**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:

студент групи ІП-93

Домінський Валентин Олексійович

номер залікової книжки: 9311

Перевірив:

Порєв Віктор Миколайович

Київ 2020

## Мета:

Мета роботи – отримати вміння та навички використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів C++, запрограмувавши графічний інтерфейс користувача.

## 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

- статичний масив для Shape (10 mod 3 = 1) обсягом 110 об’єктів

- "гумовий" слід (10 mod 4 = 2) – суцільна лінія синього кольору

- прямокутник:

    - по двом протилежним кутам (10 mod 2 = 0)

    - чорний контур з білим заповненням (10 mod 5 = 0)

- еліпс:

    - від центру до одного з кутів охоплюючого прямокутника (10 mod 2 = 0)

    - чорний контур еліпсу без заповнення (10 mod 5 = 0)

-позначка поточного типу об’єкту:-в меню (метод OnInitMenuPopup) (10 mod 2 = 0)

# Вихідні тексти файлів:

## Lab2.cpp:

```
// Lab1.cpp : Defines the input point for the application.
//
// First Part

#include "framework.h"
#include "pch.h"
#include "Lab3.h"
#include "Resource.h"
#include "shape_editor.h"
#include "toolbar.h"

#define MAX_LOADSTRING 100

#pragma region VariablesAndFunctions

// Global variables:
HINSTANCE hInst;                                // Current instance
WCHAR szTitle[MAX_LOADSTRING];                  // Header row text
WCHAR szWindowClass[MAX_LOADSTRING];            // Class name of main window

ShapeObjectsEditor editorShape;
LPCSTR currentShape;
const LPCSTR POINT_NAME = "Крапка";
const LPCSTR LINE_NAME = "Лінія";
const LPCSTR RECTANGLE_NAME = "Прямокутник";
const LPCSTR ELLIPSE_NAME = "Овал";
Toolbar toolbar;

// Send declarations of functions included in this code module:
ATOM             MyRegisterClass(HINSTANCE hInstance);
BOOL             InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
static void CallToolPoint();
static void CallToolLine();
static void CallToolRectangle();
static void CallToolEllipse();

#pragma endregion VariablesAndFunctions

#pragma region DefaultFunctions

// Second Part
// Enter Point "wWinMain"
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
   _In_opt_ HINSTANCE hPrevInstance,
   _In_ LPWSTR    lpCmdLine,
   _In_ int       nCmdShow)
{
   UNREFERENCED_PARAMETER(hPrevInstance);
   UNREFERENCED_PARAMETER(lpCmdLine);

   InitCommonControls();
   // TODO: Place the code here.

   // Global line initialization
   LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
```

```
    LoadStringW(hInstance, IDC_LAB3, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_LAB3));

    MSG msg;

    // Main message cycle:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int)msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
//  OBJECTIVE: To register the window class.
// Text of Function
/// <summary>
/// Register the window class.
/// </summary>
/// <param name="hInstance">The h instance.</param>
/// <returns></returns>
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);
}

//
//   FUNCTION: InitInstance(HINSTANCE, int)
//
```

```
//  OBJECTIVE: Saves the instance marker and creates the main window
//
//  COMMENTARIES:
//
//      In this function, the instance marker is saved in a global variable, and also
//      the main program window is created and displayed.
//
/// <summary>
/// Saves the instance marker and creates the main window
/// </summary>
/// <param name="hInstance">The h instance.</param>
/// <param name="nCmdShow">The n command show.</param>
/// <returns></returns>
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // Save instance marker in global variable

   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;
}

/// <summary>
/// Message handler for "About" window.
/// </summary>
/// <param name="hDlg">The h dialog.</param>
/// <param name="message">The message.</param>
/// <param name="wParam">The w parameter.</param>
/// <param name="lParam">The l parameter.</param>
/// <returns></returns>
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;
}

#pragma endregion
```

```
#pragma region ModifiedFuntions

// Third Part
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  OBJECTIVE: Processes messages in the main window.
//
//  WM_COMMAND  - Process the application menu
//  WM_PAINT    - Drawing of the main window
//  WM_DESTROY  - Send message about exit and return
//
//
/// <summary>
/// Processes messages in the main window.
/// </summary>
/// <param name="hWnd">The h WND.</param>
/// <param name="message">The message.</param>
/// <param name="wParam">The w parameter.</param>
/// <param name="lParam">The l parameter.</param>
/// <returns></returns>
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   switch (message)
   {
   case WM_CREATE:
      toolbar.OnCreate(hWnd); // here we will create Toolbar
      break;
   case WM_SIZE: // this message is sent if the window resizes
      toolbar.OnSize(hWnd);
      break;
   case WM_NOTIFY: // message from the buttons
      toolbar.OnNotify(hWnd, lParam);
      break;
   case WM_LBUTTONDOWN:
      editorShape.OnLBdown(hWnd);
      break;
   case WM_LBUTTONUP:
      editorShape.OnLBup(hWnd);
      break;
   case WM_MOUSEMOVE:
      editorShape.OnMouseMove(hWnd);
      break;
   case WM_PAINT:
      editorShape.OnPaint(hWnd);
      break;
   case WM_INITMENUPOPUP:
      editorShape.OnInitMenuPopup(hWnd, wParam);
      break;
   case WM_COMMAND:
   {
      int wmId = LOWORD(wParam);
      switch (wmId)
      {
      case ID_TOOL_POINT:
         CallToolPoint();
         break;
      case ID_TOOL_LINE:
         CallToolLine();
         break;
      case ID_TOOL_RECTANGLE:
```

```
            CallToolRectangle();
            break;
        case ID_TOOL_ELLIPSE:
            CallToolEllipse();
            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;
}

/// <summary>
/// Do something when Point tool is used
/// </summary>
void CallToolPoint()
{
    toolbar.OnToolPoint();
    editorShape.StartPointEditor();
}

/// <summary>
/// Do something when Line tool is used
/// </summary>
void CallToolLine()
{
    toolbar.OnToolLine();
    editorShape.StartLineEditor();
}

/// <summary>
/// Do something when Rectangle tool is used
/// </summary>
void CallToolRectangle()
{
    toolbar.OnToolRectangle();
    editorShape.StartRectangleEditor();
}

/// <summary>
/// Do something when Ellipse tool is used
/// </summary>
void CallToolEllipse()
{
    toolbar.OnToolEllipse();
    editorShape.StartEllipseEditor();
}
```

```
/// <summary>
/// Sets the shape name
/// </summary>
void SetShape(int ShapeNumber)
{
   switch (ShapeNumber)
   {
   case(0):
      currentShape = POINT_NAME;
      break;
   case(1):
      currentShape = LINE_NAME;
      break;
   case(2):
      currentShape = RECTANGLE_NAME;
      break;
   case(3):
      currentShape = ELLIPSE_NAME;
      break;
   default:
      break;
   }
}

#pragma endregion ModifiedFuntions
```

## Shape.cpp:

```
#include "framework.h"
#include "pch.h"
#include "shape.h"
#include "colors.h"

#pragma region Functions

/// <summary>
/// // Get coords of points
/// </summary>
/// <param name="x1">first point</param>
/// <param name="y1">second point</param>
/// <param name="x2">third point</param>
/// <param name="y2">fourth point</param>
void Shape::Set(long x1, long y1, long x2, long y2)
{
   xs1 = x1;
   ys1 = y1;
   xs2 = x2;
   ys2 = y2;
}

/// <summary>
/// Shows the pixel
/// </summary>
/// <param name="hdc">handle to a device context</param>
void PointShape::Show(HDC hdc)
{
   SetPixel(hdc, xs1, ys1, black);
}
```

```
/// <summary>
/// Shows the line
/// </summary>
/// <param name="hdc">handle to a device context</param>
void LineShape::Show(HDC hdc)
{
   HPEN hPen, hPenOld;
   hPen = CreatePen(PS_SOLID, 1, black);
   hPenOld = (HPEN)SelectObject(hdc, hPen);
   MoveToEx(hdc, xs1, ys1, NULL);
   LineTo(hdc, xs2, ys2);
   SelectObject(hdc, hPenOld);
   DeleteObject(hPen);
}

/// <summary>
/// Shows the rectangle
/// </summary>
/// <param name="hdc">handle to a device context</param>
void RectangleShape::Show(HDC hdc)
{
   HPEN hPen, hPenOld;
   HBRUSH hBrush, hBrushOld;
   hPen = CreatePen(PS_SOLID, 1, black);
   hPenOld = (HPEN)SelectObject(hdc, hPen);
   hBrush = CreateSolidBrush(white);
   hBrushOld = (HBRUSH)SelectObject(hdc, hBrush);
   SelectObject(hdc, hBrush);
   Rectangle(hdc, xs1, ys1, xs2, ys2);
   SelectObject(hdc, hBrushOld);
   DeleteObject(hBrush);
   SelectObject(hdc, hPenOld);
   DeleteObject(hPen);
}

/// <summary>
/// Shows the ellipse
/// </summary>
/// <param name="hdc">handle to a device context</param>
void EllipseShape::Show(HDC hdc)
{
   HPEN hPen, hPenOld;
   HBRUSH hBrush, hBrushOld;
   hPen = CreatePen(PS_SOLID, 1, black);
   hPenOld = (HPEN)SelectObject(hdc, hPen);
   Arc(hdc, xs1, ys1, xs2, ys2, 0, 0, 0, 0);
   SelectObject(hdc, hPenOld);
   DeleteObject(hPen);
};

#pragma endregion Functions
```

## Shape.h:

```
#include "pch.h"

/// <summary>
/// Main class for shapes
/// </summary>
class Shape
```

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

/// <summary>
/// Class for points
/// </summary>
class PointShape : public Shape
{
public:
   void Show(HDC);
};

/// <summary>
/// Class for lines
/// </summary>
class LineShape : public Shape
{
public:
   void Show(HDC);
};

/// <summary>
/// Class for rectangles
/// </summary>
class RectangleShape : public Shape
{
public:
   void Show(HDC);
};

/// <summary>
/// Class for ellipses
/// </summary>
class EllipseShape : public Shape
{
public:
   void Show(HDC);
};
```

## Shape_editor.cpp:

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

#pragma region Variables

const int Size_Of_Array = 110;
Shape* pcshape[Size_Of_Array];
int size = 0;
bool isPressed;

#pragma endregion Variables

#pragma region Functions
```

```
#pragma region ShapeObjectsEditor

/// <summary>
/// Constructor
/// </summary>
ShapeObjectsEditor::ShapeObjectsEditor()
{
   pse = new PointEditor;
}

/// <summary>
/// Destructor
/// </summary>
ShapeObjectsEditor::~ShapeObjectsEditor()
{
   for (int i = 0; i < size; i++)
   {
      delete pcshape[i];
   }
}

/// <summary>
/// Starts the PointEditor
/// </summary>
void ShapeObjectsEditor::StartPointEditor()
{
   if (pse)
   {
      delete pse;
   }
   pse = new PointEditor;
   SetShape(0);
}

/// <summary>
/// Starts the LineEditor
/// </summary>
void ShapeObjectsEditor::StartLineEditor()
{
   if (pse)
   {
      delete pse;
   }
   pse = new LineEditor;
   SetShape(1);
}

/// <summary>
/// Starts the RectangleEditor
/// </summary>
void ShapeObjectsEditor::StartRectangleEditor()
{
   if (pse)
   {
      delete pse;
   }
   pse = new RectangleEditor;
   SetShape(2);
}

/// <summary>
/// Starts the EllipseEditor
/// </summary>
void ShapeObjectsEditor::StartEllipseEditor()
{
   if (pse)
```

```
  {
    delete pse;
  }
  pse = new EllipseEditor;
  SetShape(3);
}

/// <summary>
/// Do something on left mouse button clicked
/// </summary>
/// <param name="hWnd">window</param>
void ShapeObjectsEditor::OnLBdown(HWND hWnd)
{
  if (pse)
  {
    pse->OnLBdown(hWnd);
  }
}

/// <summary>
/// Do something on left mouse button unclicked
/// </summary>
/// <param name="hWnd">window</param>
void ShapeObjectsEditor::OnLBup(HWND hWnd)
{
  if (pse)
  {
    pse->OnLBup(hWnd);
  }
}

/// <summary>
/// Do something on left mouse moving
/// </summary>
/// <param name="hWnd">window</param>
void ShapeObjectsEditor::OnMouseMove(HWND hWnd)
{
  if (pse && isPressed)
  {
    pse->OnMouseMove(hWnd);
  }
}

/// <summary>
/// Do something on paint
/// </summary>
/// <param name="hWnd">window</param>
void ShapeObjectsEditor::OnPaint(HWND hWnd)
{
  ShapeEditor* draw = new ShapeEditor;
  draw->OnPaint(hWnd);
}

/// <summary>
/// Sets the mark in figures menu
/// </summary>
/// <param name="hWnd"></param>
/// <param name="wParams"></param>
void ShapeObjectsEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParams)
{
  if (pse)
  {
    pse->OnInitMenuPopup(hWnd, wParams);
  }
}
```

```
#pragma endregion ShapeObjectsEditor

#pragma region ShapeEditor

void ShapeEditor::OnMouseMove(HWND hWnd) {};

/// <summary>
/// Do something on left mouse button clicked
/// </summary>
/// <param name="hWnd">window</param>
void ShapeEditor::OnLBdown(HWND hWnd)
{
   isPressed = TRUE;
   POINT pt;
   GetCursorPos(&pt);
   ScreenToClient(hWnd, &pt);
   x1 = x2 = pt.x;
   y1 = y2 = pt.y;
}

/// <summary>
/// Do something on left mouse button unclicked
/// </summary>
/// <param name="hWnd">window</param>
void ShapeEditor::OnLBup(HWND hWnd)
{
   POINT pt;
   GetCursorPos(&pt);
   ScreenToClient(hWnd, &pt);
   x2 = pt.x;
   y2 = pt.y;
   isPressed = FALSE;
}

/// <summary>
/// InitMenu Popup
/// </summary>
/// <param name="hWnd"></param>
/// <param name="wParams"></param>
void ShapeEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParams) {};

/// <summary>
/// Do something on paint
/// </summary>
/// <param name="hWnd">window</param>
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);
}

#pragma endregion ShapeEditor

#pragma region PointEditor

/// <summary>
/// Do something on left mouse button unclicked
```

```
/// </summary>
/// <param name="hWnd">window</param>
void PointEditor::OnLBup(HWND hWnd)
{
   __super::OnLBup(hWnd);
   PointShape* Point = new PointShape;
   Point->Set(x1, y1, x2, y2);
   pcshape[size] = Point;
   size++;
   InvalidateRect(hWnd, NULL, TRUE);
}

/// <summary>
/// Sets the Check
/// </summary>
/// <param name="hWnd"></param>
/// <param name="wParams"></param>
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_RECTANGLE, MF_UNCHECKED);
      CheckMenuItem(hSubMenu, IDM_ELLIPSE, MF_UNCHECKED);
   }
}

#pragma endregion PointEditor

#pragma region LineEditor

/// <summary>
/// Do something on left mouse button unclicked
/// </summary>
/// <param name="hWnd">window</param>
void LineEditor::OnLBup(HWND hWnd)
{
   __super::OnLBup(hWnd);
   LineShape* Line = new LineShape;
   Line->Set(x1, y1, x2, y2);
   pcshape[size] = Line;
   size++;
   InvalidateRect(hWnd, NULL, TRUE);
}

/// <summary>
/// Sets the Check
/// </summary>
/// <param name="hWnd"></param>
/// <param name="wParams"></param>
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_RECTANGLE, MF_UNCHECKED);
      CheckMenuItem(hSubMenu, IDM_ELLIPSE, MF_UNCHECKED);
   }
```

```
}

/// <summary>
/// Do something on Mouse moving
/// </summary>
/// <param name="hWnd">window</param>
void LineEditor::OnMouseMove(HWND hWnd)
{
   POINT pt;
   HPEN hPen, hPenOld;
   HDC hdc = GetDC(hWnd);
   SetROP2(hdc, R2_NOTXORPEN);
   hPen = CreatePen(PS_SOLID, 1, blue);
   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);
}

#pragma endregion LineEditor

#pragma region RectangleEditor

/// <summary>
/// Do something on left mouse button unclicked
/// </summary>
/// <param name="hWnd">window</param>
void RectangleEditor::OnLBup(HWND hWnd)
{
   __super::OnLBup(hWnd);
   RectangleShape* Rectangle = new RectangleShape;
   Rectangle->Set(x1, y1, x2, y2);
   pcshape[size] = Rectangle;
   size++;
   InvalidateRect(hWnd, NULL, TRUE);
}

/// <summary>
/// Sets the Check
/// </summary>
/// <param name="hWnd"></param>
/// <param name="wParams"></param>
void RectangleEditor::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_RECTANGLE, MF_CHECKED);
      CheckMenuItem(hSubMenu, IDM_ELLIPSE, MF_UNCHECKED);
   }
}

/// <summary>
/// Do something on Mouse moving
```

```
/// </summary>
/// <param name="hWnd">window</param>
void RectangleEditor::OnMouseMove(HWND hWnd)
{
   POINT pt;
   HPEN hPen, hPenOld;
   HDC hdc = GetDC(hWnd);
   SetROP2(hdc, R2_NOTXORPEN);
   hPen = CreatePen(PS_SOLID, 1, blue);
   hPenOld = (HPEN)SelectObject(hdc, hPen);
   Rectangle(hdc, x1, y1, x2, y2);
   GetCursorPos(&pt);
   ScreenToClient(hWnd, &pt);
   x2 = pt.x;
   y2 = pt.y;
   Rectangle(hdc, x1, y1, x2, y2);
   SelectObject(hdc, hPenOld);
   DeleteObject(hPen);
   ReleaseDC(hWnd, hdc);
}

#pragma endregion RectangleEditor

#pragma region EllipseEditor

/// <summary>
/// Do something on left mouse button unclicked
/// </summary>
/// <param name="hWnd">window</param>
void EllipseEditor::OnLBup(HWND hWnd)
{
   __super::OnLBup(hWnd);
   EllipseShape* Ellipse = new EllipseShape;
   Ellipse->Set(2 * x1 - x2, 2 * y1 - y2, x2, y2);
   pcshape[size] = Ellipse;
   size++;
   InvalidateRect(hWnd, NULL, TRUE);
}

/// <summary>
/// Sets the Check
/// </summary>
/// <param name="hWnd"></param>
/// <param name="wParams"></param>
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_RECTANGLE, MF_UNCHECKED);
      CheckMenuItem(hSubMenu, IDM_ELLIPSE, MF_CHECKED);
   }
}

/// <summary>
/// Do something on Mouse moving
/// </summary>
/// <param name="hWnd">window</param>
void EllipseEditor::OnMouseMove(HWND hWnd)
{
   POINT pt;
   HPEN hPen, hPenOld;
```

```
    HDC hdc = GetDC(hWnd);
    SetROP2(hdc, R2_NOTXORPEN);
    hPen = CreatePen(PS_SOLID, 1, blue);
    hPenOld = (HPEN)SelectObject(hdc, hPen);
    Arc(hdc, 2 * x1 - x2, 2 * y1 - y2, x2, y2, 0, 0, 0, 0);
    GetCursorPos(&pt);
    ScreenToClient(hWnd, &pt);
    x2 = pt.x;
    y2 = pt.y;
    Arc(hdc, 2 * x1 - x2, 2 * y1 - y2, x2, y2, 0, 0, 0, 0);
    SelectObject(hdc, hPenOld);
    DeleteObject(hPen);
    ReleaseDC(hWnd, hdc);
}

#pragma endregion EllipseEditor

#pragma endregion Functions
```

## Shape_editor.h:

```
#pragma once
#include "pch.h"
#include "editor.h"
#include "Resource.h"

#pragma region Editors

/// <summary>
/// Shape editor class for figures
/// </summary>
class ShapeEditor : public Editor
{
protected:
    long x1, x2, y1, y2;
public:
    void OnLBdown(HWND);
    void OnLBup(HWND);
    void OnMouseMove(HWND);
    void OnPaint(HWND);
    virtual void OnInitMenuPopup(HWND, WPARAM);
};

/// <summary>
/// Shape editor class for figure objects
/// </summary>
class ShapeObjectsEditor
{
private:
    ShapeEditor* pse;
public:
    ShapeObjectsEditor(void);
    ~ShapeObjectsEditor();
    void StartPointEditor();
    void StartLineEditor();
    void StartRectangleEditor();
    void StartEllipseEditor();
    void OnLBdown(HWND);
    void OnLBup(HWND);
    void OnMouseMove(HWND);
    void OnPaint(HWND);
    void OnInitMenuPopup(HWND, WPARAM);
```

```
};

/// <summary>
/// Point editor class for points
/// </summary>
class PointEditor : public ShapeEditor
{
public:
   void OnLBup(HWND);
   void OnInitMenuPopup(HWND, WPARAM);
};

/// <summary>
/// Line editor class for lines
/// </summary>
class LineEditor : public ShapeEditor
{
public:
   void OnLBup(HWND);
   void OnMouseMove(HWND);
   void OnInitMenuPopup(HWND, WPARAM);
};

/// <summary>
/// Rectangle editor class for rectangles
/// </summary>
class RectangleEditor : public ShapeEditor
{
public:
   void OnLBup(HWND);
   void OnMouseMove(HWND);
   void OnInitMenuPopup(HWND, WPARAM);
};

/// <summary>
/// Ellipse editor class for ellipses
/// </summary>
class EllipseEditor : public ShapeEditor
{
public:
   void OnLBup(HWND);
   void OnMouseMove(HWND);
   void OnInitMenuPopup(HWND, WPARAM);
};

#pragma endregion Editors
```

## Editor.h:

```
#pragma once
#include "pch.h"

/// <summary>
/// Main interface
/// </summary>
class Editor
{
public:
   virtual void OnLBdown(HWND) = 0;
   virtual void OnLBup(HWND) = 0;
   virtual void OnMouseMove(HWND) = 0;
```

```
    virtual void OnPaint(HWND) = 0;
};
```

## Toolbar.cpp:

```
#include "framework.h"
#include "pch.h"
#include "lab3.h"
#include "toolbar.h"
#include "resource1.h"

#pragma region Variables

HWND hwndToolBar = NULL;
int point, line, rectangle, ellipse, buttonToChange = 0;
const int allShapes = 5;
int shapes[allShapes] = { point ,line ,rectangle ,ellipse ,buttonToChange };
const LPCSTR pointName = "Крапка";
const LPCSTR lineName = "Лінія";
const LPCSTR rectangleName = "Прямокутник";
const LPCSTR ellipseName = "Овал";
const LPCSTR unnkownName = "Щось невідоме";

#pragma endregion Variables

#pragma region Functions

/// <summary>
/// Creates toolbar
/// </summary>
/// <param name="hWnd"></param>
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; // image index in BITMAP
    tbb[2].fsState = TBSTATE_ENABLED;
    tbb[2].fsStyle = TBSTYLE_BUTTON;
    tbb[2].idCommand = ID_TOOL_RECTANGLE;
    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; // separator of groups of buttons
    tbb[4].idCommand = 0;
    hwndToolBar = CreateToolbarEx(hWnd,
        WS_CHILD | WS_VISIBLE | WS_BORDER | WS_CLIPSIBLINGS | CCS_TOP | TBSTYLE_TOOLTIPS,
        IDC_MY_TOOLBAR,
        4, // number of images in BITMAP
        hInst,
        IDB_BITMAP1, // BITMAP resource ID
```

```
        tbb,
        5, // number of buttons (with separator)
        24, 24, 24, 24, // BITMAP button and image sizes
        sizeof(TBBUTTON));
}

// --- message handler WM_SIZE ---
/// <summary>
/// Change size of toolbar
/// </summary>
/// <param name="hWnd"></param>
void Toolbar::OnSize(HWND hWnd)
{
    RECT rc, rw;
    if (hwndToolBar)
    {
        GetClientRect(hWnd, &rc); // new dimensions of the main window
        GetWindowRect(hwndToolBar, &rw); // we need to know the height of the Toolbar
        MoveWindow(hwndToolBar, 0, 0, rc.right - rc.left, rw.bottom - rw.top, FALSE);
    }
}

/// <summary>
/// UnClick button and click button
/// </summary>
/// <param name="button"> button to unclick/click </param>
/// <param name="shape"> shape element </param>
void Toolbar::ChangeButton(int button, int shape)
{
    SendMessage(hwndToolBar, TB_PRESSBUTTON, buttonToChange, 0);

    buttonToChange = button;

    SendMessage(hwndToolBar, TB_PRESSBUTTON, buttonToChange, shape);
}

/// <summary>
/// Set all elements to zero
/// </summary>
void Toolbar::SetToZeros()
{
    for (auto& item : shapes)
    {
        item = 0;
    }
}

/// <summary>
/// Sets value to opposite value
/// </summary>
/// <param name="value"></param>
void Toolbar::SetToOpposite(int value)
{
    shapes[value] = !shapes[value];
}

/// <summary>
/// Function for drawing points with buttons animation
/// </summary>
void Toolbar::OnToolPoint()
```

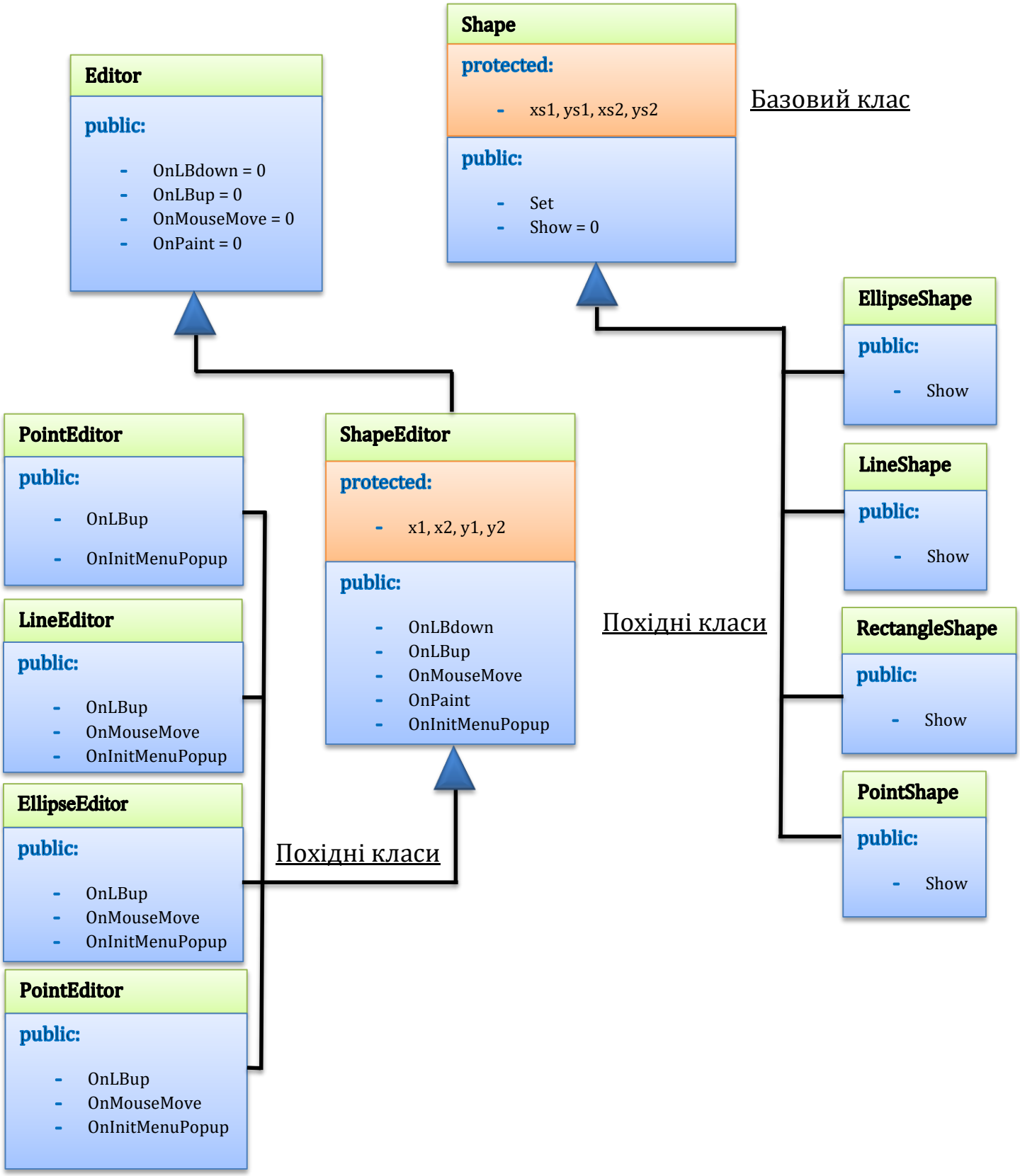
```
{
   SetToZeros();

   SetToOpposite(0);

   ChangeButton(ID_TOOL_POINT,shapes[0]);
}

/// <summary>
/// Function for drawing lines with buttons animation
/// </summary>
void Toolbar::OnToolLine()
{
   SetToZeros();

   SetToOpposite(1);

   ChangeButton(ID_TOOL_LINE, shapes[1]);
}

/// <summary>
/// Function for drawing rectangles with buttons animation
/// </summary>
void Toolbar::OnToolRectangle()
{
   SetToZeros();

   SetToOpposite(2);

   ChangeButton(ID_TOOL_RECTANGLE, shapes[2]);
}

/// <summary>
/// Function for drawing ellipses with buttons animation
/// </summary>
void Toolbar::OnToolEllipse()
{
   SetToZeros();

   SetToOpposite(3);

   ChangeButton(ID_TOOL_ELLIPSE, shapes[3]);
}

/// <summary>
/// Function for tooltips
/// </summary>
/// <param name="hWnd"></param>
/// <param name="lParam"></param>
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 = pointName;
```

```
        break;
    case ID_TOOL_LINE:
        pText = lineName;
        break;
    case ID_TOOL_RECTANGLE:
        pText = rectangleName;
        break;
    case ID_TOOL_ELLIPSE:
        pText = ellipseName;
        break;
    default: pText = unnkownName;
    }
    lstrcpy(lpttt->szText, pText);
  }
}

#pragma endregion Functions
```

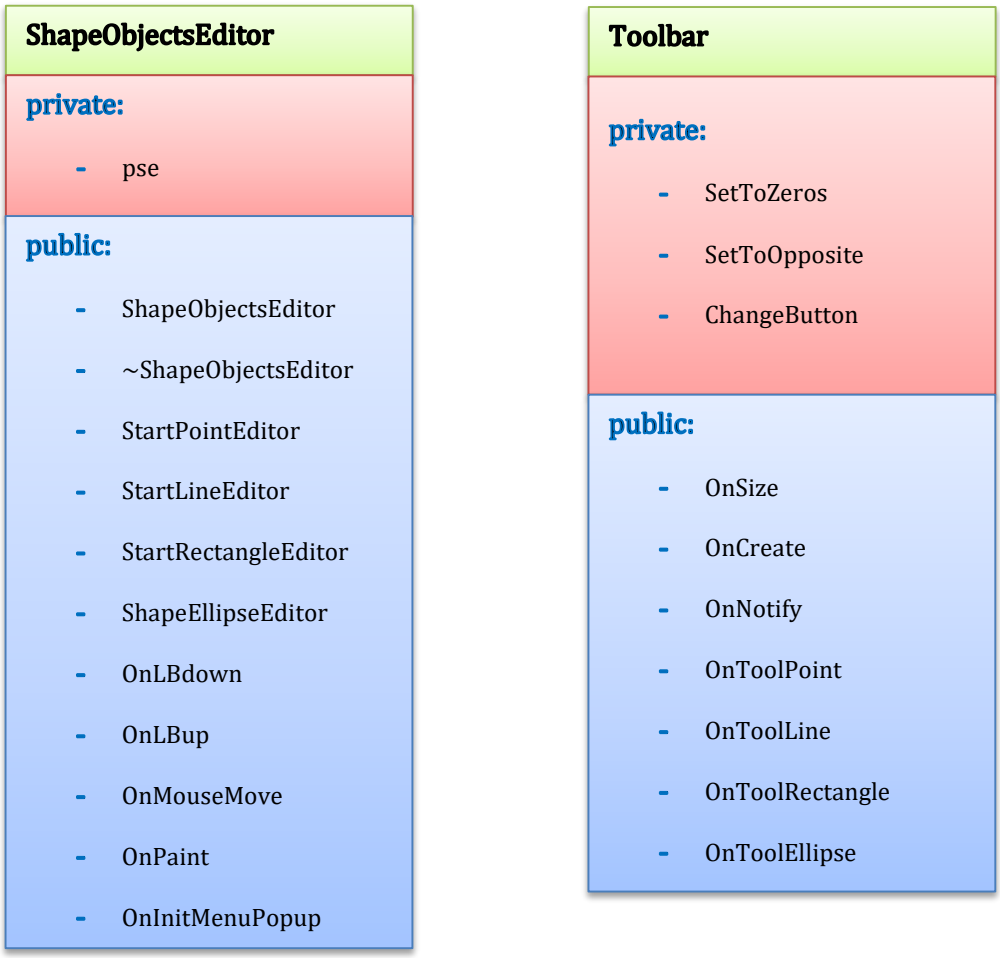
## Toolbar.h:

```
#pragma once

#define ID_TOOL_POINT           32805
#define ID_TOOL_LINE            32806
#define ID_TOOL_RECTANGLE       32807
#define ID_TOOL_ELLIPSE         32809
#define IDC_MY_TOOLBAR          32811

/// <summary>
/// Toolbar class for creating toolbar
/// </summary>
class Toolbar
{
private:
  static void SetToZeros();
  static void SetToOpposite(int value);
  static void ChangeButton(int button, int shape);
public:
  void OnSize(HWND hWnd);
  void OnCreate(HWND hWnd);
  void OnNotify(HWND hWnd, LPARAM lParam);
  void OnToolPoint();
  void OnToolLine();
  void OnToolRectangle();
  void OnToolEllipse();
};
```

# Діаграма класів (1.1)



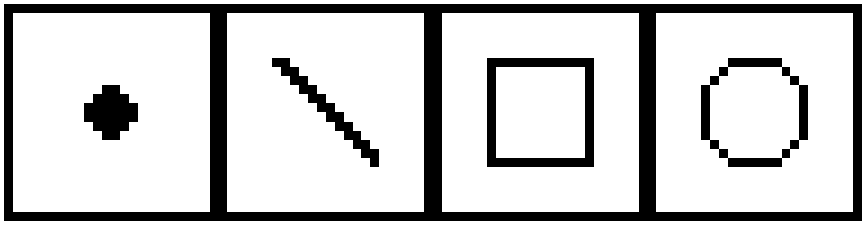
# Діаграма класів (1.2)



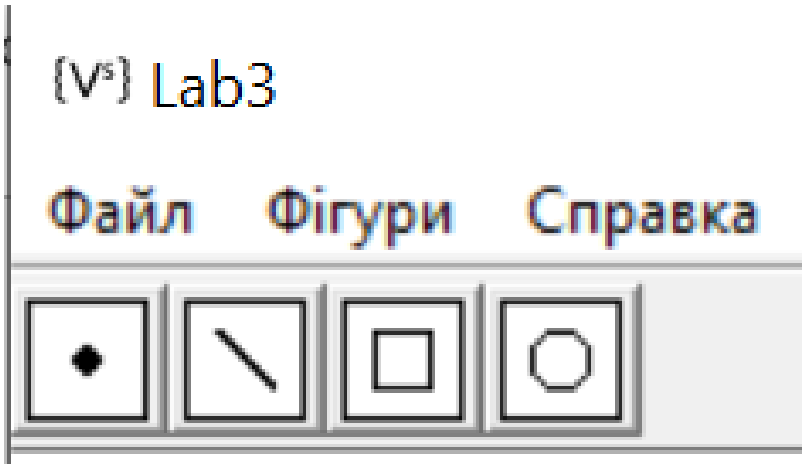
## Скріншоти програми:

Початкове вікно:

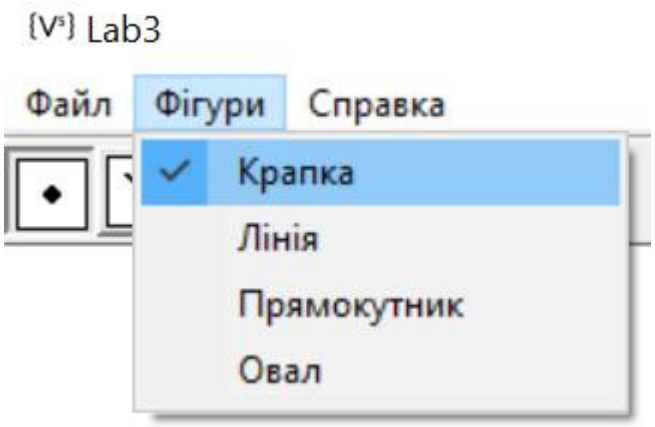

Бітмап:



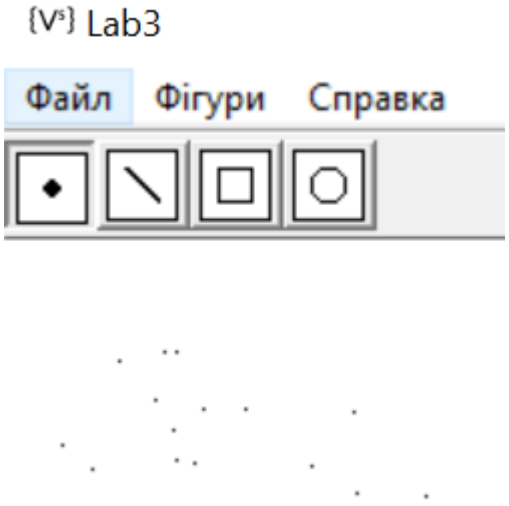
Тулбар:



Вибір у меню:

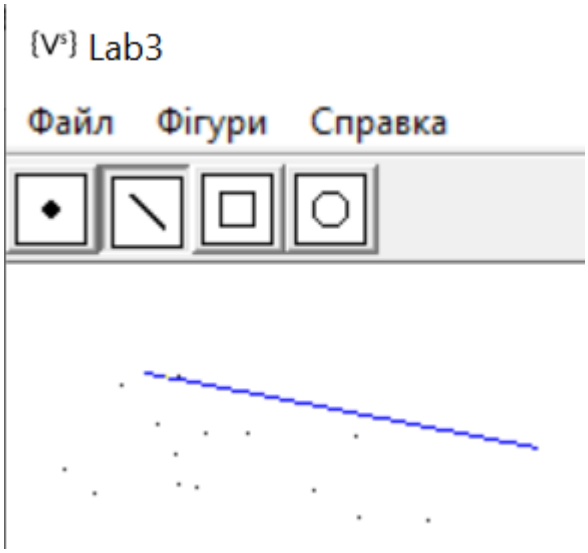



Введення крапок:

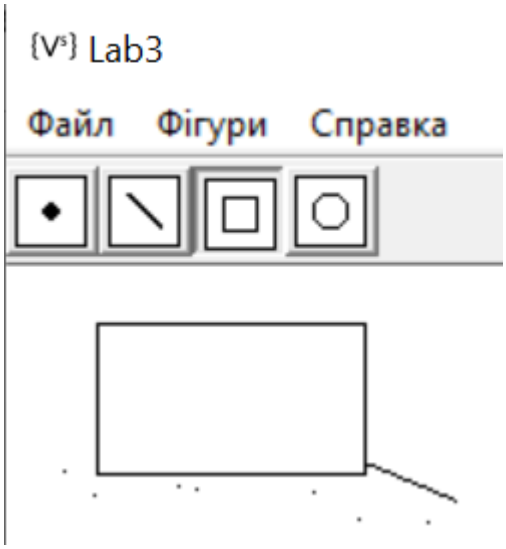



Введення ліній:

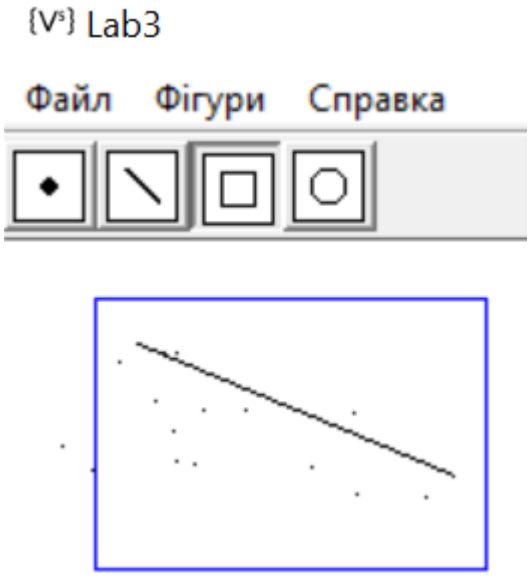

Гумовий слід ліній:



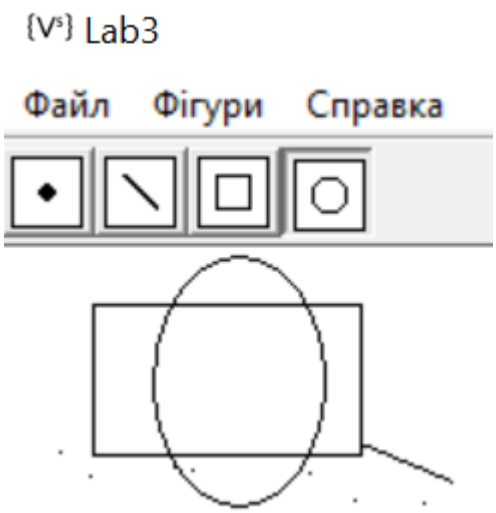

Введення прямокутників:



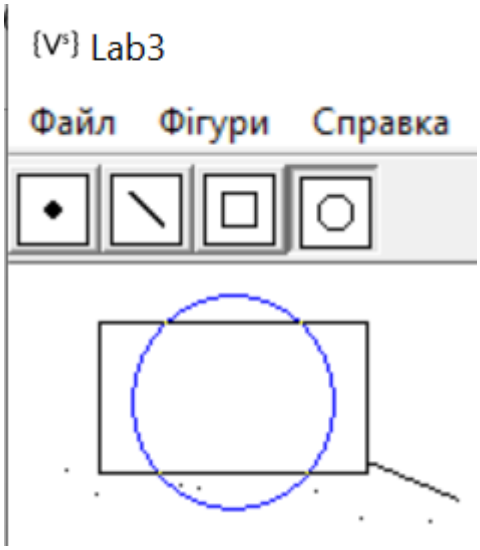

Гумовий слід прямокутників:



Введення еліпсів (овалів):




Гумовий слід еліпсів (овалів):




Також разом з іншими файлами є анімація (.gif) роботи програми

## Контрольні питання

1. Обробку яких повідомлень потрібно виконувати у програмі Лаб3?

    Початок вводу об’єктів (коли натискаєш на пункт у меню «Фігури»), натискання/відпускання лівої кнопки миші, рух миші, натискання на елементи toolbar, методи OnSize(), OnCreate(), OnNotify(),OnInitMenuPopup()

2. Що таке абстрактний клас і скільки їх у цій програмі?

    Абстрактний клас – це базовий клас, від якого не можна створити екземпляру. В абстрактному класі можна описати абстрактні методи та властивості

    У Нас їх три: Shape, ShapeEditor та Editor.

3. Як забезпечити відповідність пунктів меню і кнопок Toolbar?

За допомогою команди ВашМасив[ВашаКнопкаУмасиві].idCommand = IDвашоїКнопки;

4. Як запрограмувати показ власних зображень на кнопках Toolbar?

Під час створення toolbar треба надати бітмап та кількість зображень у ньому, розміри, кількість кнопок у toolbar

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

Треба намалювати (або взяти в іншому місці) бітмап для зображень кнопок. Зображення зберігаються у .rc

6. Як запрограмувати текст підказок (tooltips)?

При створенні toolbar треба додати "TBSTYLE_TOOLTIPS" як стиль, запрограмувати OnNotify та додати WM_NOTIFY у Lab3.cpp

## Висновок:

Навчився малювати фігури. Ознайомився з ООП, абстрактними класами, рівнями захисту, створенням класів. Також навчився працювати з toolbar, з його елементами, створювати бітмапи та використовувати її для кнопок.