**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 №3**

з дисципліни

«ООП»

на тему «Розробка інтерфейсу користувача 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 використовувати інкапсуляцію,

абстракцію типів, успадкування та поліморфізм на основі класів С++, запрограмувавши

графічний інтерфейс користувача.

Зміст

[Мета 2](#_Toc52813091)

[Завдання 3](#_Toc52813092)

[Варіанти завдань 3](#_Toc52813093)

[Вихідні тексти файлів 4](#_Toc52813094)

[Lab3.cpp 4](#_Toc52813095)

[Shape.cpp 8](#_Toc52813096)

[Shape\_editor.cpp 9](#_Toc52813097)

[Shape\_editor.h 13](#_Toc52813098)

[Editor.h 14](#_Toc52813099)

[Shape.h 14](#_Toc52813100)

[Toolbar.h 15](#_Toc52813101)

[Toolbar.cpp 15](#_Toc52813102)

[Діаграма класів 18](#_Toc52813103)

[Скріншоти програми 19](#_Toc52813104)

[Відображення точки 19](#_Toc52813105)

[Відображення прямої 19](#_Toc52813106)

[Відображення прямокутника 20](#_Toc52813107)

[Відображення еліпсу 20](#_Toc52813108)

[Відображення вікна Toolbar 21](#_Toc52813109)

[Контрольні запитання 21](#_Toc52813110)

[Висновок 22](#_Toc52813111)

Завдання**:**

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab3.
2. Написати вихідний текст програми згідно варіанту завдання.
3. Скомпілювати вихідний текст і отримати виконуваний файл програми.
4. Перевірити роботу програми. Налагодити програму.
5. Проаналізувати та прокоментувати результати та вихідний текст програми.
6. Оформити звіт.

# Варіанти завдань

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

**11 mod 4 = 3 –** 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

# Вихідні тексти файлів

## Lab3.cpp

// Lab3.cpp : Определяет точку входа для приложения.

//

#include "framework.h"

#include "Lab3.h"

#include "resource.h"

#include "shape\_editor.h" // import shape\_editor class

#include "toolbar.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

ShapeObjectsEditor editorShape; // 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\_LAB3, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LAB3));

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\_LAB3));

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

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

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_LAB3);

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,

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

editorShape.OnLBdown(hWnd);

break;

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

editorShape.OnLBup(hWnd);

break;

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

editorShape.OnMouseMove(hWnd);

break;

case WM\_INITMENUPOPUP:

editorShape.OnInitMenuPopup(hWnd, wParam);

break;

case WM\_CREATE: // Create toolbar

tool.OnCreate(hWnd);

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

editorShape.OnPaint(hWnd);

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case ID\_TOOL\_POINT:

case IDM\_POINT:

tool.OnPointPressed();

editorShape.StartPointEditor(); //початок вводу точкових об’єктів

SetWindowTextA(hWnd, "Крапка");

break;

case ID\_TOOL\_LINE:

case IDM\_LINE:

tool.OnLinePressed();

editorShape.StartLineEditor(); //початок вводу об’єктів-ліній

SetWindowTextA(hWnd, "Лінія");

break;

case ID\_TOOL\_RECT:

case IDM\_RECT:

tool.OnRectPressed();

editorShape.StartRectEditor(); //початок вводу прямокутників

SetWindowTextA(hWnd, "Прямокутник");

break;

case ID\_TOOL\_ELLIPSE:

case IDM\_ELLIPSE:

tool.OnEllipsePressed();

editorShape.StartEllipseEditor(); //початок вводу еліпсів

SetWindowTextA(hWnd, "Еліпс");

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"

// Get coords of points

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 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 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, xs1, ys1, xs2, ys2); // Create rectangle

SelectObject(hdc, hBrushOld);

DeleteObject(hBrush);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

}

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);

};

## Shape\_editor.cpp

#include "framework.h"

#include "shape\_editor.h"

// Variables

const int MY\_SHAPE\_ARRAY\_SIZE = 111;

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

int size = 0;

bool pressed;

// Constructor

ShapeObjectsEditor::ShapeObjectsEditor()

{

pse = new PointEditor;

}

// Destructor

ShapeObjectsEditor::~ShapeObjectsEditor()

{

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

}

// ShapeObjectsEditor functions

void ShapeObjectsEditor::StartPointEditor()

{

if (pse) delete pse;

pse = new PointEditor;

}

void ShapeObjectsEditor::StartLineEditor()

{

if (pse) delete pse;

pse = new LineEditor;

}

void ShapeObjectsEditor::StartRectEditor()

{

if (pse) delete pse;

pse = new RectEditor;

}

void ShapeObjectsEditor::StartEllipseEditor()

{

if (pse) delete pse;

pse = new EllipseEditor;

}

void ShapeObjectsEditor::OnLBdown(HWND hWnd) {

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

}

void ShapeObjectsEditor::OnLBup(HWND hWnd) {

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

}

void ShapeObjectsEditor::OnMouseMove(HWND hWnd) {

if (pse && pressed) pse->OnMouseMove(hWnd);

}

void ShapeObjectsEditor::OnPaint(HWND hWnd) {

ShapeEditor \*draw = new ShapeEditor;

draw->OnPaint(hWnd);

}

void ShapeObjectsEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParams) {

if (pse)

pse->OnInitMenuPopup(hWnd, wParams);

}

// ShapeEditor Functions

void ShapeEditor::OnLBdown(HWND hWnd) {

pressed = true;

POINT pt;

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x1 = x2 = pt.x;

y1 = y2 = pt.y;

}

void ShapeEditor::OnLBup(HWND hWnd) {

POINT pt;

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x2 = pt.x;

y2 = pt.y;

pressed = false;

}

void ShapeEditor::OnMouseMove(HWND hWnd) {}

void ShapeEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParams) {};

void ShapeEditor::OnPaint(HWND hWnd) {

PAINTSTRUCT ps;

HDC hdc;

hdc = BeginPaint(hWnd, &ps);

for (int i = 0; i < size; i++)

if (pcshape[i])

pcshape[i]->Show(hdc);

EndPaint(hWnd, &ps);

}

// Point:

void PointEditor::OnLBdown(HWND hWnd) {

\_\_super::OnLBdown(hWnd); // Calling a base-class implementation

}

void PointEditor::OnLBup(HWND hWnd) {

\_\_super::OnLBup(hWnd); // Calling a base-class implementation

PointShape\* Point = new PointShape;

Point->Set(x1, y1, x2, y2);

pcshape[size] = Point;

size++;

InvalidateRect(hWnd, NULL, TRUE);

}

void PointEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParams) {

HMENU hMenu, hSubMenu;

hMenu = GetMenu(hWnd);

hSubMenu = GetSubMenu(hMenu, 1);

if ((HMENU)wParams == hSubMenu) {

CheckMenuItem(hSubMenu, IDM\_POINT, MF\_CHECKED);

CheckMenuItem(hSubMenu, IDM\_LINE, MF\_UNCHECKED);

CheckMenuItem(hSubMenu, IDM\_RECT, MF\_UNCHECKED);

CheckMenuItem(hSubMenu, IDM\_ELLIPSE, MF\_UNCHECKED);

}

}

// Line:

void LineEditor::OnLBdown(HWND hWnd) {

\_\_super::OnLBdown(hWnd); // Calling a base-class implementation

}

void LineEditor::OnLBup(HWND hWnd) {

\_\_super::OnLBup(hWnd); // Calling a base-class implementation

LineShape\* Line = new LineShape;

Line->Set(x1, y1, x2, y2);

pcshape[size] = Line;

size++;

InvalidateRect(hWnd, NULL, TRUE);

}

void LineEditor::OnMouseMove(HWND hWnd) {

POINT pt;

HPEN hPen, hPenOld;

HDC hdc = GetDC(hWnd);

SetROP2(hdc, R2\_NOTXORPEN);

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

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

MoveToEx(hdc, x1, y1, NULL);

LineTo(hdc, x2, y2);

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x2 = pt.x;

y2 = pt.y;

MoveToEx(hdc, x1, y1, NULL);

LineTo(hdc, x2, y2);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

ReleaseDC(hWnd, hdc);

}

void LineEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParams) {

HMENU hMenu, hSubMenu;

hMenu = GetMenu(hWnd);

hSubMenu = GetSubMenu(hMenu, 1);

if ((HMENU)wParams == hSubMenu) {

CheckMenuItem(hSubMenu, IDM\_POINT, MF\_UNCHECKED);

CheckMenuItem(hSubMenu, IDM\_LINE, MF\_CHECKED);

CheckMenuItem(hSubMenu, IDM\_RECT, MF\_UNCHECKED);

CheckMenuItem(hSubMenu, IDM\_ELLIPSE, MF\_UNCHECKED);

}

}

// Rect:

void RectEditor::OnLBdown(HWND hWnd) {

\_\_super::OnLBdown(hWnd); // Calling a base-class implementation

}

void RectEditor::OnLBup(HWND hWnd) {

\_\_super::OnLBup(hWnd); // Calling a base-class implementation

RectShape\* Rect = new RectShape;

//Rect->Set(x1, y1, x2, y2);

Rect->Set(2 \* x1 - x2, 2 \* y1 - y2, x2, y2);

pcshape[size] = Rect;

size++;

InvalidateRect(hWnd, NULL, TRUE);

}

void RectEditor::OnMouseMove(HWND hWnd) {

POINT pt;

HPEN hPen, hPenOld;

HDC hdc = GetDC(hWnd);

SetROP2(hdc, R2\_NOTXORPEN);

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

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

//Rectangle(hdc, x1, y1, x2, y2);

Rectangle(hdc, 2 \* x1 - x2, 2 \* y1 - y2, x2, y2);

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x2 = pt.x;

y2 = pt.y;

//Rectangle(hdc, x1, y1, x2, y2);

Rectangle(hdc, 2 \* x1 - x2, 2 \* y1 - y2, x2, y2);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

ReleaseDC(hWnd, hdc);

}

void RectEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParams) {

HMENU hMenu, hSubMenu;

hMenu = GetMenu(hWnd);

hSubMenu = GetSubMenu(hMenu, 1);

if ((HMENU)wParams == hSubMenu) {

CheckMenuItem(hSubMenu, IDM\_POINT, MF\_UNCHECKED);

CheckMenuItem(hSubMenu, IDM\_LINE, MF\_UNCHECKED);

CheckMenuItem(hSubMenu, IDM\_RECT, MF\_CHECKED);

CheckMenuItem(hSubMenu, IDM\_ELLIPSE, MF\_UNCHECKED);

}

}

// Ellipse:

void EllipseEditor::OnLBdown(HWND hWnd) {

\_\_super::OnLBdown(hWnd); // Calling a base-class implementation

}

void EllipseEditor::OnLBup(HWND hWnd) {

\_\_super::OnLBup(hWnd);

EllipseShape\* Ellipse = new EllipseShape;

Ellipse->Set(x1, y1, x2, y2);

pcshape[size] = Ellipse;

size++;

InvalidateRect(hWnd, NULL, TRUE);

}

void EllipseEditor::OnMouseMove(HWND hWnd) {

POINT pt;

HPEN hPen, hPenOld;

HDC hdc = GetDC(hWnd);

SetROP2(hdc, R2\_NOTXORPEN);

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

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

Arc(hdc, x1, y1, x2, y2, 0, 0, 0, 0);

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x2 = pt.x;

y2 = pt.y;

Arc(hdc, x1, y1, x2, y2, 0, 0, 0, 0);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

ReleaseDC(hWnd, hdc);

}

void EllipseEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParams) {

HMENU hMenu, hSubMenu;

hMenu = GetMenu(hWnd);

hSubMenu = GetSubMenu(hMenu, 1);

if ((HMENU)wParams == hSubMenu) {

CheckMenuItem(hSubMenu, IDM\_POINT, MF\_UNCHECKED);

CheckMenuItem(hSubMenu, IDM\_LINE, MF\_UNCHECKED);

CheckMenuItem(hSubMenu, IDM\_RECT, MF\_UNCHECKED);

CheckMenuItem(hSubMenu, IDM\_ELLIPSE, MF\_CHECKED);

}

}

## Shape\_editor.h

#pragma once

#include "editor.h"

#include "shape.h"

#include "resource.h"

class ShapeEditor : public Editor // Shape editor class for figures

{

protected:

int x1, x2, y1, y2;

public:

void OnLBdown(HWND);

void OnLBup(HWND);

void OnMouseMove(HWND);

void OnPaint(HWND);

virtual void OnInitMenuPopup(HWND, WPARAM);

};

class ShapeObjectsEditor // Shape editor class for figure objects

{

private:

ShapeEditor \*pse;

public:

ShapeObjectsEditor(void);

~ShapeObjectsEditor();

void StartPointEditor();

void StartLineEditor();

void StartRectEditor();

void StartEllipseEditor();

void OnLBdown(HWND);

void OnLBup(HWND);

void OnMouseMove(HWND);

void OnPaint(HWND);

void OnInitMenuPopup(HWND, WPARAM);

};

class PointEditor : public ShapeEditor

{

public:

void OnLBdown(HWND);

void OnLBup(HWND);

void OnInitMenuPopup(HWND, WPARAM);

};

class LineEditor : public ShapeEditor

{

public:

void OnLBdown(HWND);

void OnLBup(HWND);

void OnMouseMove(HWND);

void OnInitMenuPopup(HWND, WPARAM);

};

class RectEditor : public ShapeEditor

{

public:

void OnLBdown(HWND);

void OnLBup(HWND);

void OnMouseMove(HWND);

void OnInitMenuPopup(HWND, WPARAM);

};

class EllipseEditor : public ShapeEditor

{

public:

void OnLBdown(HWND);

void OnLBup(HWND);

void OnMouseMove(HWND);

void OnInitMenuPopup(HWND, WPARAM);

};

## Editor.h

class Editor // Main class

{

public:

virtual void OnLBdown(HWND) = 0;

virtual void OnLBup(HWND) = 0;

virtual void OnMouseMove(HWND) = 0;

virtual void OnPaint(HWND) = 0;

};

## Shape.h

class Shape // Shape class for figures

{

protected:

long xs1, ys1, xs2, ys2;

public:

void Set(long x1, long y1, long x2, long y2);

virtual void Show(HDC) = 0;

};

class PointShape : public Shape

{

public:

void Show(HDC);

};

class LineShape : public Shape

{

public:

void Show(HDC);

};

class RectShape : public Shape

{

public:

void Show(HDC);

};

class EllipseShape : public Shape

{

public:

void Show(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 IDC\_MY\_TOOLBAR 32811

// Declare Toolbar class

class Toolbar {

public:

void OnCreate(HWND hWnd);

void OnSize(HWND);

void OnPointPressed();

void OnLinePressed();

void OnRectPressed();

void OnEllipsePressed();

void OnNotify(HWND, LPARAM);

void OffPressed(int);

};

## Toolbar.cpp

#include "framework.h"

#include "resource1.h"

#include "toolbar.h"

// Required variables

HWND hwndToolBar = NULL;

extern HINSTANCE hInst;

bool point, line, rect, ellipse = 0;

int whatPressed = NULL;

// Create toolbar listener

void Toolbar::OnCreate(HWND hWnd)

{

TBBUTTON tbb[5];

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\_SEP; //роздільник груп кнопок

tbb[4].idCommand = 4;

hwndToolBar = CreateToolbarEx(hWnd,

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

IDC\_MY\_TOOLBAR,

4, //кількість зображень у BITMAP

hInst,

IDB\_BITMAP1, //ID ресурсу BITMAP

tbb,

5, //кількість кнопок (разом з роздільником)

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

sizeof(TBBUTTON));

}

// Change toolbar listener

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);

}

}

// Unpress toolbar button

void Toolbar::OffPressed(int id) {

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

whatPressed = id;

}

// Pressed toolbar point button

void Toolbar::OnPointPressed() {

point = !point;

line = rect = ellipse = 0;

OffPressed(ID\_TOOL\_POINT);

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

}

// Pressed toolbar line button

void Toolbar::OnLinePressed() {

line = !line;

point = rect = ellipse = 0;

OffPressed(ID\_TOOL\_LINE);

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

}

// Pressed toolbar rectangle button

void Toolbar::OnRectPressed() {

rect = !rect;

point = line = ellipse = 0;

OffPressed(ID\_TOOL\_RECT);

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

}

// Pressed toolbar ellipse button

void Toolbar::OnEllipsePressed() {

ellipse = !ellipse;

point = line = rect = 0;

OffPressed(ID\_TOOL\_ELLIPSE);

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

}

// Notify current pressed button

void Toolbar::OnNotify(HWND hWnd, LPARAM lParam)

{

LPNMHDR pnmh = (LPNMHDR)lParam;

LPCSTR pText;

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

{

LPTOOLTIPTEXT lpttt = (LPTOOLTIPTEXT)lParam;

switch (lpttt->hdr.idFrom)

{

case ID\_TOOL\_POINT:

pText = "Крапка";

break;

case ID\_TOOL\_LINE:

pText = "Лінія";

break;

case ID\_TOOL\_RECT:

pText = "Прямокутник";

break;

case ID\_TOOL\_ELLIPSE:

pText = "Еліпс";

break;

default: pText = "Щось невідоме";

}

lstrcpy(lpttt->szText, pText);

}

}

# Діаграма класів

public:

* OnCreate
* OnSize
* OnNotify
* offPressed

Toolbar

* OnPointPressed
* OnLinePressed
* OnRectPressed
* OnEllipsePressed

Похідні класи

Похідні класи

Базовий клас

Editor

public:

* OnLBdown = 0
* OnLBup = 0
* OnMouseMove = 0
* OnPaint = 0

ShapeEditor

protected:

* x1, x2, y1, y2

public:

* OnLBdown
* OnLBup
* OnMouseMove
* OnPaint
* OnInitMenuPopup

ShapeObjectsEditor

private:

* pse

public:

* ShapeObjectsEditor
* ~ShapeObjectsEditor
* StartPointEditor
* StartLineEditor
* StartRectEditor
* StartEllipseEditor
* OnLBdown
* OnLBup
* OnMouseMove
* OnPaint
* OnInitMenuPopup

PointEditor

public:

* OnLBdown
* OnLBup
* OnInitMenuPopup

LineEditor

public:

* OnLBdown
* OnLBup
* OnMouseMove
* OnInitMenuPopup

RectEditor

public:

* OnLBdown
* OnLBup
* OnMouseMove
* OnInitMenuPopup

EllipseEditor

public:

* OnLBdown
* OnLBup
* OnMouseMove
* OnInitMenuPopup

Shape

protected:

* xs1, ys1, xs2, ys2

public:

* Set
* Show = 0

PointShape

public:

* Show

LineShape

public:

* Show

RectShape

public:

* Show

EllipseShape

public:

* Show

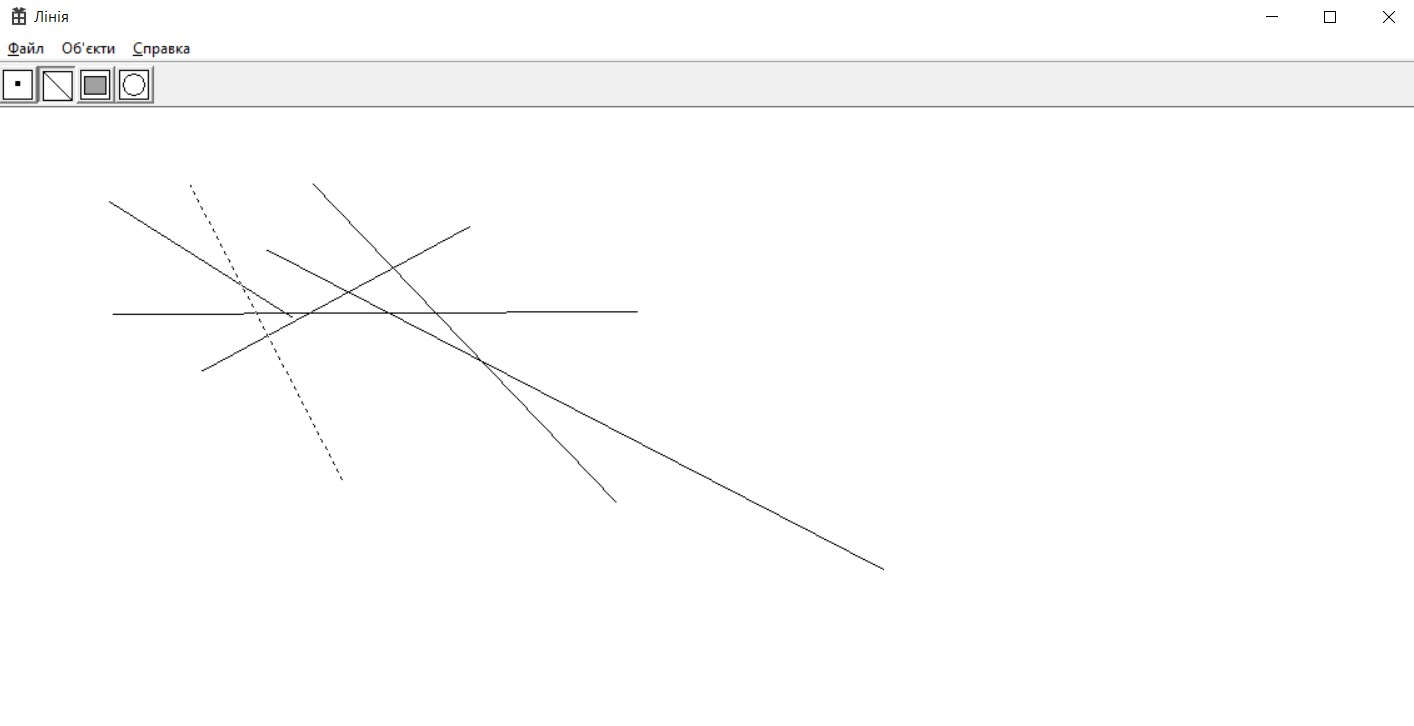
# Скріншоти програми

## Відображення точки

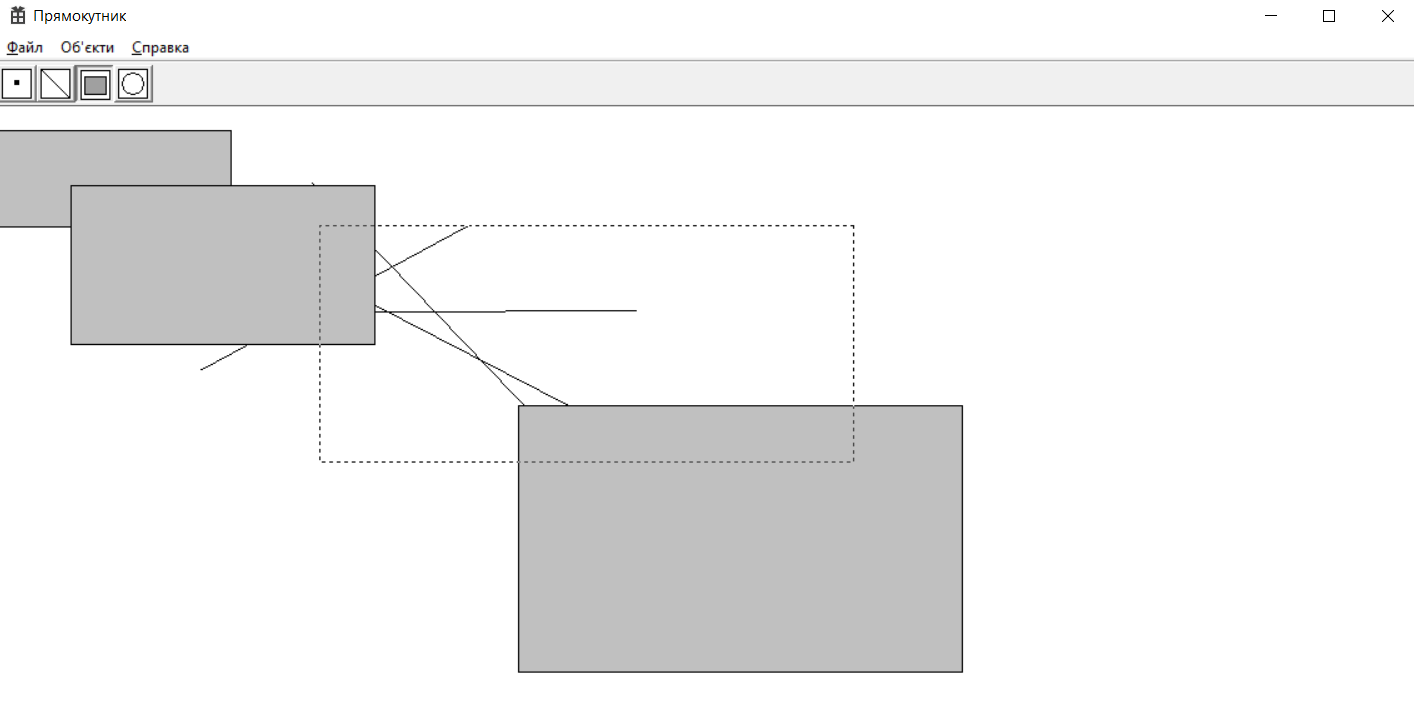


Точки є малими, тому я збільшив для комфортності скріншот

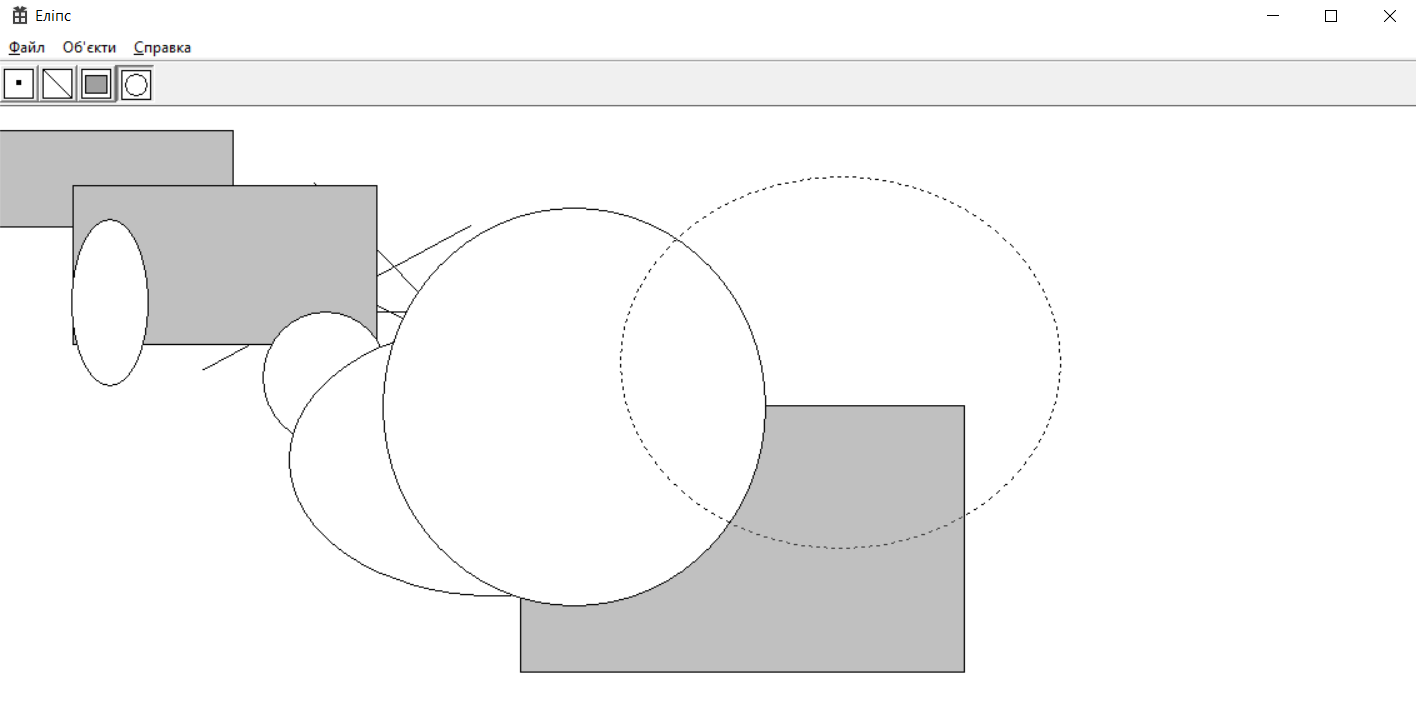
## Відображення прямої



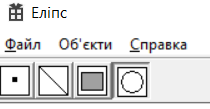
## Відображення прямокутника



## Відображення еліпсу



## Відображення вікна Toolbar



# Контрольні запитання

1. Обробку яких повідомлень потрібно виконувати у програмі Лаб3?

У даній програмі потрібно відслідковувати початок вводу об’єктів при натисканні на пункт у меню «Об’єкти, натискання та відпускання лівої кнопки миші, рух самої миші, натискання на елементи **Toolbar**, а також методи **OnSize()**, **OnCreate()**, **OnNotify()**, **OnInitMenuPopup()**.

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

В ООП **абстрактний клас** – це базовий клас, від якого не можна 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 **три** штуки, а саме – *Editor*, *Shape*, *ShapeEditor*.

1. Як забезпечити відповідність пунктів меню і кнопок Toolbar?

За допомогою призначення значення toolbar\_array[index].idCommand = button\_id; , де toolbar\_array – масив типу *TBBUTTON,* index – індекс пункту, button\_id – “айді” необхідної кнопки

1. Як запрограмувати показ власних зображень на кнопках Toolbar?

Під час створення **Toolbar** за допомогою методу **CreateToolbarEx()** задаються такі параметри як *ID ресурсу BITMAP, кількість зображень у BITMAP, а також* розмір та кількість кнопок.

1. Як створити власні зображення кнопок і де вони зберігаються?

Для цього можна створити *ресурс-bitmap*, який буде містити потрібні зображення, які потім можна відобразити на кнопках. Щоб створити такий ресурс, треба у вікні Solution Explorer клікнути на файлі ресурсів, потім додати у ресурси Bitmap. Потрібно вказати розміри цього BITMAP у пікселах. Зображення зберігаються у ресурсах у форматі .rc.

1. Як запрограмувати текст підказок (tooltips)?

Для цього потрібно при створенні **Toolbar** за допомогою функції *CreateToolbarEx()* вказати стиль **TBSTYLE\_TOOLTIPS.** Після цього потрібно зробити обробник повідомлення **WM\_NOTIFY** та запрогромувати його, а потім викликати у *Lab3.cpp*

# Висновок

У ході виконання лабораторної роботи ознайомився та застосував на практиці інкапсуляцію та абстракцію типів, успадкування та поліморфізм на основі класів С++, а також створення та запрограмування графічного інтерфейсу користувача.