Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №6

з дисципліни
«ООП»

Виконав:

Студент 2-го курсу групи ІМ-13
Нестеров Дмитро Васильович
номер у списку групи: 17

Перевірив:

Порєв Віктор Миколайович

Київ 2022

Мета:

Мета роботи – отримати вміння та навички використовувати засоби обміну інформацією та запрограмувати взаємодію незалежно працюючих програмних компонентів.

Завдання:

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab6.

2. Написати вихідний текст програми згідно варіанту завдання.

3. Скомпілювати вихідний текст і отримати виконуваний файл програми.

4. Перевірити роботу програми. Налагодити програму.

5. Проаналізувати та прокоментувати результати та вихідний текст програми.

6. Оформити звіт.

Вихідні тексти файлів:

Lab6.h

```
#pragma once

#include "resource.h"
```

Lab6.cpp

```
// Lab6.cpp : Defines the entry point for the application.
//

#include "framework.h"
#include "Lab6.h"
#include "matrix_dialog.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name

// Forward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

MatrixDialog& matrixDialog = matrixDialog.getInstance();

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR    lpCmdLine,
                      _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);
```

```
    // TODO: Place code here.

    // Initialize global strings
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LAB6, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_LAB6));

    MSG msg;

    // Main message loop:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
}



//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LAB6));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_LAB6);
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}

//
//   FUNCTION: InitInstance(HINSTANCE, int)
```

```
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // Store instance handle in our global variable

   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}

//
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE: Processes messages for the main window.
//
//  WM_COMMAND  - process the application menu
//  WM_PAINT    - Paint the main window
//  WM_DESTROY  - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_CREATE:
        SetWindowPos(hWnd, NULL, 40, 40, 500, 500, SWP_SHOWWINDOW);
        break;
    case WM_COMMAND:
        {
            int wmId = LOWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
            case IDM_MATRIX:
                matrixDialog.Start(hInst, hWnd);
                break;
            case IDM_ABOUT:
                DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                break;
            case IDM_EXIT:
                DestroyWindow(hWnd);
                break;
            default:
                return DefWindowProc(hWnd, message, wParam, lParam);
            }
```

```
        }
        break;
    case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);
            // TODO: Add any drawing code that uses hdc here...
            EndPaint(hWnd, &ps);
        }
        break;
    case WM_DESTROY:
        matrixDialog.End();
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}
```

Object2.cpp

```
// Object2.cpp : Defines the entry point for the application.
//

#include "framework.h"
#include "Object2.h"
#include "matrix_build.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name

// Forward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
```

```
MatrixBuild& matrixBuild = matrixBuild.getInstance();

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR    lpCmdLine,
                      _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Place code here.

    // Initialize global strings
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_OBJECT2, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_OBJECT2));

    MSG msg;

    // Main message loop:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
}



//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_OBJECT2));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
```

```
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_OBJECT2);
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}

//
//   FUNCTION: InitInstance(HINSTANCE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // Store instance handle in our global variable

   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}

//
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE: Processes messages for the main window.
//
//  WM_COMMAND  - process the application menu
//  WM_PAINT    - Paint the main window
//  WM_DESTROY  - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_COPYDATA:
        SetWindowPos(hWnd, NULL, 540, 40, 600, 500, SWP_SHOWWINDOW);
        matrixBuild.OnCopyData(hWnd, wParam, lParam);
        break;
    case WM_COMMAND:
        {
            int wmId = LOWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
            case IDM_ABOUT:
```

```
                DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                break;
            case IDM_EXIT:
                DestroyWindow(hWnd);
                break;
            default:
                return DefWindowProc(hWnd, message, wParam, lParam);
            }
        }
        break;
    case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);
            matrixBuild.OnPaint(hWnd, hdc);
            EndPaint(hWnd, &ps);
        }
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}
```

Object3.cpp

```
// Object3.cpp : Defines the entry point for the application.
//

#include "framework.h"
#include "Object3.h"
#include "matrix_det.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
```

```
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name

// Forward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

MatrixDet& matrixDet = matrixDet.getInstance();

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR    lpCmdLine,
                      _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Place code here.

    // Initialize global strings
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_OBJECT3, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_OBJECT3));

    MSG msg;

    // Main message loop:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
}



//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);
```

```
    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_OBJECT3));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_OBJECT3);
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}

//
//   FUNCTION: InitInstance(HINSTANCE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // Store instance handle in our global variable

   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}

//
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE: Processes messages for the main window.
//
//  WM_COMMAND  - process the application menu
//  WM_PAINT    - Paint the main window
//  WM_DESTROY  - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_CREATE:
    case WM_CLIPBOARDUPDATE:
        SetWindowPos(hWnd, NULL, 1140, 40, 400, 200, SWP_SHOWWINDOW);
```

```
            matrixDet.OnCreate(hWnd);
            break;
        case WM_COMMAND:
            {
                int wmId = LOWORD(wParam);
                // Parse the menu selections:
                switch (wmId)
                {
                case IDM_ABOUT:
                    DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                    break;
                case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam, lParam);
                }
            }
            break;
        case WM_PAINT:
            {
                PAINTSTRUCT ps;
                HDC hdc = BeginPaint(hWnd, &ps);
                matrixDet.OnPaint(hWnd, hdc);
                EndPaint(hWnd, &ps);
            }
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}
```

Matrix_det.cpp

```
#include "matrix_det.h"
```

```
void MatrixDet::OnCreate(HWND hWnd)
{
	GetFromClipboard(hWnd, 10009);
	matrix = GetMatrix(matrixText);
	det = GetDet(matrix, N);

	InvalidateRect(hWnd, NULL, TRUE);
}

int MatrixDet::GetFromClipboard(HWND hWnd, long maxsize)
{
	HGLOBAL hglb;
	LPTSTR lptstr;
	long size, res;
	res = 0;
	char* dest = new char[maxsize];
	if (!IsClipboardFormatAvailable(CF_TEXT)) return 0;
	if (!OpenClipboard(hWnd)) return 0;
	hglb = GetClipboardData(CF_TEXT);
	if (hglb != NULL)
	{
		lptstr = (LPTSTR)GlobalLock(hglb);
		if (lptstr != NULL)
		{
			size = (long)strlen((char*)lptstr);
			if (size > maxsize)
			{
				lptstr[maxsize] = 0;
				size = (long)strlen((char*)lptstr);
			}
			res = size;
			strcpy_s(dest, maxsize, (char*)lptstr);
			GlobalUnlock(hglb);
		}
	}
	CloseClipboard();
	matrixText = dest;

	return res;
}

int** MatrixDet::GetMatrix(std::string text)
{
	std::stringstream stream(text);

	int size = (int)std::count(text.cbegin(), text.cend(), '\n');
	N = size;
	int** result = new int* [size];
	for (int i = 0; i < size; i++)
		result[i] = new int[size];

	int element;
	int i = 0;
	int j = 0;

	for (int i = 0; i < size; i++)
	{
		for (int j = 0; j < size; j++)
		{
			stream >> element;
			result[i][j] = element;
```

```
			}
		}

		return result;
}

void MatrixDet::GetSubmatrix(int** src, int** dest, int N, int row, int col)
{
		int di = 0;
		int dj = 0;

		int size = N - 1;
		for (int i = 0; i < size; i++)
		{
			if (i == row) di = 1;

			dj = 0;
			for (int j = 0; j < size; j++)
			{
				if (j == col) dj = 1;
				dest[i][j] = src[i + di][j + dj];
			}
		}
}

int MatrixDet::GetDet(int** matrix, int size)
{
		int det = 0;
		int degree = 1;

		if (size == 1)
			return matrix[0][0];

		if (size == 2)
			return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];

		int** temp = new int* [size - 1];
		for (int i = 0; i < size - 1; i++)
			temp[i] = new int[size - 1];

		for (int j = 0; j < size; j++)
		{
			GetSubmatrix(matrix, temp, size, 0, j);
			det = det + (degree * matrix[0][j] * GetDet(temp, size - 1));
			degree = -degree;
		}

		for (int i = 0; i < size - 1; i++)
			delete[] temp[i];
		delete[] temp;

		return det;
}

void MatrixDet::OnPaint(HWND hWnd, HDC hdc)
{
		using convert_type = std::codecvt_utf8<wchar_t>;
		std::wstring_convert<convert_type, wchar_t> converter;

		std::string sDet;
		std::wstring wsDet;
```

```
        std::stringstream ss;
        ss << std::fixed << std::setprecision(0) << det;
        ss >> sDet;
        wsDet = converter.from_bytes(sDet);

        TextOut(hdc, 50, 50, (LPCWSTR)wsDet.c_str(),
(int)wcslen((LPCWSTR)wsDet.c_str()));
}
```

Matrix_dialog.**cpp**

```
#include "matrix_dialog.h"

INT_PTR CALLBACK Matrix(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK)
        {
            int N = 0;
            int min = 0;
            int max = 0;

            try
            {
                N = (int)round(stod(MatrixDialog::GetBoxText(hDlg, IDC_N)));
                min = (int)round(stod(MatrixDialog::GetBoxText(hDlg, IDC_MIN)));
                max = (int)round(stod(MatrixDialog::GetBoxText(hDlg, IDC_MAX)));
            }
            catch (...)
            {
                MessageBox(hDlg, L"Input numbers!!", L"Error", MB_OK | MB_ICONERROR);
                break;
            }

            if (min > max || N < 1 || N > 10 || abs(min) > 99999 || abs(max) > 99999)
            {
                MessageBox(hDlg, L"Invalid numbers", L"Error", MB_OK | MB_ICONERROR);
                break;
            }

            HWND hWnd2 = FindWindow(L"OBJECT2", NULL);
            if (!hWnd2)
            {
                WinExec("Object2.exe", SW_SHOW);
                hWnd2 = FindWindow(L"OBJECT2", NULL);
            }

            int data[3] = { N, min, max };
            MatrixDialog::SendData(hWnd2, GetParent(hDlg), data, sizeof(data));

            HWND hWnd3 = FindWindow(L"OBJECT3", NULL);
            if (!hWnd3)
            {
                WinExec("Object3.exe", SW_SHOW);
```

```
                hWnd3 = FindWindow(L"OBJECT3", NULL);
            }
            PostMessage(hWnd3, WM_CLIPBOARDUPDATE, NULL, NULL);

            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        if (LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}

void MatrixDialog::Start(HINSTANCE hInst, HWND hWnd)
{
    DialogBox(hInst, MAKEINTRESOURCE(IDD_MATRIX), hWnd, Matrix);
}

std::wstring MatrixDialog::GetBoxText(HWND hWnd, int boxID)
{
    WCHAR buff[7];
    GetWindowText(GetDlgItem(hWnd, boxID), buff, 7);

    return buff;
}

void MatrixDialog::SendData(HWND hWndDest, HWND hWndSrc, void* lp, long cb)
{
    COPYDATASTRUCT cds;

    cds.dwData = 1;
    cds.cbData = cb;
    cds.lpData = lp;

    SendMessage(hWndDest, WM_COPYDATA, (WPARAM)hWndSrc, (LPARAM)&cds);
}



void MatrixDialog::End()
{
    HWND hWnd2 = FindWindow(L"OBJECT2", NULL);
    HWND hWnd3 = FindWindow(L"OBJECT3", NULL);

    if (hWnd2) PostMessage(hWnd2, WM_DESTROY, NULL, NULL);
    if (hWnd3) PostMessage(hWnd3, WM_DESTROY, NULL, NULL);
}
```

matrix_build.cpp

```
#include "matrix_build.h"


void MatrixBuild::OnCopyData(HWND hWnd, WPARAM wParam, LPARAM lParam)
{

      COPYDATASTRUCT* cds;
```

```
	cds = (COPYDATASTRUCT*)lParam;
	int* data = (int*)cds->lpData;
	N = data[0];
	min = data[1];
	max = data[2];

	matrix = CreateMatrix(N, min, max);
	matrixText = GetMatrixString(matrix, N);

	PutToClipboard(hWnd, matrixText.c_str());

	InvalidateRect(hWnd, NULL, TRUE);
}

void MatrixBuild::OnPaint(HWND hWnd, HDC hdc)
{
	using convert_type = std::codecvt_utf8<wchar_t>;
	std::wstring_convert<convert_type, wchar_t> converter;

	std::string sElem;
	std::wstring wsElem;
	double elem;

	int x = 10;
	int y = 10;
	for (int i = 0; i < N; i++)
	{
		for (int j = 0; j < N; j++)
		{
			std::stringstream ss;
			elem = matrix[i][j];
			ss << elem;
			ss >> sElem;
			wsElem = converter.from_bytes(sElem);
			TextOut(hdc, x, y, (LPCWSTR)wsElem.c_str(),
(int)wcslen((LPCWSTR)wsElem.c_str()));

			x += 50;
		}
		x = 10;
		y += 35;
	}
}

int** MatrixBuild::CreateMatrix(int size, int minimum, int maximum)
{
	int** result = new int* [size];
	for (int i = 0; i < size; i++)
		result[i] = new int[size];

	std::random_device rd;
	std::mt19937 gen(rd());
	std::uniform_int_distribution<> dist(minimum, maximum);

	for (int i = 0; i < size; i++)
		for (int j = 0; j < size; j++)
			result[i][j] = dist(gen);

	return result;
}
```

```
std::string MatrixBuild::GetMatrixString(int** matrixSrc, int size)
{
	using convert_type = std::codecvt_utf8<wchar_t>;
	std::wstring_convert<convert_type, wchar_t> converter;

	std::string result;
	std::ostringstream stream;
	int element;

	for (int i = 0; i < size; i++)
	{
		for (int j = 0; j < size; j++)
		{
			element = matrixSrc[i][j];
			stream << element << '\t';
		}
		stream << '\n';
	}
	result = stream.str();

	return result;
}

int MatrixBuild::PutToClipboard(HWND hWnd, const char* src)
{
	HGLOBAL hglbCopy;
	BYTE* pTmp;
	long len;

	if (src == NULL) return 0;
	if (src[0] == 0) return 0;

	len = (long)strlen(src);
	hglbCopy = GlobalAlloc(GHND, len + 1);

	if (hglbCopy == NULL) return 0;

	pTmp = (BYTE*)GlobalLock(hglbCopy);
	memcpy(pTmp, src, len + 1);
	GlobalUnlock(hglbCopy);

	if (!OpenClipboard(hWnd))
	{
		GlobalFree(hglbCopy);
		return 0;
	}

	EmptyClipboard();
	SetClipboardData(CF_TEXT, hglbCopy);
	CloseClipboard();

	return 1;
}
```

matrix_det.h

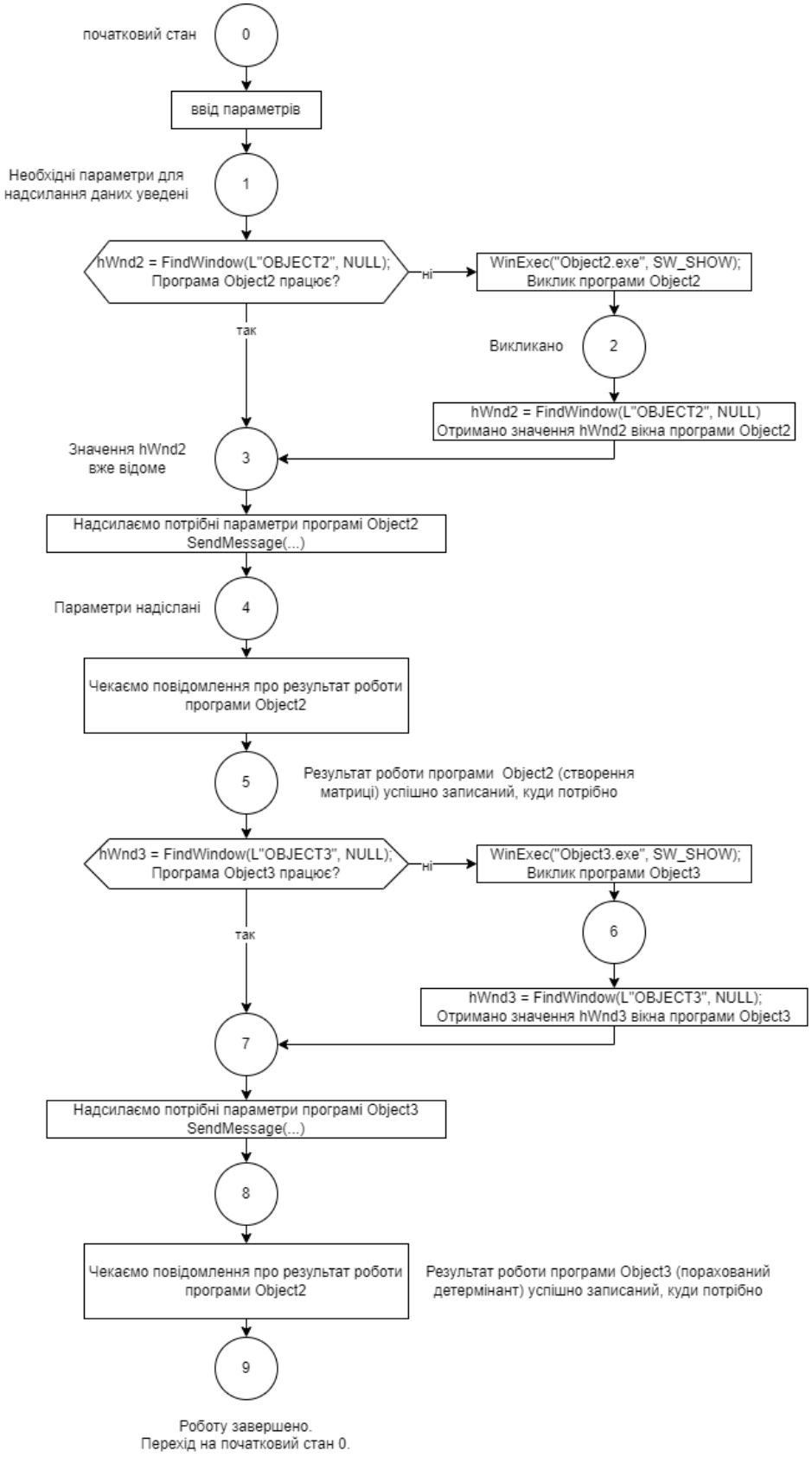
```
#pragma once
#include "framework.h"
#include <string>
#include <sstream>
```

```
#include <codecvt>
#include <iomanip>

class MatrixDet
{
private:
	MatrixDet() {}
	MatrixDet(const MatrixDet& root) = delete;
	MatrixDet& operator = (const MatrixDet&) = delete;

	double det = 0;
	int N = 0;
	std::string matrixText = "";
	int** matrix = 0;
public:
	static MatrixDet& getInstance()
	{
		static MatrixDet instance;
		return instance;
	}

	void OnCreate(HWND);

	int GetFromClipboard(HWND, long);
	int** GetMatrix(std::string);
	void GetSubmatrix(int**, int**, int, int, int);
	int GetDet(int**, int);

	void OnPaint(HWND, HDC);
};
```

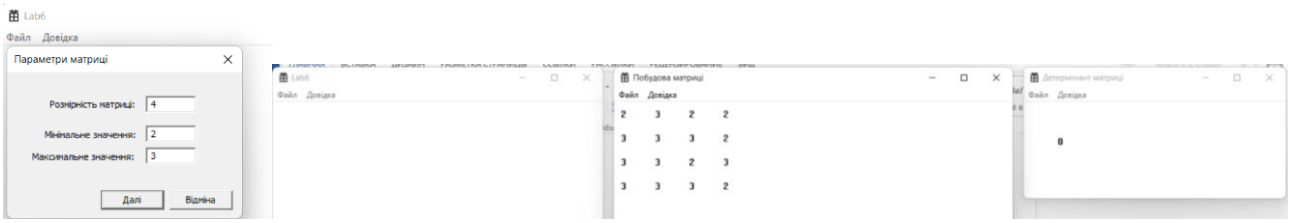
matrix_dialog.h

```
#pragma once
#include "framework.h"
#include "resource1.h"
#include <string>
#include <cmath>

class MatrixDialog
{
private:
	MatrixDialog() {}
	MatrixDialog(const MatrixDialog& root) = delete;
	MatrixDialog& operator = (const MatrixDialog&) = delete;
public:
	static MatrixDialog& getInstance()
	{
		static MatrixDialog instance;
		return instance;
	}

	static std::wstring GetBoxText(HWND, int);
	static void SendData(HWND, HWND, void*, long);

	void Start(HINSTANCE, HWND);
	void End();
};
```

Схема послідовності надсилання-обробки повідомлень



Скріншоти



Висновки. Під час виконання лабораторної роботи №6 я отримав вміння та навички використання засобів обміну інформації між незалежно працюючими програмними компонентами.