

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

«Побудування програмної системи з множини об'єктів, керованих
повідомленнями»

Виконав:

студент групи ІП-93

Домінський Валентин Олексійович

номер залікової книжки: 9311

Номер у списку: 9

Перевірив:

Порєв Віктор Миколайович

Київ 2020

Мета:

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 вихідні тексти усіх програм-компонентів згідно варіанту завдання.
3. Скомпілювати вихідні тексти і отримати виконувані файли програм.
4. Перевірити роботу програм. Налаштувати взаємодію програм.
5. Проаналізувати та прокоментувати результати та вихідні тексти програм.
6. Оформити звіт.

Варіант завдання

$9 \bmod 4 = 1$;

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.
2. 1. Створює матрицю $n \times n$ цілих (int) чисел у діапазоні $Min - Max$. 2. Показує числові значення у власному головному вікні. 3. Записує дані в Clipboard Windows у текстовому форматі.
3. 1. Зчитує дані з Clipboard Windows. 2. Відображає значення детермінанту матриці у власному головному вікні.

Вихідні тексти файлів:

Lab6.cpp:

```
// Lab6.cpp : Defines the input point for the application.
//
// First Part

#include "framework.h"
#include "pch.h"
#include "Lab6.h"
#include "InputValuesModule.h"

#define MAX_LOADSTRING 100

#pragma region VariablesAndFunctions

// Global variables:
HINSTANCE hInst;           // Current instance
WCHAR szTitle[MAX_LOADSTRING]; // Header row text
WCHAR szWindowClass[MAX_LOADSTRING]; // Class name of main window

// Send 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);
static void CallInputValues(HWND hWnd); // Declaration of our function
HWND hWndObj2;
HWND hWndObj3;

#pragma endregion VariablesAndFunctions

#pragma region DefaultFunctions

// Second Part
// Enter Point "wWinMain"
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
    _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPWSTR lpCmdLine,
    _In_int_ nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Place the code here.

    // Global line initialization
    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 cycle:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
```

```

    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}

return (int)msg.wParam;
}

//
// FUNCTION: MyRegisterClass()
//
// OBJECTIVE: To register the window class.
// Text of Function
/// <summary>
/// Register the window class.
/// </summary>
/// <param name="hInstance">The h instance.</param>
/// <returns></returns>
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)
//
// OBJECTIVE: Saves the instance marker and creates the main window
//
// COMMENTARIES:
//
// In this function, the instance marker is saved in a global variable, and also
// the main program window is created and displayed.
//
/// <summary>
/// Saves the instance marker and creates the main window
/// </summary>
/// <param name="hInstance">The h instance.</param>
/// <param name="nCmdShow">The n command show.</param>
/// <returns></returns>
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hInst = hInstance; // Save instance marker in global variable

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

    if (!hWnd)
    {
        return FALSE;
    }
}

```

```

}

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

return TRUE;
}

/// <summary>
/// Message handler for "About" window.
/// </summary>
/// <param name="hDlg">The h dialog.</param>
/// <param name="message">The message.</param>
/// <param name="wParam">The w parameter.</param>
/// <param name="lParam">The l parameter.</param>
/// <returns></returns>
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;
}

#pragma endregion

#pragma region ModifiedFuntions

// Third Part
// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
// OBJECTIVE: Processes messages in the main window.
//
// WM_COMMAND - Process the application menu
// WM_PAINT - Drawing of the main window
// WM_DESTROY - Send message about exit and return
//
//
/// <summary>
/// Processes messages in the main window.
/// </summary>
/// <param name="hWnd">The h WND.</param>
/// <param name="message">The message.</param>
/// <param name="wParam">The w parameter.</param>
/// <param name="lParam">The l parameter.</param>
/// <returns></returns>
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;
}

/// <summary>
/// Function-handler of the menu item "Work1"
/// </summary>
/// <param name="hWnd">The h WND.</param>
void CallInputValues(HWND hWnd)
{
    // What we program here that will be done
    Func_MOD1(hInst, hWnd);

    // The update region represents the portion of the window's
    // client area that must be redrawn.
    InvalidateRect(hWnd, 0, TRUE);
}

```

```
#pragma endregion ModifiedFuntions
```

InputValuesModule.cpp:

```

#include "pch.h"
#include "framework.h"
#include "InputValuesModule.h"

#pragma region VariablesAndFunctionsDeclarations

HINSTANCE hInstCurrent;

long n_MOD1;
long Min_MOD1;
long Max_MOD1;

const int allValues = 3;

```

```
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);
```

```
#pragma endregion
```

```
#pragma region Functions
```

```
/// <summary>  
/// dialog box creation function  
/// </summary>  
/// <param name="hInst">The hInst.</param>  
/// <param name="hWnd">The hWnd.</param>  
/// <returns></returns>
```

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

```
/// <summary>  
/// Callback-function for calling window with inputs  
/// </summary>  
/// <param name="hDlg"></param>  
/// <param name="iMessage"></param>  
/// <param name="wParam"></param>  
/// <param name="lParam"></param>  
/// <returns></returns>
```

```
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;  
}
```

```
/// <summary>  
/// Called when IDOK clicked
```

```

/// </summary>
/// <param name="hDlg">The dialog.</param>
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 )
    {
        // call "enter a values" window
        DialogBox(hInstCurrent, MAKEINTRESOURCE(IDD_WARNING_NULL), hDlg, Warning_MOD1);
        return;
    }

    // check if min is less or equals to max
    if (Min_MOD1 <= Max_MOD1)
    {
        // call two object2 and object3 windows
        hWndDataCreator = FindWindow("OBJECT2", NULL);
        if (hWndDataCreator == NULL) // the required program is already running
        {
            // call to run the desired program
            WinExec("Object2.exe", SW_SHOW);
            hWndDataCreator = FindWindow("OBJECT2", NULL);
        }

        // form the data as a solid array, for example:
        long params[allValues] = { n_MOD1, Min_MOD1, Max_MOD1 };

        // and now send an array of params to the hWndOther window of another program
        SendCopyData(hWndDataCreator, GetParent(hDlg), params, sizeof(params));

        return;
    }
    else
    {
        DialogBox(hInstCurrent, MAKEINTRESOURCE(IDD_WARNING_VALUES),
            hDlg, Warning_MOD1);
        return;
    }
}

/// <summary>
/// Callback-function for calling window with inputs
/// </summary>
/// <param name="hDlg"></param>
/// <param name="iMessage"></param>
/// <param name="wParam"></param>
/// <param name="lParam"></param>
/// <returns></returns>
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;
}

/// <summary>
/// Called when IDCANCEL clicked
/// </summary>
/// <param name="hDlg">The dialog.</param>
void OnCancel(HWND hDlg)
{
    EndDialog(hDlg, 0);
}

/// <summary>
/// Called when window is closing
/// </summary>
/// <param name="hDlg">The dialog.</param>
void OnClose(HWND hDlg)
{
    EndDialog(hDlg, 0);
}

/// <summary>
/// Sends copydata
/// </summary>
/// <param name="hWndDest"></param>
/// <param name="hWndSrc"></param>
/// <param name="lp"></param>
/// <param name="cb"></param>
/// <returns></returns>
int SendCopyData(HWND hWndDest, HWND hWndSrc, void* lp, long cb)
{
    COPYDATASTRUCT cds{};
    cds.dwData = 1; // and any other value is possible
    cds.cbData = cb;
    cds.lpData = lp;
    return SendMessage(hWndDest, WM_COPYDATA, (WPARAM)hWndSrc, (LPARAM)&cds);
}

```

```
#pragma endregion
```

InputValuesModule.h:

```
#pragma once
```

```
#pragma once
#include "resource1.h"
```

```
#pragma region Function
```

```
extern int Func_MOD1(HINSTANCE hInst, HWND hWnd);
```

```
#pragma endregion
```

Object2.cpp:

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

```
//
```

```
// First Part
```

```
#include "framework.h"

#include "pch.h"

#include "Object2.h"

#include <vector>

#include <random>

#include "Resource.h"

#include <sstream>

#include <iostream>

using namespace std;


#define MAX_LOADSTRING 100


#pragma region VariablesAndFunctions


// Global variables:

HINSTANCE hInst;           // Current instance

WCHAR szTitle[MAX_LOADSTRING]; // Header row text

WCHAR szWindowClass[MAX_LOADSTRING]; // Class name of main window


// Send 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);

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 const allValues = 3;

int values_MOD2[allValues];

HWND hWndDataCreator = NULL;
```

```
int n_MOD2;
```

```
int Min_MOD2;
```

```
int Max_MOD2;
```

```
BOOL Counter = FALSE;
```

```
std::string copyMatrix = "";
```

```
#pragma endregion VariablesAndFunctions
```

```
#pragma region DefaultFunctions
```

```
// Second Part
```

```
// Enter Point "wWinMain"
```

```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
```

```
    _In_opt_ HINSTANCE hPrevInstance,
```

```
    _In_ LPWSTR lpCmdLine,
```

```
    _In_ int nCmdShow)
```

```
{
```

```
    UNREFERENCED_PARAMETER(hPrevInstance);
```

```
    UNREFERENCED_PARAMETER(lpCmdLine);
```

```
// TODO: Place the code here.
```

```
// Global line initialization
```

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

//Compiler version g++ 6.3.0

return (int)msg.wParam;
}

//
// FUNCTION: MyRegisterClass()
//
// OBJECTIVE: To register the window class.
// Text of Function
/// <summary>
/// Register the window class.
/// </summary>
/// <param name="hInstance">The h instance.</param>
/// <returns></returns>
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style = CS_HREDRAW | CS_VREDRAW;

```

```

wcex.lpfWndProc = 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);
}


//
// FUNCTION: InitInstance(HINSTANCE, int)
//
// OBJECTIVE: Saves the instance marker and creates the main window
//
// COMMENTARIES:
//
// In this function, the instance marker is saved in a global variable, and also
// the main program window is created and displayed.
//
/// <summary>
/// Saves the instance marker and creates the main window
/// </summary>
/// <param name="hInstance">The h instance.</param>
/// <param name="nCmdShow">The n command show.</param>
/// <returns></returns>
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hInst = hInstance; // Save instance marker in global variable


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

```

```

    if (!hWnd)
    {
        return FALSE;
    }

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

    return TRUE;
}

/// <summary>
/// Message handler for "About" window.
/// </summary>
/// <param name="hDlg">The h dialog.</param>
/// <param name="message">The message.</param>
/// <param name="wParam">The w parameter.</param>
/// <param name="lParam">The l parameter.</param>
/// <returns></returns>
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;
}

#pragma endregion

#pragma region ModifiedFuntions

// Third Part
// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
// OBJECTIVE: Processes messages in the main window.
//
// WM_COMMAND - Process the application menu
// WM_PAINT - Drawing of the main window
// WM_DESTROY - Send message about exit and return
//
//
/// <summary>
/// Processes messages in the main window.
/// </summary>
/// <param name="hWnd">The h WND.</param>
/// <param name="message">The message.</param>
/// <param name="wParam">The w parameter.</param>
/// <param name="lParam">The l parameter.</param>
/// <returns></returns>
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_CREATE:
        {
            SetWindowPos(hWnd, HWND_TOPMOST, 610, 190, 200, 200, SWP_DEFERERERASE);
        }
        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* tempStr = new char[copyMatrix.size() + 1];
```

```
    strcpy_s(tempStr, copyMatrix.size() + 1, copyMatrix.c_str());
```



```

        PutTextToClipboard(hWnd, tempStr);

        DrawTextA(hdc, tempStr, -1, &rc, DT_TOP);

        EndPaint(hWnd, &ps);
    }
    break;

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

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

/// <summary>
/// Creates matrix
/// </summary>
/// <param name="hWnd"></param>
/// <returns></returns>
void CreateMatrix(HWND hWnd)
{
    copyMatrix = "";

    // dynamic allocation
    int** matrix = new int* [n_MOD2];
    for (int i = 0; i < n_MOD2; ++i)
    {
        matrix[i] = new int[n_MOD2];
    }
}

```

```

// fill
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 += std::to_string(matrix[i][j]);

        if (j < n_MOD2-1)
        {
            copyMatrix += " ";
        }
    }

    if (i < n_MOD2)
    {
        copyMatrix += "\n";
    }
}

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

delete[] matrix;

StartObj3(hWnd);
}

```

```

/// <summary>
/// Sends copydata
/// </summary>
/// <param name="hWndDest"></param>
/// <param name="hWndSrc"></param>
/// <param name="lp"></param>

```

```

/// <param name="cb"></param>
/// <returns></returns>
int SendCopyData(HWND hWndDest, HWND hWndSrc, void* lp, long cb)
{
    COPYDATASTRUCT cds{};

    cds.dwData = 1; // or you can have any other value

    cds.cbData = cb;

    cds.lpData = lp;

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

```

```

/// <summary>
/// Creates random number for matrix
/// </summary>
/// <param name="low"></param>
/// <param name="high"></param>
/// <returns></returns>
int RandomInt(int low, int high)
{
    std::random_device rd;

    std::mt19937 gen(rd()); // seed generator

    std::uniform_int_distribution<> distr(low, high);

    return distr(gen);
}

```

```

/// <summary>
/// Function to Count how many digits are in int
/// </summary>
/// <param name="pos"></param>
/// <returns></returns>
int Count(int element)
{
    int count_MOD1 = 0;

    while (element != 0)
    {
        element = element / 10;
    }
}

```

```

        ++count_MOD1;
    }

    return count_MOD1;
}

/// <summary>
/// Copy the data from another window
/// </summary>
/// <param name="hWnd"></param>
/// <param name="wParam"></param>
/// <param name="lParam"></param>
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];

    InvalidateRect(hWnd, 0, TRUE);
}

/// <summary>
/// Put text to clipboard
/// </summary>
/// <param name="hWnd"></param>
/// <param name="src"></param>
/// <returns></returns>
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();

return 1;
}

/// <summary>
/// Starts the object3
/// </summary>
/// <param name="hWnd"></param>
/// <returns></returns>
void StartObj3(HWND hWnd)
{
    hWndDataCreator = FindWindow("OBJECT3", NULL);

    if (hWndDataCreator == NULL) // the required program is already running
    {
        // call to run the desired program
        WinExec("Object3.exe", SW_SHOW);

        hWndDataCreator = FindWindow("OBJECT3", NULL);
    }

    // form the data as a solid array, for example:
    long params[allValues] = { n_MOD2};

    SendCopyData(hWndDataCreator, hWnd, params, sizeof(params));
}

```

```
#pragma endregion ModifiedFuntions
```

Object3.cpp:

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

```
//
```

```
// First Part
```

```
#include "framework.h"
```

```
#include "pch.h"
```

```
#include "Object3.h"
```

```
#include "Resource.h"
```

```
#include <string>
```

```
#include <vector>
```

```
#include <iostream>
```

```
#include <algorithm>
```

```
using namespace std;
```

```
#define MAX_LOADSTRING 100
```

```
#pragma region VariablesAndFunctions
```

```
// Global variables:
```

```
HINSTANCE hInst; // Current instance
```

```
WCHAR szTitle[MAX_LOADSTRING]; // Header row text
```

```
WCHAR szWindowClass[MAX_LOADSTRING]; // Class name of main window
```

```
char bufferText[1024];
```

```
int n_MOD3;
```

```
std::vector<int> buffer;
```

```
int determinant;
```

```
// Send 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);
```

```
long GetTextFromClipboard(HWND, char*, long);
```

```
void CalculateDeterminant(HWND hWnd);
```

```
void OnCopyData(HWND hWnd, WPARAM wParam, LPARAM lParam);
```

```
void GetMatrixWithoutRowAndColumn(int** matrix, int size, int row, int col, int** newMatrix);
```

```
int MatrixDeterminant(int** matrix, int size);
```

```
#pragma endregion VariablesAndFunctions
```

```
#pragma region DefaultFunctions
```

```
// Second Part
```

```
// Enter Point "wWinMain"
```

```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
```

```
_In_opt_ HINSTANCE hPrevInstance,
```

```
_In_ LPWSTR lpCmdLine,
```

```
_In_ int nCmdShow)
```

```
{
```

```
UNREFERENCED_PARAMETER(hPrevInstance);
```

```
UNREFERENCED_PARAMETER(lpCmdLine);
```

```
// TODO: Place the code here.
```

```
// Global line initialization
```

```

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

return (int)msg.wParam;
}

//
// FUNCTION: MyRegisterClass()
//
// OBJECTIVE: To register the window class.
// Text of Function
/// <summary>
/// Register the window class.
/// </summary>
/// <param name="hInstance">The h instance.</param>
/// <returns></returns>
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);
}

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

```

```

//
// OBJECTIVE: Saves the instance marker and creates the main window
//
// COMMENTARIES:
//
// In this function, the instance marker is saved in a global variable, and also
// the main program window is created and displayed.
//
/// <summary>
/// Saves the instance marker and creates the main window
/// </summary>
/// <param name="hInstance">The h instance.</param>
/// <param name="nCmdShow">The n command show.</param>
/// <returns></returns>
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hInst = hInstance; // Save instance marker in global variable

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

    if (!hWnd)
    {
        return FALSE;
    }

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

    return TRUE;
}

/// <summary>
/// Message handler for "About" window.
/// </summary>
/// <param name="hDlg">The h dialog.</param>
/// <param name="message">The message.</param>
/// <param name="wParam">The w parameter.</param>
/// <param name="lParam">The l parameter.</param>
/// <returns></returns>
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;
}

#pragma endregion

```


#pragma region ModifiedFuntions

```
// Third Part
// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
// OBJECTIVE: Processes messages in the main window.
//
// WM_COMMAND - Process the application menu
// WM_PAINT - Drawing of the main window
// WM_DESTROY - Send message about exit and return
//
//
/// <summary>
/// Processes messages in the main window.
/// </summary>
/// <param name="hWnd">The h WND.</param>
/// <param name="message">The message.</param>
/// <param name="wParam">The w parameter.</param>
/// <param name="lParam">The l parameter.</param>
/// <returns></returns>
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_CREATE:
        {
            GetTextFromClipboard(hWnd, bufferText, sizeof(bufferText));

            SetWindowPos(hWnd, HWND_TOPMOST, 810, 190, 200, 200, SWP_DEFERERASE);
        }
        break;
        case WM_COPYDATA:
        {
            OnCopyData(hWnd, wParam, lParam);

            // CalculateDeterminant(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_DESTROY:
            PostQuitMessage(0);
            break;
    }
}
```

```

case WM_PAINT:
{
    PAINTSTRUCT ps;
    UpdateWindow(hWnd);
    HDC hdc = BeginPaint(hWnd, &ps);

    wchar_t tempValue[256];
    wsprintfW(tempValue, L"%d", determinant);
    TextOut(hdc, 0, 0, tempValue, lstrlen(tempValue));

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

/// <summary>
/// Calculate determinant
/// </summary>
/// <param name="matrix"></param>
/// <param name="n"></param>
/// <returns></returns>
int MatrixDeterminant(int** matrix, int n)
{
    int deter = 0;
    int counter = 1;

    if (n == 1)
    {
        return matrix[0][0];
    }

    else if (n == 2)
    {
        return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
    }
    else
    {
        int** newMatrix = new int* [n - 1];
        for (int i = 0; i < n - 1; i++)
        {
            newMatrix[i] = new int[n - 1];

            for (int j = 0; j < n; j++)
            {
                GetMatrixWithoutRowAndColumn(matrix, n, 0, j, newMatrix);

                deter = deter + (counter * matrix[0][j] * MatrixDeterminant(newMatrix, n - 1));
                counter = -counter;
            }

            for (int i = 0; i < n - 1; i++)
            {
                delete[] newMatrix[i];
            }
            delete[] newMatrix;
        }
    }
}

```

```

    }

    return deter;
}

/// <summary>
/// Gets matrix without row and column
/// </summary>
/// <param name="matrix"></param>
/// <param name="n"></param>
/// <param name="row"></param>
/// <param name="column"></param>
/// <param name="newMatrix"></param>
void GetMatrixWithoutRowAndColumn(int** matrix, int n, int row, int column, int** newMatrix)
{
    int offsetRow = 0;
    int offsetCol = 0;
    for (int i = 0; i < n - 1; i++)
    {
        if (i == row)
        {
            offsetRow = 1;
        }

        offsetCol = 0;
        for (int j = 0; j < n - 1; j++)
        {
            if (j == column)
            {
                offsetCol = 1;
            }

            newMatrix[i][j] = matrix[i + offsetRow][j + offsetCol];
        }
    }
}

/// <summary>
/// Calculates determinant
/// </summary>
/// <param name="hWnd"></param>
void CalculateDeterminant(HWND hWnd)
{
    // dynamic allocation
    int** matrix = new int* [n_MOD3];
    for (int i = 0; i < n_MOD3; ++i)
    {
        matrix[i] = new int[n_MOD3];
    }

    std::string tempBufferForMatrixString = bufferText;

    string currentNumber;

    std::replace(tempBufferForMatrixString.begin(),
        tempBufferForMatrixString.end(), '\n', ' ');

    while (tempBufferForMatrixString != "")
    {
        currentNumber = tempBufferForMatrixString.substr(

```

```

    0, tempBufferForMatrixString.find_first_of(" "));

    buffer.push_back(stod(currentNumber));

    tempBufferForMatrixString = tempBufferForMatrixString.substr(
        tempBufferForMatrixString.find_first_of(" ") + 1);
}

//fill
for (int i = 0; i < n_MOD3; ++i)
{
    for (int j = 0; j < n_MOD3; ++j)
    {
        matrix[i][j] = buffer[i];
    }
}

determinant = MatrixDeterminant(matrix, n_MOD3);

// free
for (int z = 0; z < n_MOD3; ++z)
{
    delete[] matrix[z];
}
delete[] matrix;
}

/// <summary>
/// Copy the data from another window
/// </summary>
/// <param name="hWnd"></param>
/// <param name="wParam"></param>
/// <param name="lParam"></param>
void OnCopyData(HWND hWnd, WPARAM wParam, LPARAM lParam)
{
    COPYDATASTRUCT* cds;
    cds = (COPYDATASTRUCT*)lParam;
    long* p = (long*)cds->lpData;
    n_MOD3 = p[0];

    InvalidateRect(hWnd, 0, TRUE);
}

/// <summary>
/// Get text from Clipboard
/// </summary>
/// <param name="hWnd"></param>
/// <param name="dest"></param>
/// <param name="maxsize"></param>
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;
}

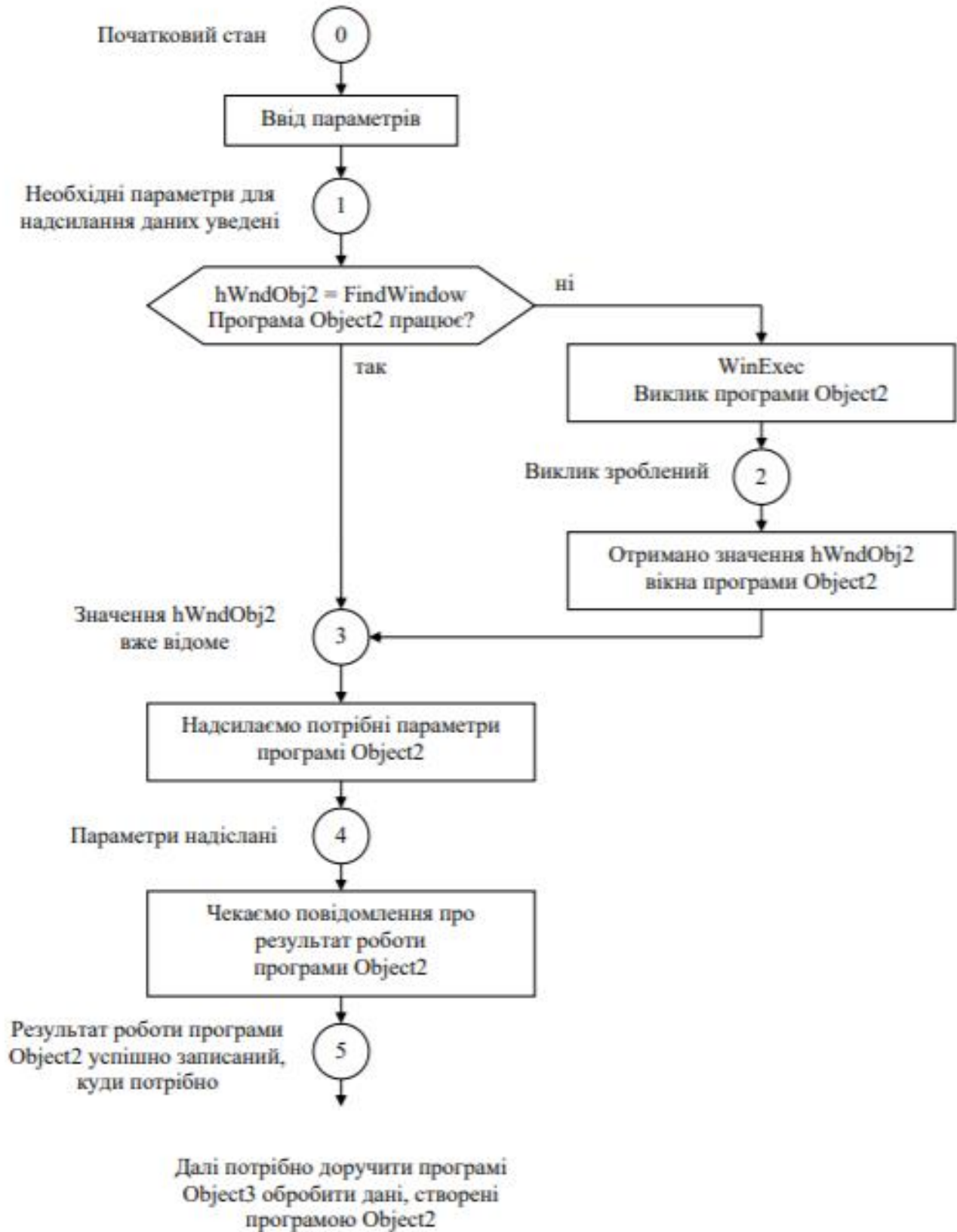
```

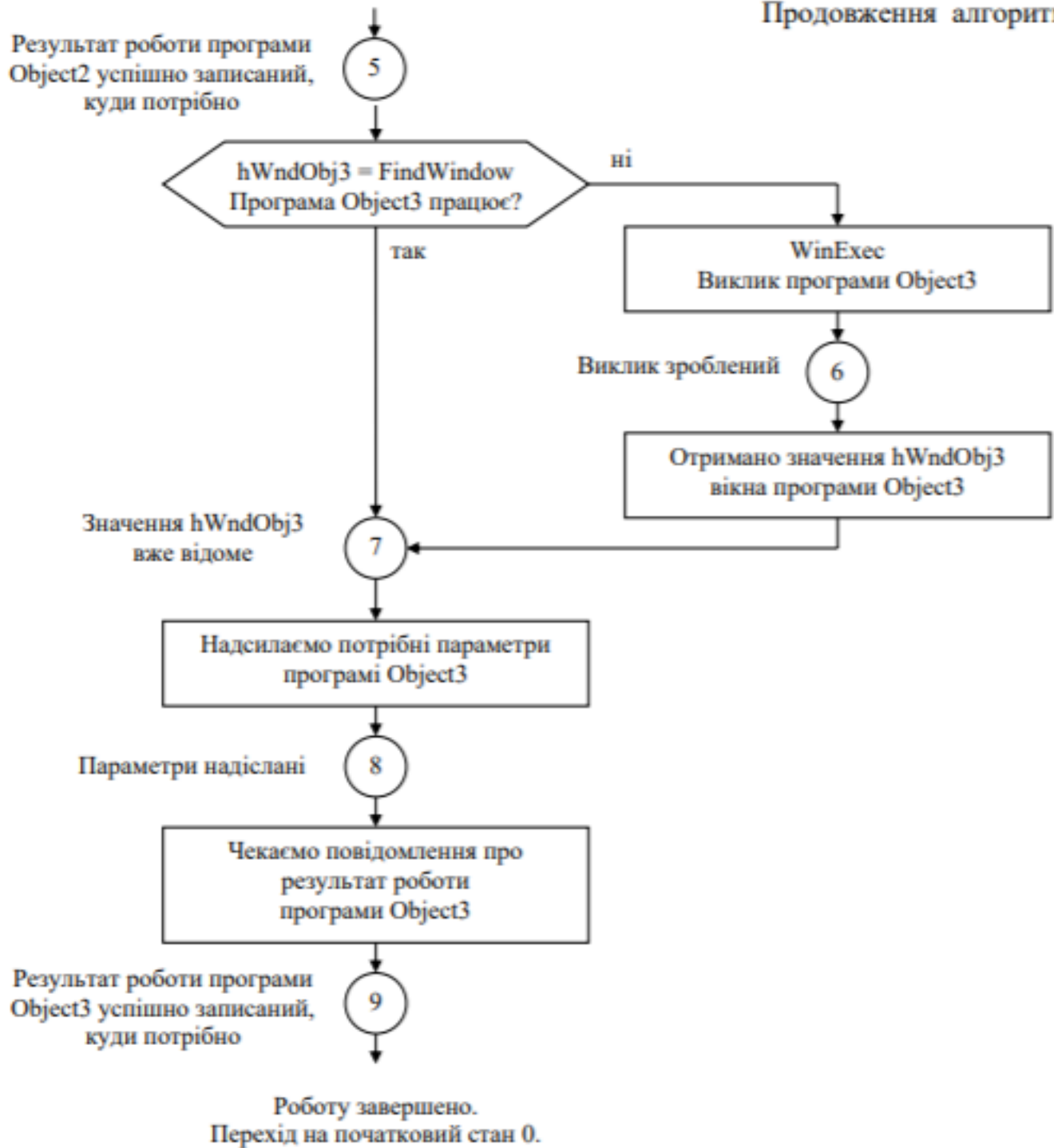
#pragma endregion ModifiedFuntions

Схема послідовності надсилання-обробки повідомлень

Алгоритм обміну повідомленнями об'єктів-компонентів системи

Програма **Object1** спочатку просить щось зробити програму **Object2**





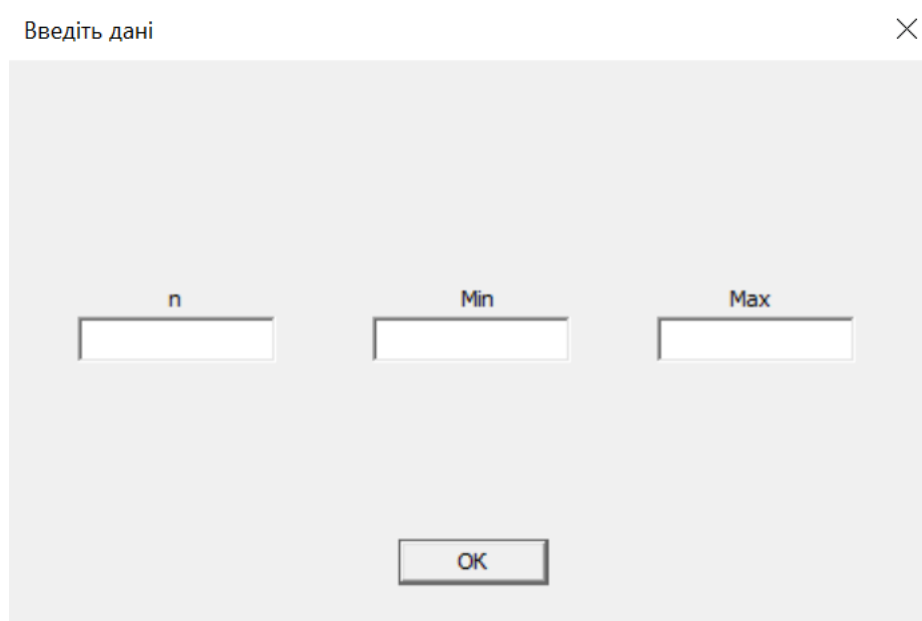
Скріншоти програми:

Також разом з іншими файлами є анімація (.gif) роботи програми

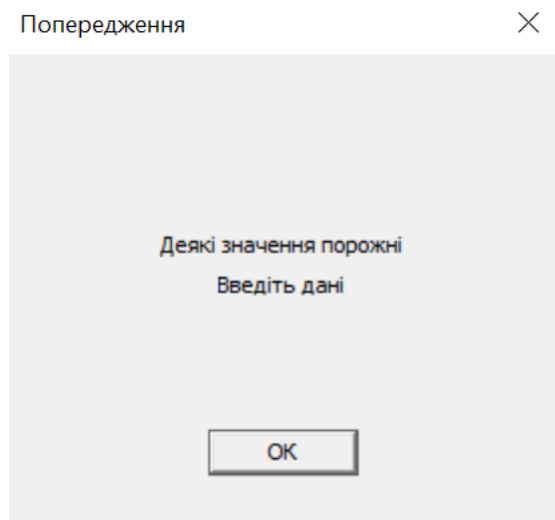
Головне вікно:



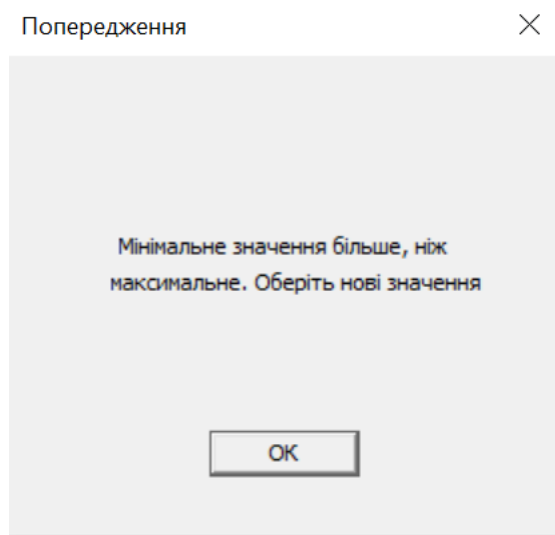
Вікно вводу даних:

The image shows a small dialog box titled "Введіть дані" (Enter data). It has a close button (X) in the top right corner. Inside the dialog, there are three input fields arranged horizontally, labeled "n", "Min", and "Max" above them. Below these fields is a single "OK" button.

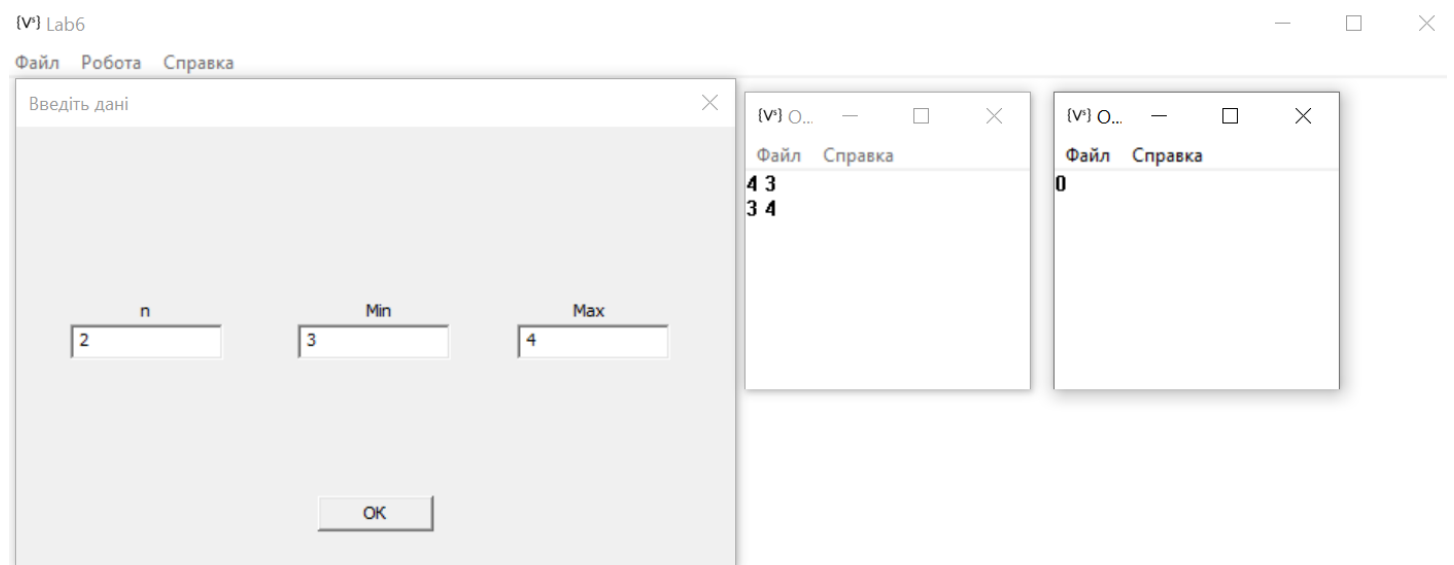
Попередження, коли якийсь із значень дорівнює нулю:



Попередження, коли мінімальне значення більше максимального:



Вікна всіх програм:



Контрольні питання

1. Як запрограмувати виклик у програмі іншої програми?

За допомогою функції WinExec або CreateProcess

2. Як отримати hWnd іншої програми?

Ввести у командний рядок як один з параметрів

3. Як надіслати іншій програмі повідомлення?

За допомогою команд PostMessage та SendMessage

4. Як надіслати іншій програмі одне число?

Щоб надіслати іншій програмі 32-бітове значення value, можна порекомендувати надіслати повідомлення, наприклад, WM_COMMAND, у якому в якості параметра lParam буде записане потрібне значення

5. Як надіслати іншій програмі масив числових значень?

За допомогою SendCopyData, в якій ми використовуємо COPYDATASTRUCT та SendMessage

6. Як запрограмувати обмін інформацією з Clipboard Windows?

Для реалізації Clipboard Windows використовується глобальна динамічна віртуальна пам'ять. Для того, щоб програма записала дані у Clipboard, потрібно викликати функцію API Windows SetClipboardData та нашу власну функцію PutTextToClipboard. А для читання вмісту Clipboard у форматі ANSI-тексту можна скористатися функцією GetTextFromClipboard, яка копіює вміст Clipboard у буфер, вказаний параметром dest

Висновок:

Навчився працювати з кількома програмами, буфером обміну. Записувати та видаляти дані з буферу обміну. Дізнався про обмін інформацією між кількома програмами