Lecture 20

Template I

Function Template

Prof. Hyeong-Seok Ko
Seoul National University
Graphics & Media Lab

Contents

- Generic Programming
- Function Template (16.1.1, 16.1.6)

Generic Programming

- A style of computer programming in which algorithms/classes are written in terms of **to-be-specified-later types**.

Motivation: Avoid Code Duplication

- Example

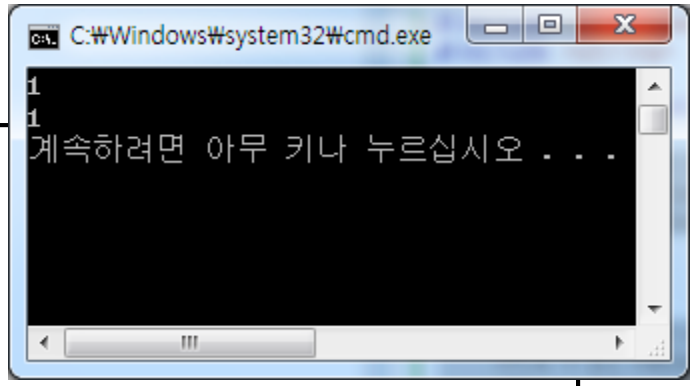
```
int compare(const string& a, const string& b) {
    if(a < b) return -1;
    if(a > b) return +1;
    return 0;
}

int compare(const double& a, const double& b) {
    if(a < b) return -1;
    if(a > b) return +1;
    return 0;
}

void main() {
    const string a("Program"), b("Methodology");
    const double c(2.0), d(1.0);

    cout << compare(a, b) << endl;
    cout << compare(c, d) << endl;
}
```

Duplication !

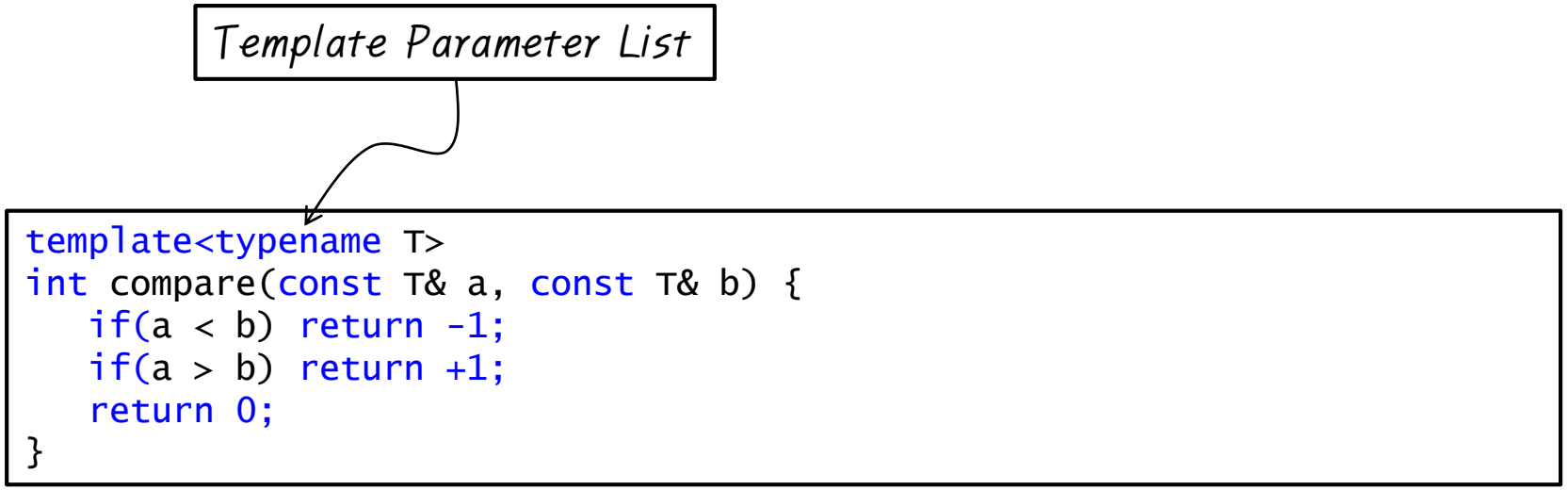


Template

- A feature of the C++ programming language which allows functions and classes to operate with generic types.

- Template allows a function or a class to work on many different data types without being rewritten for each one.

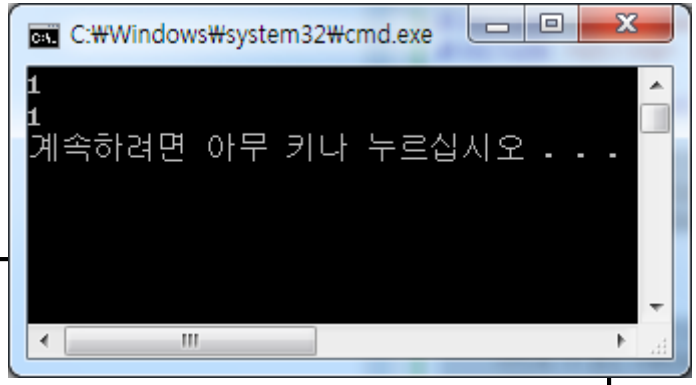
Function Template



- Actual instantiation of T is determined based on how the function is called.
  - T can be `int` or `double` or `std::string` or …

Instantiation

- Example




```
template<typename T>
int compare(const T& a, const T& b) {
    if(a < b) return -1;
    if(a > b) return +1;
    return 0;
}

void main() {
    const string a("Program"), b("Methodology");
    const double c(2.0), d(1.0);

    cout << compare(a, b) << endl;
    cout << compare(c, d) << endl;
}
```

Inline Function Template

- A function template can be declared `inline` in the same way as a non-template function.

```
template<typename T>
inline int compare(const T& a, const T& b) {
    if(a < b) return -1;
    if(a > b) return +1;
    return 0;
}

void main() {
    const string a("Program"), b("Methodology");
    const double c(2.0), d(1.0);

    cout << compare(a, b) << endl;
    cout << compare(c, d) << endl;
}
```