CPP Base

CHAPTER 5 (함수와 참조, 복사 생성자)

** 함수의 인자 전달 방식 **

*값에 의한 호출
- 설명 : 호출하는 코드에서 넘겨주는 실인자 값이 함수의 매개 변수에 복사되어 전달(void swap (int a, int b) {})

-'값에 의한 호출'로 객체를 전달 할 때 문제점 :
객체를 매개변수로 가져가는 함수의 경우 c++ 컴파일러는 매개 변수 객체의 생성자를 실행하지 않고 소멸자만 실행한다. (객체 복사가 이루어짐)

*주소에 의한 호출
- 설명 : 주소를 직접 포인터 타입의 매개변수에 전달받는 방식. 따라서 함수 호출 시 배열이 전달되는 경우, 배열의 이름이 전달되므로 자연스럽게 '주소에 의한 호출'이 이루어진다. (배열의 이름은 곧 배열의 주소임.) (void swap (int *a, int *b){})

-주소에 의한 호출의 특징 :
'값에 의한 호출'에 비해서 원본 객체를 복사하는 시간 소모가 없으며, 매개 변수가 단수 포인터 이므로 값에 의한 호출에서 발생했던 생성자, 소멸자 비대칭 문제가 없다. 하지만 배개 변수 포인터로 의도하지 않은 원본 객체 훼손이 발생할 수 있기에 코딩에 주의하여야 한다.

** 객체 치환 및 객체 리턴 ** (1.CPP 코드 참고)

*객체 치환
- 객체 치환 시 객체의 모든 데이터가 비트 단위로 복사된다.

*함수의 객체 리턴
CharacterBase GetCharacterBase() { return *this; } 부분에서 현재 클래스 자신의 복사본이 생성되어 GetCharacterBase()가 호출한 곳으로 전달된다.

** 참조화 함수 **(2.CPP코드 참고) [매우 중요함]

* 참조 개념
c++에서는 C언어에 없는 참조 개념을 도입하였다다. 포인터 변수를 사용하기 위해서는 '*' 기호를 사용하지만, 참조 변수는 '&' 변수를 사용한다. 참조 변수는 이미 선언된 변수에 대한 alias(별명)이다.

* C++ 참조의 활용도
-참조 변수 선언 : 참조 변수는 이미 선언된 변수(원본 변수)에 대한 별명으로서, 참조자(&)를 이용하여 선언하며, 선언시 반드시 원본 변수로 초기화 하여야 한다.

-참조 변수 사용 : 참조 변수를 사용하는 방법은, 보통 변수와 동일하며 참조 변수에 대한 사용은 바로 원본 변수의 사용이다.

-참조에 의한 호출 : 참조는 '참조에 의한 호출'에 많이 서용된다. '참조에 의한 호출'은 함수의 매개 변수를 참조를 타입으로 선언하여, 매개 변수가 함수를 호출하는 쪽의 실인자를 참조하여 실인자와 공간을 공유하도록 하는 인자 전달 방식이다.

-참조에 의한 호출의 장점 : '주소에 의한 호출'의 경우 포인터 타입으로 매개 변수를 선언하므로 호출 하는 쪽에서 주소를 전달하기 위해 & 연산자를 사용해야 하고, * 기호를 반복적으로 사용함에 따라서 실수의 가능성과 코드 작성의 긴장감이 크기 때문에 코드의 가독성이 떨어진다. 하지만 참조 매개 변수를 사용하면 간단히 변수만 넘겨주면 되고, 함수 내에서도 참조 매개 변수를 보통 변

-참조 매개 변수로 이루어진 모든 연산은 원본 객체에 대한 연산이 되며, 참조 매개 변수는 이름만 생성 되므로, 생성자와 소멸자는 아예 실행되지 않는다.

-참조 리턴(중요함) [3.cpp 참조 리턴 참고]
1. C언어에서 함수가 리턴하도록 허용된 값은 오직 값 뿐이었지만, c++ 에서는 함수가 참조를 리턴할 수 있다.
2. 참조 리턴이란 변수 등과 같이 현존하는 공간에 대한 참조의 리턴이다.

복사 생성자 [중요함]

*복사 생성자란(4.cpp참조)
-복사 생성은 객체가 생성될 때 원본 객체를 복사하여 생성되는 경우로서, c++에는 복사 생성 시에만 실행되는 특별한 복사 생성자가 있다.

-복사 생성자의 매개 변수는 오직 하나이며, 자기 클래스에 대한 참조로 선언된다. 또한 복사 생성자는 클래스에 오직 한 개만 선언할 수 있다.
ex) CharacterBase(const CharacterBase& Character); //복사생성자

-만약 복사 생성자를 가지고 있지 않는데 복사 생성문을 작성하게 되면 '디폴트 복사 생성자'를 묵시적으로 삽입하고 이 생성자를 호출하도록 컴파일 한다.

-디폴트 복사 생성자 코드는 얕은 복사를 실행하도록 만들어진 코드이다. 하지만 얕은 복사이기 때문에 원본과 사본이 각각 포인터로 공유하고 있는 문자열 배열 같은 경우 둘중 어느 하나라도 문자열 배열을 변경하면 원본과 사본 모두 변경되는 문제가 발생한다.

*얕은 복사의 문제점.
-문제점 : 포인터 타입의 멤버 변수가 없는 클래스의 경우, 얕은 복사는 문제가 없다. 하지만 클래스가 포인터 멤버 변수가 있는 경우 원본 객체의 포인터 멤버 변수가 사본 객체의 포인터 멤버 변수에 복사되면 이둘은 같은 메모리를 가리키게 되어 심각한 문제가 발생한다.

-예제 코드 실행시 Player1의 경우 이름이 깨져야 하며, 최종적으로 오류를 발생시킨다.

-무심코 얕은 복사를 사용하면 다양한 문제가 발생할 수 있으니 깊은 복사 생성자도 이용할 수 있도록 한다.

*깊은 복사 예제(5.cpp)

*묵시적 복사 생성
-개발자도 모르게 묵시적 복사로 인해 디폴트 복사 생성자가 생성되어 오류를 발생 시킬 수 있기 때문에 묵시적 복사가 발생하는 3가지 경우를 알아야 한다.
1. 객체로 초기화 하여 객체가 생성될 때 ex) CharacterBase Player2 = Player;
2. '값에 의한 호출'로 객체가 전달될 때
ex) void f(CharacterPlayer Character) {...} //여기서 매개변수 Character가 생성될 때 복사 생성자 호출
CharacterBase Player("Moon");
void f(Player);

3.함수가 객체를 리턴할 때
ex)
CharacterBase f() { CharacterBase Player(“Moon”); return Player; }
여기서 Player의 복사본을 생성하여 복사본을 리턴 함. 즉 복사본이 만들어 질대 복사 생성자 호출함.