C++ 프로그래밍

김 형 기

hk.kim@jbnu.ac.kr

Generic Programming과 템플릿

Generic Programming과 템플릿

- Generic Programming이란
  - Macro를 활용한 Generic programming
- 템플릿을 활용한 Generic Programming
  - 함수 템플릿
  - 클래스 템플릿

문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming과 템플릿

- Generic Programming이란
  - Macro를 활용한 Generic programming
- Generic Programming using Template
- 함수 템플릿
- 클래스 템플릿

문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming

- 제네릭 프로그래밍
  - 간략한 정의 → 타입에 관계없이 동작하는 "일반적인" 코드를 작성하는 방법
  - Ex) 여러가지 덧셈을 수행하는 함수들을 작성할 때, 기존에는 아래와 같은 함수들을 개별적으로 구현해야 했음
    - add(int,int) / add(double,double) / add(Point,Point)
  - 이를 한 번의 코드 작성으로 가능하게 하는 방법이 제네릭 프로그래밍 기법

- 제네릭 프로그래밍의 구현 방법
  - 매크로 사용 ← 주의가 필요함!
  - 함수/클래스 템플릿 사용

문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming using Macro

- 매크로를 사용한 제네릭 프로그래밍
- 매크로 (#define)
  - 코드의 단순 대체(복붙)

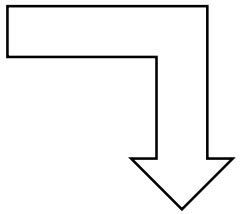
```
1 #define MAX_SIZE 100
2
3 #define PI 3.14159
```


문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming using Macro

- 매크로를 사용한 제네릭 프로그래밍

```
1 #define MAX_SIZE 100
2
3 #define PI 3.14159
4
5 if(num > MAX_SIZE)
6     cout << "Too big" << endl;
7
8 double area = PI * r * r;
```



전처리 후 아래와 같은
코드가 생성 되는 개념임

```
1 if(num > 100)
2     cout << "Too big" << endl;
3
4 double area = 3.14159 * r * r;
```


문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming using Macro

- Generic 하지 않은 프로그래밍
  - int 타입에 대한 Max 함수의 구현

```
1 int Max(int a, int b)
2 {
3   return (a>b)?a:b;
4 }
5
6 int x = 100;
7 int y = 200;
8 cout << max(x,y); //200
```

[주의] 비주얼 스튜디오의 경우 "Max"처럼 대문자를 사용하지 않으면, 커본 라이브러리에 있는 "max"함수와 충돌 발생할 수 있음

Generic Programming using Macro

- Generic 하지 않은 프로그래밍
  - double, char 타입으로 확장하려면 추가적인 구현이 필요
    - (암시적 형 변환을 통해 추가 구현하지 않아도 가능한 경우도 있으나 제한적)

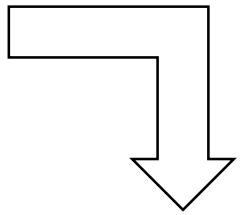
```
1 int Max(int a, int b)
2 {
3   return (a>b)?a:b;
4 }
5
6 int x = 100;
7 int y = 200;
8 cout << max(x,y); //200
```

```
1 int Max(int a, int b)
2 {
3     return (a>b)?a:b;
4 }
5
6 double Max(double a, double b)
7 {
8     return (a>b)?a:b;
9 }
10
11 char Max(char a, char b)
12 {
13     return (a>b)?a:b;
14 }
```

Generic Programming using Macro

- 매크로를 사용한 max함수의 일반적인 구현
  - 매크로에 인수(a,b)를 정의해 줄 수 있음

```
1 #define MAX(a,b) ((a>b)?a:b)
2
3 cout << MAX(10,20) << endl; //20
4 cout << MAX(2.4,3.5) << endl; //3.5
5 cout << MAX('A','C') << endl; //C
```



전처리 후 아래와 같은
코드가 생성 되는 개념임

```
1 cout << ((10>20)?10:20) << endl; //20
2 cout << ((2.4>3.5)?2.4:3.5) << endl; //3.5
3 cout << (('A'>'C')?'A':'C') << endl; //C
```


문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming using Macro

- 매크로를 사용할 때의 유의사항
  - 코드가 단순 대체됨에 유의해야 함
  - 매크로문을 괄호로 감싸는 것이 안전
    - ➢ 세미콜론(;)은 포함하지 않는 것이 기본 약속

```
1 #define SQUARE(a) a*a
2
3 result = SQUARE(5); //25
4 result = 5*5; //25
5
6 result = 100/SQUARE(5); //we expect 4
7 result = 100/5*5; //100
```

```
1 #define SQUARE(a) ((a)*(a))
2
3 result = SQUARE(5); //25
4 result = ((5)*(5))
5
6 result = 100/SQUARE(5); //we expect 4
7 result = 100/((5)*(5)); //4
```


문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

C++ preprocessor

고급

- #define ?
  - 단순한 복사 붙여넣기!
  - Platform independency 구현, 코드 단축, debug 목적 등으로 다양하게 활용

```
1  #include <iostream>
2
3  #define DEBUG
4
5  #ifdef DEBUG
6  #define LOG(x) std::cout << x << std::endl;
7  #else
8  #define LOG(x)
9  #endif
10
11 int main()
12 {
13     LOG("Hello");
14 }
```

3번 라인의 #define … 이 있는 경우와 없는 경우 동작을 비교

Generic Programming using Macro

Why are preprocessor macros evil and what are the alternatives?

71

46

I have always asked this but I have never received a really good answer; I think that almost any programmer before even writing the first "Hello World" had encountered a phrase like "macro should never be used", "macro are evil" and so on, my question is: why? With the new C++11 is there a real alternative after so many years?

The easy part is about macros like `#pragma`, that are platform specific and compiler specific, and most of the time they have serious flaws like `#pragma once` that is e
situation: same name in different paths and with some network setu

But in general, what about macros and alternatives to their usage?

c++ c++11 compiler-construction c-preprocessor

asked

viewed

active

Macros are just like any other tool - a hammer used in a murder is not evil because it's a hammer. It is evil in the way the person uses it in that way. If you want to hammer in nails, a hammer is a perfect tool.

There are a few aspects to macros that make them "bad" (I'll expand on each later, and suggest alternatives):

1. You can not debug macros.
2. Macro expansion can lead to strange side effects.
3. Macros have no "namespace", so if you have a macro that clashes with a name used elsewhere, you get macro replacements where you didn't want it, and this usually leads to strange error messages.
4. Macros may affect things you don't realize.

Generic Programming과 템플릿

- Generic Programming이란 : 타입에 관계없이 작동하는 코드 작성 개념
  - Macro를 활용한 Generic programming : 매크로를 활용하여 코드를 대체하는 방식, 주의가 필요
- 템플릿을 활용한 Generic Programming
  - 함수 템플릿
  - 클래스 템플릿

문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming using Template

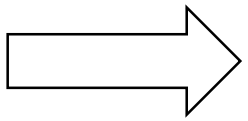
- C++ 템플릿
  - "설계도" 개념
  - 함수 템플릿과 클래스 템플릿 구현을 지원
  - 어떤 데이터 타입이든, 컴파일러가 적절한 함수/클래스를 설계도를 기반으로 생성함
    - 다른 언어의 경우 런타임에 생성하는 경우도 있음
    - 사용하지 않는경우 생성하지 않음
  - 제네릭 프로그래밍 / 메타 프로그래밍

문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming using Template

- 템플릿을 사용한 max 함수의 구현
  1. 타입명을 T로 대체
  2. T가 템플릿 인수라는 것을 명시
     - ➢ template <typename T> or template <class T>
     - ➢ (꼭 T를 써야 하는 것은 아니지만 일반적인 약속으로 T를 우선적으로 사용함)

```
1 int Max(int a, int b)
2 {
3     return (a>b)?a:b;
4 }
```



```
1 template <typename T>
2 T Max(T a, T b)
3 {
4     return (a>b)?a:b;
5 }
```

Generic Programming using Template

- 템플릿을 사용한 max 함수의 사용
  - 템플릿 함수를 사용할 때는 자료형을 < >안에 명시
    - ➢ 컴파일러가 타입을 추론 가능할 때는 생략 가능
  - 호출할 타입이 결정되면, 컴파일러가 템플릿을 기반으로 실제 함수를 생성
    - ➢ 템플릿 코드만 존재할 때는 아무 함수도 생성되지 않음. 실제 사용이 되어야만 코드가 생성됨

```
1 int a = 10;
2 int b = 20;
3
4 cout << Max<int>(a,b);
```

```
1 double c = 1.23;
2 double d = 4.56;
3
4 cout << Max<double>(c,d);
5 or
6 cout << Max(c,d); //생략 가능
```

Generic Programming using Template

- 템플릿이 사용 가능한 경우
  - max함수의 경우 >가 정의되어 있어야 (a > b)가 연산 가능
  - 즉, 클래스의 경우 필요한 연산자가 오버로딩되어 있어야 템플릿 max함수 사용 가능

```
1 template <typename T>
2 T Max(T a, T b)
3 {
4     return (a>b)?a:b;
5 }
```

(a>b)의 결과를 계산할 수 있어야만 함수 본문이 실행 가능함!

문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming using Template

- 템플릿이 사용 가능한 경우
  - 만일 아래의 경우, Point 클래스에 대해 operator>가 오버로딩되어 있지 않다면 컴파일 오류 발생

```
1 Point p1{10,20};
2 Point p2{20,30};
3
4 cout << Max<Point>(p1,p2);
```

Generic Programming using Template

- 템플릿의 다중 매개변수
  - 서로 다른 이름을 사용하여 다중 매개변수를 정의 가능
  - 매개변수가 다르다면 서로 타입이 다를 수 있음

```
1 template <typename T1, typename T2>
2 void func(T1 a, T2 b)
3 {
4     cout << a << " " << b;
5 }
6
7 func<int, double> (10, 20.05);
8
9 func('A', 12.4);
```

T1은 int고 T2는 double인 func 함수가 생성됨

타입을 명시하진 않았지만, T1은 char이고 T2가 double인 것이 명확하므로 해당하는 함수가 생성됨

문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming using Template

- 템플릿의 특수화
  - 특정 자료형에 대해서는 템플릿을 사용하지 않고 별도 구현한 함수를 사용하도록 구현 가능
    - ➢ 즉, string 타입에 대해 Min함수를 호출할 때는 위의 함수가 아닌 아래 함수가 실행됨

```
1  template <typename T>
2  T Min(T a, T b) {
3      return (a < b) ? a : b;
4  }
5
6  template<>
7  std::string Min(std::string a, std::string b)
8  {
9      return (a.length() < b.length()) ? a : b;
10 }
```

std::string type의 인자에 대해 사용하는 (명시적) 특수화

문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming과 템플릿

- Generic Programming이란 : 타입에 관계없이 작동하는 코드의 작성
  - Macro를 활용한 Generic programming : 매크로를 활용하여 코드를 대체하는 방식, 주의가 필요
- Generic Programming using Template : 타입이 정해지지 않은 템플릿을 만들고, 컴파일시 해당 템플릿이 사용되는 경우 코드가 생성됨
  - 함수 템플릿 : template<typename T>, 템플릿 인자와 특수화
  - 클래스 템플릿

Generic Programming using Template

- 클래스 템플릿
  - 함수 템플릿과 유사한, 클래스에 대한 템플릿 구현 지원
  - 클래스의 “설계도”
  - 컴파일러가 타입에 따라 적절한 클래스를 생성해 줌

문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming using Template

- 클래스 템플릿, Item 클래스 예제

```
1  class Item
2  {
3  private:
4      string name;
5      int value;
6  public:
7      Item(string name, int value)
8          :name{name},value{value}
9      {}
10     string getName() const {return name;}
11     int getValue() const {return value;}
12 };
```

Generic Programming using Template

- 클래스 템플릿의 정의

```
template <typename T>
class Item
{
private:
    string name;
    T value;
public:
    Item(string name, T value)
        :name{name},value{value}
    {}
    string getName() const {return name;}
    T getValue() const {return value;}
};
```


문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming using Template

- 클래스 템플릿의 사용

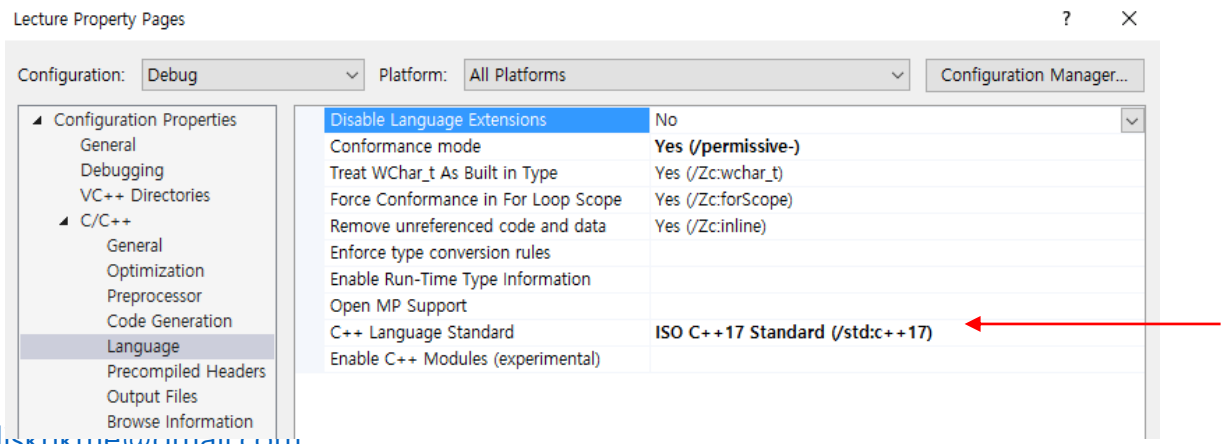
```
1 Item<int> item1 {"Kim", 1};
2
3 Item<double> item2 {"Lee", 10.5};
4
5 Item<string> item3 {"Park", "Hello"};
```


문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

- 클래스 템플릿의 사용
  - 함수처럼 타입을 적지 않아도 타입을 추론하는 기능이 C++ 17부터 추가됨

```
1 Item item1 {"Kim", 1}; // Item<int>로 추론
2
3 Item item2 {"Lee", 10.5}; // Item<double>로 추론
4
5 Item item3 {"Park", "Hello"}; // Item<std::string>으로 추론
```

  - VS2017에서는 프로젝트 속성 → C/C++ → 언어 → C++ 언어 표준을 C++17로 바꾸어 주어야 가능



문의사항: hk.kim@jbnu.ac.kr || diskhkime@gmail.com

Generic Programming using Template

- 클래스 템플릿의 다중 매개변수
  - 선언과 사용

```
template <typename T1, typename T2>
class MyPair {
private:
    T1 first;
    T2 second;
public:
    MyPair(T1 val1, T2 val2)
        : first{val1}, second{val2}
    {}
};

MyPair<string, int> p1{"Kim", 1};

MyPair<int, double> p2{123, 45.6};
```


문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming using Template

- 클래스 템플릿의 (부분)특수화
  - class classname<type 명시>

```
template <typename T1, typename T2>
class Item {
private:
    T1 key;
    T2 value;
public:
    …
};

template <typename T>
class Item<T, double> {
private:
    T key;
    double value;
public:
    …
};
```

value가 double 형일때 사용할
클래스 템플릿의 특수화

Generic Programming using Template

중요!

- 클래스 템플릿 매개변수
  - 타입이 아닌 다른 매개변수를 사용할 수 있음(non-type template argument)
    - 아래 예제에서 N의 값을 템플릿 클래스의 사용시에 직접 명시해 줌

```
1 template <typename T, int N>
2 class Array {
3     int size = N;
4     T values[N];
5     …
6 }
```

정적 배열이기 때문에 고정된 크기가 필요해서, 숫자 N 자체를 템플릿 인수로 같이 받도록 정의함!

```
1 int main() {
2
3     Array<int, 5> nums;
4     Array<double, 10> nums2;
5     ...
6 }
```

왼쪽 사용 코드를 보고 컴파일러가 오른쪽과 같은 두 개의 클래스를 만들어냄

```
1 class Array<int, 5> {
2     int size = 5;
3      int values[5];
4     …
5 }
6
7 class Array<double, 10> {
8     int size = 10;
9     double values[10];
10    …
11 }
```


문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

Generic Programming과 템플릿

- Generic Programming이란 : 타입에 관계없이 작동하는 코드의 작성
  - Macro를 활용한 Generic programming : 매크로를 활용하여 코드를 대체하는 방식, 주의가 필요
- Generic Programming using Template : 타입이 정해지지 않은 템플릿을 만들고, 컴파일시 해당 템플릿이 사용되는 경우 코드가 생성됨
  - 함수 템플릿 : template<typename T>, 템플릿 인자와 특수화
  - 클래스 템플릿 : 클래스에도 동일하게 사용 가능, 템플릿 인자에 int N 등도 사용 가능함

문의사항: hk.kim@jbnu.ac.kr || diskhkme@gmail.com

추가 슬라이드

Generic Programming using Template

- 템플릿의 특수화
  - 특정 자료형에 대해서는 템플릿을 사용하지 않고 별도 구현한 함수를 사용하도록 구현 가능
  - 사실 아래 경우에는 템플릿 특수화를 하지 않아도 오버로딩을 통해 동일하게 동작
  - 하지만 미묘한 차이가 있으므로 실제 사용할 때 문제가 생기면 참고
    - https://stackoverflow.com/questions/7108033/template-specialization-vs-function-overloading
    - https://stackoverflow.com/questions/5986935/difference-between-explicit-specialization-and-regular-functions-when-overloadin

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

template<>
std::string min(std::string a, std::string b)
{
        return (a.length() < b.length()) ? a : b;
}
```

사실 이 예제에서는 특수화 명시를 하지 않아도 std::string에 대해서 잘 동작