**항목 5. C++가 은근 슬쩍 만들어 호출해 버리는 함수들에 촉각을 세우자.**

C++의 어떤 멤버 함수는 사용자가 클래스에 직접 선언하지 않으면 **컴파일러가 저절로 선언해 주도록 되어 있다.**

**1. 기본 생성자.**

**2. 복사 생성자.**

**3. 복사 대입 연산자.**

**4. 소멸자**

이 함수는 모두 기본형이다. 이들 모두 **public 으로 선언 되어 있으며 inline 함수로 정의** 되어 있다. 즉,

|  |
| --- |
| class Empty {}; |

이 클래스 구조는

|  |
| --- |
| class Empty  {  public:  Empty() {}  Empty(const Empty& rhs) {}  ~ Empty() {}    Empty& operator=(const Emtpy& rhs ) {}  } |

의 구조와 **같다는 것을 의미** 한다.

이들은 꼭 필요한 경우에만 컴파일러가 자동으로 만들어 주지만, **필요한 조건이 매우 간단** 합니다.

|  |
| --- |
| Empty e1; 기본생성자, 소멸자  Empty e2(e1); 복사 생성자  e2 = e1; 복사 대입 연산자 |

기본 생성자 / 소멸자가 생성될 때에는 기반 클래스 및 비정적 데이터 멤버의 생성자 / 소멸자가 자동으로 호출된다.

이때, **이 클래스가 상속한 기반 클래스의 소멸자가 가상 함수로 선언되어 있지 않으면, 비가상 소멸자로 만들어진다.**

컴파일러가 만들어낸 **복사 생성자와 복사 대입 연산자는 원본 객체의 비정적 데이터를 사본 객체로 복사**한다.

|  |
| --- |
| **template**<**typename** T>  **class** NamedObject  {  **public**:  NamedObject(**const** **char** \*name, **const** T& value);  NamedObject(**const** **std**::**string**& name, **const** T& value);  ...  **private**:  **std**::**string** nameValue;  T objectValue;  }; |

이 템플릿 클래스 안에는 생성자가 선언되어 있으므로, 컴파일러는 기본 생성자를 만들어 내지 않는다. 반면에 복사 생성자나복사 대입 연산자는 선언되어 있지 않기 때문에, 이 두 함수의 기본형이 컴파일러에 의해 만들어 진다.

|  |
| --- |
| NamedObject<**int**> no1("Smallest Prime Number", 2);  NamedObject<**int**> no2(no1); |

- 컴파일러는 no1.nameValue와 no1.objectValue를 사용하여 no2.nameValue 및 no2.objectValue를 각각 초기화 한다.

nameValue의 타입은 std::string 이고, std::string 은 자체적으로 복사 생성자를 갖고 있기 때문에 no2.nameValue의 초기화는 **std::string의 복사 생성자에 no1.nameValue를 인자로 호출**하여 이루어 진다.

objectValue의 타입은 템플릿 인스턴스화에 의해 int 형이 되는데, int는 기본 제공 타입이므로 no2.objectValue는 no1.objectValue의 값을 그대로 복사해 오는 것으로 끝난다.

따라서 다음과 같은 **기본형의 복사생성자가 만들어 질 것이다**.

|  |
| --- |
| NamedObject(**const** NamedObject& other)  : nameValue(other.nameValue), objectValue(other.objectValue)  {  } |

**복사 대입 연산자가 자동 생성되지 않는 경우**

컴파일러가 만들어 주는 복사 대입 연산자도 근본적으로는 복사 생성자와 원리가 똑같다. 그러나 복사 대입 연산자의 자동 생성이 되려면 최종 코드가 **‘적법해야(legal)’하고 ‘이치에 맞아야(reasonable)’**한다. 둘 중 하나라도 만족 하지 못하면 컴파일러는 operator= 함수를 만들어 내지 않는다.

|  |
| --- |
| **template**<**class** T> **class** NamedObject  {  **public**:  // 이 생성자는 이제 상수 타입의 name을 취하지 않는다.  // nameValue가 비상수 string의 참조자가 되었기 때문이다.  // 또한, 참조할 string을 가져야 하기 때문에 char\* 는 없애 버렸다.  NamedObject(**std**::**string**& name, **const** T& value);  ...  // operator= 는 선언되지 않았다고 가정  **private**:  **std**::**string**& nameValue;  // 참조자  **const** T objectValue;  // 상수   }; |

자 그럼 다음과 같은 코드에선 어떤 일이 일어날까?

|  |
| --- |
| **std**::**string** newDog("Persephone");  **std**::**string** oldDog("Satch");  NamedObject<**int**> p(newDog, 2);  NamedObject<**int**> s(oldDog, 36);  p = s;  // p의 데이터 멤버에는 어떤 일이 일어날까? |

C++에서 **참조자와 상수는 중간에 변경이 불가능 하므로 컴파일이 되지 않는다**. 따라서 참조자나 상수 데이터 멤버를 갖고 있는 클래스의 경우, 대입 연산을 지원하기 위해서는 **직접 복사 대입 연산자를 정의**해 주어야 한다.

추가로 기반 클래스가 **복사 대입 연산자를 private으로 선언한 경우**, 파생 클래스는 **암시적 복사 대입 연산자**를 가질 수 없다.

파생 클래스에 대해 컴파일러가 만들어 주는 복사 대입 연산자는 기반 클래스의 복사 대입 연산자를 호출해야 하는데, private으로 되어 있으면 **호출할 권한이 없다.**

**요약**

- 컴파일러는 경우에 따라 클래스에 대해 기본 생성자, 복사 생성자, 복사 대입 연산자, 소멸자를 암시적으로 만들어 놓을 수 있다.