МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ
«КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ»

DLL C++

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторной работе № 9

по дисциплине «ООП»

Киев 2015

СОДЕРЖАНИЕ

1 Цель лабораторной работы 3

2 Теоретические положения 4

2.1. Создание проекта библиотеки динамической компоновки (DLL) в Visual Studio 2013 4

2.2. Добавление класса в библиотеку динамической компоновки 5

2.3. Связывание исполняемого файла с библиотекой DLL 8

2.4. Неявное связывание (статическая загрузка) 9

2.5. Явное связывание (Динамическая загрузка) 10

2.6. Создание приложения, ссылающегося на DLL (статическая загрузка) 10

2.7. Создание приложения, ссылающегося на DLL (динамическая загрузка) 19

3 Задания 20

4 Требования к отчету 24

5 Контрольные вопросы 25

1 ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Цель работы – изучить особенности использования DLL, рассмотреть принципы статического и динамического подключения.

2 ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

DLL (англ. *Dynamic Link Library* - «библиотека динамической компоновки», «динамически подключаемая библиотека») в операционных системах *Microsoft Windows* и *IBM OS/2* - динамическая библиотека, позволяющая многократное использование различными программными приложениями. К *DLL* относятся также элементы управления *ActiveX* и драйверы. В системах *UNIX* аналогичные функции выполняют так называемые общие объекты (англ. *shared objects*).

Формат файлов *DLL* придерживается тех же соглашений, что и формат исполняемых файлов, сочетая код, таблицы и ресурсы, отличаясь лишь интерпретацией некоторых полей.

2.1. Создание проекта библиотеки динамической компоновки (DLL) в Visual Studio 2013

В строке меню выберите **File**, **New**, **Project** и создайте консольное приложение (рисунок 2.1).

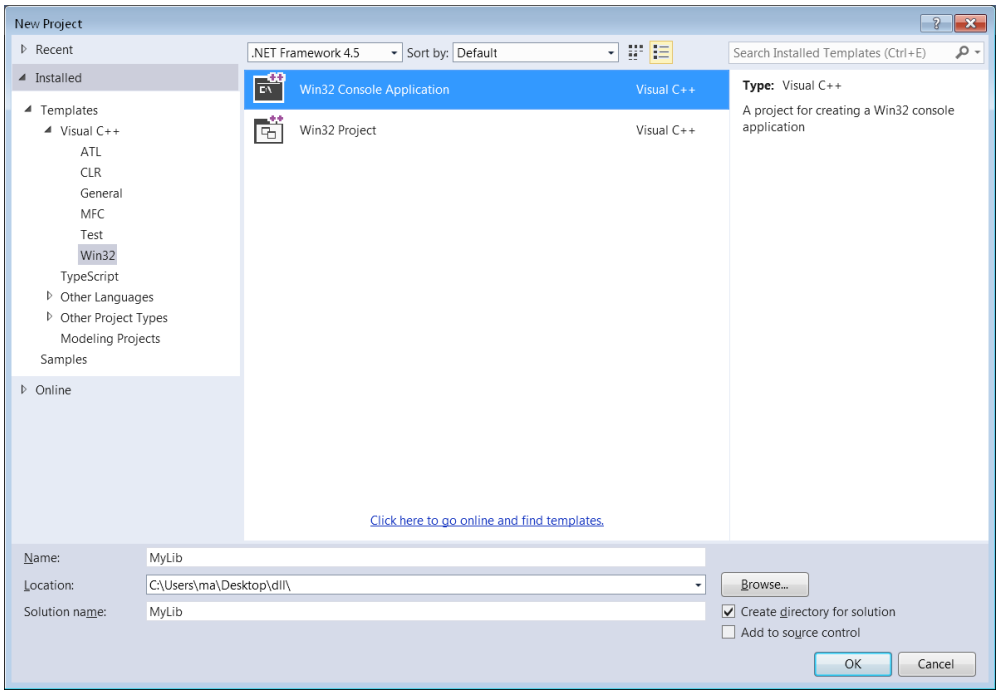


Рисунок 2.1

На странице **Overview** диалогового окна **Win32 Application Wizard** нажмите кнопку **Next**.

На странице **Application Settings** выберите в поле **Application type** пункт **DLL** (рисунок 2.2)**.**

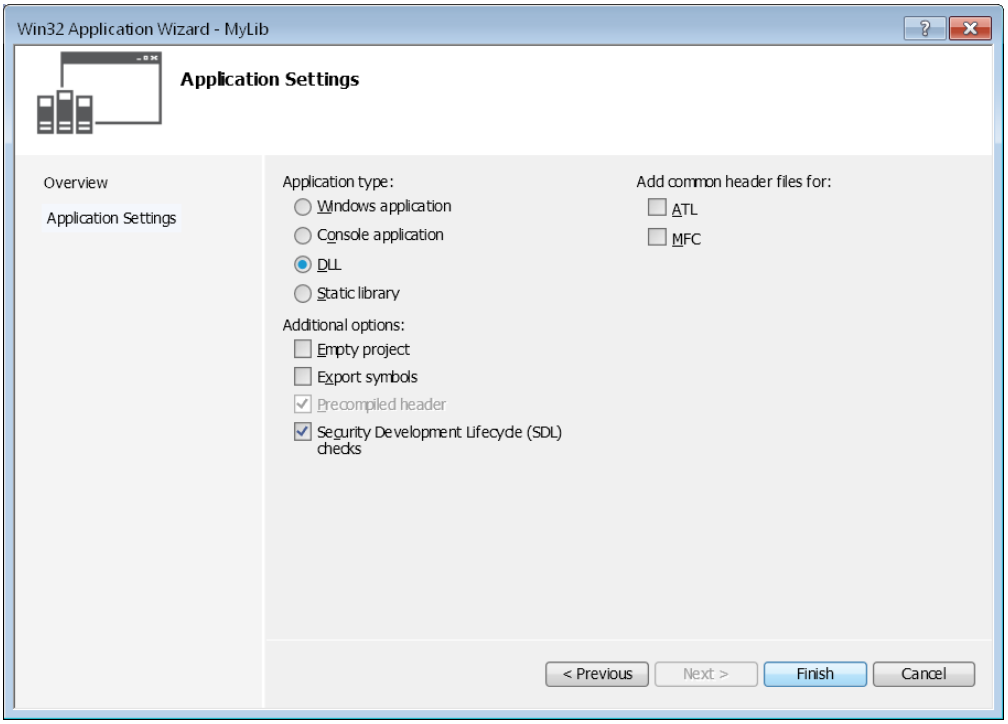


Рисунок 2.2

Нажмите кнопку **Finish**, чтобы создать проект.

2.2. Добавление класса в библиотеку динамической компоновки

Чтобы создать файл заголовка для нового класса, в меню **Проект** выберите пункт **Добавить новый элемент**. В диалоговом окне **Добавить новый элемент** в левой области в разделе **Visual C++** выберите **Код**. В центральной области выберите **Заголовочный файл (.h)**. Укажите имя для файла заголовка, например MyLib.h, а затем нажмите кнопку **Добавить.** Показан пустой заголовочный файл.

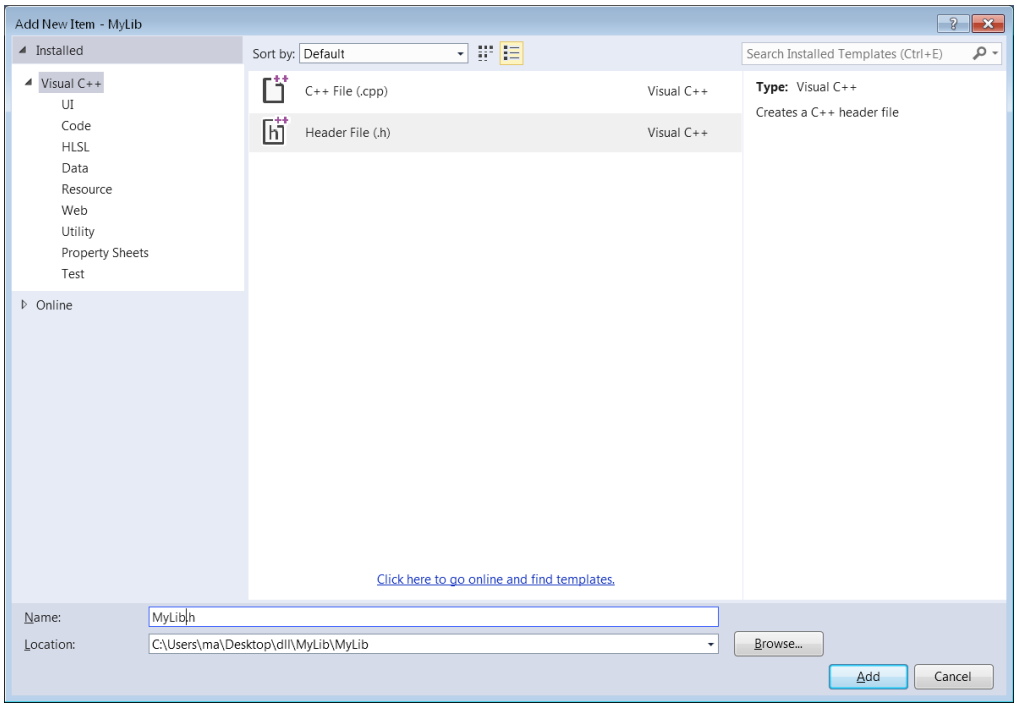


Рисунок 2.3

Добавьте следующий код в начало файла заголовка:

```
#ifdef MATHFUNCSDLL_EXPORTS
#define MATHFUNCSDLL_API __declspec(dllexport)
#else
#define MATHFUNCSDLL_API __declspec(dllimport)
#endif
```

Добавьте базовый класс с именем MyMathFuncs для выполнения общих математических операций, таких как сложение, вычитание, умножение и деление. Код должен выглядеть примерно следующим образом:

```
namespace MathFuncs
{
	class MyMathFuncs
	{
	public:
		// Returns a + b
		static MyLib_API double Add(double a, double b);

		// Returns a - b
		static MyLib_API double Subtract(double a, double b);

		// Returns a * b
```

```
            static MyLib_API double Multiply(double a, double b);

            // Returns a / b
            // Throws const std::invalid_argument& if b is 0
            static MyLib_API double Divide(double a, double b);
      };
}
```

В проекте **MyLib** откройте файл **MyLib**.cpp.

Реализуйте функциональность класса MyMathFuncs. Код должен выглядеть примерно следующим образом:

```
#include "stdafx.h"
#include "MyLib.h"
#include <stdexcept>
using namespace std;

namespace MathFuncs
{
      double MyMathFuncs::Add(double a, double b)
      {
            return a + b;
      }

      double MyMathFuncs::Subtract(double a, double b)
      {
            return a - b;
      }

      double MyMathFuncs::Multiply(double a, double b)
      {
            return a * b;
      }

      double MyMathFuncs::Divide(double a, double b)
      {
            if (b == 0)
            {
                  throw invalid_argument("b cannot be zero!");
            }
```


```
            return a / b;
        }
    }
```

Скомпилируйте библиотеку динамической компоновки, выбрав **Build Solution** в меню **Build**.

В результате будет получена готовая DLL библиотека и сопутствующие компоненты, необходимые для использования ее в приложениях.

2.3. Связывание исполняемого файла с библиотекой DLL

Исполняемый файл связывает (или загружает) библиотеку DLL одним из двух способов:

– **неявное связывание (статическая загрузка);**

– **явное связывание (динамическая загрузка).**

Неявное связывание иногда называют статической загрузкой или динамической компоновкой во время загрузки.

Явное связывание иногда называют динамической загрузкой или динамической компоновкой во время выполнения.

При неявном связывании исполняемый файл, использующий ссылки DLL на библиотеку импорта (LIB-файл), предоставляется автором библиотеки DLL. Операционная система загружает библиотеку DLL после загрузки исполняемого файла. Клиентский исполняемый файл вызывает экспортированные функции библиотеки DLL, таким способом, как если бы функции содержались в исполняемом файле.

При явном связывании исполняемый файл, использующий библиотеку DLL, должен делать вызовы функции для явной загрузки и выгрузки библиотеки DLL и осуществления доступа к экспортированным функциям библиотеки DLL. Клиентский исполняемый файл вызывает экспортированные функции с помощью указателя функции.

Независимо от выбранного метода исполняемый файл может использовать одну и ту же библиотеку DLL. Более того, эти механизмы не

являются взаимоисключающими, поскольку в то время как один исполняемый файл неявно связывается с библиотекой DLL, другой может выполнять явное связывание.

2.4. Неявное связывание (статическая загрузка)

Для неявного связывания с библиотекой DLL в исполняемых файлах необходимо использовать следующие компоненты, предоставляемые поставщиком библиотеки DLL:

1) Файл заголовка с расширением .h, в котором содержатся объявления экспортируемых функций и классов C++. Для объявления классов, функций и данных необходимо использовать ключевое слово **__declspec(dllimport)**. Дополнительные сведения см. в описании ключевых слов dllexport, dllimport.

2) Библиотека импорта (файлы с расширением LIB), с которой выполняется связывание. Библиотека импорта автоматически создается компоновщиком при построении библиотеки DLL.

3) Сама библиотека DLL (файл с расширением DLL).

Если в исполняемом файле используется библиотека DLL, в каждый исходный файл, в котором содержатся вызовы экспортируемых функций, необходимо включить файл заголовка, содержащий эти функции (или классы C++).С точки зрения написания кода вызов экспортированной функции аналогичен вызову любой другой функции.

Чтобы построить вызывающий исполняемый файл, необходимо связать его с библиотекой импорта. Если используется внешний файл makefile, укажите имя файла библиотеки импорта, в котором перечисляются другие файлы или библиотеки объектов (OBJ), с которыми выполняется связывание.

В момент загрузки вызывающего исполняемого файла в операционной системе должен быть доступен DLL-файл.

2.5. Явное связывание (Динамическая загрузка)

При явном связывании приложения должны выполнять вызов функции для явной загрузки библиотеки DLL во время выполнения.Чтобы выполнить явное связывание с библиотекой DLL, приложение должно выполнить следующие действия.

1) Вызвать функцию **LoadLibrary** (или аналогичную функцию) для загрузки библиотеки DLL и получения дескриптора модуля.

2) Вызвать функцию **GetProcAddress**, чтобы получить указатель для каждой экспортируемой функции, вызываемой приложением.Поскольку приложения вызывают функции библиотек DLL с помощью указателя, компилятор не создает внешних ссылок, поэтому нет необходимости выполнять связывание с библиотекой импорта.

3) Вызвать функцию **FreeLibrary** по завершении всех действий с библиотекой DLL.

2.6. Создание приложения, ссылающегося на DLL (статическая загрузка)

Чтобы создать приложение C++, которое будет ссылаться и использовать созданную ранее библиотеку DLL, в меню **Файл** выберите пункт **Создать** и затем пункт **Проект**.

В левой области в категории **Visual C++** выберите **Win32**.

В центральной области выберите **Консольное приложение Win32**.

Укажите имя проекта, например MyExecRefsDll, в поле **Имя**.В раскрывающемся списке рядом с полем **Решение** выберите **Добавить в решение**.В результате новый проект будет добавлен в решение, содержащее библиотеку DLL.Нажмите кнопку **ОК**.

На странице **Обзор** диалогового окна **Мастер приложений Win32** нажмите кнопку **Далее**.

На странице **Параметры приложения** выберите в поле **Тип приложения** пункт **Консольное приложение**.

На странице **Параметры приложения** в разделе **Дополнительные параметры** снимите флажок **Предкомпилированный заголовок (рисунок 2.5)**.

Нажмите кнопку **Готово**, чтобы создать проект.

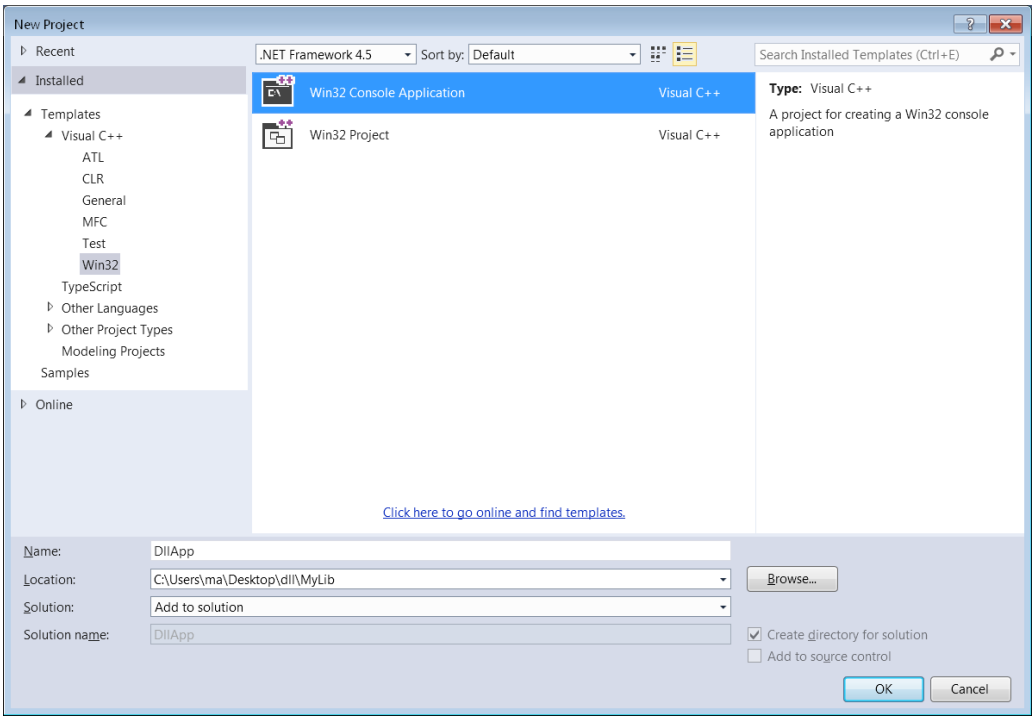


Рисунок 2.4

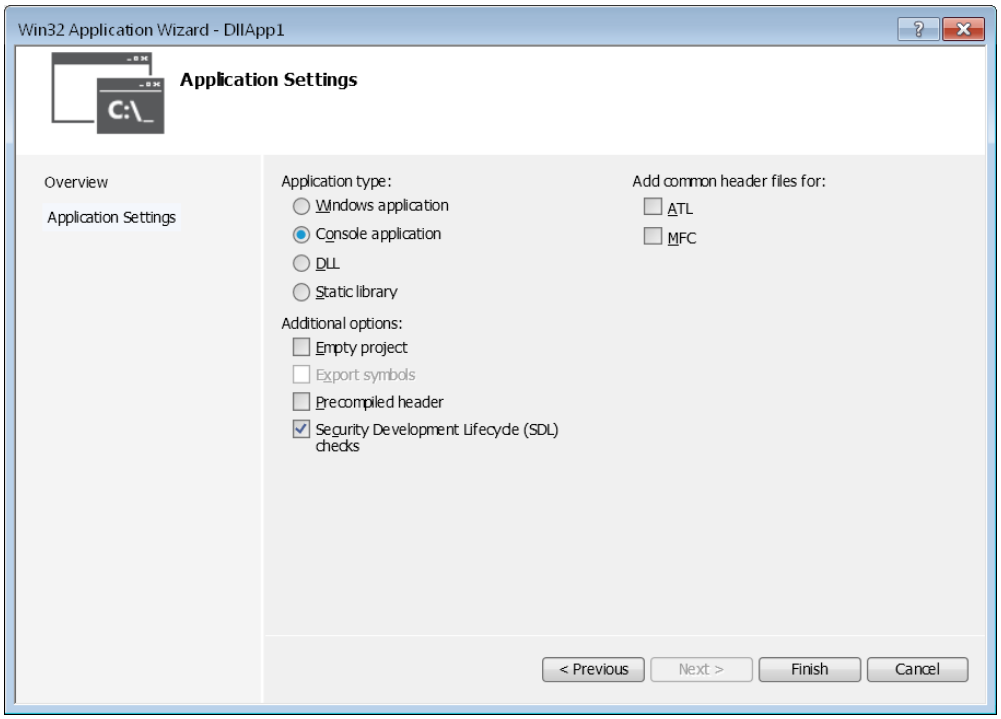


Рисунок 2.5

После создания консольного приложения будет создана пустая программа. Имя исходного файла будет совпадать с ранее выбранным именем. В этом примере он имеет имя DllApp.cpp.

Для использования в приложении математических процедур, созданных в библиотеке DLL, необходимо сослаться на эту библиотеку. Для этого в **обозревателе решений** выберите проект DllApp, а затем в меню **Проект** выберите пункт **Ссылки** (рисунок 2.6). В диалоговом окне **Страницы свойств** разверните узел **Общие свойства**, выберите **.NET Framework и ссылки** и нажмите кнопку **Добавить новую ссылку** (рисунок 2.7).

В диалоговом окне **Добавление ссылки** перечислены библиотеки, на которые можно создать ссылку. На вкладке **Проект** перечислены все проекты текущего решения и включенные в них библиотеки, если они есть. Установите флажок рядом с MyLib на вкладке **Проекты**, а затем нажмите кнопку **ОК (Рисунок 2.8)**.

Для создания ссылки на файлы заголовка DLL необходимо изменить путь к каталогам включения. Для этого в диалоговом окне **Страницы свойств**

последовательно разверните узлы **Свойства конфигурации** и **C/C++**, а затем выберите **Общие (рисунок 2.9)**. В поле **Дополнительные каталоги включаемых файлов** укажите путь к месту размещения файла заголовка MyLib.h. Можно использовать относительный путь, например ..\MyLib\. Затем нажмите кнопку **ОК** (рисунки 2.10-2.11).

Теперь класс MyMathFuncs можно использовать в приложении. Замените содержимое файла DllApp.cpp следующим кодом:

```
#include "stdafx.h"

#include <iostream>

#include "MyLib.h"

using namespace std;
using namespace MathFuncs;

int main()
{
	double a = 7.4;
	int b = 99;

	cout << "a + b = " <<
		MyMathFuncs::Add(a, b) << endl;
	cout << "a - b = " <<
		MyMathFuncs::Subtract(a, b) << endl;
	cout << "a * b = " <<
		MyMathFuncs::Multiply(a, b) << endl;
	cout << "a / b = " <<
		MyMathFuncs::Divide(a, b) << endl;

	try
	{
		cout << "a / 0 = " <<
			MyMathFuncs::Divide(a, 0) << endl;
	}
	catch (const invalid_argument &e)
```


```
        {
                cout << "Caught exception: " << e.what() << endl;
        }

        return 0;
}
```

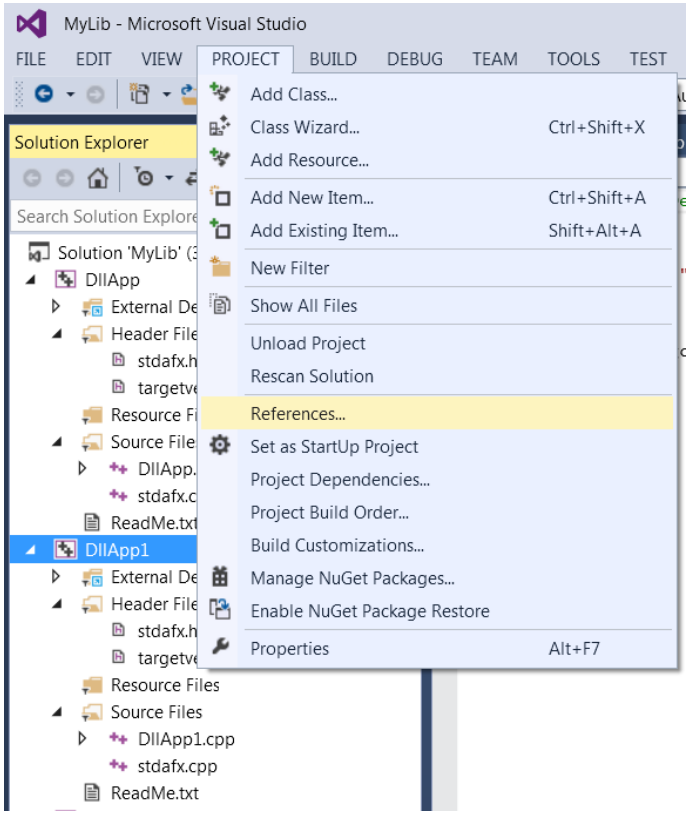


Рисунок 2.6

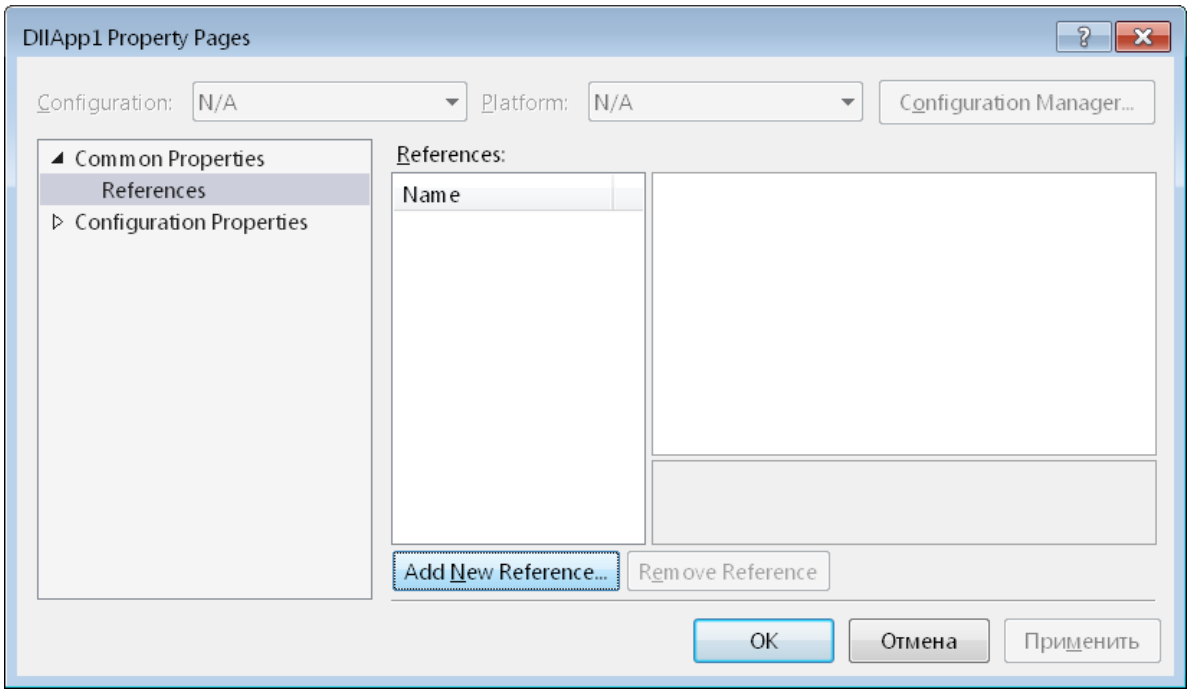


Рисунок 2.7

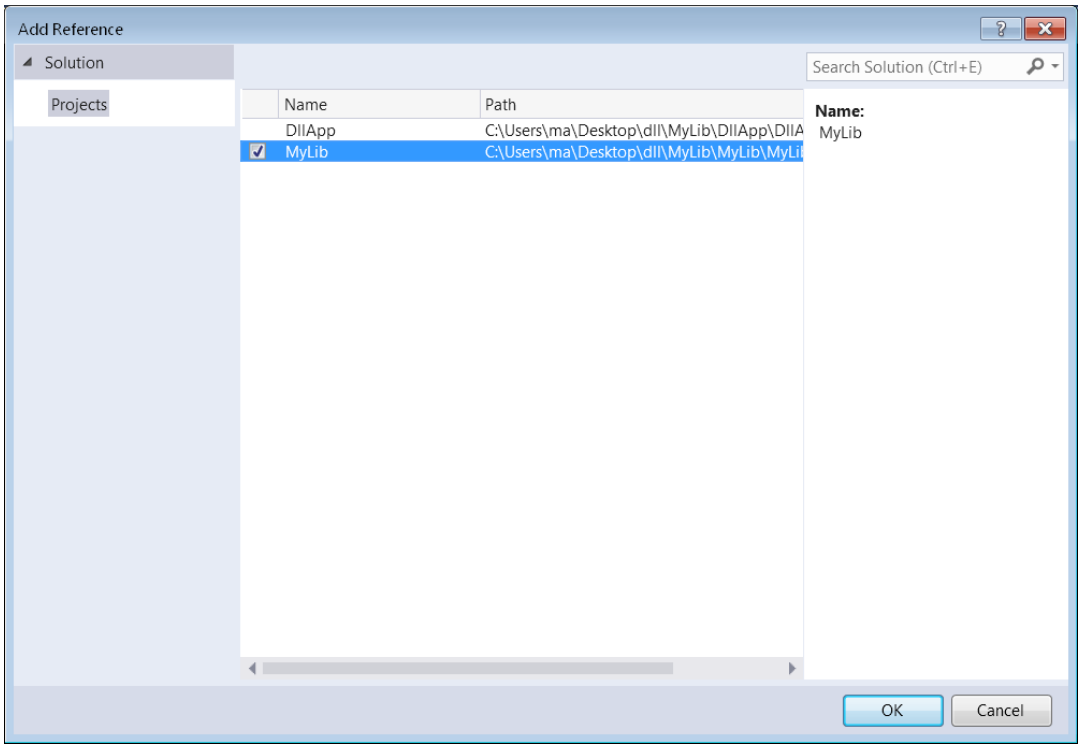


Рисунок 2.8

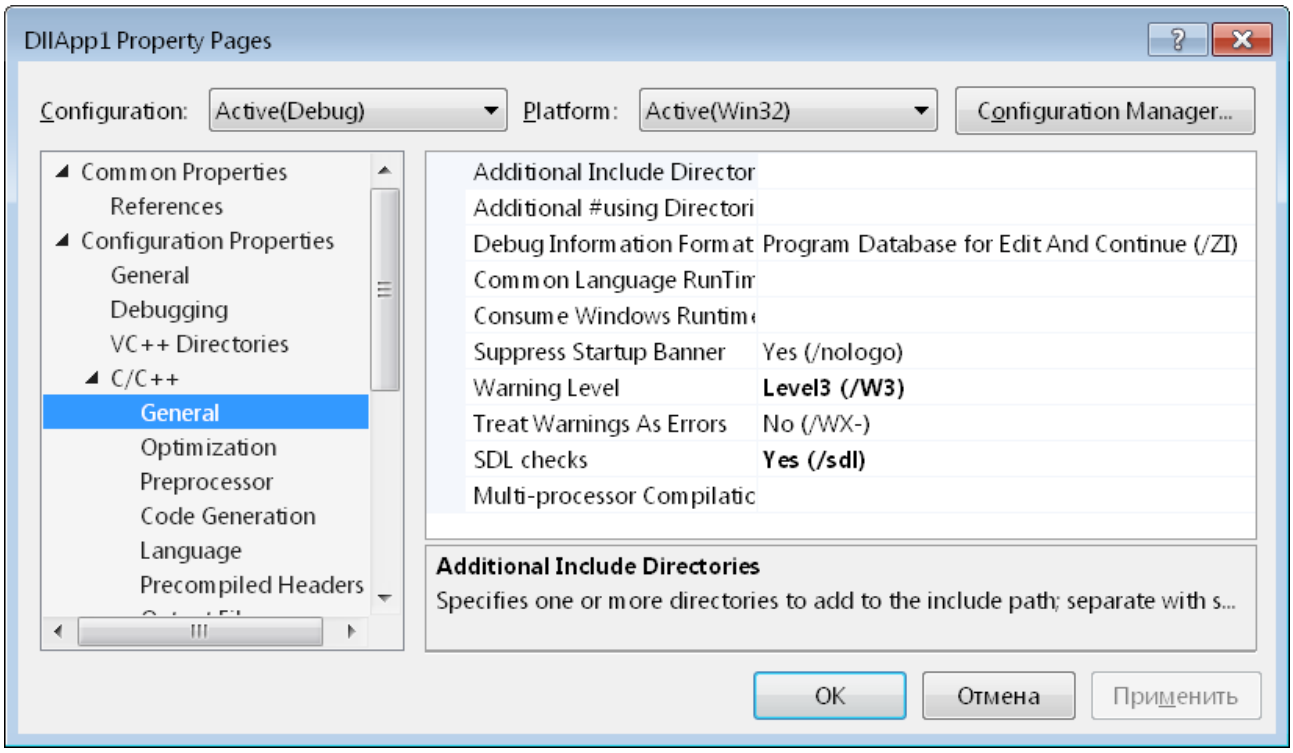


Рисунок 2.9

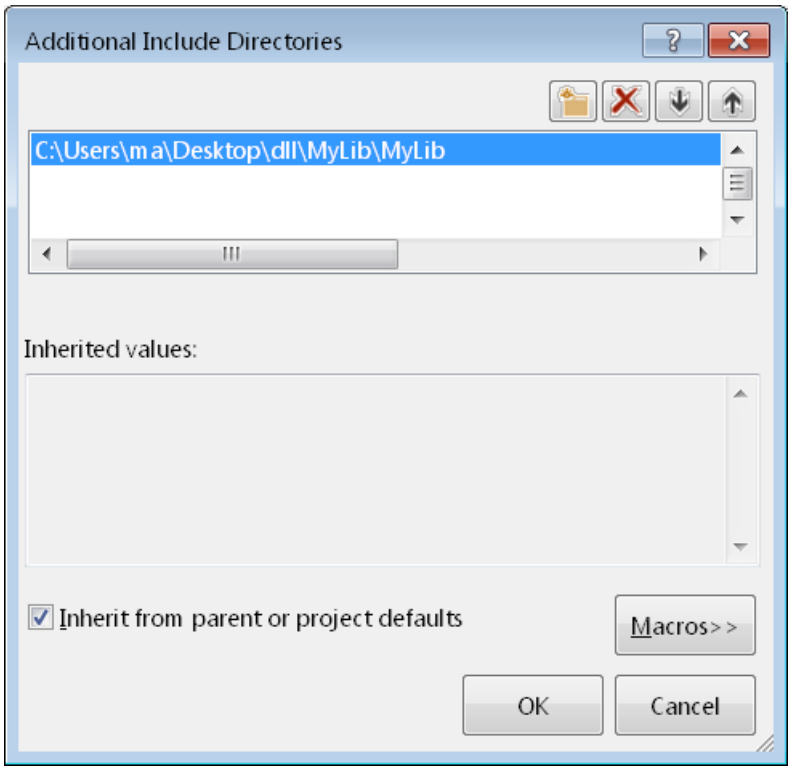


Рисунок 2.10

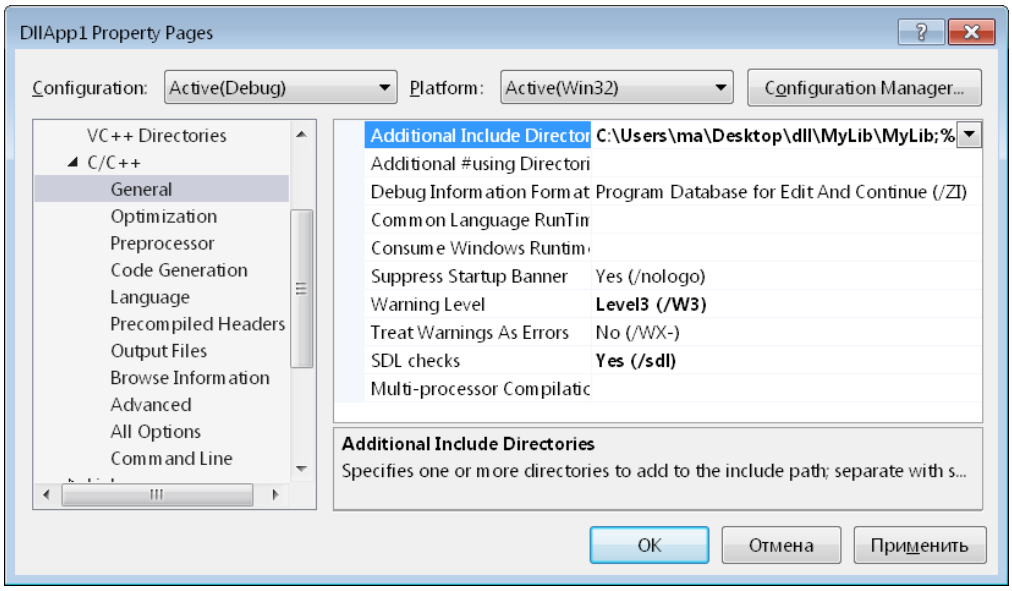


Рисунок 2.11

Соберите исполняемый файл, выбрав команду **Собрать решение** в меню **Сборка**.

Убедитесь в том, что проект DllApp выбран в качестве проекта по умолчанию. В **обозревателе решений** выберите DllApp и затем в меню **Проект** выберите **Назначить запускаемым проектом**. (рисунок 2.12)

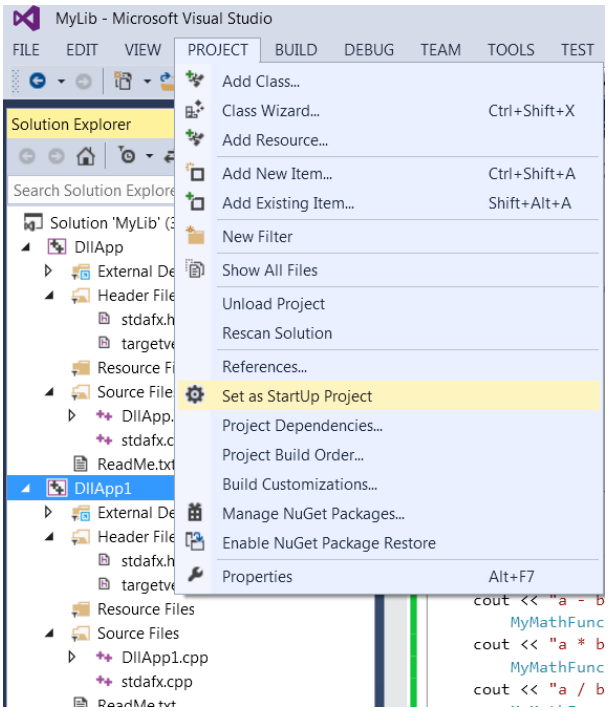
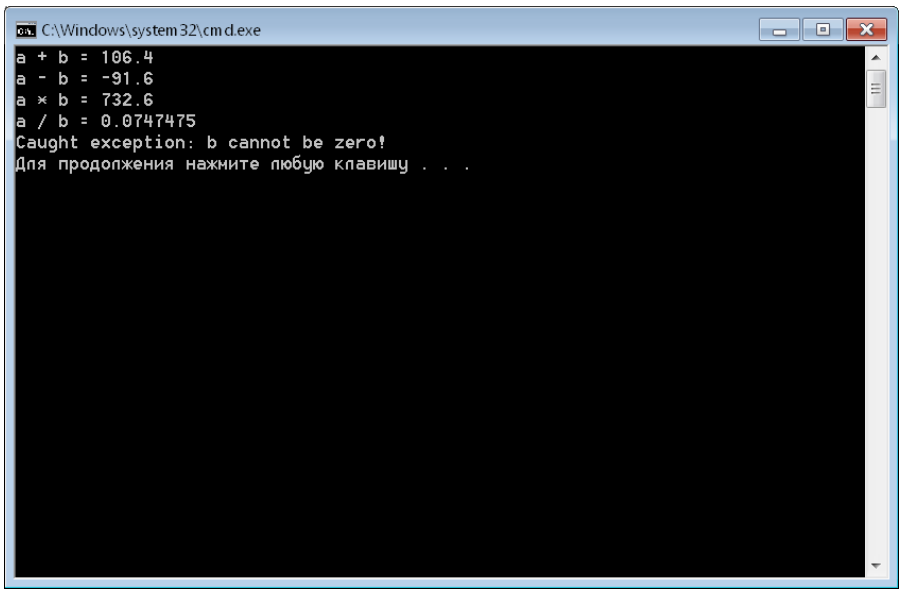


Рисунок 2.12

Чтобы запустить проект, в строке меню выберите **Отладка**, **Запуск без отладки**. Результат выполнения должен выглядеть примерно следующим образом:



2.7. Создание приложения, ссылающегося на DLL (динамическая загрузка)

Создадим простую Dll библиотеку, как было описано ранее и в ее заголовочный файл поместим одну функцию.

```
extern "C" _declspec(dllexport) double Addd(double, double);
```

В .cpp файле реализуем ее:

```
extern "C" _declspec(dllexport) double Addd(double a, double b){
	return a + b;
}
```

Далее, создадим консольное приложение и в его главную функцию добавим следующий код:

```
	typedef double(*funcDll)(double,double); //указатель на функцию, которая будет подгружена из dll, поскольку заглавного файла нет, прототип нужно создавать здесь же, в виде указателя на функцию
	HINSTANCE hDllInstance = LoadLibraryEx(L"MyLib.dll",0, DONT_RESOLVE_DLL_REFERENCES); //загрузка библиотеки, получили дескриптор
	funcDll fpFunction = (funcDll)GetProcAddress(hDllInstance, "Addd"); //получили адрес функции Addd из dll
	double A = fpFunction(3.56, 3.66); // работа с функцией
	cout << A << endl;
	FreeLibrary(hDllInstance); // выгрузили библиотеку.
```

Здесь нужен только файл *.dll, который должен лежать, там где и исполняемый файл приложения, или по известному пути.

Заглавный файл и файл lib не нужны, в отличии от статического.

3 ЗАДАНИЯ

Разработать DLL, содержащий класс для решения задачи согласно варианту. Разработать тестовое приложение для демонстрации работы библиотеки. Предусмотреть в классе обработку всех особых случаев и стандартных ошибок.

Уровень А (1 балл)

Использовать статическое подключение библиотеки.

1. Ввести матрицу размера m×n. Найти максимальный элемент матрицы и его координаты (индекс столбца и индекс строки).
2. Ввести матрицу размера m×n. Найти минимальный элемент и его координаты (индекс столбца и индекс строки).
3. Ввести матрицу размера m×n. Найти максимальный элемент матрицы. Найти количество максимальных элементов.
4. Ввести матрицу размера m×n. Найти минимальный элемент матрицы. Найти количество минимальных элементов.
5. Ввести квадратную матрицу. Найти сумму элементов, лежащих на главных диагоналях матрицы.
6. Ввести две матрицы размера m×n. Посчитать их поэлементную сумму (результат записать в матрицу). Вывести исходные и полученную матрицы.
7. Ввести матрицу размера m×n. Каждый элемент матрицы умножить на минимальный элемент в текущем столбце. Вывести исходную и полученные матрицы.
8. Ввести две матрицы размера m×n. Определить, у какой матрицы среднее арифметическое больше.

9. Ввести матрицу размера m×n. Определить среднее арифметическое матрицы. Найти координаты элемента, наиболее близкого к среднему арифметическому.

10. Ввести квадратную матрицу и число k. Разделить k-ю строку на диагональный элемент этой строки. Вывести исходную и полученную матрицы.

11. Ввести матрицу размера m×n и число k. Удалить k-ю строку из матрицы. Вывести исходную и полученную матрицы.

12. Ввести матрицу размера m×n и число k. Удалить k-й столбец из матрицы. Вывести исходную и полученную матрицы.

13. Ввести матрицу размера m×n. Каждый элемент матрицы умножить на минимальный элемент в текущей строке. Вывести исходную и полученные матрицы.

14. Ввести матрицу размера m×n. Найти сумму элементов в каждой строке. Результат записать в одномерный массив. Вывести исходную матрицу и полученный одномерный массив.

15. Ввести матрицу размера m×n. Заменить все отрицательные элементы нулём. Подсчитать количество замен. Вывести исходную и полученную матрицы.

16. Ввести квадратную матрицу. Поменять местами максимальный и минимальный элементы главной диагонали. Вывести на экран обе матрицы.

17. Ввести матрицу размера m×n. Подсчитать сумму элементов чётных строк, записать результат в одномерный массив.

18. Ввести матрицу размера m×n и число k. Те элементы массива, которые меньше числа k, заменить нулями. Вывести исходную и полученную матрицы.

19. Ввести матрицу размера m×n. Найти строку, сумма элементов которой минимальна. Вывести исходную матрицу и найденную строку.

20. Преобразовать матрицу так, чтобы все элементы выше главной диагонали были нулевыми. Вывести исходную и полученную матрицы.

21. Ввести матрицу размера m×n. Каждый элемент матрицы умножить на максимальный элемент в текущем столбце. Вывести исходную и полученные матрицы.

22. Ввести матрицу размера m×n. Удалить строки и столбцы, состоящие из отрицательных элементов. Вывести исходную и полученную матрицы.

23. Ввести квадратную матрицу. Найти сумму элементов тех строк, у которых на главной диагонали стоят отрицательные числа.

24. Ввести матрицу размера m×n. Найти строку, сумма элементов которой максимальна. Вывести исходную матрицу и найденную строку.

25. Ввести матрицу размера m×n. Найти сумму элементов в каждом столбце. Результат записать в одномерный массив. Вывести исходную матрицу и полученный одномерный массив.

26. Ввести матрицу размера m×n. Подсчитать сумму элементов нечётных строк, записать результат в одномерный массив.

27. Ввести квадратную матрицу. Найти сумму элементов тех строк, у которых на главной диагонали стоят неотрицательные числа.

28. Ввести матрицу размера m×n и число k. К элементам k-й строки добавить соответствующие элементы 1-й строки. Вывести исходную и полученную матрицы.

29. Ввести матрицу размера m×n. Каждый элемент матрицы умножить на максимальный элемент в текущей строке. Вывести исходную и полученные матрицы.

30. Преобразовать матрицу так, чтобы все элементы ниже главной диагонали были нулевыми. Вывести исходную и полученную матрицы.

Уровень Б (+50%)

Выполнить задание уровня А. Подключение созданной библиотеки осуществлять динамически.

4 ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет подается после полной сдачи и защиты лабораторной работы в электронном виде (документ Word).

Отчет должен быть оформлен согласно ДСТУ 3008-95.

В отчет должен содержать следующие пункты:

- титульный лист;
- содержание;
- цель работы;
- постановка задачи;
- аналитические выкладки;
- исходный код программы с комментариями;
- примеры работы программы;
- выводы, с обоснованием результата.

Отчет оценивается максимально в 0,5 балла.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое DLL?
2. Опишите принципы статического подключения DLL.
3. Опишите принципы динамического подключения DLL.