문제 정의

과제 5(9장 10번문제)에서 하였던 여러 도형의 삽입, 삭제, 도형 출력 및 종료의 기능이 있는 그래픽 에디터를 vector<Shape*> v;를 이용하여 구현하는 것이고, 생성된 도형의 객체를 이 v에 삽입하여 관리하는 프로그램을 작성하는 것이다. 또한, 동적으로 생성된 객체이기 때문에 소멸자를 사용하여 효율적인 관리를 하여야 한다.

문제 해결 방법

1. vector<Shape*>를 사용하여 도형 객체를 동적으로 생성 및 관리

```
class GraphicEditor {
    vector<Shape*> shapes;
public:
    GraphicEditor() {
```

2. 벡터에서 삽입(push_back), 삭제(erase) 사용.

```
            return;
        }
        shapes.push_back(newShape);
    }
```

```
    }
    delete shapes[index];
    shapes.erase(shapes.begin() + index);
}
```

아이디어 평가

1. vector<Shape *>를 사용하여 도형 객체의 동적 생성으로 인한 효율적 관리가 가능하다.

>> 삭제와 삽입이 간단해지고, 연결 리스트 방식보다 코드가 직관적이며 가독성의 상승으로 이어진다.

```
~GraphicEditor() {
    for (Shape* shape : shapes) {
        delete shape;
    }
}

void create(int num){
    Shape* newShape = nullptr;
    switch (num) {
    case 1:
        newShape = new Line();
        break;
    case 2:
        newShape = new Circle();
        break;
    case 3:
        newShape = new Rectangle();
        break;
    default:
        cout << "잘못된 도형입니다." << endl;
        return;
    }
    shapes.push_back(newShape);
}

void del(int index) {
    if (index < 0 || index >= shapes.size()) {
        cout << "잘못된 숫자입니다." << endl;
        return;
    }
    delete shapes[index];
    shapes.erase(shapes.begin() + index);
}

void showALL() const {
    if (shapes.empty()) {
        cout << "도형이 존재하지 않음." << endl;
        return;
    }
    for (int i = 0; i < shapes.size(); ++i) {
        cout << i << ": ";
        shapes[i]->paint();
    }
}
```

위의 스크린샷은 shapes를 사용하여 확실히 간결하고 가독성이 좋아진 모습을 보여준다.

문제를 해결한 키 아이디어 또는 알고리즘 설명

1. Shape 클래스를 추상 클래스로 설계함 >> Line, Circle, Rectangle이 이를 상속받음 >> 다형성을 구현하였고 이는 곧 모든 도형 객체를 Shape* 포인터로 쉽게 관리할 수 있다.

```
class Shape {
protected:
    virtual void draw() = 0;
public:
    virtual ~Shape() {}
    void paint() { draw(); }
};
```

2. vector<Shape *>를 활용하여 도형을 효율적 저장 및 관리, 벡터는 크기가 동적으로 확장되며 push_back, erase를 통하여 도형을 쉽게 추가 또는 삭제가 가능하다.

```
void create(int num){
    Shape* newShape = nullptr;
    switch (num) {
    case 1:
        newShape = new Line();
        break;
    case 2:
        newShape = new Circle();
        break;
    case 3:
        newShape = new Rectangle();
        break;
    default:
        cout << "잘못된 도형입니다." << endl;
        return;
    }
    shapes.push_back(newShape);
}
```

위 그림은 new를 이용하여 도형 객체를 동적으로 생성하였고, push_back을 이용하여 추가하였다.

```
void del(int index) {
    if (index < 0 || index >= shapes.size()) {
        cout << "잘못된 숫자입니다." << endl;
        return;
    }
    delete shapes[index];
    shapes.erase(shapes.begin() + index);
}
```

위 그림은 erase를 사용하여 벡터의 해당 위치에서 객체 제거 및 delete로 메모리 해제를 하였다.

3. ~GraphicEditor()라는 소멸자를 사용하여, 벡터에 저장된 모든 도형 객체를 프로그램 종료 전에 해제하도록 설정하였다.

```
~GraphicEditor() {
    for (Shape* shape : shapes) {
        delete shape;
    }
}
```