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:
«Об’єктно-орієнтоване програмування»

Виконав:
Баран Павло Ю.
студент групи ІП-03
Номер у списку групи: 1

Перевірив:
Порєв В. М.

2021

Розробка графічного редактора об’єктів на C++

Варіант:

1. Статичний масив Shape *pcshape[N];
2. "Гумовий" слід при вводі об’єктів: суцільна лінія червоного кольору.
3. Ввід прямокутника: від центру до одного з кутів.
4. Відображення прямокутника: чорний контур з світло-зеленим заповненням.
5. Ввід еліпсу: по двом протилежним кутам охоплюючого прямокутника.
6. Відображення еліпсу: чорний контур з білим заповненням.
7. Позначка поточного типу об’єкту, що вводиться, в заголовку вікна.

Код програми:

Lab2.cpp

```
#include "framework.h"
#include "lab2.h"
#include "shape_editor.h"
#include "toolbar.h"

#define MAX_LOADSTRING 100

// Глобальные переменные:
ShapeObjectsEditor Editor;
Toolbar CreateToolbar;
HINSTANCE hInst;                                // текущий экземпляр
WCHAR szTitle[MAX_LOADSTRING];                  // Текст строки заголовка
WCHAR szWindowClass[MAX_LOADSTRING];            // имя класса главного окна

// 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_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
//
//   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:
        Editor.OnLBdown(hWnd);
        break;
    case WM_LBUTTONUP:
        Editor.OnLBup(hWnd);
        break;
    case WM_MOUSEMOVE:
        Editor.OnMouseMove(hWnd);
        break;
    case WM_CREATE:
        CreateToolbar.OnCreate(hWnd);
        break;
    case WM_NOTIFY:
        CreateToolbar.OnNotify(hWnd, lParam, wParam);
        break;
    case WM_PAINT:
        Editor.OnPaint(hWnd);
        break;
    case WM_COMMAND:
    {
        int wmId = LOWORD(wParam);
        // Разобрать выбор в меню:
        switch (wmId)
        {
        case IDM_POINT:
            SetWindowTextW(hWnd, L"Режим вводу точок");
            Editor.StartPointEditor();
            break;
        case IDM_LINE:
            SetWindowTextW(hWnd, L"Режим вводу ліній");
            Editor.StartLineEditor();
```

```
            break;
        case IDM_RECTANGLE:
            SetWindowTextW(hWnd, L"Режим вводу прямокутників");
            Editor.StartRectEditor();
            break;
        case IDM_ELLIPSE:
            SetWindowTextW(hWnd, L"Режим вводу еліпсів");
            Editor.StartEllipseEditor();
            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_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(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"

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));
}


void LineShape::Show(HDC hdc)
{
	HBRUSH hBrush, hBrushOld;
	hBrush = (HBRUSH)CreateSolidBrush(RGB(0, 0, 0)); //створюється пензль
	hBrushOld = (HBRUSH)SelectObject(hdc, hBrush);
	MoveToEx(hdc, xs1, ys1, NULL);
	LineTo(hdc, xs2, ys2);
	SelectObject(hdc, hBrushOld); //відновлюється пензль-попередник
	DeleteObject(hBrush);
}



void RectShape::Show(HDC hdc)
{
	HBRUSH hBrush, hBrushOld;
	hBrush = (HBRUSH)CreateSolidBrush(RGB(0, 255, 0)); //створюється пензль
	hBrushOld = (HBRUSH)SelectObject(hdc, hBrush);
	Rectangle(hdc, xs1, ys1, xs2, ys2);
	SelectObject(hdc, hBrushOld); //відновлюється пензль-попередник
	DeleteObject(hBrush);
}

void EllipseShape::Show(HDC hdc)
{
```

```
	HBRUSH hBrush, hBrushOld;
	hBrush = (HBRUSH)CreateSolidBrush(RGB(255, 255, 255)); //створюється пензль
	hBrushOld = (HBRUSH)SelectObject(hdc, hBrush);
	Ellipse(hdc, xs1, ys1, xs2, ys2);
	SelectObject(hdc, hBrushOld); //відновлюється пензль-попередник
	DeleteObject(hBrush);
}
```

shape.h

```
#pragma once

class Shape {
protected:
	long xs1, ys1, xs2, ys2;
public:
	Shape() : xs1{ 0 }, ys1{ 0 }, xs2{ 0 }, ys2{ 0 } {}
	void Set(long x1, long y1, long x2, long y2);
	virtual void Show(HDC hdc) = 0;
};

class PointShape : public Shape
{
	void Show(HDC);
};

class LineShape : public Shape
{
	void Show(HDC);
};

class RectShape : public Shape
{
	void Show(HDC);
};

class EllipseShape : public Shape
{
	void Show(HDC);
};
```

editor.h

```
#pragma once

class Editor
{
public:
	virtual void OnLBdown(HWND hWnd) = 0;
	virtual void OnLBup(HWND hWnd) = 0;
	virtual void OnMouseMove(HWND hWnd) = 0;
	virtual void OnPaint(HWND hWnd) = 0;
};
```

editor.cpp

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

class ShapeEditor : public Editor
{
protected:
	int x1, y1, x2, y2;
public:
	void OnLBdown(HWND);
	void OnLBup(HWND, COLORREF, BOOL);
	void OnMouseMove(HWND);
	void OnPaint(HWND);
};
```

shape_editor.h

```
#pragma once
#include "shape.h"
#include "editor.h"

class ShapeEditor : public Editor
{
protected:
	int x1, y1, x2, y2;
public:
	ShapeEditor() : x1{0}, y1{0}, x2{0}, y2{0} {}
	void OnLBdown(HWND);
	void OnLBup(HWND);
	void OnMouseMove(HWND);
	void OnPaint(HWND);
	virtual void OnInitMenuPopup(HWND); //додатковий інтерфейсний метод
```

```
};

class ShapeObjectsEditor
{
private:
	ShapeEditor *pse = NULL;
public:
	ShapeObjectsEditor();
	~ShapeObjectsEditor();
	void StartPointEditor(HWND);
	void StartLineEditor(HWND);
	void StartRectEditor(HWND);
	void StartEllipseEditor(HWND);
	void OnLBdown(HWND);
	void OnLBup(HWND);
	void OnMouseMove(HWND);
	void OnPaint(HWND);
	void OnInitMenuPopup(HWND);
};

class PointEditor : public ShapeEditor
{
public:
	void OnLBdown(HWND);
	void OnLBup(HWND);
	void OnInitMenuPopup(HWND);
};

class LineEditor : public ShapeEditor
{
public:
	void OnLBdown(HWND);
	void OnMouseMove(HWND);
	void OnLBup(HWND);
	void OnInitMenuPopup(HWND);
};

class RectEditor : public ShapeEditor
{
public:
	void OnLBdown(HWND);
	void OnMouseMove(HWND);
	void OnLBup(HWND);
	void OnInitMenuPopup(HWND);
};
```

```
class EllipseEditor : public ShapeEditor
{
public:
	void OnLBdown(HWND);
	void OnMouseMove(HWND);
	void OnLBup(HWND);
	void OnInitMenuPopup(HWND);
};
```

shape_editor.cpp

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

Shape* pcshape[101];
static int arrayCounter = 0;
static int check = 0;

ShapeObjectsEditor::ShapeObjectsEditor()
{
	pse = new PointEditor;
}

ShapeObjectsEditor::~ShapeObjectsEditor()
{
	for (int i = 0; i < arrayCounter; i++)
		delete pcshape[i];
}

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 && check)
		pse->OnMouseMove(hWnd);
}

void ShapeObjectsEditor::OnPaint(HWND hWnd) {
	ShapeEditor *draw = new ShapeEditor;
	draw->OnPaint(hWnd);
}

//ShapeEditor

void ShapeEditor::OnLBdown(HWND hWnd)
{
	POINT pt;
	GetCursorPos(&pt);
	ScreenToClient(hWnd, &pt);
	x1 = x2 = pt.x;
	y1 = y2 = pt.y;
	check = 1;
}

void ShapeEditor::OnLBup(HWND hWnd) {
	POINT pt;
	GetCursorPos(&pt);
	ScreenToClient(hWnd, &pt);
	x2 = pt.x;
	y2 = pt.y;
```

```
	check = 0;
}

void ShapeEditor::OnPaint(HWND hWnd) {
	PAINTSTRUCT ps;
	HDC hdc;
	hdc = BeginPaint(hWnd, &ps);
	for (int i = 0; i < arrayCounter; i++) {
		if (pcshape[i])
			pcshape[i]->Show(hdc);
	}
	EndPaint(hWnd, &ps);
}

void ShapeEditor::OnMouseMove(HWND hWnd) {}

//Point

void PointEditor::OnLBdown(HWND hWnd) {
	__super::OnLBdown(hWnd);
}

void PointEditor::OnLBup(HWND hWnd) {
	__super::OnLBup(hWnd);
	PointShape* Point = new PointShape;
	Point->Set(x1, y1, x2, y2);
	pcshape[arrayCounter] = Point;
	arrayCounter++;
	InvalidateRect(hWnd, NULL, TRUE);
}

//Line

void LineEditor::OnLBdown(HWND hWnd) {
	__super::OnLBdown(hWnd);
}

void LineEditor::OnLBup(HWND hWnd) {
	__super::OnLBup(hWnd);
	LineShape* Line = new LineShape;
	Line->Set(x1, y1, x2, y2);
	pcshape[arrayCounter] = Line;
	arrayCounter++;
	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_SOLID, 1, RGB(255, 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);
}

//Rect

void RectEditor::OnLBdown(HWND hWnd) {
	__super::OnLBdown(hWnd);
}

void RectEditor::OnLBup(HWND hWnd) {
	__super::OnLBup(hWnd);
	RectShape* Rect = new RectShape;
	Rect->Set(x1, y1, x2, y2);
	pcshape[arrayCounter] = Rect;
	arrayCounter++;
	InvalidateRect(hWnd, NULL, TRUE);
}

void RectEditor::OnMouseMove(HWND hWnd)
{
	POINT pt;
	HPEN hPen, hPenOld;
	HDC hdc = GetDC(hWnd);
	int xc1, yc1;
	xc1 = 2 * x1 - x2;
	yc1 = 2 * y1 - y2;
	SetROP2(hdc, R2_NOTXORPEN);
	hPen = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
	hPenOld = (HPEN)SelectObject(hdc, hPen);
```

```
		MoveToEx(hdc, xc1, yc1, NULL);
		LineTo(hdc, xc1, y2);
		LineTo(hdc, x2, y2);
		LineTo(hdc, x2, yc1);
		LineTo(hdc, xc1, yc1);
		GetCursorPos(&pt);
		ScreenToClient(hWnd, &pt);
		x2 = pt.x;
		y2 = pt.y;
		xc1 = 2 * x1 - x2;
		yc1 = 2 * y1 - y2;
		MoveToEx(hdc, xc1, yc1, NULL);
		LineTo(hdc, xc1, y2);
		LineTo(hdc, x2, y2);
		LineTo(hdc, x2, yc1);
		LineTo(hdc, xc1, yc1);
		SelectObject(hdc, hPenOld);
		DeleteObject(hPen);
		ReleaseDC(hWnd, hdc);
}

// Ellipse

void EllipseEditor::OnLBdown(HWND hWnd) {
		__super::OnLBdown(hWnd);
}


void EllipseEditor::OnLBup(HWND hWnd) {
		__super::OnLBup(hWnd);
		EllipseShape* Ellipse = new EllipseShape;
		Ellipse->Set(x1, y1, x2, y2);
		pcshape[arrayCounter] = Ellipse;
		arrayCounter++;
		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_SOLID, 1, RGB(255, 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);
}
```

shape_editor.h

```
#pragma once
#include "editor.h"
#include "shape.h"

class ShapeEditor : public Editor
{
protected:
        int x1, y1, x2, y2;
public:
        ShapeEditor() : x1{ 0 }, y1{ 0 }, x2{ 0 }, y2{ 0 } {}
        void OnLBdown(HWND);
        void OnLBup(HWND);
        void OnMouseMove(HWND);
        void OnPaint(HWND);
};

class ShapeObjectsEditor
{
private:
        ShapeEditor *pse = NULL;
public:
        ShapeObjectsEditor();
        ~ShapeObjectsEditor();
        void StartPointEditor();
        void StartLineEditor();
        void StartRectEditor();
        void StartEllipseEditor();
        void OnLBdown(HWND);
        void OnLBup(HWND);
        void OnMouseMove(HWND);
        void OnPaint(HWND);
};

class PointEditor : public ShapeEditor
```

```
{
public:
	void OnLBdown(HWND);
	void OnLBup(HWND);
};

class LineEditor : public ShapeEditor
{
public:
	void OnLBdown(HWND);
	void OnLBup(HWND);
	void OnMouseMove(HWND);
};

class RectEditor : public ShapeEditor
{
public:
	void OnLBdown(HWND);
	void OnLBup(HWND);
	void OnMouseMove(HWND);
};

class EllipseEditor : public ShapeEditor
{
public:
	void OnLBdown(HWND);
	void OnLBup(HWND);
	void OnMouseMove(HWND);
};
```

toolbar.cpp

```
#include "toolbar.h"
#include "lab2.h"

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::OnCreate(HWND hWndmother)
{
   HINSTANCE hInst = (HINSTANCE)GetWindowLong(hWndmother, GWL_HINSTANCE);
   TBBUTTON tbb[4];
   ZeroMemory(tbb, sizeof(tbb));
   tbb[0].iBitmap = 0;
   tbb[0].fsState = TBSTATE_ENABLED;
   tbb[0].fsStyle = TBSTYLE_BUTTON;
   tbb[0].idCommand = IDM_POINT;

   tbb[1].iBitmap = 1;
   tbb[1].fsState = TBSTATE_ENABLED;
   tbb[1].fsStyle = TBSTYLE_BUTTON;
   tbb[1].idCommand = IDM_LINE;

   tbb[2].iBitmap = 2;
   tbb[2].fsState = TBSTATE_ENABLED;
   tbb[2].fsStyle = TBSTYLE_BUTTON;
   tbb[2].idCommand = IDM_RECTANGLE;

   tbb[3].iBitmap = 3;
   tbb[3].fsState = TBSTATE_ENABLED;
   tbb[3].fsStyle = TBSTYLE_BUTTON;
   tbb[3].idCommand = IDM_ELLIPSE;
   SendMessage(hWndToolBar, TB_ADDBUTTONS, 4, (LPARAM)&tbb);
   hWndToolBar = CreateToolbarEx(hWndmother,
      WS_CHILD | WS_VISIBLE | WS_BORDER | WS_CLIPSIBLINGS | CCS_TOP |
TBSTYLE_TOOLTIPS, IDC_TOOLBAR,
      4, hInst, IDB_TOOLBAR,
      tbb,
      4,
      24, 24, 24, 24,
      sizeof(TBBUTTON));
}

void Toolbar::OnNotify(HWND hWnd, LPARAM lParam, WPARAM wParam)
{
   LPNMHDR pnmh = (LPNMHDR)lParam;
   if (pnmh->code == TTN_NEEDTEXT)
   {
      LPTOOLTIPTEXT lpttt = (LPTOOLTIPTEXT)lParam;
      switch (lpttt->hdr.idFrom)
      {
      case IDM_POINT:
         lstrcpy(lpttt->szText, L"Точка");
```

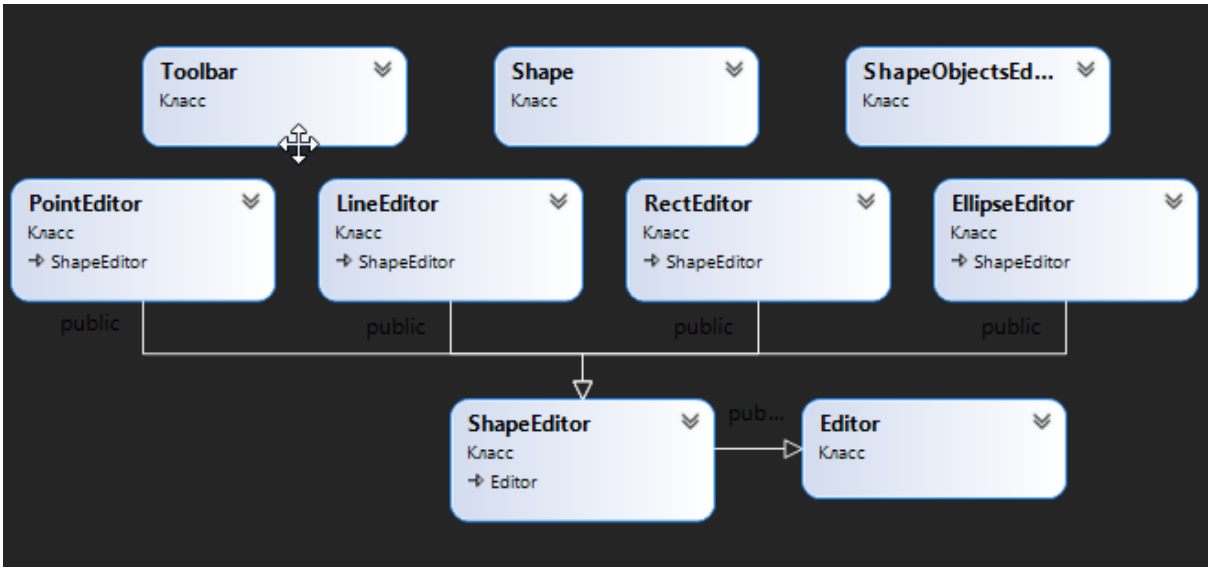
```
            break;
        case IDM_LINE:
            lstrcpy(lpttt->szText, L"Лінія");
            break;
        case IDM_RECTANGLE:
            lstrcpy(lpttt->szText, L"Прямокутник");
            break;
        case IDM_ELLIPSE:
            lstrcpy(lpttt->szText, L"Еліпс");
            break;
        default: lstrcpy(lpttt->szText, L"Нічого");
        }
    }
}
```

toolbar.h

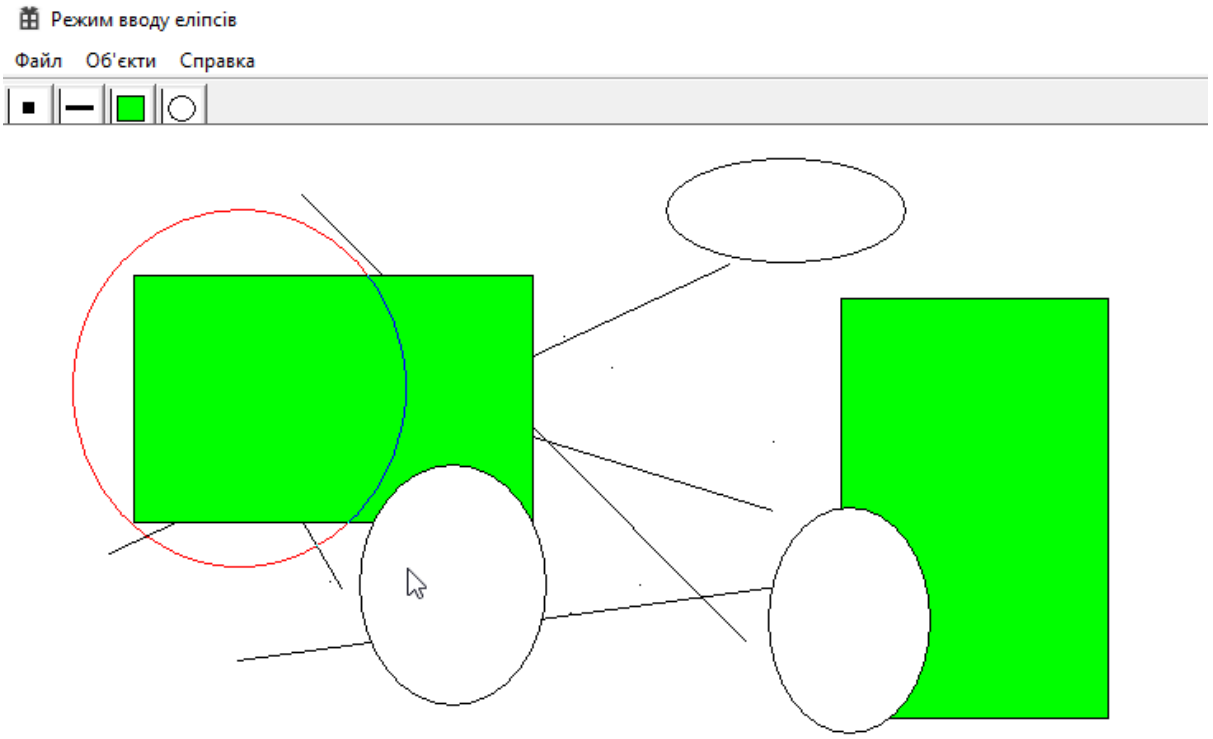
```
#pragma once
#include "framework.h"
#include <commctrl.h>
#pragma comment(lib, "comctl32.lib")

class Toolbar
{
private:
        HWND hWndToolBar;
public:
        void OnCreate(HWND);
        void OnSize(HWND);
        void OnNotify(HWND, LPARAM, WPARAM);
};
```

Діаграма класів:



Ілюстрації роботи програми:



Висновки:

Виконуючи третю лабораторну роботу я познайомився з розробкою тулбару на C++.