Міністерство освіти і науки України
Національний технічний університет України „КПІ”
Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки
інформації та управління

# ЗВІТ

до лабораторної роботи № 9

з дисципліни ООП

**Виконав студент** *ІП-61 Кушка Михайло Олександрович*

(№ групи, прізвище, ім’я, по батькові )

**Прийняв** *Головченко М.М.*

(посада, прізвище, ім’я, по батькові )

Київ 2017

**ЗМІСТ**

# 1. МЕТА РОБОТИ

Мета роботи - вивчити особливості використання DLL, розглянути принципи статичного і динамічного підключення.

# 2. ПОСТАНОВКА ЗАДАЧІ

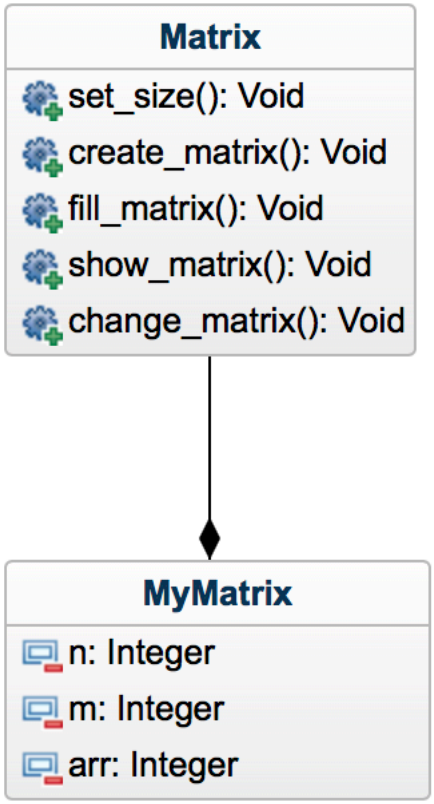
Ввести матрицю розміру m x n. Замінити всі негативні елементи нулем. Підрахувати кількість замін. Вивести вихідну і отриману матриці.

# 3. АНАЛІТИЧНІ ВИКЛАДКИ

DLL (англ. Dynamic Link Library - «бібліотека динамічного компонування», «динамічно підключається») в операційних системах Microsoft Windows і IBM OS/2 - динамічна бібліотека, що дозволяє багаторазове використання різними програмними додатками. До DLL відносяться також елементи управління ActiveX і драйвери. У системах UNIX аналогічні функції виконують так звані загальні об'єкти (англ. shared objects).

Формат файлів DLL дотримується тих самих домовленостей, що і формат виконуваних файлів, поєднуючи код, таблиці і ресурси, відрізняючись лише інтерпретацією деяких полів.

# 4. UML-ДІАГРАМА КЛАСІВ



# 5. ВИХІДНИЙ КОД ПРОГРАМИ

## MatrixDll

### MatrixDll.h

```
// MatrixDll.h


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


class Matrix {
public:
        virtual void set_size(int, int) = 0;
    virtual void create_matrix() = 0;
    virtual void fill_matrix(int min=-9, int max=9) =
0;
    virtual void show_matrix() = 0;
        virtual void change_matrix() = 0;
};
```

### MatrixDll.cpp

```
//
MatrixDll.cpp
: Defines the
exported
functions for
the DLL
application.
                //


#include "stdafx.h"
#include "MatrixDll.h"


class MyMatrix : public Matrix {
    int n, m;
       int** arr;
public:
```

```
        // MyMatrix constructor
        MyMatrix() {};


        // MyMatrix destructor
        ~MyMatrix() {
                for (int i = 0; i < n; ++i) {
                        delete arr[i];
                }
        }


        // Set size
        void Matrix::set_size(int _n, int _m) {
                n = _n;
                m = _m;
        }


    // Create matrix
    void Matrix::create_matrix() {
         arr = new int*[n];
                for (int i = 0; i < n; ++i) {
                        arr[i] = new int[m];
                }
    }


        // Fill matrix
        void Matrix::fill_matrix(int min, int max) {
                // Init random
                std::random_device rd;
         // only used once to initialise (seed) engine
                std::mt19937 rng(rd());                                                 //
random-number engine used (Mersenne-Twister in this case)
                std::uniform_int_distribution<int> uni(min,max); // guaranteed unbiased


                for (int i = 0; i < n; ++i) {
                        for (int j = 0; j < m; ++j) {
                                arr[i][j] = uni(rng);
                        }
                }
        }


        // Show matrix
        void Matrix::show_matrix() {
```

```
                cout << endl;
                for (int i = 0; i < n; ++i) {
                        for (int j = 0; j < m; ++j) {
                                cout << setw(4) << arr[i][j] << " ";
                        }
                        cout << endl;
                }
                cout << endl;
        }


        // Change matrix
        void Matrix::change_matrix() {
                int counter = 0;
                for (int i = 0; i < n; ++i) {
                        for (int j = 0; j < m; ++j) {
                                if (arr[i][j] < 0) {
                                        arr[i][j] = 0;
                                        counter++;
                                }
                        }
                }
                cout << "Number of numbers less than zero: " << counter << endl;
        }
};


extern "C" __declspec(dllexport) Matrix* __cdecl create_klass()
{
    return new MyMatrix();
}
```

## stdafx.h

```
//
stdafx.h:
включаемый
файл для
стандартных
системных
включаемых
файлов
        // или включаемых файлов для конкретного проекта, которые часто используются, но
        // не часто изменяются
        //


        #pragma once
```

```
#include "targetver.h"


#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <iostream>
#include <random>
#include <iomanip>


using namespace std;
```

# MatrixExec

## MatrixExec.cpp

```
/
//
MatrixExec.cpp


    #include "stdafx.h"
    #include "MatrixDll.h"


    // A factory of IKlass-implementing objects looks thus
    typedef Matrix* (__cdecl *iklass_factory)();


    int main()
    {
        // Load the DLL
        HINSTANCE dll_handle = ::LoadLibrary(TEXT("MatrixDll.dll"));
        if (!dll_handle) {
            cerr << "Unable to load DLL!\n";
            return 1;
        }


        // Get the function from the DLL
        iklass_factory factory_func = reinterpret_cast<iklass_factory>(
            ::GetProcAddress(dll_handle, "create_klass"));
        if (!factory_func) {
            cerr << "Unable to load create_klass from DLL!\n";
            ::FreeLibrary(dll_handle);
```

```
        return 1;
    }


    // Ask the factory for a new object implementing the IKlass
    // interface
    Matrix* instance = factory_func();


       string n_str, m_str;
       int n, m;
       while (true) {
              try {
                     cout << "Please enter size of the matrix" << endl;
                     cout << "n: ";
                     cin >> n_str;
                     cout << "m: ";
                     cin >> m_str;


                     n = stoi(n_str);
                     m = stoi(m_str);
                     if (n < 1 || m < 1) {
                            throw invalid_argument("N and M nust be greater than
zero");
                     }
                     break;
              } catch (const invalid_argument &e) {
                     cerr << e.what() << endl;
              } catch(...) {
                     cerr << "N and M must be integers, not string" << endl;
              }
       }


    // Main part
       instance->set_size(n, m);
    instance->create_matrix();
    instance->fill_matrix();
       instance->show_matrix();
       instance->change_matrix();
       instance->show_matrix();


    // Destroy it explicitly
    ::FreeLibrary(dll_handle);
```

```
    return 0;
}
```

## stdafx.h

```
// stdafx.h: включаемый файл для стандартных системных включаемых файлов
// или включаемых файлов для конкретного проекта, которые часто используются, но
// не часто изменяются
//

#pragma once

#include "targetver.h"
#include <Windows.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include <tchar.h>

using namespace std;
```

# 6. ПРИКЛАДИ РОБОТИ ПРОГРАМИ

```
Enter dimensions:
3
7
Enter fill range:
-90
5

 -88  -27  -66  -59  -25   -2  -10
 -85    3  -12  -20  -72   -4  -51
 -48   -5  -68  -80  -43  -34  -47

Number of numbers less than zero: 20

   0    0    0    0    0    0    0
   0    3    0    0    0    0    0
   0    0    0    0    0    0    0

Для продолжения нажмите любую клавишу . . .
```

```
Enter dimensions:
4
5
Enter fill range:
-8
8

   0   -3    0   -1    2
  -3    0    2    2    5
  -6    0    7    0   -6
  -1    7   -3    6    8

Number of numbers less than zero: 7

   0    0    0    0    2
   0    0    2    2    5
   0    0    7    0    0
   0    7    0    6    8

Для продолжения нажмите любую клавишу . . .
```

# 7. ВИСНОВКИ

У даній лабораторній роботі я навчився створювати та використовувати власні dll-файли, ознайомився з особливостями їхнього оголошення та реалізації. Розробив власну бібліотеку динамічної компоновки та використав її у своїй програмі.