자료구조와 C++ 프로그래밍

이름 : 이인
학번 : 2021125050

강의시간 : 월 9:00, 수 10:30
레포트 번호 : 과제1

1-1.

Rectangle.h

```
#pragma once // 헤더 파일의 내용이 여러개 인 것을 방지하기 위해 컴파일 시 컴파일러에 한 번만 포함되도록 설정함
    #include <iostream> // iostream 사용한다고 선언

    class Rectangle  // Rectangle 객체에 대한 정의
    {
    private: // Rectangle 객체 외부에서 해당 변수에 직접적으로 접근할 수 없도록 private scope에 멤버 변수 선언
    static int id;
    int width;// 넓이
    int height;//높이
    int xLow, yLow; // x좌표, y좌표

    public:// 외부에서 클래스 메소드를 통해 객체에 접근할 수 있도록 함
    Rectangle(int, int, int, int); // int인자 4개를 가지는 constructor 선언
    ~Rectangle(); // destructor 선언

    friend std::ostream& operator <<(std::ostream&, Rectangle&); //private 멤버나 protected 멤버를 외부 클래스에서 접근할 수 있도록 함

    Rectangle operator +(Rectangle&); //+ 오퍼레이터에 대한 해당 클래스의 오버로딩 선언
    };
```

Rectangle.cpp

```
#include "Rectangle.h" // Rectangle Class 선언에 대한 헤더파일을 가져와 메소드와 오퍼레이터에 대한 구체적인 정의를 할 것임
#include <iostream> // std::iostream 사용을 위해 선언
//using namespace std;

int Rectangle::id = 0;  //Rectangle 객체가 몇 번 생성되었는지를 나타내기 위해 id에 0 할당

Rectangle::Rectangle(int x, int y, int w, int h) { //constructor가 x좌표, y좌표, 넓이, 높이 parameter를 받아 Rectangle 인스턴스 생성시 자동으로 할당
    xLow = x;
    yLow = y;
    width = w;
    height = h;

    std::cout << ++id << "번째 Rectangle 객체 생성₩n₩n"; //constructor 실행 시 자동으로 id에 1을 더하여 객체가 생성될 때 마다 몇번 째 객체 생성인지 알 게 함
}

Rectangle::~Rectangle() { // destructor에 소멸 표시를 함으로써 해당 좌표값을 가진 Rectangle 객체가 메모리 상에서 소멸된 것을 명시적으로 나타냄
    std::cout << "좌표 (" << xLow << ',' << yLow << ')'
        << " Rectangle 객체 소멸₩n₩n";
}

std::ostream& operator <<(std::ostream& os, Rectangle& r) { // cout이 필요한 오퍼레이터 << 에 대해 오버로딩을 시행하기에 os에 대한 참조를 parmeter로 설정, Rectangle 객체에 대한 레퍼런스를 parameter로 받아 해당 객체의 멤버변수에 접근
    os << "height : " << r.height << std::endl
        << "width : " << r.width << std::endl
        << "x : " << r.xLow << std::endl
        << "y : " << r.yLow << std::endl << std::endl;

    return os;
}

Rectangle Rectangle::operator +(Rectangle& r) { //operator +가 참조 객체에 대해 피연산되는 객체와 레퍼런스 객체의 멤버 변수 값을 더한 새로운 Rectangle 객체를 반환함. => constructor 실행으로 id값 +1;

    int sum_x, sum_y, sum_w, sum_h;

    sum_x = xLow + r.xLow;
    sum_y = yLow + r.yLow;
    sum_w = this->width + r.width;  //this를 통해 피연산되는 객체의 멤버변수에 접근
    sum_h = this->height + r.height;

    return Rectangle(sum_x, sum_y, sum_w, sum_h);
}
```

main.cpp

```cpp
#include <iostream> //스탠다드 라이브러리 iostream 삽입
#include "Rectangle.h" // 클래스 Rectangle에 대한 정의가 담겨져 있는 헤더파일 삽입

int main()
{
Rectangle r1(1, 1, 3, 4); // x:1 y:1 width:3 height:4 인 Rectangle 객체 생성
std::cout << r1; // 오버로딩된 << 연산자로 r1객체 출력

Rectangle* r2 = new Rectangle(2, 3, 5, 5); // Rectangle 객체에 대한 포인터 변수를 생성하면서 new를 통해 동적 할당과 동시에 constructor 호출
std::cout << *r2; // 참조를 통한 출력

r1 = r1 + *r2; // 오버로딩된 + 연산자를 이용해 두 객체를 더한 값을 r1에 할당

std::cout << r1; // r1출력

delete r2; // 동적할당된 메모리가 더이상 필요 없어졌으니 delete로 삭제
}
```

출력 결과 설명

```
1번째 Rectangle 객체 생성

height : 4
width : 3
x : 1
y : 1

2번째 Rectangle 객체 생성

height : 5
width : 5
x : 2
y : 3

3번째 Rectangle 객체 생성

좌표 (3,4) Rectangle 객체 소멸

height : 9
width : 8
x : 3
y : 4

좌표 (2,3) Rectangle 객체 소멸

좌표 (3,4) Rectangle 객체 소멸

Program ended with exit code: 0
```

1번째 객체 생성 - main.cpp에서 Rectangle 클래스의 constructor를 호출하여 출력됨, constructor의 std::cout에 대해 ++id를 통해 생성과 동시에 몇 번 째 객체인지를 표시하며 << 오퍼레이터 오버로딩으로 Rectangle 객체의 멤버변수를 출력하게 됨

2번째 객체 생성단도 마찬가지

3번쨔 객체 생성 - operator +에 대한 오버로딩 로직이 Rectangle객체 에대한 참조를 받고, return값이 Rectangle 객체이므로 생성 메세지 발생, +연산이 다 이루어진 이후에 해당 객체에 멤버에 값이 할당됨 -> return이후에는 값이 할당된 3번째 객체가 소멸되므로 좌표값을 포함한 객체가 소멸된다고 표시됨.

+연산으로 새롭게 정의된 r1객체가 출력된다

이후 delete로 r2의 동적할당을 명시적으로 제거해서 r2가 먼저 되고, main함수의 역할이 끝났기 때문에 남은 r1객체가 소멸된다.

1-2

2-1.문제 인식

표준 cin 객체에 대해 오버로딩을 구현하여 커스텀 객체에 대해 cin을 할당했을 때 자동으로 해당 멤버변수에 대해 입력값을 할당하도록 구현하기로 함.

이를 통해 r1,r2 객체에 대한 멤버변수가 정의된다면 두 객체가 겹치는 부분의 Rectangle객체에 대한 정보를 구함 -> 결국 x,y,width,height에 해당하는 값을 구해야 함

2-2.해결 방안 구상

Rectanlge 객체를 받는 오버로드된 연산자 >>를 구현함. std::cin를 임의로 지정해 줘야 하기에 인자를 istream 객체에 대한 참조와 Rectangle 객체에 대한 오버로드를 구현하기에 Rectanlge 객체에 대한 참조를 인자로 받기로 함.

-> 사각형이 겹치는 케이스들을 분석해보니 약 7가지의 경우의 수가 나타남.

-> 공통적으로 나타나는 특징은 사각형의 왼쪽 아래 꼭짓점을 기준으로 했을 때

겹치는 영역(Intersection)의

왼쪽 변은, 기존 두 사각형의 왼쪽 변 중에서 더 오른쪽에 있는 것
오른쪽 변은, 기존 두 사각형의 오른쪽 변 중에서 더 왼쪽에 있는 것
위쪽 변은, 기존 두 사각형의 위쪽 변 중에서 더 아래쪽에 있는 것
아래쪽 변은, 기존 두 사각형의 아래쪽 변 중에서 더 위쪽에 있는 것

으로 파악할 수 있다.

->

이를 ADT화 시켜본다면 x,y는 겹치는 사각형이 존재 할 때 두 사각형 중 큰 값의 x,y가 intersection의 x,y값이 되고, width는 겹치는 두 사각형의 x+width 중 작은 값 - intersection의 x값이 된다.

height는 width와 같다.

이 ADT를 통해 두 사각형이 겹치지 않는 경우도 알아낼 수 있었는데
한 사각형의 x좌표값이 다른 사각형의 x+width 보다 큰 경우,
혹은 한 사각형의 y값이 다른 사각형의 y+height보다 큰 경우로 나눌 수 있다. 따라서 해당 ADT를 구체화 시킨 코드는 다음과 같다.

3.문제해결

```
bool Rectangle::isIntersected(Rectangle& r){
    if(r.height == 0 or r.width == 0){
        return false;
    } else return true;

}
```

```
std::istream& operator >>(std::istream& is, Rectangle& r){
    is >> r.xLow >> r.yLow >> r.width >> r.height;
    return is;
}
```

```
Rectangle Rectangle::generateIntersection(Rectangle &square1, Rectangle &square2) { // 두 사각형의 겹치는
    if (square1.xLow >= square2.xLow + square2.width) return Rectangle( x: 0, y: 0, w: 0, h: 0);
    else if (square2.xLow >= square1.xLow + square1.width) return Rectangle( x: 0, y: 0, w: 0, h: 0);
    else if (square1.yLow >= square2.yLow + square2.height) return Rectangle( x: 0, y: 0, w: 0, h: 0);
    else if (square2.yLow >= square1.yLow + square1.height) return Rectangle( x: 0, y: 0, w: 0, h: 0);

    int tmpX, tmpY, tmpWidth, tmpHeight;

    if(square1.xLow>square2.xLow){
        tmpX = square1.xLow;
    } else tmpX = square2.xLow;

    if(square1.yLow>square2.yLow){
        tmpY = square1.yLow;
    } else tmpY = square2.yLow;

    if (square1.yLow+square1.height > square2.yLow+square2.height){
        tmpHeight = square2.height+square2.yLow - tmpY;
    } else tmpHeight = square1.height+square1.yLow - tmpY;

    if (square1.xLow+square1.width > square2.yLow+square2.width){
        tmpWidth = square2.width+square2.xLow -tmpX;
    } else tmpWidth = square1.width+square1.xLow - tmpX;

    return Rectangle( x: tmpX, y: tmpY, w: tmpWidth, h: tmpHeight);
}
```

3-1.로직 설명

두 rectangle 객체를 비교할 것이기 때문에 r1,r2에 대한 참조를 parmeter로 받음,

```
if (square1.xLow >= square2.xLow + square2.width) return Rectangle(0,0,0,0);
else if (square2.xLow >= square1.xLow + square1.width) return Rectangle(0,0,0,0);
else if (square1.yLow >= square2.yLow + square2.height) return Rectangle(0,0,0,0);
else if (square2.yLow >= square1.yLow + square1.height) return Rectangle(0,0,0,0);
```

ADT를 통해 겹치지 않는 경우의 수 4가지를 if문을 통해 구별하여 겹치지 않는다면 x,y==0 이고 width,height==0인 Rectangle객체를 반환하도록 설정함.

```
int tmpX, tmpY, tmpWidth, tmpHeight;
```

intersectioned된 rectangle 객체를 반환하기 위해 각 멤버 변수에 할당하기 위한 temporary local variable을 선언

```
if(square1.xLow>square2.xLow){
        tmpX = square1.xLow;
    } else tmpX = square2.xLow;
    if(square1.yLow>square2.yLow){
        tmpY = square1.yLow;
    } else tmpY = square2.yLow;
    if (square1.yLow+square1.height > square2.yLow+square2.height){
        tmpHeight = square2.height+square2.yLow - tmpY;
    } else tmpHeight = square1.height+square1.yLow - tmpY;
    if (square1.xLow+square1.width > square2.yLow+square2.width){
        tmpWidth = square2.width+square2.xLow -tmpX;
    } else tmpWidth = square1.width+square1.xLow - tmpX;
    return Rectangle(tmpX,tmpY,tmpWidth,tmpHeight);
```

}ADT로 설계한 intersected Rectangle에 대한 정보를 각 논리에 맞게 local variable에 할당하고 마지막으로 그 값들을 가지는 Rectanle 객체를 반환하도록 함. Rectangle 객체를 반환하기에 함수 타입은 Rectangle이 된다.

```
bool Rectangle::isIntersected(Rectangle& r){
    if(r.height == 0 or r.width == 0){
```

```
        return false;
    } else return true;
```
}만약 겹치는 부분이 존재하지 않는다면 width나 height 둘 중 하나는 값이 0 일 것이기 때문에 이에 대한 판별을 하여 bool값 true 혹은 false를 반환해주는 함수를 만듬

```
std::istream& operator >>(std::istream& is, Rectangle& r){
    is >> r.xLow >> r.yLow >> r.width >> r.height;
    return is;
```
}std라이브러리의 istream에 대한 참조를 반환하도록 하는, 오퍼레이터 >>에 대한 오버로딩은 istream 객체가 필요하기 때문에 참조값 is를 받고 또한 Rectangle 객체에 대해서 오퍼레이터 오버로딩을 진행하기에 Rectangle객체에 대한 참조값이 존재해야하기에 이를 받음. std::cin과 동일한 역할을 하는 istream객체 is에 대한 참조를 반환함.

4-1-1.겹치는 부분에 대한 출력 결과 값

1번째 Rectangle 객체 생성 // r1 생성

2번째 Rectangle 객체 생성 // r2 생성

3번째 Rectangle 객체 생성 // r3 생성, 멤버변수가 모두 현재 0 0 0 0 인 상태

1 1 4 3 // 오버로딩된 연산자 >> 통한 r1의 멤버변수 값 입력 받기

height : 3
width : 4
x : 1
y : 1

2 1 1 5 // 오버로딩된 연산자 >> 통한 r1의 멤버변수 값 입력 받기

height : 5
width : 1
x : 2
y : 1

4번째 Rectangle 객체 생성 // generateIntersection 함수 호출로 반환을 위한 객체가 생성됨

좌표 (2,1) Rectangle 객체 소멸 // 4번째 객체에 값이 할당 된 이후 return으로 반환 후 4번째 객체 소멸

height : 3 // r3 객체에 대한 출력
width : 1
x : 2
y : 1

좌표 (2,1) Rectangle 객체 소멸 // r3 소멸

좌표 (2,1) Rectangle 객체 소멸 // r2 소멸

좌표 (1,1) Rectangle 객체 소멸 // r1 소멸

4-1-2. 겹치는 부분이 존재하지 않는 경우에 대한 출력 결과 값

1번째 Rectangle 객체 생성 // r1 생성

2번째 Rectangle 객체 생성 // r2 생성

3번째 Rectangle 객체 생성 // r3 생성

0 0 1 1 // 오버로딩된 연산자 >> 통한 r1의 멤버변수 값 입력 받기

height : 1 // r1출력
width : 1
x : 0
y : 0

1 1 2 2 //오버로딩된 연산자 >> 통한 r2의 멤버변수 값 입력 받기

height : 2 // r2출력
width : 2
x : 1
y : 1

4번째 Rectangle 객체 생성 // 겹치지 않는 조건을 만족하는 조건문에 대한 return 객체 생성

좌표 (0,0) Rectangle 객체 소멸 // 0 0 0 0 가 전달된 rectangle 객체를 return이 반환한 후 역할을 다 했으니 소멸

겹치는 부분이 존재하지 않습니다. // r3는 0 0 0 0 이 들어간 RectanIge객체를 바인딩 하게 되었으니 isIntersction 메소드가 false값을 반환하여 출력된다.

좌표 (0,0) Rectangle 객체 소멸 // 역할을 다 한 r3객체가 소멸

좌표 (1,1) Rectangle 객체 소멸 // r2 소멸

좌표 (0,0) Rectangle 객체 소멸 // r1 소멸

4-2 알게된 점.

call by pointer 와 call by reference의 차이가 불분명하여 구글링을 통해 해당 로직의 차이점을 알게 되었다. 또한 반환값을 커스텀 클래스 값으로 설정할 수 있다는 부분과 오퍼레이터또한 연산에 필요한 객체가 반드시 요구된다는 점에서 비롯하여 오퍼레이터 오버로딩 시에 해당 오퍼레이터가 포함된 라이브러리에 대한 명시 ex) >> 인 경우 std::istream operator >> 가 필요하다는 것을 알게되고, 해당 클래스에 대해서만 오퍼레이터 오버로딩을 실행하기 때문에 해당 클래스에 대한 객체를 인자로 받아야 한다는 부분을 알게되었다. 또 return값이 존재하는 함수의 경우 함수 호출과 함께 미리 함수가 반환할 type의 객체를 생성해 놓는 다는 부분도 알게되었다.

참조에 의한 호출과 주소에 의한 호출의 차이점을 이해하는데 특히 어려웠지만, 이번 과제를 통해 그 차이를 명확히 알고 가는 것 같아 의미있는 것 같다.

++ 넓이 구하는 함수 추가 부분

```
int Rectangle::areaShow(Rectangle& r3){
    return (r3.width * r3.height);
}
```

겹치는 부분을 구하여 새로운 Rectangle 객체 r3에 대하여 width와 height를 곱한 값을 반환해 main 영역에서 넓이를 출력함

```
if(Rectangle::isIntersected( &: r3)){ // r3가 가로, 세로가 존재한다면
    std::cout << r3; // r3 객체를 출력
    std::cout << "넓이:" <<Rectangle::areaShow( &: r3) << std::endl;
}else std::cout << "겹치는 부분이 존재하지 않습니다." << std::endl; // 가로 혹은
```