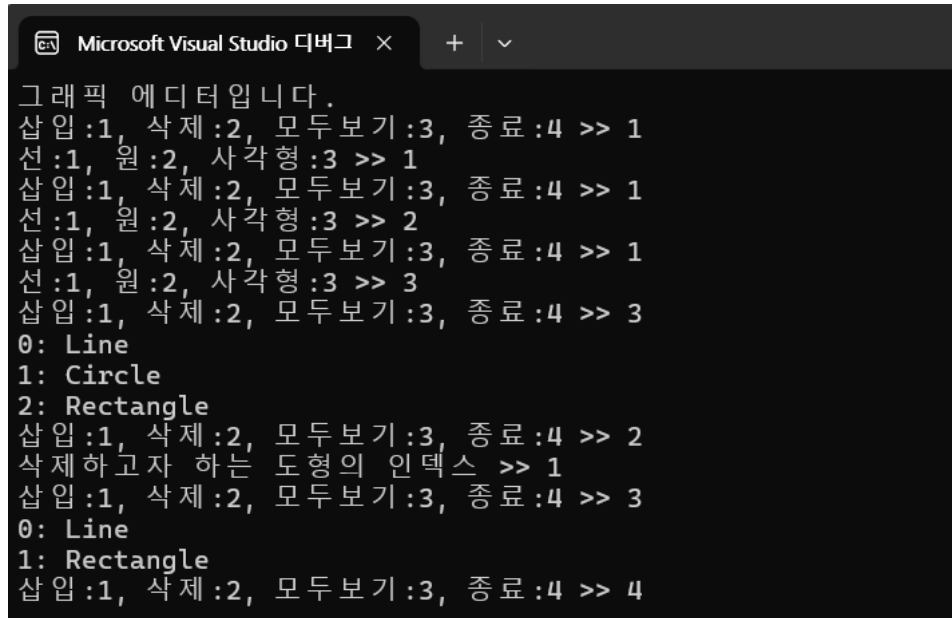


객체지향프로그래밍 과제#5

소프트웨어전공 202284012 김주원 / 소프트웨어전공 202304012 김도연

1. 소스 수행 결과 화면



```
Microsoft Visual Studio 디버그 x + v
그래픽 에디터입니다.
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 1
선:1, 원:2, 사각형:3 >> 1
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 1
선:1, 원:2, 사각형:3 >> 2
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 1
선:1, 원:2, 사각형:3 >> 3
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 3
0: Line
1: Circle
2: Rectangle
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 2
삭제하고자 하는 도형의 인덱스 >> 1
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 3
0: Line
1: Rectangle
삽입:1, 삭제:2, 모두보기:3, 종료:4 >> 4
```

2. 문제 정의

이 문제는 사용자로 하여금 선, 원, 사각형 등의 도형을 추가, 삭제, 표시할 수 있는 그래픽 에디터를 구현한다. 도형은 Shape 추상 클래스를 상속받아 개별적으로 정의되며 연결 리스트 형태로 관리된다. 사용자가 도형을 추가하면 새로운 도형 객체가 생성되어 리스트 끝에 연결되고 삭제 시에는 해당 인덱스의 도형을 제거하며 메모리를 해제한다. GraphicEditor 클래스는 도형 리스트의 시작과 끝을 관리하며 이를 통해 도형 삽입과 삭제를 처리한다. show 기능은 리스트를 순회하며 도형의 종류와 인덱스를 출력한다. UI 클래스는 정적 메서드를 통해 사용자로 부터 메뉴, 도형 타입, 삭제할 인덱스 등을 입력받아 처리한다. 프로그램은 run 메서드를 통해 도형 관리 작업을 반복적으로 수행하며 종료 시 메모리를 정리한다.

3. 문제 해결 방법

<Shape 클래스 선언부 - Shape.h>

1. 기본 로직

Shape 클래스는 도형을 표현하는 추상 클래스이며 각 도형(Line, Circle, Rectangle)은

이를 상속받아 개별적으로 draw() 메서드를 구현하고 Shape* next 포인터를 통해 도형 객체들이 연결 리스트 형태로 관리된다. 도형은 add() 메서드를 통해 리스트에 추가되며 paint() 메서드를 호출하면 현재 도형의 draw() 메서드를 실행해 화면에 그린다. getNext() 메서드는 리스트를 순회하며 다음 도형으로 이동할 수 있도록 연결을 관리한다.

2. 헤더 파일

#ifndef SHAPE_H, #define SHAPE_H, #endif를 사용하여 구성된 헤더 가드는 헤더 파일이 여러 번 포함되더라도 컴파일 에러가 발생하지 않도록 방지한다.

3. 함수 구현

Shape() (생성자)

- 기본 생성자로, **Shape 객체를 초기화하며 next 포인터를 nullptr로 설정**
- 도형 객체 간의 연결을 초기화하기 위해 사용

~Shape() (소멸자)

- 가상 소멸자이며 **상속받는 클래스에서 동적 메모리를 해제하기 위해 선언됨**
- Shape와 이를 상속받는 객체를 동적으로 할당하고 삭제할 때 메모리 누수를 방지함

add(Shape* p)

- p로 전달된 **새로운 도형을 현재 도형의 next에 순차적으로 연결하여 연결 리스트 형태로 도형을 관리하는 메서드**

paint()

- **draw() 메서드를 호출하여 현재 도형의 정보를 출력함**
- 상속받은 각 도형 클래스에서 구현된 draw()를 실행함으로써 **다형성을 제공함**

Shape* getNext()

- **next 포인터 값을 반환하여 연결 리스트의 다음 도형으로 이동할 수 있게 함**
- **도형 리스트를 순회하거나 특정 도형을 찾기 위해 사용**
- next 포인터가 마지막 도형이면 nullptr를 반환함

`draw()` (순수 가상 함수)

- Shape 클래스에서 정의되지 않고 이를 **상속받은 클래스에서 개별적으로 구현**
- Line, Circle, Rectangle에서 각각 다른 방식으로 도형을 출력
- Shape 클래스가 추상 클래스로 동작하게 만들며 **각 도형이 자신만의 출력 방식을 정의**하도록 강제함

Line, Circle, Rectangle의 `draw()`

- 상속받은 **각 도형 클래스에서 고유의 `draw()` 메서드를 구현**하여 도형의 정보를 출력함
- 각 도형마다 다르게 그려지는 동작을 정의함

4. 헤더 파일 보호

헤더 파일이 여러 번 포함되는 것을 방지하고 `#ifndef`와 `#define`으로 시작된 조건부 컴파일을 종료하며 코드의 안정성과 가독성을 높이기 위해 `#endif` 사용한다.

< Shape 클래스 구현부 - Shape.cpp >

1. 기본 로직

Shape 클래스와 그 파생 클래스(Line, Circle, Rectangle)의 메서드를 구현하였다. 다형성과 동적 연결 리스트 구조를 활용하여 도형 객체를 유연하게 관리하며 도형 추가하고 정보를 출력하는 작업을 수행한다.

2. 헤더 파일

`#include " Shape.h"`를 통해 Shape.h에서 선언된 Shape 클래스의 정의를 포함하여 해당 클래스의 멤버 변수와 함수들에 접근할 수 있게 한다.

`#include <iostream>`으로 `cout`, `cin`과 같은 표준 입출력 객체를 사용하기 위한 라이브러리 포함한다.

`using namespace std;`를 통해 std 네임스페이스에 있는 식별자를 쉽게 사용하도록 설정한다.

3. 함수 구현

Shape()

- 생성자: Shape 객체 생성 시 **next를 nullptr로 초기화함**
- 이는 객체가 연결 리스트의 끝에 위치하고 있음을 나타냄

~Shape()

- 가상 소멸자: 파생 클래스가 소멸될 때 올바른 소멸자 호출하여 메모리 관리를 안전하게 해제하도록 보장함

add(Shape* p)

- Shape 클래스에서 사용하는 단순한 **연결 리스트의 구조를 형성함**
- 주어진 **새로운 도형 객체(p)**를 **현재 도형 객체의 next 포인터에 연결하여** 두 도형 객체를 서로이어줌

구체적인 동작

1) this -> next

- this는 현재 호출된 shape 객체
- next는 이 객체가 가리키고 있는 다음 도형 객체를 저장하는 포인터
- this -> next = p;는 현재 객체의 **next 포인터를 입력받은 도형 객체(p)로 설정하여 연결 리스트를 형성하고 현재 객체 뒤에 새로운 도형 객체 p를 추가함**

2) p

- Shape* p는 새로운 도형 객체를 가리키는 포인터
- 이 포인터는 외부에서 전달되며 새로운 객체를 현재 객체의 다음 위치에 추가하는 역할을 함

3) 결과

- 현재 객체와 p가 연결되어 현재 객체 뒤에 새로운 도형 객체(p)가 위치하게 됨

paint()

- 현재 객체의 draw() 메서드를 호출하여 **도형의 정보를 출력**하고 next가 가리키는 다음 도형이 있을 경우 **재귀적으로 호출**하여 리스트의 모든 도형의 정보를 순차적으로 출력함
- 재귀적 구조를 활용하여 **연결 리스트의 모든 도형을 순회함**

getNext()

- 다음 도형을 가리키는 포인터(next)를 반환함

- 연결 리스트의 구조를 탐색 및 관리가 가능함

Line::draw()

: 도형이 선(Line) 임을 출력

Circle::draw()

: 도형이 원(Circle) 임을 출력

Rectangle::draw()

: 도형이 사각형(Rectangle) 임을 출력

연결 리스트

Shape 클래스는 **연결 리스트 구조를 사용하여** 도형 객체를 동적으로 관리함

각 객체의 **next 포인터**를 통해 다음 객체를 참조하며 **유연한 크기와 구조를 지원함**

동작 원리

1) 추가

- **add()** 메서드를 호출하면 **새로운 객체를 현재 객체의 next에 연결**
- 새로운 객체를 추가할 때 **리스트의 끝까지 탐색하거나 연결을 갱신함**

2) 탐색 및 순회

- **paint()** 메서드가 재귀적으로 호출되면서 **연결 리스트를 순차적으로 탐색하며**
모든 객체를 차례대로 출력함
- **재귀 호출을 통해 코드의 간결성과 직관성을 유지**

<GraphicEditor 클래스 선언부 – GraphicEditor.h>

1. 기본 로직

GraphicEditor 클래스는 도형을 관리하는 기능을 제공한다. pStart와 pLast는 도형 리스트의 시작과 끝을 가리키는 포인터이다. insertItem() 메서드는 주어진 타입의 도형을 리스트에 추가하고 deleteItem() 메서드는 지정된 인덱스의 도형을 삭제한다. Show() 메서드는 모든 도형을 출력하고 run() 메서드는 사용자와 상호작용을 통해 프로그램을 실행한다.

2. 헤더 파일

#ifndef GRAPHIC_EDITOR_H, #define GRAPHIC_EDITOR_H, #endif를 사용하여 구성된 헤더 가드는 헤더 파일이 여러 번 포함되더라도 컴파일 에러가 발생하지 않도록 방지한다.

#include "Shape.h"와 #include "UI.h"를 통해 GraphicEditor 클래스는 도형 관리와 사용자 상호작용을 구현할 수 있다. Shape.h는 도형 관련 정의를 포함하고 UI.h는 사용자 인터페이스 기능을 제공한다.

3. 함수 구현

GraphicEditor() (생성자)

- GraphicEditor 객체가 생성될 때 호출됨
- 생성자는 도형 리스트의 시작과 끝을 나타내는 포인터 pStart와 pLast를 초기화하는 역할

~GraphicEditor() (소멸자)

- 소멸자는 동적으로 할당된 메모리나 생성된 도형 객체를 정리하는 역할
- Shape 객체들에 대한 메모리 해제를 처리함

insertItem(int type)

- type 매개변수에 따라 새로운 도형을 리스트에 추가하는 메서드
- 도형의 타입에 따라 생성할 도형 객체를 선택하고 그것을 도형 리스트의 끝에 추가한 뒤 리스트의 시작과 끝을 나타내는 포인터(pStart, pLast)를 갱신함

deleteItem(int index)

- 주어진 인덱스에 해당하는 도형을 리스트에서 삭제하는 메서드
- 리스트에서 도형을 찾고 삭제한 뒤 리스트의 포인터를 수정하여 연결을 유지함
- 삭제된 도형에 대한 메모리 해제를 명시적으로 처리함

show()

- 리스트에 있는 모든 도형의 정보를 순차적으로 출력하는 메서드

run()

- GraphicEditor의 주요 실행 로직을 담당하는 메서드
- 사용자와 상호작용하여 도형을 추가하거나 삭제하는 등의 작업을 처리하고 사용

자의 입력을 받아 insertItem, deleteItem 등을 호출하여 프로그램 흐름을 제어함

4. 헤더 파일 보호

헤더 파일이 여러 번 포함되는 것을 방지하고 #ifndef와 #define으로 시작된 조건부 컴파일을 종료하며 코드의 안정성과 가독성을 높이기 위해 #endif 사용한다.

< **GraphicEditor** 클래스 구현부 – **GraphicEditor.cpp**>

1. 기본 로직

GraphicEditor 애플리케이션의 주요 동작을 관리하는 클래스 GraphicEditor의 메서드를 구현하였다. 사용자의 입력을 처리하여 도형을 동적으로 추가, 삭제, 표시하며 프로그램의 주요 동작과 연결 리스트 구조를 관리한다.

2. 헤더 파일

#include " GraphicEditor.h"를 통해 GraphicEditor.h에서 선언된 GraphicEditor 클래스의 정의를 포함하여 해당 클래스의 멤버 변수와 함수들에 접근할 수 있게 한다.

#include <iostream>으로 cout, cin과 같은 표준 입출력 객체를 사용하기 위한 라이브러리 포함한다.

using namespace std;를 통해 std 네임스페이스에 있는 식별자를 쉽게 사용하도록 설정한다.

3. 함수 구현

생성자 및 소멸자

GraphicEditor()

- **pStart와 pLast를 nullptr로 초기화**하여 연결 리스트를 초기 상태를 설정
- 연결 리스트의 시작과 끝을 관리할 준비함

~GraphicEditor()

- 연결 리스트를 순회하여 동적으로 생성된 모든 도형 객체를 안전하게 삭제하고 프로그램 종료 시 메모리 누수를 방지함

도형 추가

insertItem(int type)

- 사용자가 선택한 타입에 따라 새로운 도형 객체(Line, Circle, Rectangle)를 생성하고 연결 리스트에 추가
- **pStart == nullptr**인 경우 새로운 도형 객체를 리스트의 첫 번째 도형으로 설정
- **pStart != nullptr**인 경우 **add()** 메서드를 사용해 리스트 끝에 새로운 도형 객체를 추가하고 **pLast**를 갱신해 연결 리스트의 끝을 관리함

도형 삭제

deleteItem(int index)

- 사용자가 입력한 인덱스를 받아 연결 리스트에서 해당하는 도형을 삭제함

구현 세부사항

- 1) 연결 리스트에서 첫 번째 도형 삭제
 - pStart를 다음 도형으로 이동시키고 현재 도형을 삭제함
- 2) 중간 또는 끝 도형 삭제
 - 삭제할 도형의 이전(pre)와 현재(tmp)를 추적하여 **pre->add(tmp->getNext())**로 **연결 리스트의 구조를 유지**하며 삭제를 처리함
- 3) 예외 처리
 - 유효하지 않은 인덱스인 경우 오류 메시지를 출력함

도형 표시

show()

- 각 도형의 paint() 메서드를 호출해 도형의 정보를 출력하며 연결 리스트 구조를 따라 재귀적으로 모든 도형의 정보를 표시함

프로그램 실행

run()

- UI::getMenu() 메서드를 호출하여 사용자로부터 메뉴 입력을 받고 이를 기반으로 프로그램의 각 기능을 실행함
- 메뉴를 통해 도형 추가, 삭제, 출력, 종료 기능을 제공함

Switch 문

- 1: insertItem()을 호출해 새로운 도형을 추가함

- 2: deleteItem()을 호출해 선택한 도형을 삭제함
- 3: show()를 호출해 도형을 출력함
- 기타: 프로그램을 종료

<UI 클래스 선언부 - UI.h>

1. 기본 로직

UI 클래스는 사용자 인터페이스와 관련된 기능을 제공한다. getMenu() 메서드는 메뉴를 표시하고 사용자가 선택한 값을 반환하고 getShapeTypeToInsert() 메서드는 추가할 도형의 타입을 입력받고 반환한다. getShapeIndexToDelete() 메서드는 삭제할 도형의 인덱스를 입력받고 반환한다.

2. 헤더 파일

#ifndef UI_H, #define UI_H, #endif를 사용하여 구성된 헤더 가드는 헤더 파일이 여러 번 포함되더라도 컴파일 에러가 발생하지 않도록 방지한다.

3. 함수 구현

아래 함수 모두 사용자로부터 입력을 받아 필요한 값을 반환하는 정적 함수로 객체를 생성하지 않고 클래스 이름을 통해 호출할 수 있으며 GraphicEditor 클래스와 상호작용하여 사용자 입력을 받아 도형을 추가하거나 삭제하는 등의 기능을 지원함

getMenu()

- 프로그램에서 제공하는 메뉴를 사용자에게 표시하고 메뉴 선택을 요구하여 **사용자가 선택한 메뉴 번호를 반환함**
- 예를 들어, 메뉴 옵션이 출력된 후 사용자가 입력한 번호를 받아서 해당 옵션을 처리하는 데 사용됨

getShapeTypeToInsert()

- **사용자가 추가하려는 도형의 타입을 입력하도록 요구하여** 사용자는 도형의 종류를 선택하기 위한 값을 입력하고 이 함수는 그 값을 반환함
- 예를 들어, 원, 사각형, 삼각형 등의 도형 타입을 선택하는 데 사용됨
- 도형 타입은 정수형 값으로 처리되며 이 값은 **GraphicEditor의 insertItem 함수에 전달되어 도형을 추가하는 데 사용됨**

`getShapeIndexToDelete()`

- 사용자가 삭제하려는 도형의 인덱스를 입력하도록 요구하여 사용자는 도형 목록에서 삭제하고 싶은 항목의 번호를 입력하고 이 함수는 해당 인덱스를 반환함
- 반환된 인덱스 값은 **GraphicEditor**의 `deleteItem()` 메서드에 전달되어 해당 도형을 삭제하는 데 사용됨

4. 헤더 파일 보호

헤더 파일이 여러 번 포함되는 것을 방지하고 `#ifndef`와 `#define`으로 시작된 조건부 컴파일을 종료하며 코드의 안정성과 가독성을 높이기 위해 `#endif` 사용한다.

< UI 클래스 구현부 - UI.cpp >

1. 기본 로직

UI 클래스의 정적 메서드를 구현하여 사용자 입력을 처리하고 메뉴나 작업 선택을 받을 수 있도록 설계되었다. 사용자와 프로그램 간의 인터페이스 역할을 하며 사용자의 입력을 기반으로 프로그램 동작을 제어할 데이터를 반환하고 메뉴 선택, 도형 추가를 위한 타입 입력, 도형 삭제를 위한 인덱스 입력 등을 처리한다.

2. 헤더 파일

`#include "UI.h"`를 통해 UI.h에서 선언된 UI 클래스의 정의를 포함하여 해당 클래스의 멤버 변수와 함수들에 접근할 수 있게 한다.

`#include <iostream>`으로 `cout`, `cin`과 같은 표준 입출력 객체를 사용하기 위한 라이브러리 포함한다.

`using namespace std;`를 통해 `std` 네임스페이스에 있는 식별자를 쉽게 사용하도록 설정한다.

3. 함수 구현

`getMenu()`

- 사용자로부터 프로그램의 주요 메뉴(삽입, 삭제, 보기, 종료)를 입력받아 반환함

- 반환된 값은 `GraphicEditor::run()` 메서드에서 호출되어 Switch 문을 통해 프로그램의 동작을 결정함

구현 구조

- 1) 메뉴 출력 및 입력 요청
 - cout으로 메뉴 옵션을 출력하여 사용자의 선택을 유도함
 - Ex) "삽입:1, 삭제:2, 모두보기:3, 종료:4 >>"
- 2) 입력값 반환
 - 사용자가 입력한 값을 정수형으로 반환
 - 반환된 값은 프로그램의 주요 동작(삽입, 삭제, 보기, 종료)을 결정하는 데 사용됨

`getShapeTypeToInsert()`

- 삽입할 도형의 타입(선, 원, 사각형)을 사용자로 부터 입력받아 반환함
- 반환된 값은 `GraphicEditor::insertItem()` 메서드에서 적절한 도형 객체(`Line`, `Circle`, `Rectangle`)를 생성하는 데 사용됨

구현 구조

- 1) 도형 옵션 출력 및 입력 요청
 - cout으로 도형 옵션을 출력하여 사용자의 선택을 유도함
 - Ex) "선:1, 원:2, 사각형:3 >>"
- 2) 입력값 반환
 - 사용자가 입력한 값을 정수형으로 반환하며 '1: 선, 2: 원, 3: 사각형'에 대응함

`getShapeIndexToDelete()`

- 삭제할 도형의 인덱스를 사용자로 부터 입력받아 반환함
- 반환된 값은 `GraphicEditor::deleteItem()` 메서드에서 삭제 대상 도형을 탐색하는 데 사용됨

구현 구조

- 1) 삭제할 도형의 인덱스 요청
 - cout을 사용해 삭제할 도형의 인덱스를 입력받도록 요청함
 - 예: "삭제하고자 하는 도형의 인덱스 >>"
- 2) 입력값 반환
 - 사용자가 입력한 값을 정수형으로 반환하며 반환된 값은 삭제할 도형의 인덱스를 나타냄

<main 함수 – main.cpp>

1. 기본 로직

GraphicEditor 클래스의 기능을 시작하고 실행하는 기본적인 흐름을 제공한다.

GraphicEditor 클래스의 객체를 생성하고 run() 메서드를 호출하여 사용자와의 상호 작용을 통해 도형을 추가하거나 삭제하는 등의 작업을 처리한다. 프로그램이 종료되면 main 함수는 0을 반환하며 정상적으로 종료된다.

2. 헤더 파일

#include " GraphicEditor.h"를 통해 GraphicEditor.h에서 선언된 GraphicEditor 클래스의 정의를 포함하여 해당 클래스의 멤버 변수와 함수들에 접근할 수 있게 한다.

3. 함수 구현

GraphicEditor graphicEditor;

- GraphicEditor 클래스의 객체인 graphicEditor를 생성함
- 이 객체는 도형을 관리하고 사용자와 상호작용하며 프로그램의 핵심 기능을 처리함

graphicEditor.run();

- graphicEditor 객체의 run() 메서드를 호출하여 프로그램의 주요 로직을 실행함
- run()은 사용자 입력을 처리하고 도형을 추가하거나 삭제하는 등의 작업을 수행하는 메서드로 반복문을 포함하고 있어 계속해서 사용자의 입력을 받으며 도형 관리 작업을 수행함

return 0;

- main 함수가 성공적으로 종료되었음을 나타내는 반환값 0을 반환하면 운영체제에 정상 종료를 알리고 프로그램이 종료됨