9주차 정리노트

5조(김은미, 김승현)

<7장. 프렌드와 연산자 중복>

1. **프렌드 함수**

- 은미 : 친구? 가족은x 동일한 권한 -> 프렌드 함수: 클래스의 멤버 함수가 아닌 외부 함수
- 클래스의 멤버함수가 아닌 외부 함수
- 프렌드 함수는 개수 제한이 없다!

2. **프렌드 함수 3가지 유형**
   (1) 전역함수 : 외부 선언된 전역 함수
   (2) 다른 클래스의 멤버 함수 : 다른 클래스의 특정 멤버 함수
   (3) 다른 클래스 전체

3. **예제 7-1 : 프렌드 함수 만들기**

```
#include <iostream>
using namespace std;

class Rect;
bool equals(Rect r, Rect s);  //equals() 함수 선언

class Rect {  //Rect 클래스 선언
 int width, height;
public:
 Rect(int width, int height) { this->width = width; this->height = height; }
 friend bool equals(Rect r, Rect s); //equals()함수를 프렌드로 선언
};

bool equals(Rect r, Rect s) { //외부 함수
 if(r.width == s.width && r.height == s.height) return true;
 else return false; //equals()함수는 private 속성을 가진 width, height에 접근 가능
```

```
}

int main() {
 Rect a(3,4) b(4,5);
 if(equals(a, b)) cout << "equal" << endl;
 else cout << "not equal" << endl;
}


* 결과: not equal (객체 a와 b는 동일한 크기의 사각형이 아니므로)
```

4. 예제 7-2 : 다른 클래스의 멤버 함수를 프렌드로 선언

```
#include <iostream>
using namespace std;

class Rect;

class RectManager {  //RectManager 클래스 선언
public:
bool equals(Rect r, Rect s);
};

class Rect {  //Rect 클래스 선언
 int width, height;
public:
 Rect(int width, int height) { this->width = width; this->height = height; }
 friend bool RectManager::equals(Rect r, Rect s);
};

bool RectManager::equals(Rect r, Rect s) {
 if(r.width == s.width && r.height == s.height) return true;
 else return false;
}

int main() {
 Rect a(3,4) b(3,4);
 RectManager man;

 if(man.equals(a, b)) cout << "equal" << endl;
 else cout << "not equal" << endl;
}


* 결과: equal
```

5. **연산자 함수 : +와 == 연산자의 작성 사례**
   예시 코드)

```
Color a(BLUE). b(BLUE), c;
c = a+b;
if(a==b) {
...
}

* 이때 a,b가 일반 변수가 아니라 객체이기 때문에 위 코드처럼 a+b 로 작성하면 오류가 난다.

<해결 방법>
1. 클래스의 멤버 함수로 작성한다
Color operators+ (Color op2);
bool operators== (Color op2);

2. 클래스에 프렌드로 선언한다
friend Color operators+ (Color op1, Color op2);
friend Color operators== (Color op1, Color op2);
```

6. **예제 7-4 : 두 개의 Power 객체를 더하는 + 연산자 작성**
   - 묵시적 복사 생성자
     (call by value / call by reference)

7. **문제) 옳은 것들은? (정답 : 1, 2, 4)**

```
(1) //맞음
Power& return_This() {
return *this;
}

* 이유 :
Power& 는 Power 클래스 객체를 참조로 반환하겠다는 의미이고
this는 현재 객체를 가리키는 포인터이다.
*this는 포인터가 가리키는 실체 객체를 반환한다.
반환 형식이 일치하므로 맞다.

(2) //맞음
Power return_This() {
return *this;
}
```

```
* 이유 :
문법적으로 문제는 없다.
return *this 에서 나 자신을 넘기는게 아니라
나 자신을 복사하여 넘기는 것 즉, 복사 생성자를 호출한 것이다.
(*this의 내용이 복사된 후 반환됨)

(3) //틀림
Power& return_This() {
return this;
}

* 이유 :
Power&와 this가 가리키는 것이 다르므로 틀렸다.
this는 현재 객체를 가리키는 포인터이다.
*this라고 수정하면 올바른 문법이 된다.

(4) //맞음
Power * return_This() {
return this;
}

* 이유 :
this : 현재 객체를 가리키는 포인터
위 함수는 객체의 주소를 가리키는 포인터를 반환하는 것이기 때문에
Power * 의 형태가 맞다.

(5) //틀림
Power return_This() {
return this;
}

* 이유 :
this : 객체(나 자신)을 가리키는 포인터 => 즉, 주소를 가지고 있다.
반환 형식(Power)과 반환 값(this)가 일치하지 않기 때문에 오류가 난다.
this가 포인터 형식이기 때문에 형식이 맞지 않다.
```

8. 예제 7-8 : 전위 ++ 연산자 작성

```
#include <iostream>
using namespace std;

class Power {
int kick;
int punch;
public:
Power(int kick = 0, int punch = 0) {
this->kick = kick;
this->punch = punch;
}
void show();
```

```
Power& operator++ ();
};

void Power::show() {
cout << "kick=" << kick << "punch=" << punch << endl;
}

Power& Power::operator++ () {
kick++;
punch++;
return *this;
}

int main() {
Power a(3, 5), b;
a.show();
b.show();
b = ++a;
a.show();
b.show();
}
```

9. 예제 7-12 : a+b를 위한 연산자 함수를 프렌드로 작성

```
c = a + b 는 아래와 같다.

해석 1 : c = a + (b)
해석 2 : c =+ (a, b)
```