자료구조와 알고리즘

이름	황현택
학번	2019124206
강의 시간	월 10:30 / 수 09:00
레포트 번호	과제1

한국항공대학교
KOREA AEROSPACE UNIVERSITY

#(코드 진행)초반 출력 부분

```
C:₩Users₩hhtbo₩OneDrive₩문서₩GitHub₩Algorith
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


새로운 Rectangle 객체를 생성했습니다.
height 입력 :
```

<코드 실행 화면>

```cpp
#include <iostream>
#include "Rectangle.h"

int main()
{
    //기존 r1, r2 이름을 R1, R2로 변경함
    Rectangle R1(1, 1, 3, 4);
    std::cout << R1;

    Rectangle* R2 = new Rectangle(2, 3, 5, 5);
    std::cout << *R2;

    R1 = R1 + *R2;

    std::cout << R1<<std::endl;
```

<main.cpp>

```cpp
//생성자
Rectangle::Rectangle(int x, int y, int w, int h)
{
    xLow = x;
    yLow = y;
    width = w;
    height = h;
    std::cout << ++id << "번째 Rectangle 객체 생성\n";
}
```

<Rectangle.cpp>

- 7번 코드로 R1 객체가 정적으로 생성됨.
- 8번 코드의 출력 연산자(<<) 오버로딩으로 객체의 데이터들을 출력하고 있음.

- 10번 코드로 r2 객체가 동적으로 생성됨. (new)
- 11번 코드도 마찬가지로 생성된 객체에 대한 데이터를 출력 연산자 오버로딩으로 출력하고 있음.

- 13번 코드에서 + 연산자 오버로딩으로 R1과 R2의 데이터들을 합해주고 그 합한 데이터를 기반으로 생성자를 이용해 새로운 객체를 생성하고, 그 객체를 리턴해줌.
- 리턴한 객체는 R1의 자리에 들어가게 되는데, 들어간 이후 기존의 R1은 정적이기 때문에 해당 객체는 삭제됨. 따라서 (3,4) 소멸이 출력됨을 볼 수 있음.
- 이후에 새롭게 생성된 R1의 데이터가 출력이 됨.

#(코드 진행)중반 입력 부분

```
새로운 Rectangle 객체를 생성했습니다.
height 입력 : 3
width 입력 : 4
xLow 입력 : 1
yLow 입력 : 1

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

넓이 : 12

새로운 Rectangle 객체를 생성했습니다.
height 입력 : 8
width 입력 : 6
xLow 입력 : 2
yLow 입력 : 2

height : 8
width : 6
x : 2
y : 2

넓이 : 48
```

<코드 실행 화면>

```cpp
    //r1 생성
    Rectangle* r1 = new Rectangle();
    std::cin >> *r1;
    std::cout << std::endl;
    std::cout << *r1;
    r1->CulRectArea();

    std::cout << std::endl;

    //r2 생성
    Rectangle* r2 = new Rectangle();
    std::cin >> *r2;
    std::cout << std::endl;
    std::cout << *r2;
    r2->CulRectArea();
    std::cout << std::endl;
```

<main.cpp>

```cpp
//입력 연산자
std::istream& operator >>(std::istream& os, Rectangle& r)
{
    std::cout << "height 입력 : ";
    os >> r.height;
    std::cout << "width 입력 : ";
    os >> r.width;
    std::cout << "xLow 입력 : ";
    os >> r.xLow;
    std::cout << "yLow 입력 : ";
    os >> r.yLow;;


    return os;
}
```

<Rectangle.cpp>

• 이후 19번 줄에서 동적으로 새로운 객체를 생성한다. 이때, 생성자 오버로딩을 통해 멤버변수를 모두 0으로 초기화 한 객체를 리턴하고, 입력 연산자(>>) 오버로딩을 통해 멤버변수를 직접 입력받게 한다.

• 입력 받은 뒤 생성된 객체에 대한 데이터를 출력해준다.

• 클래스의 멤버함수 CulRectArea를 호출해서 해당 객체의 넓이를 출력해준다.

```cpp
//사각형 넓이 계산
void Rectangle::CulRectArea()
{
    std::cout << "넓이 : " << width * height << std::endl;
}
```

<Rectangle.cpp>

#(코드 진행)후반 겹치는 영역 생성 부분

```
영역의 height : 2
영역의 width : 3
영역의 xLow : 2
영역의 yLow : 2
영역의 넓이 : 6
4번째 Rectangle 객체 생성
좌표 (2,3) Rectangle 객체 소멸

좌표 (1,1) Rectangle 객체 소멸

좌표 (2,2) Rectangle 객체 소멸

좌표 (2,2) Rectangle 객체 소멸

좌표 (3,4) Rectangle 객체 소멸
```

<코드 실행 화면>

- 입력받은 r1, r2에서 겹치는 영역에 대한 객체 r3를 생성하고, r3의 데이터를 출력
  - 이후는 동적으로 생성한 R2,r1,r2의 삭제를 해주고, 프로그램이 종료하면서 정적으로 생성된 r3와 R1이 삭제가 된다(자세한 내용은 뒤에)

```
38    //동적으로 생성한 R2,r1,r2 객체 삭제
39    delete R2;
40    delete r1;
41    delete r2;
```

<main.cpp>

```
41  //겹치는 영역에 대한 정보 출력
42  Rectangle Rectangle::OverlapRect(Rectangle& r1, Rectangle& r2)
43  {
44      int xLow = Max(r1.xLow, r2.xLow);
45      int yLow = Max(r1.yLow, r2.yLow);
46      int width = Min(r1.xLow + r1.width, r2.xLow + r2.width) - xLow;
47      int height = Min(r1.yLow + r1.height, r2.yLow + r2.height) - yLow;
48
49      //겹치는 영역 체크
50      if (width <= 0 || height <= 0)
51      {
52          std::cout << "겹치는 영역이 없습니다." << std::endl;
53          return r1;
54      }
55
56      std::cout << "영역의 height : " << height<<std::endl;
57      std::cout << "영역의 width : " << width << std::endl;
58      std::cout << "영역의 xLow : " << xLow <<std::endl;
59      std::cout << "영역의 yLow : " << yLow <<std::endl;
60      std::cout << "영역의 넓이 : " << width * height <<std::endl;
61
62      return Rectangle(xLow, yLow, width, height);
63  }
```

<Rectangle.cpp>

- main에서 가져온 r1과 매개변수로 입력받은 r2의 겹치는 부분에 대해 계산을 하고, 새로운 멤버 변수들을 만들어 데이터를 출력한다.
- 또한 생성자를 이용해 정적으로 객체를 리턴하고, 이는 main에서 r3에게 들어가게 된다.
- 겹치는 영역이 나오지 않을 시 예외 처리를 해, r1 그대로의 객체를 리턴하게 했다.

(겹치는 영역에 대한 자세한 내용은 뒤에)

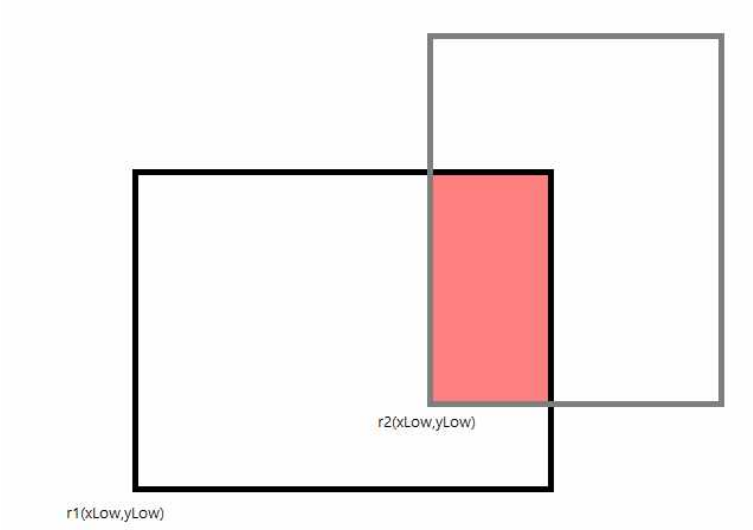
- 이로써 프로그램은 종료가 되었고, 이후엔 프로그램의 작동 원리에 대해 보다 더 자세히 알아보도록 하겠다.

#겹치는 영역에 대한 DATA 계산

• r1, r2에 따른 겹치는 영역은 크게 3가지로 나눌 수 있다.

1. 일부 영역이 겹침
2. 하나가 다른 하나에 포함됨.
3. 겹치는 영역이 없음.

• 1번과 2번의 경우 추가로 더 나눌 수 있지만, 겹치는 영역이 존재한다면 모든 경우에 대해 한 가지 방법으로 영역을 나타내는 것이 가능하다.




```
107  //편의를 위해 만든 Max,Min 함수
108 int Rectangle::Max(int a, int b)
109  {
110      return (a >= b) ? a : b;
111  }
112
113 int Rectangle::Min(int a, int b)
114  {
115      return (a <= b) ? a : b;
116  }
```

• 다음과 같이 그림이 있을 때, r3의 좌표는 (r2.xLow,r2.yLow)가 되고,
r3의 Height = (r1.yLow + r1.Height - r2.yLow) , Width = (r1.xLow + r1.width - r2.xLow) 이다.
하지만 영역이 모양이 다른 경우, 이는 틀린 공식이 된다.

• 결론적으로는 r1과 r2의 데이터에 대해 max, min값을 안다면 이를 모든 경우에 대입해서 해결할 수 있게 된다. 따라서 int 값을 두 개를 전달받아 최댓값, 최솟값을 리턴하는 Max,Min 함수를 만들어 주었다.

• Max함수와 Min 함수를 사용해서 다시 r3에 대한 값을 표현해보자.

1) r3.xLow는 r1.xLow와 r2.xLow 중 큰 값이 되고, r3.yLow도 동일하다.
2) 1)에서 더 작은 값이 나온 객체를 r, 큰 값이 나온 객체를 R이라 하자.
r3의 width는 r과 R의 우변 중 더 왼쪽에 있는 우변의 x좌표에서, R.xLow를 빼준 값이다.
height도 비슷하게 r,R 의 윗변 중 더 아래쪽에 있는 윗변의 y좌표에서, R.yLow를 빼면 된다.
3) 이를 공식화 한 함수가 OverlapRect가 된다.

• 이렇게 성공적으로 영역이 존재하는 모든 경우에 대해 표현을 했지만, 겹치는 영역이 없는 경우에 대해서는 아직 표현하지 못했다. 겹치는 영역이 없는 경우, 앞서 계산한 값 중 r3의 width나 height가 0 이하의 수가 나오게 되는데, 이를 예외처리하여 해결하면 된다.

#객체의 생성과 삭제

• 기본적으로 new 연산자로 생성한 동적 데이터는 delete 연산자로 반환시켜야 메모리 누수가 발생하지 않는다. 또한 정적으로 생성한 데이터는 스택이라는 메모리 공간에 쌓이게 된다. 정적으로 생성한 데이터는 프로그램이 종료 시 자동으로 반환이 되고, 이는 스택이라는 메모리 특성상 'FILO(first in last out)'이라는 구조를 띄는데, 따라서 반환이 될 때도 이 순서를 따르게 된다.

• main함수의 흐름을 보면 초반에 R1은 정적, R2는 동적으로 생성하였다. 이후에 곧바로 R1에 + 연산자로 새로운 객체를 생성하여 할당하였는데, 이 또한 +연산자 함수를 타고 들어가면 정적 생성이다.

• OverlapRect함수에서 리턴하는 객체는 정적으로, main함수의 r3도 정적 생성으로 볼 수 있다.

• 따라서 생성 순서에 따른 정적/동적 생성을 구분해 보면

정적 : R1, r3

동적 : R2, r1, r2

• 이로써 마지막 객체 소멸 부분을 보면 소멸 순서는 (R1 = R1 + *R2 부분은 제외 ; #초반 참고)

R2 -> r1 -> r2 -> r3 -> R1 순서로 소멸된다. (r3가 R1보다 먼저 소멸되는 이유는 스택구조이기 때문)

#마무리

• 새롭게 배운 점 : 동적, 정적 객체가 생성되고 반환되는 과정을 디버깅을 통해 알아보았다. 정적 객체의 경우 스택구조로 되어 있어 FILO순서로 반환이 된다는 것을 알게 되었고, 동적 객체는 delete 연산자로 따로 반환하지 않으면 프로그램이 종료되더라도 알 방법이 없다는 걸 알았다.

• 어려웠던 점 : 사각형의 겹치는 영역을 구하는 함수를 구현하는 것이 생각보다 어려웠다. 상황이 여러가지가 나올 수 있었고, 각 상황별로 구현한 함수가 오류없이 작동을 하는지 확인을 해야만 했다.
결국은 모든 경우에 대해 하나의 공식으로 표현을 할 수 있다는 걸 알게 되었고, 함수는 더 간결해 진 것 같다.

• 발전할 점 : 여러 함수와 생성자를 쓰다보니 출력의 endline 관리가 어려웠다. 좀 더 출력값을 보기쉽게 하기 위해 노력해야 할 것 같다.