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:
«Об’єктно-орієнтоване програмування»

Виконав:
Баран Павло Юрійович
студент групи ІП-03
Номер у списку групи: 1

Перевірив:
Порєв В. М.

Київ 2021

Варіант:

1	1. Користувач вводить значення **n, Min, Max** у діалоговому вікні. 2. Програма викликає програми Object2, 3 і виконує обмін повідомленнями з ними для передавання, отримання інформації.	1. Створює матрицю **n×n** цілих (**int**) чисел у діапазоні **Min – Max** 2. Показує числові значення у власному головному вікні 3. Записує дані в Clipboard Windows у текстовому форматі	1. Зчитує дані з Clipboard Windows 2. Відображає значення детермінанту матриці у власному головному вікні

Програмний код:
Lab6.cpp

```
#include "framework.h"
#include "pch.h"
#include "Lab6.h"
#include "Module1.h"

#define MAX_LOADSTRING 100

HINSTANCE hInst;
WCHAR szTitle[MAX_LOADSTRING];
WCHAR szWindowClass[MAX_LOADSTRING];

ATOM            MyRegisterClass(HINSTANCE hInstance);
BOOL            InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
static void CallInputValues(HWND hWnd);
HWND hWndObj2;
HWND hWndObj3;

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

    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LAB6, szWindowClass, MAX_LOADSTRING);
```

```
    MyRegisterClass(hInstance);

    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

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

    MSG msg;

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

    return (int)msg.wParam;
}

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);
}
```

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance;

   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW |
WS_CLIPCHILDREN,
      200, 350, 300, 300, nullptr, nullptr, hInstance, nullptr);

   if (!hWnd)
   {
      return FALSE;
   }
   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
   return TRUE;
}

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;
}


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
   switch (message)
   {
   case WM_COMMAND:
   {
```

```
        int wmId = LOWORD(wParam);
        switch (wmId)
        {
        case IDM_WORK:
            CallInputValues(hWnd);
            break;
        case IDM_ABOUT:
            DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
            break;
        case IDM_EXIT:
            DestroyWindow(hWnd);
            break;
        default:
            return DefWindowProcW(hWnd, message, wParam, lParam);
        }
    }
    break;
    case WM_DESTROY:
        PostQuitMessage(0);

        hWndObj2 = FindWindow("OBJECT2", NULL);
        if (hWndObj2)
        {
            PostMessage(hWndObj2, WM_DESTROY, (WPARAM)wParam, 0);
        }

        hWndObj3 = FindWindow("OBJECT3", NULL);
        if (hWndObj3)
        {
            PostMessage(hWndObj3, WM_DESTROY, (WPARAM)wParam, 0);
        }

        break;
    default:
        return DefWindowProcW(hWnd, message, wParam, lParam);
    }
    return 0;
}

void CallInputValues(HWND hWnd)
{
    Func_MOD1(hInst, hWnd);
    InvalidateRect(hWnd, 0, TRUE);
}
```

Module1.cpp

```
#include "pch.h"
#include "framework.h"
#include "Module1.h"

HINSTANCE hInstCurrent;

long n_MOD1;
long Min_MOD1;
long Max_MOD1;

HWND hWndDataCreator = NULL;

static INT_PTR CALLBACK InputValues_MOD1(HWND hDlg, UINT iMessage, WPARAM
wParam, LPARAM lParam);
static INT_PTR CALLBACK Warning_MOD1(HWND hDlg, UINT iMessage, WPARAM
wParam, LPARAM lParam);
static void OnOk(HWND hDlg);
static void OnCancel(HWND hDlg);
static void OnClose(HWND hDlg);
int SendCopyData(HWND hWndDest, HWND hWndSrc, void* lp, long cb);

int Func_MOD1(HINSTANCE hInst, HWND hWnd)
{
   return DialogBox(hInst, MAKEINTRESOURCE(IDD_INPUT), hWnd, InputValues_MOD1);
}

INT_PTR CALLBACK InputValues_MOD1(HWND hDlg, UINT iMessage, WPARAM wParam,
LPARAM lParam)
{
   switch (iMessage)
   {
   case WM_INITDIALOG:
      return (INT_PTR)TRUE;
      break;

   case WM_COMMAND:
      switch (LOWORD(wParam))
      {
      case IDOK:
         OnOk(hDlg);
         return (INT_PTR)TRUE;
         break;
      case IDCANCEL:
         OnCancel(hDlg);
         return (INT_PTR)TRUE;
         break;
      }
      break;
   case WM_CLOSE:
```

```
    {
        OnClose(hDlg);
    }
    break;
    default: break;
    }
    return FALSE;
}

void OnOk(HWND hDlg)
{
    n_MOD1 = GetDlgItemInt(hDlg, IDC_EDIT_N, NULL, FALSE);
    Min_MOD1 = GetDlgItemInt(hDlg, IDC_EDIT_MIN, NULL, FALSE);
    Max_MOD1 = GetDlgItemInt(hDlg, IDC_EDIT_MAX, NULL, FALSE);

    if (n_MOD1 == NULL  || Min_MOD1 == NULL || Max_MOD1 == NULL )
    {
        DialogBox(hInstCurrent, MAKEINTRESOURCE(IDD_WARNING_NULL), hDlg,
Warning_MOD1);
        return;
    }
    if (Max_MOD1 > 10)
    {
        DialogBox(hInstCurrent, MAKEINTRESOURCE(IDD_WARNING_MAX), hDlg,
Warning_MOD1);
        return;
    }
    if (Min_MOD1 <= Max_MOD1)
    {
        hWndDataCreator = FindWindow("OBJECT2", NULL);
        if (hWndDataCreator == NULL)
        {
            WinExec("Object2.exe", SW_SHOW);
            hWndDataCreator = FindWindow("OBJECT2", NULL);
        }
        else {
            InvalidateRect(hWndDataCreator, 0, TRUE);
        }
        long params[3] = { n_MOD1, Min_MOD1, Max_MOD1 };

        SendCopyData(hWndDataCreator, GetParent(hDlg), params, sizeof(params));
        return;
    }
    else
    {
        DialogBox(hInstCurrent, MAKEINTRESOURCE(IDD_WARNING_VALUES),
            hDlg, Warning_MOD1);
        return;
    }
}

INT_PTR CALLBACK Warning_MOD1(HWND hDlg, UINT iMessage, WPARAM wParam,
LPARAM lParam)
```

```
{
   switch (iMessage)
   {
   case WM_INITDIALOG:
      return (INT_PTR)TRUE;
      break;
   case WM_COMMAND:
      switch (LOWORD(wParam))
      {
      case IDOK:
         EndDialog(hDlg, 0);
         return (INT_PTR)TRUE;
         break;
      }
      break;
   case WM_CLOSE:
   {
      OnClose(hDlg);
   }
   break;
   default: break;
   }
   return FALSE;
}

void OnCancel(HWND hDlg)
{
   EndDialog(hDlg, 0);
}

void OnClose(HWND hDlg)
{
   EndDialog(hDlg, 0);
}

int SendCopyData(HWND hWndDest, HWND hWndSrc, void* lp, long cb)
{
   COPYDATASTRUCT cds{};
   cds.dwData = 1;
   cds.cbData = cb;
   cds.lpData = lp;
   return SendMessage(hWndDest, WM_COPYDATA, (WPARAM)hWndSrc,
(LPARAM)&cds);
}
```

Object2.cpp

```
#include "framework.h"
#include "pch.h"
#include "Object2.h"
```

```
#include <random>
#include "Resource.h"
#include <iostream>
#include <time.h>
using namespace std;

#define MAX_LOADSTRING 100

HINSTANCE hInst;
WCHAR szTitle[MAX_LOADSTRING];
WCHAR szWindowClass[MAX_LOADSTRING];

ATOM             MyRegisterClass(HINSTANCE hInstance);
BOOL             InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
int RandomInt(int low, int high);
static int Count(int element);
void OnCopyData(HWND hWnd, WPARAM wParam, LPARAM lParam);
int PutTextToClipboard(HWND hWnd, char* src);
void StartObj3(HWND hWnd);
void CreateMatrix(HWND hWnd);
int SendCopyData(HWND hWndDest, HWND hWndSrc, void* lp, long cb);

int values_MOD2[3];
HWND hWndDataCreator = NULL;

int n_MOD2;
int Min_MOD2;
int Max_MOD2;

BOOL Counter = FALSE;
std::string copyMatrix = "";

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

   LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
   LoadStringW(hInstance, IDC_OBJECT2, szWindowClass, MAX_LOADSTRING);
   MyRegisterClass(hInstance);

   if (!InitInstance(hInstance, nCmdShow))
   {
      return FALSE;
   }

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

```
    MSG msg;
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    return (int)msg.wParam;
}

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(IDC_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);
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hInst = hInstance;

    HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW |
WS_CLIPCHILDREN,
        150, 100, 200, 200, nullptr, nullptr, hInstance, nullptr);

    if (!hWnd)
    {
        return FALSE;
    }

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

    return TRUE;
}

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;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
    switch (message)
    {
    case WM_CREATE:
    {
        srand(time(0));
        SetWindowPos(hWnd, HWND_BOTTOM, 150, 100, 200, 200, SWP_DEFERERASE);
    }
    break;
    case WM_COPYDATA:
    {
        OnCopyData(hWnd, wParam, lParam);

        if (n_MOD2 > 0)
        {
            CreateMatrix(hWnd);
        }
        InvalidateRect(hWnd, 0, TRUE);
    }
        break;
    case WM_COMMAND:
    {
        int wmId = LOWORD(wParam);
        switch (wmId)
        {
        case IDM_ABOUT:
            DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
            break;
        case IDM_EXIT:
            DestroyWindow(hWnd);
            break;
        default:
            return DefWindowProcW(hWnd, message, wParam, lParam);
        }
```

```
    }
    break;

    case WM_PAINT:
    {
        RECT rc = { 0 };
        GetClientRect(hWnd, &rc);
        PAINTSTRUCT ps;
        HDC hdc = BeginPaint(hWnd, &ps);

        char* cstr = new char[copyMatrix.size() + 1];
        strcpy_s(cstr, copyMatrix.size() + 1, copyMatrix.c_str());
        PutTextToClipboard(hWnd, cstr);

        DrawTextA(hdc, cstr, -1, &rc, DT_TOP);
        EndPaint(hWnd, &ps);
    }
    break;

    case WM_DESTROY:
    {
        PostQuitMessage(0);
    }
    break;

    default:
        return DefWindowProcW(hWnd, message, wParam, lParam);
    }
    return 0;
}

void CreateMatrix(HWND hWnd)
{
    int** matrix = new int* [n_MOD2];
    for (int i = 0; i < n_MOD2; ++i)
    {
        matrix[i] = new int[n_MOD2];
    }
    for (int i = 0; i < n_MOD2; ++i)
    {
        for (int j = 0; j < n_MOD2; ++j)
        {
            matrix[i][j] = RandomInt(Min_MOD2, Max_MOD2);

            copyMatrix += to_string(matrix[i][j]);
            if (j < n_MOD2-1)
            {
                copyMatrix += " ";
            }
        }
        if (i < n_MOD2)
        {
            copyMatrix += "\n";
```

```
        }
    }
    for (int i = 0; i < n_MOD2; ++i)
    {
        delete[] matrix[i];
    }
    delete[] matrix;
}

int SendCopyData(HWND hWndDest, HWND hWndSrc, void* lp, long cb)
{
    COPYDATASTRUCT cds{};
    cds.dwData = 1;
    cds.cbData = cb;
    cds.lpData = lp;
    return SendMessage(hWndDest, WM_COPYDATA, (WPARAM)hWndSrc,
(LPARAM)&cds);
}

int RandomInt(int low, int high)
{
    return rand() % high + low;
}

int Count(int element)
{
    int count_MOD1 = 0;
    while (element != 0)
    {
        element = element / 10;
        ++count_MOD1;
    }
    return count_MOD1;
}

void OnCopyData(HWND hWnd, WPARAM wParam, LPARAM lParam)
{
    COPYDATASTRUCT* cds;
    cds = (COPYDATASTRUCT*)lParam;
    long* p = (long*)cds->lpData;
    n_MOD2 = p[0];
    Min_MOD2 = p[1];
    Max_MOD2 = p[2];
}

int PutTextToClipboard(HWND hWnd, char* src)
{
    HGLOBAL hglbCopy;
    BYTE* pTmp;
    long len;
    if (src == NULL) return 0;
    if (src[0] == 0) return 0;
    len = strlen(src);
```

```
    hglbCopy = GlobalAlloc(GHND, len + 1);
    if (hglbCopy == NULL) return FALSE;
    pTmp = (BYTE*)GlobalLock(hglbCopy);
    memcpy(pTmp, src, len + 1);
    GlobalUnlock(hglbCopy);
    if (!OpenClipboard(hWnd))
    {
        GlobalFree(hglbCopy);
        return 0;
    }
    EmptyClipboard();
    SetClipboardData(CF_TEXT, hglbCopy);
    CloseClipboard();

    StartObj3(hWnd);
    return 1;
}

void StartObj3(HWND hWnd)
{
    hWndDataCreator = FindWindow("OBJECT3", NULL);
    if (hWndDataCreator == NULL)
    {
        WinExec("Object3.exe", SW_SHOW);
        hWndDataCreator = FindWindow("OBJECT3", NULL);
    }
    else {
        InvalidateRect(hWndDataCreator, 0, TRUE);
    }

    long params[3] = { n_MOD2};
    SendCopyData(hWndDataCreator, hWnd, params, sizeof(params));
}
```

Object3.cpp

```
#include "framework.h"
#include "pch.h"
#include "Object3.h"
#include "Resource.h"
#include <string>
using namespace std;

#define MAX_LOADSTRING 100

HINSTANCE hInst;
WCHAR szTitle[MAX_LOADSTRING];
WCHAR szWindowClass[MAX_LOADSTRING];
char bufferText[2048];
int n_MOD3;
int* buffer;
```

```
int determinant;

ATOM             MyRegisterClass(HINSTANCE hInstance);
BOOL             InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
long GetTextFromClipboard(HWND, char*, long);
void OnCopyData(HWND hWnd, WPARAM wParam, LPARAM lParam);
int CalculateDeterminant(int * matrix, int rows);

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

    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_OBJECT3, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);
    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_OBJECT3));
    MSG msg;
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    return (int)msg.wParam;
}


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(IDC_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);
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hInst = hInstance;

    HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW |
WS_CLIPCHILDREN,
        400, 120, 200, 200, nullptr, nullptr, hInstance, nullptr);

    if (!hWnd)
    {
        return FALSE;
    }
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}

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;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
    switch (message)
    {
    case WM_CREATE:
    {
```

```
    SetWindowPos(hWnd, HWND_BOTTOM, 400, 120, 200, 200, SWP_DEFERERASE);
    GetTextFromClipboard(hWnd, bufferText, sizeof(bufferText));
}
    break;
case WM_COPYDATA:
{
    OnCopyData(hWnd, wParam, lParam);
    InvalidateRect(hWnd, 0, TRUE);
}
break;
case WM_COMMAND:
{
    int wmId = LOWORD(wParam);
    switch (wmId)
    {
    case IDM_ABOUT:
        DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
        break;
    case IDM_EXIT:
        DestroyWindow(hWnd);
        break;
    default:
        return DefWindowProcW(hWnd, message, wParam, lParam);
    }
}
break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
case WM_PAINT:
{
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);
    string tempBufferForMatrixString;

    if (bufferText != tempBufferForMatrixString) {
        buffer = new int[n_MOD3 * n_MOD3];

        tempBufferForMatrixString = bufferText;
        string num;
        int tempCounter = 0;
        while (tempCounter != n_MOD3 * n_MOD3)
        {
            num = tempBufferForMatrixString[tempCounter * 2];

            buffer[tempCounter] = stod(num);
            tempCounter++;
        }

        int determinant = CalculateDeterminant(buffer, n_MOD3);
        CHAR buf[100];
        sprintf_s(buf, ("%d"), determinant);
```

```
            TextOutA(hdc, 0, 0, buf, 10);
        }
        EndPaint(hWnd, &ps);
    }

        break;
    default:
        return DefWindowProcW(hWnd, message, wParam, lParam);
    }
    return 0;
}

int CalculateDeterminant(int * matrix, int rows) {
    int determinant = 0;
    if (rows == 1) {
        determinant = matrix[0];
    }
    else if (rows == 2) {
        determinant = matrix[0] * matrix[3] - matrix[1] * matrix[2];
    }
    else {
        for (int i = 0; i < rows; i++) {
            int nextSize = rows - 1;
            int d;
            int* newMatrix = new int[nextSize * nextSize];
            int divine = (i % 2 == 0) ? 1 : -1;
            int counter = 0;
            for (int j = rows; j < rows * rows; j++) {
                if ((j) % rows == i) continue;
                newMatrix[counter] = matrix[j];
                counter++;
            }
            d = CalculateDeterminant(newMatrix, nextSize);
            determinant += divine * d * matrix[i];
            delete[] newMatrix;
        }
    }
    return determinant;
}

void OnCopyData(HWND hWnd, WPARAM wParam, LPARAM lParam)
{
    COPYDATASTRUCT* cds;
    cds = (COPYDATASTRUCT*)lParam;
    long* p = (long*)cds->lpData;
    n_MOD3 = p[0];
}

long GetTextFromClipboard(HWND hWnd, char* dest, long maxsize)
{
    HGLOBAL hglb;
    LPTSTR lptstr;
    long size, res;
```

```
    res = 0;
    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 = strlen((char*)lptstr);
            if (size > maxsize)
            {
                lptstr[maxsize] = 0;
                size = strlen((char*)lptstr);
            }
            strcpy_s(dest, maxsize, (char*)lptstr);
            res = size;
            GlobalUnlock(hglb);
        }
    }
    CloseClipboard();
    return res;
}
```

Результати тестування програми:

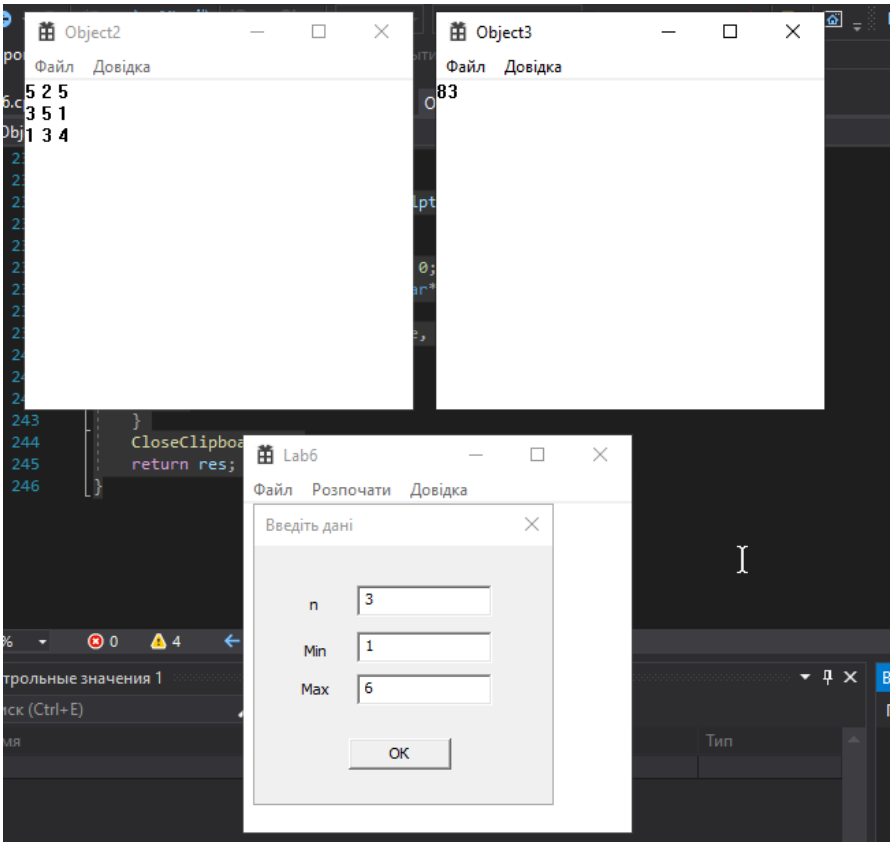
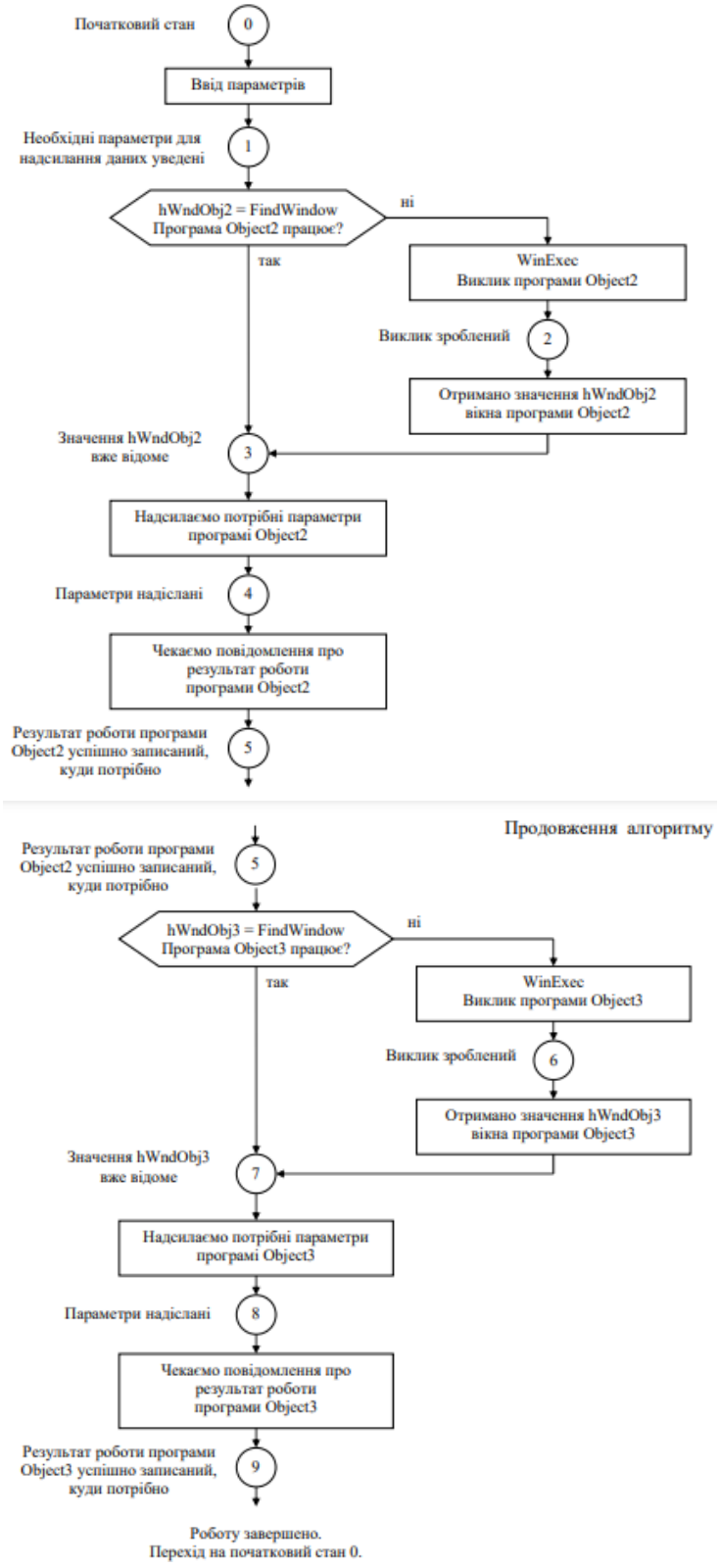


Схема послідовності надсилання-обробки повідомлень:



Висновки:

Виконуючи шосту лабораторну роботу я навчився передавати повідомлення між незалежними модулями програми та познайомився з роботою ClipBoard на C++.