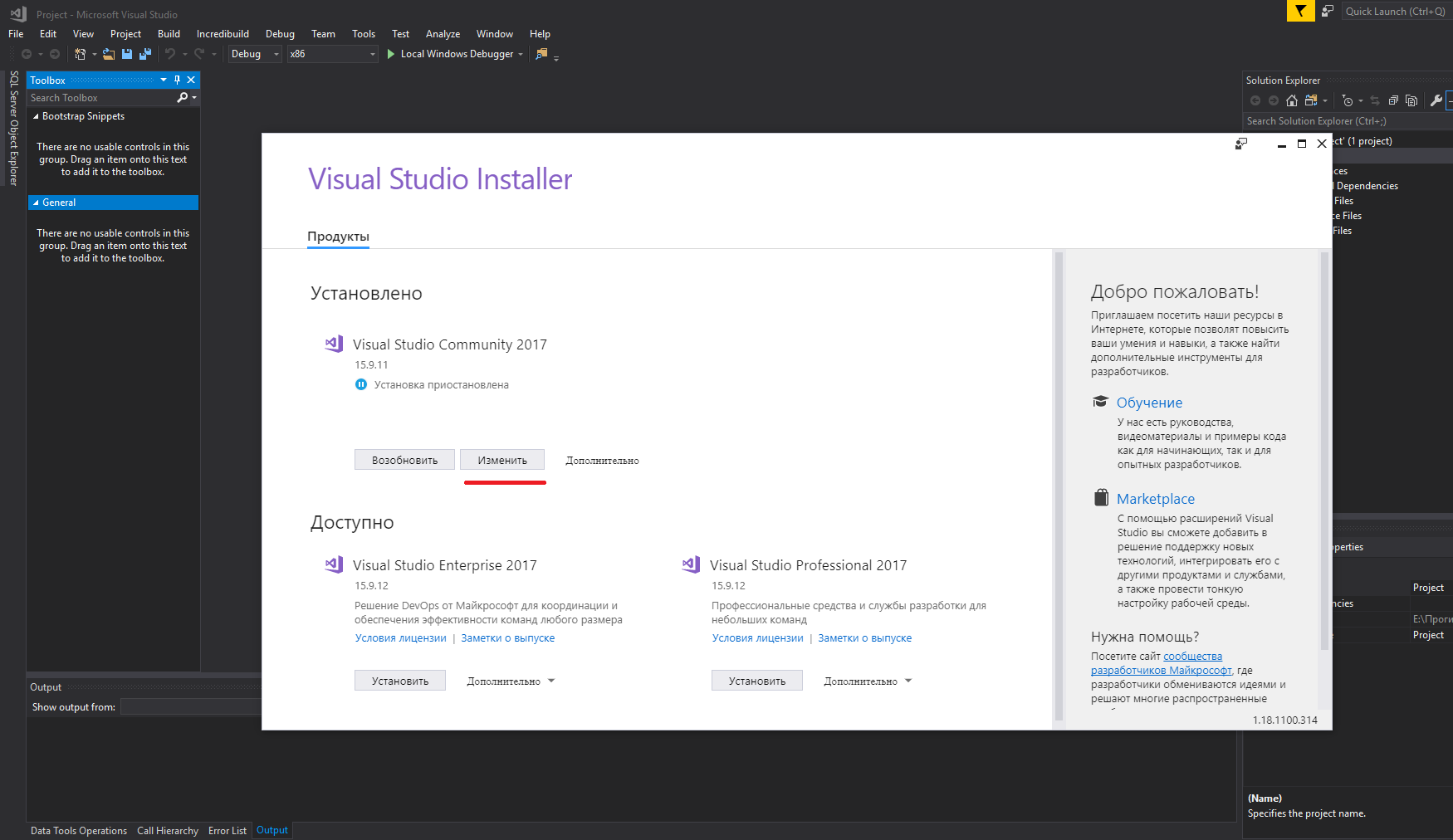


Если у вас нет CLR , то надо его скачать.

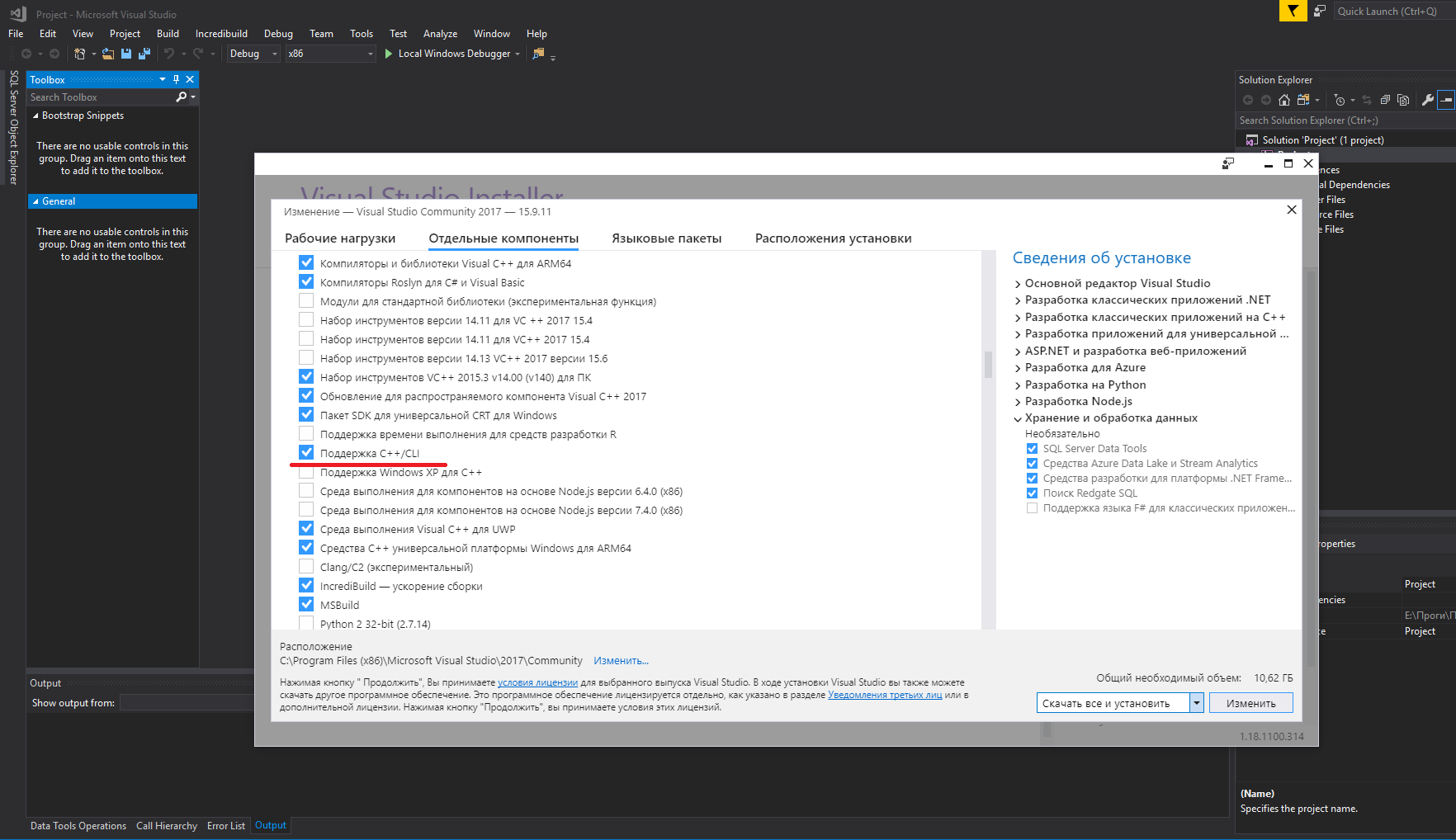
Открываем установщик visual.



Нажимаем на кнопку «Изменить(Modify)».



В открывшемся окне выбираем вкладку "Individual Components" (Индивидуальные компоненты).И докачиваем CLR.

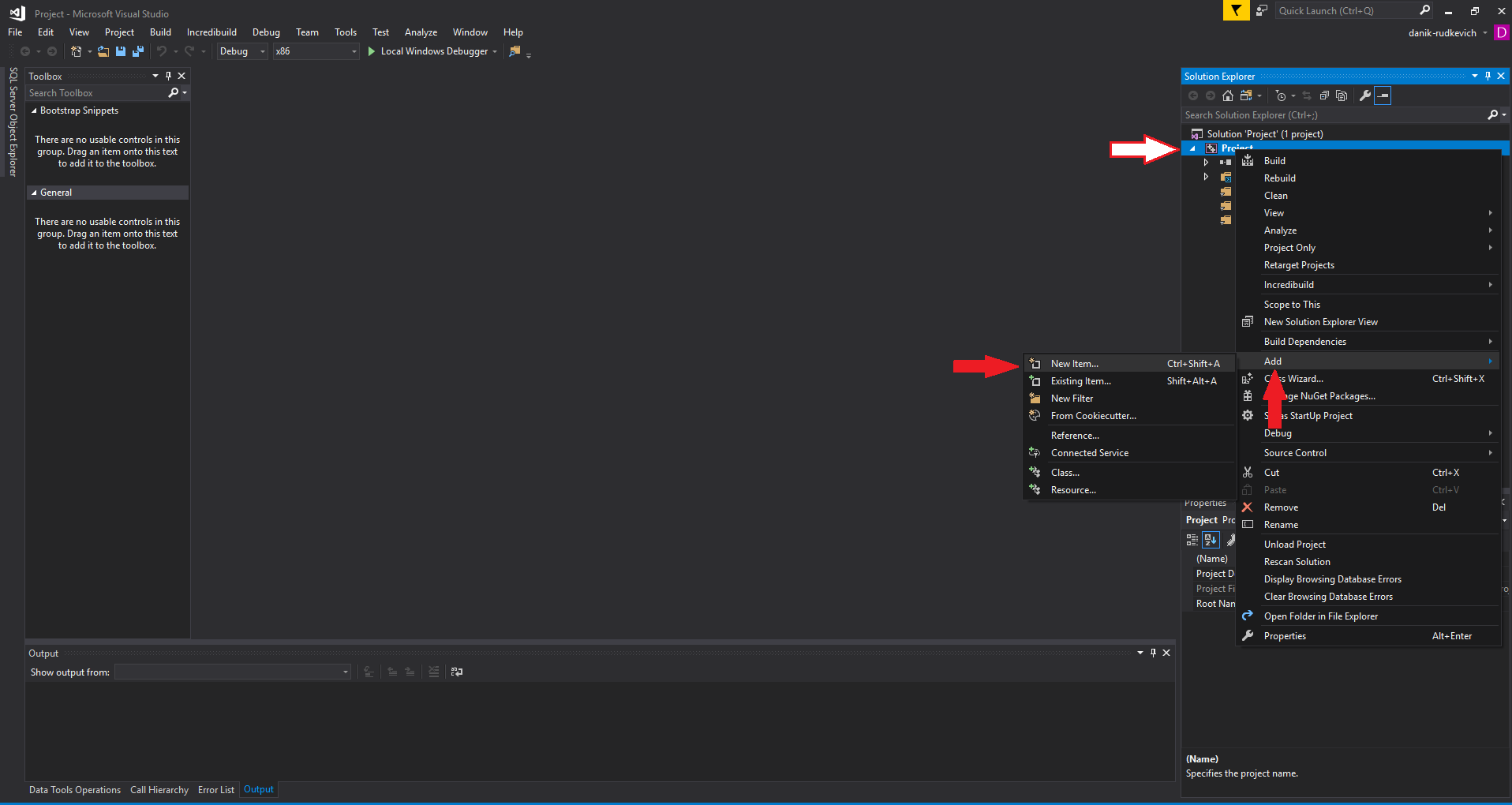


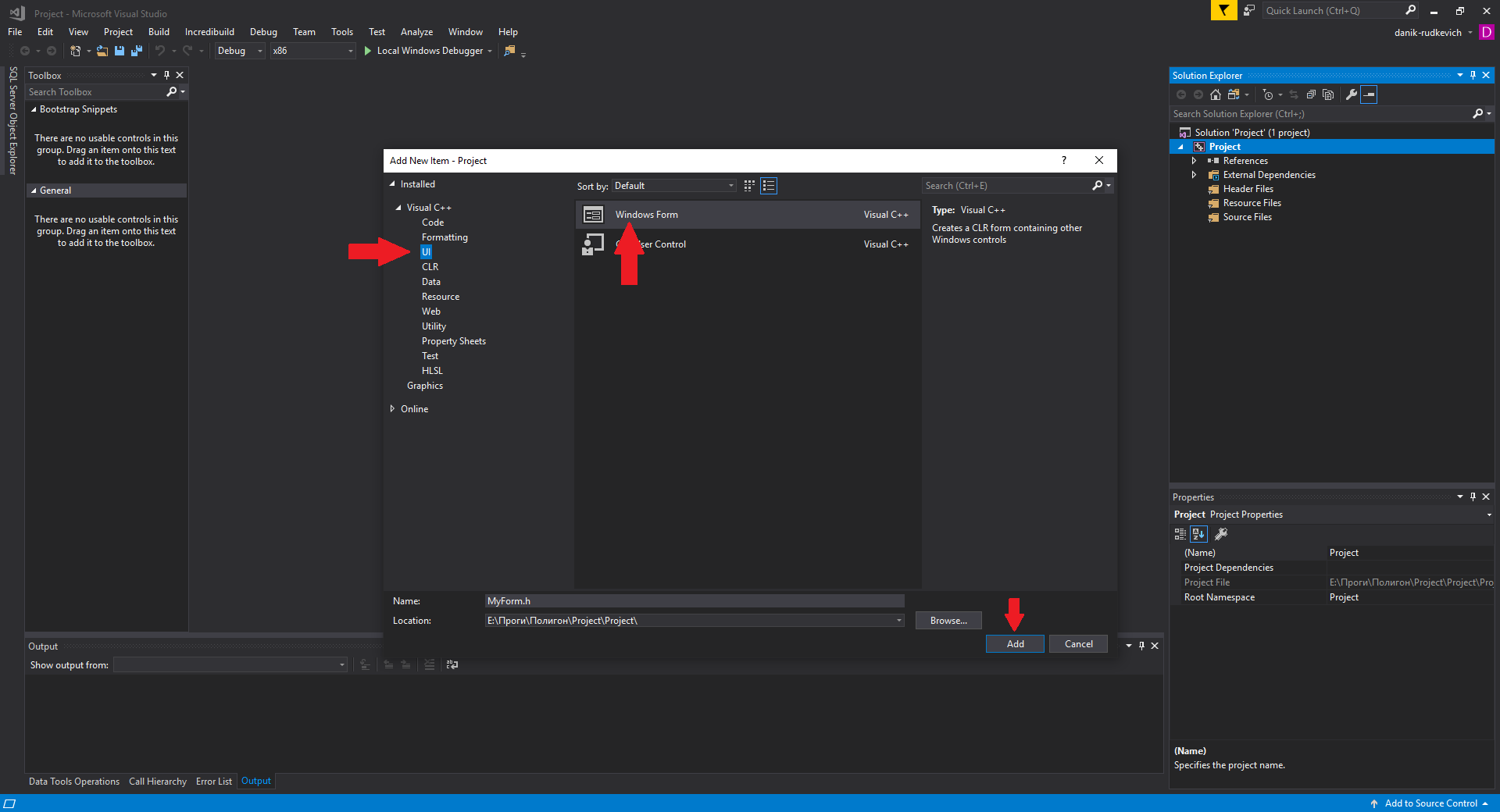
Готово!

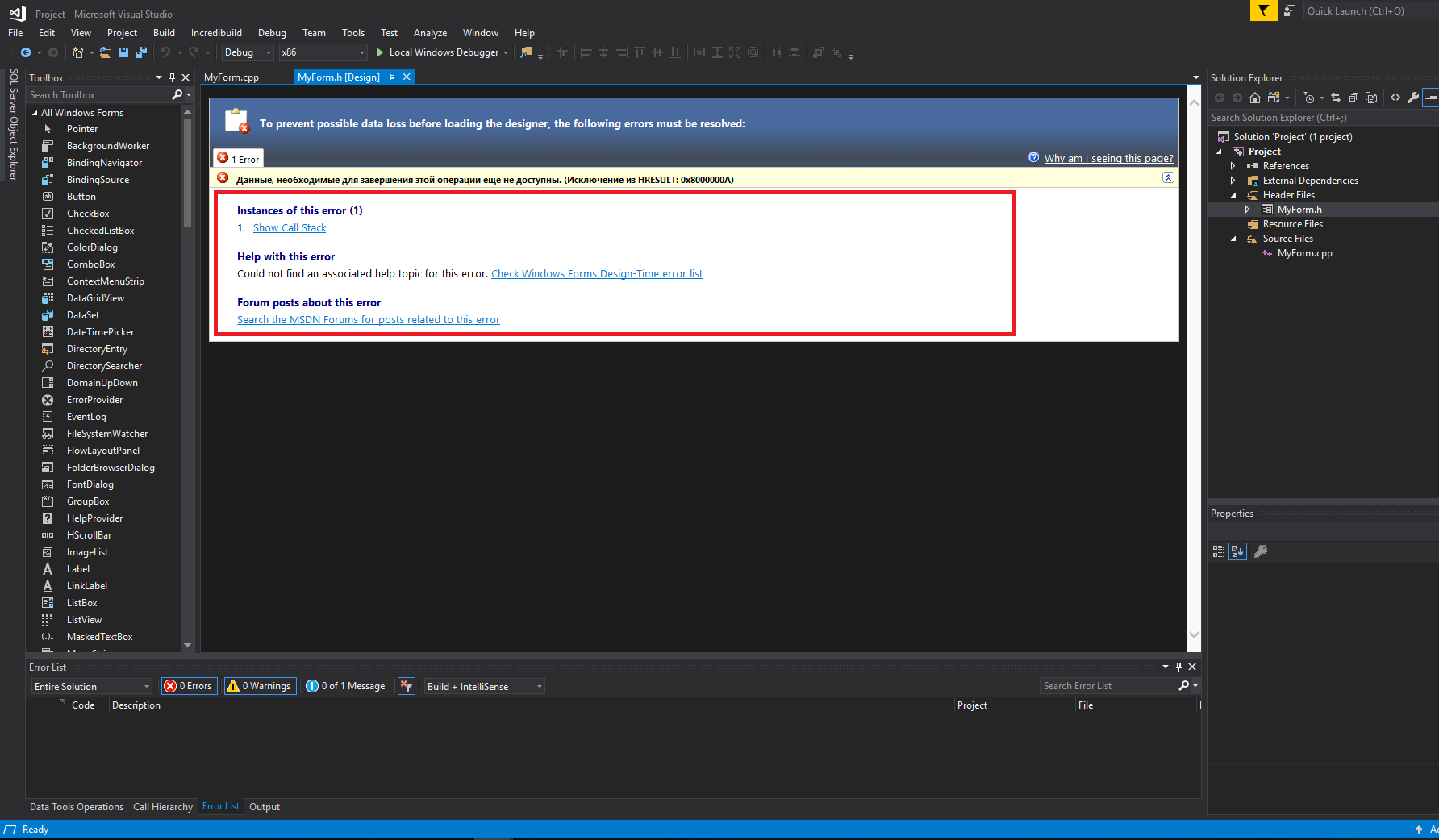
Дописание кода

В созданном проекте форма ещё не добавлена поэтому нам надо добавить её самостоятельно.

Щелкаем правой кнопкой по названию проекта и следуем инструкциям на картинке.

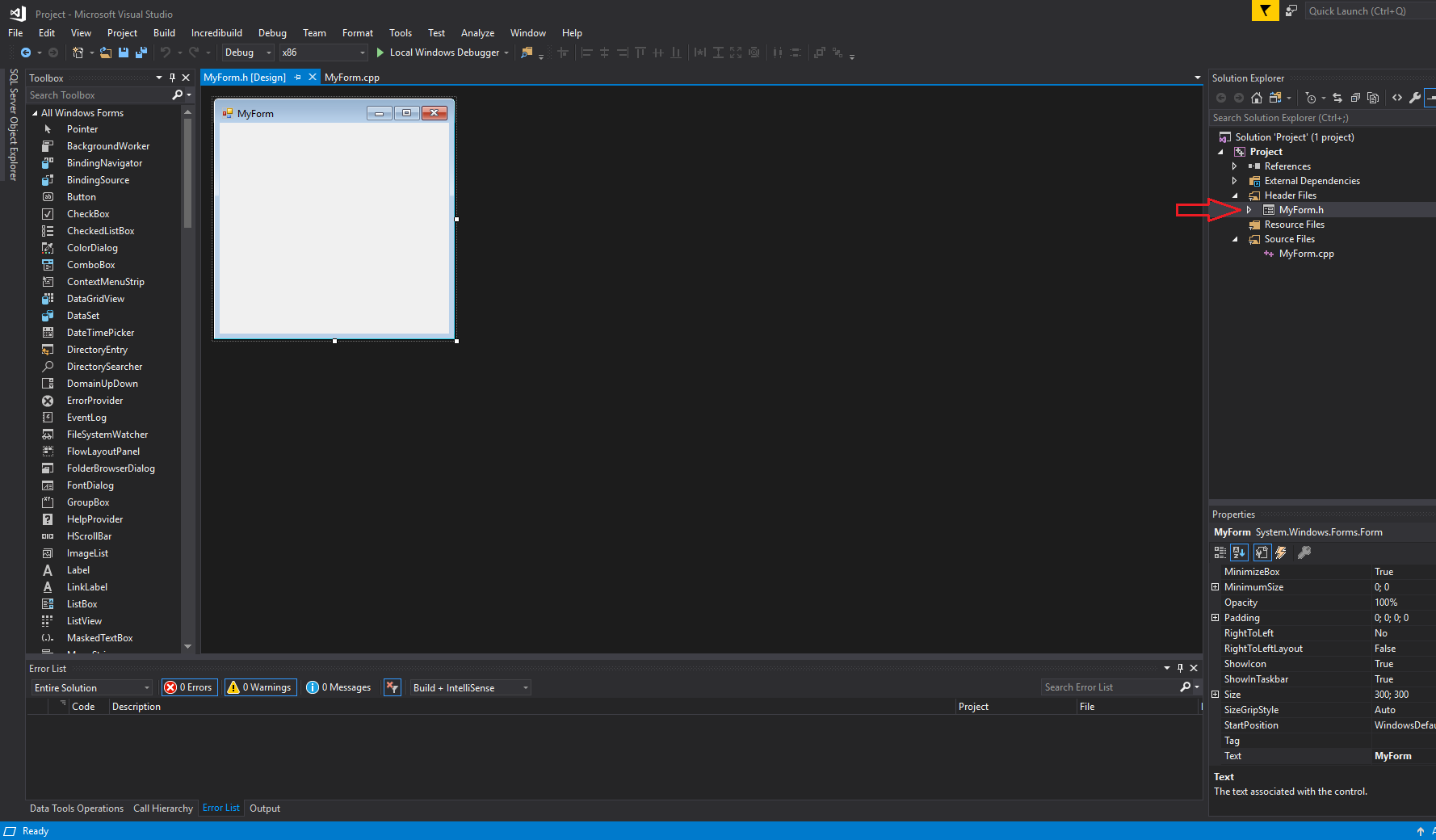
Нажимаем правой кнопкой на проект и нажимаем на Add->New Item

Переходим на вкладку UI и выбираем там Windows Form

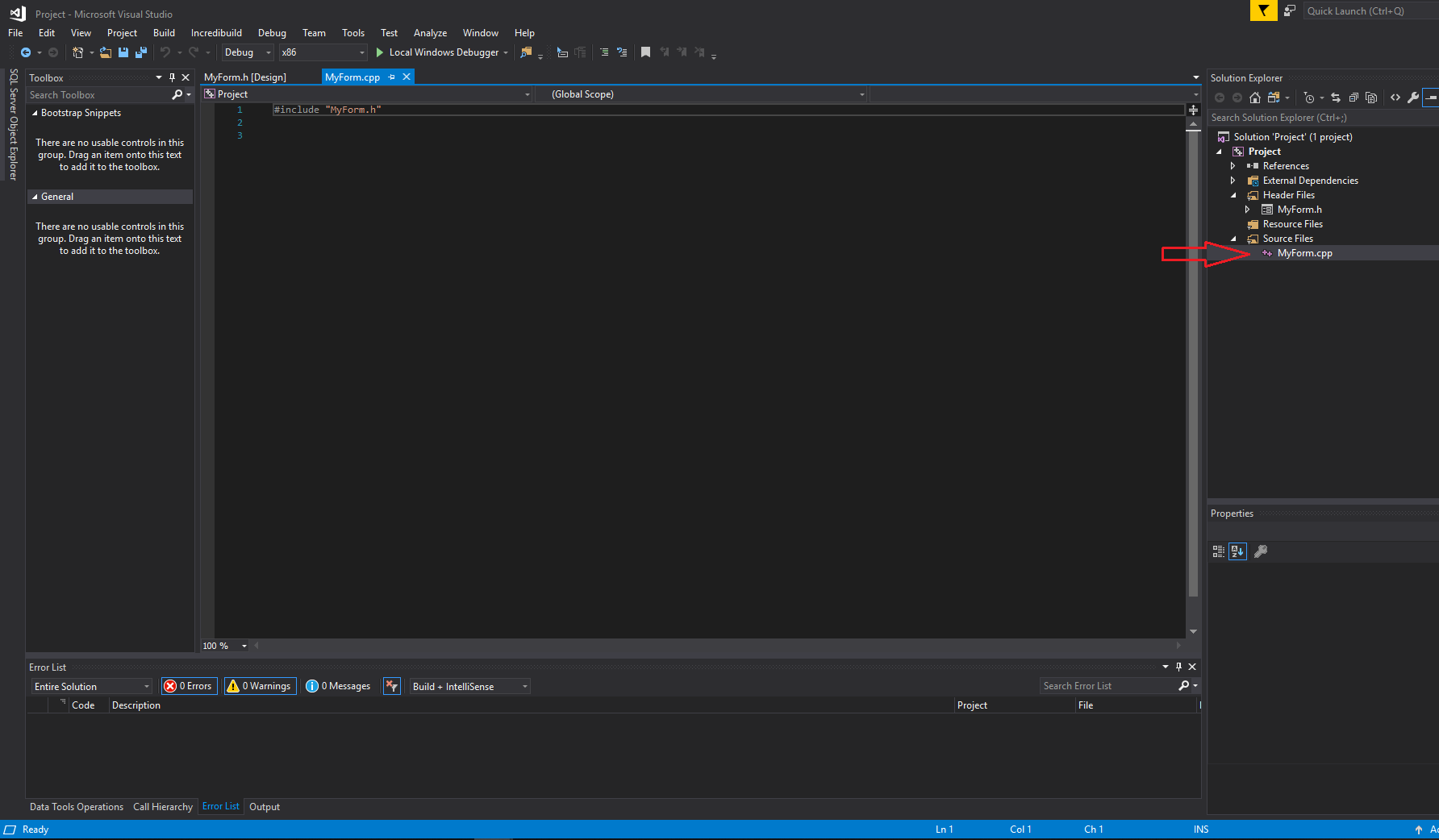


Если у вас выскочило сообщение об ошибке, значит вы все сделали правильно!

Чтобы появился конструктор просто закройте сообщение об ошибке и нажмите на MyForm.h заново.



Далее перейдем в файл cpp нашей формы:



Допишем там данный код после единственной строчки:

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void main(array<String^>^args)

{

Application::EnableVisualStyles();

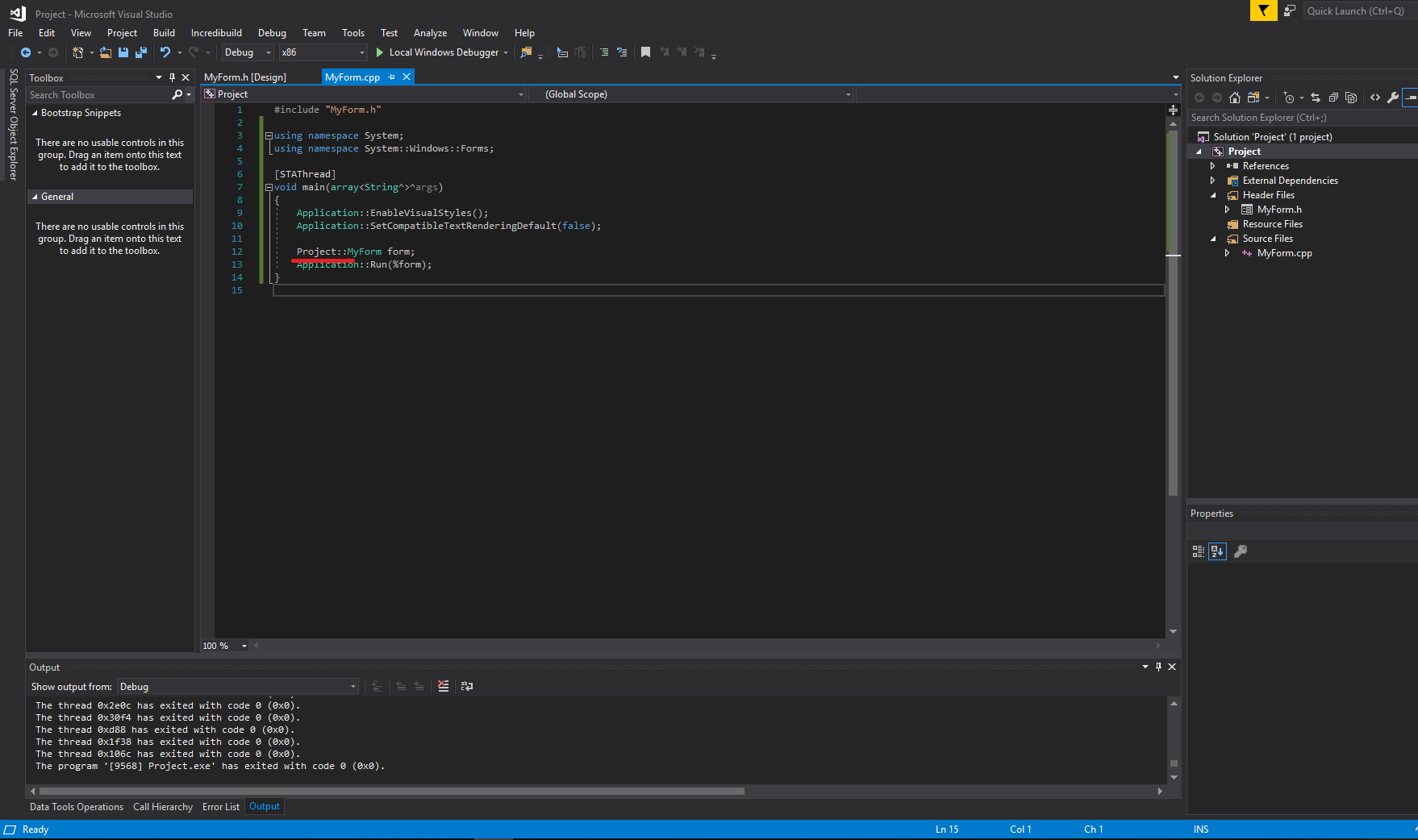
Application::SetCompatibleTextRenderingDefault(false);

Project::MyForm form;

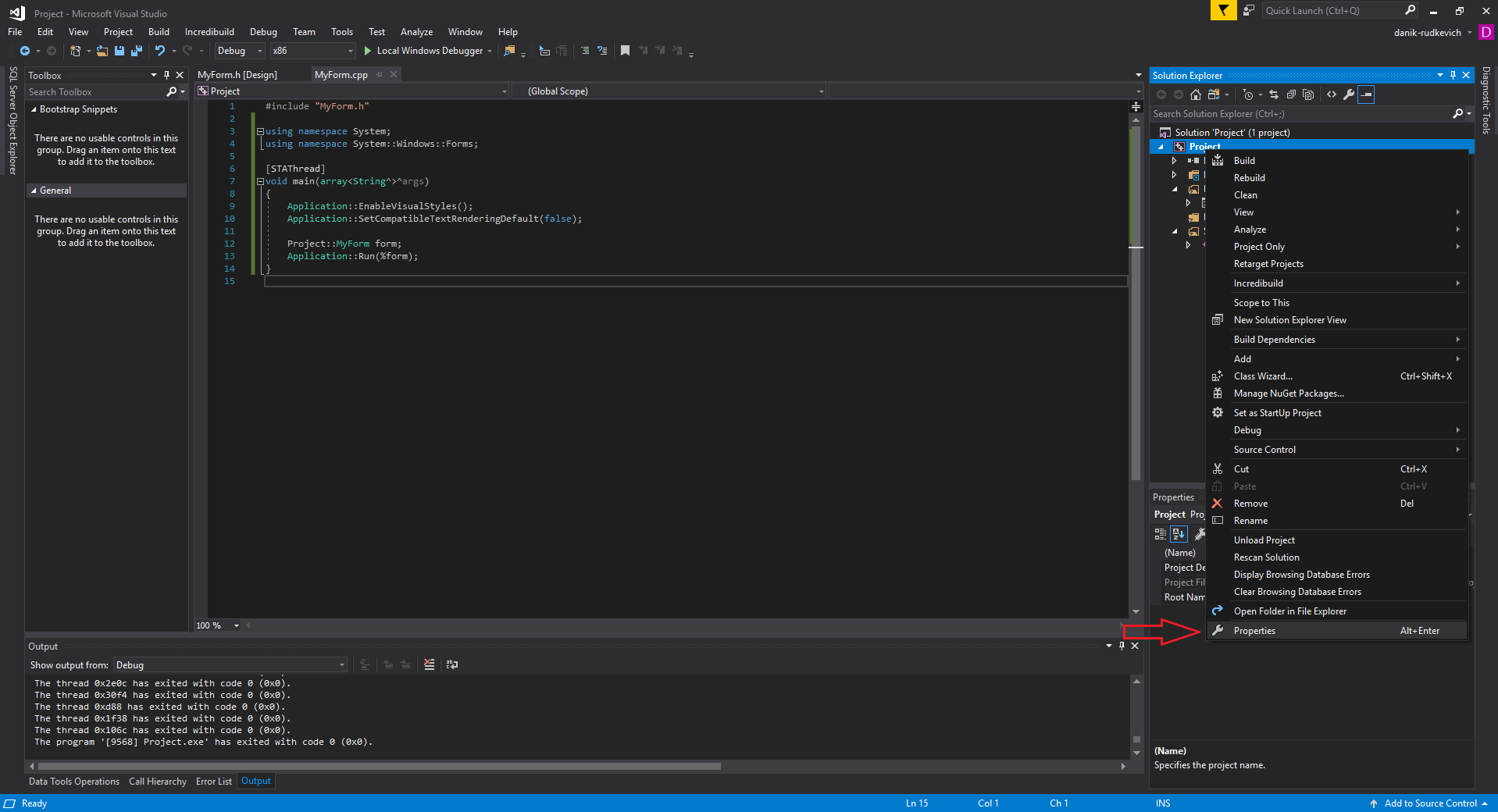
Application::Run(%form);

}

Обратите внимание, что Project — это название вашего проекта в коде выше. Поэтому, если ваш проект называется по другому, то вставьте своё название проекта.

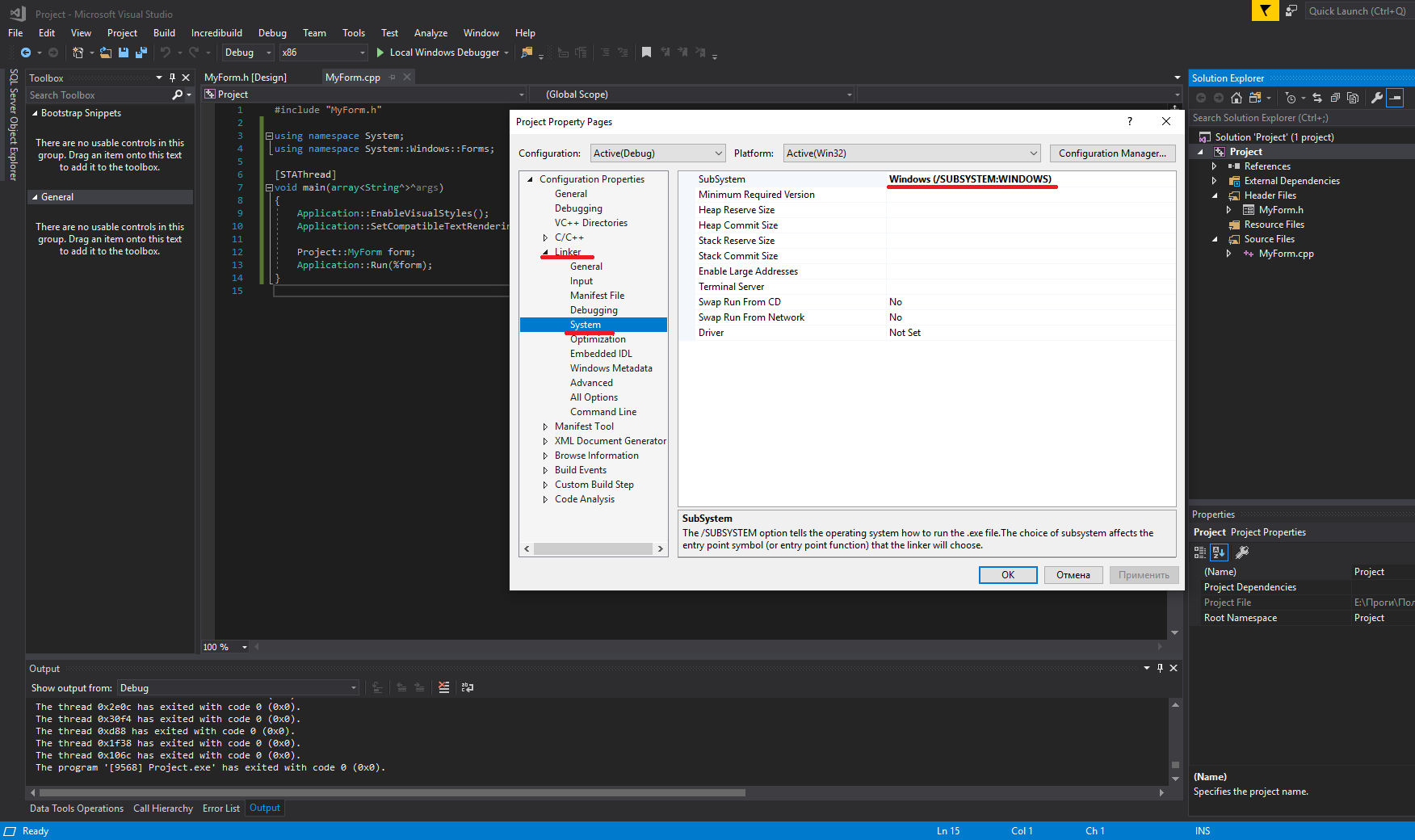
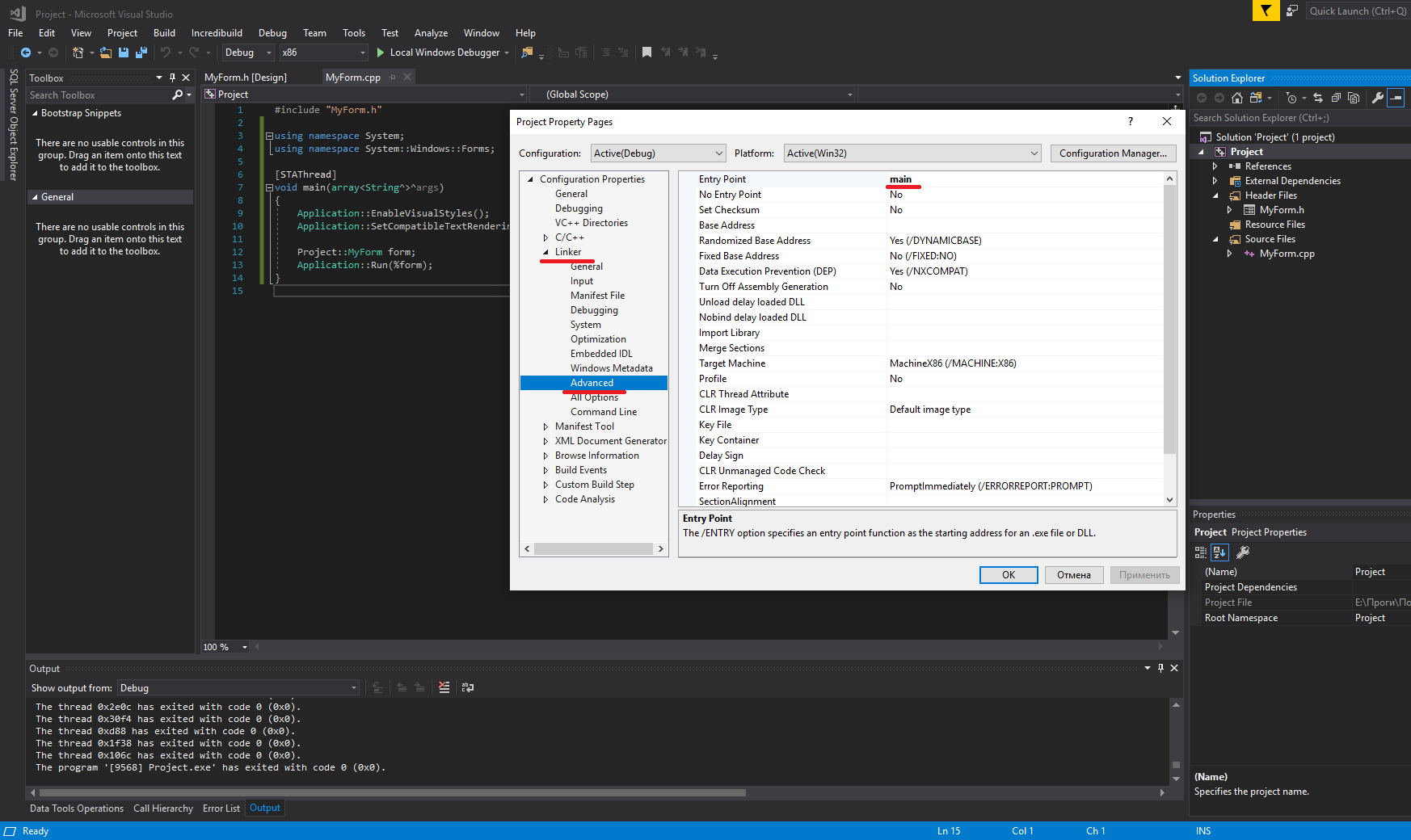


Далее щелкнем по проекту правой кнопкой и перейдем в его свойства:

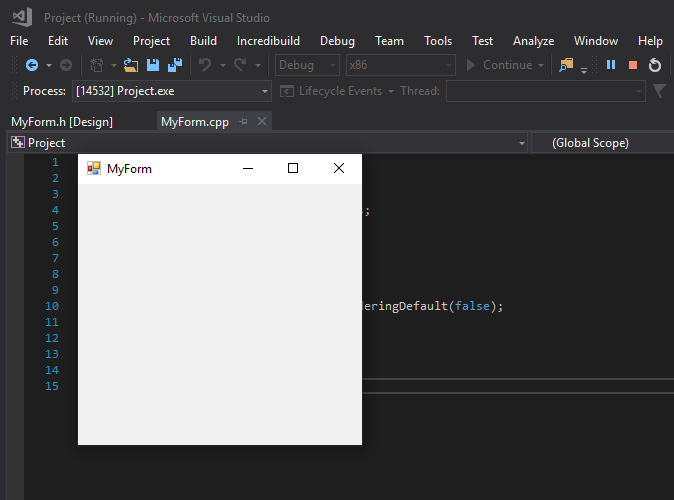
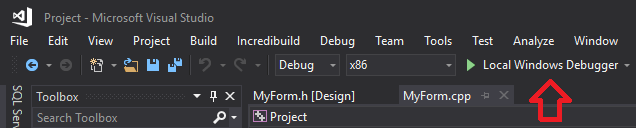


В настройках компоновщика нужно в двух местах поменять настройки:

Во вкладке Linker->System полю SubSystem нужно установить значение «Windows (/SUBSYSTEM:WINDOWS)» , а во вкладке Linker->Advanced полю Entry Point нужно установить значение main.



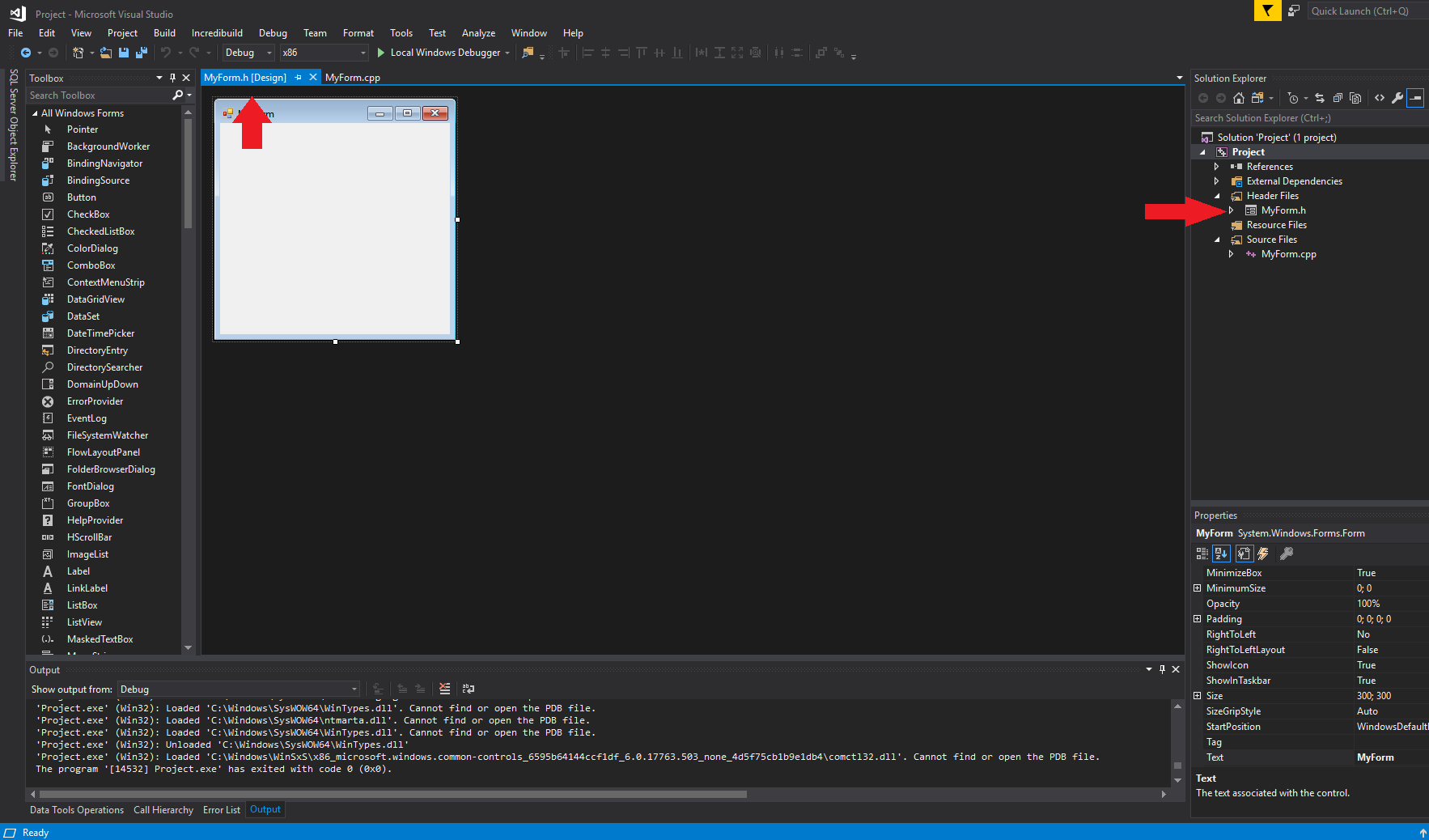
После выполнения всех этих действий мы можем официально считать что добавили формы в наш проект. Давайте запустим наше приложение чтобы проверить его работоспособность.



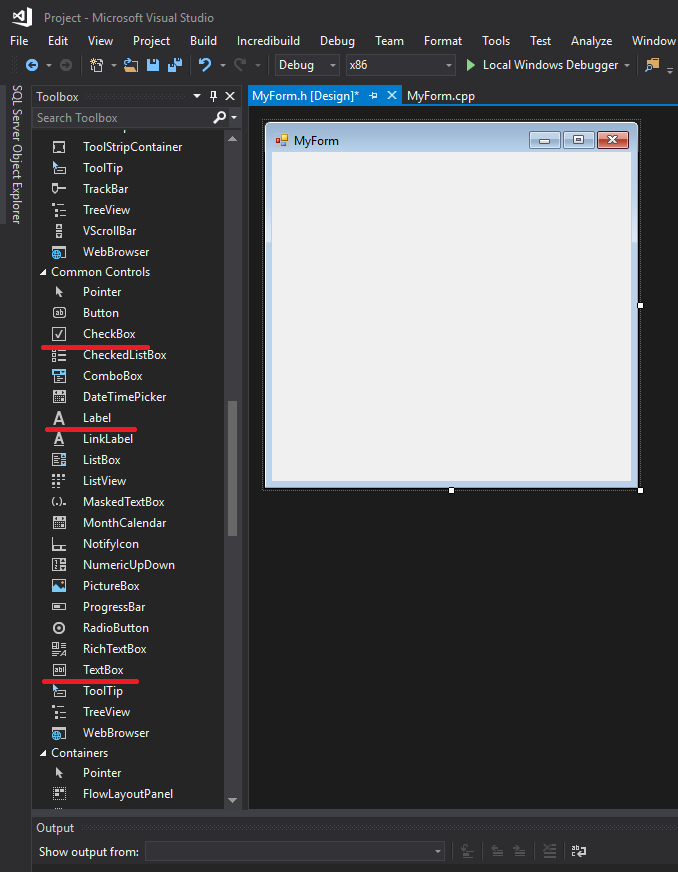
Если форма появилась , то вы сделали все правильно. Если нет , то проверьте всё с самого начала.

Интерфейс приложения

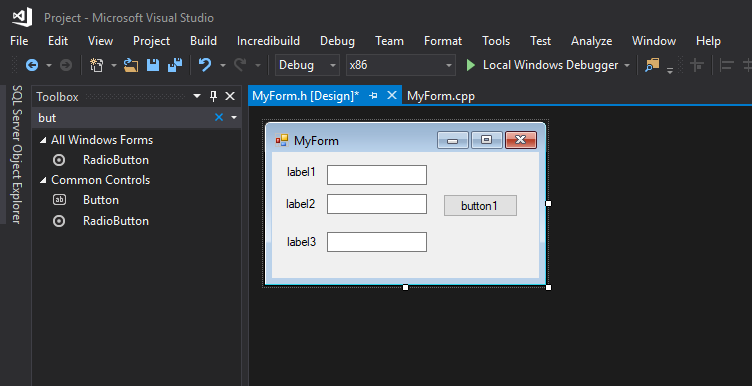
Перейдем на форму. Это можно сделать двумя способами:



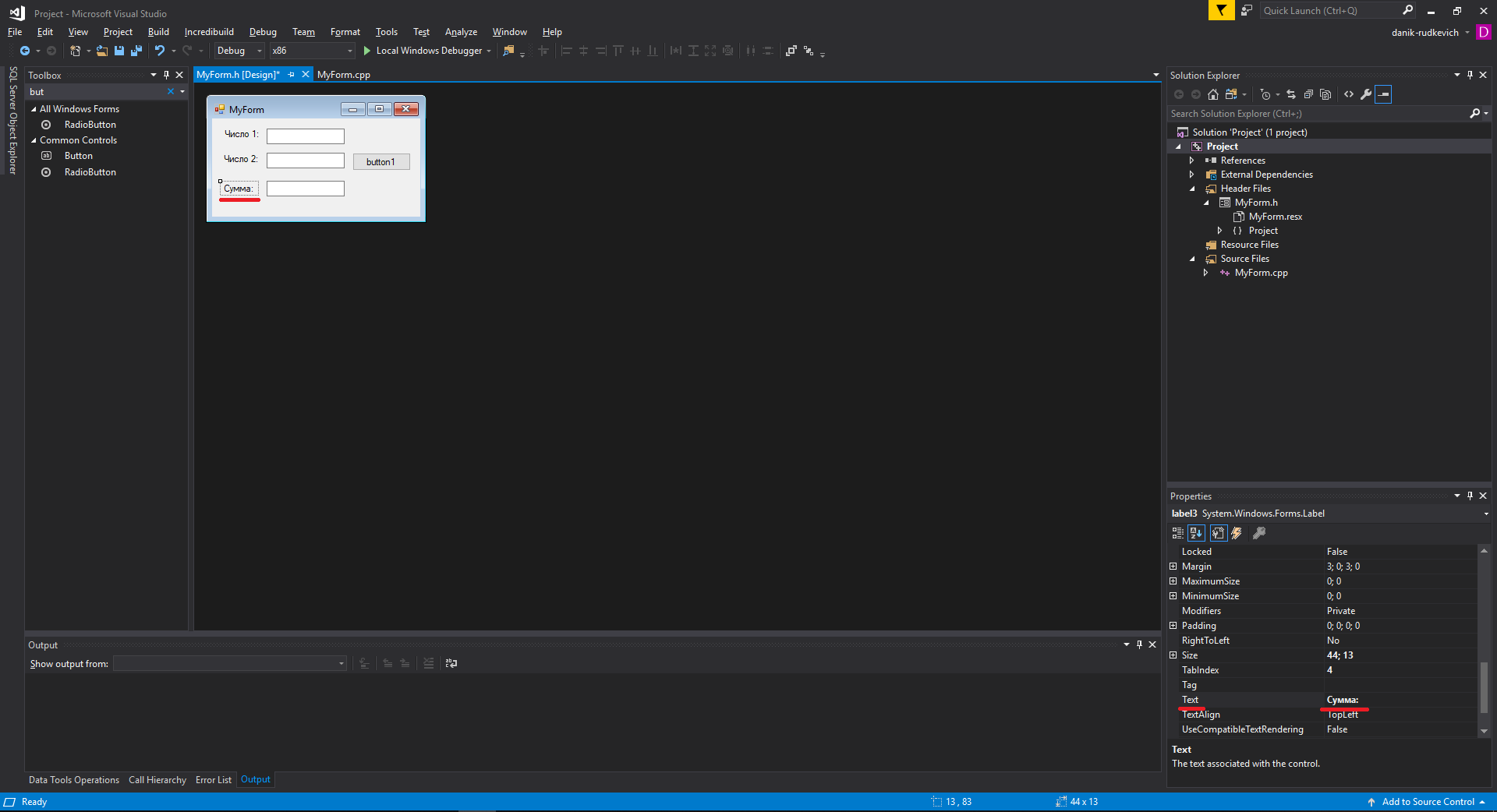
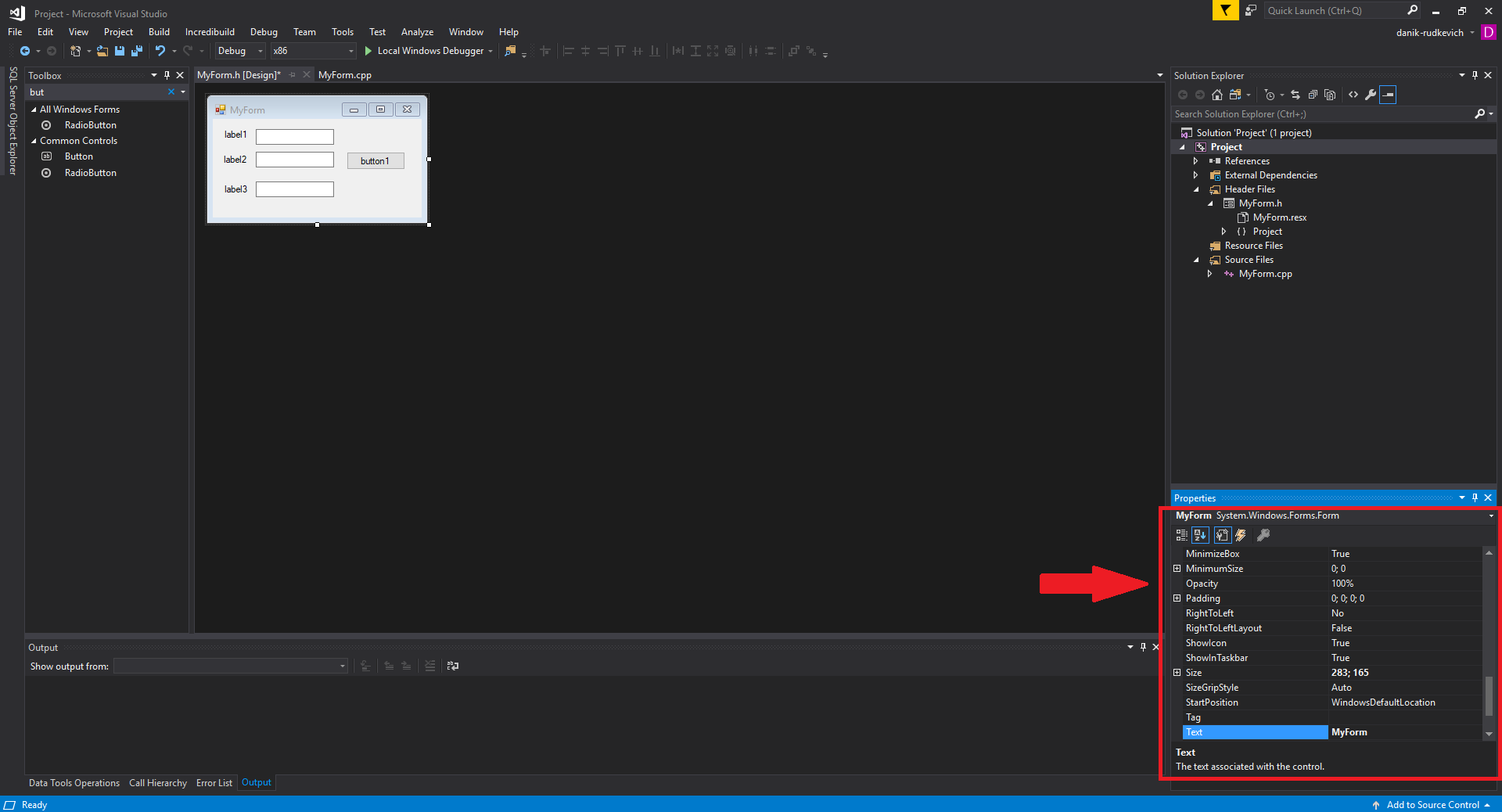
Давайте для примера сделаем программу которая бы находила сумму двух чисел .Первым делам разместим на форму необходимые объекты : 3 label,3 textbox,1 button



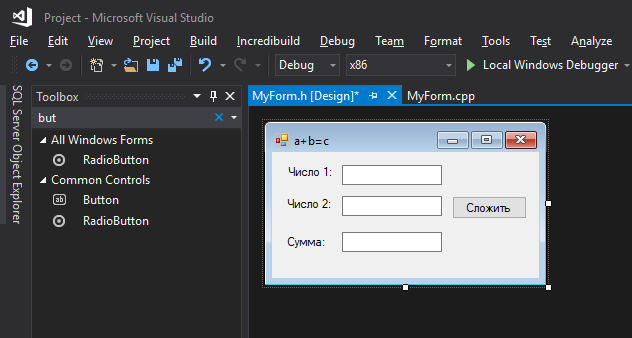
Просто перетаскиваем их на форму. В итоге должны получить следующее:



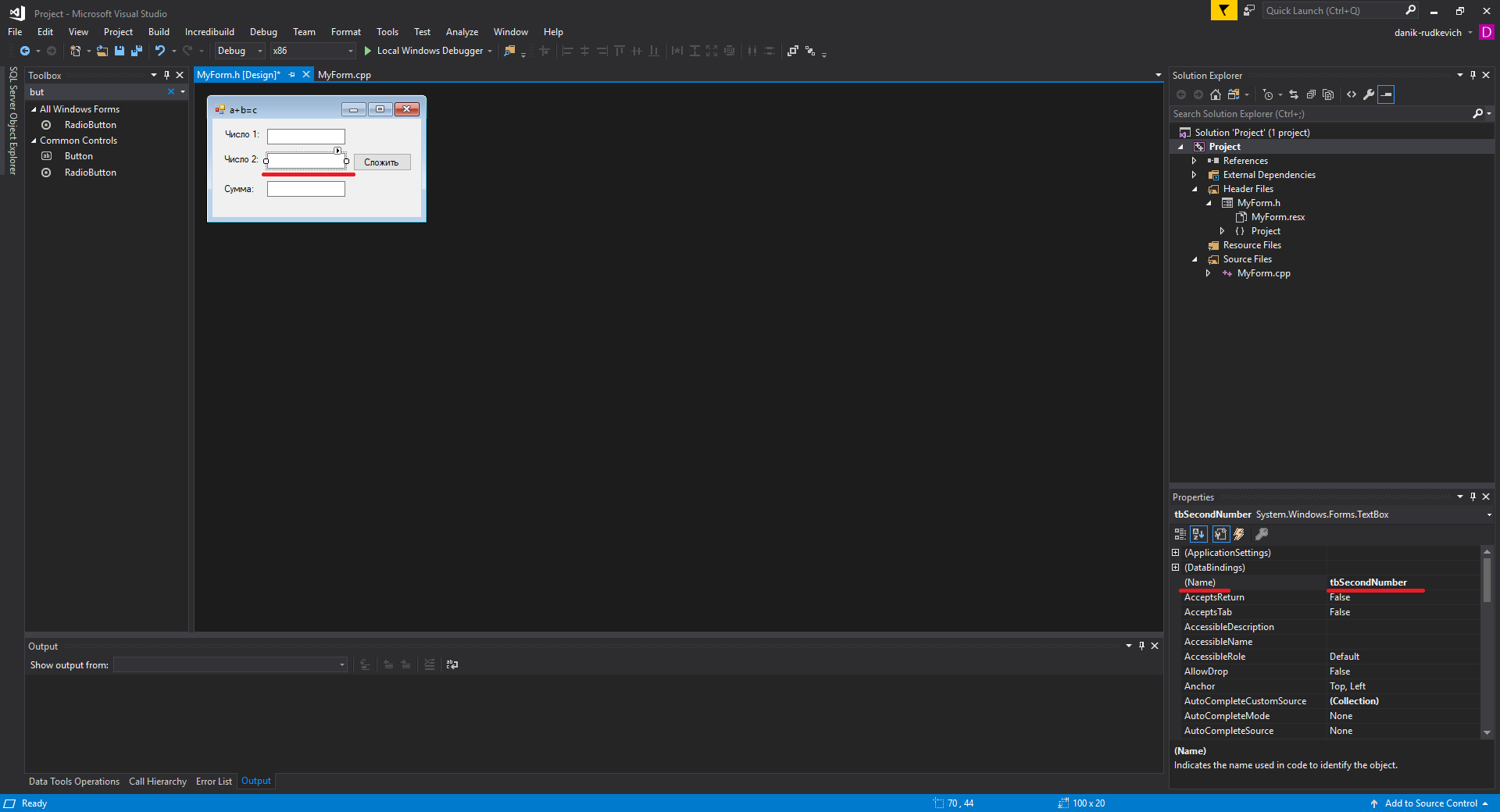
Теперь давайте установим новый текст для label1,label2 и label3.  
Сделать это можно двумя способами.  
Первый способ, воспользоваться окошком «Свойства(properties)»  
В этом окошке будут выводится свойства выделенного в конструкторе объекта. Выделяем нужный нам объект label и меняем значение его поля «Text» на нужное нам значение.



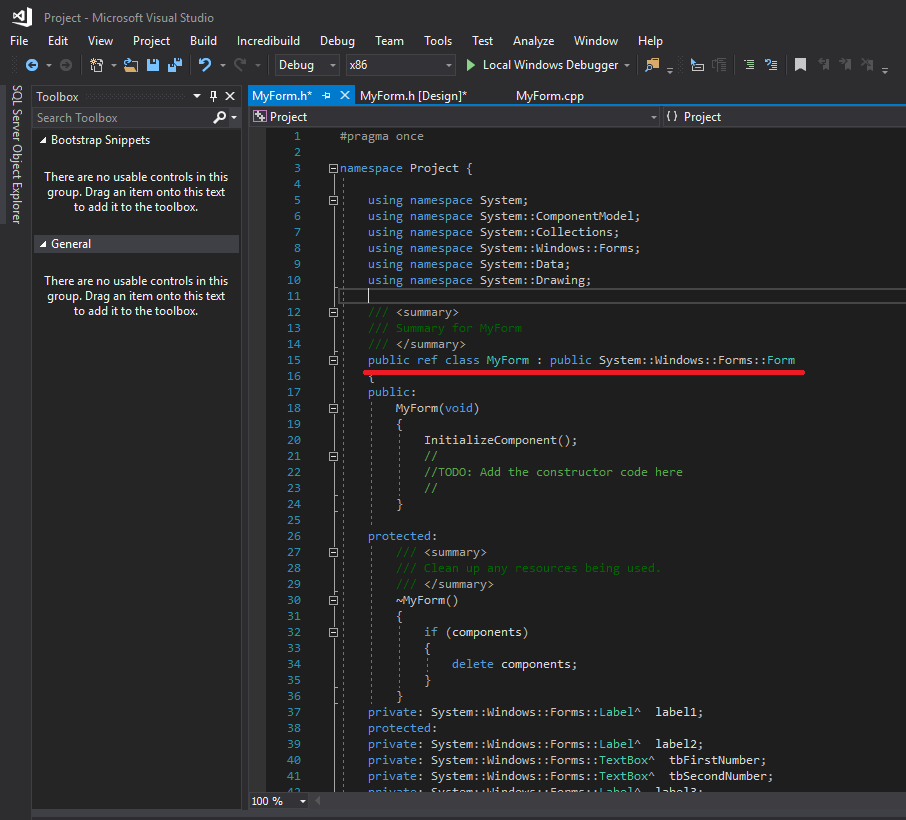
Переименовав всё по своему вкусу в результате моя форма стала выглядеть вот-так:



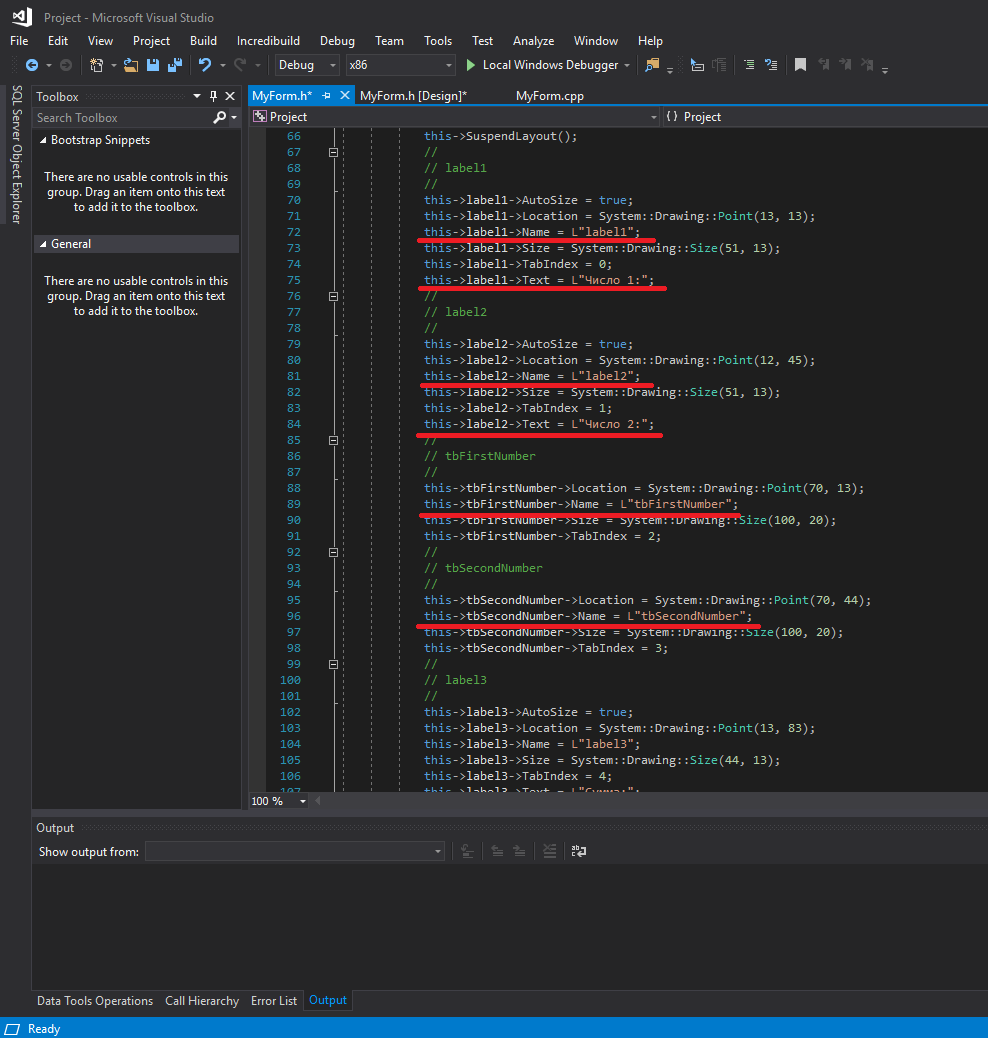
Перед тем как перейти к коду нам нужно задуматься о своём будущем удобстве. При нажатии на кнопку «Сложить» наша форма должна взять значения из двух верхних полей и положить сумму этих полей в самое нижнее поле и чтобы взять или положить значение в это поле к нему нужно как-то обратится. Для этого у каждого объекта на форме есть своё уникальное имя которое обычно генерируется автоматически при добавлении элемента на форму. И так по умолчанию поля с которыми нам надо работать имеют стандартные имена для обращения textbox1,textbox2 и textbox3. Конечно можно оставить всё так как оно есть , но мы же знаем правила «Хорошего тона» и следуем ему при создании программ. Чтобы нам было понятно в коде с каким объектом мы работаем и что в нём хранится , мы дадим полям уникальные имена со смыслом и теперь они будут называться tbFirstNumber, tbSecondNumber и tbSum , ну и кнопке мы тоже дадим уникальное имя для обращения.Для label придумывать имена не будем , так как из кода мы к ним обращаться не будем. Теперь подробнее разберём «Хороший тон» создания имен для обращения. Первым идет сокращение типа объекта таким образом textbox превращается в tb, label в lbl , button в btn, RichTextBox в rtb(Сокращение можете придумывать свои, я вам просто показал самые распространённые варианты сокращения), а далее идёт «смысловая нагрузка» нашего объекта(Что в нём хранится, что этот объект делает и тд.).

Дать новое уникальное имя для обращения к объекту вы можете здесь: 

Мы рассмотрели первый способ оформления когда мы просто говорим Visual где, что и как поправить , а она сама вносит нужные изменения в код куда надо чтобы всё выглядело так , как вы хотите. Как вы догадались второй способ заключается в ручном оформлении нашей формы прямо в коде. Для начала нам нужно открыть код заголовочного файла нашей формы (В данном случае – это MyForm.h) при помощи клавиши «f7».Как только вы открыли этот файл вы обнаружили , что он далеко не пустой! Всё верно, в этом файле описывается класс нашей формы.



Объекты добавленные на форму автоматически добавляются в этот класс и автоматически получают стандартные настройки. Разумеется мы можете внести изменения вручную(Если знаете куда вносить и не боитесь что все сломается).В данном случаи за уникальное имя объекта и текст выводимый на форму отвечают вот эти поля:

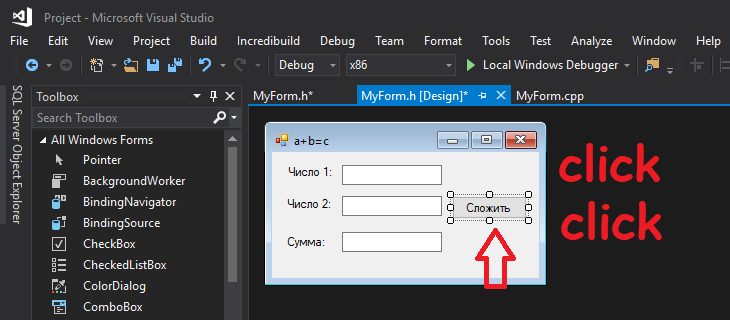


Все изменения внесенные вами вручную немедленно отобразятся на вашей форме в конструкторе.

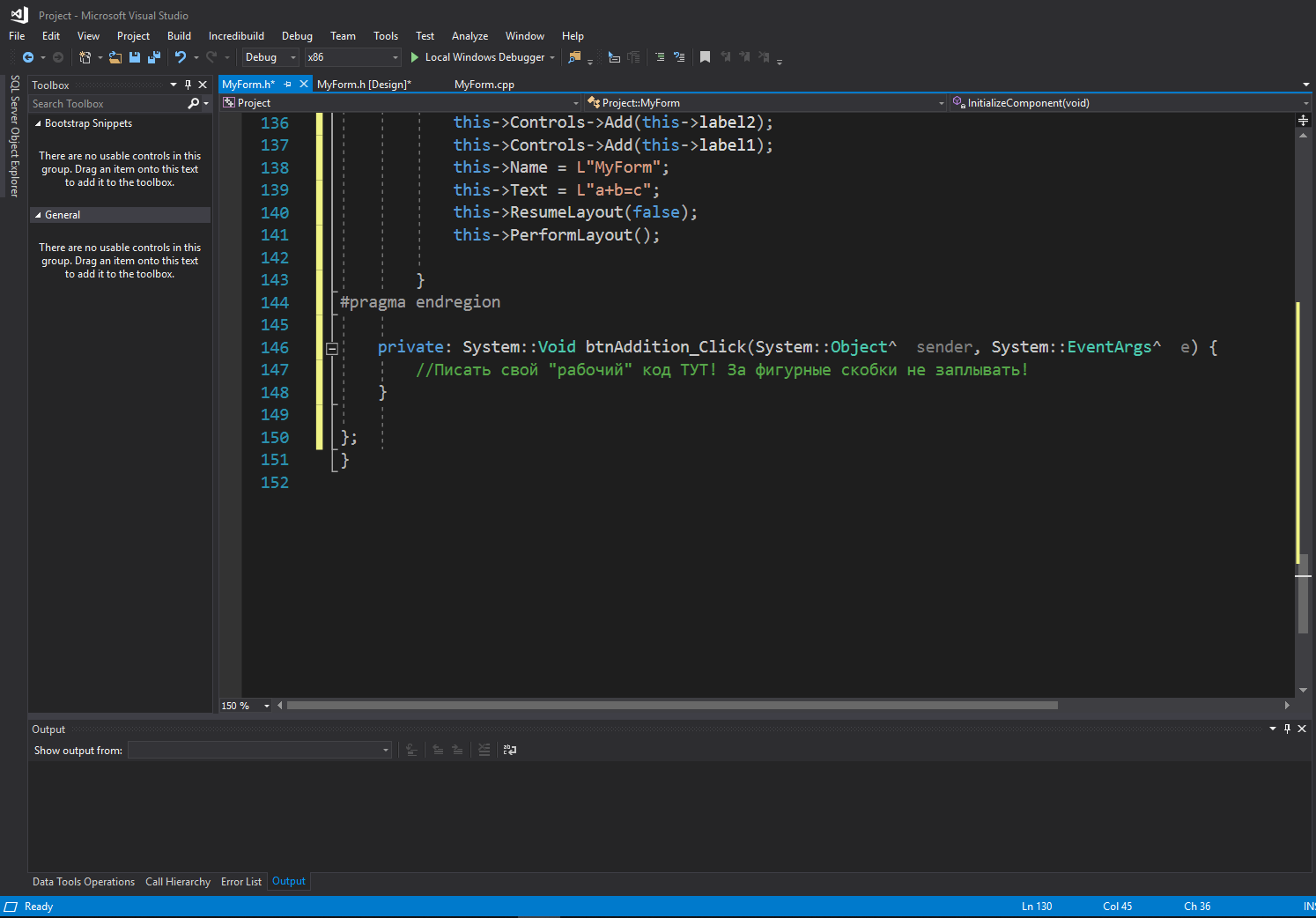
Ну , на этом оформление нашей формы можем считать завершённым. Приступаем непосредственно к программированию.

Написание кода основной программы

Если вы сейчас запустите форму и нажмете на кнопку , то ничего не произойдет. Чтобы наша кнопка при нажатии выводила в поле tbSum сумму содержимого tbFirstNumber и tbSecondNumber на нужно создать обработчик события «нажатия» для этой кнопки в котором будет описан алгоритм действий который должен быть выполнен если кто-то нажал на эту кнопку. Обработчик «нажатия» делается очень просто, нужно 2 раза нажать на нашу кнопку в конструкторе.



После этого нас сразу перенесёт к нашему методу обработки события «нажатия»:



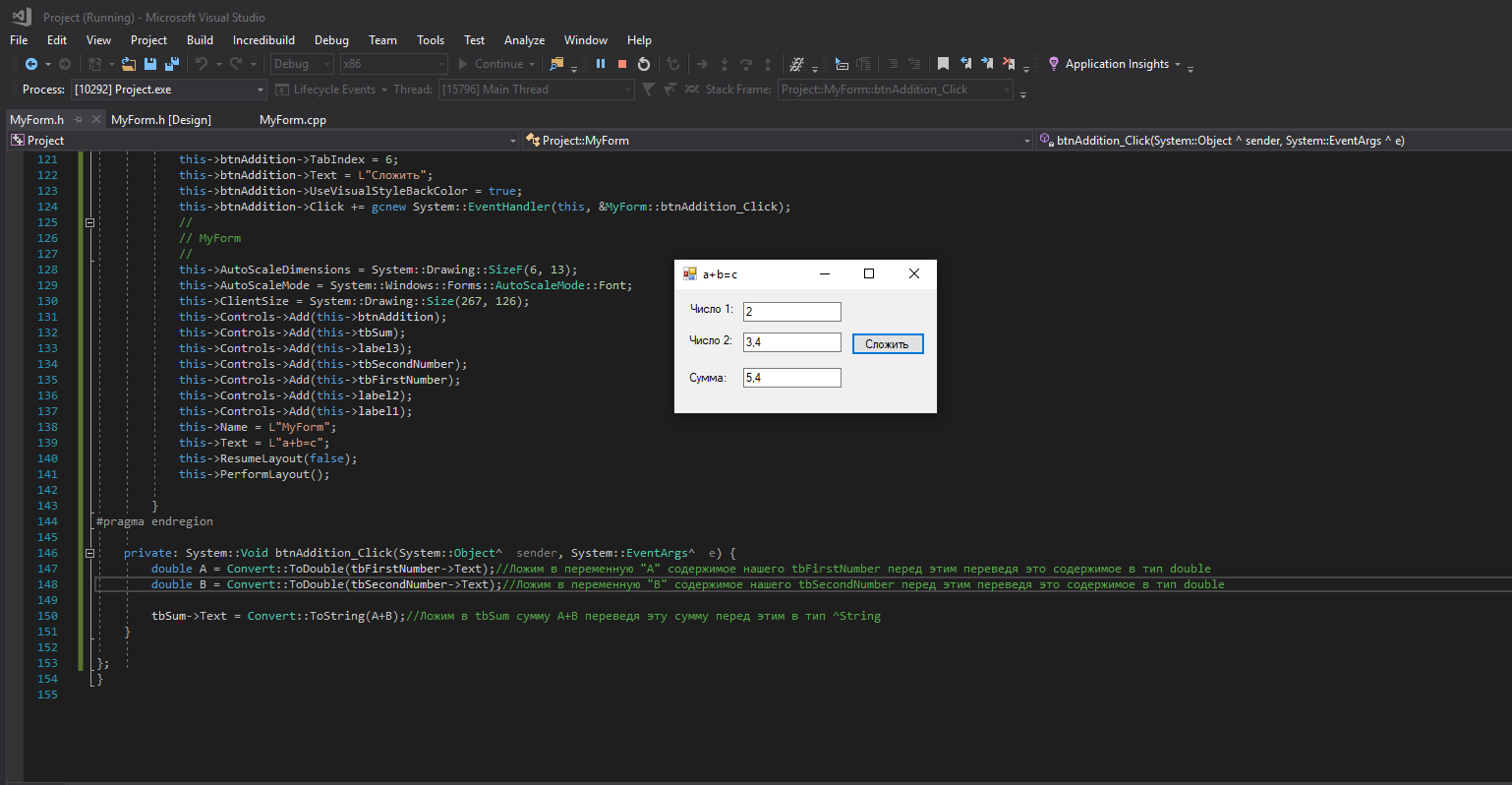
Теперь давайте добавим туда код который будет выводить в поле tbSum сумму содержимого tbFirstNumber и tbSecondNumber:

double A = Convert::ToDouble(tbFirstNumber->Text);//Передаём в переменную "A" содержимое нашего tbFirstNumber перед этим переведя это содержимое в тип double

double B = Convert::ToDouble(tbSecondNumber->Text);// Передаём в переменную "B" содержимое нашего tbSecondNumber перед этим переведя это содержимое в тип double

tbSum->Text = Convert::ToString(A+B);// Передаём в tbSum сумму A+B переведя эту сумму перед этим в тип ^String

И получаем готовое приложение которое умеет складывать 2 числа.



Часть 2.MessageBox

Давайте модифицируем наш калькулятор , а если конкретней ,то добавим выскакивание окошка с сообщением которое убедится в ваших намерениях запустить программу. Выглядеть окошко будет примерно так:

Но перед этим давайте разберёмся , что же такое MessageBox, что он умеет и для чего он нужен.

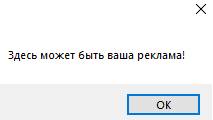
Функция MessageBox отображает модальное диалоговое окно, которое содержит системный значок, набор кнопок и краткое сообщение для конкретного приложения, такое как информация о состоянии или об ошибке. Окно сообщения возвращает целочисленное значение, которое указывает, на какую кнопку нажал пользователь.

Итак, значит MessageBox можно использовать для простого вывода какого-нибудь сообщения или же для подтверждения того или иного действия.

Самый простой вызов MessageBox имеет следующий синтаксис:

MessageBox::Show("Здесь может быть ваша реклама!");

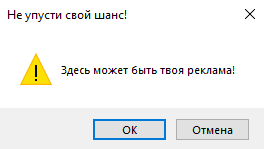
А выглядеть это будет вот так:



Более сложный пример вызова MessageBox будет иметь следующий синтаксис :

MessageBox::Show("Здесь может быть твоя реклама!", "Не упусти свой шанс!", MessageBoxButtons::OKCancel, MessageBoxIcon::Exclamation);

А выглядеть это будет вот так:



Как мы видим у нас появился заголовок , иконка и новая кнопка.

MessageBox может принимать в качестве аргументов 4 параметра. Первый параметр принимает основной текст, который нужно вывести, второй параметр принимает заголовок нашего MessageBox , третий параметр принимает кнопки которые должны быть в нашем MessageBox, а четвертый принимает иконку которую надо вывести в MessageBox.

Как мы видим у нас в MessageBox есть 2 кнопки «Ок» и «Отмена» на которые пользователь может нажать и в зависимости от того , какую кнопку нажал пользователь должно произойти какое-нибудь действие.

Как вы читали ранее функция MessageBox возвращает целочисленное значение, которое указывает, на какую кнопку нажал пользователь.

Давайте напишем код который при нажатии на кнопку «Ok» выведет MessageBox с текстом «Ok» , если пользователь нажал на «Отмена» то выводился бы MessageBox с текстом «Пока».

System::Windows::Forms::DialogResult mb = MessageBox::Show("А вы точно хотите работать в этой программе?", "Подумай", MessageBoxButtons::YesNo, MessageBoxIcon::Question);

if (mb == System::Windows::Forms::DialogResult::Yes) //Если нажата кнопка с положительным ответом

{

MessageBox::Show("OK");

}

else if(System::Windows::Forms::DialogResult::No == mb) //Если нажата кнопка с отрицательным ответом

{

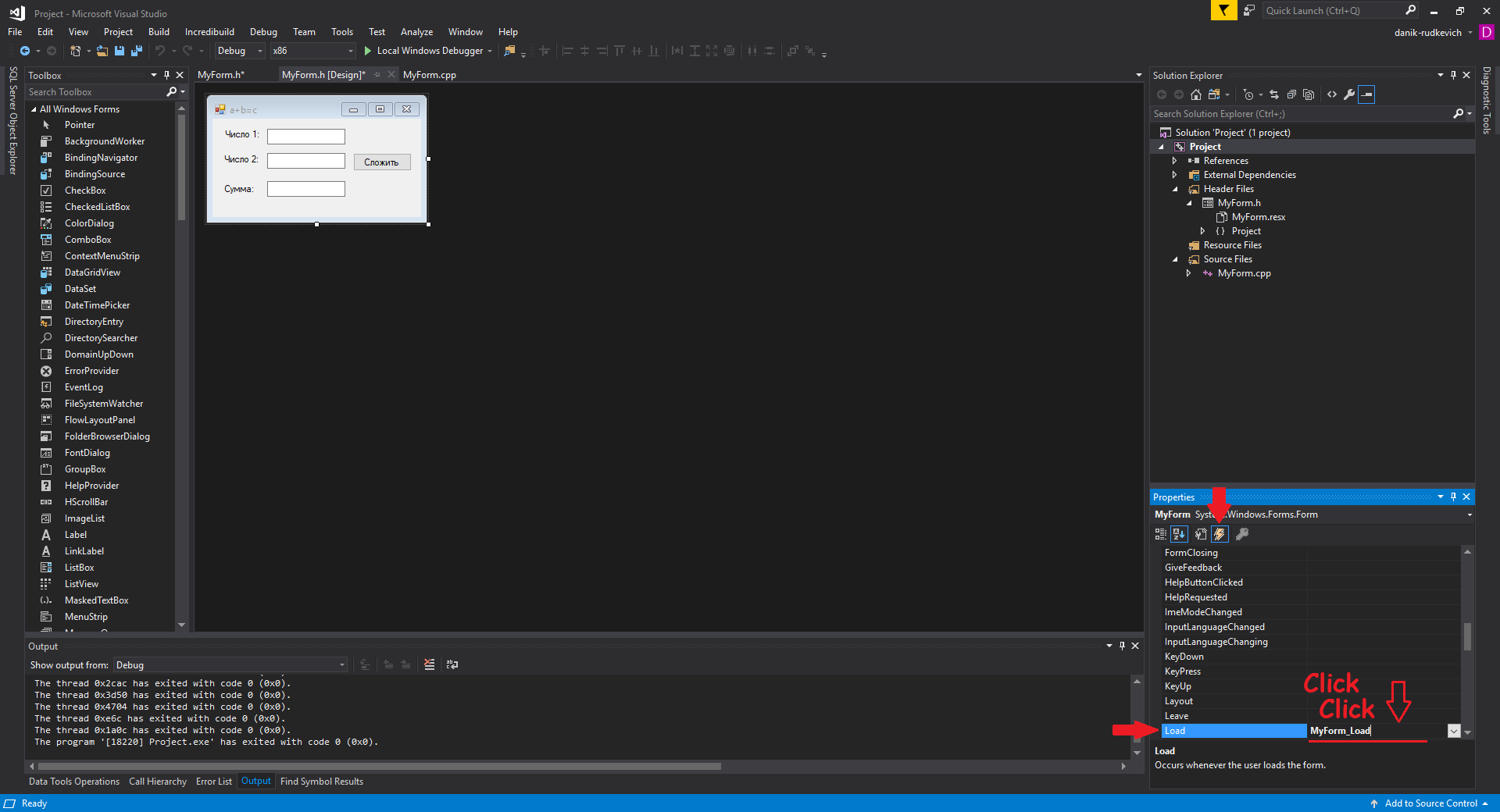
MessageBox::Show("Пока");

this->Close();//Закрываем программу

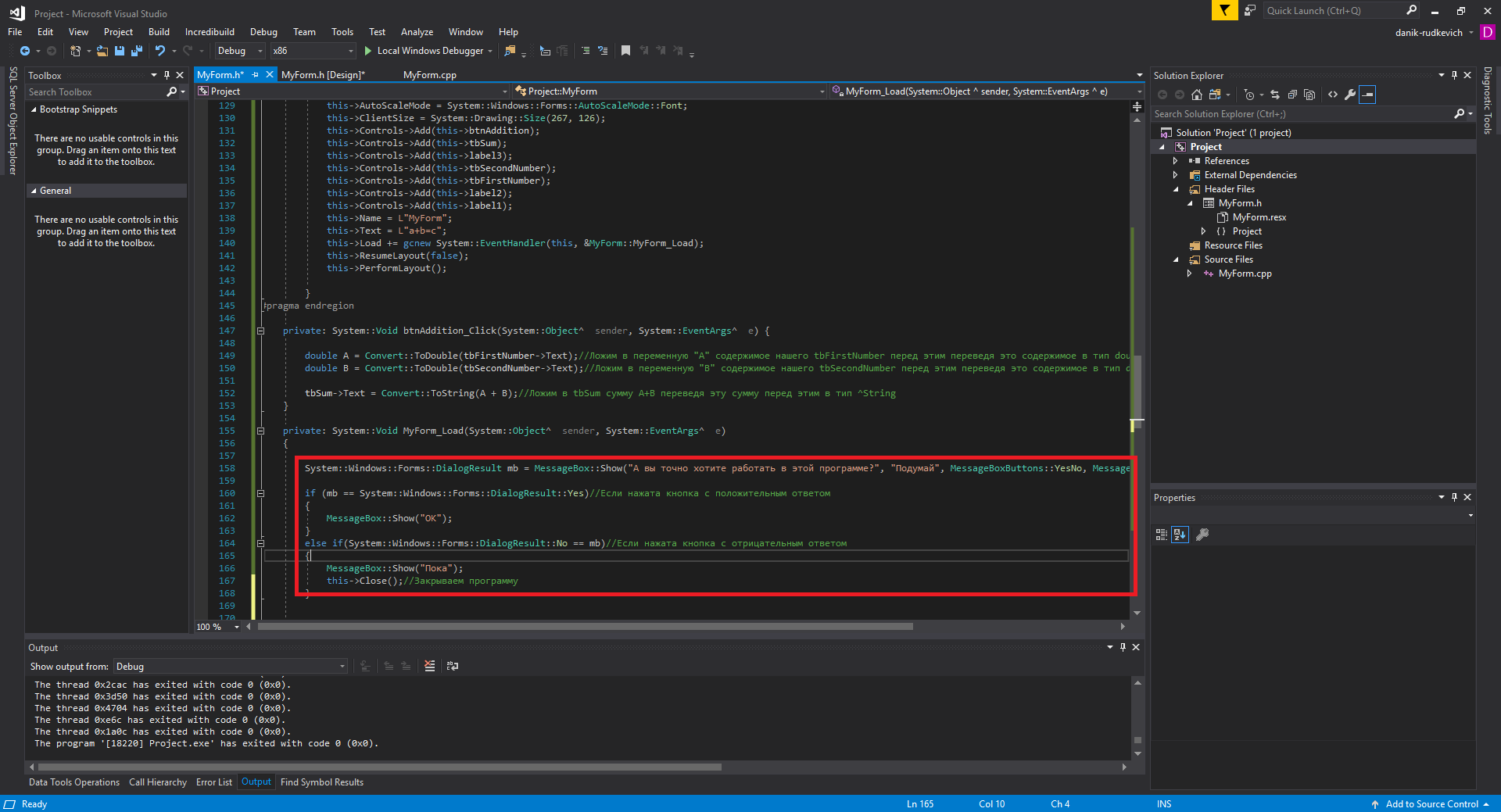
}

Переменна «mb» принимает в себя результат нажатия кнопки в MessageBox.Теперь в mb хранится информация о том , какая кнопка была нажата и мы теперь можем использовать if и else.

Теперь нужно сделать так , чтобы этот MessageBox появлялся сразу же при запуске нашей программы. Для этого давайте добавим обработчик события «Запуска» формы. Делается это очень просто при помощи двойного нажатия мышью по полю напротив необходимого события.



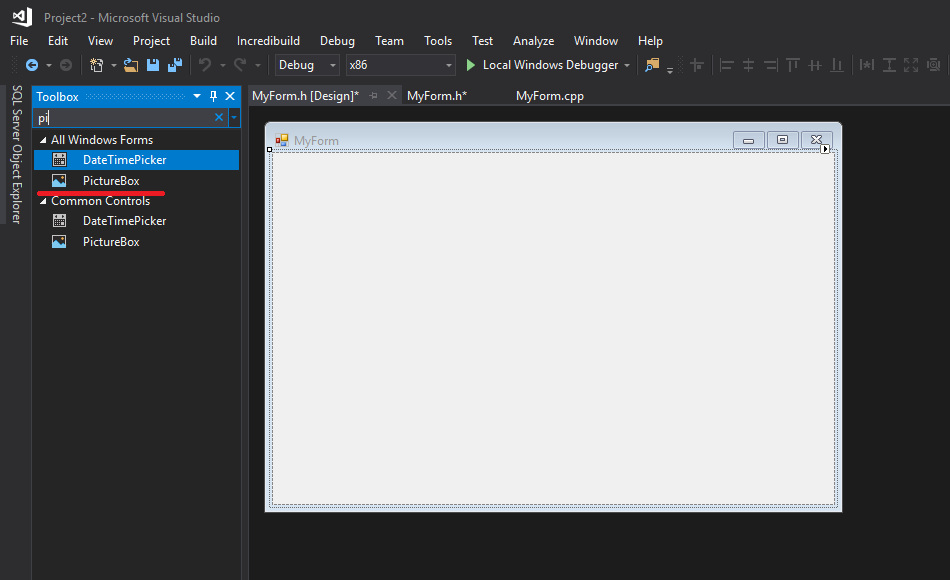
После чего создастся обработчик события «Запуска» который будет вызываться при запуске нашей формы. Поместим туда код нашего MessageBox и посмотрим на результат.



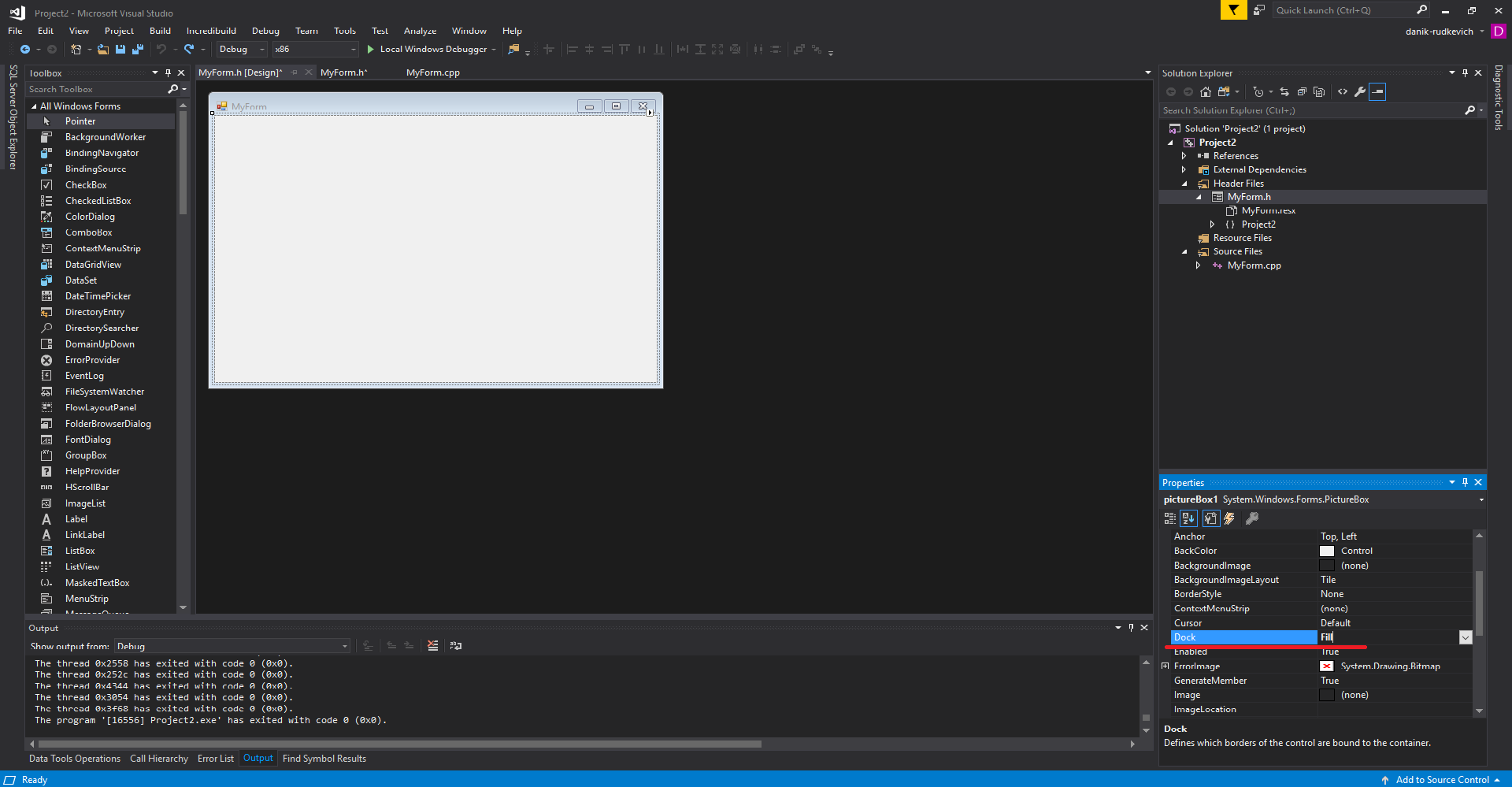
Теперь наш MessageBox появляется до запуска основного окна.Для закрепления полученного материала добавьте ещё как минимум 5 MessageBox в которых у пользователя будут спрашивать действительно ли он хочет запустить данную программу и если хоть один раз пользователь нажал на кнопку которая подтверждает его отказ , то программа должна закрыться.

Часть 3. Рисование

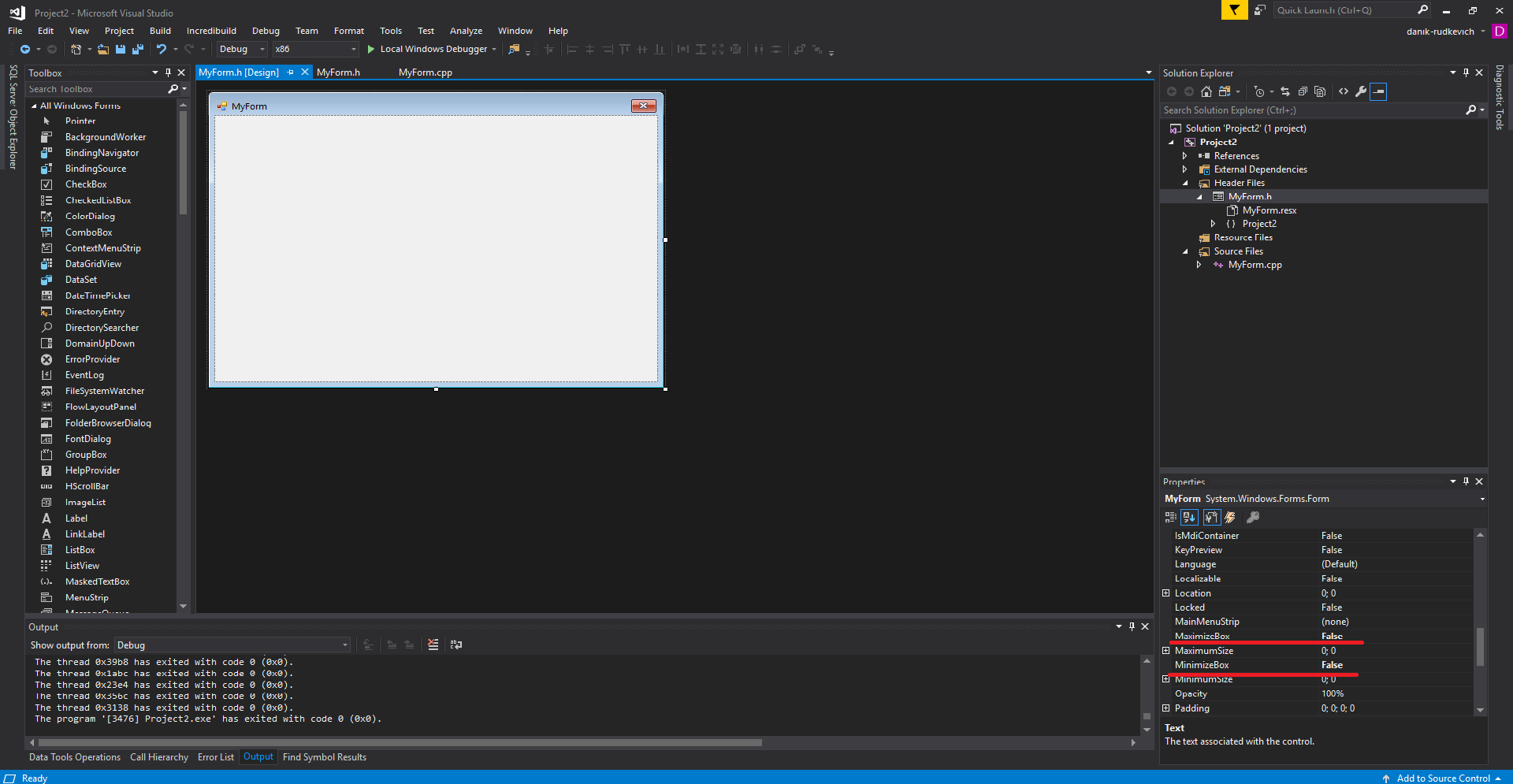
Давайте научимся рисовать в нашей форме для этого создадим новый проект и поместим туда элемент pictureBox



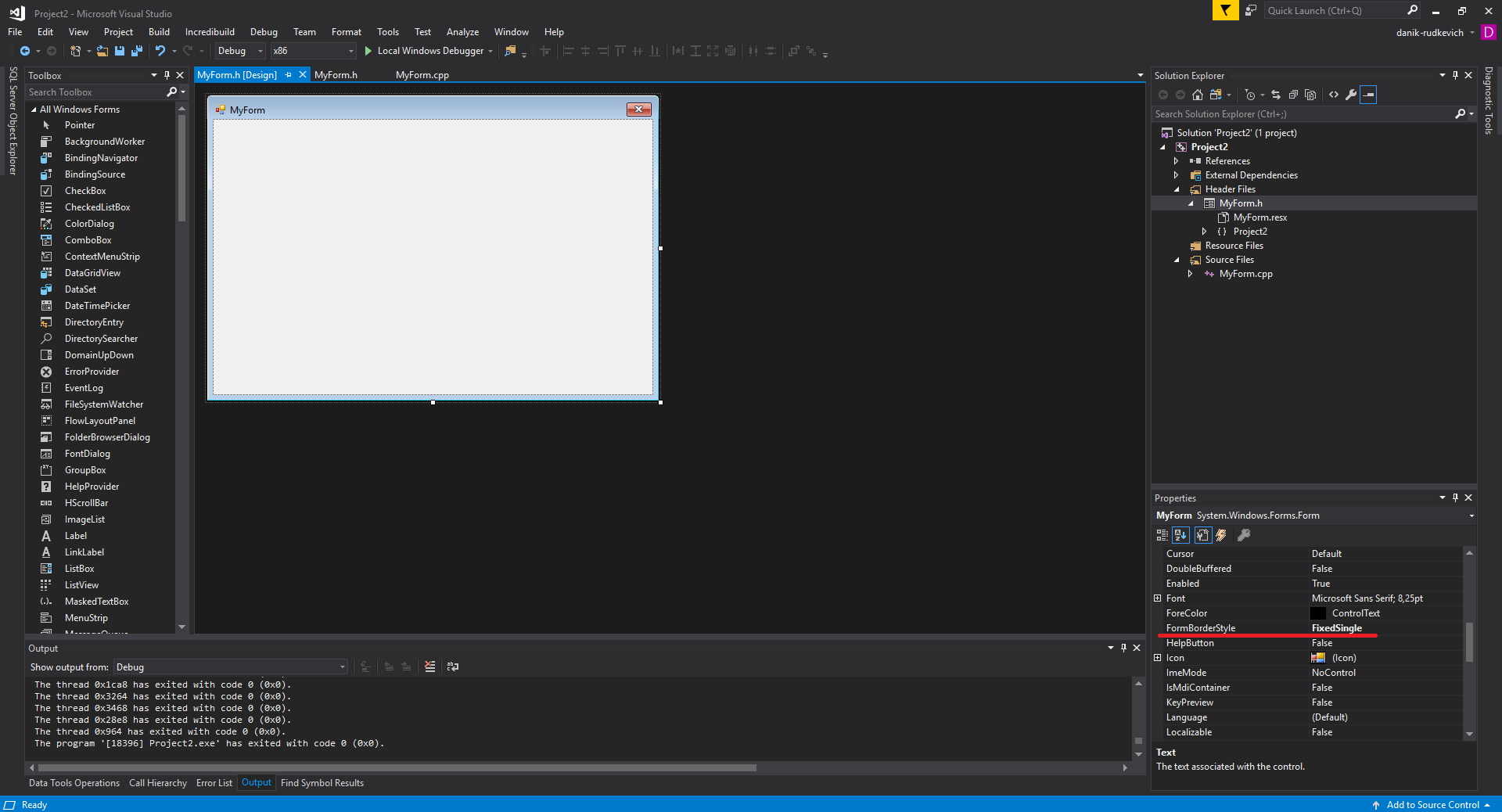
После этого растянем его по ширине всей формы



Теперь отключим нашей форме способность изменять размер.Для начала отключим кнопки «развертывания» и «свертывания».

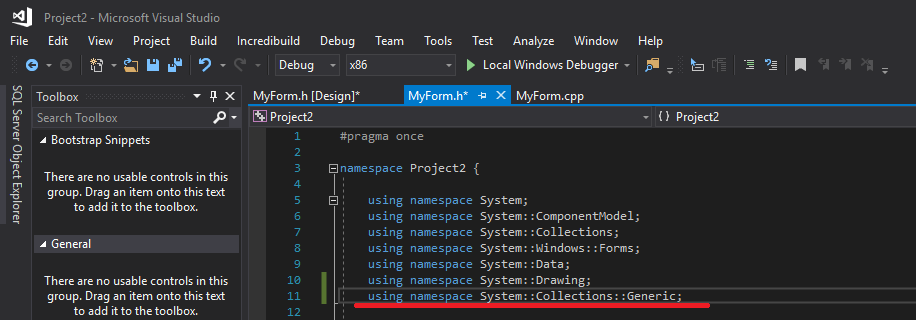


Теперь установим параметр FixedSingle для FormBorderStyle чтобы запретить изменение размеров нашей формы.



Теперь приступим к написанию нашего кода. Для начало нужно подключить ещё одно пространство имён в MyForm.h:  
using namespace System::Collections::Generic;

Оно нужно для использования в проекте списка точек (Point) шаблонного класса List.

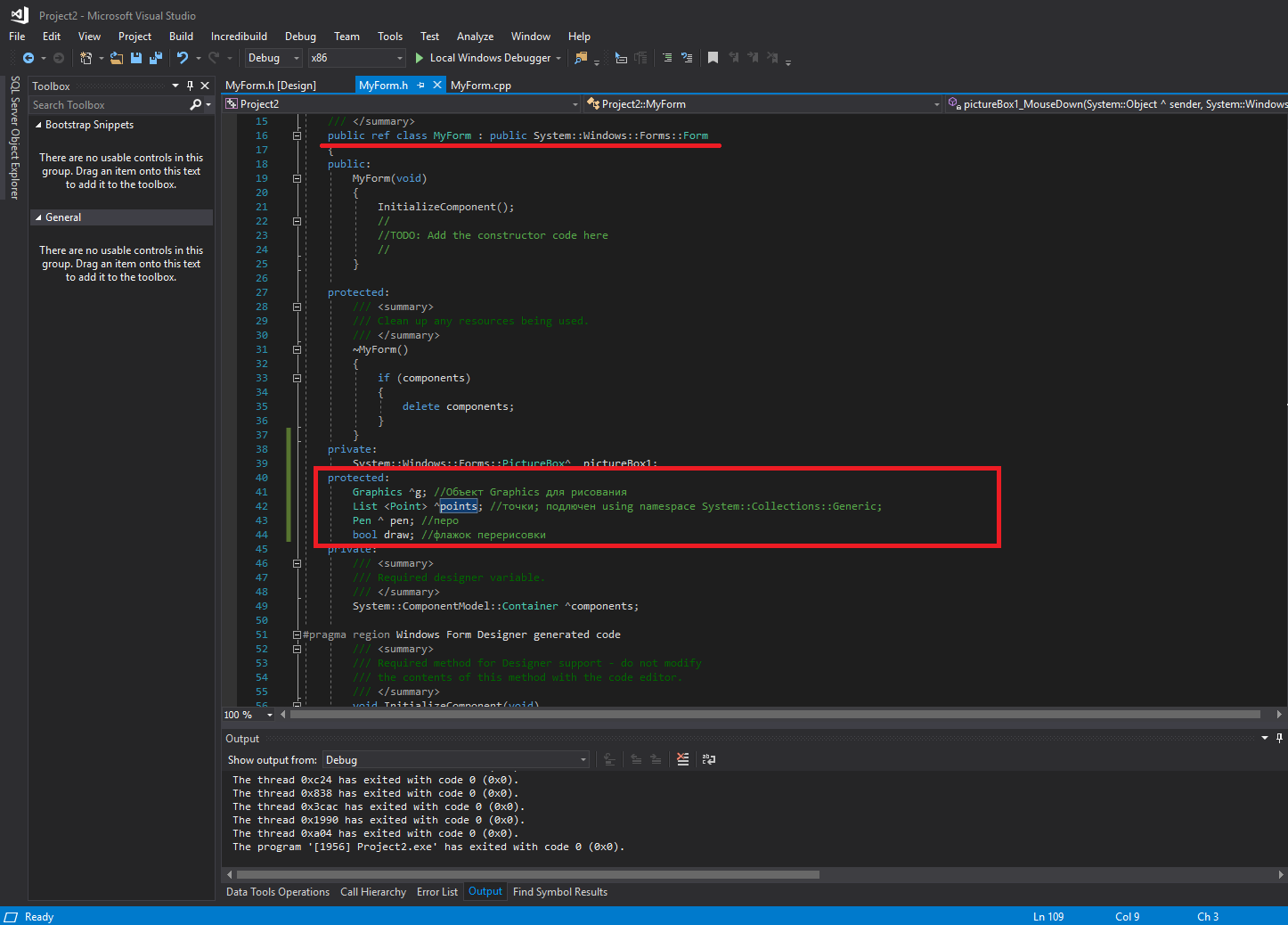


Теперь нужно добавить необходимые переменные в нашу форму:  
Graphics ^g; //Объект Graphics для рисования

List <Point> ^points; //точки; подлючен using namespace System::Collections::Generic;

Pen ^ pen; //перо

bool draw; //флажок перерисовки



Теперь добавим необходимые обработчики событий.

1)Событие «load» для нашей формы. В него вставляем код:

Bitmap ^ Img = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height);//Переменная в которой будет формироваться наше изображение

g = Graphics::FromImage(Img);//Создаём новую графику из задоного изображения

pictureBox1->Image = Img;//Ложим наше изображение в pictureBox1

pen = gcnew Pen(Color::Red);//Устанавливаем цвет пера

points = gcnew List <Point>();//Создаем наш массив для хранения точек

draw = false;//Отключаем рисование

2)Событие «MouseDown» для нашего pictureBox. В него вставляем код:

points->Clear();//Очищаем массив points благодоря чему мы можем добалять новые линии которые не будут связаны со старыми

points->Add(e->Location);//Помещаем в массив координаты точки

draw = true;//Разрешаем рисование

pictureBox1->Invalidate();//Перерисовываем pictureBox1

3)Событие «MouseUp» для нашего pictureBox. В него вставляем код:

draw = false;//Заканчиваем рисование

pictureBox1->Invalidate();//Перерисовываем pictureBox1

4)Событие «MouseMove» для нашего pictureBox. В него вставляем код:

if (draw && points->Contains(Point(e->X, e->Y)) == false)

{

//если включён флажок и текущая точка не содержится в списке точек

if (e->Button == System::Windows::Forms::MouseButtons::Left)

{

//если нажата левая кнопка

points->Add(e->Location);//Помещаем в массив координаты точки

pictureBox1->Invalidate();//Перерисовываем pictureBox1

}

}

5)Событие «Paint» для нашего pictureBox. В него вставляем код:

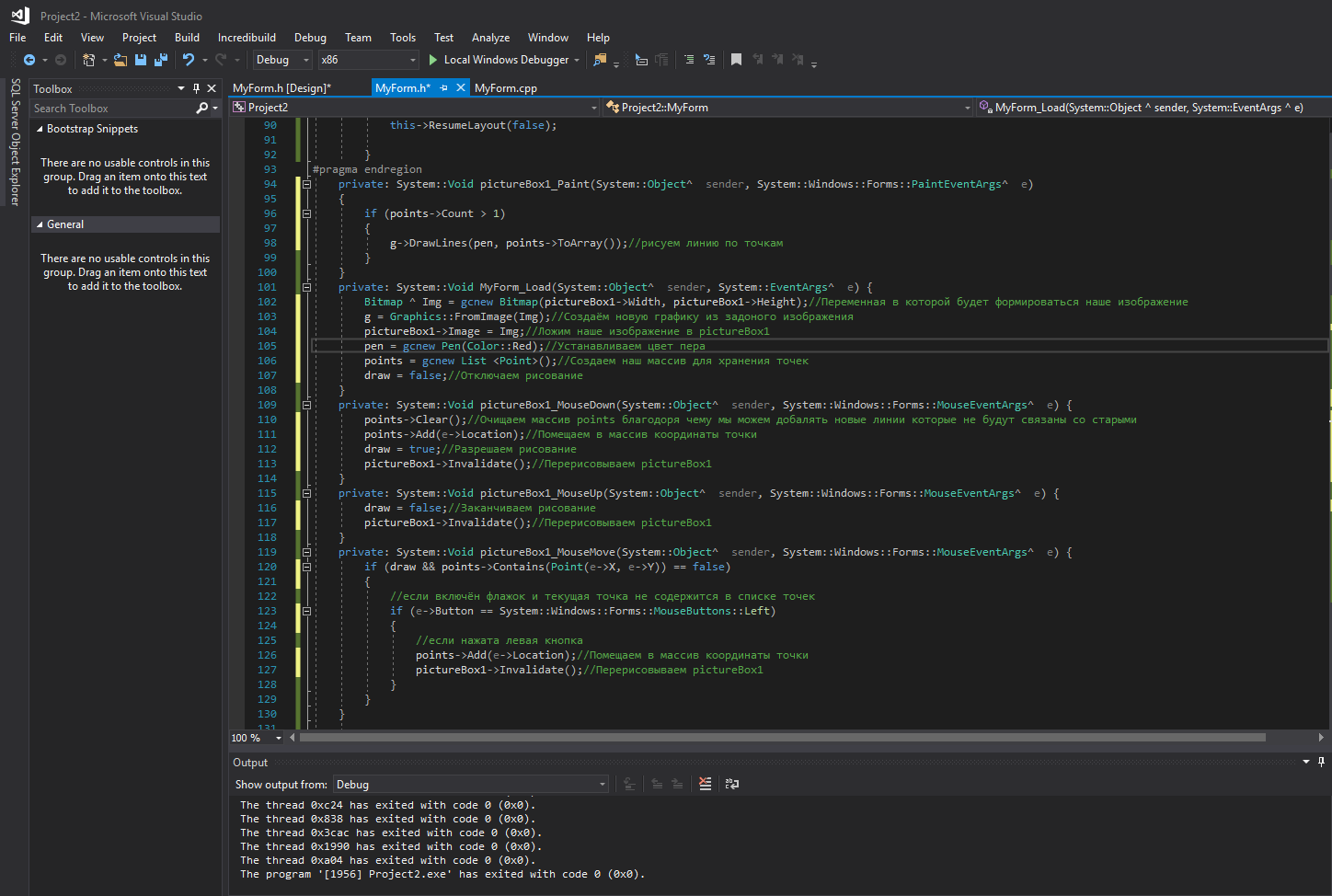
if (points->Count > 1)

{

g->DrawLines(pen, points->ToArray());//рисуем линию по точкам

}

В итоге у нас должно получится что-то такое:



По желанию можете добавить ещё 2 события которые будут очищать наш холст от старых рисунков и событие для установки нового цвета пера. Для очистки холста я использовал событие «DoubleClick» для pictureBox,а для установки нового цвета пера событие «KeyPress» для формы.

Код обработчика события «DoubleClick»:

Bitmap ^ Img = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height);//Переменная в которой будет формироваться наше изображение

g = Graphics::FromImage(Img);//Создаём новую графику из задоного изображения

pictureBox1->Image = Img;//Ложим наше изображение в pictureBox1

Код обработчика события «KeyPress»:

switch (e->KeyChar)

{

case '1':

{

pen = gcnew Pen(Color::Black);

break;

}

case '2':

{

pen = gcnew Pen(Color::Blue);

break;

}

case '3':

{

pen = gcnew Pen(Color::Green);

break;

}

case '4':

{

pen = gcnew Pen(Color::Yellow);

break;

}

case '5':

{

pen = gcnew Pen(Color::Brown);

break;

}

default:

pen = gcnew Pen(Color::Red);

break;

}

Ну и в качестве домашнего задания нарисуйте портрет вашего преподавателя в этой форме.

Часть 4. Обработка события нажатия клавиши

Событие нажатия «KeyPress» уже был затронут в предыдущей теме , но сейчас мы разберём его и другие события связанные с нажатием клавиш более подробно.

Событие «KeyPress» вызывается при нажатии на клавишу клавиатуры.

Событие «KeyDown» вызывается при нажатии на клавишу клавиатуры.

Событие «KeyUp» вызывается после нажатия на кнопку.

Прочитав характеристику каждого события возникает вполне логичный вопрос. В чём отличие между «KeyPress» и «KeyDown»? Единственное практическое различие между «KeyDown» и «KeyPress» заключается в том, что «KeyPress» передает символ, полученный в результате нажатия клавиши, и вызывается только в том случае, если он есть. Другими словами при нажатии на «Ctrl» , «Shift» , «Alt» , «f1-f12» , «Клавиши со стрелками» событие «KeyPress» не вызывается , а событие «KeyDown» вызывается. Второй вопрос который может у вас возникнуть – что значит «вызывается после нажатия на кнопку» в событии «KeyUp». Это значит что если вы добавили своему объекту события «KeyPress» или «KeyDown» и событие «KeyUp» , то сначала вызовется событие «KeyPress» или «KeyDown» , а потом событие «KeyUp».

Давайте модифицируем нашу первую программу которая складывает 2 числа и запретим ввод не корректных данных типа букв и тд. , а так же добавим событие которое закроет нашу программу при нажатии «Ctrl»+c.

Для предотвращения ввода не корректных данных добавим каждому нашему textbox событие «KeyPress» и вставим туда этот код:

//Объект "e" хранит в себе информацию о нажатой клавиши.В поле KeyChar обекта "e" хранится символ введённой с клавиатуры.

if ((e->KeyChar <= 47 || e->KeyChar >= 58) && e->KeyChar != 8 && e->KeyChar != ',' && e->KeyChar !='-') //Если введённый символ не принадлежит диапазону цифр (e->KeyChar <= 47 || e->KeyChar >= 58) и не является запятой,минусом или Backspace , то

{

e->Handled = true;//Отменяем передачу введённого с клавиатуры символа в textbox

}

//Если условие не сработало , значит введённый символ помещается в textbox

Теперь давайте научим нашу программу выключаться если пользователь введёт комбинацию «Ctrl»+c.Для этого нам нужно создать обработчик события «KeyPress» для нашей формы.

Но почему мы создаем обработчик для «KeyPress» если событие «KeyPress» не вызывается при нажатии на «Ctrl»?

Всё верно , «KeyPress» по прежнему не может среагировать на нажатие кнопки «Ctrl» так как у «Ctrl» нет своего кода в ASCII table , а вот у комбинации «Ctrl»+c свой код в ASCII table есть!

Коды с 1 по 26 являются кодами комбинаций от «Ctrl»+a до «Ctrl»+z.Комбинаций для «Ctrl»+1 , «Ctrl»+0 и тд. не существует , поэтому «KeyPress» на них просто не среагирует.

А вот с комбинацией «Shift»+символ уже интересней. При нажатии этой комбинации «KeyPress» вернёт нам код символа который вводился бы при применении «Shift»+символ. К примеру при вводе «Shift»+f «KeyPress» поймает символ ‘F’ , при вводе «Shift»+1 поймает символ ‘!’ (При включённом **Caps Lock** результат меняется на противоположный!).

Теперь перейдем к практическому применению полученных знаний и напишем код для события «KeyPress» нашей формы:

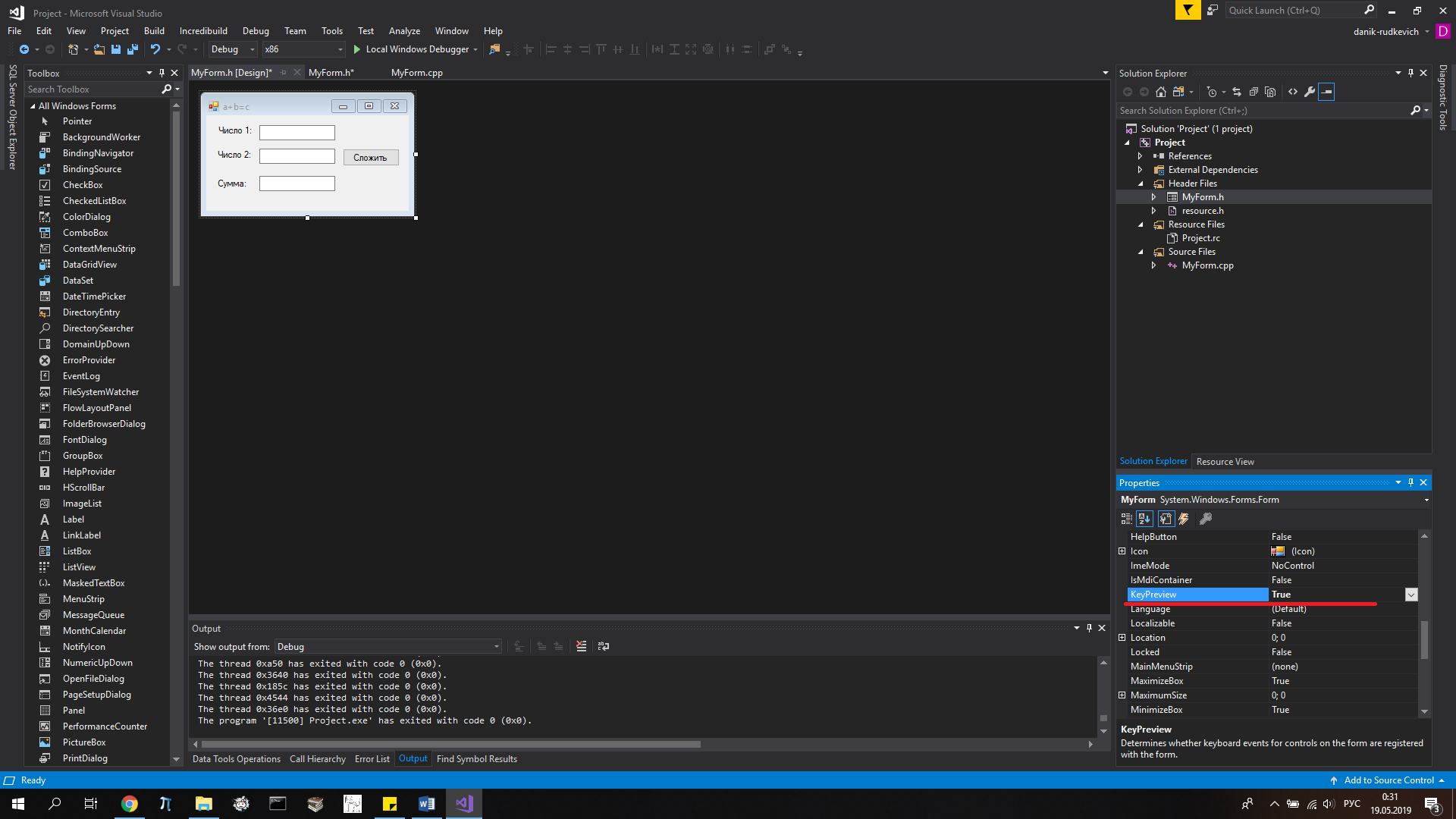
if (e->KeyChar == 3)//Если мы поймали нажатие «Ctrl»+c , то

{

this->Close();//Закрываем нашу форму.

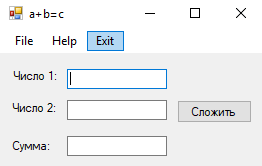
}

Для того чтобы событие «KeyPress» выполнялось не только тогда когда фокус стоит на нашей форме , нам нужно свойству формы KeyPreview установить значение True

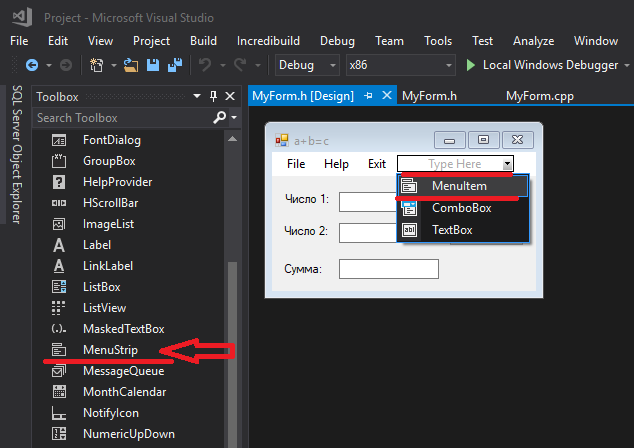


Часть 5. Работа с меню

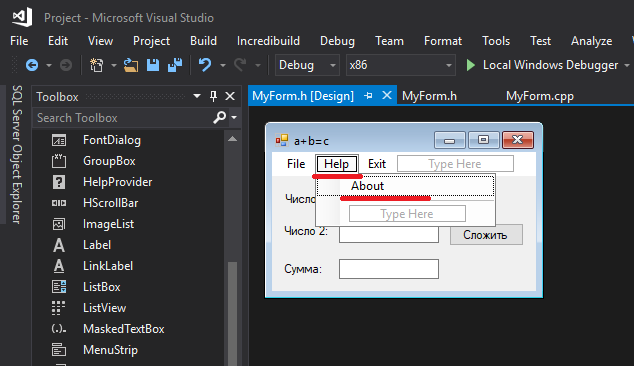
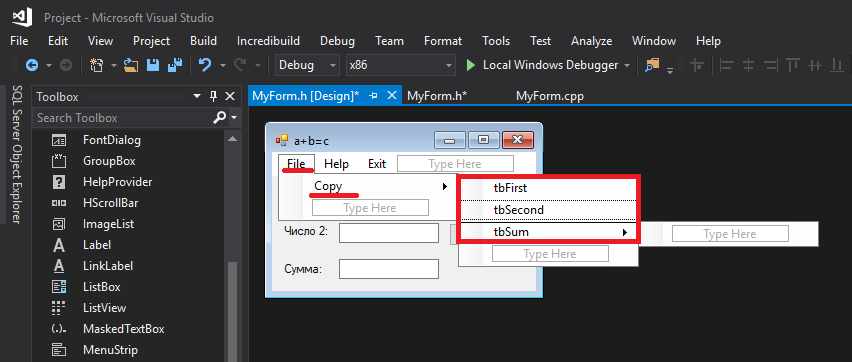
Для общего развития и более красивого оформления нашей программы добавим нашей форме меню с полезными кнопками.



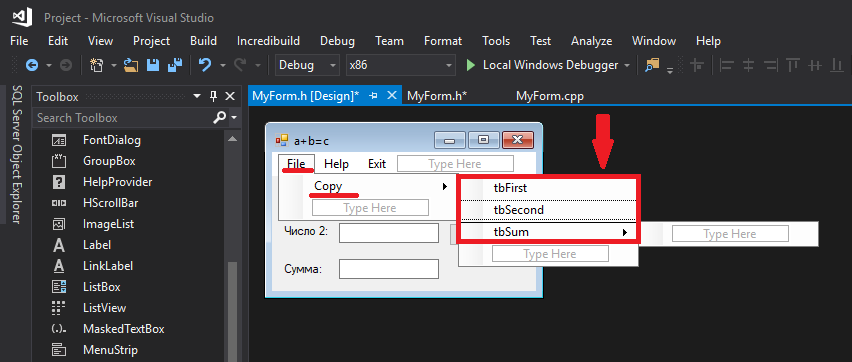
Добавим элемент MenuStrip на нашу форму. Далее в ToolStripMenuItem выбираем тип MenuItem и в поле Text прописываем необходимый текст «File,Help,Exit».



Каждый ToolStripMenuItem может вести ещё на другой ToolStripMenuItem. Создадим подпункты для кнопок File и Help.



Создадим обработчики событий «Click» для кнопок tbFirst,tbSecond,tbSum.



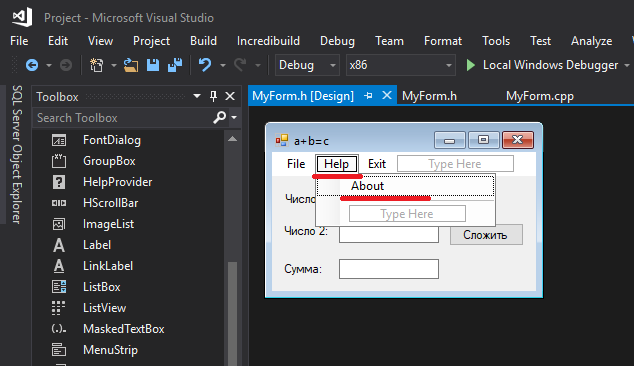
Давайте напишем код для обработчика события «Click» который будет вызываться при нажатии кнопки tbFirst и помещать содержимое tbFirstNumber в буфер :

Clipboard ^clip;//Создаем объект буферa обмена

clip->SetText(tbFirstNumber->Text);//Помещаем в наш буфер содержимое tbFirstNumber

Код для обработчиков события «Click» кнопок tbSecond и tbSum который поместит содержимое tbSum и tbSecondNumber в буфер напишите самостоятельно.

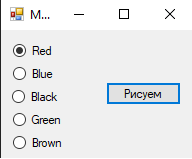
Теперь давайте напишем обработчик события «Click» для кнопки About которое вызовет MessageBox в котором будет выведено имя создателя программы(Вы даже можете вписать туда своё имя!).



Думаю с этим заданием вы так же можете справится самостоятельно ,а так же самостоятельно добавьте обработчик события «Click» для кнопки Exit при нажатие на которую ваша программа завершит работу.

Часть 6. Работа с несколькими формами

Для обучения работы с несколькими формами нужно создать проект с несколькими формами. Создайте новый проект и добавьте туда 2 формы(Одна форма у вас будет основная и запускаться при запуске программы, заранее продумайте её название. В примере название основной формы осталось неизменным MyForm , а второй форме было дано название secondForm).На главной форме разместите 5 элементов radioButton и кнопку.



В главной форме мы будем производить выбор пера для рисования на второй форме при помощи элементов radioButton и вызывать вторую форму в которой будем рисовать.

Давайте создадим обработчик события «Click» для нашей кнопки и напишем там следующий код:

Pen ^ pen;//Создаем наше перо

if (rbRed)//Если rbRed активна , то устанавливаем перу красный цвет

{

pen = gcnew Pen(Color::Red);

}

else if(rbBlack)// Если rbRed активна, то устанавливаем перу черный цвет

{

pen = gcnew Pen(Color::Black);

}

else if (rbBlue)//Если rbRed активна , то устанавливаем перу синий цвет

{

pen = gcnew Pen(Color::Blue);

}

else if (rbBrown)//Если rbRed активна , то устанавливаем перу коричневый цвет

{

pen = gcnew Pen(Color::Brown);

}

else if (rbGreen)//Если rbRed активна , то устанавливаем перу зелёный цвет

{

pen = gcnew Pen(Color::Green);

}

SecondForm ^sf = gcnew SecondForm(pen);//Создаем объект нашей второй формы и передаем в неё наше перо

sf->Show();//Показываем нашу форму

для того чтобы мы могли создать объект класса SecondForm нам нужно его от куда-то взять. А взять его можно из SecondForm.h. Поэтому подключим после #pragma once #include "SecondForm.h" в MyForm.h.

Теперь займемся оформлением второй формы. Установим для параметра FormBorderStyle значение FixedToolWindow у нашей формы. Так же разместим picturebox на нашу форму и установим для параметра Dock значение Fill.Самому классу второй формы добавим 2 новых поля Pen ^pen и Graphics ^g (смотреть Часть 3.Рисование).Теперь создадим обработчик события «Load» для нашей формы напишем там следующий код:

Bitmap ^ Img = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height);//Переменная в которой будет формироваться наше изображение

g = Graphics::FromImage(Img);//Создаём новую графику из задоного изображения

pictureBox1->Image = Img;//Ложим наше изображение в pictureBox1

Далее создадим обработчик события «MouseClick» для нашего pictureBox и вставим туда следующий код:

g->DrawEllipse(pen, e->X-25, e->Y-25, 50, 50);

pictureBox1->Invalidate();

Как вы наверное заметили при создании объекта SecondForm ^sf в MyForm.h мы передаём в него в качестве параметра перо которое было выбрано в Главной форме. Так как конструктора принимающий в качестве параметра данное перо у нас нет ,то нам нужно его добавить в классе SecondForm:

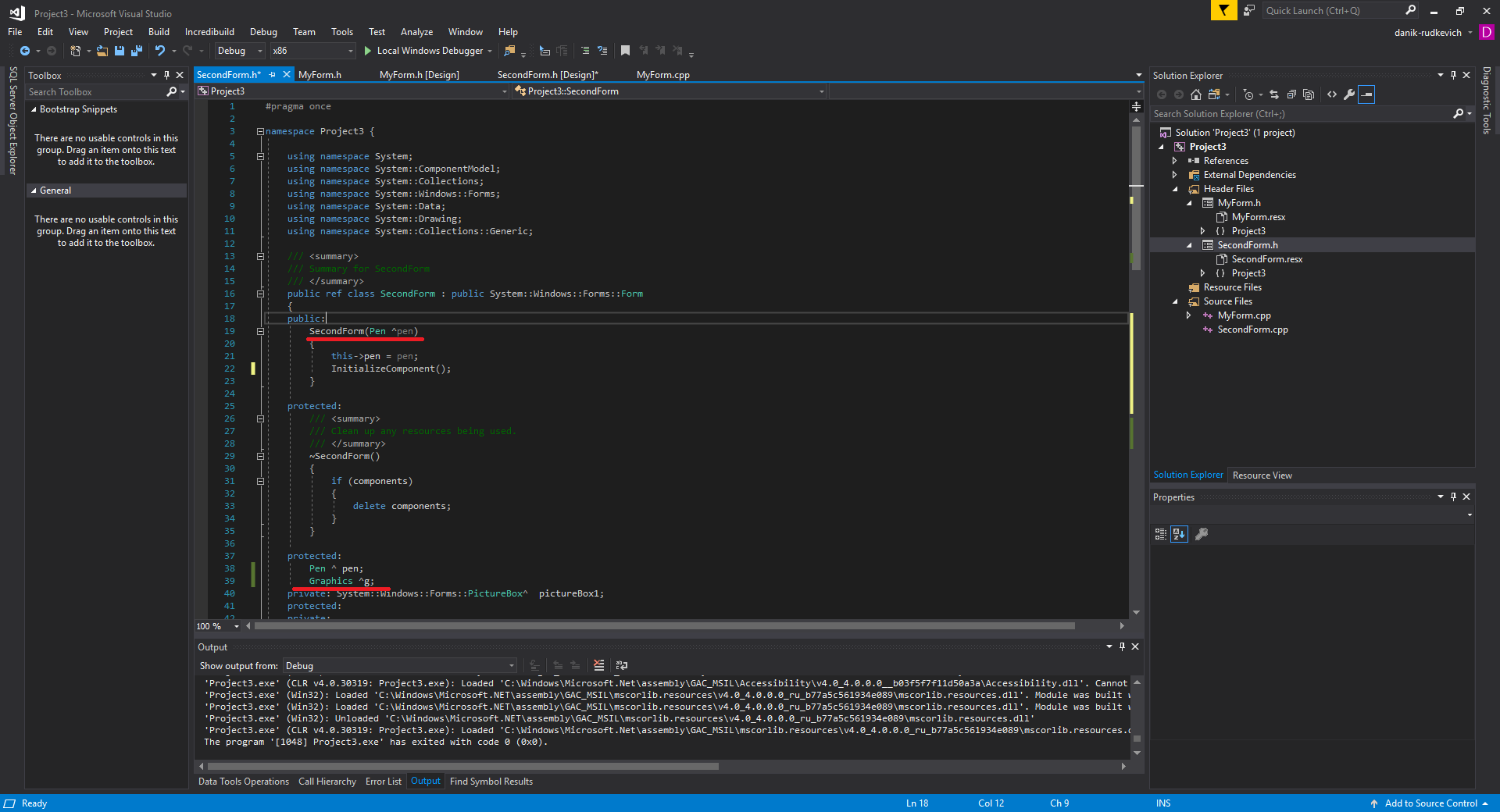
SecondForm(Pen ^pen)

{

this->pen = pen;

InitializeComponent();

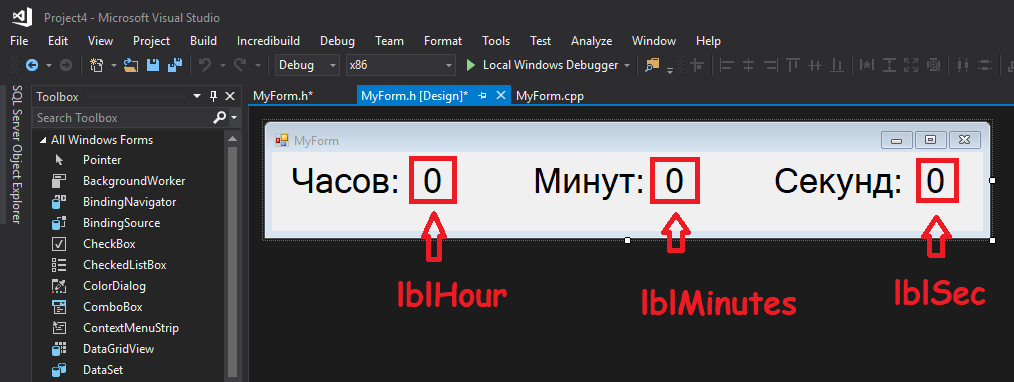
}



В качестве домашнего задания создайте ещё одну форму на которой будут рисоваться квадраты. Переход на форму осуществляется по нажатию другой кнопки из основной формы, цвет пера так же передаётся из главной формы.

Часть 7. Таймер

Создадим новый проект , добавим в него форму , а на форму разместим 6 элементов label,3 будут служить просто для вывода текста «Часов:», «Минут:», «Секунд:» , а остальные 3 будут служить для вывода времени, поэтому мы дадим им нормальные имена lblHour,lblMinutes,lblSec.



Так же добавим на нашу форму элемент timer. Полю Interval элемента timer дадим значение 1000(Интервал у таймера в миллисекундах).

Добавим событие «load» для нашей формы и напишем там следующий код:

time\_t seconds = time(NULL);//Берём текущее время

tm\* timeinfo = localtime(&seconds);//Ложим полученное время в timeinfo

lblHour->Text = Convert::ToString(timeinfo->tm\_hour);//Устанавливаем кол-во часов

lblMinutes->Text = Convert::ToString(timeinfo->tm\_min);//Устанавливаем кол-во минут

lblSec->Text = Convert::ToString(timeinfo->tm\_sec);//Устанавливаем кол-во секунд

Time->Enabled = true;//Включаем таймер

Добавим событие Tick для нашего таймера и напишем там следующий код:

if (Convert::ToInt32(lblSec->Text) + 1 == 60)

{

if (Convert::ToInt32(lblMinutes->Text) + 1 == 60)

{

if (Convert::ToInt32(lblMinutes->Text) + 1 == 24)

{

lblHour->Text = "0";

}

else

{

lblHour->Text = Convert::ToString(Convert::ToInt32(lblHour->Text) + 1);

}

lblMinutes->Text = "0";

}

else

{

lblMinutes->Text = Convert::ToString(Convert::ToInt32(lblMinutes->Text) + 1);

}

lblSec->Text = "0";

}

else

{

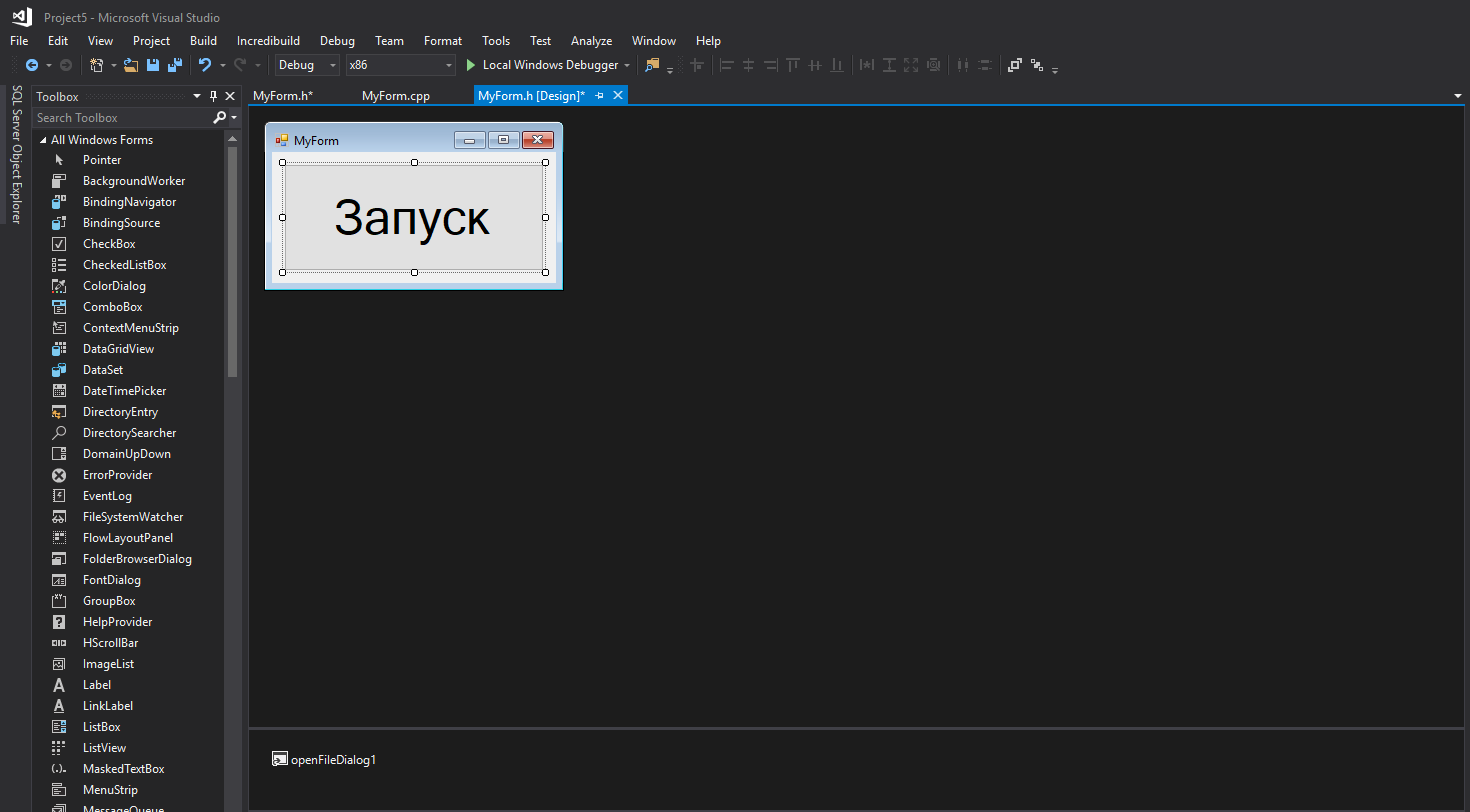
lblSec->Text = Convert::ToString(Convert::ToInt32(lblSec->Text) + 1);

}

В качестве домашнего задания придумайте задачу и реализуйте программу работающую в фоновом режиме.

Часть 8. Звуки

Создадим новый проект , добавим в него форму , а на форму разместим одну кнопку. Так же добавим на нашу форму элемент openFileDialog для того , чтобы мы могли загрузить файл в нашу программу.



Создадим обработчик события «Click» для нашей кнопки и напишем туда следующий код:

if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)//Если мы выбрали файл который будем загружать и нажали на кнопку "Ок", то

{

System::Media::SoundPlayer(openFileDialog1->FileName).Play();//Проигрываем выбранный в openFileDialog файл

}

else

{

System::Media::SystemSounds::Asterisk->Play();//Проигрываем системный звук Asterisk в случаи если мы ничего не выбрали

}

Так как System::Media::SoundPlayer может проигрывать только файлы с расширением .wav , то установим фильтр нашему openFileDialog чтобы он показывал нам сразу файлы только с этим расширением. Для этого в свойствах openFileDialog полю Filter установите значение «Sound file (\*.wav)|\*.wav».

