**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 №5**

з дисципліни

«ООП»

на тему «Розробка багатовіконного інтерфейсу користувача

для графічного редактора об’єктів»

Виконав: Перевірив:

студент групи ІП-93 Порєв Віктор Миколайович

Завальнюк Максим Євгенович

номер залікової книжки: 9312

Київ 2020

Мета**:** Мета роботи – отримати вміння та навички програмувати багатовіконний

інтерфейс програми на С++ в об’єктно-орієнтованому стилі.

Зміст

[Мета 2](#_Toc57060978)

[Завдання 3](#_Toc57060979)

[1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab5. 3](#_Toc57060980)

[2. Написати вихідний текст програми згідно варіанту завдання. 3](#_Toc57060981)

[3. Скомпілювати вихідний текст і отримати виконуваний файл програми.. 3](#_Toc57060982)

[4. Перевірити роботу програми. Налагодити програму. 3](#_Toc57060983)

[5. Проаналізувати та прокоментувати результати та вихідний текст програми. 3](#_Toc57060984)

[6. Оформити звіт. 3](#_Toc57060985)

[Варіанти завдань 3](#_Toc57060986)

[Вихідні тексти файлів 4](#_Toc57060987)

[Lab5.cpp 4](#_Toc57060988)

[Shape.cpp 8](#_Toc57060989)

[myeditor.cpp 12](#_Toc57060990)

[myeditor.h 13](#_Toc57060991)

[Shape.h 13](#_Toc57060992)

[Toolbar.h 15](#_Toc57060993)

[Toolbar.cpp 15](#_Toc57060994)

[mytable.cpp 18](#_Toc57060995)

[mytable.h 18](#_Toc57060996)

[Діаграми класів 19](#_Toc57060997)

[Скріншоти програми 20](#_Toc57060998)

[Відображення точки 20](#_Toc57060999)

[Відображення прямої 20](#_Toc57061000)

[Відображення прямокутника 21](#_Toc57061001)

[Відображення еліпсу 21](#_Toc57061002)

[Відображення лінії з точками 22](#_Toc57061003)

[Відображення куба 22](#_Toc57061004)

[Відображення вікна Toolbar 22](#_Toc57061005)

[Контрольні запитання 23](#_Toc57061006)

[Висновок 24](#_Toc57061007)

Завдання**:**

# Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab5.

# Написати вихідний текст програми згідно варіанту завдання.

# Скомпілювати вихідний текст і отримати виконуваний файл програми..

# Перевірити роботу програми. Налагодити програму.

# Проаналізувати та прокоментувати результати та вихідний текст програми.

# Оформити звіт.

# Варіанти завдань

**Number 10 –** програмування об’єкту класу MyEditor на основі класичної реалізації **Singleton**.

Усі **стилі** фігур із **минулої** лабораторної роботи.

# Вихідні тексти файлів

## Lab5.cpp

#include "framework.h"

#include "Lab2.h"

#include "resource.h"

#include "toolbar.h"

#include "myeditor.h"

#include "MyTable.h"

#include <fstream>

using namespace std;

#define MAX\_LOADSTRING 100

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окн

MyEditor\* editorShape = MyEditor::getInstance();// Create editor shape object

Toolbar tool; // Create toolbar object

HWND dlg = NULL;

MyTable\* table = new MyTable;

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

BOOL CALLBACK DlgProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// Инициализация глобальных строк

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_LAB2, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LAB2));

MSG msg;

// Цикл основного сообщения:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// ЦЕЛЬ: Регистрирует класс окна.

//

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\_LAB2));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_LAB2 );

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

//

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

//

// ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

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;

}

// ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

std::string str = "";

switch (message)

{

case WM\_LBUTTONDOWN: //натиснуто ліву кнопку миші у клієнтській частині вікна

if (editorShape) editorShape->OnLBdown(hWnd);

break;

case WM\_LBUTTONUP: //відпущено ліву кнопку миші у клієнтській частині вікна

if (editorShape) editorShape->OnLBup(hWnd);

str = editorShape->GetString();

table->Add(dlg, str);

break;

case WM\_MOUSEMOVE: //пересунуто мишу у клієнтській частині вікна

if (editorShape) editorShape->OnMouseMove(hWnd);

break;

case WM\_INITMENUPOPUP:

//editorShape.OnInitMenuPopup(hWnd, wParam);

break;

case WM\_CREATE:

tool.OnCreate(hWnd); // Create toolbar

if (editorShape) editorShape->Start(new EmptyShape);

break;

case WM\_SIZE:

tool.OnSize(hWnd); // Size of toolbar

break;

case WM\_NOTIFY:

tool.OnNotify(hWnd, lParam); // Notify pressed button

break;

case WM\_PAINT: //потрібно оновлення зображення клієнтської частині вікна

if (editorShape) editorShape->OnPaint(hWnd);

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case IDD\_TABLEMENU:

dlg = CreateDialog(hInst, MAKEINTRESOURCE(IDD\_TABLE), 0, DlgProc);

ShowWindow(dlg, SW\_SHOW);

SetWindowTextA(dlg, "Таблиця");

break;

case IDM\_POINT:

tool.OnPointPressed(hWnd);

if (editorShape) editorShape->Start(new PointShape);

break;

case IDM\_LINE:

tool.OnLinePressed(hWnd);

if (editorShape) editorShape->Start(new LineShape);

break;

case IDM\_RECT:

tool.OnRectPressed(hWnd);

if (editorShape) editorShape->Start(new RectShape);

break;

case IDM\_ELLIPSE:

tool.OnEllipsePressed(hWnd);

if (editorShape) editorShape->Start(new EllipseShape);

break;

case IDM\_CUBE:

tool.OnCubePressed(hWnd);

if (editorShape) editorShape->Start(new CubeShape);

break;

case IDM\_POINT\_LINE:

tool.OnPointLinePressed(hWnd);

if (editorShape) editorShape->Start(new PointLineShape);

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);

break;

default:

return DefWindowProcW(hWnd, message, wParam, lParam);

}

return 0;

}

// Обработчик сообщений для окна "О программе".

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;

}

BOOL CALLBACK DlgProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

string path = "./table.txt";

ifstream f;

switch (uMsg)

{

case WM\_INITDIALOG:

f.open(path);

if (!f.is\_open()) {

throw new exception("Can't open the file");

}

else {

string str;

while (!f.eof()) {

str = "";

getline(f, str);

if (str != "") SendDlgItemMessage(hWnd, IDC\_LIST, LB\_ADDSTRING, 0, (LPARAM)str.c\_str());

}

}

f.close();

return (INT\_PTR)TRUE;

break;

case WM\_COMMAND:

if (LOWORD(wParam) == IDCLEAR)

{

std::ofstream clear;

clear.open(path, std::ofstream::out | std::ofstream::trunc);

clear.close();

SendDlgItemMessage(hWnd, IDC\_LIST, LB\_RESETCONTENT, 0, 0);

}

if (LOWORD(wParam) == IDCANCEL)

{

DestroyWindow(hWnd);

return TRUE;

}

}

return (INT\_PTR)FALSE;

}

## Shape.cpp

#include "framework.h"

#include "shape.h"

Shape::~Shape() {};

void Shape::Set(long x1, long y1, long x2, long y2)

{

xs1 = x1;

ys1 = y1;

xs2 = x2;

ys2 = y2;

}

void PointShape::Show(HDC hdc) {

SetPixel(hdc, xs1, ys1, RGB(0, 0, 0)); // Show point

}

void PointShape::WayTrack(HDC hdc) {}

Shape\* PointShape::Copy() {

return new PointShape();

}

std::string PointShape::GetName() {

return "Point";

}

void LineShape::Show(HDC hdc) {

HPEN hPen, hPenOld;

hPen = CreatePen(PS\_SOLID, 1, RGB(0, 0, 0)); // Create pen

hPenOld = (HPEN)SelectObject(hdc, hPen);

MoveToEx(hdc, xs1, ys1, NULL);

LineTo(hdc, xs2, ys2); // Create line

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

}

void LineShape::WayTrack(HDC hdc) {

HPEN hPen, hPenOld;

hPen = CreatePen(PS\_DOT, 1, RGB(0, 0, 0));

hPenOld = (HPEN)SelectObject(hdc, hPen);

MoveToEx(hdc, xs1, ys1, NULL);

LineTo(hdc, xs2, ys2);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

}

Shape\* LineShape::Copy() {

return new LineShape();

}

std::string LineShape::GetName() {

return "Line";

}

void RectShape::Show(HDC hdc) {

HPEN hPen, hPenOld;

HBRUSH hBrush, hBrushOld;

hPen = CreatePen(PS\_SOLID, 1, RGB(0, 0, 0)); // Create pen

hPenOld = (HPEN)SelectObject(hdc, hPen);

hBrush = CreateSolidBrush(RGB(192, 192, 192));

hBrushOld = (HBRUSH)SelectObject(hdc, hBrush);

SelectObject(hdc, hBrush);

Rectangle(hdc, 2 \* xs1 - xs2, 2 \* ys1 - ys2, xs2, ys2); // Create rectangle

SelectObject(hdc, hBrushOld);

DeleteObject(hBrush);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

}

void RectShape::WayTrack(HDC hdc) {

HPEN hPen, hPenOld;

hPen = CreatePen(PS\_DOT, 1, RGB(0, 0, 0));

hPenOld = (HPEN)SelectObject(hdc, hPen);

Rectangle(hdc, 2 \* xs1 - xs2, 2 \* ys1 - ys2, xs2, ys2);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

}

Shape\* RectShape::Copy() {

return new RectShape();

}

std::string RectShape::GetName() {

return "Rectangle";

}

void EllipseShape::Show(HDC hdc) {

HPEN hPen, hPenOld;

HBRUSH hBrush, hBrushOld;

hPen = CreatePen(PS\_SOLID, 1, RGB(0, 0, 0)); // Create pen

hPenOld = (HPEN)SelectObject(hdc, hPen);

hBrush = CreateSolidBrush(RGB(255, 255, 255));

hBrushOld = (HBRUSH)SelectObject(hdc, hBrush);

SelectObject(hdc, hBrush);

Ellipse(hdc, xs1, ys1, xs2, ys2);

SelectObject(hdc, hBrushOld);

DeleteObject(hBrush);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

};

void EllipseShape::WayTrack(HDC hdc) {

HPEN hPen, hPenOld;

hPen = CreatePen(PS\_DOT, 1, RGB(0, 0, 0));

hPenOld = (HPEN)SelectObject(hdc, hPen);

MoveToEx(hdc, xs1, ys1, NULL);

Ellipse(hdc, xs1, ys1, xs2, ys2);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

}

Shape\* EllipseShape::Copy() {

return new EllipseShape();

}

std::string EllipseShape::GetName() {

return "Ellipse";

}

void PointLineShape::Show(HDC hdc) {

long x1, x2, y1, y2;

x1 = xs1; y1 = ys1; x2 = xs2; y2 = ys2;

LineShape::Set(x1, y1, x2, y2);

LineShape::Show(hdc);

EllipseShape::Set(x1 + 10, y1 + 10, x1 - 10, y1 - 10);

EllipseShape::Show(hdc);

EllipseShape::Set(x2 + 10, y2 + 10, x2 - 10, y2 - 10);

EllipseShape::Show(hdc);

LineShape::Set(x1, y1, x2, y2);

}

void PointLineShape::WayTrack(HDC hdc) {

long x1, x2, y1, y2;

x1 = xs1; y1 = ys1; x2 = xs2; y2 = ys2;

LineShape::Set(x1, y1, x2, y2);

LineShape::WayTrack(hdc);

EllipseShape::Set(x1 + 10, y1 + 10, x1 - 10, y1 - 10);

EllipseShape::WayTrack(hdc);

EllipseShape::Set(x2 + 10, y2 + 10, x2 - 10, y2 - 10);

EllipseShape::WayTrack(hdc);

LineShape::Set(x1, y1, x2, y2);

}

Shape\* PointLineShape::Copy() {

return new PointLineShape();

}

std::string PointLineShape::GetName() {

return "Points with line";

}

void CubeShape::Show(HDC hdc) {

long x1, x2, y1, y2;

x1 = xs1; y1 = ys1; x2 = xs2; y2 = ys2;

RectShape::Set(x1, y1, x1 + 75, y1 + 75);

RectShape::Show(hdc);

RectShape::Set(x2, y2, x2 + 75, y2 + 75); // Create rectangle

RectShape::Show(hdc);

LineShape::Set(x1 - 75, y1 + 75, x2 - 75, y2 + 75);

LineShape::Show(hdc);

LineShape::Set(x1 + 75, y1 + 75, x2 + 75, y2 + 75);

LineShape::Show(hdc);

LineShape::Set(x1 - 75, y1 - 75, x2 - 75, y2 - 75);

LineShape::Show(hdc);

LineShape::Set(x1, y1, x2, y2);

}

void CubeShape::WayTrack(HDC hdc) {

long x1, x2, y1, y2;

x1 = xs1; y1 = ys1; x2 = xs2; y2 = ys2;

RectShape::Set(x1, y1, x1 + 75, y1 + 75);

RectShape::WayTrack(hdc);

RectShape::Set(x2, y2, x2 + 75, y2 + 75); // Create rectangle

RectShape::WayTrack(hdc);

LineShape::Set(x1 - 75, y1 + 75, x2 - 75, y2 + 75);

LineShape::WayTrack(hdc);

LineShape::Set(x1 + 75, y1 + 75, x2 + 75, y2 + 75);

LineShape::WayTrack(hdc);

LineShape::Set(x1 - 75, y1 - 75, x2 - 75, y2 - 75);

LineShape::WayTrack(hdc);

LineShape::Set(x1, y1, x2, y2);

}

Shape\* CubeShape::Copy() {

return new CubeShape();

}

std::string CubeShape::GetName() {

return "Cube";

}

void EmptyShape::Show(HDC hdc) {}

void EmptyShape::WayTrack(HDC hdc) {}

Shape\* EmptyShape::Copy() {

return new EmptyShape();

}

std::string EmptyShape::GetName() {

return "Empty shape";

}

## myeditor.cpp

#include "framework.h"

#include "myeditor.h"

int c = 0;

static bool isPressed = false;

static const int MY\_SHAPE\_ARRAY\_SIZE = 110;

Shape\*\* pcshape = new Shape \* [MY\_SHAPE\_ARRAY\_SIZE];

MyEditor\* MyEditor::p\_instance = 0;

MyEditor\* MyEditor::getInstance() {

if (!p\_instance) p\_instance = new MyEditor();

return p\_instance;

}

MyEditor::~MyEditor() {

for (int i = 0; i < c; i++) delete pcshape[i];

delete pcshape;

}

void MyEditor::Start(Shape\* shape) {

pcshape[c] = shape;

}

void MyEditor::OnLBdown(HWND hWnd) {

POINT pt;

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x1 = x2 = pt.x;

y1 = y2 = pt.y;

isPressed = true;

}

void MyEditor::OnLBup(HWND hWnd) {

POINT pt;

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x2 = pt.x;

y2 = pt.y;

isPressed = false;

pcshape[c]->Set(x1, y1, x2, y2);

c++;

InvalidateRect(hWnd, NULL, TRUE);

pcshape[c] = pcshape[c - 1]->Copy();

}

void MyEditor::OnMouseMove(HWND hWnd) {

if (isPressed) {

POINT pt;

HDC hdc = GetDC(hWnd);

SetROP2(hdc, R2\_NOTXORPEN);

MoveToEx(hdc, x1, y1, NULL);

pcshape[c]->Set(x1, y1, x2, y2);

pcshape[c]->WayTrack(hdc);

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x2 = pt.x;

y2 = pt.y;

MoveToEx(hdc, x1, y1, NULL);

pcshape[c]->Set(x1, y1, x2, y2);

pcshape[c]->WayTrack(hdc);

ReleaseDC(hWnd, hdc);

}

}

void MyEditor::OnPaint(HWND hWnd) {

PAINTSTRUCT ps;

HDC hdc;

hdc = BeginPaint(hWnd, &ps);

for (int i = 0; i < c; i++) {

if (pcshape[i]) pcshape[i]->Show(hdc);

}

EndPaint(hWnd, &ps);

}

std::string MyEditor::GetString() {

std::string str = "Figure: " + pcshape[c]->GetName() + "|"

+ "x1: " + std::to\_string(x1) + "|"

+ "y1: " + std::to\_string(y1) + "|"

+ "x2: " + std::to\_string(x2) + "|"

+ "y2: " + std::to\_string(y2) + "|";

return str;

}

## myeditor.h

#pragma once

#include "shape.h"

#include "resource.h"

class MyEditor {

private:

static MyEditor\* p\_instance;

MyEditor() {};

MyEditor(const MyEditor&);

MyEditor& operator=(MyEditor&);

public:

static MyEditor\* getInstance();

int x1, y1, x2, y2;

~MyEditor();

void Start(Shape\*);

void OnLBdown(HWND);

void OnLBup(HWND);

void OnMouseMove(HWND);

void OnPaint(HWND);

std::string GetString();

};

extern Shape\*\* pcshape;

extern int c;

## Shape.h

class Shape {

protected:

long xs1, ys1, xs2, ys2;

public:

~Shape();

void Set(long, long, long, long);

virtual void Show(HDC) = 0;

virtual void WayTrack(HDC) = 0;

virtual Shape\* Copy() = 0;

};

class PointShape : public Shape

{

void Show(HDC);

void WayTrack(HDC);

virtual Shape\* Copy();

};

class LineShape : public virtual Shape

{

public:

virtual void Show(HDC);

void WayTrack(HDC);

virtual Shape\* Copy();

};

class RectShape : public virtual Shape

{

public:

virtual void Show(HDC);

void WayTrack(HDC);

virtual Shape\* Copy();

};

class EllipseShape : public virtual Shape

{

public:

virtual void Show(HDC);

void WayTrack(HDC);

virtual Shape\* Copy();

};

class PointLineShape : public LineShape, public EllipseShape {

public:

void Show(HDC);

void WayTrack(HDC);

virtual Shape\* Copy();

};

class CubeShape : public RectShape, public LineShape {

public:

void Show(HDC);

void WayTrack(HDC);

virtual Shape\* Copy();

};

class EmptyShape : public Shape {

virtual void Show(HDC);

virtual Shape\* Copy();

virtual void WayTrack(HDC);

};

## Toolbar.h

#pragma once

#define ID\_TOOL\_POINT 32805

#define ID\_TOOL\_LINE 32806

#define ID\_TOOL\_RECT 32807

#define ID\_TOOL\_ELLIPSE 32809

#define ID\_TOOL\_LINE\_POINT 32855

#define ID\_TOOL\_CUBE 32813

#define IDC\_MY\_TOOLBAR 32811

// Declare Toolbar class

class Toolbar {

public:

void OnCreate(HWND hWnd);

void OnSize(HWND);

void OnPointPressed(HWND hWnd);

void OnLinePressed(HWND hWnd);

void OnRectPressed(HWND hWnd);

void OnCubePressed(HWND hWnd);

void OnEllipsePressed(HWND hWnd);

void OnPointLinePressed(HWND hWnd);

void OnNotify(HWND, LPARAM);

void OffPressed(int);

};

## Toolbar.cpp

#include "framework.h"

#include "resource1.h"

#include "toolbar.h"

HWND hwndToolBar = NULL;

extern HINSTANCE hInst;

bool point, line, rect, ellipse, pointline, cube = 0;

int whatPressed = NULL;

void Toolbar::OnCreate(HWND hWnd)

{

TBBUTTON tbb[7];

ZeroMemory(tbb, sizeof(tbb));

tbb[0].iBitmap = 0;

tbb[0].fsState = TBSTATE\_ENABLED;

tbb[0].fsStyle = TBSTYLE\_BUTTON;

tbb[0].idCommand = ID\_TOOL\_POINT;

tbb[1].iBitmap = 1;

tbb[1].fsState = TBSTATE\_ENABLED;

tbb[1].fsStyle = TBSTYLE\_BUTTON;

tbb[1].idCommand = ID\_TOOL\_LINE;

tbb[2].iBitmap = 2; //індекс зображення у BITMAP

tbb[2].fsState = TBSTATE\_ENABLED;

tbb[2].fsStyle = TBSTYLE\_BUTTON;

tbb[2].idCommand = ID\_TOOL\_RECT;

tbb[3].iBitmap = 3;

tbb[3].fsState = TBSTATE\_ENABLED;

tbb[3].fsStyle = TBSTYLE\_BUTTON;

tbb[3].idCommand = ID\_TOOL\_ELLIPSE;

tbb[4].iBitmap = 4;

tbb[4].fsState = TBSTATE\_ENABLED;

tbb[4].fsStyle = TBSTYLE\_BUTTON;

tbb[4].idCommand = ID\_TOOL\_LINE\_POINT;

tbb[5].iBitmap = 5;

tbb[5].fsState = TBSTATE\_ENABLED;

tbb[5].fsStyle = TBSTYLE\_BUTTON;

tbb[5].idCommand = ID\_TOOL\_CUBE;

tbb[6].iBitmap = 6;

tbb[6].fsState = TBSTATE\_ENABLED;

tbb[6].fsStyle = TBSTYLE\_SEP; //роздільник груп кнопок

tbb[6].idCommand = 6;

hwndToolBar = CreateToolbarEx(hWnd,

WS\_CHILD | WS\_VISIBLE | WS\_BORDER | WS\_CLIPSIBLINGS | CCS\_TOP | TBSTYLE\_TOOLTIPS,

IDC\_MY\_TOOLBAR,

6, //кількість зображень у BITMAP

hInst,

IDB\_BITMAP1, //ID ресурсу BITMAP

tbb,

6, //кількість кнопок (разом з роздільником)

24, 24, 24, 24, //розміри кнопок та зображень BITMAP

sizeof(TBBUTTON));

}

void Toolbar::OnSize(HWND hWnd) {

RECT rc, rw;

if (hwndToolBar) {

GetClientRect(hWnd, &rc);

GetWindowRect(hwndToolBar, &rw);

MoveWindow(hwndToolBar, 0, 0, rc.right - rc.left, rw.bottom - rw.top, FALSE);

}

}

void Toolbar::OnPointPressed(HWND hWnd) {

SetWindowTextA(hWnd, "Крапка");

point = 1;

line = rect = ellipse = pointline = cube = 0;

OffPressed(ID\_TOOL\_POINT);

SendMessage(hwndToolBar, TB\_PRESSBUTTON, ID\_TOOL\_POINT, point);

}

void Toolbar::OnLinePressed(HWND hWnd) {

SetWindowTextA(hWnd, "Лінія");

line = 1;

point = rect = ellipse = pointline = cube = 0;

OffPressed(ID\_TOOL\_LINE);

SendMessage(hwndToolBar, TB\_PRESSBUTTON, ID\_TOOL\_LINE, line);

}

void Toolbar::OnRectPressed(HWND hWnd) {

SetWindowTextA(hWnd, "Прямокутник");

rect = 1;

point = line = ellipse = pointline = cube = 0;

OffPressed(ID\_TOOL\_RECT);

SendMessage(hwndToolBar, TB\_PRESSBUTTON, ID\_TOOL\_RECT, rect);

}

void Toolbar::OnEllipsePressed(HWND hWnd) {

SetWindowTextA(hWnd, "Еліпс");

ellipse = 1;

point = line = rect = pointline = cube = 0;

OffPressed(ID\_TOOL\_ELLIPSE);

SendMessage(hwndToolBar, TB\_PRESSBUTTON, ID\_TOOL\_ELLIPSE, ellipse);

}

void Toolbar::OnPointLinePressed(HWND hWnd) {

SetWindowTextA(hWnd, "Лінія з точками");

pointline = 1;

point = line = rect = ellipse = cube = 0;

OffPressed(ID\_TOOL\_LINE\_POINT);

SendMessage(hwndToolBar, TB\_PRESSBUTTON, ID\_TOOL\_LINE\_POINT, pointline);

}

void Toolbar::OnCubePressed(HWND hWnd) {

SetWindowTextA(hWnd, "Куб");

cube = 1;

point = line = rect = ellipse = pointline = 0;

OffPressed(ID\_TOOL\_CUBE);

SendMessage(hwndToolBar, TB\_PRESSBUTTON, ID\_TOOL\_CUBE, cube);

}

void Toolbar::OnNotify(HWND hWnd, LPARAM lParams) {

LPNMHDR pnhm = (LPNMHDR)lParams;

LPSTR pText;

if (pnhm->code == TTN\_NEEDTEXT)

{

LPTOOLTIPTEXT lpttt = (LPTOOLTIPTEXT)lParams;

switch (lpttt->hdr.idFrom)

{

case ID\_TOOL\_POINT:

pText = (LPSTR)"Точка";

break;

case ID\_TOOL\_LINE:

pText = (LPSTR)"Лінія";

break;

case ID\_TOOL\_RECT:

pText = (LPSTR)"Прямокутник";

break;

case ID\_TOOL\_ELLIPSE:

pText = (LPSTR)"Еліпс";

break;

case ID\_TOOL\_LINE\_POINT:

pText = (LPSTR)"Лінія з точками";

break;

case ID\_TOOL\_CUBE:

pText = (LPSTR)"Куб";

break;

default: pText = (LPSTR)"Щось невідоме";

}

lstrcpy(lpttt->szText, pText);

}

}

void Toolbar::OffPressed(int id) {

SendMessage(hwndToolBar, TB\_PRESSBUTTON, whatPressed, 0);

whatPressed = id;

}

## mytable.cpp

#include "framework.h"

#include "Mytable.h"

#include <fstream>

using namespace std;

void MyTable::Add(HWND hWndDlg, string str) {

ofstream f;

string path = "./table.txt";

f.open(path, ofstream::app);

if (!f.is\_open()) {

throw new exception("Can't open the file");

}

else {

f << str << "\n";

}

f.close();

SendDlgItemMessage(hWndDlg, IDC\_LIST, LB\_ADDSTRING, 0, (LPARAM)str.c\_str());

}

## mytable.h

#pragma once

#include "resource2.h"

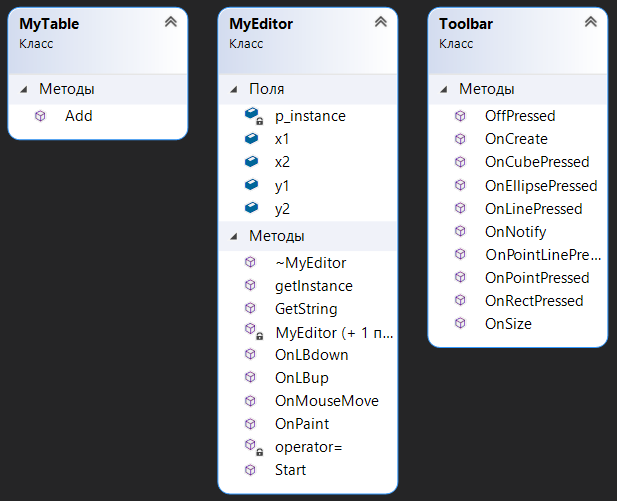
class MyTable {

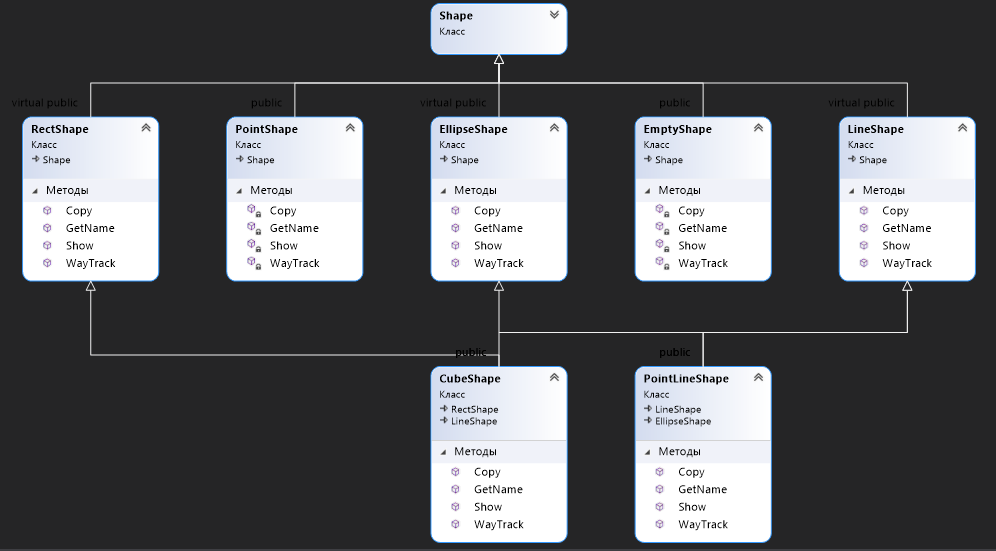
public:

void Add(HWND, std::string);

};

# Діаграми класів





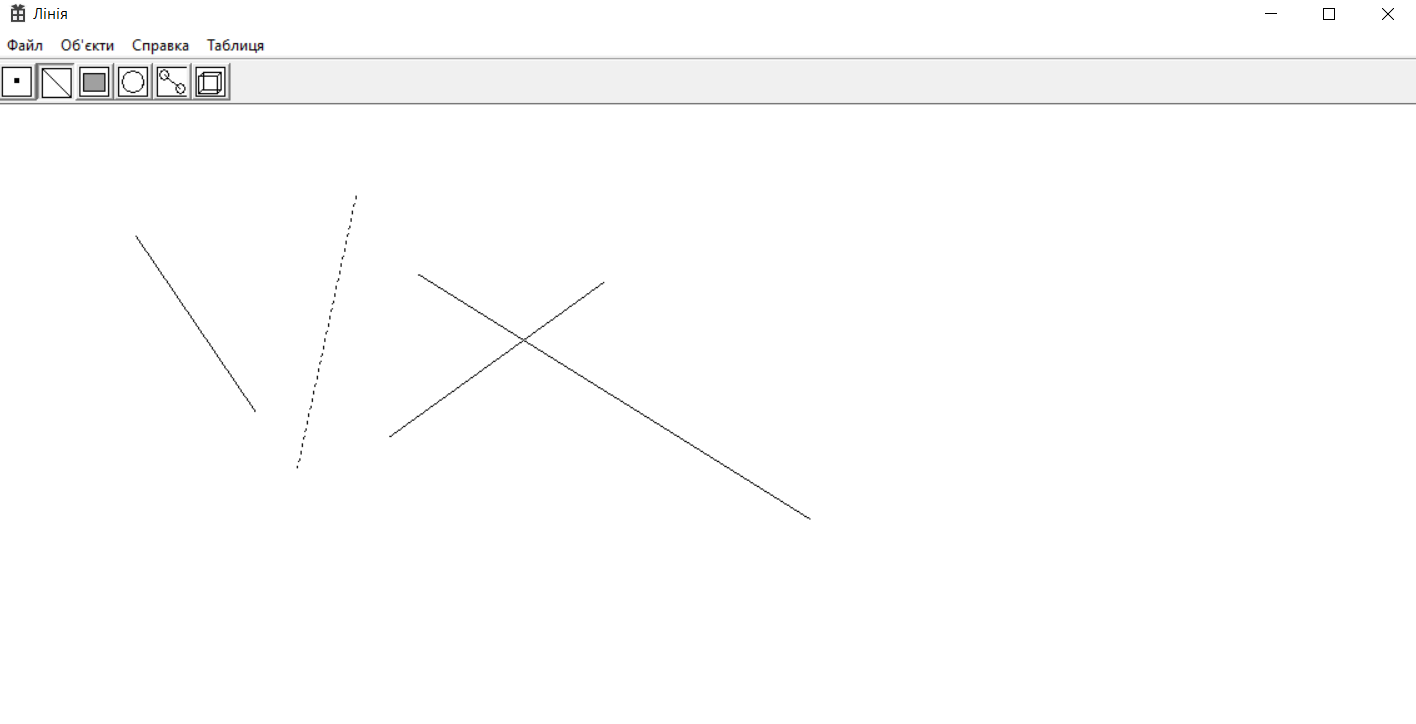
# Скріншоти програми

## Відображення точки

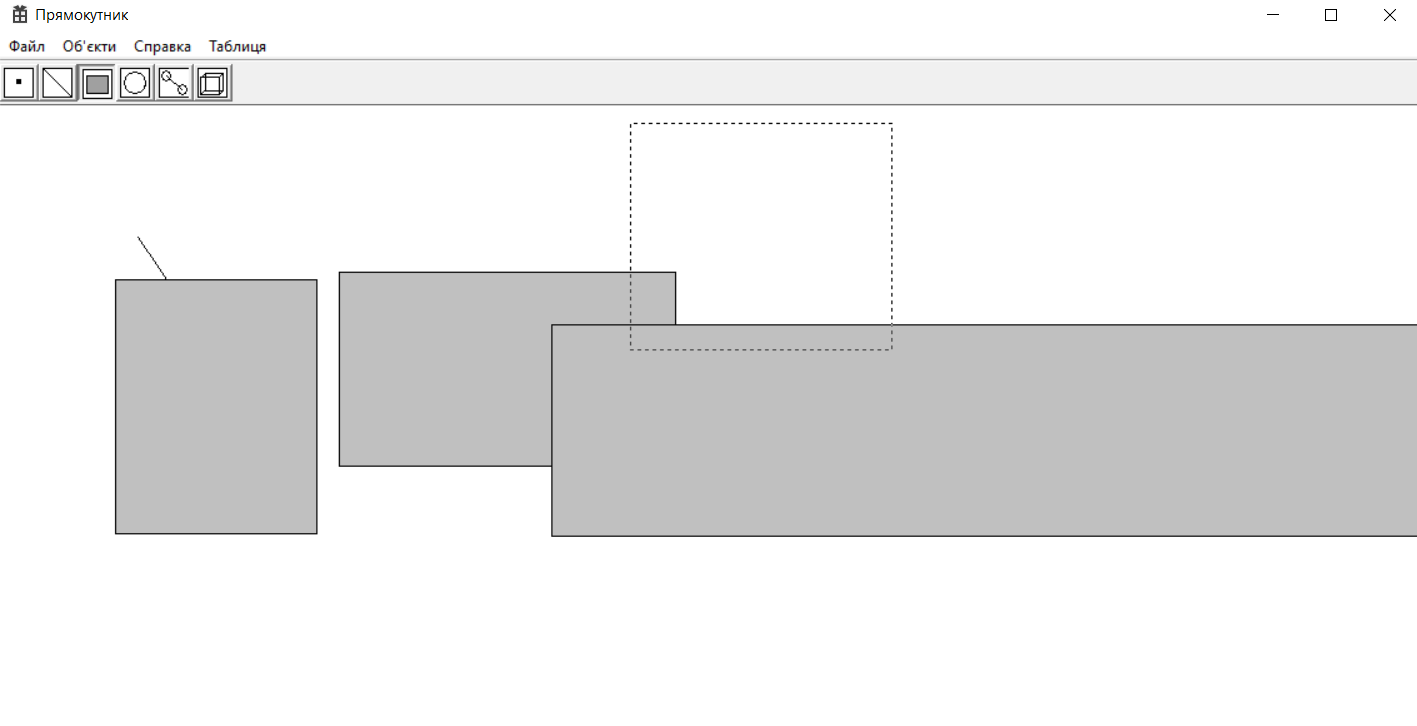


Точки є малими, тому я збільшив для комфортності скріншот

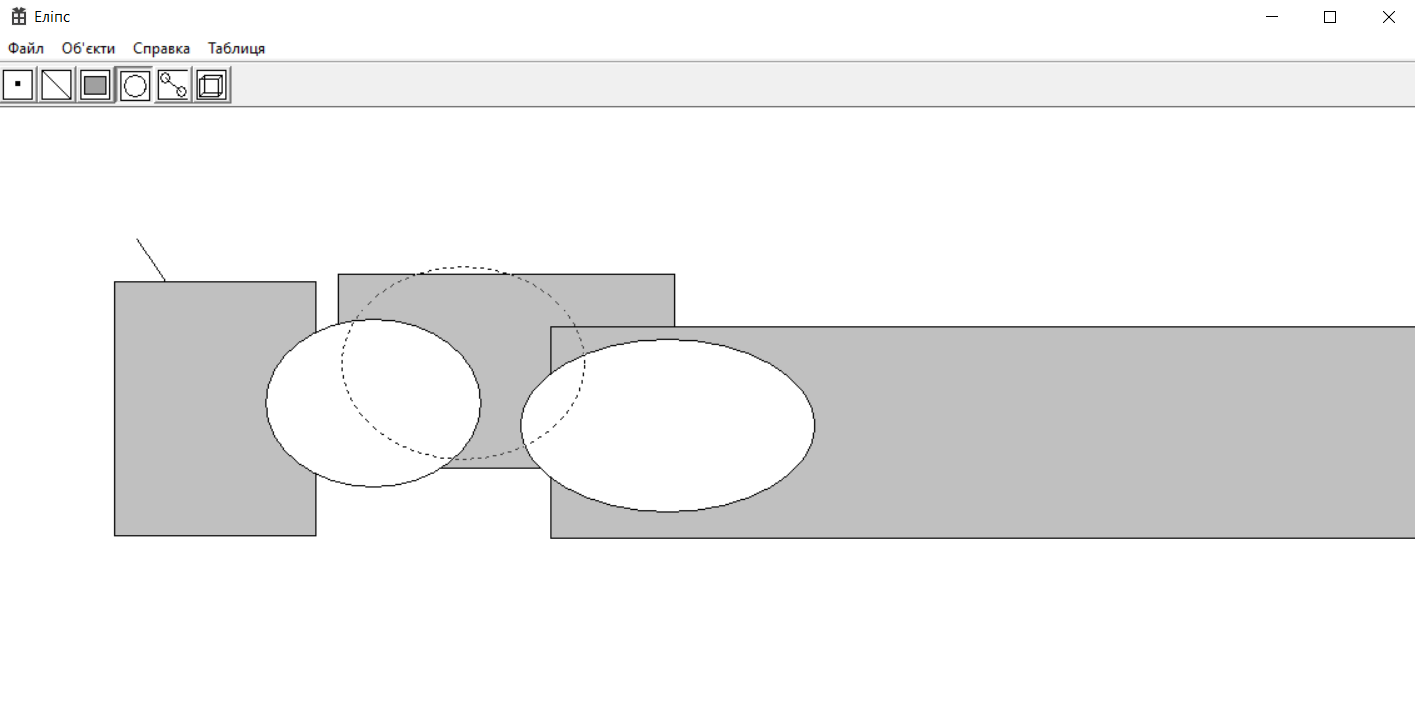
## Відображення прямої



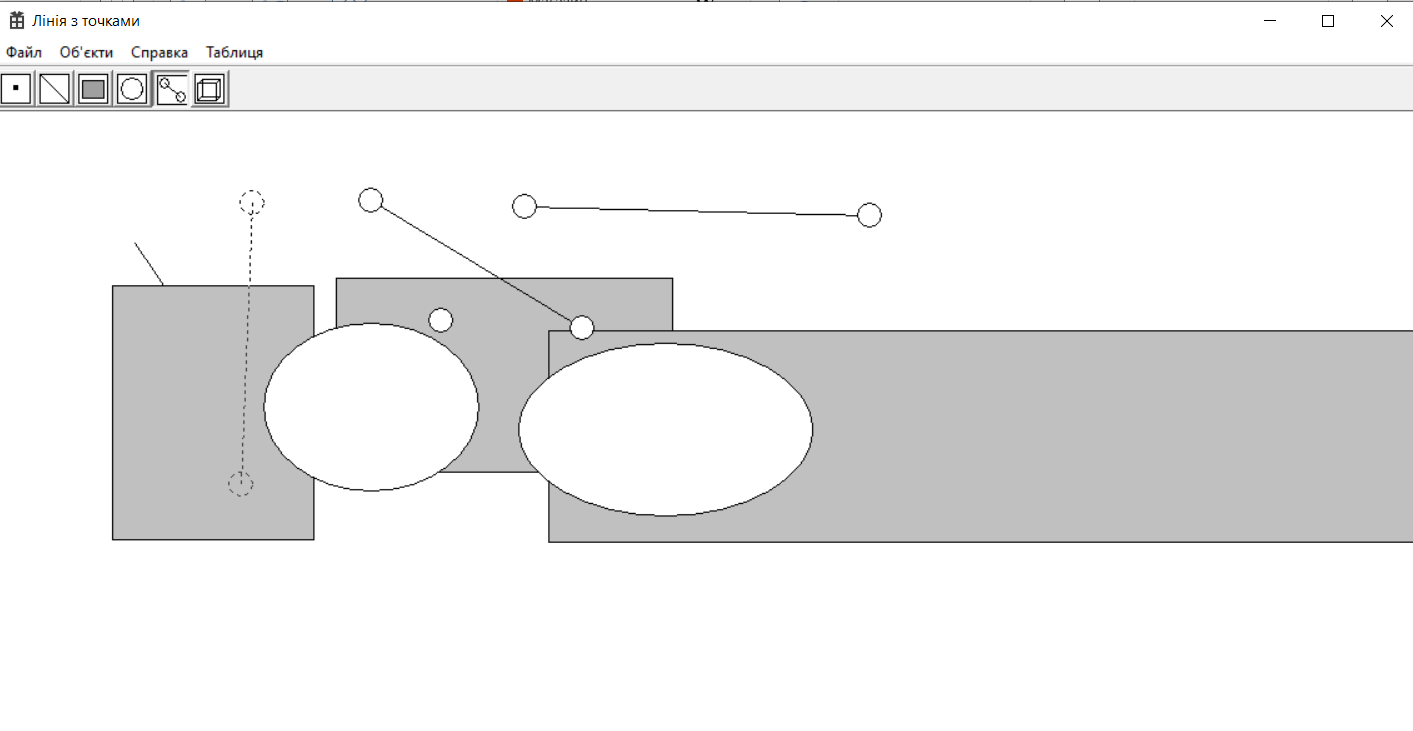
## Відображення прямокутника



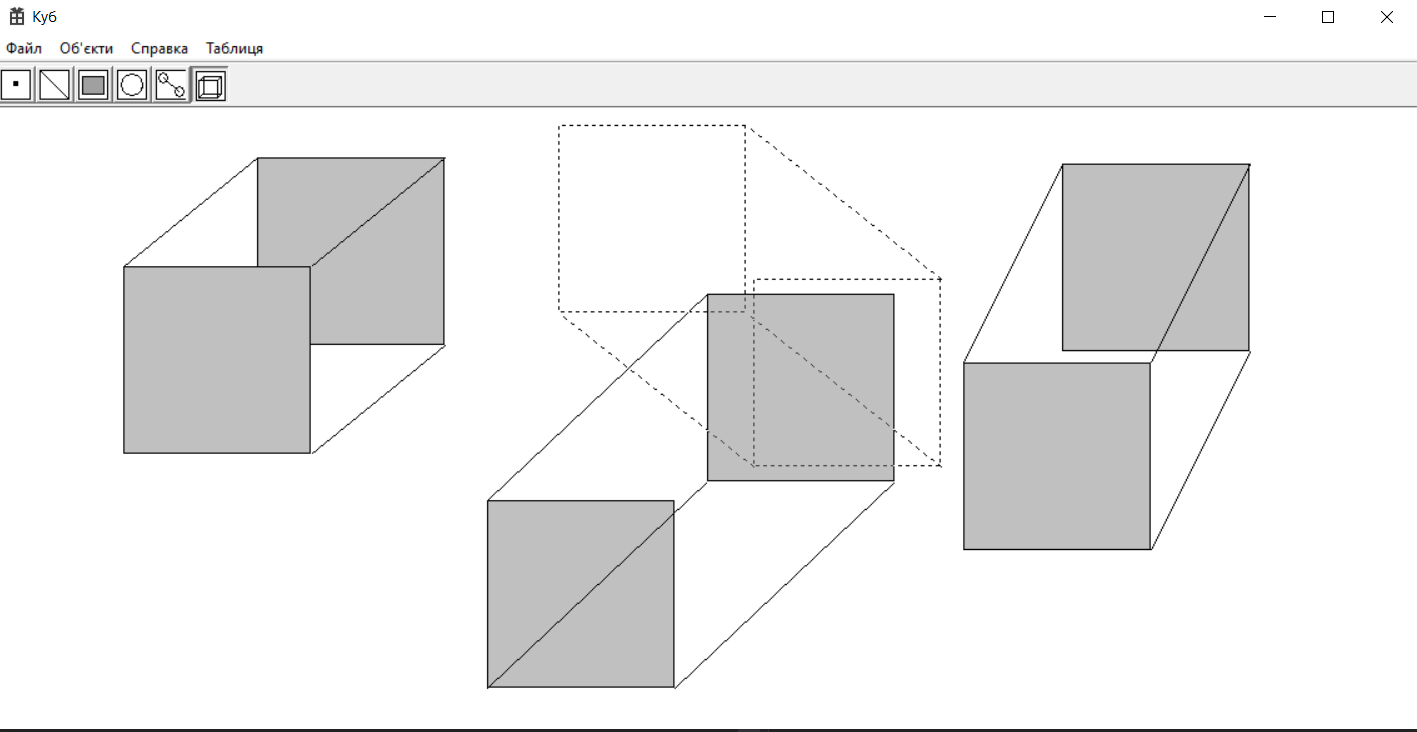
## Відображення еліпсу



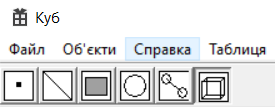
## Відображення лінії з точками



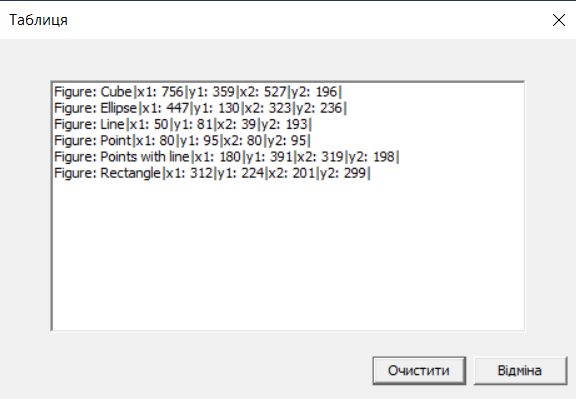
## Відображення куба



## Відображення вікна Toolbar



## Відображення таблиці



# Контрольні запитання

1. Що таке Singleton?

**Singleton** (у перекладі з англ. Одинак) — шаблон проектування, відноситься до класу твірних шаблонів. Гарантує, що клас матиме тільки один екземпляр, і забезпечує глобальну точку доступу до цього екземпляра.

1. Чим відрізняється класична реалізація Singleton від Singleton Меєрса?

Тим, що у класичній реалізації створюється окрема змінна instance та операції проводяться з нею, а у *Singleton Меєрса* цю змінну приховують у функцію.

1. Як запрограмувати немодальне діалогове вікно?

Потрібно створити окремий ресурс та додати там вікно. Далі слід задекларувати функцію, подібну до функції головного вікна, але замінити назву. При натисканні на пункт меню немодального діалогового вікна викликати цю функцію вікна з його власним ID.

1. Як запрограмувати запис у файл об’єктів - геометричних форм?

Ми використовуємо бібліотеку **ofstream**. Спершу відкриваємо файл функцією file.open(), потім записуємо у файл інформацію: file << str << "\n". Потім не забуваємо закрити файл функцією file.close(). А щоб отримати інформацію геометричних форм, потрібно додати до класу **MyEditor** функцію **GetString** і з неї повертати рядок типу string зі всіма даними.

1. Покажіть у програмі поліморфізм.

Прикладом поліморфізму в нас є клас **MyEditor,** тому що він має багато функцій, які можуть обробляти не один тип.

# Висновок

У ході виконання лабораторної роботи я отримав вміння та навички у програмуванні багатовіконного інтерфейсу на С++ в ООП, а також навички у роботі із файлами.