과제 6

202104183 김현중

문제 정의)

지난번 과제 5의 그래픽 에디터를 vector<Shape*> v;를 이용하여 콘솔 바탕으로 만드는 것이다. 프로그램의 구성은 저번과 같고 기능 또한 같다.

문제 해결 방법)

지난번과 다르게 vector을 사용해서 만들기 때문에 지난번에 사용한 주소를 next로 넘기며 할당해서 하던 함수들을 모두 고쳐야 한다. 함수를 저장하는 것은 vector의 컨테이너에 저장되므로 vector을 사용하면 주소를 넘기며 값을 할당할 필요가 없어진다. 저번에 도형을 저장하던 함수에는 push_back을 이용해서 추가하고 도형을 삭제하던 함수에는 earse를 사용하여 없애도록 한다. 도형을 모두 출력하는 것은 at를 사용한다.

아이디어 평가)

함수를 저장하는 것은 위에 써있는 대로 push_back을 해주고 break를 하면 되지만 삭제하는 것은 it을 사용해 인덱스를 지정해주어야한다. 그러기 위해서는 for문을 사용하여 입력받은 인덱스만큼 it++을 해주어 삭제하려는 도형을 it으로 가리킨다. 그 후 earse(it)을 하여 삭제한다. 다음으로는 도형을 모두 보여주는 것인데 이는 for문으로 v.size만큼 반복하여 v.at(i)->paint로 출력해주면 된다.

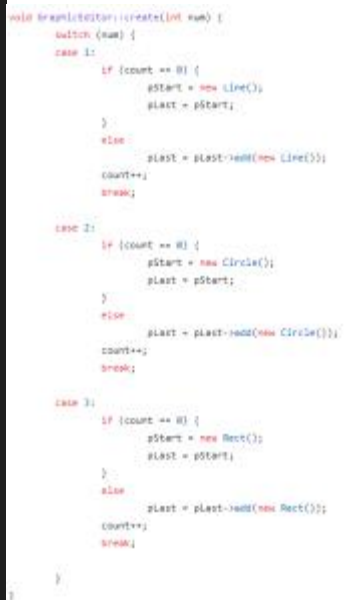
문제를 해결한 키 아이디어 또는 알고리즘 설명)

이번 과제는 저번 과제5와 같은 코드가 많아서 수정된 코드들만 설명하겠다.

```
#pragma once

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

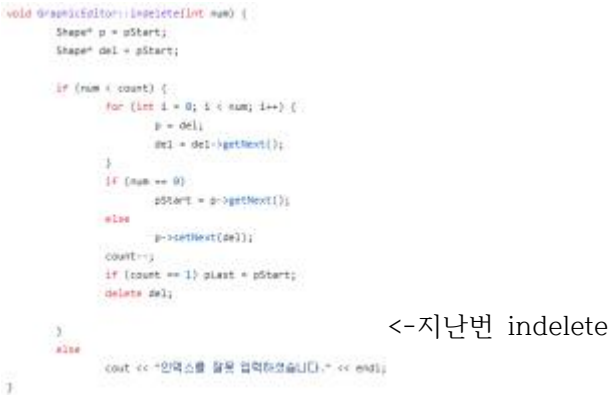
우선 Shape의 헤더파일이다. 저번에 next를 사용하기 위한 함수들을 모두 삭제하였다.



<-지난번 create

다음으로는 GraphicEdior의 선언부이다.

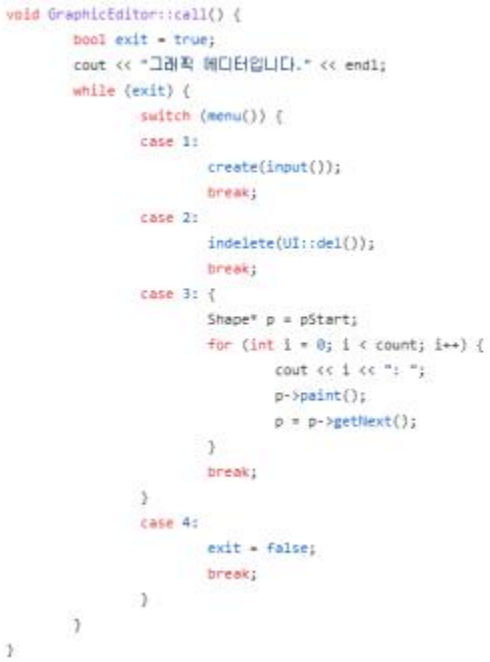
먼저 create함수부터 보겠다. switch case문은 그대로 사용했지만 case 안의 구문들이 모두 삭제되었다. 저번에는 count, pStart를 이용하여 처음 생성하면 pStart=new Line()을 하여 만들고 아니면 add함수로 만들었다면 이번에는 그냥 v.push_back(new Line())을 사용하여 vector의 컨테이너에 저장하였다. 지난번과 비교했을 때 매우 간결해진 것을 볼 수 있다.



<-지난번 indelete

다음은 indelete이다. 이번 그래픽 에디터의 indelete에는 it을 v.begin으로 설정하고 for문을 사용하여 num만큼 it++을 해주어 it을 삭제하려는 도형을 가리키게 하였다. 그리고 it=v.erase(it)으로 삭제하면 되지만 단순히 이렇게 삭제시키면 도형의 메모리가 삭제되지 않고 vector에서만 삭제되기 때문에 Shape* del을 만들어주고 삭제하기 전에 it이 가르키고 있는 주소를 del에 저장한다. 그리고 vector가 이 주소를 컨테이너로부터 삭제하면 del을 삭제하여 본래 메모리 또한 삭제해 버린다. 이 또한 지난 번의 indelete 함수와 비교하면 간결해진 것을 알 수 있다.

```
void GraphicEditor::call() {
    int i = 0;
    bool exit = true;
    cout << "그래픽 에디터입니다." << endl;
    while (exit) {
        switch (menu()) {
        case 1:
            create(input());
            break;
        case 2:
            indelete(UI::del());
            break;
        case 3:
            for (int i = 0; i < v.size(); ++i) {
                cout << i << ": ";
                v.at(i)->paint();
            }
            break;
        case 4:
            exit = false;
            break;
        }
    }
}
```



```
void GraphicEditor::call() {
        bool exit = true;
        cout << "그래픽 에디터입니다." << endl;
        while (exit) {
                switch (menu()) {
                case 1:
                        create(input());
                        break;
                case 2:
                        indelete(UI::del());
                        break;
                case 3: {
                        Shape* p = pStart;
                        for (int i = 0; i < count; i++) {
                                cout << i << ": ";
                                p->paint();
                                p = p->getNext();
                        }
                        break;
                }
                case 4:
                        exit = false;
                        break;
                }
        }
}
```

마지막으로 call함수이다. 1, 2번은 위에서 다루었고 4번은 같다.
3번은 본래 pStart를 가져와서 pLast까지 이어진 것을 모두 출력했다. 이번에도 같은 방식으로 했지만 vector을 이용하여 v.at(i)->paint()를 사용했다.

저번과 함수들의 기능은 전체적으로 비슷하다 지난번은 주소를 이용하여 next로 넘기며 하나하나 도형들을 엮었다면 이번에는 vector을 이용해 이미 만들어져 있는 vector 컨테이너에 저장을 해서 지난번처럼 주소를 일일이 컨트롤 하지 않아도 되는 편리함이 있었다.