ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ

за курсом «Архітектура операційних систем»

студентки групи ПА-18-2

Лобань Ганни Максимівни

кафедра комп’ютерних технологій, ДНУ

2020/2021 н.р.

# Умова

## Загальна умова

Створити проект бібліотеки DLL з однією експортованою та однією не експортованою функцією, яка отримує параметри та виконує операції з ними повертаючи результат (конкретний вид операції вказаний у варіантах завдань ). Обробити стани бібліотеки DLL\_PROCESS\_ATTACH та DLL\_PROCESS\_DETACH, наприклад, вивести повідомлення про підключення та відключення бібліотеки, та про явне чи неявне підключення.

Створити проект двох консольних додатків, які будуть неявно та явно використовувати DLL бібліотеку та її функцію.

Індивідуальний варіант (9):

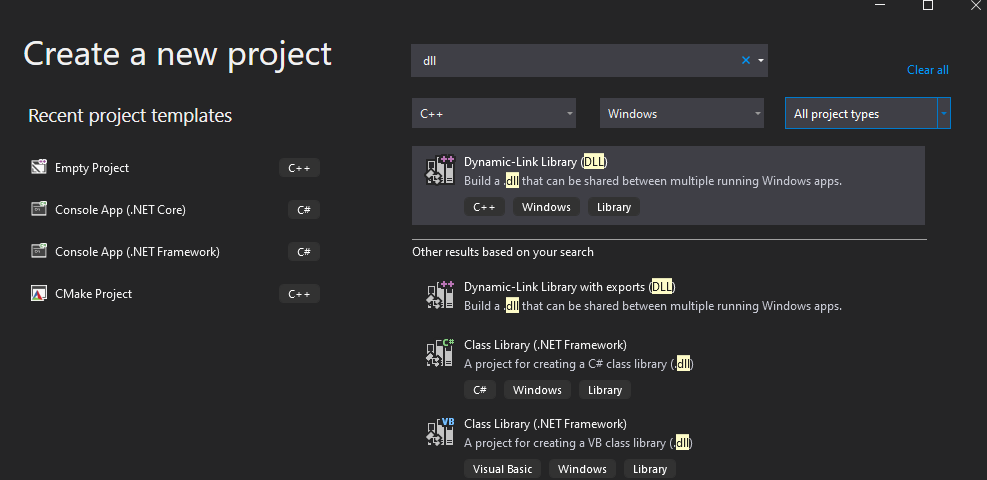
В масиві розміром 5х3 переставити місцями стовбці і рядки.

# Хід виконання

## Створення DLL бібліотеки

Для створення DLL бібліотеки потрібно виконати наступні кроки:

1. Створити проект типу Dynamic-Link Library (DLL):



Visual Studio згенерує для нас проект, де підготує все необхідно заради того, аби спростити нам розробку. Особливу увагу варто приділити файлу dllmain.cpp, де ми будемо виводити на екран події, що відбуваються під час викликів DLL.

1. У файлі dllmain.cpp ми бачимо функцію DllMain – вхідна точка до нашої dll-бібліотеки. Додаємо наступний код, аби повідомляти користувача о звертаннях до бібліотки:

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved) {

switch (ul\_reason\_for\_call) {

case DLL\_PROCESS\_ATTACH:

std::cout << "DLL Process Attached." << std::endl;

break;

case DLL\_THREAD\_ATTACH:

std::cout << "DLL Thread Attached." << std::endl;

break;

case DLL\_THREAD\_DETACH:

std::cout << "DLL Thread Detached." << std::endl;

break;

case DLL\_PROCESS\_DETACH:

std::cout << "DLL Process Detached." << std::endl;

break;

}

return TRUE;

}

1. Створюємо окремий файл matrix.cpp та functions.hpp

#pragma once

#ifdef DEMODLL\_EXPORTS //Visual Studio specifies this name.

#define FUNC\_API \_\_declspec(dllexport)

#else

#define FUNC\_API \_\_declspec(dllimport)

#endif

extern "C" FUNC\_API int\*\* TransformMatrix(

int\*\* matrix, const size\_t rows, const size\_t cols);

1. Створюємо файл matrix.cpp

#include "pch.h"

#include "functions.hpp"

void CreateMatrix(int\*\*\* matrix, const size\_t rows, const size\_t cols) {

(\*matrix) = new int\* [cols];

for (size\_t i = 0; i < cols; ++i) {

(\*matrix)[i] = new int[rows];

}

}

int\*\* TransformMatrix(int\*\* matrix, const size\_t rows, const size\_t cols) {

int\*\* new\_matrix = nullptr;

CreateMatrix(&new\_matrix, rows, cols);

for (size\_t i = 0; i < rows; ++i) {

for (size\_t j = 0; j < cols; ++j) {

new\_matrix[j][i] = matrix[i][j];

}

}

return new\_matrix;

}

1. Компілюємо нашу бібліотеку та отримуємо .dll файл – це і є наша бібліотека. Але, у доданок до нього для неявного зв’язування, Visual Studio створюємо ще й .lib файл, що потрібний для підключення динамічної бібліотеки.

## Неявне зв’язування DLL

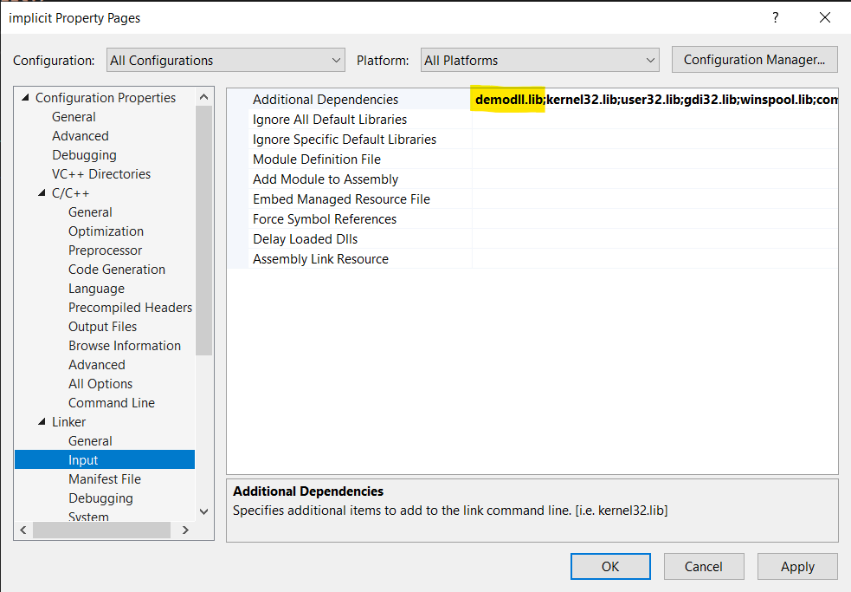
Неявне зв'язування – операційна система завантажує бібліотеку DLL в той момент, коли вона використовується виконуваним файлом. Виконуваний файл клієнта викликає експортовані функції бібліотеки DLL так само, як статично скомпоновані і включені до складу самого виконуваного файлу функції. Процес неявного зв'язування також іноді називають статичної завантаженням або динамічної компонуванням часу завантаження. Щоб використовувати dll неявно необхідно зробити наступне:

1) Створити або пусте, або консольний додаток (в даному випадку, я робив з порожнім, але можна і з консольним), можна в тому ж Solution, робимо стандартний шаблон програми, пишемо основну логіку нашої програми.

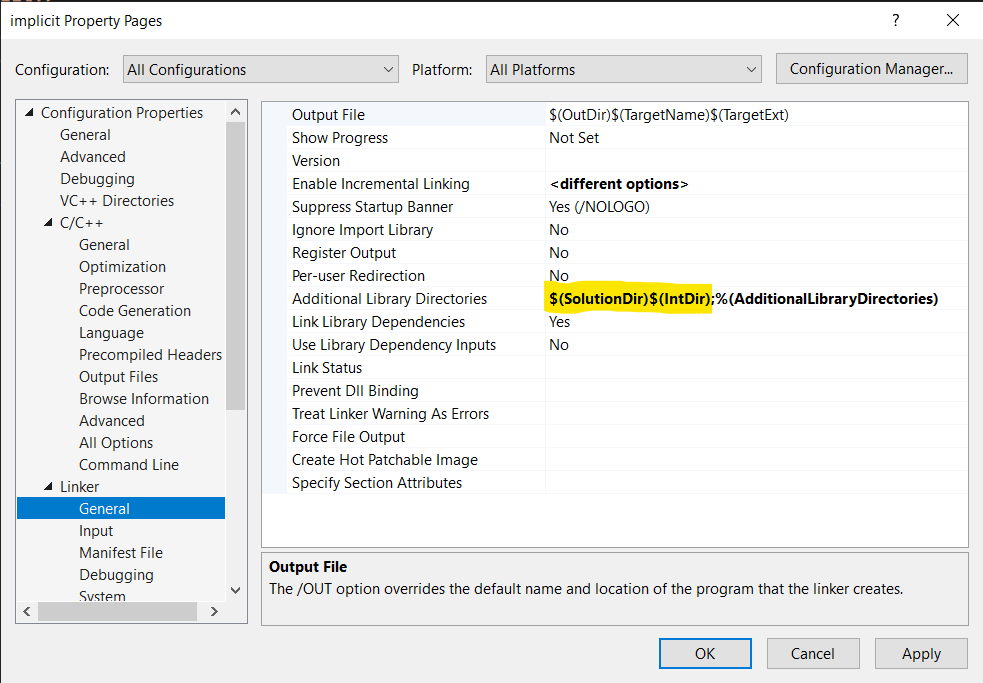
2) Додаємо .h\.hpp хедер нашої .dll бібліотеки (в моєму випадку, це functions.hpp), в якому визначені всі прототипи функцій, які визначені в динамічній бібліотеці.

3) У властивостях проекту підключаємо додаткову бібліотеку (яку для нас люб'язно згенерувала Visual Studio) .lib, що містить покажчики потрібних нам функцій.

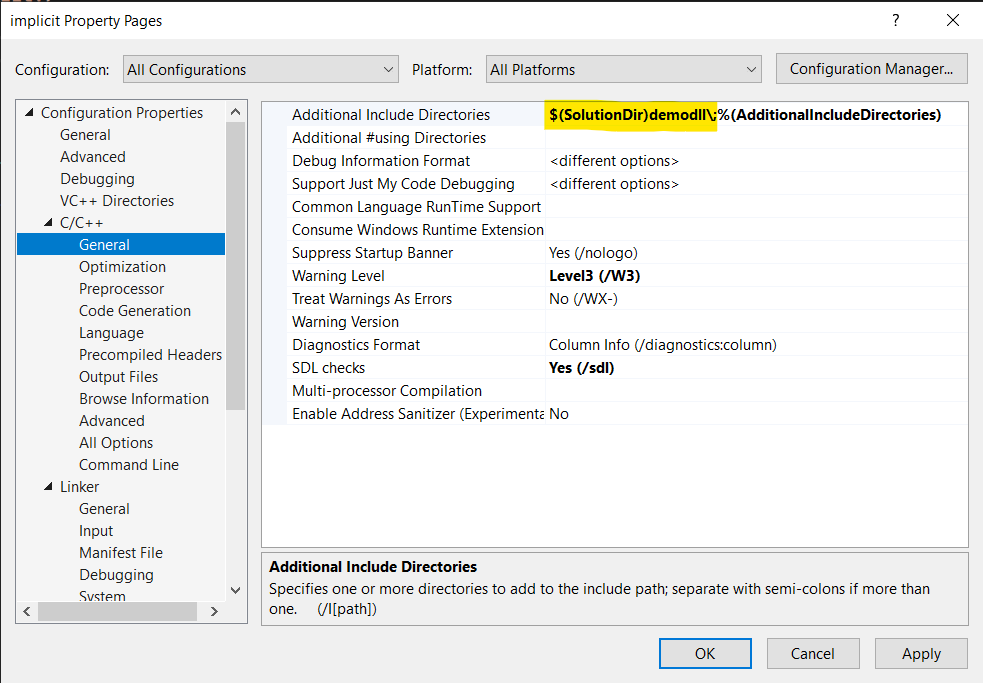
У моєму випадку, студія згенерувала файли з назвами demodll.dll і demodll.lib, так ось другий з них я і підключаю. Це видно на скріншоті нижче.



1. 1) Налаштовуємо області видимості, щоб Visual Studio знала, де знаходяться і .lib, і .dll файли – я це зробив тут же у властивостях проекту. За замовчуванням, Visual Studio всі виконувані бінарники кидає в папку Debug в однойменному режимі збірки (або в Release, якщо режим збірки – Release) – я просто додав цю папку в Additional Library Directories, правда зробив це псевдонімом, і тепер воно буде шукати відповідний файл у відповідній папці незалежно від режиму збірки.



До речі, те ж саме треба зробити і для .h\.hpp файлу, щоб студія його теж бачила (ну, або використовувати відносний шлях). Особисто я додав шлях до нашого проекту з динамічної бібліотекою.



Після всіх цих маніпуляцій, код у мене став таким:

#include <iostream>

#include "functions.hpp"

void PrintMatrix(int\*\* matrix, const size\_t rows, const size\_t cols) {

for (size\_t i = 0; i < rows; ++i) {

printf("+");

for (size\_t q = 0; q < 7 \* cols; ++q) putchar('-');

printf("\b+\n");

for (size\_t j = 0; j < cols; ++j) {

printf("| %4d ", matrix[i][j]);

}

std::cout << "|\n";

}

printf("+");

for (size\_t q = 0; q < 7 \* cols; ++q) putchar('-');

printf("\b+\n");

}

int main() {

constexpr size\_t DIM1 = 5;

constexpr size\_t DIM2 = 3;

auto\*\* matrix = new int\*[DIM1];

int\*\* new\_matrix = nullptr;

for (size\_t i = 0; i < DIM1; ++i) {

matrix[i] = new int[DIM2];

for (size\_t j = 0; j < DIM2; ++j) {

matrix[i][j] = rand() % 199 - 99;

}

}

PrintMatrix(matrix, DIM1, DIM2);

new\_matrix = TransformMatrix(matrix, DIM1, DIM2);

PrintMatrix(new\_matrix, DIM2, DIM1);

for (size\_t i = 0; i < DIM1; ++i)

delete[] matrix[i];

delete[] matrix;

if (new\_matrix) {

for (size\_t i = 0; i < DIM2; ++i)

delete[] new\_matrix[i];

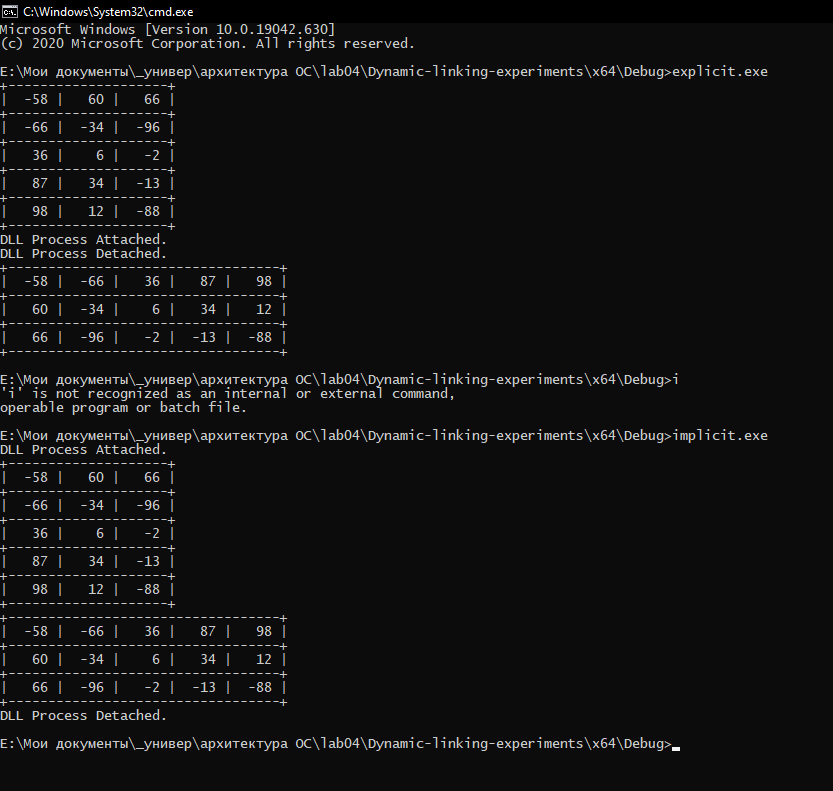
delete[] new\_matrix;

}

return 0;

}

Тепер можна запускати компіляцію проекту:



Як бачимо, проект запустився и працює.

## Явне зв’язування DLL

Явне зв'язування – операційна система завантажує бібліотеку DLL за запитом під час виконання. Виконуваний файл, який використовує бібліотеку DLL, повинен явно завантажувати і вивантажувати її. Крім того, в ньому повинен бути налаштований покажчик функції для доступу до кожної використовуваної функції з бібліотеки DLL. На відміну від викликів функцій в статично скомпонована або неявно пов'язаної бібліотеці DLL, при роботі з явно пов'язаної DLL виконуваний файл клієнта повинен викликати експортовані функції за допомогою покажчиків функцій. Процес явного зв'язування також іноді називають динамічної завантаженням або динамічної компонуванням часу виконання.

Щоб зв'язати динамічну бібліотеку явно, потрібно:

1) Створити новий проект (консольний або порожній)

2) Підключити бібліотеку "Windows.h", оскільки саме в ній знаходяться API для підключення DLL явно.

3) Далі буде досить складно щось пояснювати без прикладу, так що я просто вставлю тут код з коментарями в місцях, що стосуються імпортування та використання dll.

#include <windows.h>

#include <iostream>

#define DLL\_NAME "demodll.dll"

typedef int\*\* (\_\_cdecl\* FUNC)(int\*\*, const size\_t, const size\_t);

void PrintMatrix(int\*\* matrix, const size\_t rows, const size\_t cols) {

for (size\_t i = 0; i < rows; ++i) {

printf("+");

for (size\_t q = 0; q < 7 \* cols; ++q) putchar('-');

printf("\b+\n");

for (size\_t j = 0; j < cols; ++j) {

printf("| %4d ", matrix[i][j]);

}

std::cout << "|\n";

}

printf("+");

for (size\_t q = 0; q < 7 \* cols; ++q) putchar('-');

printf("\b+\n");

}

int main() {

HINSTANCE hinstLib;

FUNC ProcAdd;

BOOL fFreeResult;

constexpr size\_t DIM1 = 5;

constexpr size\_t DIM2 = 3;

auto\*\* matrix = new int\* [DIM1];

int\*\* new\_matrix = nullptr;

for (size\_t i = 0; i < DIM1; ++i) {

matrix[i] = new int[DIM2];

for (size\_t j = 0; j < DIM2; ++j) {

matrix[i][j] = rand() % 199 - 99;

}

}

PrintMatrix(matrix, DIM1, DIM2);

hinstLib = LoadLibrary(TEXT(DLL\_NAME));

if (hinstLib) {

ProcAdd = (FUNC)GetProcAddress(hinstLib, "TransformMatrix");

if (ProcAdd) {

new\_matrix = (ProcAdd)(matrix, DIM1, DIM2);

} else {

std::cerr << "Failed to find the function. Error: " << GetLastError() << std::endl;

}

fFreeResult = FreeLibrary(hinstLib);

} else {

std::cerr << "Error: failed to find demodll.dll! Error: " << GetLastError() << std::endl;

}

PrintMatrix(new\_matrix, DIM2, DIM1);

for (size\_t i = 0; i < DIM1; ++i) {

delete[] matrix[i];

}

delete[] matrix;

if (new\_matrix) {

for (size\_t i = 0; i < DIM2; ++i) {

delete[] new\_matrix[i];

}

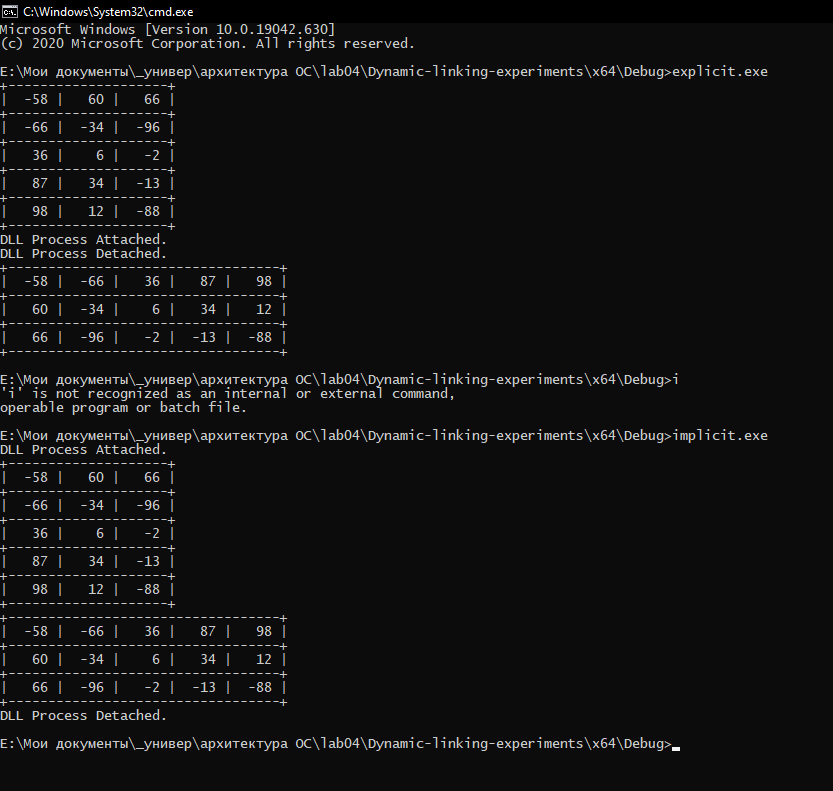
delete[] new\_matrix;

}

return 0;

}

Тепер можна запускати проект на виконання:



Як ми бачимо, все запустилось. І тепер ми можемо наочно побачити відмінності явного і неявного підключення бібліотек - в неявному бібліотека завантажилася перед запуском програми, а в явному – тільки тоді, коли ми явно це вказали. Звідси ми і бачимо відмінності в розташуваннях виведених рядків DLL Process Attached/Detached.