Практическая работа №2

СозданиеUnit-теставMicrosoftVisualStudio - C++. Пример.

Рассмотрим пошаговый процесс создания простейшего Unit-теста в системе MicrosoftVisualStudio 2010 (C++) для приложения типа CLR ConsoleApplication.

В примере демонстрируется использование класса Assert для проведения тестирования работы функций.

Постановка задачи:

1. Для приложения **CLR Console Application** разработать **Unit-тест**, который тестирует работу функции***Max3()*** - определяет максимальный элемент из трех чисел;числа функция получает входными параметрами.
2. Для функции***Max3()***установить метод тестирования **TestMax()**.
3. Проверить работу функции.

Выполнение:

1. Создание приложение по шаблону **CLR ConsoleApplication.**

Запуск Microsoft Visual Studio.

ВыбратьFile->New Project...

ВыбратьшаблонVisual C++ -> CLR Console Application

Задать Имя проектаMaxApp(можно другое).

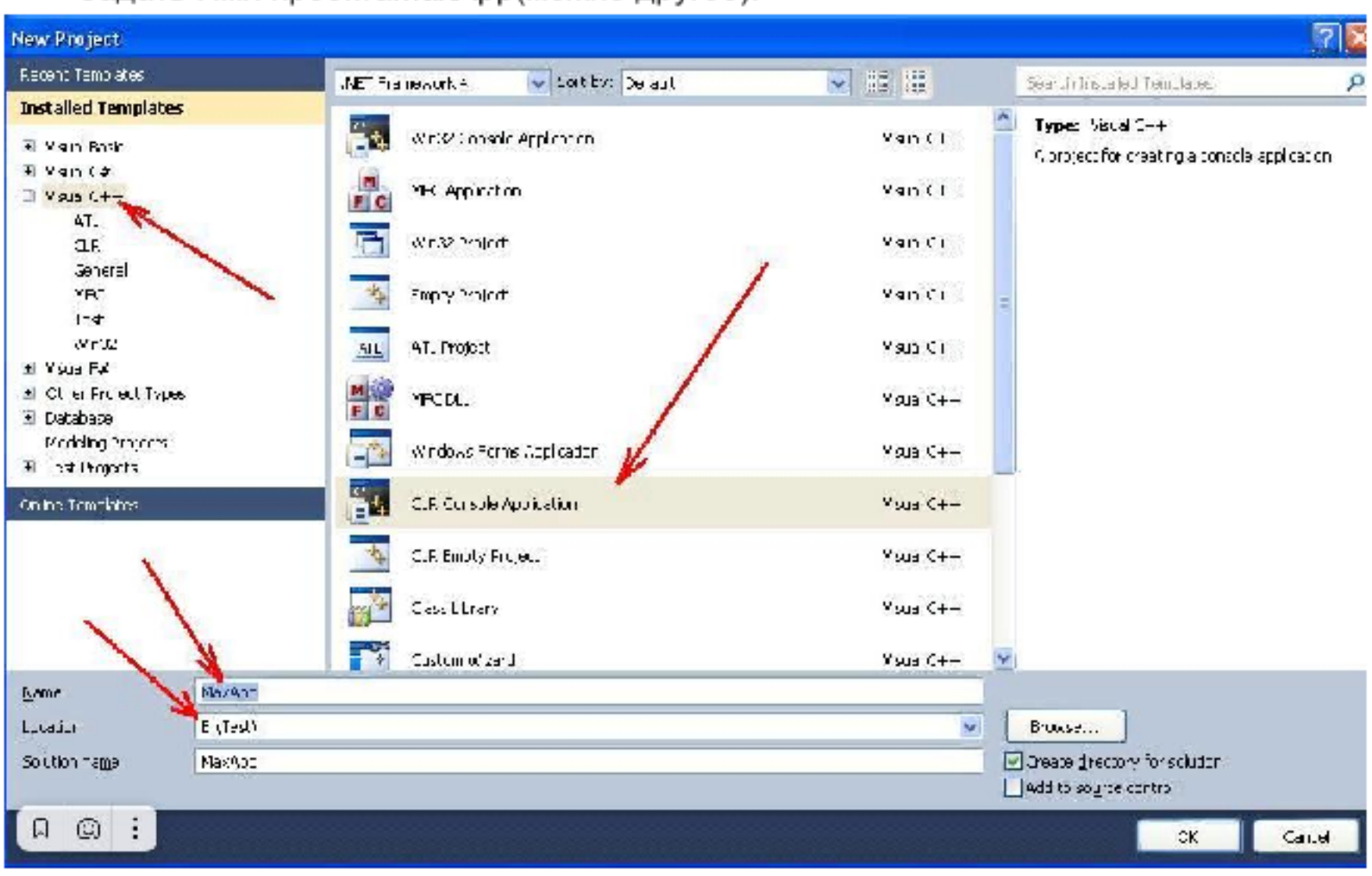

Рис. 1. Создание приложения по шаблону ***CLR ConsoleApplication***

После выбора OK, будет создано приложение типа **CLR ConsoleApplication.**

Окно текстовой части MicrosoftVisualStudio будет иметь приблизительный вид, как показано на рис. 2.

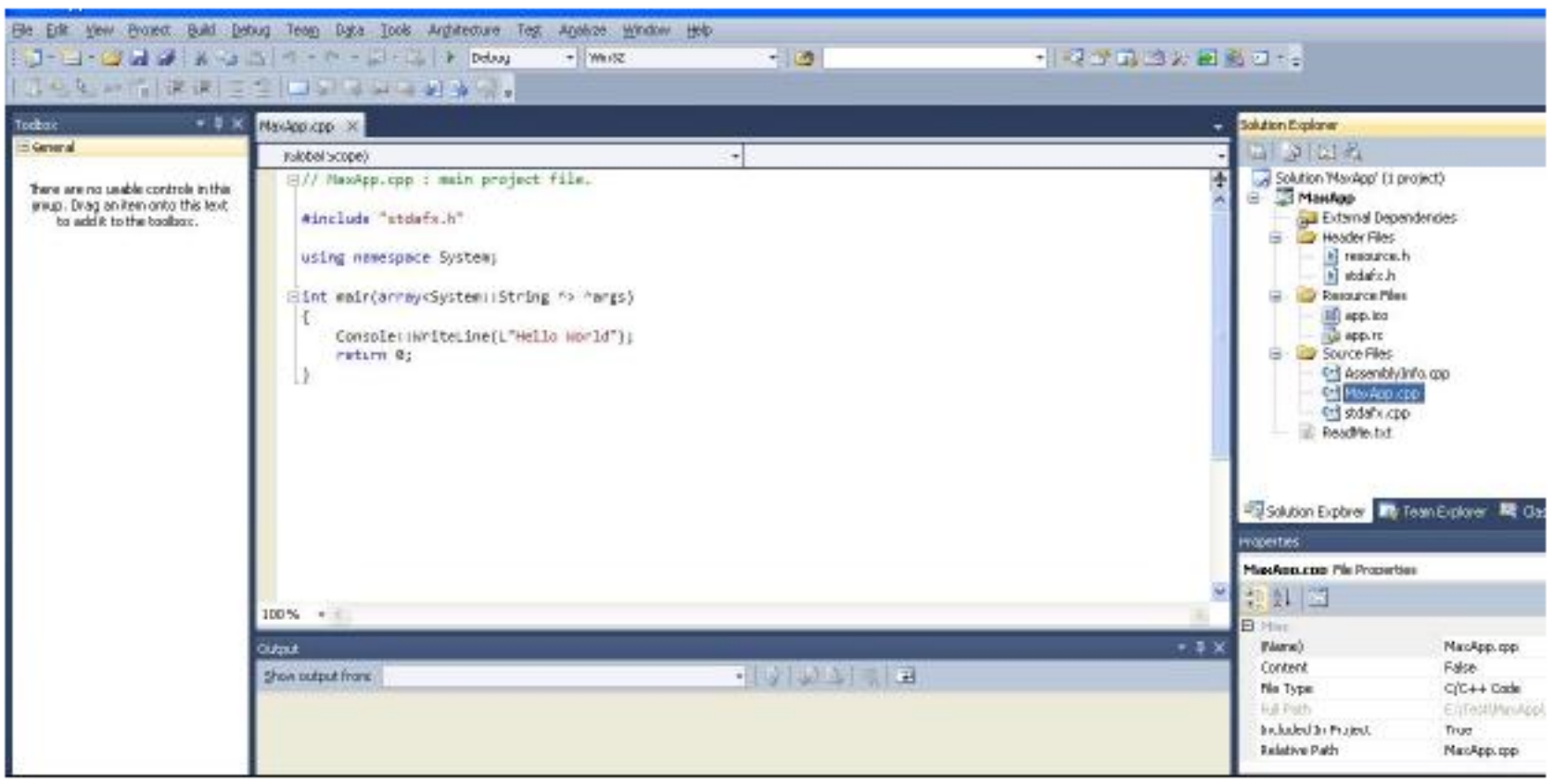

Рис. 2. Вид приложения

Как видно из рис. 2, модуль **MyApp.cpp** содержит подключение модуля **«stdafx.h»**и функцию **main()**, которая есть точкой входа в программу.

2. Реализация функции Max()

Перед объявлением функции main()вводится текст функции **Max()**, который имеет следующий вид:

```
// функция, которая находит максимальное значение между тремя целыми числами
int Max(int a, int b, int c)
{
int max = a;
if (max<b) max = b;
if (max<c) max = c;
returnmax;
}
```

Именно эту функцию нужно протестировать с помощьюUnit-теставMicrosoft Visual Studio.

Текст программы, которую нужно протестировать:

```
// UnitTestApp01.cpp : Defines the entry point for the console application.
// MaxApp.cpp : main project file.
#include "stdafx.h"
usingnamespaceSystem;

// функция, которая находит максимальное значение между тремя целыми числами
int Max(int a, int b, int c)
{
int max = a;
if (max<b) max = b;
if (max<c) max = c;
return max;
```

```
}

int main(array<System::String ^> ^args)
{
    Console::WriteLine(L"Unit-test in MS Visual Studio.");
return 0;
}
```

Поскольку, эта программа будет тестироваться из другого модуля тестирования, то в функции **main()**большее ничего вводить не нужно. Это будет осуществляться из модуля тестирования. Программа готова к тестированию.

3. Создание теста

Тест создается отдельным проектом (**Project**) в решении (**Solution**). Программа, которая будет тестироваться не знает о тесте.

Программа-тест, которая будет тестировать вызывает функции тестируемой программы. В рассматриваемом примере программа-тест будет вызывать функцию:

```
intMax(int,  int,int);
```

3.1. Добавление нового проекта к решению

Для создания теста нужно создать новый проект в решении (**Solution**):

```
File ->Add ->NewProject ...
```

В результате откроется окно«AddNewProject», изображенное на рис. 3.

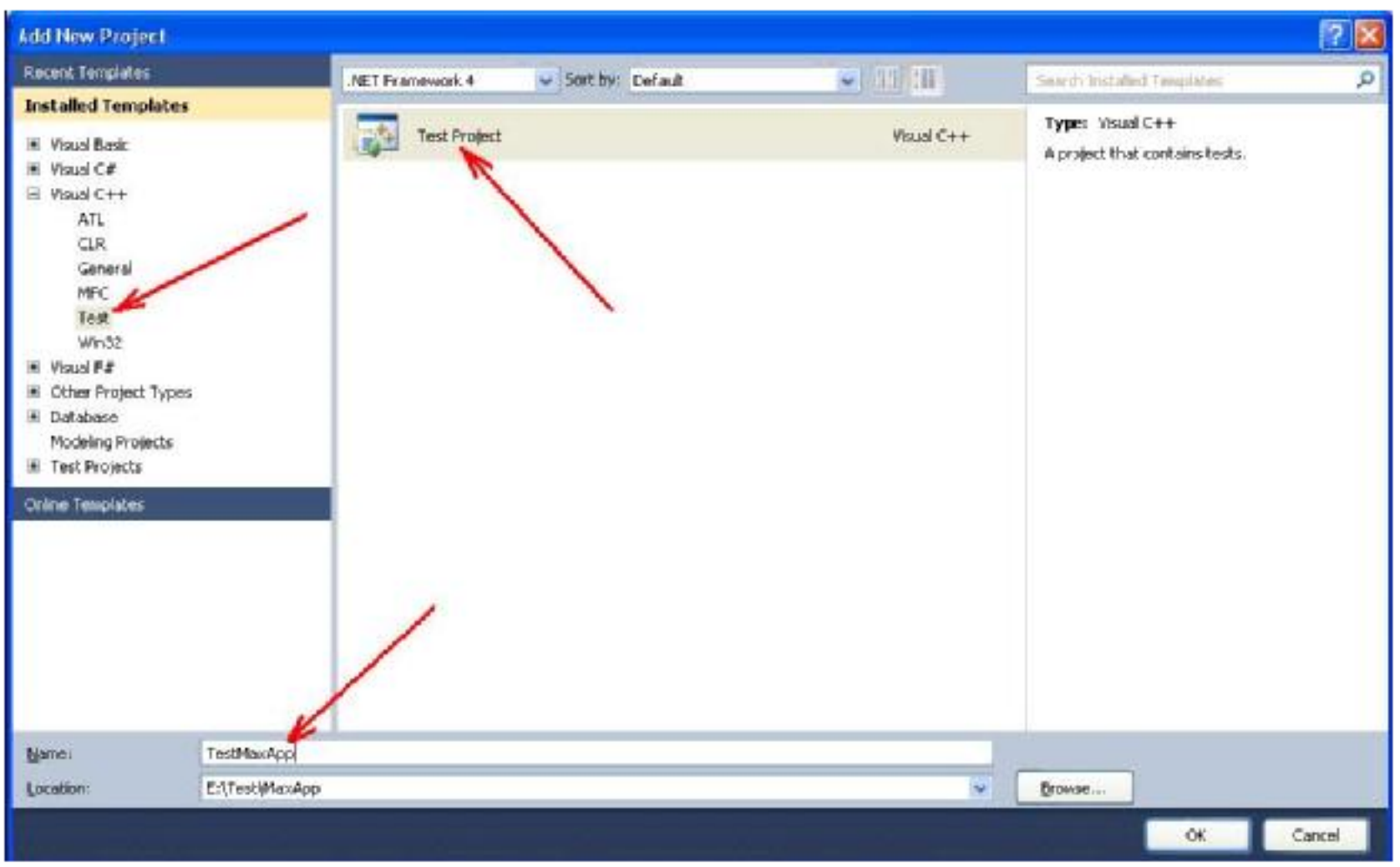

Рис. 3. Окно«AddNewProject»

В окне выбирается группа шаблонов **Visual C++ — Test**. Из отображаемых шаблонов выбирается шаблон проекта «TestProject«. В поле**«Name»** - имя проекта, который будет тестировать программу.

Задать имя, например, **TestMaxApp**. Проект размещается в отдельной папке.

Система создает новые файлы проекта, которые будет тестировать, как показано на рис. 4.

Создается файл (модуль) с именем**«UnitTest1.cpp»**. В этом файле нужно ввести программный код, который будет тестировать функцию ***Max()***из модуля **MaxApp.cpp.**

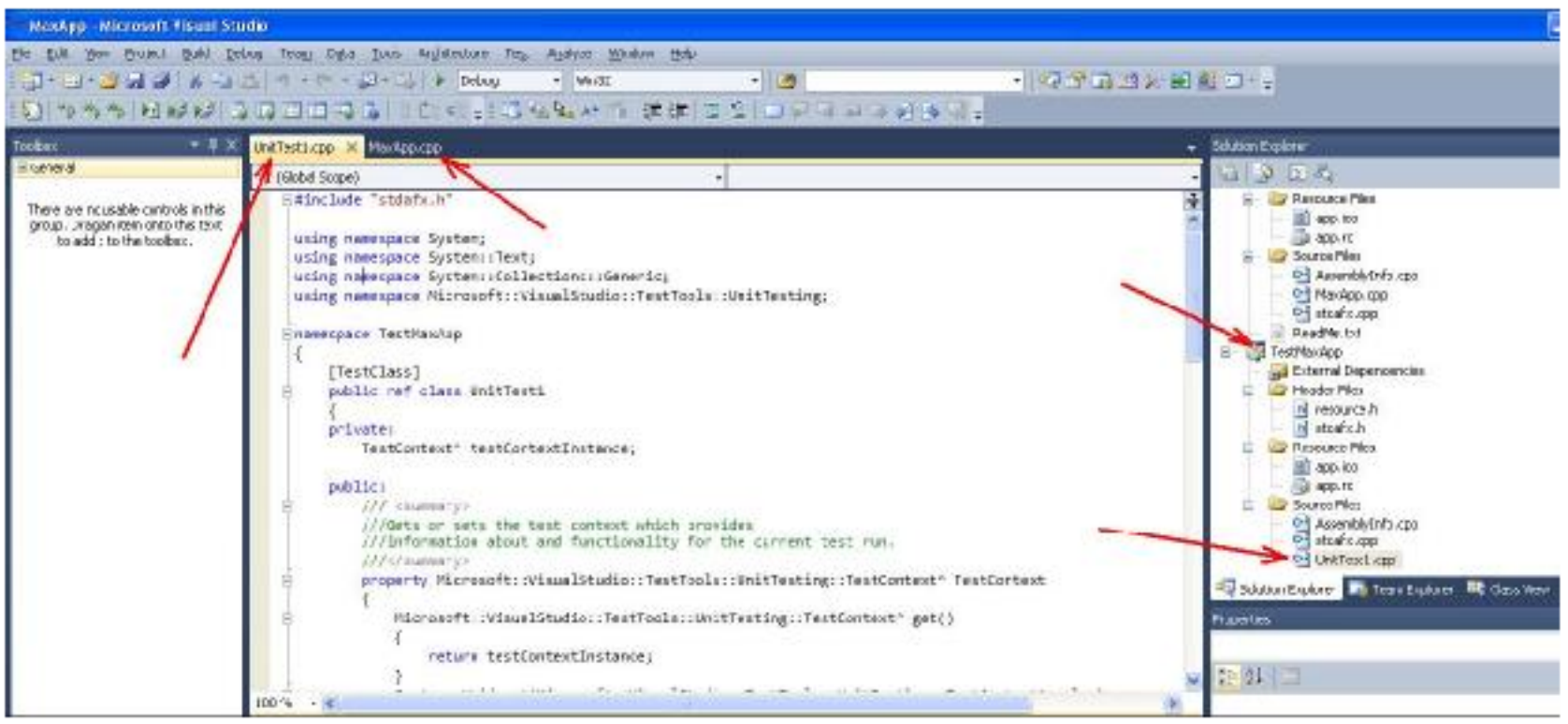

Рис. 4. Текст модуля**UnitTest1.cpp**.Окно утилиты **SolutionExplorer** с отображенными проектами **TestMaxApp**и**MaxApp**

3.2. Структура решения

Как видно из рис. 4, утилита **SolutionExplorer** отображает структуру решения (**SolutionItems**), которое содержит два проекта:

∞проект **MaxApp**. Это проект, созданный по шаблону **CLR ConsoleApplication**с функцией **Max(),** которую нужно протестовать;

∞проект **TestMaxApp**. Этот проект предназначен для тестирования функций проекта**MaxApp**. Программный код, который будет тестировать функцию **Max(),** будет вноситься в файл проекта **UnitTest1** проекта **TestMaxApp**.

Оба проекта могут выполняться независимо друг от друга.

3.3. Текст файла теста«UnitTest1.cpp». Атрибуты [TestMethod] и [TestClass]

В проекте **TestMaxApp** главным есть файл теста**UnitTest1.cpp**. В этом файле размещаются методы, которые будут тестировать функции проекта**«MaxApp.cpp».** Проект **TestMaxApp** может содержать любое количество файлов, которые содержат тесты (например,**UnitTest2.cpp,UnitTest3.cpp**и т.д.).

Листинг файла**UnitTest1.cpp**, сформированный MS VisualStudio, следующий:

```
#include"stdafx.h"
using namespace System;
using namespace System::Text;
using namespace System::Collections::Generic;
using namespace Microsoft::VisualStudio::TestTools::UnitTesting;

namespaceTestMaxApp
{
     [TestClass]
public ref class UnitTest1
     {
private:
TestContext^ testContextInstance;

public:
/// <summary>
      ///Gets or sets the test context which provides
      ///information about and functionality for the current test run.
      ///</summary>

property Microsoft::VisualStudio::TestTools::UnitTesting::TestContext^
TestContext
{
             Microsoft::VisualStudio::TestTools::UnitTesting::TestContext^
get()
             {
returntestContextInstance;
             }

             System::Void
set(Microsoft::VisualStudio::TestTools::UnitTesting::TestContext^ value)
             {
testContextInstance = value;
             }
         };

#pragma region Additional test attributes
//
      //You can use the following additional attributes as you write your tests:
      //
      //Use ClassInitialize to run code before running the first test in the class
      //[ClassInitialize()]
      //static void MyClassInitialize(TestContext^ testContext) {};
      //
      //Use ClassCleanup to run code after all tests in a class have run
      //[ClassCleanup()]
      //static void MyClassCleanup() {};
      //
      //Use TestInitialize to run code before running each test
      //[TestInitialize()]
      //void MyTestInitialize() {};
      //
      //Use TestCleanup to run code after each test has run
      //[TestCleanup()]
      //void MyTestCleanup() {};
      //
#pragma endregion

          [TestMethod]
void TestMethod1()
          {
```

```
//
        // TODO: Add test logic here
//
            };
      };
}
```

Как видно из вышеприведенного кода, файл содержит класс с именем**UnitTest1**. В классе есть общедоступный (public) метод с именем**TestMethod1().** Перед реализацией метода **TestMethod1()** размещается атрибут [TestMethod]. Это значит, что в тело метода нужно вписать код, который будет тестировать функции проекта **MaxApp.**

В классе можно вписывать любое количество методов, которые будут тестировать разные функции из разных модулей. Главное, чтобы эти методы были помечены атрибутом **[TestMethod]**. Например, если нужно добавить второй метод тестирования с именем **MySecondTestMethod()**, то в тело класса нужно вписать приблизительно следующий код:

```
[TestMethod]

voidMySecondTestMethod()
{
// тестирующий программный код метода MySecondTestMethod()
   // ...
}
```

Аналогично, перед классом UnitTest1 размещается атрибут [TestClass]. Это означает, что в классе есть тестирующие методы.

3.4. Выполнение изменений в тексте модуляUnitTest1

Допускается изменять названия методов и добавлять новые методы, которые помечены атрибутом [TestMethod] в модуле UnitTest1.cpp. Учитывая это, в тексте модуля UnitTest1.cpp метод TestMethod1() необходимо переименовать на TestMax().

После выполненных изменений, сокращенный текст модуля файла UnitTest1.cpp :

```
#include"stdafx.h"
using namespace System;
using namespace System::Text;
using namespace System::Collections::Generic;
using namespace Microsoft::VisualStudio::TestTools::UnitTesting;

namespace TestProjectApp01
{
    [TestClass]
public ref class UnitTest1
    {
private:
TestContext^ testContextInstance;

public:

        ...

        [TestMethod]
voidTestMax()
```

```
            {
//
            // TODO: Add test logic here
            //
            };
        };
}
```

3.5. Подключение модуля «MaxApp.cpp» проекта MaxApp к проекту TestMaxApp

Чтобы иметь доступ к функции ***Max()*** модуля ***«MaxApp.cpp»*** проекта ***MaxApp*** из проекта ***TestMaxApp***, нужно вызвать этот модуль с помощью директивы #include.

Существует 2 способа подключения:

- подключить файл ***«MaxApp.cpp»***, указав его полное имя на диске (или на другом источнике);
- в ссылках на сборки MicrosoftVisualStudio настроить папку с проектом ***MaxApp***, чтобы она подключалась автоматически. После этого можно обращаться к файлу***«MaxApp.cpp»***по его сокращенному имени.

Далее описываются оба способа.

3.5.1. Способ 1. Подключение с заданием полного имени

Способ упрощенный. Он удобен, когда нужно подключить в проект небольшое количество файлов для тестирования.

В проекте ***TestMaxApp*** в файле ***«UnitTest1.cpp»*** после подключения пространств имен нужно задать следующую строку:

#include "E:\\Test\\MaxApp\\MaxApp\\MaxApp.cpp"

Здесь указывается полное имя на диске файла ***MaxApp.cpp***, где размещается функция ***Max()***, которую нужно протестировать.

3.5.2. Способ 2. Подключение с заданием сокращенного имени

Этот способ эффективен, когда нужно протестовать несколько модулей разных проектов. Подключается каталог (папка) с файлами тестируемого проекта.

В данном случае нужно подключить папку:

E:\Test\MaxApp\\MaxApp

в перечень ссылок на сборки (проекты), которые автоматически подключаются к проекту ***TestMaxApp***. После этого, можно будет подключать модуль ***MaxApp.cpp*** по сокращенному имени:

#include"MaxApp.cpp"

избегая использования длинного имени как показано в пункте 3.5.1.

Прежде всего вызывается команда **«References…»** из контекстного меню **TestMaxApp** (рис. 5). Другой способ вызова – команда **Project->References…** (предварительно нужно выделить проект ***TestMaxApp***).

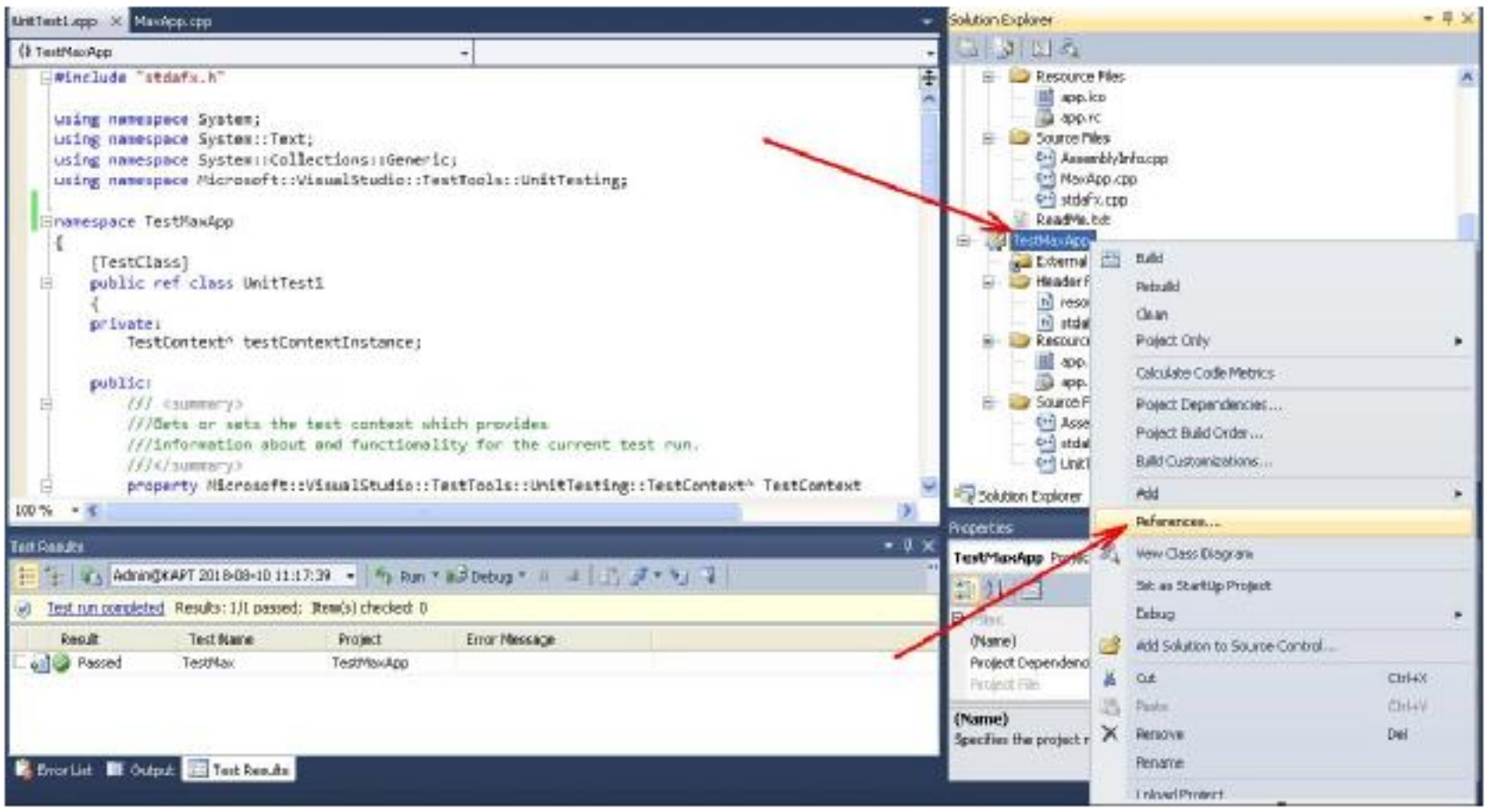

Рис. 5. Вызов окна просмотрассылок на сборки, которые используются в проекте ***TestMaxApp***

В результате откроется окно ***«TestMaxAppPropertyPages»*** которое изображено на рисунке 6.

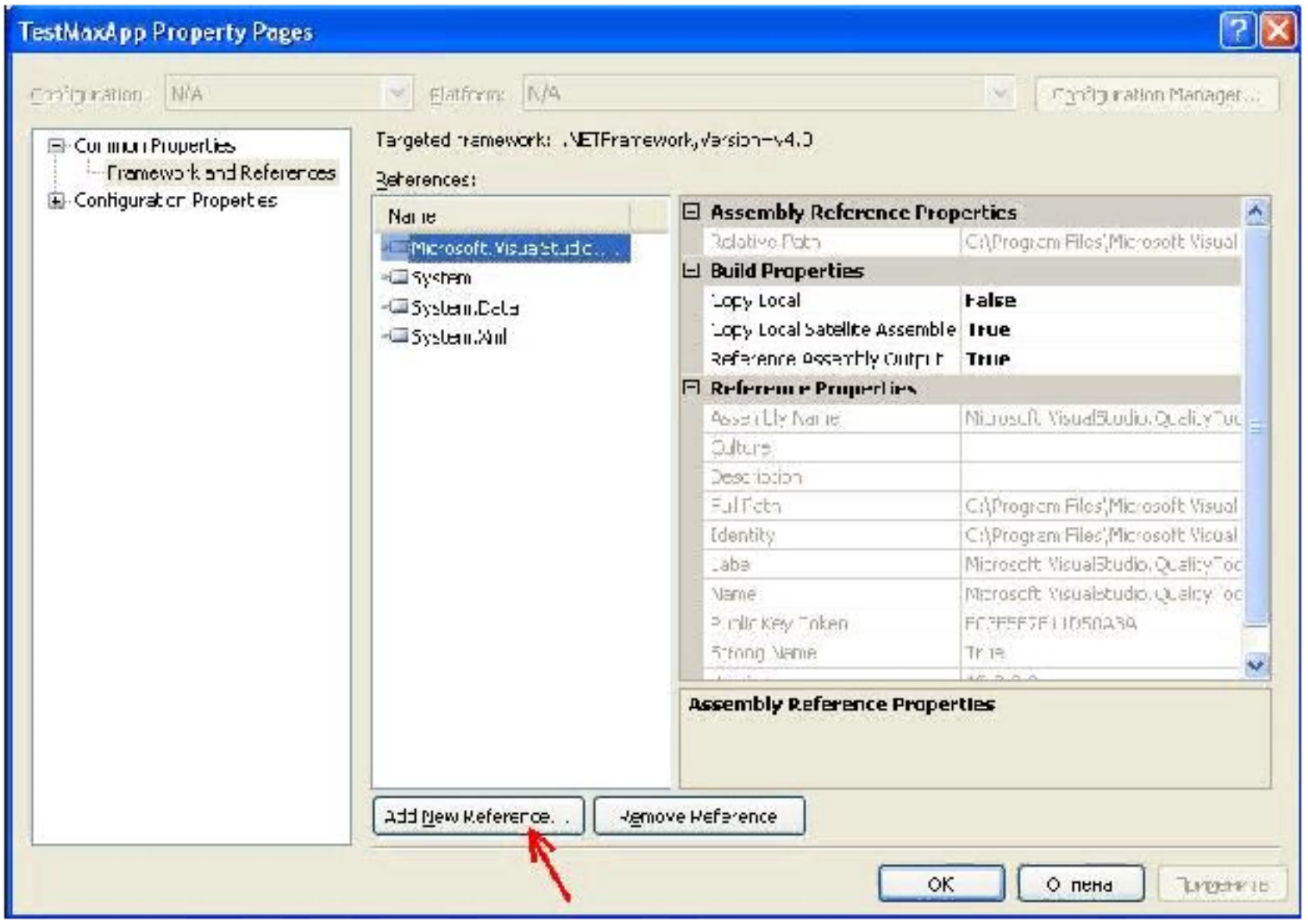

Рис. 6. Окно ***TestMaxApp Property Pages***, команда ***«Add New Reference...»***

В окне ***«TestMaxApp Property Pages»*** сначала нужно активировать вкладку ***«Common Properties» -> «Framework and References»***.

Затем нужно выбрать команду ***«AddNewReference...»***. Эта команда разрешает добавлять в проект новые папки с модулями, методы (функции, ресурсы) которых потом можно включать в проект директивой **#include**. После выбора команды откроется окно ***AddReference***, изображенное на рисунке 7.

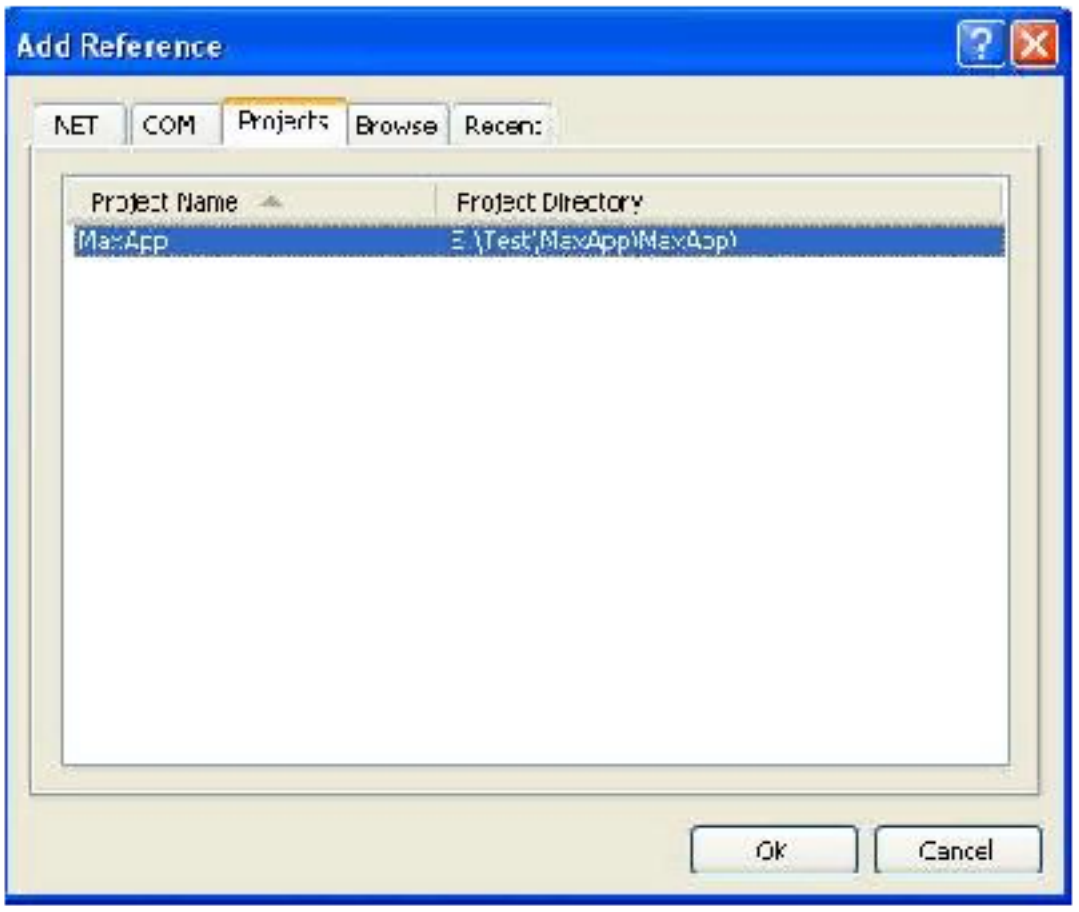

Рис. 7. Окно ***AddReference*** с отображенными проектами в текущем решении

В окне ***«AddReference»*** во вкладке ***Project*** отображаются папки проектов, которые используются в данном решении (***Solution***). В нашем случае отображается только один проект ***MaxApp***, который размещается в папке:

E:\Test\MaxApp\MaxApp

После выбора на **OK** происходит возврат в предыдущее окно, в котором в перечне ссылок на сборки будет отображена ссылка на проект ***MaxApp***.

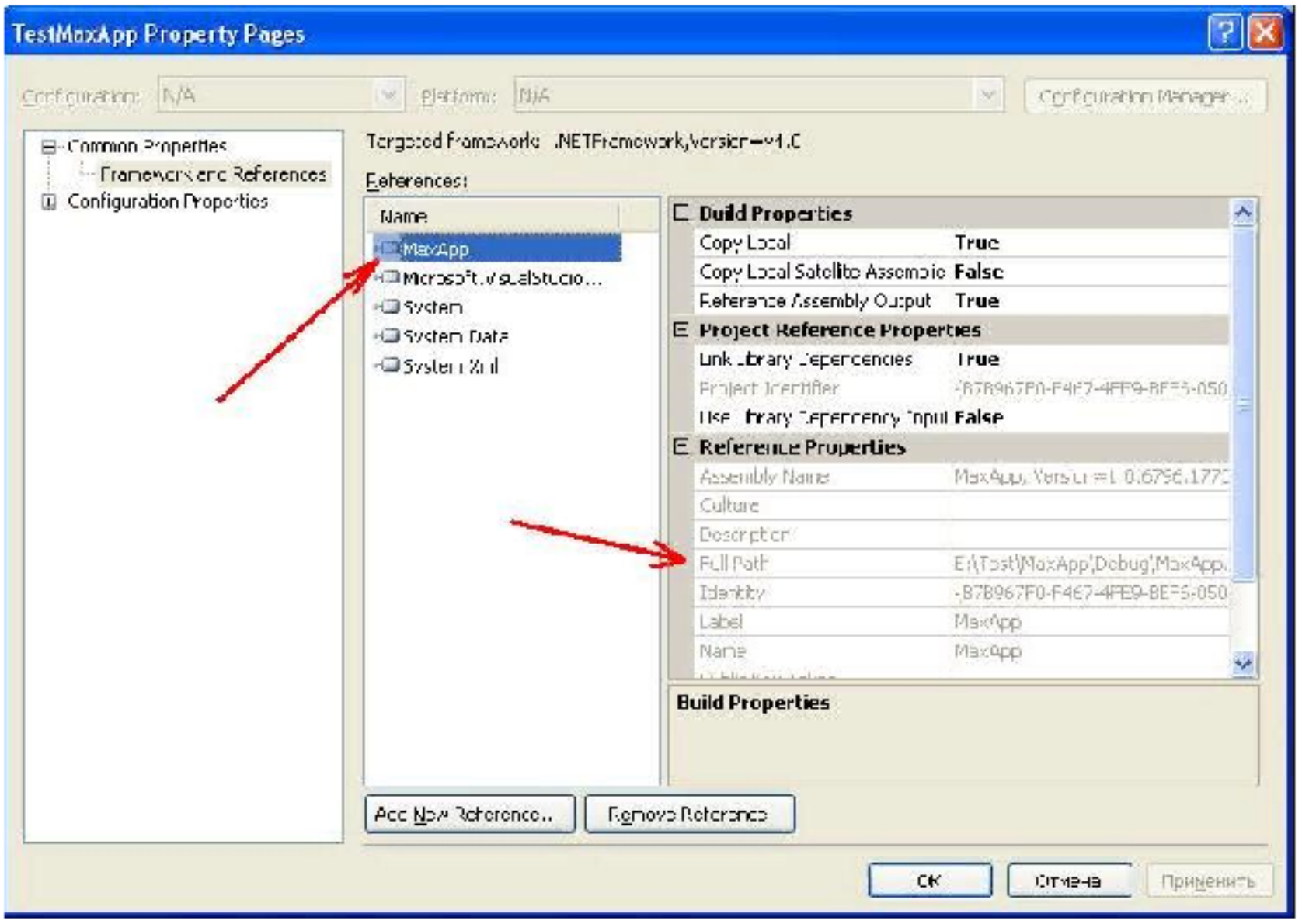

Рис. 8. Окно ***«TestMaxAppPropertyPages»*** после добавления проекта ***MaxApp*** в перечень общедоступных сборок

Следующим шагом нужно подключить папку с проектом ***MaxTest*** в перечень ***«IncludeDirectories»***.

Чтобы сделать это, нужно сначала активировать строку:

```
Configuration Properties -> VC++ Directories
```

как показано на рисунке 9.

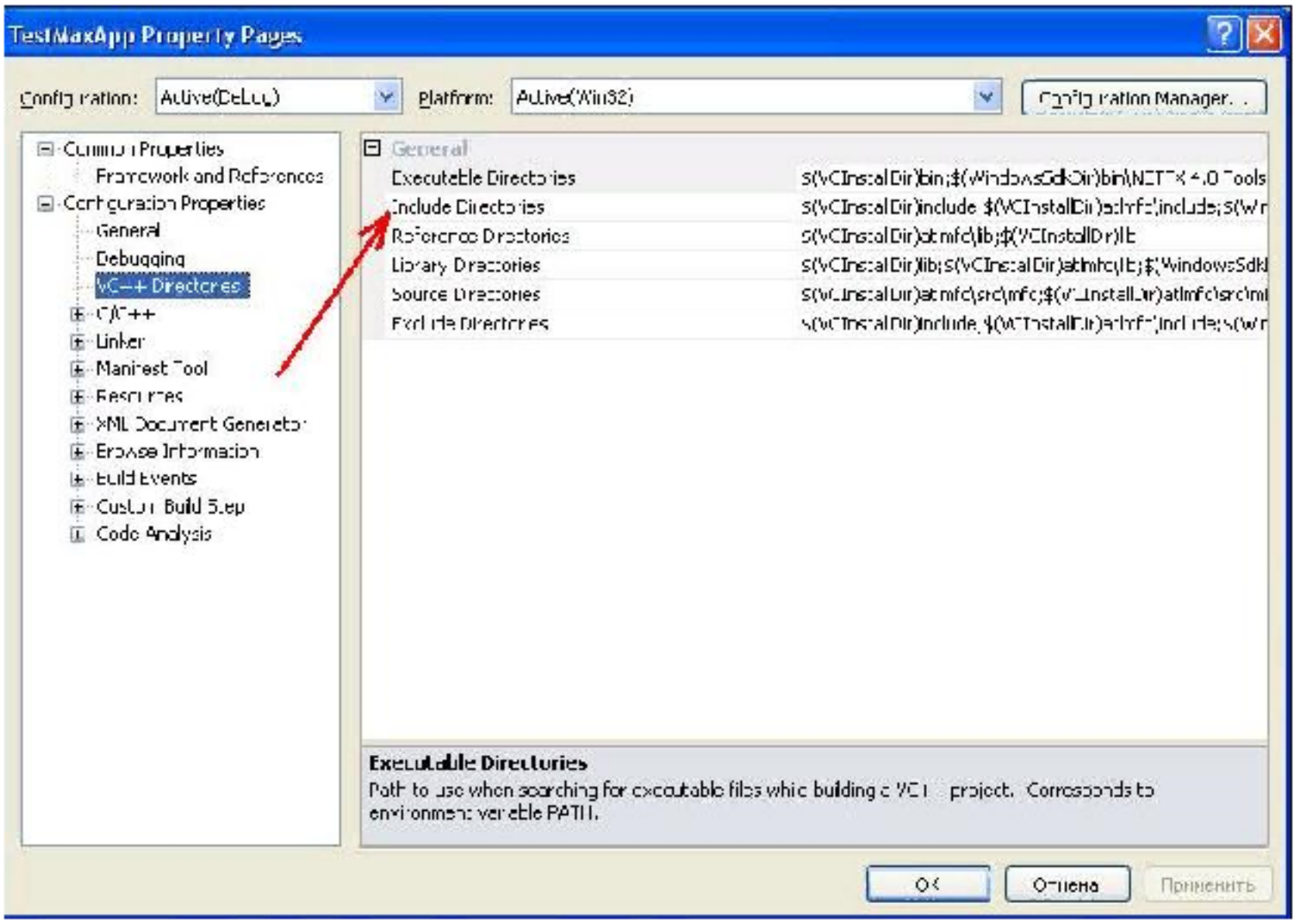

Рис. 9. Окно ***TestMaxApp Property Pages***, строка ***«Include Directories»***

В перечне ***«General»*** выбирается ***«Include Directories»***. В результате раскрывается спадающий список, в котором нужно выбрать команду ***«<Edit…>»***. После этого откроется окно ***«IncludeDirectories»*** изображенное на рисунке 10.

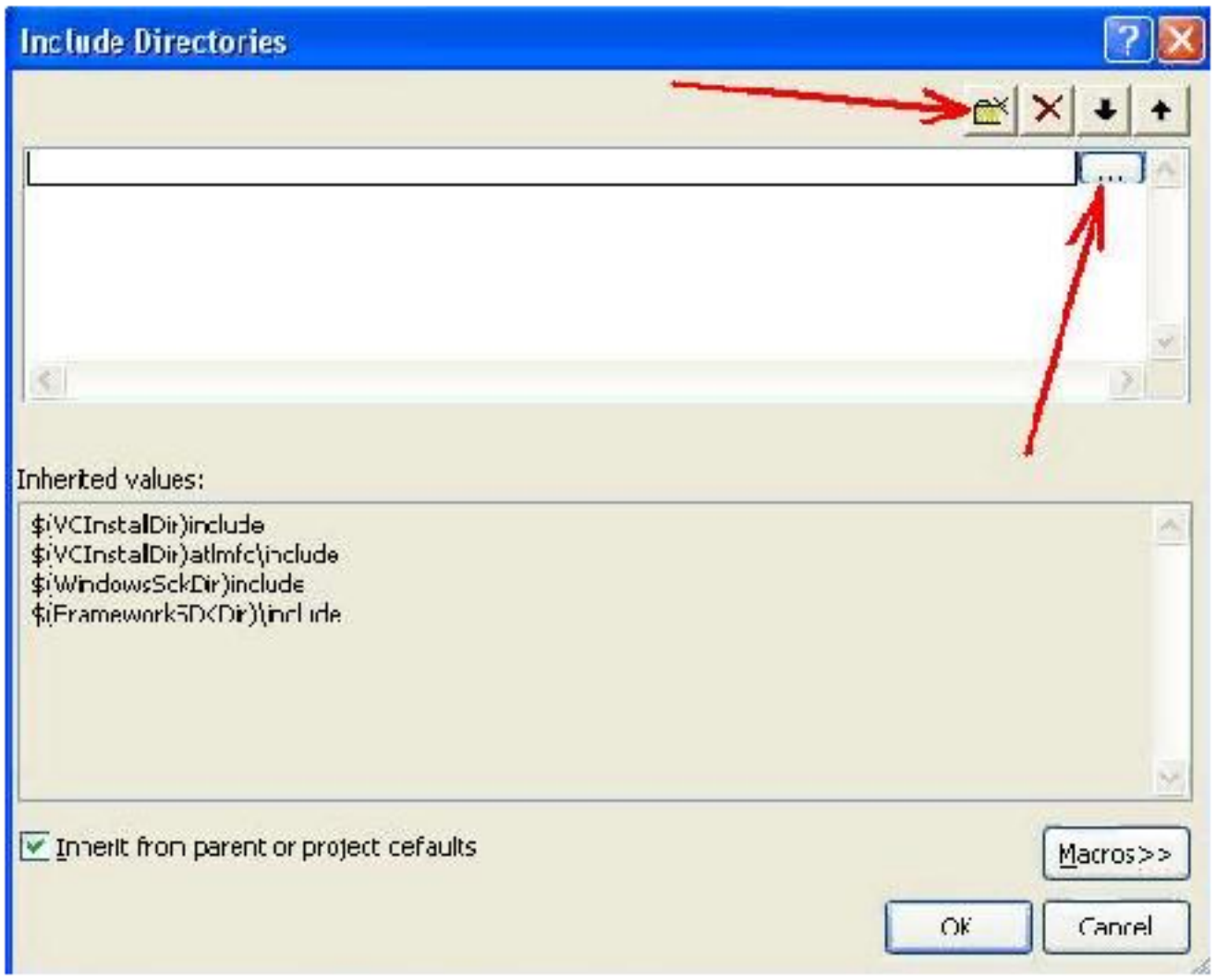

Рис. 10. Команда ***«NewLine»*** и кнопка выбора папки с файлами, которые подключаются

В окне ***«IncludeDirectories»*** нужно добавить новую строку командой ***«NewLine»*** и выбрать папку с проектом ***MaxApp***, как показано на рисунке 10. После выбора кнопки **«…»** откроется стандартное окно Windows для выбора нужной папки.

После выбора, окно ***IncludeDirectories*** будет иметь вид, как показано на рисунке 11.

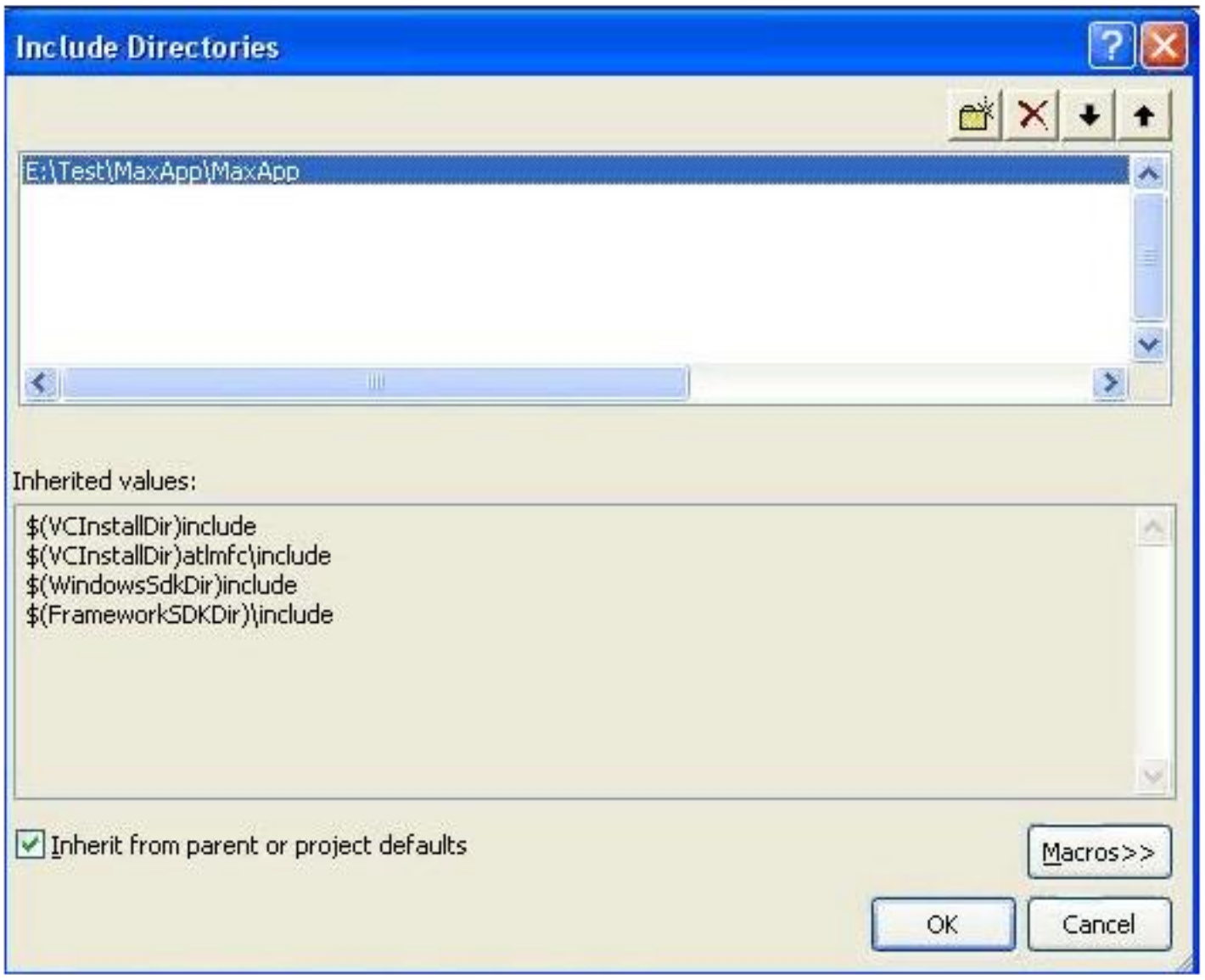

Рис. 11. Окно ***Include Directories*** после выбора папки (например, E:\Test\MaxApp\MaxApp)

Для перехода к предыдущему окну нужно выбрать **OK**.

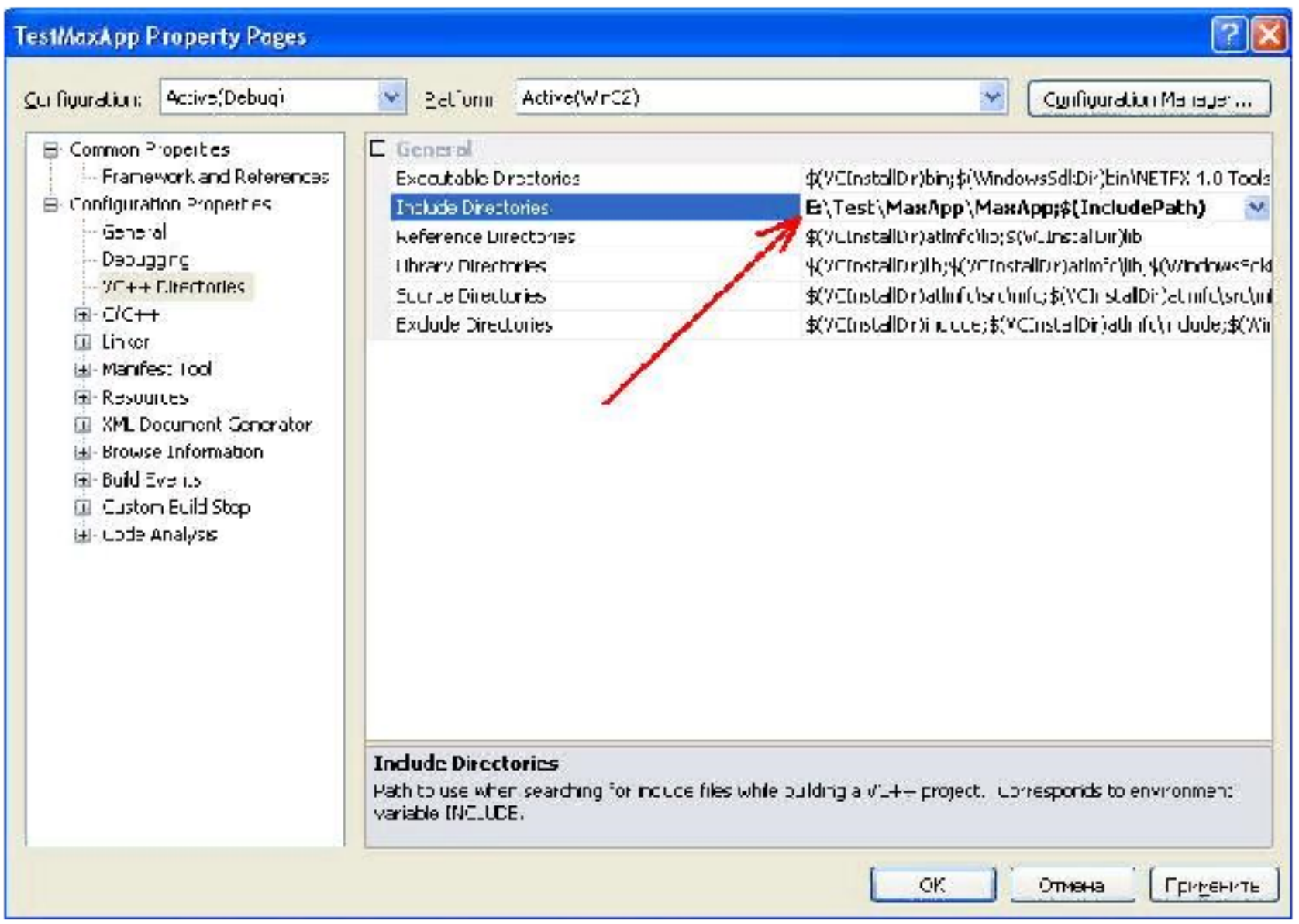

Рис. 12. Окно ***TextMaxApp Property Pages*** после настройки строки ***«Include Directories»***

Для перехода к написанию программного кода тестирования нужно выбрать OK.

После выполненных действий, все файлы из папки (например, E:\Test\MaxApp\MaxApp) можно подключать по сокращенному имени, например #include"MaxApp.cpp"

3.6. Написание программного кода для проведения тестирования в методе TestMax()

Программный код, который тестирует функцию ***Max()*** вписывается в тело метода ***TestMax()*** модуля ***«UnitTest1.cpp»***. Фрагмент программного кода метода ***TestMax()*** выглядит следующим образом:

```
...
```

```
[TestMethod]

voidTestMax()
{
//
   // TODO: Add test logic here
   //
int  t;
```

```
    t = Max (5, 6, 7);
    Microsoft::VisualStudio::TestTools::UnitTesting::Assert::AreEqual(t, 6);
};
```

...

В вышеприведенном коде вызывается функция ***AreEqual()*** из класса ***Assert***. Эта функция сравнивает значение, которое было возвращено из функции ***Max()*** и число 6. В данном случае, происходит сравнение числа 7 (максимум) с числом 6. Для такого варианту тест не будет выполняться, так как 7 не равняется 6. В результате функция ***Assert::AreEqual()*** выбросит исключительную ситуацию (exception), что будет отображаться в специальном окне.

4. Запуск теста на выполнение и проверка результата тестирования

В MicrosoftVisualStudio для работы с Unit-тестами реализовано специальное меню команд, которое называется ***Test***.

Чтобы запустить тест на выполнение, нужно выбрать одну из команд:

Test -> Run -> Tests in Current Context

или

```
Test -> Run -> All Tests in Solution
```

как изображено на рисунке 13.

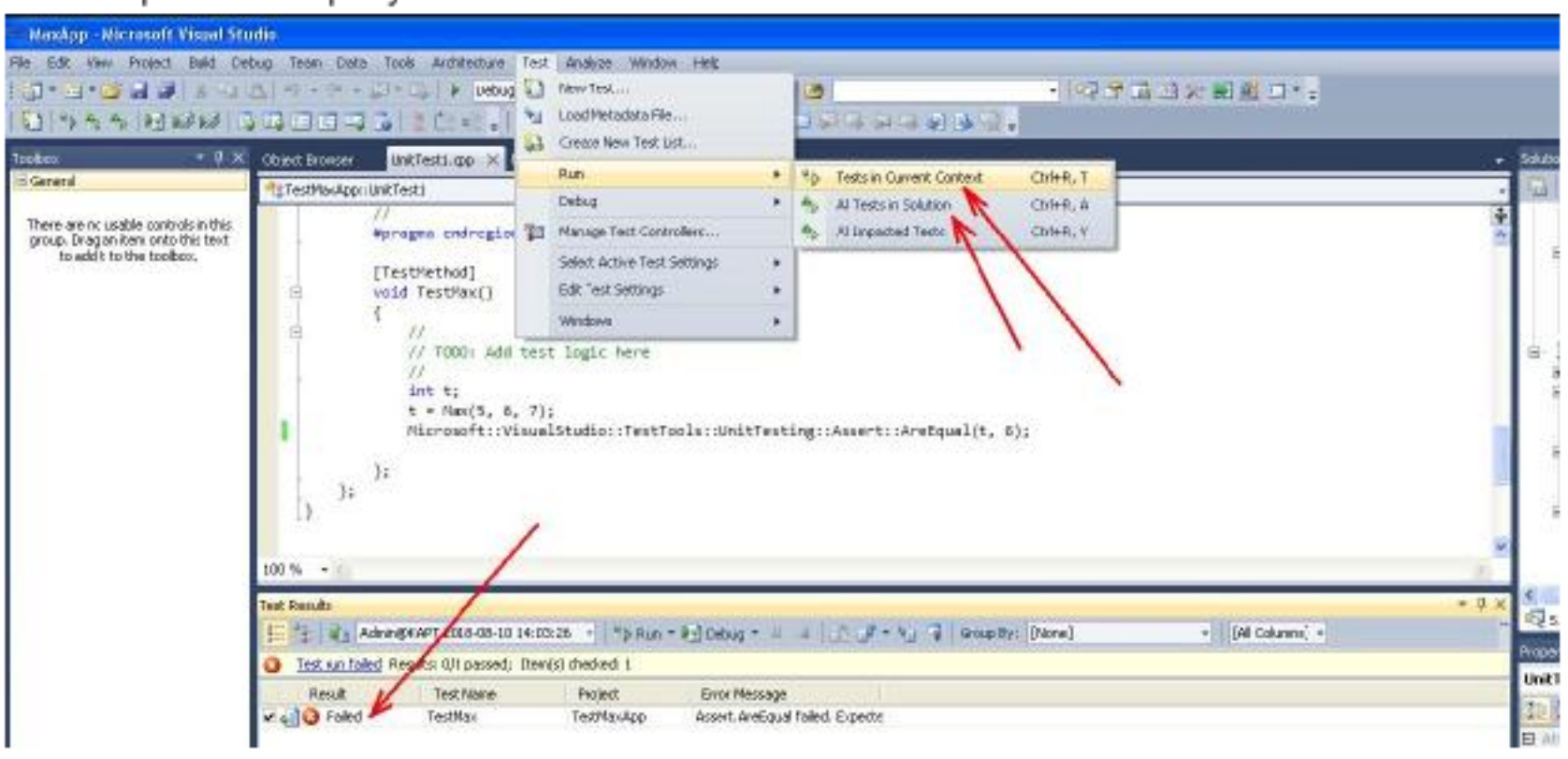

Рис. 13. Вызов команды запуска тестирования и просмотр результата

После запуска теста, результат можно просмотреть в нижней части в окне ***TestResults***. Как видно, тест не сдан. Это логично, поскольку в функции ***Assert::AreEqual()*** мы сравниваем числа 6 и 7, которые различны между собой. Здесь специально введено число 6 вместо числа 7.

Если вместо числа 6 ввести правильный ответ – число 7 (максимум между 5, 6, 7), то тест будет сдан. В этом случае текст метода ***TestMax()*** будет следующей:

...

```
[TestMethod]

voidTestMax()
{
//
   // TODO: Add test logic here
   //
int  t;
     t  =  Max(5,  6,  7);
     Microsoft::VisualStudio::TestTools::UnitTesting::Assert::AreEqual(t,  7);
};

...
```

Окно результата изображено на рисунке 14.

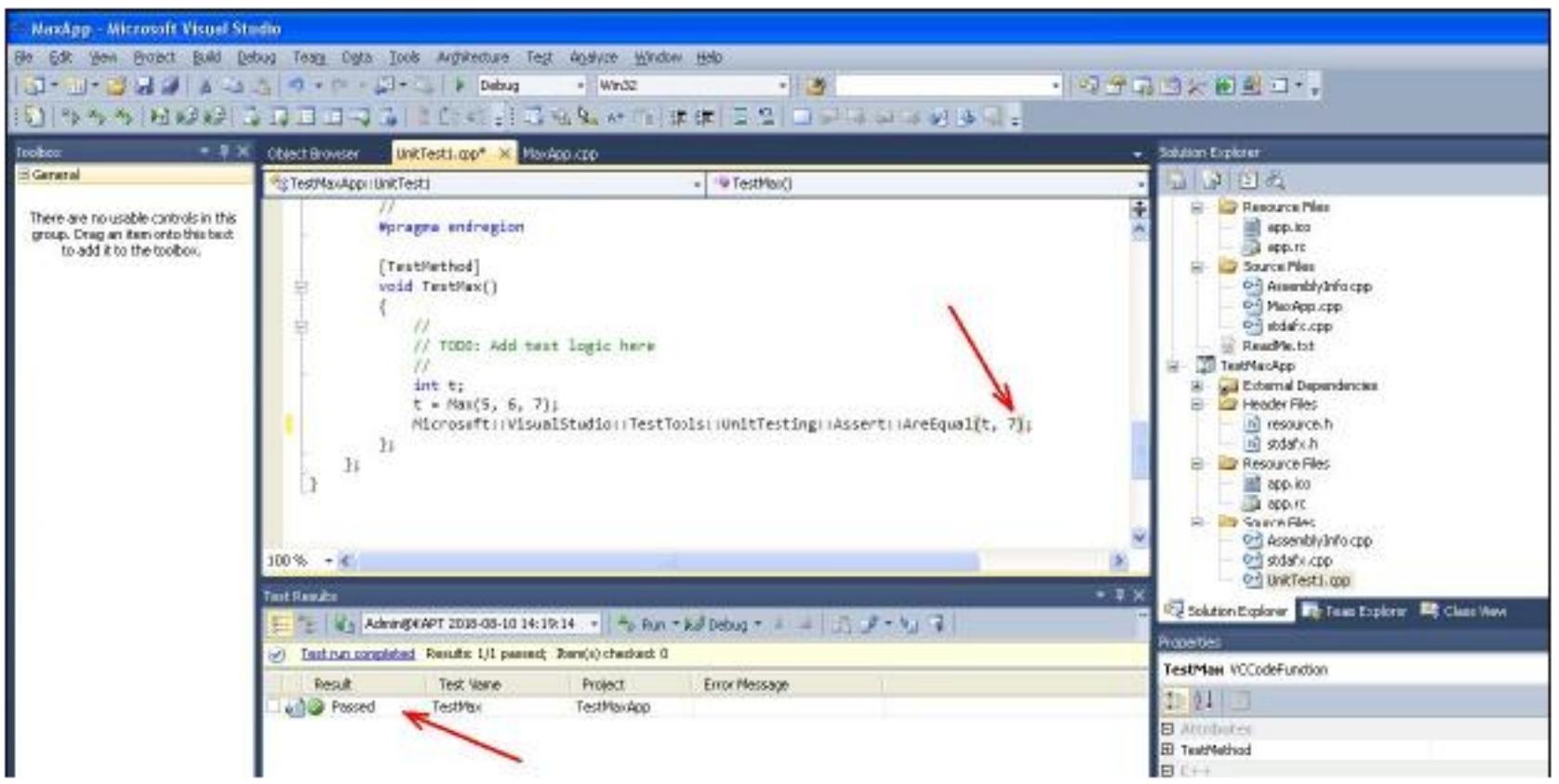

Рис. 14. Результат тестирования для случая, если ввести правильную проверку

Теперь можно сделать вывод о том, что функция ***Max()*** разработана правильно

5. Текст модуля UnitTest1

Ниже приводится текст модуля ***UnitTest1.cpp***, тестирующий функцию ***Max()*** из модуля ***«MaxApp.cpp»***:

```
#include"stdafx.h"
using namespace  System;
using namespace  System::Text;
using namespace  System::Collections::Generic;
using namespace  Microsoft::VisualStudio::TestTools::UnitTesting;
// способ 1 - заданиеполногоимени
// #include "E:\\Test\\MaxApp\\MaxApp\\MaxApp.cpp"

// способ 2 подключение MaxApp по сокращенному имени
#include"MaxApp.cpp"

namespaceTestMaxApp
{
```

```
     [TestClass]
public ref class UnitTest1
     {
private:
TestContext^ testContextInstance;

public:
/// <summary>
     ///Gets or sets the test context which provides
     ///information about and functionality for the current test run.
     ///</summary>

property Microsoft::VisualStudio::TestTools::UnitTesting::TestContext^
TestContext
          {
               Microsoft::VisualStudio::TestTools::UnitTesting::TestContext^
get()
               {
returntestContextInstance;
               }

               System::Void
set(Microsoft::VisualStudio::TestTools::UnitTesting::TestContext^ value)
               {
testContextInstance = value;
               }
          };

#pragma region Additional test attributes
//
     //You can use the following additional attributes as you write your tests:
     //
     //Use ClassInitialize to run code before running the first test in the class
     //[ClassInitialize()]
     //static void MyClassInitialize(TestContext^ testContext) {};
     //
     //Use ClassCleanup to run code after all tests in a class have run
//[ClassCleanup()]
     //static void MyClassCleanup() {};
     //
     //Use TestInitialize to run code before running each test
     //[TestInitialize()]
     //void MyTestInitialize() {};
     //
     //Use TestCleanup to run code after each test has run
     //[TestCleanup()]
     //void MyTestCleanup() {};
     //
#pragma endregion

          [TestMethod]
voidTestMax()
          {
//
       // TODO: Add test logic here
       //
int t;
               t = Max(5, 6, 7);

Microsoft::VisualStudio::TestTools::UnitTesting::Assert::AreEqual(t, 7);
};
     };
}
```

6. Итог. Взаимодействие между проектами

В данной работе в решении (Solution) сформированы два проекта. Один проект ***MaxApp*** содержит функцию ***Max()*** которую нужно протестировать. Второй проект ***TestMaxApp*** содержит методы, которые тестируют.

В MicrosoftVisualStudio каждый из проектов запускается с помощью различных команд меню. Так, проект ***MaxApp*** запускается стандартным способом из меню ***Run***. А тестирующий проект ***TestMaxApp*** запускается из специального меню ***Test***.