**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: Перевірив:

студент 2-го курсу Порєв Віктор Миколайович  
групи **ІМ-11**   
**Букач Кирило Віталійович**  
номер у списку групи: 5

Київ 2022

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

**Завдання за варіантом:**   
Ж= 5;  
**Номер варіанту = Ж mod 4 = 1;**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер варіанту | Lab6 | Object2 | Object3 |
| 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. |

**Вихідний текст головного файлу та модулів:**

**Lab6.cpp** (власний код)**:**

#include "matrix\_dialog.h"  
…

MatrixDialog& matrixDialog = matrixDialog.getInstance();

…

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;  
}

**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();  
};

**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"Неправильний формат", L"Помилка", MB\_OK | MB\_ICONERROR);  
 break;  
 }  
  
 if (min > max || N < 1 || N > 10 || abs(min) > 99999 || abs(max) > 99999)  
 {  
 MessageBox(hDlg, L"Некоректні дані", L"Помилка", 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;  
}  
  
  
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::Start(HINSTANCE hInst, HWND hWnd)  
{  
 DialogBox(hInst, MAKEINTRESOURCE(IDD\_MATRIX), hWnd, Matrix);  
}  
  
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);  
}

**Object2.cpp:**

#include "matrix\_build.h"

…

MatrixBuild& matrixBuild = matrixBuild.getInstance();

…

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  
{  
 switch (message)  
 {  
 case WM\_COPYDATA:  
 SetWindowPos(hWnd, NULL, 540, 40, 800, 800, 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;  
}

**matrix\_build.h:**

#pragma once  
#include "framework.h"  
#include <string>  
#include <random>  
#include <sstream>  
#include <codecvt>  
  
class MatrixBuild  
{  
private:  
 MatrixBuild() {}  
 MatrixBuild(const MatrixBuild& root) = delete;  
 MatrixBuild& operator = (const MatrixBuild&) = delete;  
  
 int N = 0;  
 int min = 0;  
 int max = 0;  
  
 int\*\* matrix = 0;  
 std::string matrixText = "";  
public:  
 static MatrixBuild& getInstance()  
 {  
 static MatrixBuild instance;  
 return instance;  
 }  
  
 void OnCopyData(HWND, WPARAM, LPARAM);  
 void OnPaint(HWND, HDC);  
 int\*\* CreateMatrix(int, int, int);  
 std::string GetMatrixString(int\*\*, int);  
 int PutToClipboard(HWND, const char\*);  
};

**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;  
}

**Object3.cpp:**

#include "matrix\_det.h"

…

MatrixDet& matrixDet = matrixDet.getInstance();

…

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  
{  
 switch (message)  
 {  
 case WM\_CREATE:  
 case WM\_CLIPBOARDUPDATE:  
 SetWindowPos(hWnd, NULL, 1340, 40, 200, 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;  
}

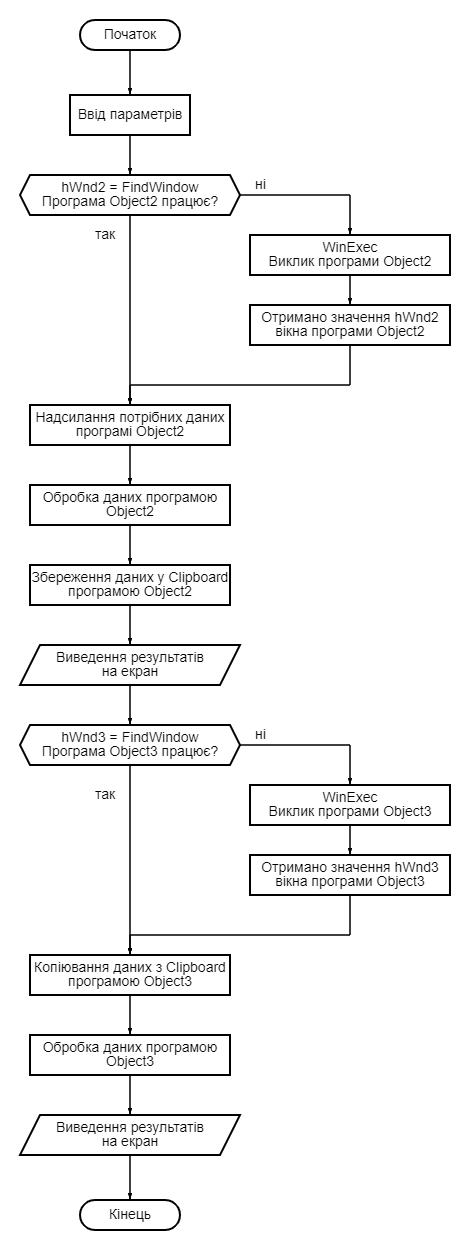
**matrix\_det.h:**

#pragma once  
#include "framework.h"  
#include <string>  
#include <sstream>  
#include <codecvt>  
  
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);  
};

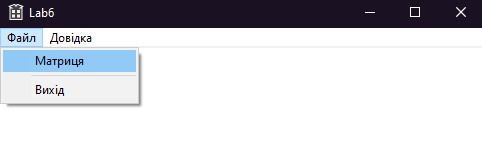
**matrix\_det.cpp:**

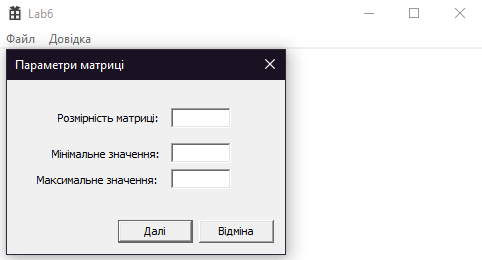
#include "matrix\_det.h"  
  
void MatrixDet::OnCreate(HWND hWnd)  
{  
 GetFromClipboard(hWnd, 10009);  
 matrix = GetMatrix(matrixText);  
 det = GetDet(matrix, N);  
  
 InvalidateRect(hWnd, NULL, TRUE);  
}  
  
int MatrixDet::GetFromClipboard(HWND hWnd, long maxsize)  
{  
 HGLOBAL hglb;  
 LPTSTR 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 << det;  
 ss >> sDet;  
 wsDet = converter.from\_bytes(sDet);  
  
 TextOut(hdc, 50, 50, (LPCWSTR)wsDet.c\_str(), (int)wcslen((LPCWSTR)wsDet.c\_str()));  
}

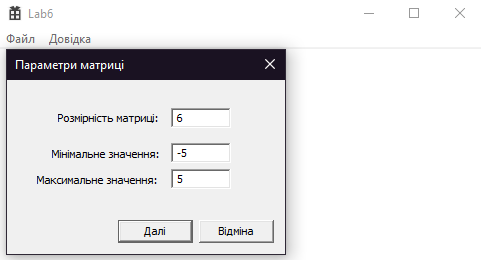
**Схема послідовності надсилання-обробки повідомлень:**

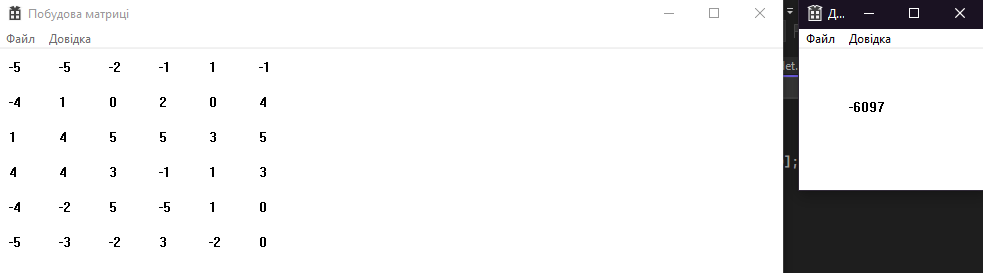


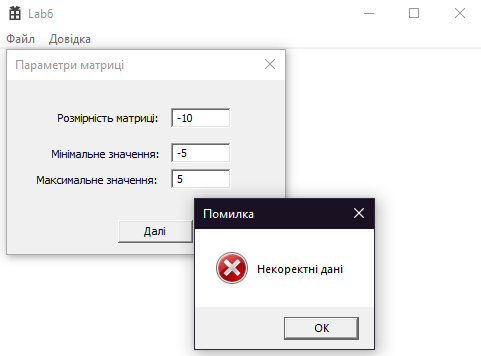
**Демонстрація роботи програми (скріншоти):**

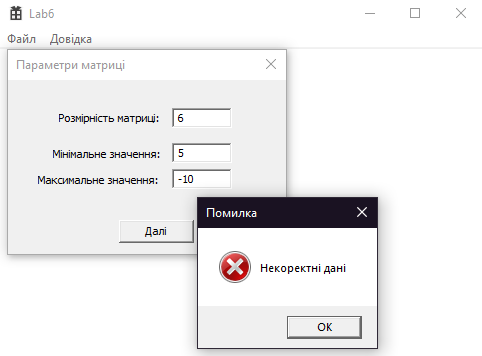
****

****

****

****

****

****

**Висновок:**У результаті виконання даної лабораторної роботи були отримані вміння використовувати засоби обміну інформацією та навички програмувати взаємодії незалежно працюючих компонентів на основі об’єктно-орієнтовного середовища мови С++.