**항목 4. 객체를 사용하기 전에 반드시 그 객체를 초기화 하자**

- 객체의 초기화하는데 C++의 행보는 이랬다가 저랬다가 한다.

|  |
| --- |
| int x; |

- **어떤 상황에서 x는 확실히 0이 되지만 다른 상황에서 그것이 보장 되지 않을 수 있다.**

|  |
| --- |
| class Point  {  int x, y;  }  Point p; |

- 이 경우에도 p의 데이터 멤버는 어떤 상황에서 초기화가 보장 되지만 어쩔 때는 보장 되지 않을 수 있다.

- **초기화 되지 않은 데이터 변수는 예상치 못한 결과를 초래**할 수 있다.

- 하지만, C++ 에서 **객체의 초기화**는 결코 제멋대로가 아니다.

- 명확한 규칙이 있지만, 모두 기억하기엔 **너무 방대하고 복잡**하다.

**- 가장 좋은 방법은 모든 객체를 사용하기 전에 항상 초기화 하는 것**이다.

**기본 제공 타입의 비 멤버 객체에 대해서는 초기화를 직접 진행**해 주어야 한다.

|  |
| --- |
| int x = 0;  const char \* text = ” ~~ ”  double d;  std::cin >> d; |

이런 부분을 제외 하면 나머지 **다른 상황의 초기화는 그** **객체의 생성자의 역할**로 귀결 될 것이다. 이때 지켜져야 할 규칙은 지극히 간단하다. **객체의 모든 것을 초기화 한다는 것**이다.

지키기 쉬운 규칙 이지만, **초기화와 대입을 혼동하여 사용하여서는 안 된다**.

|  |
| --- |
| class ABEntry {  …  private:  std::string theName;  std::string theAddr;  int numTime;  }    ABEntry::ABEntry(const std::string& name, const std::string& addr)  {  **theName = name;** // 지금은 모두 **‘대입’**을 하고 있다. **‘초기화’**가 아니다.  **theAddr = addr;**  **int numTime = 0;**  } |

위의 코드는 개선점이 있는 코드이다. **C++ 규칙에 의하면 어떤 객체든 그 객체의 데이터 멤버는 생성자의 본문이 실행 되기 이전에 초기화 되어야 한다**. 라고 명시 되어 있다.

위의 데이터 멤버는 초기화 되는 것이 아니라 **대입** 되고 있는 것이다. 이 멤버들의 생성자 호출은 ABEntry 생성자의 본문에 진입하기 이전에 완료 되어 진다. 우리는 **효율 적인 데이터 멤버의 초기화를 위해서 멤버 초기화 리스트를 사용**해야 한다.

|  |
| --- |
| ABEntry::ABEntry(const std::string& name, const std::string& addr)  **: theName(name)**  **, theAddr(addr)**  **, numTime(0)**  {} |

데이터 멤버에 원하는 값을 할당하는 건 같으나, 대입에 의한 생성자 보다는 훨씬 효율 적을 것이다.

**대입에 의한 초기화의 경우**, 멤버의 기본 생성자를 호출한 후에 새로운 값을 대입하기 때문에 불필요한 초기화 과정이 진행 된다.

**멤버 초기화 리스트의 경우**, 초기화 리스트에 들어가는 인자는 바로 데이터 멤버에 대한 **생성자의 인자로 쓰이기 때문에 복사 생성자에 의해 초기화가 바로 진행**된다.

**(기본 제공 타입의 경우, 대입과 초기화에 속도 차이가 없지만 역시 멤버리스트에 넣어주는 것이 좋다.)**

이 것은 멤버를 **기본 생성자로 초기화** 하고 싶을 때도 마찬가지이다.

|  |
| --- |
| ABEntry::ABEntry(const std::string& name, const std::string& addr)  **: theName()**  **, theAddr()**  **, numTime(0)**  {} |

멤버 초기화 리스트에 존재하지 않은 멤버들은 기본 생성자를 호출하게 되어 있어 멤버 초기화 리스트에 기본 생성자를 명시하는 데에 의아해 할 수 있다. 하지만 위와 같은 규칙이 지켜지지 않는다면, 어쩌다가 리스트에 어떤 멤버를 빼먹었을 떄 어떤 멤버가 초기화 되지 않을 수 있다는 위험 부담을 지고 가야할 것이다.

기본 제공 타입의 멤버를 **초기화 리스트로 넣는 일이** 선택이 아니라 **의무가 될 때도 있다**. 상수 이거나 참조자로 되어 있는 데이터 멤버의 경우엔 반드시 초기화 되어야 한다. **상수와 참조자는 대입 자체가 불가능** 하기 때문이다.

C++에서의 객체 초기화는 꽤나 변덕스러운 면이 있지만, 그렇지 않은 부분도 있다. 바로 객**체를 구성하는 데이터의 초기화 순서**이다. 이 순서는 어떤 컴파일러를 막론하고 항상 똑같다.

**1. 기본 클래스는 파생 클래스 보다 먼저 초기화되고,**

2. 클래스 데이터 멤버는 **그들이 선언된 순서대로 초기화** 된다.

어쩌다가 멤버의 선언 순서와 멤버 초기화 리스트에 들어간 순서가 바뀌더라도 **클래스 데이터 멤버가 먼저 선언된 멤버부터 초기화가 진행 될 것**이다.

기본 제공 타입의 비 멤버 객체의 초기화, 클래스 생성자의 멤버 초기화 리스트를 통해 객체의 초기화를 진행했다. 이제 걱정거리는 한가지 남았다. 바로 **비 지역 정적 객체의 초기화 순서는 개별 번역 단위에서 정해 진다는 것**

정적 객체(static object)는 자신이 생성된 시점부터 프로그램 종료 시까지 살아 있는 객체입니다. 그러므로 **스택 및 힙 기반 객체는 애초부터 정적 객체가 될 수 없다**.

정적 객체의 범주에 들어가는 것들은 다음과 같다.

**1. 전역 객체**

**2. 네임스페이스 유효범위에서 정의된 객체**

3. 클래스 안에서 **static으로 선언된 객체**

4. 함수 안에서 **static으로 선언된 객체**

5. 파일 유효범위에서 **static으로 정의된 객체**

이들 중 함수 안에서 있는 정적 객체는 **지역 정적 객체(local static object)**라고 하고, 나머지는 **비 지역 정적 객체(non-local static object)**라고 한다.

**번역 단위(translation unit)**는 컴파일을 통해 하나의 목적 파일(object file)을 만드는 바탕이 되는 소스 코드를 일컫는다. 여기서 **번역은 소스의 언어를 기계어로 옮긴다는 의미**. 기본적으로 소스 파일 하나가 되는데, 그 파일이 #include 하는 파일들까지 합쳐 하나의 번역 단위가 된다.

문제를 정리하면, 별도의 컴파일 된 소스 파일이 2개 이상 있으며, 각 소스 파일에 비 지역 정적 객체(전역, 네임 스페이스, 클래스, 파일에 있는 객체)가 1개 이상 들어있는 경우에 어떻게 되느냐 하는 것. 그리고 실질 적인 문제는 이것이다. 한쪽 번역 단위에 있는 비 정적 객체의 초기화가 진행되면서 다른 쪽 번역 단위에 있는 비 지역 정적 객체가 사용되는데, 불행히도 이 **(다른 쪽 번역 단위에 있는) 객체가 초기화되어 있지 않을 지도 모른다는 점**. 이유는 위에서 말했듯이 **‘별개의 번역 단위에서 정의된 비 지역 정적 객체들의 초기화 순서는 ‘정해져 있지 않다’**라는 사실 때문이다.

**해결 방법**, **비 지역 정적 객체를 하나씩 맡는 함수를 준비하고 이 안에 각 객체를 넣는 것입니다. 함수 속에서도 이들은 정적 객체로 선언하고, 그 함수에서는 이들에 대한 참조자를 반환하게 만든다**. 사용자 쪽에서는 비 지역 객체를 직접 참조하는 과거의 폐단을 버리고 이제는 함수 호출로 대신한다. 정리하면, ‘비 지역 정적 객체’가 ‘지역 정적 객체’ 바뀐 것이다.(singleton pattern의 구현 방식과 유사)

지역 정적 객체는 함수 호출 중에 그 객체의 정의에 최초로 닿았을 때 초기화되도록 만들어져 있습니다. 이것은 C++에서 보장하는 사실이며, 위의 방법은 바로 그 사실을 이용한 것

**요약**

\* 멤버가 아닌 기본 제공 타입 객체는 여러분 손으로 직접 초기화한다. 경우에 따라 저절로 되고도 하고 안되기도 하기 때문

\* 생성자에서는 데이터 멤버에 대한 대 입문을 생성자 본문 내부에 넣는 방법으로 멤버를 초기화 하지 말고 멤버 초기화 리스트를 즐겨 사용하자. 그리고 초기화 리스트에 데이터 멤버를 나열할 때는 클래스에 각 데이터 멤버가 선언된 순서와 똑같이 나열한다.

\* 여러 번역 단위에 있는 비 지역 정적 객체들의 초기화 순서 문제는 피해서 설계해야 한다. 비 지역 정적 객체를 지역 정적 객체로 바꾸면 된다.