

Лаб. 3. Створення власного класу. Метод дихотомії

Мета: Засвоїти принципи роботи динамічних бібліотек та їх підключення до проектів засобами мови C++ та середовища Visual Studio.

Завдання: Створити власну динамічну бібліотеку з функціоналом із минулих лабораторних (розв'язок квадратного рівняння або метод дихотомії), підключити її та використати у основній програмі.

1. Динамічні бібліотеки .dll

З матеріалів попередніх лабораторних ми знаємо, що для мови C++ характерна наявність двох типів файлів: заголовкових файлів із розширенням .h або .hpp, у яких описується структура (класу, функції, об'єднання, тощо) – заголовки та файли виконуваного коду із розширенням .cpp. Разом пара файлів із однаковою назвою є бібліотекою C++.

Однак, у такому вигляді розповсюджувати код не завжди доцільно, особливо, коли це бізнес логіка, алгоритми якої є закритими. Помістити увесь функціонал у один виконуваний файл часто є неможливо. Тому на практиці часто використовують скомпільовані динамічні бібліотеки DLL (*Dynamic Link Library*) із розширенням .dll. Це, фактично, частина виконуваного файлу, функціонал, реалізований у якій, можна використовувати з іншого коду.

Крім того, як і класи в ООП, динамічні бібліотеки можна використовувати з декількох точок коду (і навіть з декількох додатків), що може суттєво зменшити об'єм великого проекту.

Ще одним плюсом динамічних бібліотек є можливість використовувати їх з інших мов програмування, які інакшим чином є не сумісними. Наприклад, мова C++ є повністю сумісною із мовою C, що дозволяє у більшості випадків використовувати бібліотеки C напяму. Однак, код, написаний такими мовами, як Fortran або Delphi неможливо використати напяму (виключенням є C++ Builder, який має вбудований інтерпретатор Delphi). Тоді є можливість використати функціонал цих мов, оформлений у динамічну бібліотеку.

Варто зауважити, що використання динамічних бібліотек стоїть в основі об'єктних підходів до проектування великих систем, таких як System Object Model.

У світлі цього варто наголосити на відмінностях між власне виконуваними файлами (додатками) та динамічними бібліотеками. Незважаючи на те, що DLL

та додатки є виконуваними модулями, вони мають кілька відмінностей. Найбільш очевидною відмінністю є те, що ви не можете запустити DLL на виконання. Тому з точки зору системи, існують дві основні відмінності між додатками та DLL:

- додатків може одночасно працювати кілька екземплярів, тоді як DLL може мати лише один екземпляр;
- додаток можна запустити як процес, також він може володіти такими речами, як стек, потоки виконання, глобальна пам'ять, черга повідомлень. DLL не може володіти цими речами.

Розглянемо процес створення динамічної бібліотеки засобами Visual Studio C++:

2. Створення динамічних бібліотек

Для нашого прикладу створимо простенький клас із одним методом, який лише додає два числа:

```
class AddNumbers
{
    int Add (int, int);
    float Add (float, float);
};
```

Та його опис:

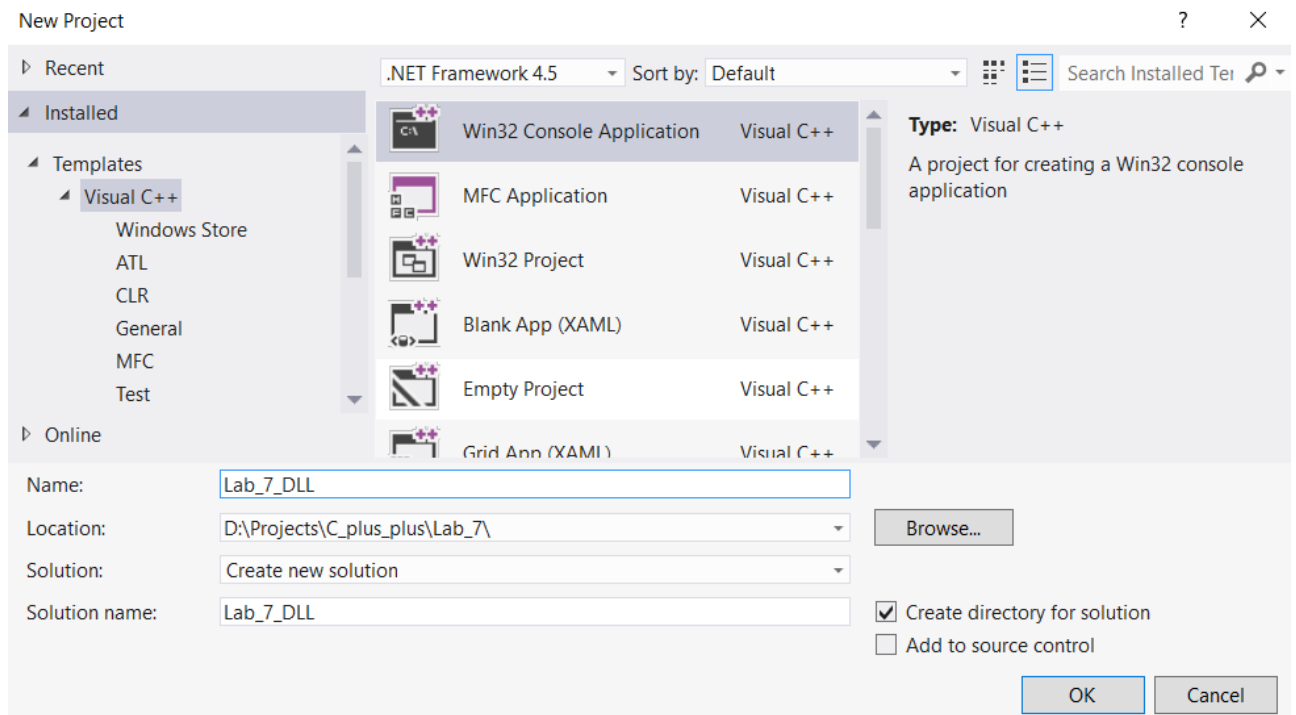
```
int AddNumbers::Add (int first, int second)
{
    return first + second;
}

float AddNumbers::Add (float first, float second)
{
    return first + second;
}
```

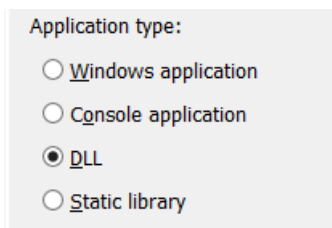
Бачимо, що клас має два методи для різних типів доданків.

Тепер оформимо наш клас у вигляді динамічної бібліотеки.

Для цього створимо новий проект **File => New => Project** та виберемо тип проекту Консольний додаток.



Тиснемо ОК, у наступному вікні – Далі.



Та обираємо тип додатку DLL, завершуємо вибір.

**Welcome to the Win32 Application Wizard**

Overview

Application Settings

These are the current project settings:

- Dynamic link library (DLL)

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

< Previous Next > Finish Cancel

У отриманому проєкті створимо новий заголовковий файл із назвою проєкту (якщо він не був створений автоматично):

Installed

Sort by: Default

Search Installed Templates (Ctrl+E)

Visual C++

Windows Store

UI

Code

HLSL

Data

Resource

Web

Utility

Property Sheets

Test

Graphics

Online

C++ File (.cpp)

Visual C++

Header File (.h)

Visual C++

Type: Visual C++

Creates a C++ header file

Name: Lab_7_DLL.h

Location: D:\Projects\C_plus_plus\Lab_7\Lab_7_DLL\Lab_7_DLL\

Browse...

Add

Cancel

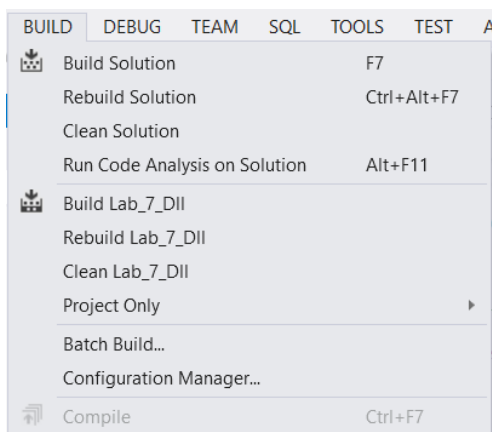
Та додамо наш код у цей файл та файл реалізації.

Треба зауважити, що перед кожним методом треба додати модифікатор `__declspec(dllexport)`. Він показує компілятору, що ці функції призначені для експорту з бібліотеки та можуть біти використані зовнішніми додатками. Тоді ми отримаємо наступний код (у файлі .h):

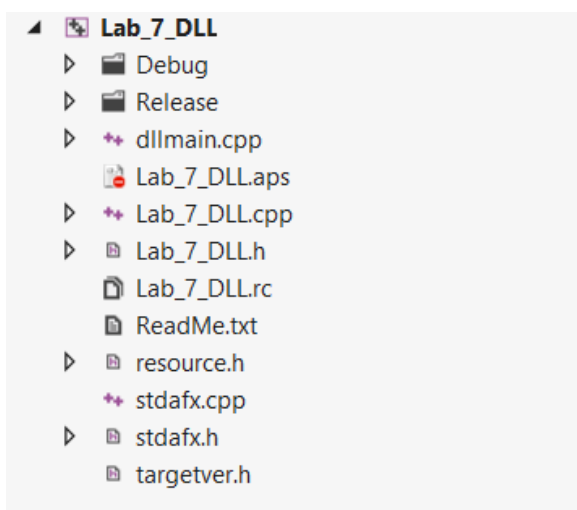
```
#pragma once

class AddNumber
{
public:
    __declspec( dllexport ) int Add (int, int);
    __declspec( dllexport ) float Add (float, float);
};
```






Зберемо проект (скомпілюємо усі файли), натиснувши F7, або через меню Build -> Build solution:



Дерево проекту може виглядати десь наступним чином після компіляції:



Наша бібліотека готова. У каталозі із вихідними файлами отримаємо

	Lab_7_DLL	ilk
	Lab_7_DLL	pdb
	Lab_7_DLL	dll
	Lab_7_DLL	exp
	Lab_7_DLL	lib

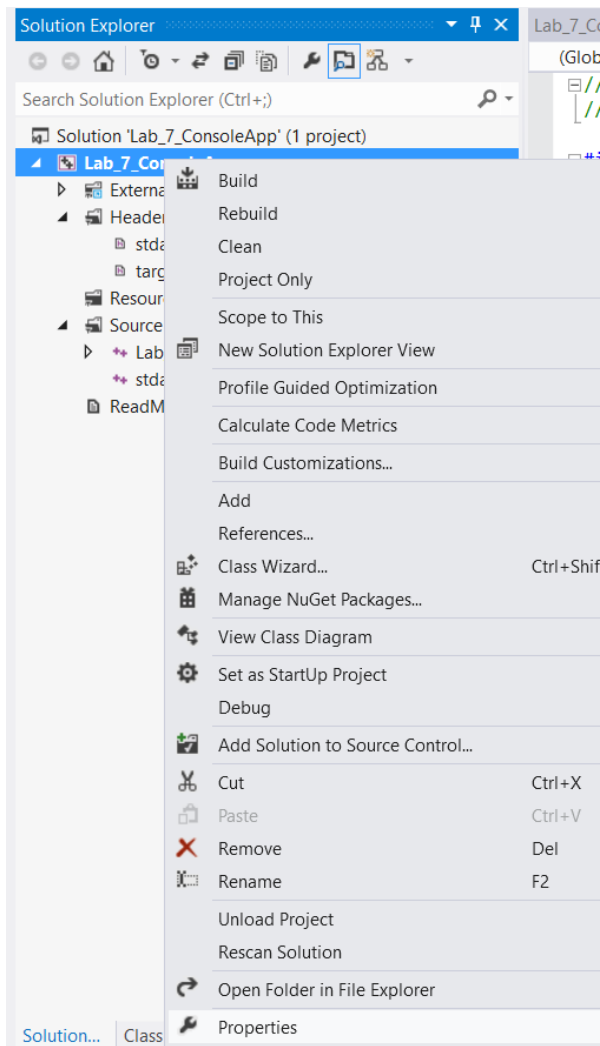
Нас цікавлять файли з розширеннями .dll та .lib. Перший файл то власне бібліотека, а другий потрібен лінкувальнику для пошуку у бібліотеці точок входу до наших функцій.

3. Приєднання динамічної бібліотеки до проекту

Існує два можливі варіанти приєднання (підключення) динамічної бібліотеки: статичний та динамічний. Статичне підключення динамічної бібліотеки відрізняється від статичної бібліотеки тим, що динамічна бібліотека не загружається одразу, а лише тоді, коли виконуваний код звертається до методів, реалізованих у ній, тоді як статична бібліотека загружається одразу у оперативну пам'ять після старту додатку. У другому випадку – динамічному підключенні – керування загрузкою-вигрузкою бібліотеки з пам'яті покладається на програміста і описується в коді. Ми поки що розглянемо перший, простіший варіант. Для цього нам потрібно правильно прописати шляхи до трьох файлів нашого проекту бібліотеки, це файли *.dll, *.lib та *.h, де * -- назва проекту. Слід, однак, зауважити, що для під'єднання бібліотеки за допомогою динамічного варіанту, необхідним є лише файл *.dll, але програміст все одно має знати внутрішню структуру бібліотеки.

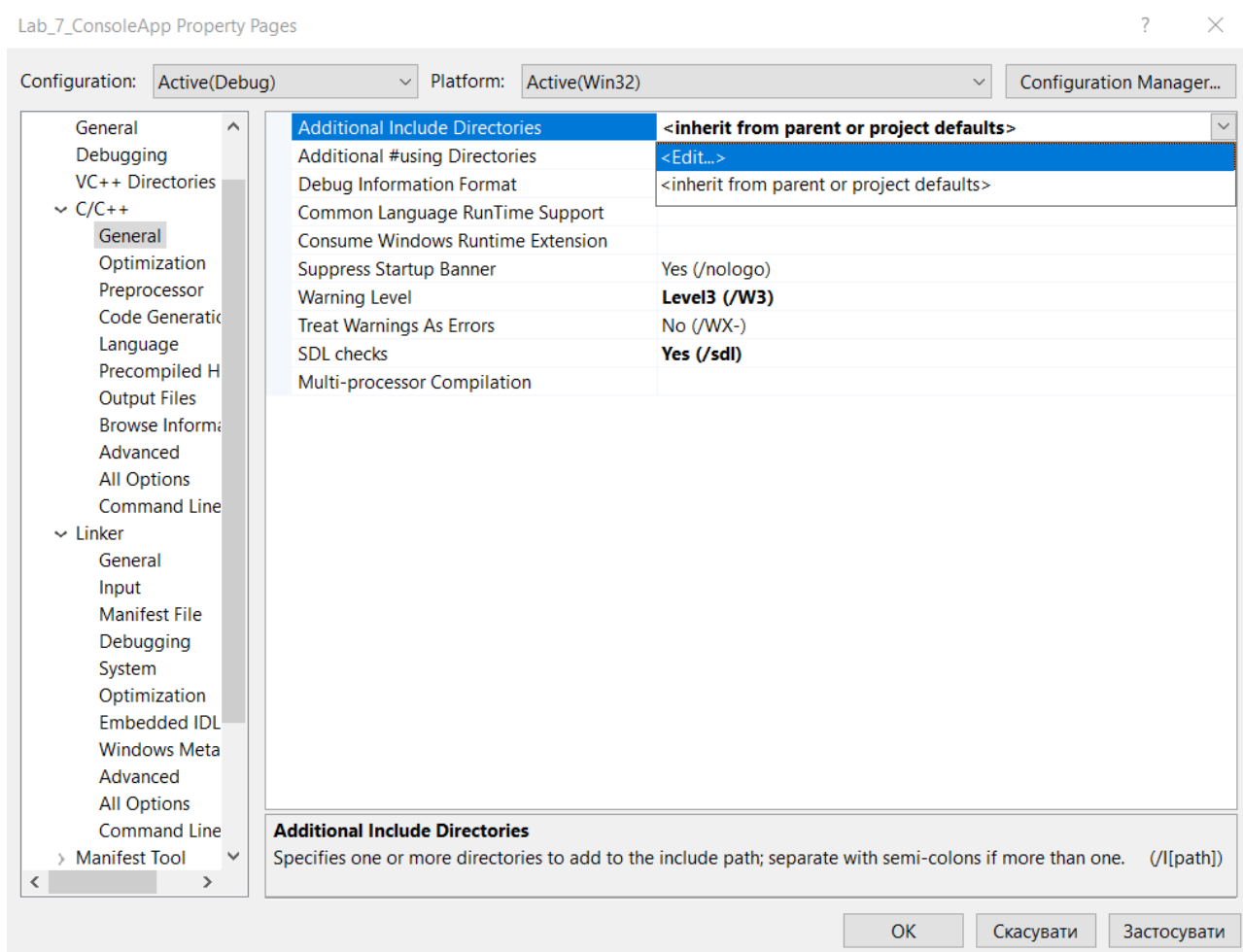
Створимо консольний додаток, у якому будемо використовувати нашу бібліотеку **File => New => Project** та виберемо тип проекту Консольний додаток.

Натиснемо ПКМ на назві проекту у лівому віконечку та виберемо властивості проекту:

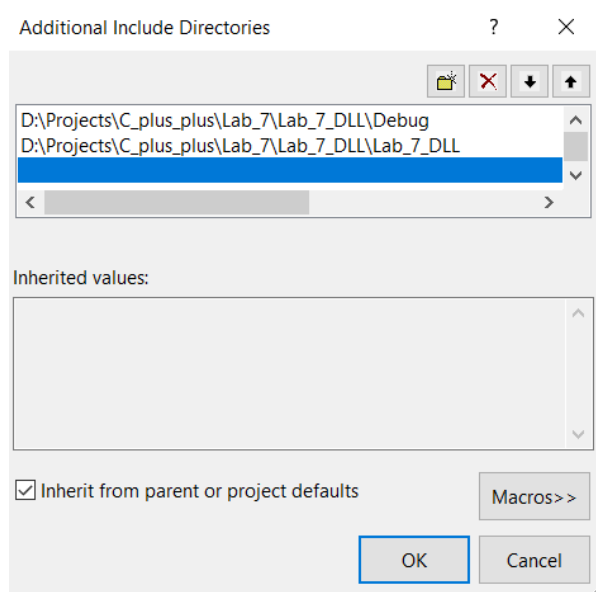


1) Шляхи до директорій із файлами бібліотеки.

У отриманому вікні перейдемо **Configuration Properties** => **C/C++** => **General** оберемо опцію **Additional Include Directories** та пропишемо шляхи до директорій із нашими файлами:

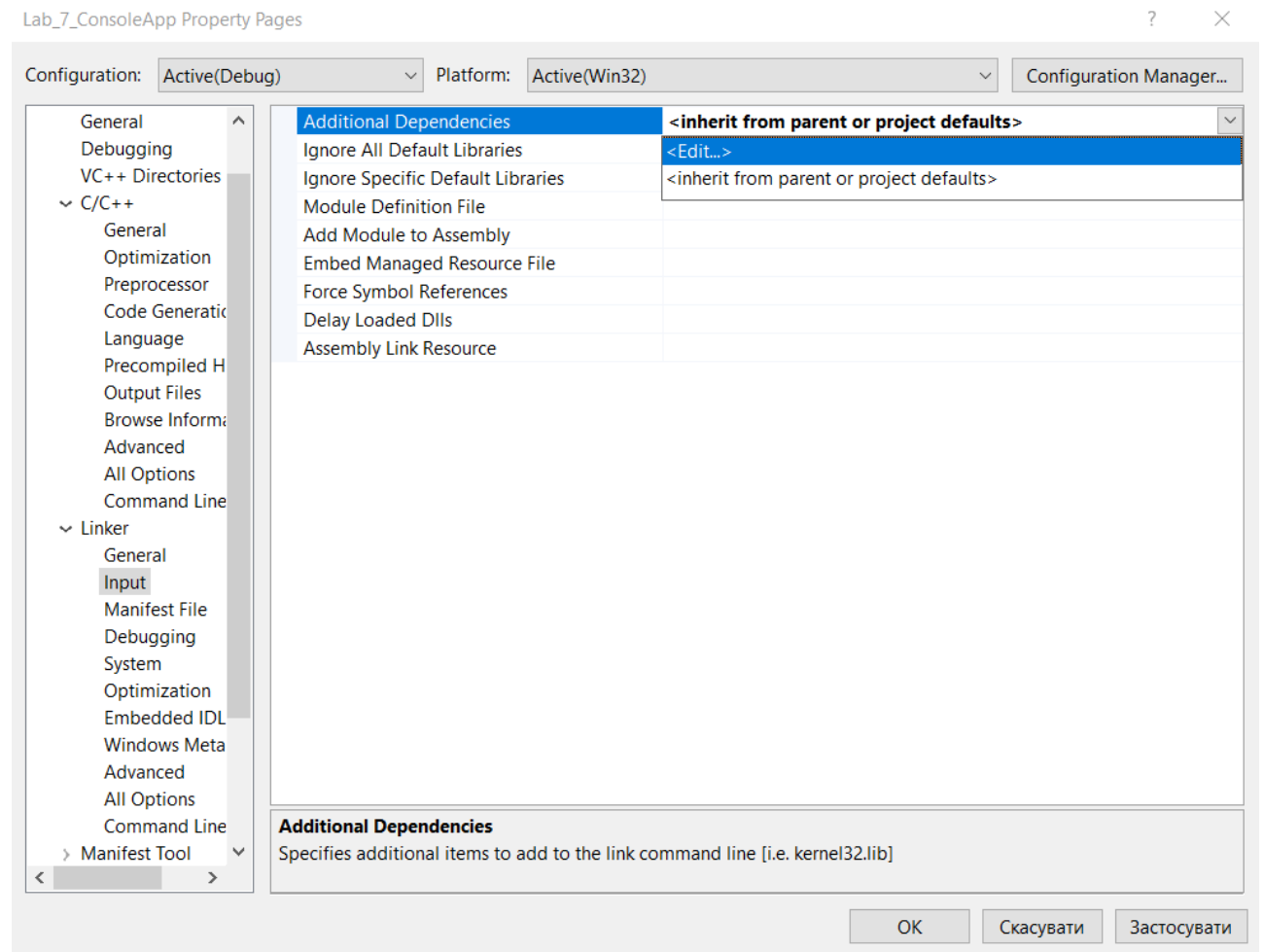


Має виглядати приблизно наступним чином:

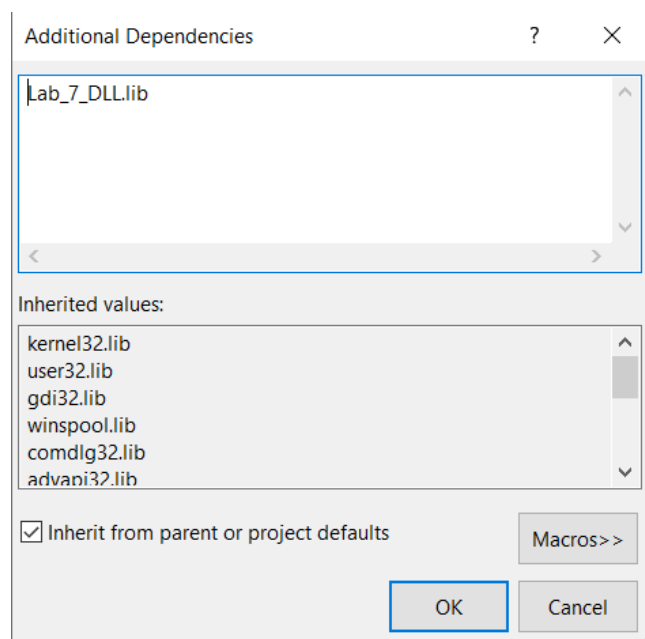


2) Додавання файлу для лінкувальника .lib

Перейдемо Configuration Properties => Linker => Input оберемо Additional Dependencies

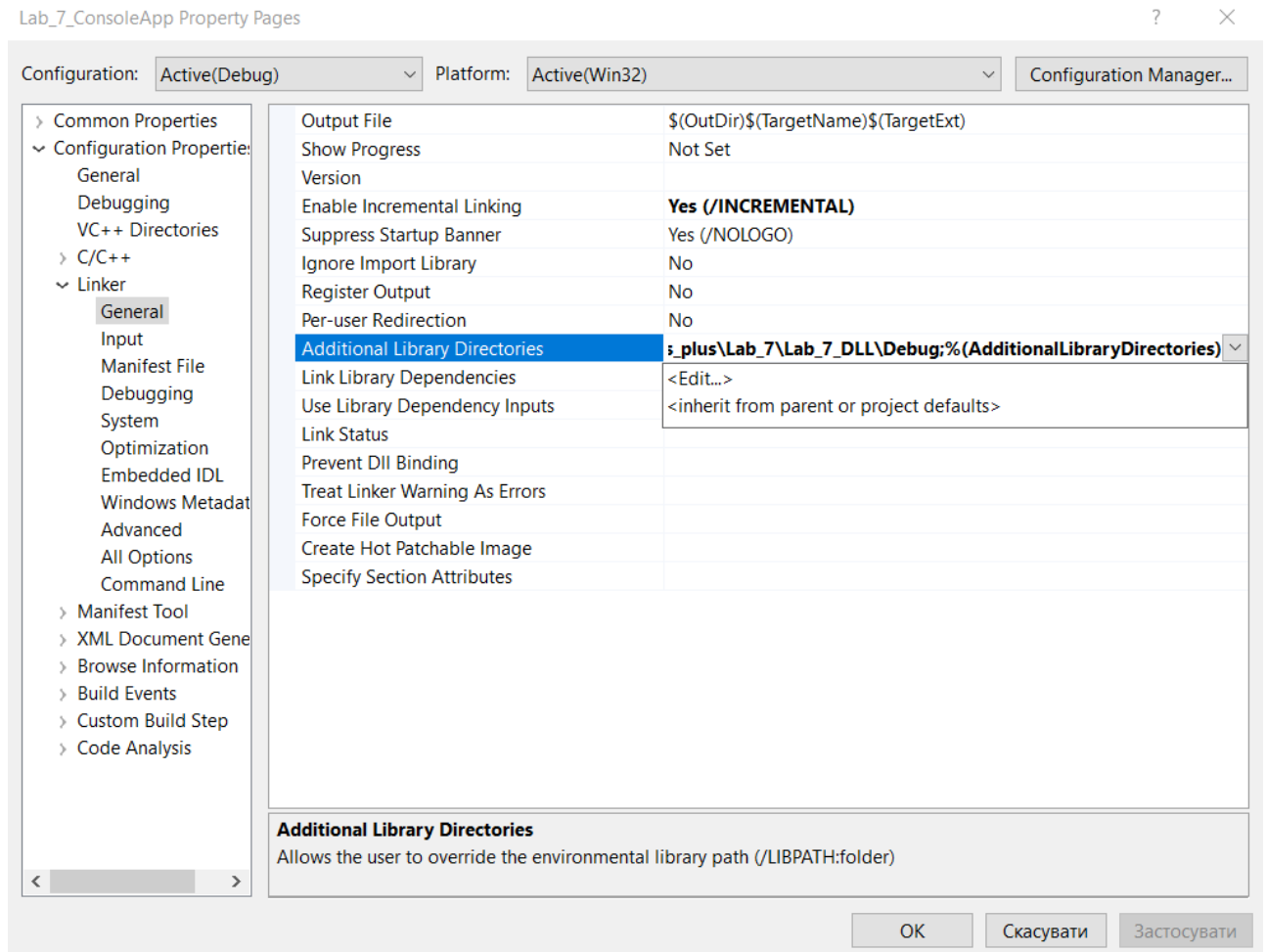


Редагування виглядає приблизно наступним чином:

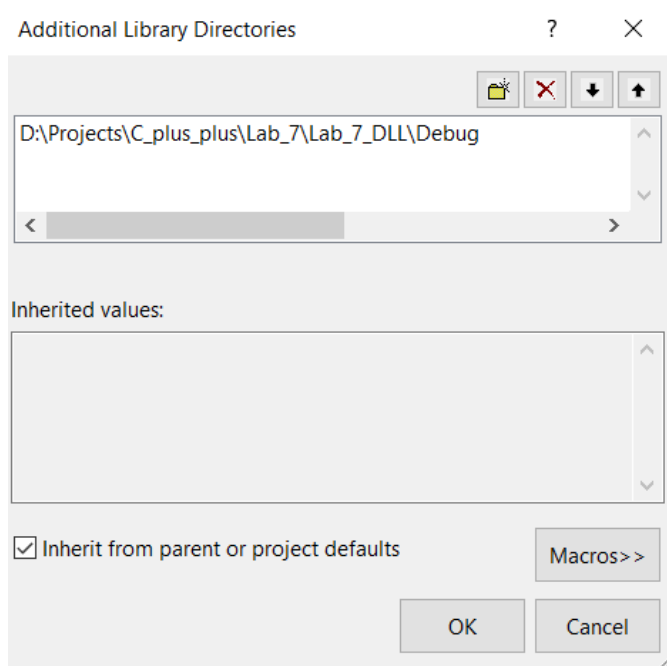


3) Вказання лікувальнику бібліотеки, яка додається та шляху до неї.

Перейдемо **Configuration Properties => Linker => General** оберемо **Additional Library Directories**



Та вкажемо адресу каталогу із бібліотекою

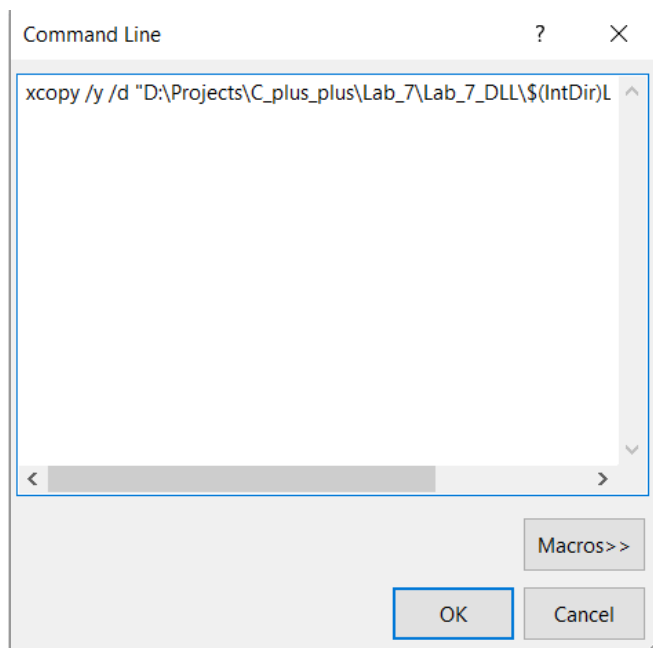
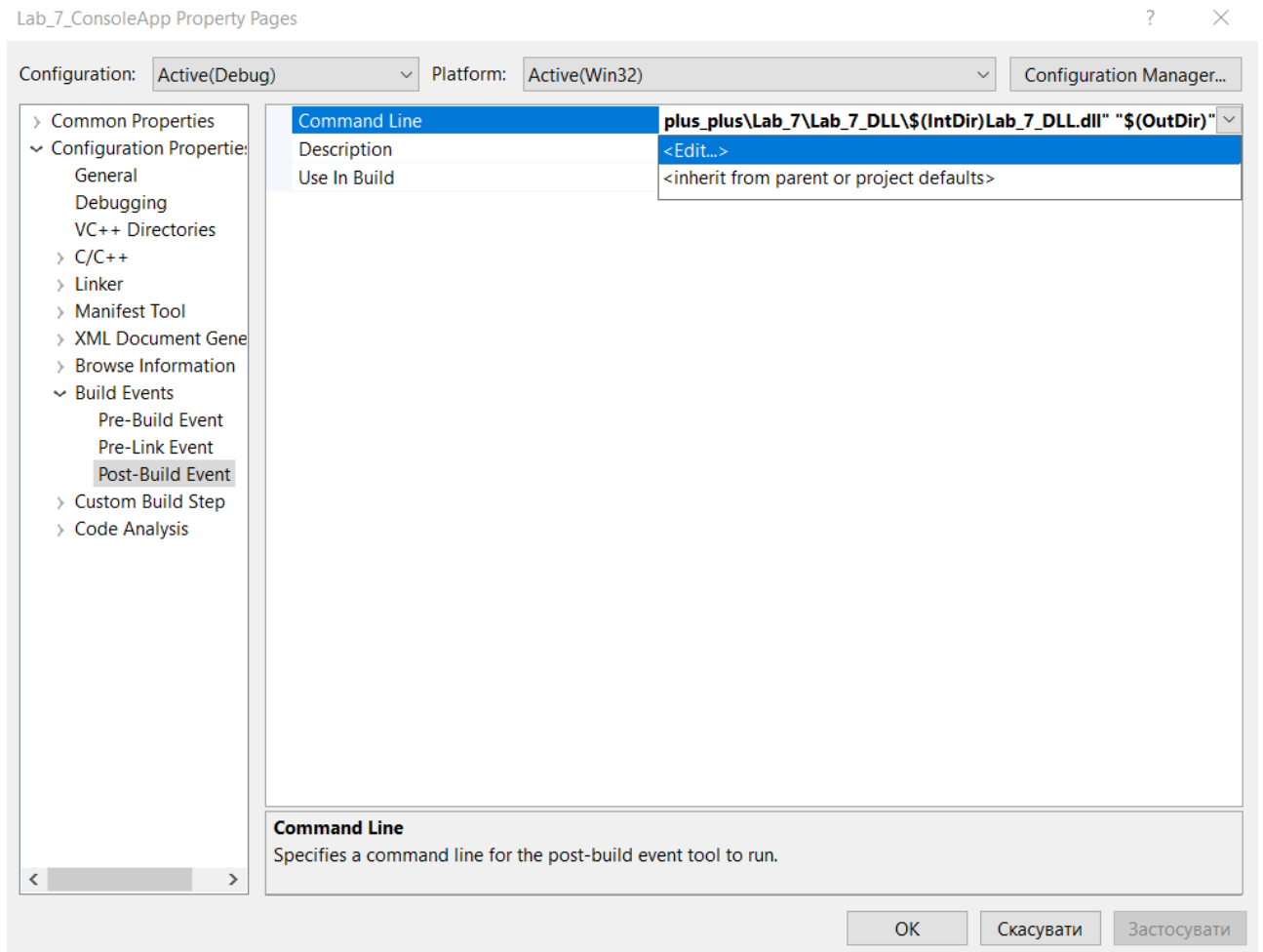


- 4) І останнє перед власне написанням коду, що треба зробити, це прописати шлях власне до самої бібліотеки. До неї буде звертатись додаток вже після компіляції – під час роботи. Тому змінюємо **Configuration Properties => Build Events => Post-Build Event**

У стрічці команд вводимо наступну команду (для нашого випадку)

```
xcopy /y /d "D:\Projects\C_plus_plus\Lab_7\Lab_7_DLL\$(IntDir)
Lab_7_DLL.dll" "$(OutDir)"
```

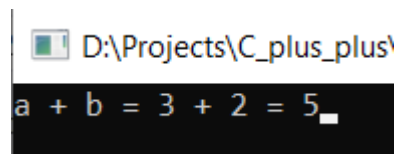
Ця команда копіює бібліотеку із каталогу із бібліотекою у каталог із скомпільованим проектом. Змінні `$(IntDir)` та `$(OutDir)` у нашому випадку `Debug`, але може бути і `Release` залежно від налаштування.



І, нарешті, додамо код у наш проект:

```
int _tmain(int argc, _TCHAR* argv[])
{
    int a = 3;
    int b = 2;
    AddNumber MyAdd;
    int c = MyAdd.Add(a,b);
    std::cout << "a + b = " << a << " + " << b << " = " << c;
    return 0;
}
```

При запуску проекту отримаємо наступний вивід:



```
D:\Projects\C_plus_plus\  
a + b = 3 + 2 = 5_
```

Тобто, наш код та бібліотека працюють.

Звісно, існують інші варіанти налаштування, наприклад, можна скопіювати .dll файл у каталог із проектом та прописати бібліотеку як відкладене завантаження у **Configuration Properties => Linker => Input** параметр **Delay loaded DLLs**.

Додаткове завдання

1. Реалізувати ієрархію класів у динамічній бібліотеці та використати цю бібліотеку у іншому додатку.