C++ Programming

14th Study: C++ template

- Generic programming
- Function template
- Class template

C++ Korea 윤석준 (icysword77@gmail.com)

Generic programming

총칭적 프로그래밍

Generic programming

제네릭 프로그래밍

위키백과, 우리 모두의 백과사전.

제네릭 프로그래밍(영어: Generic programming)은 데이터 형식에 의존하지 않고, 하나의 값이 여러 다른 데이터 타입들을 가질 수 있는 기술에 중점을 두어 재사용성을 높일 수 있는 프로그래밍 방식이다.

제네릭 프로그래밍은 여러가지 유용한 소프트웨어 컴포넌트들을 체계적으로 융합하는 방법을 연구하는 것으로 그 목적은 알고리즘, 데이터 구조, 메모리 할당 메커니즘, 그리고 기타 여러 소프트웨어적인 장치들을 발전시켜 이들의 재사용성, 모듈화, 사용 편이성을 보다 높은 수준으로 끌어올리고자 하는 것이다.

http://ko.wikipedia.org/wiki/제네릭_프로그래밍

Standard Template Library

표준 템플릿 라이브러리

위키백과, 우리 모두의 백과사전.

표준 템플릿 라이브러리(Standard Template Library: **STL**)는 C++에서 일반적인 자료 구조와 알고리즘을 구현해 놓은 라이브러리의 집합이다. 지원하는 자료구조에는 vector, map, set 등이 있으며, 여러 가지 탐색 변경 알고리즘을 지원해 주고 있다. vector와 같은 자료구조에 삽입할 변수의 형이 정해지 있지 않고, 일반적인 형이라고 가정한 뒤 vector와 같은 컨테이너가 구현되어 있다. 즉 이 때는 C++언어의 템플릿 기능을 이용하고 있다. 이처럼 자료의 유형에 상관없이 구현되어 있기 때문에 generic이라고 말하기도 한다.

http://ko.wikipedia.org/wiki/표준_템플릿_라이브러리

Function template

함수 템플릿

두 값을 비교하는 함수 (int)

```
#include <iostream>

int Max(int a, int b)
{
    return a > b ? a : b;
}

void main()
{
    int nA = 300;
    int nB = 400;
    int nC = Max(nA, nB);
    std::cout << "Max (300 , 400) = " << nC << std::endl;

    float fD = 15.1f;
    float fE = 15.3f;
    float fF = Max(fD, fE);
    std::cout << "Max (15.1 , 15.3) = " << fF << std::endl;
}
```

int에는 제대로 동작하는데,
다른 타입에 대해서는 ???

두 값을 비교하는 함수 (float)

```
float Max(float a, float b)
{
    return a > b ? a : b;
}
```

그럼 bool 끼리 비교는 ?
double 끼리 비교는 ?
__int64 끼리 비교는 ?
struct 끼리 비교는 ?
class 끼리는 ?

두 값을 비교하는 함수 (계속)

아~ MAX 함수를 하나로 만들고 싶다.

무슨 방법이 없을까 ?

➡ template를 사용하면 된다.

두 값을 비교하는 함수 (template)

```
template <typename T>
T Max(T a, T b)
{
    return a > b ? a : b;
}
```

int 를 넣으면

모든 T 가 다 int

```
int Max(int a, int b)
{
    return a > b ? a : b;
}
```

두 값을 비교하는 함수 (template)

```
template <typename T>
T Max(T a, T b)
{
    return a > b ? a : b;
}
```

int, float 등 모든 타입에서 정상적으로 동작 완전 !

```
void main()
{
    int nA = 300;
    int nB = 400;
    int nC = Max(nA, nB);
    std::cout << "Max (300 , 400) = " << nC << std::endl;

    float fD = 15.1f;
    float fE = 15.3f;
    float fF = Max(fD, fE);
    std::cout << "Max (15.1 , 15.3) = " << fF << std::endl;
}
```

두 값을 비교하는 함수 (template)

```
template <typename T1, typename T2 >
T1 Max(T1 a, T2 b)
{
    return a > b ? a : b;
}
```

non-type template

여기에 500을 넣으면

```
template <typename T, int VAL>
T AddValue(T& value)
{
    return value + VAL;
}
```

탬플릿 인스턴스화

typename의 T가 구체적인 type으로 해석 되는 시점은 ?

➡ Compile 될 때. 근데 어떤 방법으로 ?

➡ 사용된 모든 타입에 대해서 인스턴스화

```
template <typename T>
T Max(T a, T b)
{
    return a > b ? a :
b;
}
```

➡

```
int Max(int a, int b)
{
    return a > b ? a : b;
}

float Max(float a, float b)
{
    return a > b ? a : b;
}
```

Class template

클래스 템플릿

변수 하나를 저장하는 class

```
class Data
{
    int data;
public:
    Data(int d) { data = d; }

    void SetData(int d) { data = d; }

    int GetData() { return data; }
};
```

근데 float를 저장하려면 ? bool 은 ? 다른 struct 나 class 는 ?

class template

```
template <typename T>
class Data
{
    T data;
public:
    Data(T d) { data = d; }

    void SetData(T d) { data = d; }

    T GetData() { return data; }
};
```