**항목 14. 자원 관리 클래스의 복사 동작에 대해 진지하게 고찰하자.**

**이슈**.

힙 에 생기지 않는 자원은 auto\_ptr이나 tr1::shared\_ptr 등의 스마트 포인터로 처리해 주기엔 맞지 않다.

Mutex 타입의 뮤텍스 객체를 조작하는 C API를 여러분이 사용하는 중이라고 가정하고, 이 C API에서 제공하는 함수 중엔 lock 및 unlock이 있습니다.

|  |
| --- |
| void lock(Mutex \*pm); // pm이 가리키는 뮤텍스에 잠금을 겁니다.  void unlock(Mutex \*pm) // pm이 가리키는 해당 뮤텍스의 잠금을 풉니다. |

여기에 뮤텍스 잠금을 관리하는 클래스를 하나 만듭니다. 이전에 걸어 놓은 뮤텍스의 잠금을 잊지 않고 풀어줄 목적의 클래스 입니다. 이런 용도의 **클래스는 기본적으로 RAII 법칙을 따라 구성**합니다. 즉, **생성 시에 자원을 획득하고, 소멸 시에 그 자원을 해제 하는 것**입니다.

|  |
| --- |
| class Lock  {  public:  explicit Lock(Mutex \*pm) : mutexPtr(pm)  {  lock(mutexPtr); // 자원을 획득  }  ~Lock()  {  unlock(mutexPtr); // 자원을 해제  }  private:  Mutex \*mutexPtr;  }; |

사용자는 Lock을 사용할 때 RAII 방식에 맞추어 쓰면 된다.

|  |
| --- |
| Mutex m; // 쓰고 싶은 뮤덱스를 정의 한다.  {  Lock m1(&m); // 뮤텍스에 잠금을 겁니다.  // ... // 임계 영역에서 할 연산을 수행  // 블록이 끝나면 뮤텍스의 잠금이  // Lock의 소멸자 호출로 풀림  } |

그런데 Lock 객체가 복사된다면 어떻게 해야 할까요.?

Lock ml1(&m); // m에 잠금을 겁니다.

Lock ml2(ml1); // ml1을 ml2로 복사합니다

RAII 클래스에서는 고민해야 하는 문제로 다음과 같은 선택지 들이 존재한다.

**1. 복사를 금지 합니다.**

RAII 객체가 복사 되도록 놔두는 것 자체가 말이 안되는 경우가 많으며, 위의 Lock 같은 클래스도 이런 부류에 속하는 것 같다. 스레드 동기화 객체에 대한 ‘사본’은 의미가 없으며, 복사 하면 안되는 **RAII 클래스에 대해서는 복사를 막아야 한다**.   
(**복사 방지는 항목 6)**

|  |
| --- |
| class Lock : private Uncopyable  {  public:  } |

**2. 관리하고 있는 자원에 대해 참조 카운팅을 수행합니다.**

자원을 사용하고 있는 경우 마지막 객체가 소멸될 때까지 자원을 해제하면 안되는 경우도 종종 존재. 이럴 경우 해당 자원을 참조하는 개체의 개수에 대한 **카운트를 증가 시키는 식의 RAII 객체의 복사 동작**을 만들어야 한다. tr1::shared\_ptr을 사용한다.

단, tr1::shared\_ptr은 참조 카운트가 0이 될 때 자신이 가리키고 있던 대상을 삭제해 버리도록 기본 동작이 만들어져 있어서, 우리의 바람과는 다소 어긋난다.**Mutex의 잠금해제만 원하지 삭제까진 바라지 않는다.**

다행히 tr1::shared\_ptr은 삭제자 지정을 허용한다. 삭제자란, tr1::shared\_ptr이 유지하는 참조 카운트가 0이 되었을 때 호출되는 함수 혹은 함수 객체를 말한다. **삭제자는 tr1::shared\_ptr 생성자의 두번째 매개변수에 선택적으로 넣어 줄수 있다.**

|  |
| --- |
| class Lock  {  public:  explicit Lock(Mutex \*pm) : mutexPtr(pm, **unlock**) // 삭제자로 unlock을 사용하도록 선언  {  lock(mutexPtr.get()); // 자원을 획득  }  ~Lock()  {  unlock(mutexPtr); // 자원을 해제  }  private:  std::tr1::shared\_ptr<Mutex> mutexPtr; // 참조 카운팅을 위한 tr1::shared\_ptr  }; |

**2. 관리하고 있는 자원을 진짜로 복사합니다.**

때에 따라서 자원을 원하는 대로 복사할 수도 있다. 자원 객체를 복사하면 그 객체가 둘러싸고 있는 자원까지 복사해야 된다. 즉 깊은 복사가 수행되어야 한다.

**3. 관리하고 있는 자원의 소유권을 옮깁니다.**

흔한 경우가 아니나, 어떤 특정한 자원에 대해 그 자원을 실제 참조하는 RAII 객체는 딱 하나만 존재 하도록 만들고 싶어서 RAII 객체가 복사 될 때 소유권을 사본 쪽으로 넘기는 것을 말한다. **auto\_ptr을 사용하여 구현한다.**

**요약**

\* RAII 객체의 복사는 그 객체가 관리하는 자원의 복사 문제를 안고 가기 때문에, 그 자원을 어떻게 복사하느냐에 따라 RAII객체의 복사 동작이 결정된다.

\* RAII 클래스에 구현하는 일반적인 복사 동작은 복사를 금지하거나 참조 카운팅을 해주는 선으로 마무리하는 것입니다. 하지만 이 외의 방법들도 가능하니 참고해 둡시다.