

Лабораторна робота №2

Розробка графічного редактора об'єктів на C++

Мета: Мета роботи – отримати вміння та навички використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів C++, запрограмувавши простий графічний редактор в об'єктно-орієнтованому стилі.

Завдання:

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім'ям **Lab2**.
2. Написати вихідний текст програми згідно варіанту завдання.
3. Скомпілювати вихідний текст і отримати виконуваний файл програми.
4. Перевірити роботу програми. Налогодити програму.
5. Проаналізувати та прокоментувати результати та вихідний текст програми.
6. Оформити звіт.

Порядок виконання роботи та методичні рекомендації

Загальна структура програми

Програма повинна бути модульною. Вихідний текст складається з головного файлу **Lab2.cpp**, модуля **shape_editor** та інших модулів та файлів проекту Win32. Модулі, зокрема, **shape_editor**, повинні роздільно компілюватися у проекті.

У модулі **shape_editor** міститься клас **ShapeObjectsEditor**. Інтерфейс модуля у вигляді оголошення класу **ShapeObjectsEditor** повинен міститися у файлі **shape_editor.h**

```
class ShapeObjectsEditor
{
private:
    ...
public:
    ShapeObjectsEditor(void) ;
    ~ShapeObjectsEditor() ;
    void StartPointEditor(...) ;
    void StartLineEditor(...) ;
    void StartRectEditor(...) ;
    void StartEllipseEditor(...) ;
    void OnLBdown (HWND) ;
    void OnLBup (HWND) ;
    void OnMouseMove (HWND) ;
    void OnPaint (HWND) ;
    void OnInitMenuPopup (HWND, WPARAM) ;    //відповідно варіанту завдання
};
```

Примітка. Позначкою "... " вказане те, що програміст запише для вирішення завдання. Метод **OnInitMenuPopup** потрібен лише для окремих варіантів завдань.

Методи класу **ShapeObjectsEditor** є обробниками основних повідомлень, важливих для редагування:

```
//---файл Lab2.cpp---
#include "shape_editor.h"

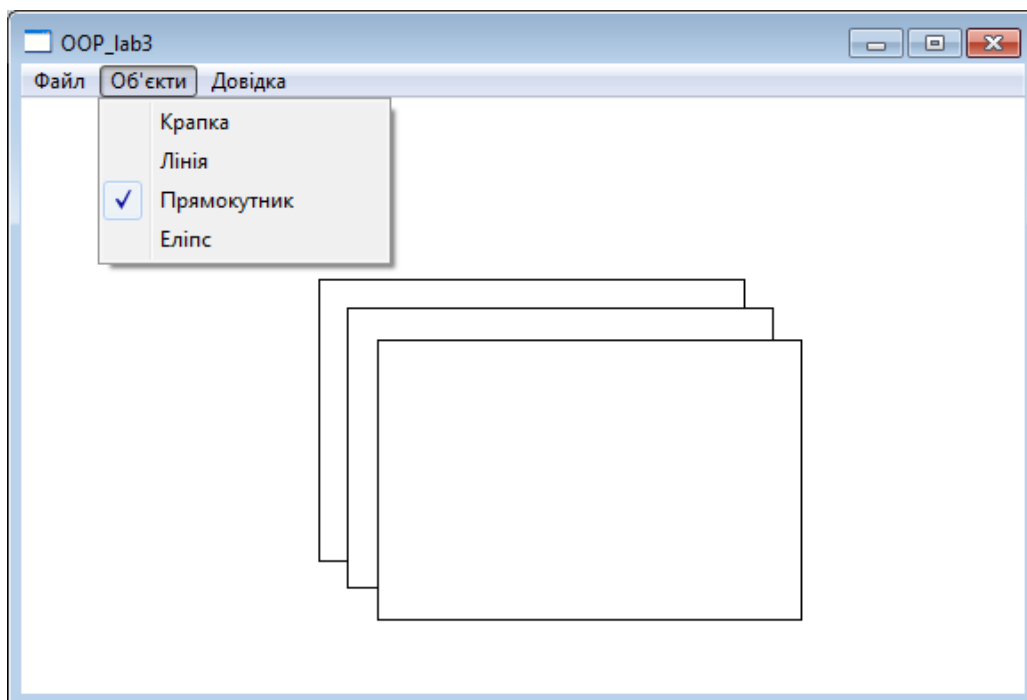
ShapeObjectsEditor Ім'я;    //для варіанту статичного екземпляру редактора

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;

    switch (message)
    {
        case WM_LBUTTONDOWN:    //натиснуто ліву кнопку миші у клієнтській частині вікна
            Ім'я.OnLButtonDown(hWnd);
            break;
        case WM_LBUTTONUP:      //відпущено ліву кнопку миші у клієнтській частині вікна
            Ім'я.OnLButtonUp(hWnd);
            break;
        case WM_MOUSEMOVE:      //пересунуто мишу у клієнтській частині вікна
            Ім'я.OnMouseMove(hWnd);
            break;
        case WM_PAINT:          //потрібно оновлення зображення клієнтської частині вікна
            Ім'я.OnPaint(hWnd);
            break;
        case WM_INITMENUPOPUP:  //позначка пунктів меню - для окремих варіантів завдань
            Ім'я.OnInitMenuPopup(hWnd, wParam);
            break;

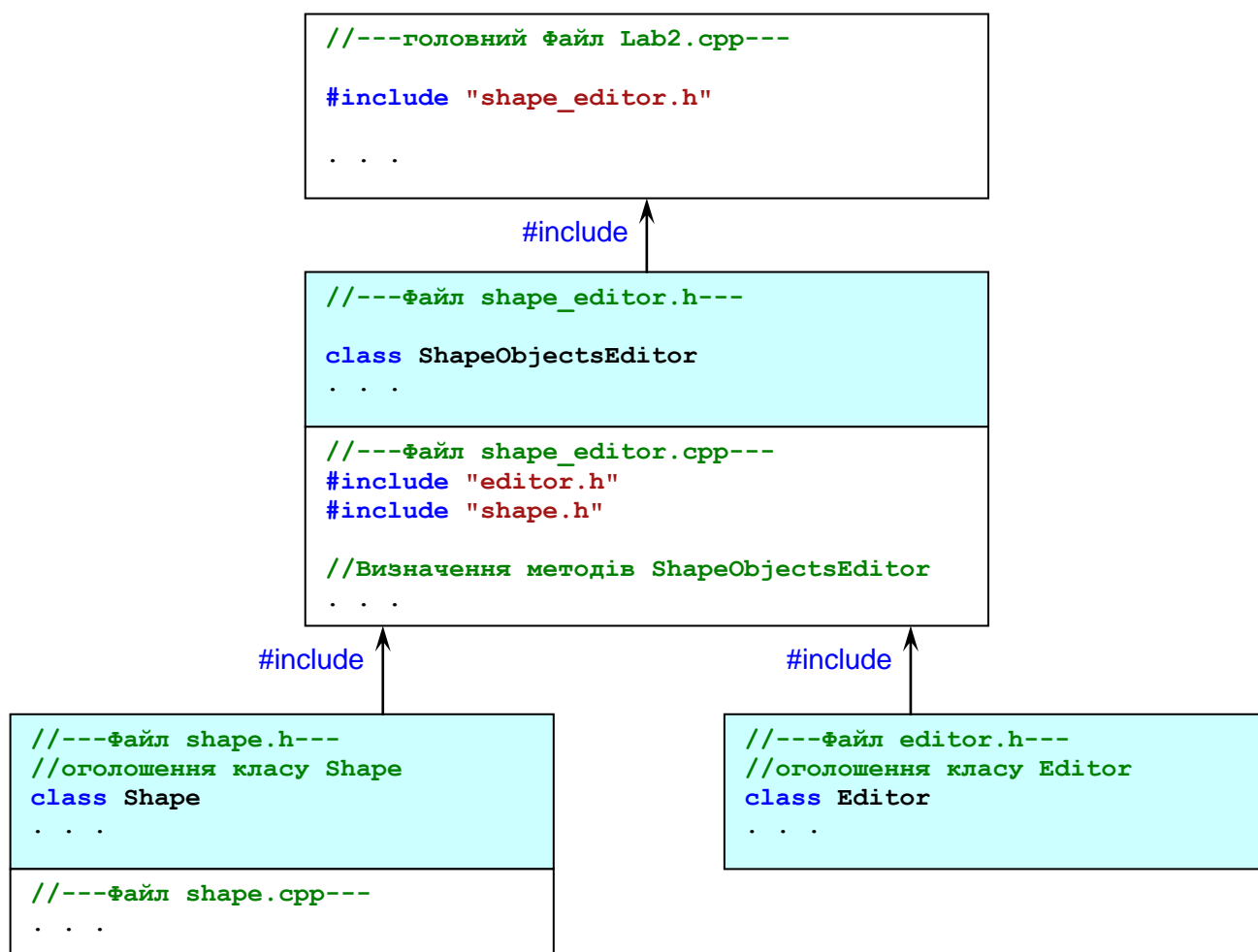
        case WM_COMMAND:
            wmId    = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
                case IDM_POINT:
                    Ім'я.StartPointEditor(...);    //початок вводу точкових об'єктів
                    break;
                case IDM_LINE:
                    Ім'я.StartLineEditor(...);      //початок вводу об'єктів-ліній
                    break;
                case IDM_RECT:
                    Ім'я.StartRectEditor(...);      //початок вводу прямокутників
                    break;
                case IDM_ELLIPSE:
                    Ім'я.StartEllipseEditor(...);   //початок вводу еліпсів
                    break;
                . . .
            }
        }
    }
}
```

Примітка. Ім'я-ідентифікатор статичного об'єкту студент вигадає сам. Для деяких варіантів завдань потрібно буде створити динамічний об'єкт типу **ShapeObjectsEditor**.



Вигляд головного вікна програми. Зараз вводяться об'єкти-прямокутники

Загальну структуру програми можна відобразити наступним чином:



Діаграма залежностей файлів та модулів (**include**-ієрархія)

Базовий клас Shape та його спадкоємці

Ієрархія графічних об'єктів у цій програмі починається з базового абстрактного класу **Shape**:

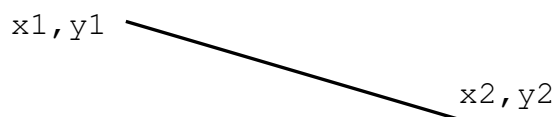
```
//---оголошення класу---
class Shape
{
protected:
    long xs1,ys1,xs2,ys2;
public:
    void Set(long x1,long y1,long x2,long y2) ;
    virtual void Show(HDC) = 0;
};

//---визначення методу Set---
void Shape::Set(long x1,long y1,long x2,long y2)
{
    xs1 = x1;
    ys1 = y1;
    xs2 = x2;
    ys2 = y2;
}
```

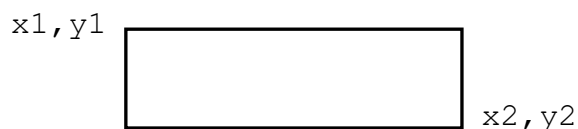
Від класу **Shape** утворюються похідні класи: **PointShape**, **LineShape**, **RectShape**, **EllipseShape**. У кожному похідному класі визначається метод показу **Show** відповідно до геометрії об'єкта.

У класі **PointShape** кожна точка малюється викликом функції
`SetPixel(hdc, x, y, колір);`

У класі **LineShape** лінія (відрізок прямої) малюється викликом функцій
`MoveToEx(hdc, x1, y1, NULL);`
`LineTo(hdc, x2, y2);`



У класі **RectShape** прямокутник малюється викликом функції
`Rectangle(hdc, x1, y1, x2, y2);` (прямокутник із заповненням)
 або чотирма лініями (якщо потрібен тільки контур).



У класі **EllipseShape** еліпс малюється викликом функції
`Ellipse(hdc, x1, y1, x2, y2);` (еліпс із заповненням)
 або `Arc(hdc, x1, y1, x2, y2, 0, 0, 0, 0);` (якщо потрібен тільки контур).



Малювання фігур із заповненням

Функції **Rectangle** та **Ellipse** малюють фігури із заповненням. За умовчанням контур – тонка чорна лінія, заповнення – суцільне білого кольору.

Визначення кольору заповнення можна запрограмувати у такий спосіб

```
HBRUSH hBrush,hBrushOld;

hBrush = (HBRUSH)CreateSolidBrush(colorfill);    //створюється пензль
hBrushOld = (HBRUSH)SelectObject(hdc, hBrush);    //пензль -> у hdc








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

SelectObject(hdc, hBrushOld);                     //відновлюється пензль-попередник
DeleteObject(hBrush);                             //створений знищується
```

Спочатку потрібно створити пензль та помістити його у контекст пристрою (hdc). Далі виклик функцій малювання потрібних одної або декількох фігур. Після того, як такий пензль вже більше не потрібен, або треба створити новий, то спочатку вибирається попередній пензль, а потім створений знищується (інакше будуть витoki пам'яті).

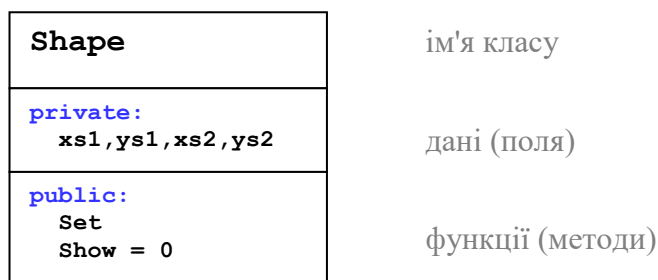
Аргументом функції **CreateSolidBrush** є ціле число, яке визначає колір заповнення. У якості аргументу можна рекомендувати макрос RGB, наприклад, для жовтого кольору заповнення:

```
hBrush = (HBRUSH)CreateSolidBrush(RGB(255,255,0));
```

Колір заповнення		RGB
	білий	RGB(255,255,255) (за умовчанням)
	жовтий	RGB(255,255,0)
	світло-зелений	RGB(0,255,0)
	блакитний	RGB(0,255,255)
	рожевий	RGB(255,0,255)
	померанчевий	RGB(255,128,0)
	сірий	RGB(192,192,192) (у градацій сірого R = G = B)

Діаграма класів. Основні поняття

Діаграма класів показує наявні класи та їхню ієрархію спадкування. Кожний клас відображається прямокутником, у якому записане ім'я класу. Також у цьому прямокутнику записуються імена членів класу. Можна позначати окремо дані-члени та функції-члени (методи). Також можна якось позначати різновиди оголошення членів – чи то `private`, чи то `protected` або `public`.

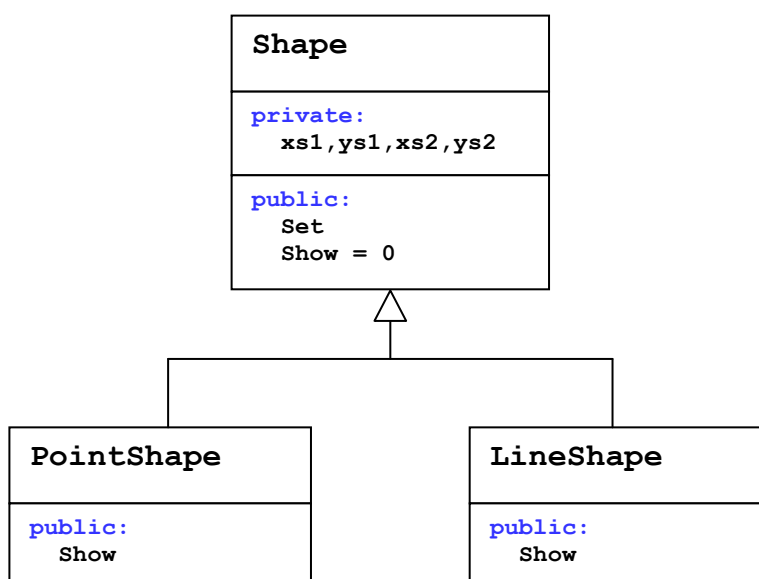


Існують багато способів відображення таких діаграм. Дуже популярною є нотація UML. Дещо інший, але подібний вигляд мають діаграми класів, які можна створити засобами Microsoft Visual Studio. У середовищі Visual Studio C++ діаграма створюється напівавтоматично. Спочатку потрібно створити і додати у проект новий елемент – "Схема класів" або "Class Diagram". А потім за допомогою панелі інструментів (toolbox) вставити потрібні елементи на діаграму. Як саме - це може відрізнятись для різних версій Visual Studio. Зазвичай це достатньо легко. Розберіться у цьому самостійно.

У найпростішому випадку, коли клас усього один, у діаграмі особливого сенсу немає. Діаграма корисна для показу двох та більше класів – тоді вона відображатиме взаємини класів. Наприклад, так

Базовий клас

Похідні класи



Базовий абстрактний клас Editor та його спадкоємці

Ієрархія класів редагування об'єктів різних типів повинна розпочинатися з абстрактного базового класу **Editor**.

```
class Editor
{
public:           //а, можливо, ще потрібен віртуальний деструктор?
    virtual void OnLBdown (HWND) = 0;
    virtual void OnLBup (HWND) = 0;
    virtual void OnMouseMove (HWND) = 0;
    virtual void OnPaint (HWND) = 0;
};
```

Навіщо цей клас потрібен?

По-перше, у ньому фіксуються імена та склад інтерфейсних методів, які необхідно реалізувати. Наприклад, метод **OnLBdown** буде викликатися для обробки повідомлення **WM_LBUTTONDOWN**, метод **OnPaint** – для обробки повідомлення **WM_PAINT**, і тому подібне.

По-друге, від базового класу буде утворена ієрархія класів-спадкоємців, які будуть описувати конкретні дії редагування для кожного типу об'єктів.

По-третє, можна запрограмувати поліморфізм виклику наведених вище інтерфейсних методів класу **Editor** по вказівнику на цей клас.

Примітка. Оскільки в деяких варіантах завдань потрібно ще й обробляти повідомлення **WM_INITMENUPOPUP**, то можливо додати відповідний інтерфейсний метод **OnInitMenuPopup** – проте тут **вимога: базовий клас Editor не змінювати**, а цей метод додати у похідний клас **ShapeEditor** і зробити вказівник вже на похідний клас **ShapeEditor**, вважаючи його базовим для забезпечення поліморфізму. Крім того, мабуть корисно такий похідний клас **ShapeEditor** зробити базовим і тому, що в ньому можна зосередити корисні члени, які враховують специфіку Windows-програм.

```
class ShapeEditor : public Editor
{
protected:
    . . .           //корисні члени, які враховують специфіку Windows-програм
public:
    ShapeEditor(void) ;
    void OnLBdown (HWND) ;
    void OnLBup (HWND) ;
    void OnMouseMove (HWND) ;
    void OnPaint (HWND) ;
    void OnInitMenuPopup (HWND, WPARAM) ;           //додатковий інтерфейсний метод
};
```

Від класу **ShapeEditor** треба утворити ієрархію похідних класів, які своїми методами будуть описувати дії, які потрібно виконувати при вводі різних форм. Так, наприклад, для вводу точкового об'єкту достатньо клікнути лівою кнопкою миші у потрібній точці вікна. А при вводі прямокутника спочатку визначаємо один кут, натискуючи кнопку миші (**OnLButtonDown**), далі пересуваємо курсор в точку іншого кута (**OnMouseMove**) і відпускаємо кнопку (**OnLButtonUp**).

Таким чином, створимо похідні класи:

```
class PointEditor : public ShapeEditor
{
    . . .
};

class LineEditor : public ShapeEditor
{
    . . .
};

class RectEditor : public ShapeEditor
{
    . . .
};

class EllipseEditor : public ShapeEditor
{
    . . .
};
```

Оголосимо вказівник на базовий клас **ShapeEditor**

```
ShapeEditor *pse = NULL;
```

Тоді в обробники потрібних повідомлень можна записувати поліморфні виклики:

```
switch (message)
{
    case WM_LBUTTONDOWN:
        if (pse) pse->OnLButtonDown(hWnd);
        break;
    case WM_PAINT:
        if (pse) pse->OnPaint(hWnd);
        break;
    . . .
    case WM_COMMAND:
        wmId = LOWORD(wParam);
        wmEvent = HIWORD(wParam);
        switch (wmId)
        {
            case IDM_POINT:
                if (pse) delete pse;
                pse = new PointEditor;
                break;
            case IDM_LINE:
                if (pse) delete pse;
                pse = new LineEditor;
                break;
            . . .
        }
    }
}
```


Можна трохи структурувати програмний код. Замість рядків на кшталт

```
case IDM_POINT:
    if (pse) delete pse;
    pse = new PointEditor;
    break;
```

для обробки основних повідомлень записати виклики методів класу **ShapeObjectsEditor**, який згідно вимогам є інтерфейсом для модуля редагування,

```
ShapeObjectsEditor Ім'я;

. . .

case IDM_POINT:
    Ім'я.StartPointEditor(...);
    break;
```

як і наведено на сторінці 2 цього документу.

Цикл відображення об'єктів

Для циклу відображення об'єктів треба використати поліморфізм.

Якщо оголосити масив вказівників на базовий клас Shape, наприклад, так:

```
Shape *pcshape[MY_SHAPE_ARRAY_SIZE]; //статичний масив вказівників
```

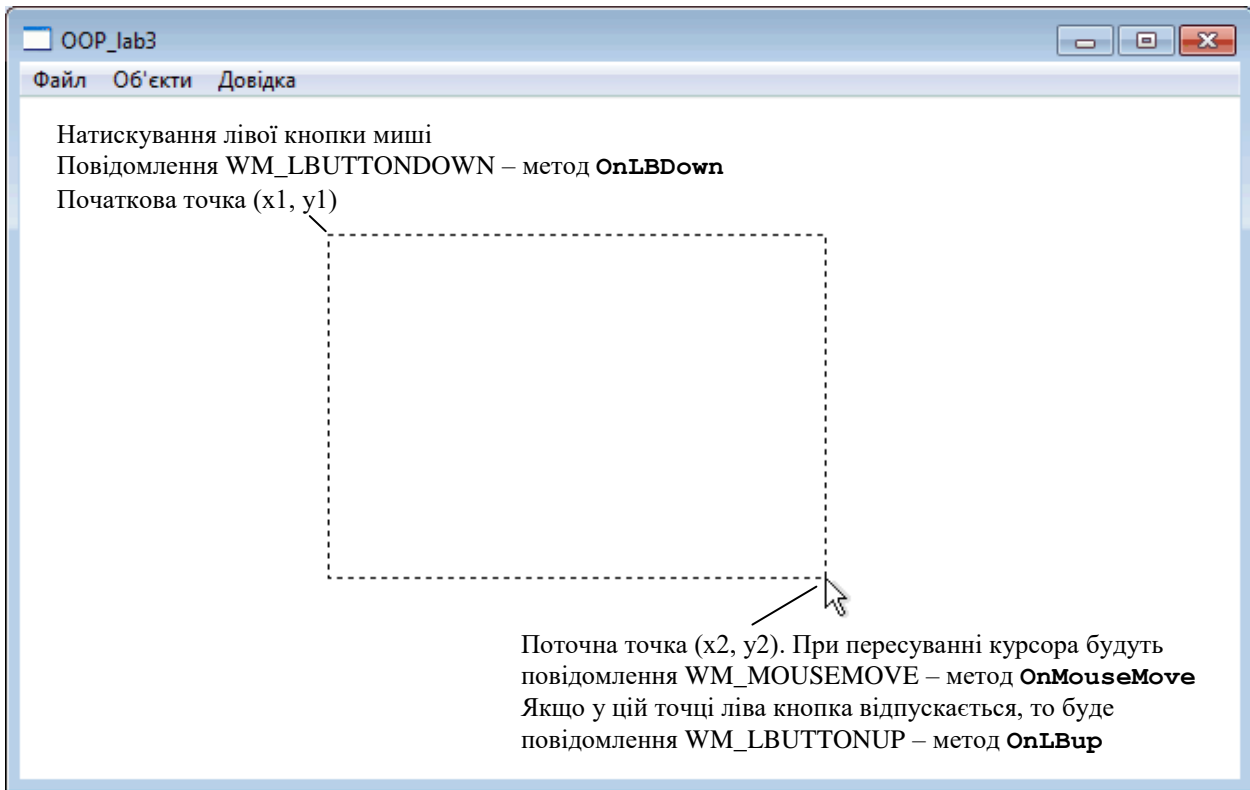
то вивід усіх об'єктів обробником повідомлень WM_PAINT можна зробити так:

```
void ShapeEditor::OnPaint(HWND hWnd)
{
    PAINTSTRUCT ps;
    HDC hdc;

    hdc = BeginPaint(hWnd, &ps);
    for (int i=0; i<MY_SHAPE_ARRAY_SIZE; i++)
        if (pcshape[i])
            pcshape[i]->Show(hdc);
    EndPaint(hWnd, &ps);
}
```

"Гумовий" слід

При вводі об'єктів інколи замість їхніх реальних зображень використовують спрощені образи, які показують лише контури об'єктів. Цим також підкреслюється, що зараз відбувається саме введення об'єкту. Можна побачити, наприклад, пунктирний "гумовий" слід, який розтягується від початкової точки до поточної точки курсору.



Як запрограмувати рухомий "гумовий" слід? Розглянемо це на прикладі об'єкта-відрізка лінії. Уявимо, що курсор миші пересунуто у початкову точку лінії. Натискається ліва кнопка миші (і далі тримається натиснутою). Нашій програмі надсилається повідомлення WM_LBUTTONDOWN. Обробником цього повідомлення є наш метод OnLBdown. Цей метод повинен записати у якісь структури даних координати початкової точки. Зробити це можна, наприклад, так:

```
POINT pt;

GetCursorPos(&pt);
ScreenToClient(hWnd, &pt);
xstart = xend = pt.x;    //кудись записуємо координати початкової точки
ystart = yend = pt.y;
```

Далі пересувається курсор миші і після кожного руху нашій програмі надсилається повідомлення WM_MOUSEMOVE, яке обробляється методом OnMouseMove. Тут потрібно малювати "гумовий" слід. Рекомендується така послідовність дій

- спочатку стирається слід для попереднього розташування курсору
- потім обчислюються нові координати курсору і для них малюється слід

```
POINT pt;

Стирається відрізок(xstart,ystart, xend,yend);

GetCursorPos(&pt);
ScreenToClient(hWnd, &pt);
xend = pt.x;      //координати поточної точки курсору
yend = pt.y;

Малюється відрізок(xstart,ystart, xend,yend);
```

Як стерти попереднє зображення "гумового" сліду, щоб не пошкодити зображення інших об'єктів? Найпростіше зробити це для лінійного сліду. Для цього лінію можна малювати у режимі комбінування R2_NOTXORPEN з попереднім зображенням.

Як зробити "гумовий" слід пунктирним потрібного кольору? За умовчанням усі лінії чорні, суцільні. Щоб змінити стиль, потрібно створити перо, намалювати щось, а коли перо стає непотрібним, його обов'язково треба знищити:

```
HPEN hPenOld,hPen;

hPen = CreatePen(PS_DOT, 1, Колір);
hPenOld = (HPEN)SelectObject(hdc, hPen);

Малюється лінія (лінії)

SelectObject(hdc, hPenOld);
DeleteObject(hPen);
```

Якщо об'єднати усі відомості, то можна порекомендувати приблизно таку обробку повідомлення WM_MOUSEMOVE:

```
POINT pt;
HPEN hPenOld,hPen;
HDC hdc;

hdc = GetDC(hWnd);      //отримуємо контекст вікна для малювання
SetROP2(hdc,R2_NOTXORPEN);
hPen = CreatePen(PS_DOT, 1, 0);
hPenOld = (HPEN)SelectObject(hdc, hPen);

Малюються лінії "гумового" сліду попереднього розташування курсору

GetCursorPos(&pt);
ScreenToClient(hWnd, &pt);
xend = pt.x;      //координати поточної точки курсору
yend = pt.y;

Малюються лінії "гумового" сліду для поточного розташування курсору

SelectObject(hdc, hPenOld);
DeleteObject(hPen);
ReleaseDC(hWnd,hdc);    //закриваємо контекст вікна
```

Запис об'єкта у масив Shape

Після того, як ми перемістили курсор миші у потрібну кінцеву точку, можна відпускати ліву кнопку миші. Нашій програмі надсилається повідомлення WM_LBUTTONDOWN, яке оброблятиме метод **OnLButtonDown**. Цей метод обчислить координати кінцевої точки і запише об'єкт у масив вказівників на об'єкти типу Shape.

Після запису об'єкту можна викликом `InvalidateRect(hWnd, NULL, TRUE)` спонукати нашу програму перемалювати зображення у головному вікні – Windows надішле повідомлення WM_PAINT, яке буде оброблено нашим методом **OnPaint**.

Позначка поточного режиму редагування

Потрібно, щоб користувач бачив, який тип об'єктів зараз вибрано для вводу.

Позначку потрібного пункту меню можна запрограмувати обробником повідомлення WM_INITMENUPOPUP – **OnInitMenuPopup**. Позначка пункту меню робиться викликом функції `CheckMenuItem`. Наприклад, якщо потрібно позначити пункт меню "Еліпс" і зняти позначку з іншого пункту, то це можна запрограмувати так:

```

HMENU hMenu, hSubMenu;

hMenu=GetMenu(hWnd);
hSubMenu=GetSubMenu(hMenu, 1);    //POPUP-меню Об'єкти
if ((HMENU)wParam == hSubMenu)
{
    CheckMenuItem(hSubMenu, IDM_POINT, MF_UNCHECKED);    //зняти позначку
    CheckMenuItem(hSubMenu, IDM_LINE, MF_UNCHECKED);
    CheckMenuItem(hSubMenu, IDM_RECT, MF_UNCHECKED);
    CheckMenuItem(hSubMenu, IDM_ELLIPSE, MF_CHECKED);    //позначити цей пункт
}

```

Як саме запрограмувати такі позначки, студент вирішує самостійно

Позначку поточного типу об'єкту, що вводиться, можна зробити по-іншому. Для деяких варіантів завдань цієї лаб. роботи пропонується виводити текст у заголовку вікна програми. Це легко зробити викликом функції `SetWindowText`, наприклад

```

SetWindowText(hWnd, "Режим вводу еліпсів");

```

Варіанти завдань та основні вимоги

1. Для усіх варіантів завдань необхідно дотримуватися вимог та положень, викладених вище у порядку виконання роботи та методичних рекомендаціях.

2. У звіті повинна бути схема успадкування класів – **діаграма класів**

3. Документи звіту – тексти, діаграми, схеми тощо оформлювати у електронному форматі так, щоб їх легко було сприймати у надрукованому звіті. Забороняється текст або графіка "світле на світлому фоні" або "темне на темному фоні". Тільки чорний текст та чорні лінії на білому фоні. Оформлення звіту впливатиме на оцінку.

4. Для вибору типу об'єкту в графічному редакторі Lab2 повинно бути меню "Об'єкти" з чотирма підпунктами. Меню "Об'єкти" повинно бути праворуч меню "Файл" та ліворуч меню "Довідка". Підпункти меню "Об'єкти" містять назви українською мовою геометричних форм – так, як наведено вище у порядку виконання роботи та методичних рекомендаціях. Геометричні форми згідно варіанту завдання.

5. Для вибору варіанту використовується Ж – номер студента в журналі. Позначка mod означає залишок від ділення.

6. Масив вказівників для динамічних об'єктів типу Shape

- динамічний масив `Shape **pcshape;`
- статичний масив `Shape *pcshape[N];`

кількість елементів масиву вказівників як для статичного, так і динамічного має бути $N = Ж + 100$.

Динамічний масив обирають студенти, у яких $(Ж \bmod 3 = 0)$. Решта студентів – статичний масив.

7. "Гумовий" слід при вводі об'єктів

- суцільна лінія чорного кольору для студентів, у яких $(Ж \bmod 4 = 0)$
- суцільна лінія червоного кольору для студентів, у яких $(Ж \bmod 4 = 1)$
- суцільна лінія синього кольору для студентів, у яких $(Ж \bmod 4 = 2)$
- пунктирна лінія чорного кольору для студентів, у яких $(Ж \bmod 4 = 3)$

8. Чотири геометричні форми (крапка, лінія, прямокутник, еліпс) можуть мати наступні різновиди вводу та відображення.

8.1. Прямокутник

Ввід прямокутника:

- по двом протилежним кутам для студентів, у яких $(Ж \bmod 2 = 0)$
- від центру до одного з кутів для $(Ж \bmod 2 = 1)$

Відображення прямокутника:

- чорний контур з білим заповненням для $(Ж \bmod 5 = 0)$
- чорний контур з кольоровим заповненням для $(Ж \bmod 5 = 1 \text{ або } 2)$

- чорний контур прямокутника без заповнення для ($J \bmod 5 = 3$ або 4)

Кольори заповнення прямокутника:

- жовтий для ($J \bmod 6 = 0$)
- світло-зелений для ($J \bmod 6 = 1$)
- блакитний для ($J \bmod 6 = 2$)
- рожевий для ($J \bmod 6 = 3$)
- померанчевий для ($J \bmod 6 = 4$)
- сірий для ($J \bmod 6 = 5$)

8.2. Еліпс

Ввід еліпсу:

- по двом протилежним кутам охоплюючого прямокутника для ($J \bmod 2 = 1$)
- від центру до одного з кутів охоплюючого прямокутника для ($J \bmod 2 = 0$)

Відображення еліпсу:

- чорний контур з білим заповненням для ($J \bmod 5 = 1$)
- чорний контур з кольоровим заповненням для ($J \bmod 5 = 3$ або 4)
- чорний контур еліпсу без заповнення для ($J \bmod 5 = 0$ або 2)

Кольори заповнення еліпсу:

- жовтий для ($J \bmod 6 = 1$)
- світло-зелений для ($J \bmod 6 = 2$)
- блакитний для ($J \bmod 6 = 3$)
- рожевий для ($J \bmod 6 = 4$)
- померанчевий для ($J \bmod 6 = 5$)
- сірий для ($J \bmod 6 = 0$)

9. Позначка поточного типу об'єкту, що вводиться

- в меню (метод OnInitMenuPopup) для студентів ($J \bmod 2 = 0$)
- в заголовку вікна для ($J \bmod 2 = 1$)

10. Приклад вибору варіанту. Для 9-го студента у списку ($J = 9$) буде:

- динамічний масив для Shape ($9 \bmod 3 = 0$) обсягом 109 об'єктів
- "гумовий" слід ($9 \bmod 4 = 1$) – суцільна лінія червоного кольору
- прямокутник:
 - ввід від центру до одного з кутів ($9 \bmod 2 = 1$)
 - чорний контур прямокутника без заповнення ($9 \bmod 5 = 4$)
- еліпс:
 - по двом протилежним кутам охоплюючого прямокутника ($9 \bmod 2 = 1$)
 - чорний контур з кольоровим заповненням ($9 \bmod 5 = 4$)
 - колір заповнення: блакитний ($9 \bmod 6 = 3$)
- позначка поточного типу об'єкту: в заголовку вікна ($9 \bmod 2 = 1$)

Контрольні запитання

1. Що таке поліморфізм?
2. Обробку яких повідомлень потрібно виконувати для вводу об'єктів?
3. Що таке абстрактний клас і скільки їх у цій програмі?
4. Як намалювати лінії та фігури потрібного кольору та стилю?
5. Як відобразити щось у вікні програми?

У ході захисту-прийняття роботи викладач може також запитувати інше, що стосується виконання роботи.

Зміст звіту

1. Титульний аркуш
2. Варіант завдання
3. Вихідний текст головного файлу .cpp (фрагменти, що ілюструють власний код), та вихідні тексти власних модулів
4. Схеми, діаграми згідно завданню
5. Ілюстрації роботи програми (скріншоти)
6. Висновки