

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 №5

з дисципліни

«Розробка багатовіконного інтерфейсу користувача
для графічного редактора об'єктів»

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 програмувати багатовіконний інтерфейс програми на С++ в об'єктно-орієнтованому стилі.

Завдання:

1. Створити у середовищі MS Visual Studio С++ проект Win32 з ім'ям Lab5.
2. Написати вихідний текст програми згідно варіанту завдання.
3. Скомпілювати вихідний текст і отримати виконуваний файл програми.
4. Перевірити роботу програми. Налаштувати програму.
5. Проаналізувати та прокоментувати результати та вихідний текст програми.
6. Оформити звіт.

Варіанти завдань

9 - Singleton Меєрса

Усе інше – з минулої роботи

Вихідні тексти файлів:

Lab5.cpp:

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

#include "framework.h"
#include "pch.h"
#include "Lab5.h"
#include "my_editor.h"
#include "toolbar.h"
#include "my_table.h"
#include "Resource.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";
string detailsOfShape;
INT countForShapes = 0;
INT tableCount = 0;
INT firstInit = 0;

Toolbar toolbar;
MyEditor& ED = ED.getInstance();
MyTable* table = new MyTable;
HWND tableHwnd = NULL;

// 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 CallLBUP(HWND hWnd);
static void CallTable();
static void OnWMCreateCall(HWND);
BOOL CALLBACK Table(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
std::string shapeDetails = "";
string pathForShapes = "objects.txt";
static LPCSTR exceptionString = "Can't open a file or find a file";

#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_LAB5, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

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

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_LAB5));

    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_LAB5));
    wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_LAB5);
    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:
            CallLBUP(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 IDD_TABLEINMENU:
                    CallTable();
                    break;
                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 Table is called
/// </summary>
void CallTable()
{
    if (tableCount == 0)
    {
        tableHwnd = CreateDialog(hInst, MAKEINTRESOURCE(IDD_TABLE), 0, Table);
        ShowWindow(tableHwnd, SW_SHOW);
        tableCount++;
    }
}

/// <summary>
/// Do something on lbup
/// </summary>
void CallLBUP(HWND hWnd)
{
    ED.OnLBup(hWnd);
    shapeDetails = ED.GetDetails();
    table->Add(tableHwnd, shapeDetails);
}

/// <summary>
/// Do something when WM_CREATE is called
/// </summary>
void OnWMCreateCall(HWND hWnd)
{
    toolbar.OnCreate(hWnd); // here we will create Toolbar
    CallToolPoint();

    if (countForShapes == 0)
    {
        ifstream myTableFile;

        myTableFile.open(pathForShapes, std::ofstream::out

```

```

        | std::ofstream::trunc);
myTableFile.close();
}

if (firstInit == 0)
{
    table->Add(tableHwnd,
        "Shape \t x1 \t y1 \t x2 \t y2");
    firstInit++;
}
}

/// <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 Line00 tool is used
/// </summary>
void CallToolLine00()
{
    toolbar.OnToolLine00();
    ED.Start(new Line00Shape);
}

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

```



```

/// <summary>
/// Do something with Table window
/// </summary>
/// <param name="hWnd"></param>
/// <param name="uMsg"></param>
/// <param name="wParam"></param>
/// <param name="lParam"></param>
/// <returns></returns>
BOOL CALLBACK Table(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    ifstream myTableFile;

    switch (uMsg)
    {
    case WM_INITDIALOG:

        myTableFile.open(pathForShapes);
        if (myTableFile.is_open())
        {
            string tempString = "";

            while (!myTableFile.eof())
            {
                getline(myTableFile, tempString);
                if (tempString != "") SendDlgItemMessage(hWnd, IDC_LIST,
                    LB_ADDSTRING, 0, (LPARAM)tempString.c_str());
            }
        }
        else
        {
            throw new exception(exceptionString);
        }

        countForShapes++;

        myTableFile.close();

        return (INT_PTR)TRUE;
        break;
    case WM_COMMAND:
        if (LOWORD(wParam) == IDCANCEL)
        {
            DestroyWindow(tableHwnd);
            tableCount--;
            return TRUE;
        }
        if (LOWORD(wParam) == IDC_EXIT)
        {
            DestroyWindow(tableHwnd);
            tableCount--;
            return TRUE;
        }
    }
    return (INT_PTR)FALSE;
}

#pragma endregion ModifiedFuntions

```

My_Editor.cpp:

```

#include "framework.h"
#include "pch.h"
#include "my_editor.h"
#include "toolbar.h"
#include <sstream>
#include <algorithm>

```

```
#include <string>
```

```
#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;
}
}
}

```

```

/// <summary>
/// Get name and coords of the shape
/// </summary>
/// <returns></returns>
std::string MyEditor::GetDetails()
{
    std::stringstream buffer;

    buffer << pcshape[size]->GetShapeName();
    buffer << " \t ";
    buffer << X1;
    buffer << " \t ";
    buffer << Y1;
    buffer << " \t ";
    buffer << X2;
    buffer << " \t ";
    buffer << Y2;

    std::string shapeString = buffer.str();

    return shapeString;
}

```

```
#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 {
private:
    MyEditor() {}
    MyEditor(const MyEditor&);
    MyEditor& operator = (MyEditor&);
public:
    static MyEditor& getInstance()
    {

```

```

    static MyEditor instance;
    return instance;
}
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;
std::string GetDetails();
};

#pragma endregion Editors

```

Shape.cpp:

```

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

#pragma region Variables

int line00Int = 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>
/// Return name for table
/// </summary>
/// <returns></returns>
std::string PointShape::GetShapeName()
{
    return "Point";
}

/// <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>
/// Return name for table
/// </summary>
/// <returns></returns>
std::string LineShape::GetShapeName()
{
    return "Line";
}

/// <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>
/// Return name for table
/// </summary>
/// <returns></returns>
std::string RectangleShape::GetShapeName()
{
    return "Rect";
}

/// <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>
/// Return name for table
/// </summary>

```



```

/// <returns></returns>
std::string EllipseShape::GetShapeName()
{
    return "Ellipse";
}

```

```

/// <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>
/// Return name for table
/// </summary>
/// <returns></returns>
std::string Line00Shape::GetShapeName()
{
    return "Line00";
}

/// <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();
}

/// <summary>
/// Return name for table
/// </summary>
/// <returns></returns>
std::string CubeShape::GetShapeName()
{
    return "Cube";
}

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;
    virtual std::string GetShapeName() = 0;
    ~Shape();
};

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

/// <summary>
/// Class for line
/// </summary>
class LineShape : public virtual Shape

```

```

{
public:
    virtual void Show(HDC);
    void Trail(HDC);
    int InitMenuPopup();
    virtual Shape* Duplicate();
    virtual std::string GetShapeName();
};

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

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

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

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

```

Toolbar.cpp:

```

#include "framework.h"
#include "pch.h"
#include "lab5.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 unknownName = "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 lptt = (LPTOOLTIPTEXT)lParam;
        switch (lptt->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 = unknkownName;
                break;
        }
        lstrcpy(lptt->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 OnToolLine();
void OnToolCube();
};

```

My_Table.cpp:

```

#include "framework.h"
#include "pch.h"
#include "my_table.h"

static string pathForShapes = "objects.txt";
static LPCSTR exceptionString = "Can't open a file or find a file";

/// <summary>
/// Add shape to table
/// </summary>
/// <param name="shapeDetails">name and coords</param>
void MyTable::Add(HWND hWndDlg, std::string shapeDetails)
{
    ofstream myTableFile;

    myTableFile.open(pathForShapes, ofstream::app);

    if (myTableFile.is_open())
    {
        myTableFile << shapeDetails << endl;
    }
    else
    {
        throw new exception(exceptionString);
    }
    myTableFile.close();

    SendDlgItemMessage(hWndDlg, IDC_LIST, LB_ADDSTRING,
        0, (LPARAM)shapeDetails.c_str());
}

```

My_Table.h:

```

#pragma once
#include "resource2.h"
using namespace std;

/// <summary>
/// Class for table
/// </summary>
class MyTable
{
    //якісь члени класу
public:
    void Add(HWND, std::string); //функція додавання у таблицю нового рядка з описом об'єкту
    //інші функції
};

```













Діаграма класів



Toolbar

Класс


Методы

-  ChangeButton
-  OnCreate
-  OnNotify
-  OnSize
-  OnToolCube
-  OnToolEllipse
-  OnToolLine
-  OnToolLineOO
-  OnToolPoint
-  OnToolRectangle
-  SetToOpposite
-  SetToZeros

MyTable

Класс





Методы

-  Add












MyEditor

Класс

Поля

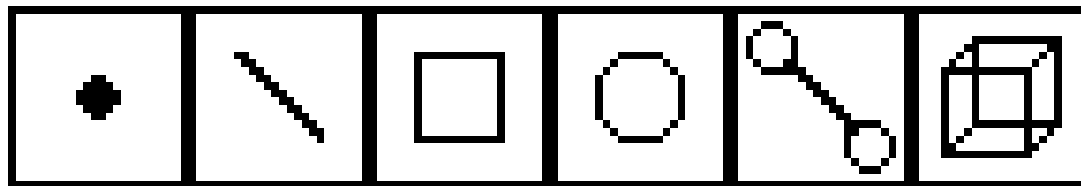
-  X1
-  X2
-  Y1
-  Y2

Методы

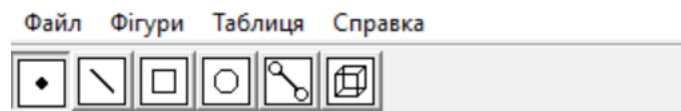
-  ~MyEditor
-  GetDetails
-  getInstance
-  MyEditor (+ 1 п...
-  OnInitMenuPo...
-  OnLBdown
-  OnLBup
-  OnMouseMove
-  OnPaint
-  operator=
-  Start

Скріншоти програми:

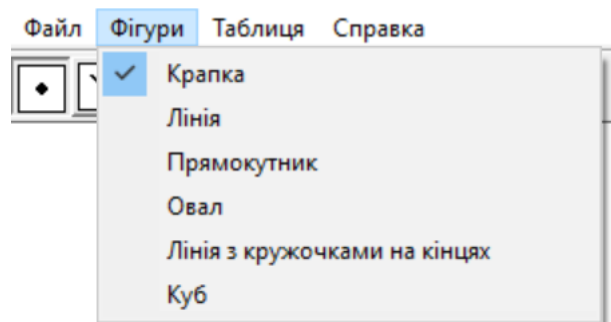
Бітмап:



Тулбар:



Вибір у меню:



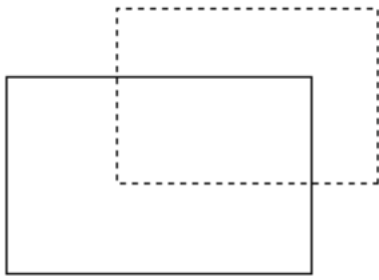
Крпки:



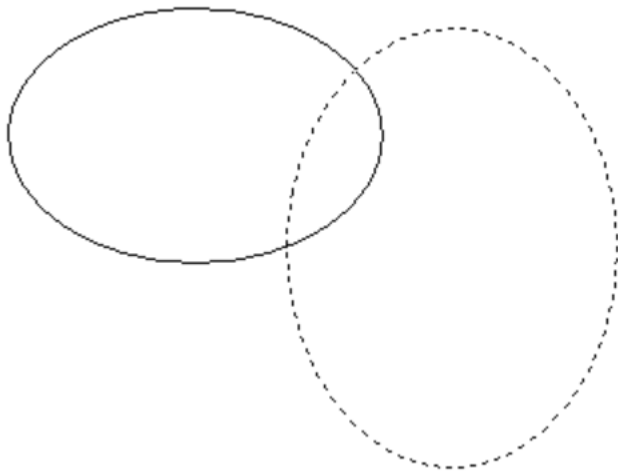
Лінії:



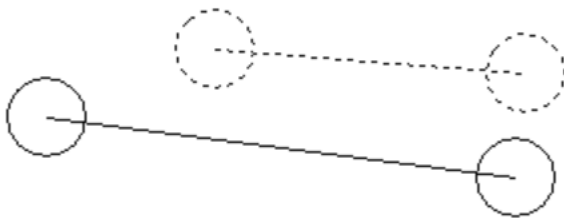
Квадрати:



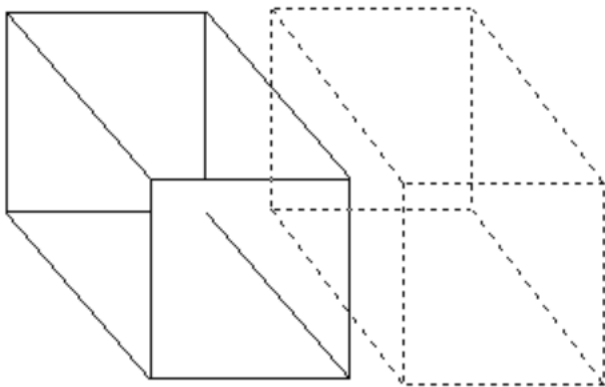
Овали:



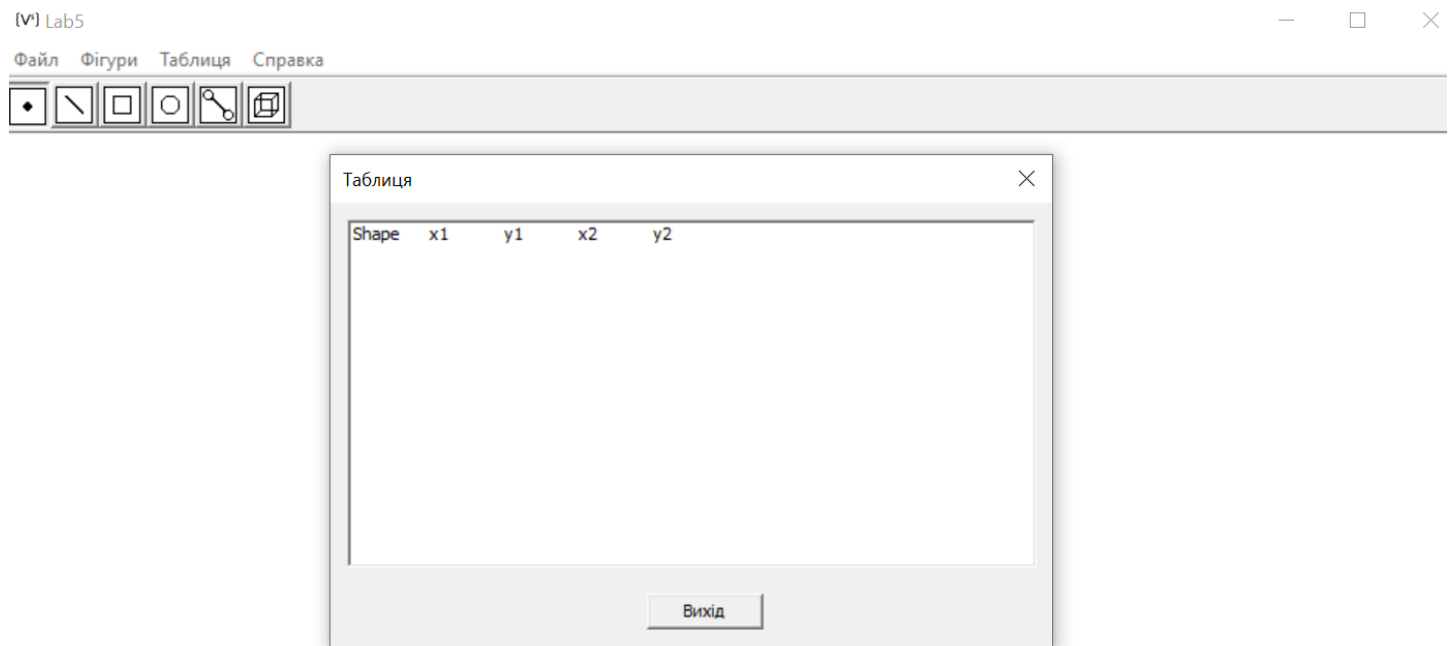
Лінії з кружечками:



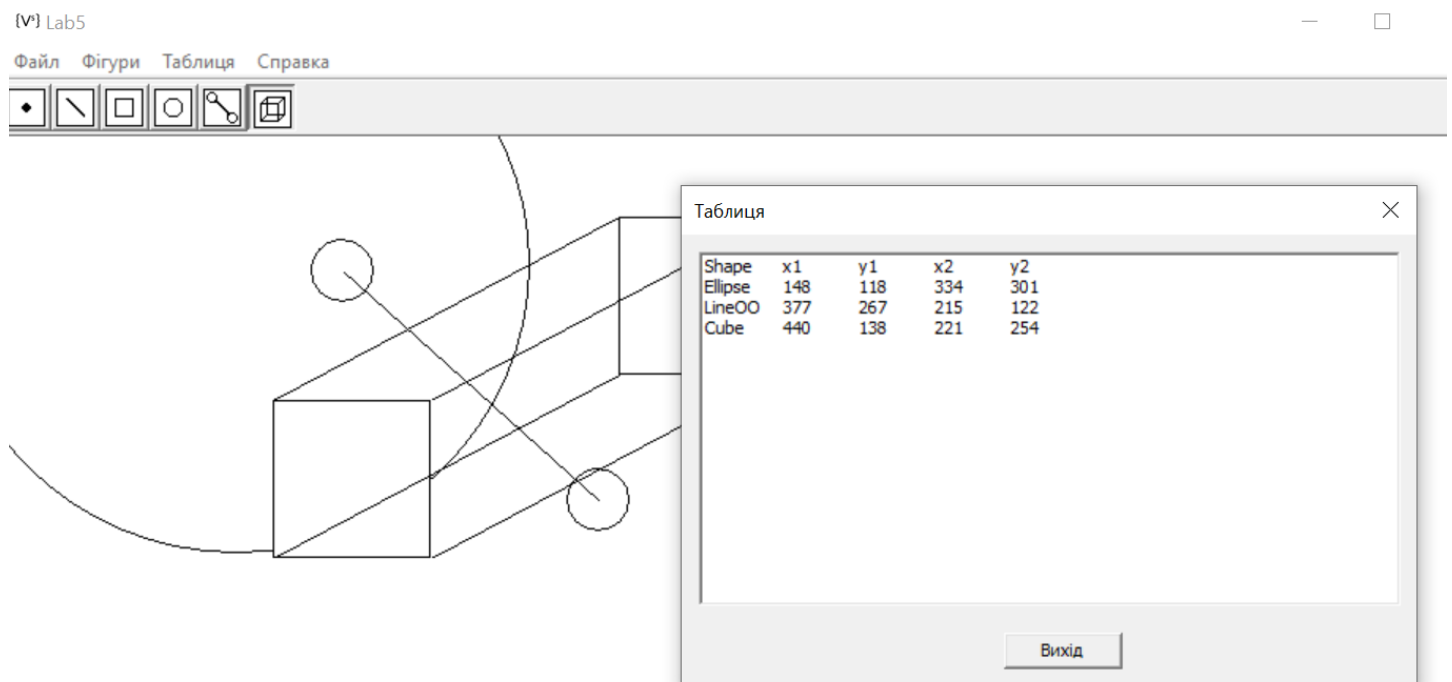
Куби:



Таблиця:



Таблиця з даними:



Контрольні питання

1. Що таке Singleton?

Клас, який забезпечує можливість створення тільки одного екземпляру об'єкта

2. Чим відрізняється класична реалізація Singleton від Singleton Меєрса?

У класичній реалізації є змінна "Instance" та код працює з нею, а в Singleton Меєрса ця змінна знаходиться у функції

3. Як запрограмувати немодальне діалогове вікно?

Потрібно мати ресурс-файл та створити в ньому вікно. Створити функцію, яка буде працювати з цим вікном. Коли користувач взаємодіє з пунктом у меню, то викликається функція, яка створює немодальне вікно

4. Як запрограмувати запис у файл об'єктів - геометричних форм?

Треба відкрити файл. Записати в нього наші дані за допомогою функцій побітового зсуву вліво. Закрити файл. Також в даному випадку можна використати "endl", щоб закінчити введення на конкретному рядку.

5. Покажіть у програмі поліморфізм

My_editor може бути гарним прикладом поліморфізму, адже ми маємо «використання єдиного інтерфейсу для різнотипних сутностей або у використанні однакового символу для маніпуляцій над даними різного типу»

Висновок:

Навчився працювати з файлами. Записувати та видаляти з них дані. Дізнався про немодальні вікна, їх функції, як їх викликати та створювати.