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

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

Виконав:

студент групи ІП-93

Домінський Валентин Олексійович

номер залікової книжки: 9311

Номер у списку: 9

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 Lab4.

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

- глобальний статичний об'єкт класу MyEditor.

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

- "гумовий" слід – пунктирна лінія чорного 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:

Lab4.cpp:

```
// Lab4.cpp : Defines the input point for the application.
//
// First Part

#include "framework.h"
#include "pch.h"
#include "Lab3.h"
#include "Resource.h"
#include "my_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

LPCSTR currentShape;
const LPCSTR POINT_NAME = "Крапка";
const LPCSTR LINE_NAME = "Лінія";
const LPCSTR RECTANGLE_NAME = "Прямокутник";
const LPCSTR ELLIPSE_NAME = "Овал";
const LPCSTR LINEOO_NAME = "Лінія з кружочками на кінцях";
const LPCSTR CUBE_NAME = "Куб";

Toolbar toolbar;
MyEditor ED;

// 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();
static void CallToolLineOO();
static void CallToolCube();
static void OnWMCreateCall(HWND);

#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:
        OnWMCreateCall(hWnd);
        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:
        ED.OnLBdown(hWnd);
        break;
    case WM_LBUTTONUP:
        ED.OnLBup(hWnd);
        break;
    case WM_MOUSEMOVE:
        ED.OnMouseMove(hWnd);
        break;
    case WM_PAINT:
        ED.OnPaint(hWnd);
        break;
    case WM_INITMENUPOPUP:
        ED.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 ID_TOOL_LINEOO:
            CallToolLineOO();
            break;
        case ID_TOOL_CUBE:
            CallToolCube();
            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 WM_CREATE is called
/// </summary>
void OnWMCreateCall(HWND hWnd)
{
    toolbar.OnCreate(hWnd); // here we will create Toolbar
    CallToolPoint();
}

/// <summary>
/// Do something when Point tool is used
/// </summary>
void CallToolPoint()
{
    toolbar.OnToolPoint();
    ED.Start(new PointShape);
}

/// <summary>
/// Do something when Line tool is used
/// </summary>
void CallToolLine()
{
```

```
    toolbar.OnToolLine();
    ED.Start(new LineShape);
}

/// <summary>
/// Do something when Rectangle tool is used
/// </summary>
void CallToolRectangle()
{
    toolbar.OnToolRectangle();
    ED.Start(new RectangleShape);
}

/// <summary>
/// Do something when Ellipse tool is used
/// </summary>
void CallToolEllipse()
{
    toolbar.OnToolEllipse();
    ED.Start(new EllipseShape);
}

/// <summary>
/// Do something when LineOO tool is used
/// </summary>
void CallToolLineOO()
{
    toolbar.OnToolLineOO();
    ED.Start(new LineOOShape);
}

/// <summary>
/// Do something when Cube tool is used
/// </summary>
void CallToolCube()
{
    toolbar.OnToolCube();
    ED.Start(new CubeShape);
}

#pragma endregion ModifiedFuntions
```

My_Editor.cpp:

```
#include "framework.h"
#include "pch.h"
#include "my_editor.h"
#include "toolbar.h"

#pragma region Variables

const int Size_Of_Array = 110;
Shape* pcshape[Size_Of_Array];
int size = 0;
bool isPressed;
int const menuCount = 6;
int allMenus[menuCount] = { ID_TOOL_POINT, ID_TOOL_LINE,
ID_TOOL_RECTANGLE, ID_TOOL_ELLIPSE, ID_TOOL_LINEOO, ID_TOOL_CUBE};

#pragma endregion Variables

#pragma region Functions
```

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

/// <summary>
/// Starts new Shape
/// </summary>
/// <param name="shape"></param>
void MyEditor::Start(Shape* shape)
{
   pcshape[size] = shape;
}

/// <summary>
/// Do something, when LB is clicked
/// </summary>
/// <param name="hWnd"></param>
void MyEditor::OnLBdown(HWND hWnd)
{
   POINT pt;
   GetCursorPos(&pt);
   ScreenToClient(hWnd, &pt);
   X1 = X2 = pt.x;
   Y1 = Y2 = pt.y;
   isPressed = true;
}

/// <summary>
/// Do something, when LB is unclicked
/// </summary>
/// <param name="hWnd"></param>
void MyEditor::OnLBup(HWND hWnd)
{
   POINT pt;
   GetCursorPos(&pt);
   ScreenToClient(hWnd, &pt);
   X2 = pt.x;
   Y2 = pt.y;
   isPressed = false;
   pcshape[size]->Set(X1, Y1, X2, Y2);
   size++;
   InvalidateRect(hWnd, NULL, TRUE);
   pcshape[size] = pcshape[size - 1]->Duplicate();
}

/// <summary>
/// Do something, when mouse is moved
/// </summary>
/// <param name="hWnd"></param>
void MyEditor::OnMouseMove(HWND hWnd)
{
   if (isPressed)
   {
      POINT pt;
      HDC hdc = GetDC(hWnd);
      SetROP2(hdc, R2_NOTXORPEN);
      MoveToEx(hdc, X1, Y1, NULL);
      pcshape[size]->Set(X1, Y1, X2, Y2);
```

```
      pcshape[size]->Trail(hdc);
      GetCursorPos(&pt);
      ScreenToClient(hWnd, &pt);
      X2 = pt.x;
      Y2 = pt.y;
      MoveToEx(hdc, X1, Y1, NULL);
      pcshape[size]->Set(X1, Y1, X2, Y2);
      pcshape[size]->Trail(hdc);
      ReleaseDC(hWnd, hdc);
    }
}

/// <summary>
/// Do something, when paint is called
/// </summary>
/// <param name="hWnd"></param>
void MyEditor::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);
}

/// <summary>
/// Change InitMenuPopup
/// </summary>
/// <param name="hWnd"></param>
/// <param name="wParams"></param>
void MyEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParams)
{
    HMENU hMenu, hSubMenu;
    hMenu = GetMenu(hWnd);
    hSubMenu = GetSubMenu(hMenu, 1);

    if ((HMENU)wParams == hSubMenu)
    {
      for (auto& item : allMenus)
      {
        CheckMenuItem(hSubMenu, item, MF_UNCHECKED);
      }

      switch (pcshape[size]->InitMenuPopup())
      {
      case ID_TOOL_POINT:
        CheckMenuItem(hSubMenu, IDM_POINT, MF_CHECKED);
        break;
      case ID_TOOL_LINE:
        CheckMenuItem(hSubMenu, IDM_LINE, MF_CHECKED);
        break;
      case ID_TOOL_RECTANGLE:
        CheckMenuItem(hSubMenu, IDM_RECTANGLE, MF_CHECKED);
        break;
      case ID_TOOL_ELLIPSE:
        CheckMenuItem(hSubMenu, IDM_ELLIPSE, MF_CHECKED);
        break;
      case ID_TOOL_LINEOO:
        CheckMenuItem(hSubMenu, IDM_LINEOO, MF_CHECKED);
        break;
```

```
      case ID_TOOL_CUBE:
        CheckMenuItem(hSubMenu, IDM_CUBE, MF_CHECKED);
        break;
      }
    }
}

#pragma endregion Functions
```

My_Editor.h:

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

#pragma region Editors

/// <summary>
/// Shape editor class for figures
/// </summary>
class MyEditor {
public:
   void Start(Shape*);
   void OnLBdown(HWND);
   void OnLBup(HWND);
   void OnMouseMove(HWND);
   void OnPaint(HWND);
   void OnInitMenuPopup(HWND, WPARAM);
   ~MyEditor();
   long X1, Y1, X2, Y2;
};

#pragma endregion Editors
```

Shape.cpp:

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

#pragma region Variables

int lineOOInt = 20;
int cubeInt = 50;
long X1, X2, Y1, Y2;

#pragma endregion Variables

#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>
```

```
/// Function for showing final shape
/// </summary>
/// <param name="hdc"></param>
void PointShape::Show(HDC hdc)
{
   SetPixel(hdc, XS1, YS1, black);
}

/// <summary>
/// Trail for point
/// </summary>
/// <param name="hdc"></param>
void PointShape::Trail(HDC hdc) {}

/// <summary>
/// Function to get id for Menu
/// </summary>
/// <returns></returns>
int PointShape::InitMenuPopup()
{
   return ID_TOOL_POINT;
}

/// <summary>
/// Function for duplicating
/// </summary>
/// <returns></returns>
Shape* PointShape::Duplicate()
{
   return new PointShape();
}

/// <summary>
/// Function for showing final shape
/// </summary>
/// <param name="hdc"></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>
/// Trail for line
/// </summary>
/// <param name="hdc"></param>
void LineShape::Trail(HDC hdc)
{
   HPEN hPen, hPenOld;
   hPen = CreatePen(PS_DOT, 1, black);
   hPenOld = (HPEN)SelectObject(hdc, hPen);
   MoveToEx(hdc, XS1, YS1, NULL);
   LineTo(hdc, XS2, YS2);
   SelectObject(hdc, hPenOld);
   DeleteObject(hPen);
}

/// <summary>
/// Function to get id for Menu
/// </summary>
/// <returns></returns>
```

```
int LineShape::InitMenuPopup()
{
   return ID_TOOL_LINE;
}

/// <summary>
/// Function for duplicating
/// </summary>
/// <returns></returns>
Shape* LineShape::Duplicate()
{
   return new LineShape();
}

/// <summary>
/// Function for showing final shape
/// </summary>
/// <param name="hdc"></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>
/// Trail for rectangle
/// </summary>
/// <param name="hdc"></param>
void RectangleShape::Trail(HDC hdc)
{
   HPEN hPen, hPenOld;
   hPen = CreatePen(PS_DOT, 1, black);
   hPenOld = (HPEN)SelectObject(hdc, hPen);
   MoveToEx(hdc, XS1, YS1, NULL);
   LineTo(hdc, XS1, YS2);
   LineTo(hdc, XS2, YS2);
   LineTo(hdc, XS2, YS1);
   LineTo(hdc, XS1, YS1);
   SelectObject(hdc, hPenOld);
   DeleteObject(hPen);
}

/// <summary>
/// Function to get id for Menu
/// </summary>
/// <returns></returns>
int RectangleShape::InitMenuPopup()
{
   return ID_TOOL_RECTANGLE;
}

/// <summary>
/// Function for duplicating
/// </summary>
/// <returns></returns>
Shape* RectangleShape::Duplicate()
```

```
{
   return new RectangleShape();
}

/// <summary>
/// Function for showing final shape
/// </summary>
/// <param name="hdc"></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, 2 * XS1 - XS2, 2 * YS1 - YS2, XS2, YS2, 0, 0, 0, 0);
   SelectObject(hdc, hPenOld);
   DeleteObject(hPen);
};

/// <summary>
/// Trail for ellipse
/// </summary>
/// <param name="hdc"></param>
void EllipseShape::Trail(HDC hdc)
{
   HPEN hPen, hPenOld;
   hPen = CreatePen(PS_DOT, 1, black);
   hPenOld = (HPEN)SelectObject(hdc, hPen);
   MoveToEx(hdc, XS1, YS1, NULL);
   Arc(hdc, 2 * XS1 - XS2, 2 * YS1 - YS2, XS2, YS2, 0, 0, 0, 0);
   SelectObject(hdc, hPenOld);
   DeleteObject(hPen);
}

/// <summary>
/// Function to get id for Menu
/// </summary>
/// <returns></returns>
int EllipseShape::InitMenuPopup()
{
   return ID_TOOL_ELLIPSE;
}

/// <summary>
/// Function for duplicating
/// </summary>
/// <returns></returns>
Shape* EllipseShape::Duplicate()
{
   return new EllipseShape();
}

/// <summary>
/// Function for showing final shape
/// </summary>
/// <param name="hdc"></param>
void LineOOShape::Show(HDC hdc)
{
            X1 = XS1;Y1 = YS1;
            X2 = XS2;Y2 = YS2;
            LineShape::Set(X1, Y1, X2, Y2);
            LineShape::Show(hdc);
            EllipseShape::Set(X1, Y1,
                        X1 - lineOOInt, Y1 - lineOOInt);
            EllipseShape::Show(hdc);
            EllipseShape::Set(X2, Y2,
```

```
                X2 - lineOOInt, Y2 - lineOOInt);
        EllipseShape::Show(hdc);
        LineShape::Set(X1, Y1, X2, Y2);}

/// <summary>
/// Trail for lineOO
/// </summary>
/// <param name="hdc"></param>
void LineOOShape::Trail(HDC hdc)
{
         X1 = XS1;Y1 = YS1;
        X2 = XS2;Y2 = YS2;
        LineShape::Set(X1, Y1, X2, Y2);
        LineShape::Trail(hdc);
        EllipseShape::Set(X1, Y1,
                X1 - lineOOInt, Y1 - lineOOInt);
        EllipseShape::Trail(hdc);
        EllipseShape::Set(X2, Y2,
                X2 - lineOOInt, Y2 - lineOOInt);
        EllipseShape::Trail(hdc);
        LineShape::Set(X1, Y1, X2, Y2);}

/// <summary>
/// Function to get id for Menu
/// </summary>
/// <returns></returns>
int LineOOShape::InitMenuPopup()
{
   return ID_TOOL_LINEOO;
}

/// <summary>
/// Function for duplicating
/// </summary>
/// <returns></returns>
Shape* LineOOShape::Duplicate()
{
   return new LineOOShape();
}

/// <summary>
/// Function for showing final shape
/// </summary>
/// <param name="hdc"></param>
void CubeShape::Show(HDC hdc)
{
   X1 = XS1; Y1 = YS1;
   X2 = XS2; Y2 = YS2;
   RectangleShape::Set(X1 - cubeInt, Y1 - cubeInt,
      X1 + cubeInt, Y1 + cubeInt);
   RectangleShape::Show(hdc);
   RectangleShape::Set(X2 - cubeInt, Y2 - cubeInt,
      X2 + cubeInt, Y2 + cubeInt);
   RectangleShape::Show(hdc);
   LineShape::Set(X1 - cubeInt, Y1 - cubeInt,
      X2 - cubeInt, Y2 - cubeInt);
   LineShape::Show(hdc);
   LineShape::Set(X1 - cubeInt, Y1 + cubeInt,
      X2 - cubeInt, Y2 + cubeInt);
   LineShape::Show(hdc);
   LineShape::Set(X1 + cubeInt, Y1 + cubeInt,
      X2 + cubeInt, Y2 + cubeInt);
   LineShape::Show(hdc);
   LineShape::Set(X1 + cubeInt, Y1 - cubeInt,
      X2 + cubeInt, Y2 - cubeInt);
   LineShape::Show(hdc);
```

```
   LineShape::Set(X1, Y1, X2, Y2);
}

/// <summary>
/// Trail for cube
/// </summary>
/// <param name="hdc"></param>
void CubeShape::Trail(HDC hdc)
{
	X1 = XS1; Y1 = YS1; X2 = XS2; Y2 = YS2;
	RectangleShape::Set(X1 - cubeInt, Y1 - cubeInt,
		X1 + cubeInt, Y1 + cubeInt);
	RectangleShape::Trail(hdc);
	RectangleShape::Set(X2 - cubeInt, Y2 - cubeInt,
		X2 + cubeInt, Y2 + cubeInt);
	RectangleShape::Trail(hdc);
	LineShape::Set(X1 - cubeInt, Y1 - cubeInt,
		X2 - cubeInt, Y2 - cubeInt);
	LineShape::Trail(hdc);
	LineShape::Set(X1 - cubeInt, Y1 + cubeInt,
		X2 - cubeInt, Y2 + cubeInt);
	LineShape::Trail(hdc);
	LineShape::Set(X1 + cubeInt, Y1 + cubeInt,
		X2 + cubeInt, Y2 + cubeInt);
	LineShape::Trail(hdc);
	LineShape::Set(X1 + cubeInt, Y1 - cubeInt,
		X2 + cubeInt, Y2 - cubeInt);
	LineShape::Trail(hdc);
	LineShape::Set(X1, Y1, X2, Y2);}

/// <summary>
/// Function to get id for Menu
/// </summary>
/// <returns></returns>
int CubeShape::InitMenuPopup()
{
   return ID_TOOL_CUBE;
}

/// <summary>
/// Function for duplicating
/// </summary>
/// <returns></returns>
Shape* CubeShape::Duplicate()
{
   return new CubeShape();
}

Shape::~Shape() {};

#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;
```

```
    virtual void Trail(HDC) = 0;
    virtual int InitMenuPopup() = 0;
    virtual Shape* Duplicate() = 0;
    ~Shape();
};

/// <summary>
/// Class for point
/// </summary>
class PointShape : public Shape
{
    virtual void Show(HDC);
    void Trail(HDC);
    int InitMenuPopup();
    virtual Shape* Duplicate();
};

/// <summary>
/// Class for line
/// </summary>
class LineShape : public virtual Shape
{
public:
    virtual void Show(HDC);
    void Trail(HDC);
    int InitMenuPopup();
    virtual Shape* Duplicate();
};

/// <summary>
/// Class for rectangle
/// </summary>
class RectangleShape : public virtual Shape
{
public:
    virtual void Show(HDC);
    void Trail(HDC);
    int InitMenuPopup();
    virtual Shape* Duplicate();
};

/// <summary>
/// Class for ellipse
/// </summary>
class EllipseShape : public virtual Shape
{
public:
    virtual void Show(HDC);
    void Trail(HDC);
    int InitMenuPopup();
    virtual Shape* Duplicate();
};

/// <summary>
/// Class for lineOO
/// </summary>
class LineOOShape : public LineShape, public EllipseShape
{
public:
    void Show(HDC);
    void Trail(HDC);
    int InitMenuPopup();
    virtual Shape* Duplicate();
};

/// <summary>
```

```
/// Class for cube
/// </summary>
class CubeShape : public RectangleShape, public LineShape
{
public:
   void Show(HDC);
   void Trail(HDC);
   int InitMenuPopup();
   virtual Shape* Duplicate();
};
```

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, lineOO, cube, buttonToChange = 0;
const int allShapes = 7;
int shapes[allShapes] = { point, line, rectangle, ellipse,
                  lineOO, cube, buttonToChange };
const LPCSTR pointName = "Крапка";
const LPCSTR lineName = "Лінія";
const LPCSTR rectangleName = "Прямокутник";
const LPCSTR ellipseName = "Овал";
const LPCSTR lineOOName = "Лінія з кружочками на кінцях";
const LPCSTR cubeName = "Куб";
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[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; // 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_BUTTON;
    tbb[4].idCommand = ID_TOOL_LINEOO;

    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; // separator of groups of buttons
    tbb[6].idCommand = 0;

    hwndToolBar = CreateToolbarEx(hWnd,
        WS_CHILD | WS_VISIBLE | WS_BORDER | WS_CLIPSIBLINGS | CCS_TOP | TBSTYLE_TOOLTIPS,
        IDC_MY_TOOLBAR,
        6, // number of images in BITMAP
        hInst,
        IDB_BITMAP1, // BITMAP resource ID
        tbb,
        7, // 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 drawing lines with ellipses with buttons animation
/// </summary>
void Toolbar::OnToolLineOO()
{
   SetToZeros();

   SetToOpposite(4);
```

```
    ChangeButton(ID_TOOL_LINEOO, shapes[4]);
}

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

    SetToOpposite(5);

    ChangeButton(ID_TOOL_CUBE, shapes[5]);
}

/// <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;
        case ID_TOOL_LINEOO:
            pText = lineOOName;
            break;
        case ID_TOOL_CUBE:
            pText = cubeName;
            break;
        default:
            pText = unnkownName;
            break;
        }
        lstrcpy(lpttt->szText, pText);
    }
}

#pragma endregion Functions
```

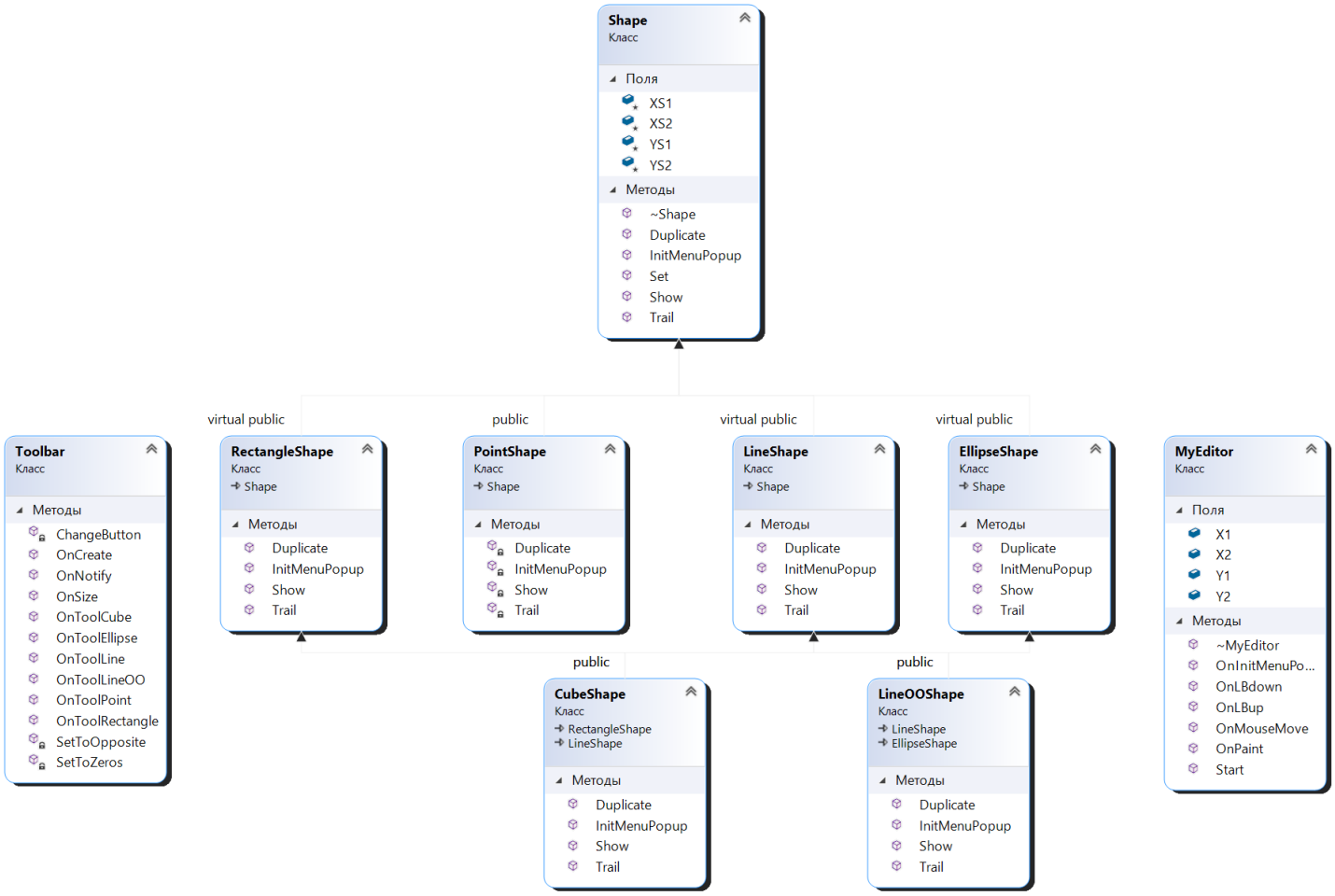
Toolbar.h:

```
#pragma once

#define ID_TOOL_POINT              32805
#define ID_TOOL_LINE               32806
#define ID_TOOL_RECTANGLE          32807
#define ID_TOOL_ELLIPSE            32809
#define ID_TOOL_LINEOO             32824
#define ID_TOOL_CUBE               32825
#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();
   void OnToolLineOO();
   void OnToolCube();
};
```

Діаграма класів

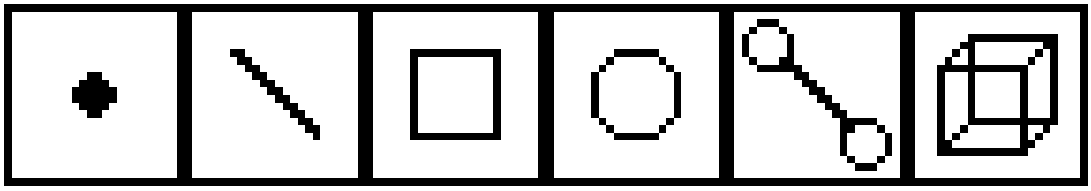


Скріншоти програми:

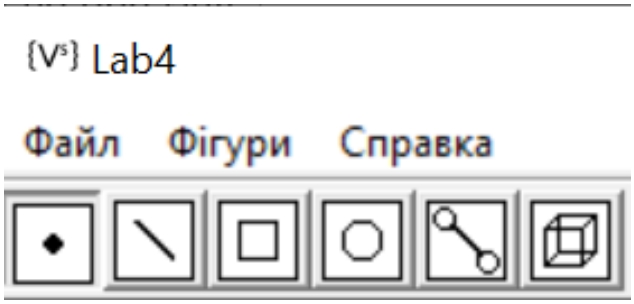
Також разом з іншими файлами є анімація (.gif) роботи програми

Початкове вікно:

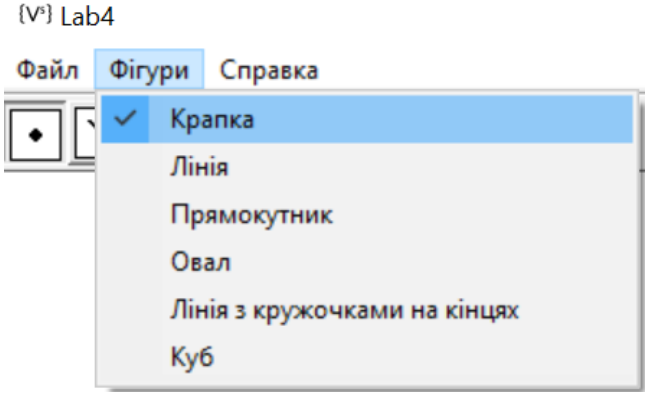
Бітмап:



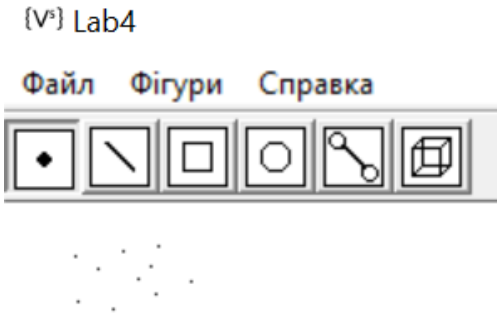
Тулбар:



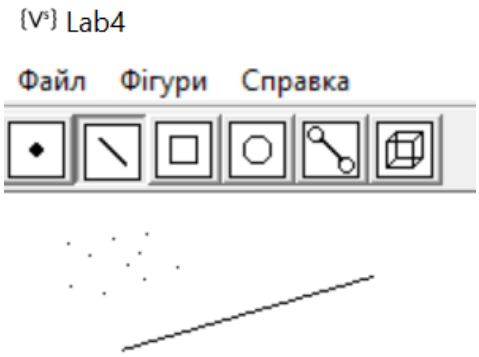
Вибір у меню:



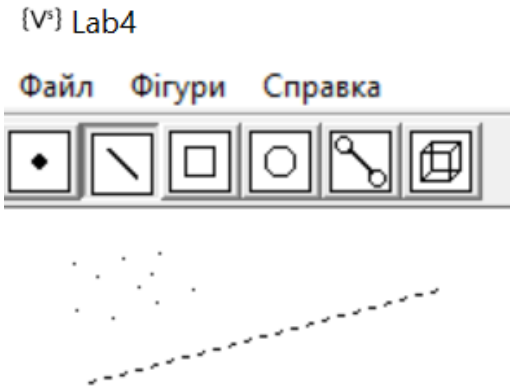
Введення крапок:



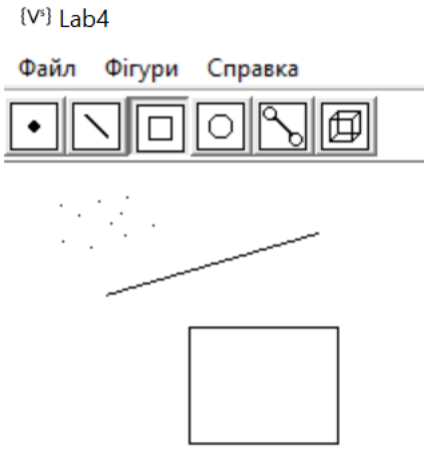
Введення ліній:



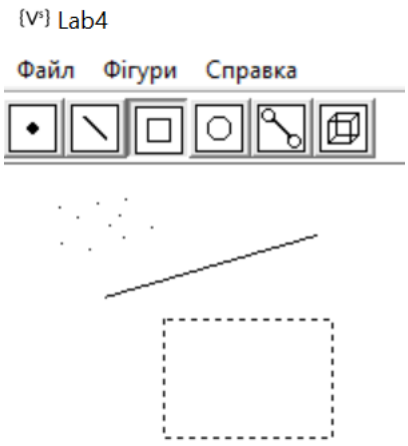
Гумовий слід ліній:



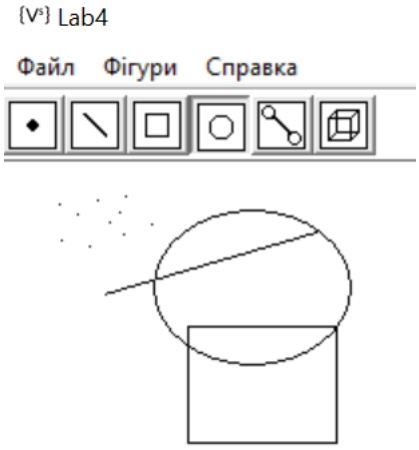
Введення прямокутників:



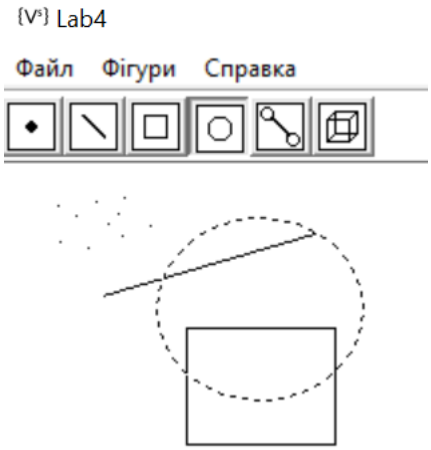
Гумовий слід прямокутників:



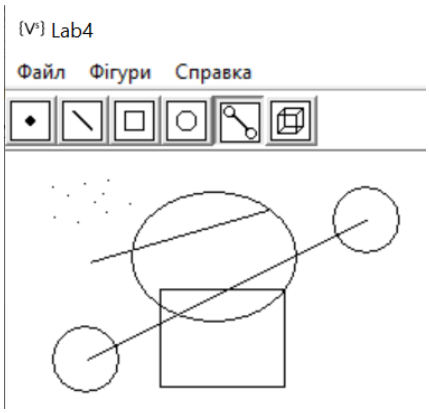
Введення еліпсів (овалів):



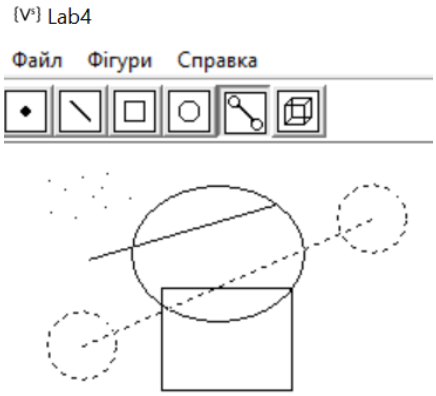
Гумовий слід еліпсів (овалів):



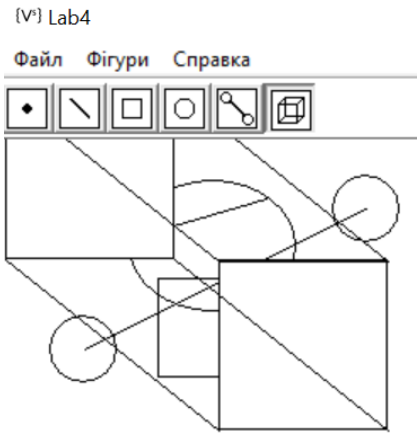
Введення ліній з кружечками на кінцях:



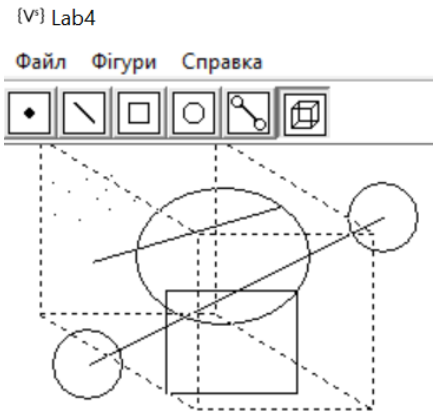
Гумовий слід ліній з кружечками на кінцях:



Введення кубів:



Гумовий слід кубів:



Контрольні питання

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

Поліморфі́зм (з грец. πολύς «багато» + μορφή «форма») — концепція в програмуванні та теорії типів, в основі якої лежить використання єдиного інтерфейсу для різнотипних сутностей або у використанні однакового символу для маніпуляцій над даними різного типу

На противагу поліморфізму, концепція мономорфізму вимагає однозначного зіставлення.

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

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()

3. Що таке абстрактний клас і скільки іх 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 сім: Shape, PointShape, LineShape, RectShape, EllipseShape, LineOOShape, CubeShape,

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

Множи́нна спадко́вість — властивість деяких об’єктно-орієнтованих мов програмування, в 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.

5. 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?

6. Як побудувати діаграму класів засобами Visual Studio?

Вікно Solution Explorer, потім клікнути ПКМ, додати, створити елемент, схема класів (або діаграма класів (також перед цим треба мати встановлений компонент для створення таких діаграм класів)). Вставляємо у вікно файли *.h

7. Як додати нові кнопки у Toolbar?

Треба новий бітмап, масив для зберігання кнопок певного розміру, елементами масиву треба мати значення (які нам самим треба додати) та надати програмі кількість кнопок для toolbar’у

Висновок:

Навчився малювати нові, більш складні фігури. Глибше ознайомився з ООП, абстрактними класами, рівнями захисту, створенням класів. Також навчився працювати з toolbar, з його елементами, створювати бітмапи та використовувати її для кнопок. Спробував працювати з множинним успадкуванням.