항목 25: 예외를 던지지 않는 swap에 대한 지원도 생각해 보자.

표준 swap함수를 호출 하면 복사가 **세 번** 일어 납니다. 세 번이나 발생하기 때문에 프로그램 속도 저하가 발생 할 수 있습니다. 특히 클래스 안에 여러 가지 컨테이너를 포함 하고 있는 객체를 3번 복사 하는 일은 정말 **비효율적** 입니다.

따라서 **클래스 전체를 복사하지 말고 특정 부분**(맞바꾸는 부분)**만 내가 원하는 방법**(포인터만 살짝 바꾸기)**으로 맞바꾸도록 표준 swap함수에게 알려주도록 하겠습니다.**

클래스 Widget {

public :

void swap(Widget& other) // 포인터만 바꾸는 Widget클래스만의 swap방법을 지정함

{

using std::swap;

swap(맞바꿀포인터, other.맞바꿀포인터);

}

};

namespace std {

template<> // 완전 템플릿 특수화

void swap<Widget>(Widget& a, Widget& b)

{

a.swap(b); // Widget클래스 멤버함수 swap 호출

}

}

완전 템플릿 특수화를 통해 Widget클래스에 대해서 **표준 swap함수가 호출 되면 표준 swap이 하던 일(맞바꾸는 작업)이 아닌 Widget클래스의 멤버 함수를 호출하도록 시키는 것 입니다.** 실제 맞바꾸는 작업은 Widget클래스 멤버 함수에서 일어 나게 됩니다.

여기서 한 걸음 더 나아가서 Widget 클래스가 템플릿으로 만들어져 있어서 특정 매개변수에 따라 여러 가지 타입 T Widget클래스를 만들 수 있고 타입 T 전용 swap버전이 만들어져 있으면 전용 swap버전이 호출 되고 만약 없으면 일반 swap버전이 호출 되도록 만들고 싶습니다.

위에서 만들었던 것처럼 하면 안됩니다. 이유는 완전 템플릿 특수화가 아닌 **새로운 템플릿을 namespace std에 추가하는 것을 허용하지 않기 때문** 입니다.

namespace WidgetStuff

{

template<typename T>

**class Widget { … }; // Widget클래스 안에 멤버 swap함수를 정의해 둔다.**

// 비멤버 swap 함수

**void swap(Widget<T>& a, Widget<T>& b)**

**{**

**a.swap(b);**

**}**

}

그래서 새로운 namespace를 만들고 템플릿 클래스를 만듭니다. 그리고 난 후에 클래스 밖에 Widget<T>매개 변수를 받는 Non-멤버 swap 함수를 만들어서 **Widget<T>로 호출 되는 부분에 대해서만 Widget클래스의 멤버 함수 swap가 호출 되도록 유도 합니다.**

이제 사용 방법에 대해 알려드리겠습니다.

어떤 함수 템플릿을 만들고 있습니다. 그 함수는 중간에 swap을 사용하여 두 객체의 값을 맞바꾸는 작업이 포함 되어 있습니다. 어떻게 만들까요?

위 템플릿 함수를 호출 하는 여러 가지 매개 변수가 올 수 있습니다. Widget객체를 사용하여 호출 하게 되면 우리가 방금 전 만들어준 멤버 swap 함수가 호출 되어지게 될 것입니다. 하지만 다른 매개 변수도 올 수 있습니다. int, double 등 다른 매개 변수는 표준 std::swap함수를 호출하게 됩니다.

그러므로 우리는 **두 가지 모두 염두 해 두고 함수를 만들어야 합니다.** 이렇게

template<typename T>

void doSomething(T& obj1, T& obj2)

{

**using std::swap; // 표준 swap이 호출 될 수 있으므로 준비.**

. . .

**swap(obj1, obj2); // T 타입 전용의 swap을 호출 합니다.**

. . .

}

위 함수 내에서 swap가 호출 될 때 obj1, obj2가 어떤 형식에 따라서 나뉘어 호출 되어집니다.