

C++ Programming

14th Study: Generic Programming

- Generic programming in C++
- Function template
- Class template



C++ Korea 옥찬호 (utilForever@gmail.com)

Generic programming in C++

A style of computer programming in which algorithms are written in terms of to-be-specified-later that are then instantiated when need for specific types provided as parameters

Generic programming in C++

- 절차적 프로그래밍 패러다임에서는 절차(Procedure) 또는 함수(Function)가 기본적인 프로그래밍 단위
 - 함수는 특정 값에 의존하지 않고 여러 값에 적용할 수 있는 알고리즘을 작성할 수 있다는 점에서 유용
- `sqrt()` 함수는 호출될 때 넘겨지는 어떤 값에 대해 제곱근을 계산
 - 만약 `sqrt()` 함수가 특정 인자 값만 계산할 수 있다면 쓸모 X
 - `sqrt()` 함수는 매개변수(Parameter)에 기반해서 작성됨
 - 매개변수는 함수를 호출하는 쪽에서 넘길 수 있는 임의의 값
 - 전산학에서는 이런 상황에 대해 값을 매개변수화(Parameterize)했다고 이야기함

Generic programming in C++

- 객체 지향 패러다임에서는 객체(Object)라는 개념을 추가해 어떤 동작과 그에 연관된 데이터를 한 곳에 묶음
 - 하지만 함수나 메서드에서 값을 파라미터화하는 개념은 바뀌지 않음
- 템플릿은 파라미터화에 대한 개념을 한 발자국 더 확장시켜 데이터 타입조차도 값으로서 파라미터화함
 - C++에서 타입은 기본 내장 타입 int, double은 물론 Bank, Complex 같은 사용자 정의 클래스까지 포함
 - 템플릿을 이용하면 값에 독립적일 뿐만 아니라 그 값의 타입에도 독립적인 코드를 만들 수 있음

The background is a solid blue color. It is decorated with several dotted white squares and rectangles of various sizes, scattered across the slide. Some are simple outlines, while others are nested or arranged in a grid-like pattern.

Function template

Defines a family of functions

Function template

- 최대값을 구하는 max 함수 : int는 OK, 하지만 double은?

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

max(14, 21) → 21

```
int main()
{
```

max(300, 400) → 400

```
    int a = 300;
    int b = 400;
```

```
    std::cout << "max(300, 400) = " << max(a, b) << std::endl;
```

```
    double c = 14.8;
    double d = 21.5;
```

```
    std::cout << "max(14.8, 21.5) = " << max(c, d) << std::endl;
```

```
}
```

Function template

- double도 OK, 하지만 bool은? 구조체는? 클래스는? ...

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

```
double max(double a, double b)
{
    return a > b ? a : b;
}
```

```
int main()
{
    int a = 300;
    int b = 400;
    std::cout << "max(300, 400) = " << max(a, b) << std::endl;

    double c = 14.8;
    double d = 21.5;
    std::cout << "max(14.8, 21.5) = " << max(c, d) << std::endl;
}
```

Diagram illustrating function calls from `main()` to the `max` functions:

- `max(300, 400) → 400` (calls the `int max` function)
- `max(14.8, 21.5) → 21.5` (calls the `double max` function)

Function template

- max 함수는 타입에 상관 없이 같은 일을 함 (최댓값 반환)
- 같은 일을 하되 타입만 바꿔야 한다면,
다른 구현 방식을 적용하는 것이 나을 수도 있음
→ 템플릿(Template)을 사용하면 됨!
- 템플릿에서는 타입을 '제외'할 수 있음
 - 함수의 호출자가 타입 목록을 받지 않고,
컴파일러가 함수에 맞춰 필요한 타입을 골라줌

Function template

- 템플릿을 선언하기 위한 문법 : `template <typename T>`

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



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

- 1. max 함수가 템플릿이라는 것부터 선언 : `template`
- 2. 템플릿 파라미터를 꺾쇠로 묶어 나열 : `<>`
 - 2-1. 템플릿 파라미터는 값이라기 보다는 일종의 타입 : `typename`
 - 2-2. `typename` 뒤에는 파라미터의 이름을 선언 (원하는 이름 가능) : `T`
- 함수의 호출자가 템플릿 파라미터로 타입을 제공하면, 템플릿은 `T` 파라미터를 해당 타입에 맞춰 다르게 됨

Function template

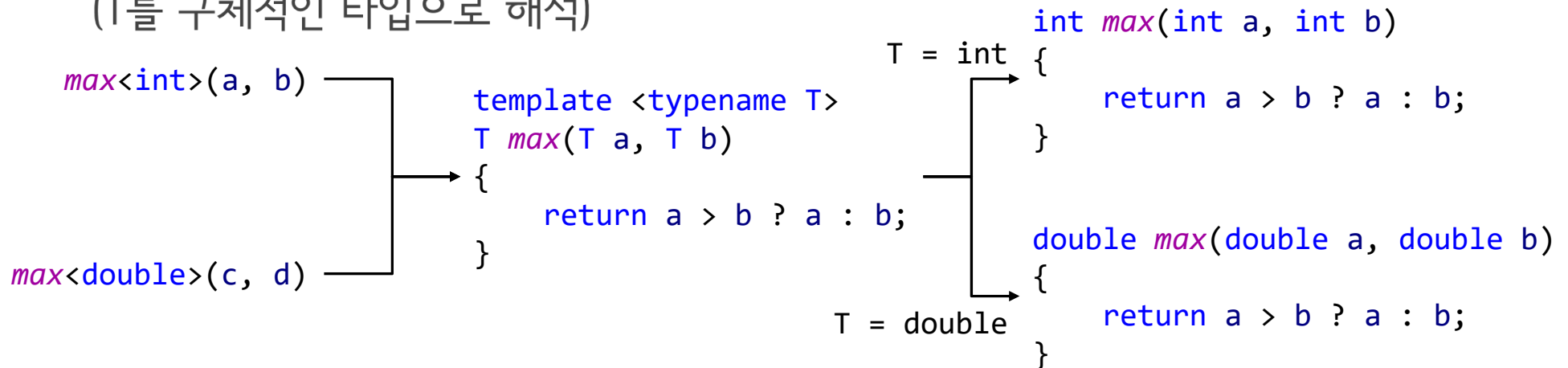
- 함수 템플릿을 호출하는 방법 : `max<타입>(매개변수);`

`max(a, b)` `max<int>(a, b)`
`max(c, d)` `max<double>(c, d)`



- 호출하면, T가 등장하는 곳마다 해당 타입으로 교체됨

- 컴파일 할 때, 사용된 모든 타입에 대해 템플릿 인스턴스화함
(T를 구체적인 타입으로 해석)



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

T = int
int max(int a, int b)
{
    return a > b ? a : b;
}

T = double
double max(double a, double b)
{
    return a > b ? a : b;
}
```

Class template

Defines a family of classes

Class template

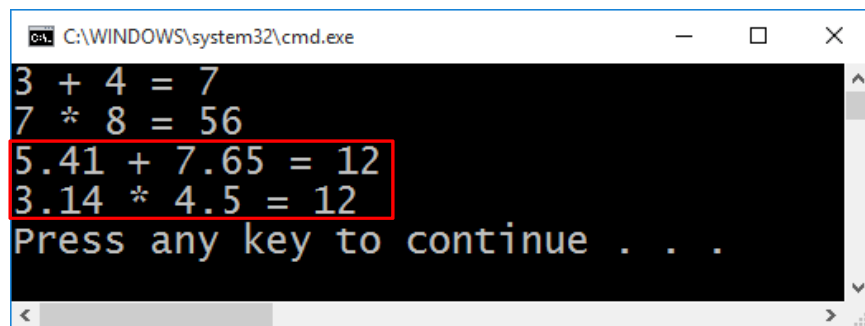
- 정수로만 동작하는 계산기 Calc 클래스 (덧셈, 곱셈 지원)

```
class Calc
{
public:
    int add(int x, int y);
    int multiply(int x, int y);
};

int Calc::add(int x, int y)
{
    return x + y;
}

int Calc::multiply(int x, int y)
{
    return x * y;
}
```

```
int main()
{
    Calc c;
    cout << c.add(3, 4) << endl;
    cout << c.multiply(7, 8) << endl;
    cout << c.add(5.41, 7.65) << endl;
    cout << c.multiply(3.14, 4.5) << endl;
}
```



```
C:\WINDOWS\system32\cmd.exe
3 + 4 = 7
7 * 8 = 56
5.41 + 7.65 = 12
3.14 * 4.5 = 12
Press any key to continue . . .
```

Class template

- 모든 타입에 대해 동작하는 계산기 Calc 클래스

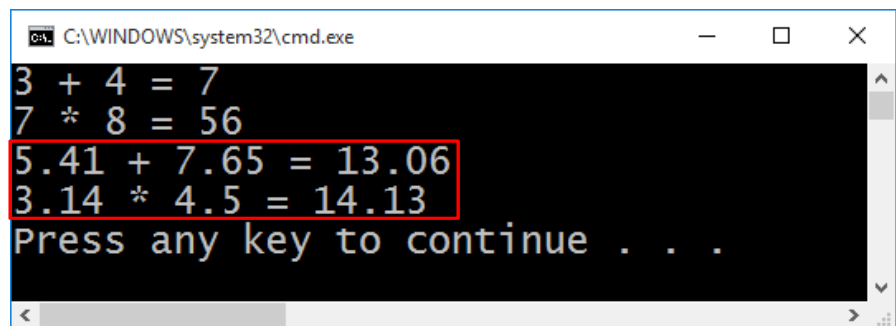
```
template <typename T>
class Calc
{
public:
    T add(T x, T y);
    T multiply(T x, T y);
};
```

```
template <typename T>
T Calc<T>::add(T x, T y)
{
    return x + y;
}
```

```
template <typename T>
T Calc<T>::multiply(T x, T y)
{
    return x * y;
}
```

```
int main()
{
    Calc<int> intCalc;
    cout << intCalc.add(3, 4) << endl;
    cout << intCalc.multiply(7, 8) << endl;

    Calc<double> doubleCalc;
    cout << doubleCalc.add(5.41, 7.65) << endl;
    cout << doubleCalc.multiply(3.14, 4.5) << endl;
}
```



```
C:\WINDOWS\system32\cmd.exe
3 + 4 = 7
7 * 8 = 56
5.41 + 7.65 = 13.06
3.14 * 4.5 = 14.13
Press any key to continue . . .
```