**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 №4**

з дисципліни

«ООП»

на тему «Вдосконалення структури коду графічного редактора об’єктів на C++»

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 проектування класів, виконавши

модернізацію коду графічного редактора в об’єктно-орієнтованому стилі для

забезпечення зручного додавання нових типів об'єктів..

Зміст

[Мета 2](#_Toc55499204)

[Завдання 3](#_Toc55499205)

[1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab4. 3](#_Toc55499206)

[2. Написати вихідний текст програми згідно варіанту завдання. 3](#_Toc55499207)

[3. Скомпілювати вихідний текст і отримати виконуваний файл програми. 3](#_Toc55499208)

[4. Перевірити роботу програми. Налагодити програму. 3](#_Toc55499209)

[5. Проаналізувати та прокоментувати результати та вихідний текст програми. 3](#_Toc55499210)

[6. Оформити звіт. 3](#_Toc55499211)

[Варіанти завдань 3](#_Toc55499212)

[Вихідні тексти файлів 4](#_Toc55499213)

[Lab4.cpp 4](#_Toc55499214)

[Shape.cpp 8](#_Toc55499215)

[myeditor.cpp 12](#_Toc55499216)

[myeditor.h 13](#_Toc55499217)

[Shape.h 13](#_Toc55499218)

[Toolbar.h 15](#_Toc55499219)

[Toolbar.cpp 15](#_Toc55499220)

[Діаграми класів 18](#_Toc55499221)

[Скріншоти програми 19](#_Toc55499222)

[Відображення точки 19](#_Toc55499223)

[Відображення прямої 19](#_Toc55499224)

[Відображення прямокутника 20](#_Toc55499225)

[Відображення еліпсу 20](#_Toc55499226)

[Відображення лінії з точками 21](#_Toc55499227)

[Відображення куба 21](#_Toc55499228)

[Відображення вікна Toolbar 21](#_Toc55499229)

[Контрольні запитання 22](#_Toc55499230)

[Висновок 23](#_Toc55499231)

Завдання**:**

# Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab4.

# 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

**11 mod 3 = 1 –** статичний масив об’єктів типу Shape

**Гумовий слід** у вигляді пунктирної лінії чорного кольору

**11 mod 2 = 1 –** ввід прямокутника від центру до одного з кутів

**11 mod 5 = 1 –** відображення прямокутника з чорним контуром і (**11 mod 6 = 5**) сірим заповненням

**11 mod 2 = 1 –** ввід еліпсу по двом протилежним кутам охоплюючого прямокутника

**11 mod 5 = 1 –** відображення еліпсу з чорним контуром і білим заповненням

**11 mod 2 = 1 –** відображення поточного типу об’єкту у заголовку в меню

Відображення **лінії з точками** з білим заповненням

Відображення кубу із сірими квадратами

# Вихідні тексти файлів

## Lab4.cpp

// Lab4.cpp : Определяет точку входа для приложения.

//

#include "framework.h"

#include "Lab4.h"

#include "resource.h"

#include "toolbar.h"

#include "myeditor.h"

#define MAX\_LOADSTRING 100

// Глобальные переменные:

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

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

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

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

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Разместите код здесь.

// Инициализация глобальных строк

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

LoadStringW(hInstance, IDC\_LAB4, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LAB4));

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\_LAB4));

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

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

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_LAB4 );

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

//

// КОММЕНТАРИИ:

//

// В этой функции маркер экземпляра сохраняется в глобальной переменной, а также

// создается и выводится главное окно программы.

//

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)

//

// ЦЕЛЬ: Обрабатывает сообщения в главном окне.

//

// WM\_COMMAND - обработать меню приложения

// WM\_PAINT - Отрисовка главного окна

// WM\_DESTROY - отправить сообщение о выходе и вернуться

//

//

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

{

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);

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 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;

}

## 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();

}

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();

}

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();

}

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();

}

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();

}

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);

if (x2 > x1 && y2 < y1)

{

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);

}

if (x2 > x1 && y2 > y1)

{

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);

}

if (x2 < x1 && y2 > y1)

{

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);

}

if (x2 < x1 && y2 < y1)

{

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);

if (x2 > x1 && y2 < y1)

{

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);

}

if (x2 > x1 && y2 > y1)

{

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);

}

if (x2 < x1 && y2 > y1)

{

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);

}

if (x2 < x1 && y2 < y1)

{

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();

}

void EmptyShape::Show(HDC hdc) {}

void EmptyShape::WayTrack(HDC hdc) {}

Shape\* EmptyShape::Copy() {

return new EmptyShape();

}

## myeditor.cpp

#include "framework.h"

#include "myeditor.h"

int c = 0;

static bool isPressed = false;

static const int MY\_SHAPE\_ARRAY\_SIZE = 106;

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);

}

## myeditor.h

#pragma once

#include "shape.h"

#include "resource.h"

class MyEditor {

private:

static MyEditor\* p\_instance;

MyEditor() {};

MyEditor(const 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);

};

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);

}

}

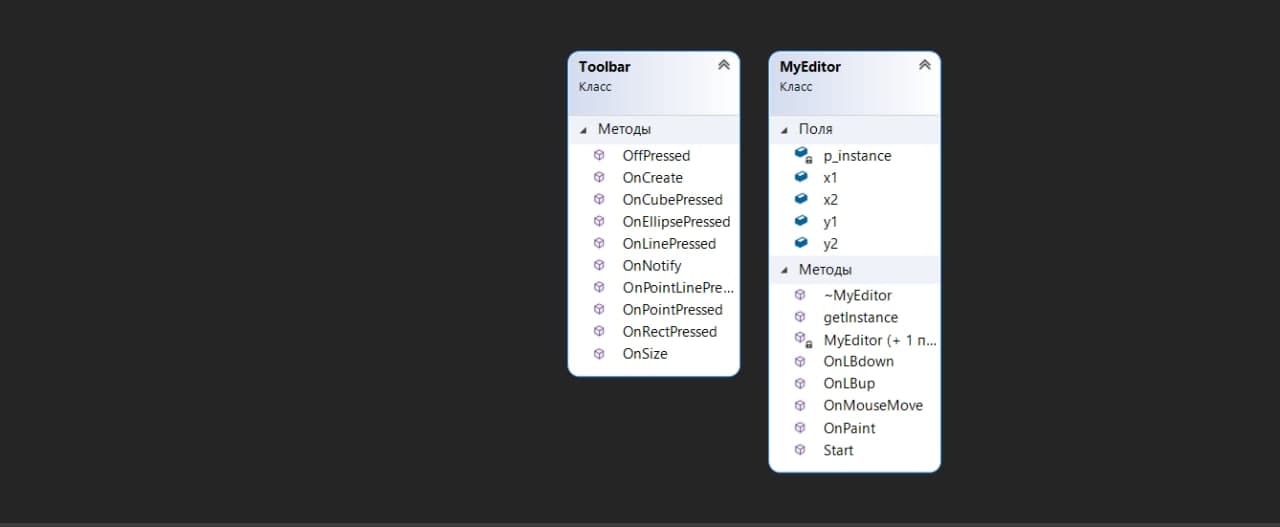
void Toolbar::OffPressed(int id) {

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

whatPressed = id;

}

# Діаграми класів





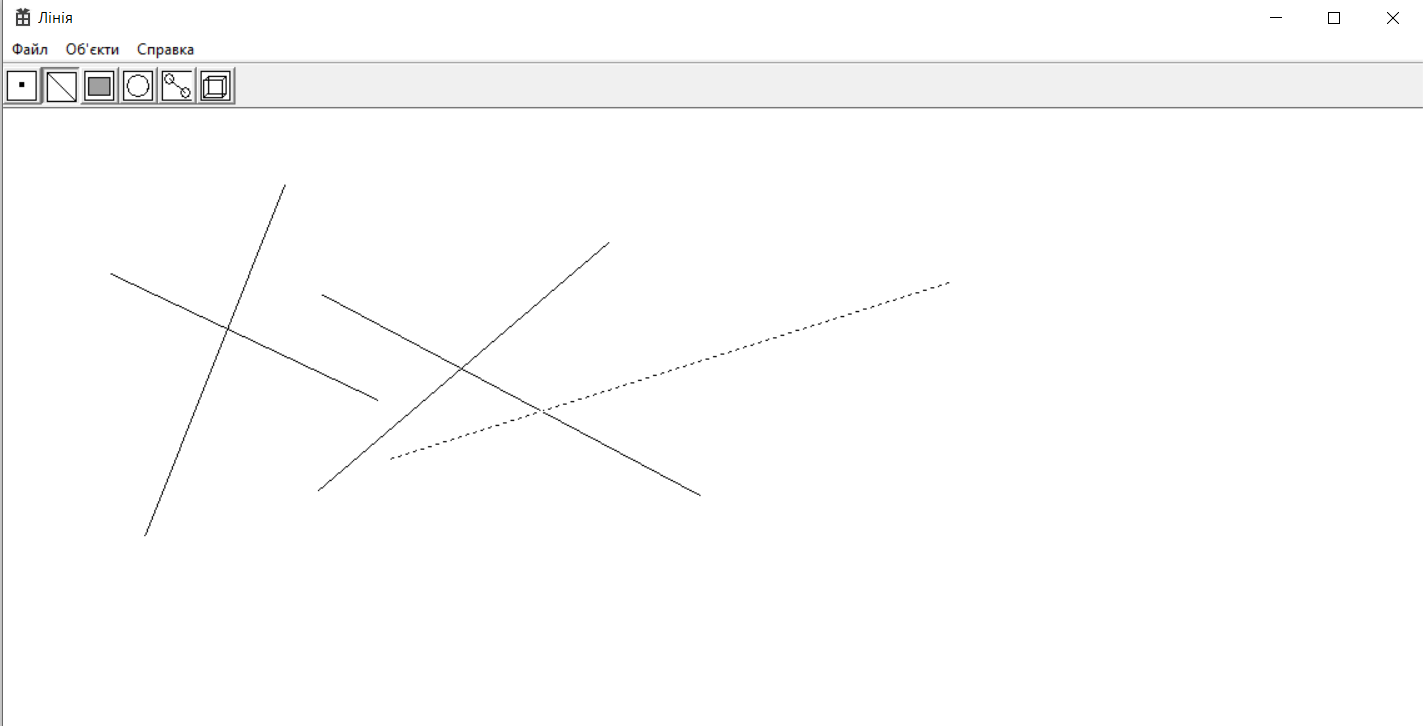
# Скріншоти програми

## Відображення точки

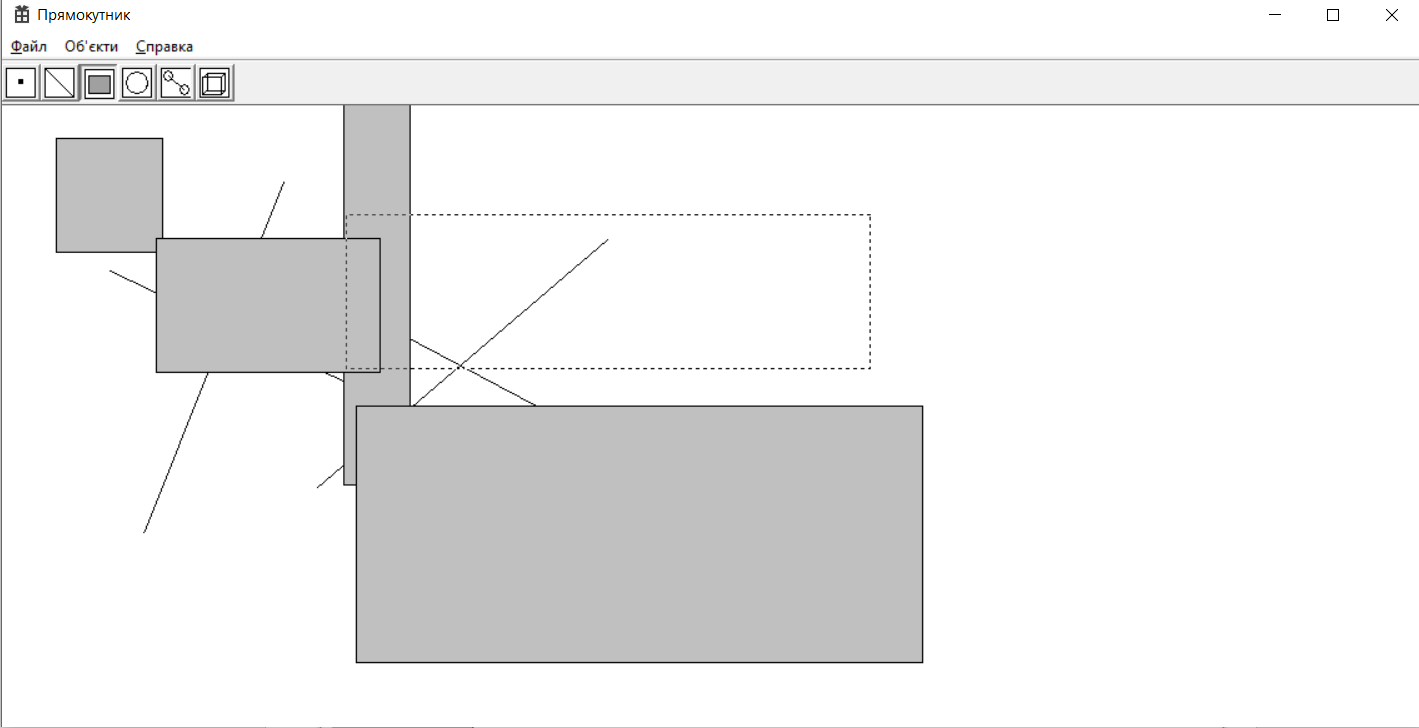


Точки є малими, тому я збільшив для комфортності скріншот

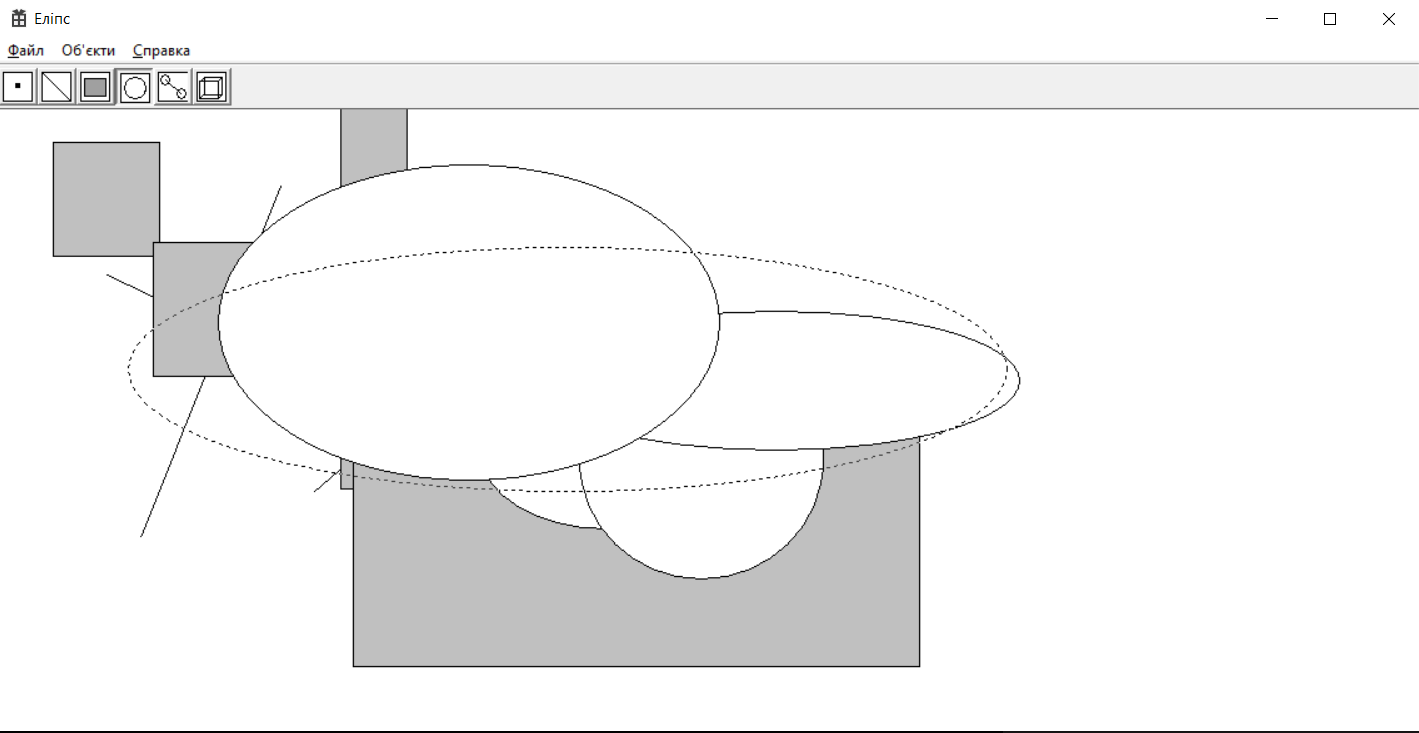
## Відображення прямої



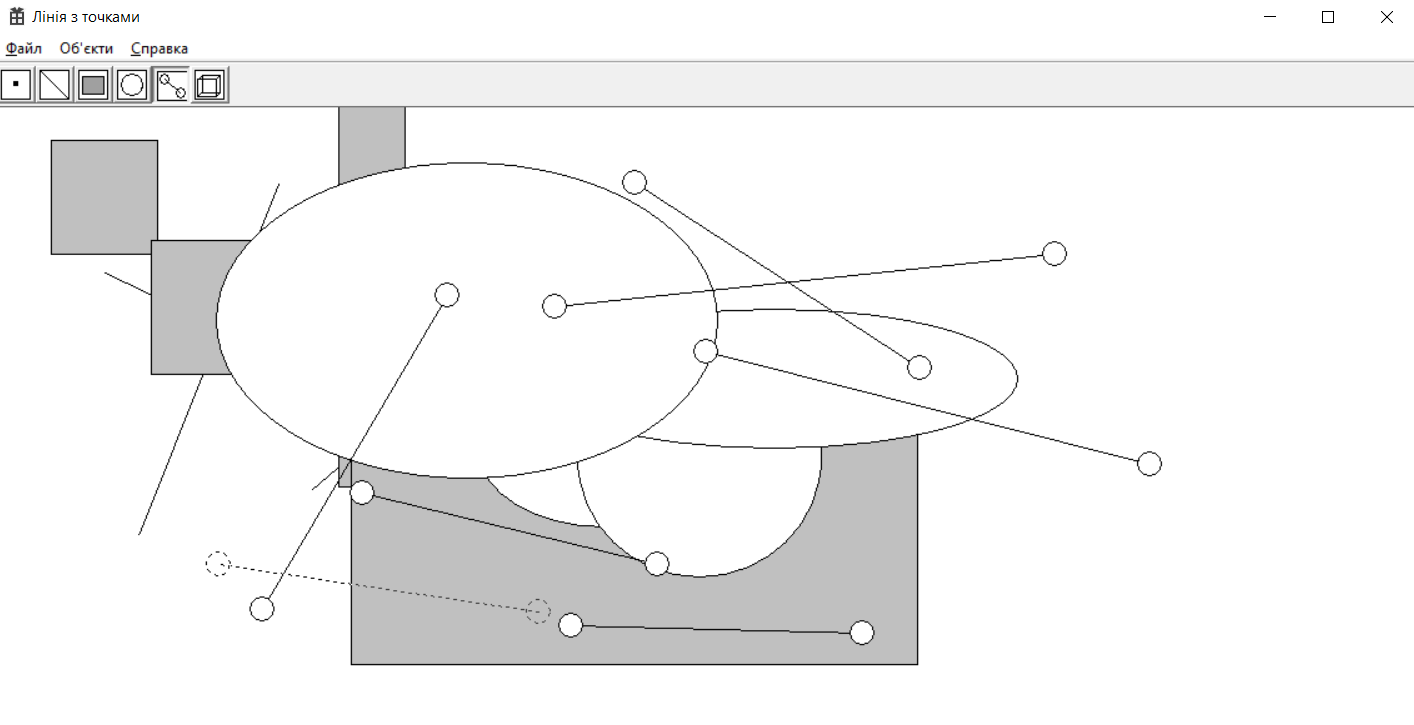
## Відображення прямокутника



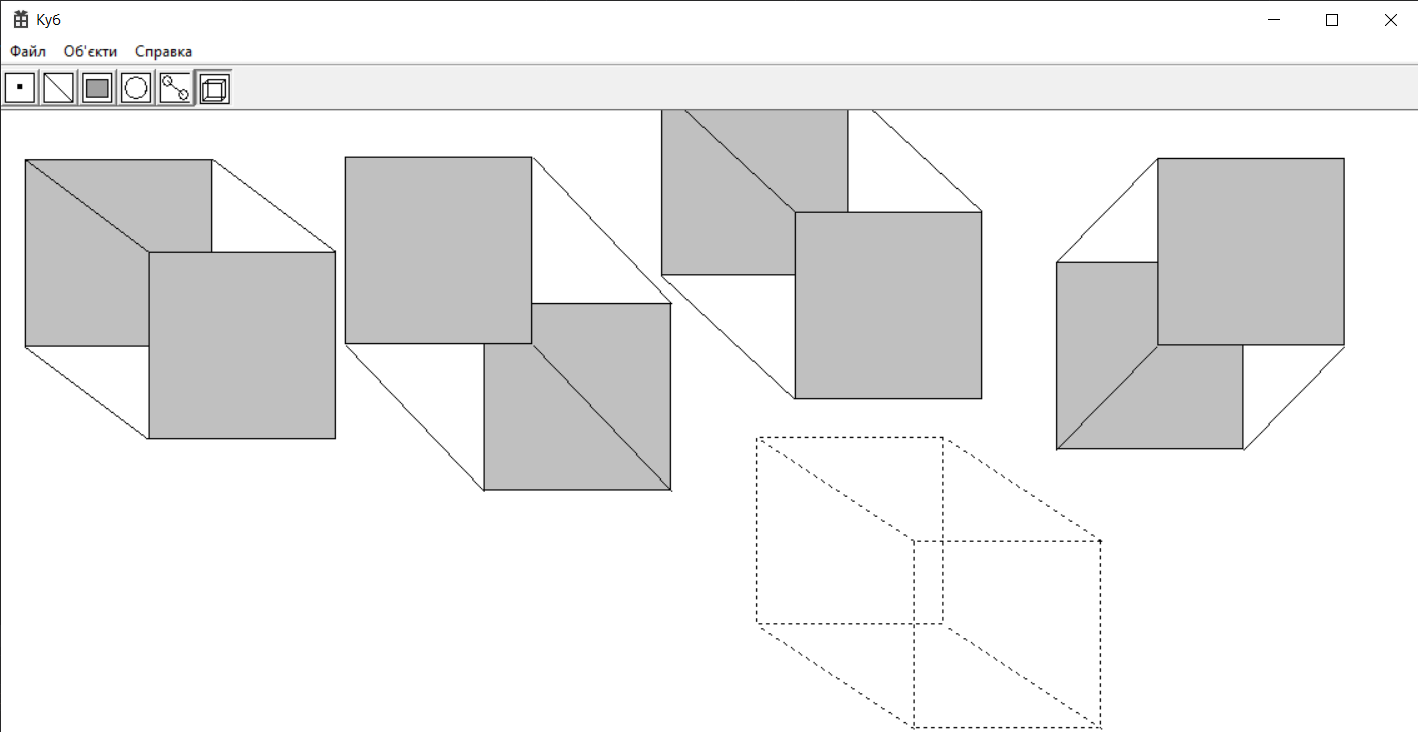
## Відображення еліпсу



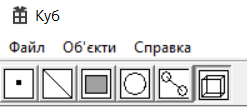
## Відображення лінії з точками



## Відображення куба



## Відображення вікна Toolbar



# Контрольні запитання

1. Що таке поліморфізм і як він використаний у цій лабораторній роботі?

**Поліморфі́зм** — концепція в програмуванні та теорії типів, в основі якої лежить використання одного єдиного інтерфейсу для різнотипних сутностей або у використанні однакового символу для маніпуляцій над даними різного типу. У нашій роботі ми використовуємо ми використовуємо його для створення двох нових фігур: *лінії з крапками* та *куба*.

1. Обробку яких повідомлень потрібно виконувати для вводу об’єктів?

Початок вводу об’єктів (при натисканні на пункт у меню «Об’єкти») та натискання на елементи **toolbar**.

1. Що таке абстрактний клас і скільки іх у цій програмі?

В ООП **абстрактний клас** – це базовий клас, від якого неможливо створити екземпляру. На практиці абстрактні класи використовують для одного із принципів ООП – поліморфізму. У такому класі можна описати (або не визначити) абстрактні методи та властивості. У даній програмі їх декілька, а саме – *Shape*, *PointShape*, *LineShape*, *RectShape*, *EllipseShape*, *PointLineShape*, *CubeShape*, *EmptyShape.*

1. Що таке множинне успадкування і як воно впливає на модульність?

**Множи́нне успадкування** — властивість деяких обʼєктно-орієнтованих мов програмування, у яких класи можуть успадкувати поведінку і властивості більш ніж від **одного суперкласу** (безпосереднього батьківського класу). Це відрізняється від простого спадкування, у випадку якого клас може мати тільки один суперклас. На мою думку, воно **негативно** впливає на модульність, тому що іноді важко відслідкувати таке успадкування, що сильно гальмує розробку програм.

1. Що таке ромбічне спадкування та які проблеми воно може спричинити?

**Ромбоподібне спадкування** - ситуація в об'єктно-орієнтованих мовах програмування з підтримкою множинного успадкування, коли два класи B і C успадковують від A, а клас D успадковує від обох класів B і C. При цій схемі успадкування може виникнути невизначеність: якщо об'єкт класу D викликає метод, визначений в класі A (і цей метод не був перевизначений в класі D), а класи B і C по-своєму перевизначити цей метод, то від якого класу його наслідувати: B або C?

1. Як побудувати діаграму класів засобами Visual Studio?

У вікні **Solution Explorer** клік правої кнопкою миші -> додати -> створити елемент -> схема класів та називаємо наш новий елемент. Далі перетягуємо у вікно, яке відкрилося, файли \*.h, \*.cpp із класами.

1. Як додати нові кнопки у Toolbar?

Створити новий **бітмап**, створити масив для кнопок відповідного розміру, додати значення елементами масиву та вказати к-сть кнопок у нашому бітмапі.

# Висновок

У ході виконання лабораторної роботи я виконав модернізацію коду для того, щоб зручно додавати нові типи об’єктів, також ознайомився із множинним та ромбічним успадкуванням.