일단 문제를 보고 나서 그래픽 편집기가 하는 일을 정의해봤을 때

1. 선, 원 그리고 사각형을 선택해 목록에 추가
2. 삭제가능
3. 내가 넣었던 것들을 볼 수 있음
4. 종료할 수 있다

이렇게 크게 4가지로 정의했습니다.

그 후 Shape라는 Class를 만들고 파생Class인 Line, Circle, Rectangle를 만들었습니다.

Shape에서 draw라는 가상 함수를 만들었고 파생 Class에서 상속받아 재정의 해서 쓰게 했습니다.

Shape.h ↓

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

Shape.cpp ↓

```
Shape::Shape() {next = NULL;}
Shape::~Shape() {}
Shape* Shape::add(Shape* p) {
        this->next = p;
        return p;
}
void Shape::paint() {draw();}
Shape* Shape::getNext() {
return next;}
```

Line.cpp ↓

```
void Line::draw() {

        cout << "Line" << endl;

}
```

그 후 UI Class를 만들어서 각 원하는 메뉴와 도형, 없애고 싶은 인덱스 값을 받아 return하게 만들었습니다.

UI.h

```
class UI {
public:
	int Menu();
	int AddShape();
	int DeleteIndex();
};
```

UI.cpp

```
int UI::Menu() {
	int choice;
	cout << "삽입:1, 삭제:2, 모두보기:3, 종료:4 >>";
	cin >> choice;
	return choice;
}

int UI::AddShape() {
	int choice;
	cout << "선:1, 원:2, 사각형:3 >>";
	cin >> choice;
	return choice;
}

int UI::DeleteIndex() {
	int choice;
	cout << "삭제하고자 하는 도형의 인덱스 >>";
	cin >> choice;
	return choice;
}
```

그 후, GraphicEditor Class를 만들고 UI에서 return한 값들을 받아 원 하는 행동을 하게 하고 (Ex) 도형 추가, 삭제 등) 함수들을 새로 만들어 그 행동에 맞는 코드를 짰습니다.

```
class GraphicEditor {
        Shape* pStart;
        Shape* pLast;
public:
        GraphicEditor();
        void insertItem(int type);
        void deleteItem(int index);
        void show();
        void RunEditor();
};
```

```
#include <iostream>
#include "GraphicEditor.h"

using namespace std;

GraphicEditor::GraphicEditor() {
        pStart = pLast = NULL;
}

void GraphicEditor::RunEditor() {
        UI ui;
        cout << "그래픽 에디터입니다." << endl;
        int menu, index, type;
        while (true) {
                menu = ui.Menu();
                switch (menu) {
                case 1:
                        type = ui.AddShape();
                        insertItem(type);
                        break;
                case 2:
                        index = ui.DeleteIndex();
                        deleteItem(index);
                        break;
                case 3:
                        show();
                        break;
                default:
                        return;
                }
        }
}

void GraphicEditor::insertItem(int type) {
        Shape* p = NULL;
        switch (type) {
        case 1:
                p = new Line();
                break;
```

```
        case 2:
                p = new Circle();
                break;
        case 3:
                p = new Rectangle();
                break;
        default:
                break;
        }
        if (pStart == NULL) {
                pStart = p;
                pLast = p;
                return;
        }
        pLast->add(p);
        pLast = pLast->getNext();
}

void GraphicEditor::deleteItem(int index) {
        Shape* pre = pStart;
        Shape* tmp = pStart;
        if (pStart == NULL) {
                cout << "도형이 없습니다!" << endl;
                return;
        }
        int i = 0;
        while (tmp != NULL && i < index) {
                pre = tmp;
                tmp = tmp->getNext();
                i++;
        }
        if (tmp == NULL) {
                cout << "인덱스가 잘못되었습니다!" << endl;
                return;
        }
        if (tmp == pStart) {
                pStart = tmp->getNext();
                delete tmp;
        }
        else {
                pre->add(tmp->getNext());
                delete tmp;
        }
}

void GraphicEditor::show() {
        Shape* tmp = pStart;
        int i = 0;
        while (tmp != NULL) {
                cout << i++ << ": ";
                tmp->paint();
                tmp = tmp->getNext();
        }
}
```

즉, 코드를 Shape Class를 기반으로 만들었으며 next 포인터를 사용과 pStart와 pLast 포인터를 사용해서 리스트를 쉽게 관리할 수 있게 만들었으며 UI와 GraphicEditor에 분리로 유지보수의 확장성을 높였습니다.