Lecture 23

Template IV

Generic Algorithms

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

Contents

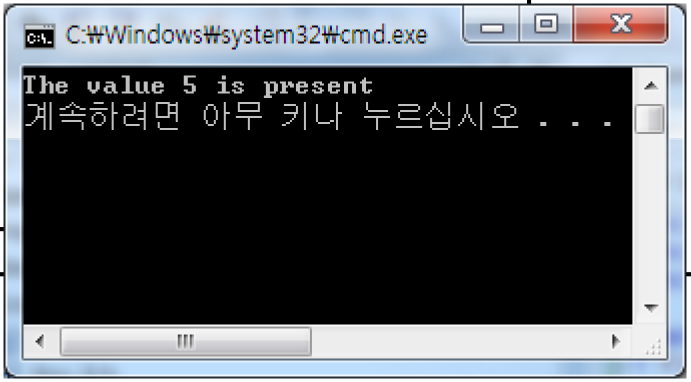
- Generic Algorithm (11.1, 16.1.6)

Generic Algorithm

- The Standard Template Library (STL) provides a set of **generic algorithms**.
  - "Algorithm" because they implement commonly used operations.
  - "Generic" because they operate across multiple container types. (not only std::vector or std::list, but also the built-in array types and so on.)

An Example of Simple Generic Algorithm

```
template<typename T, typename S>
T find(T begin, T end, S val) {
   for(T it = begin ; it != end ; it++) {
      if(*it == val)
         return it;
   }
   return end;
}
```


C:\Windows\system32\cmd.exe

```
The value 5 is present
계속하려면 아무 키나 누르십시오 . . .
```

```
void main() {
   int ia[5] = { 1,2,3,4,5 };
   int val = 5;

   int * result = find(ia,ia+5,val); // T is int*, S is int

   cout << "The value " << val
        << (result == ia+5 ? " is not present"
          : " is present") << endl;
}
```

An Example of Simple Generic Algorithm

- Same function can be applied to various types of containers.

```
template<typename T, typename S>
T find(T begin, T end, S val) {
    for(T it = begin ; it != end ; it++) {
        if(*it == val)
            return it;
    }
    return end;
}
```

```
int val = 5;

int * result = find(ia, ia+5, val);
vector<int>::const_iterator result = find(vec.begin(), vec.end(), val);
list<int>::const_iterator result = find(lst.begin(), lst.end(), val);

// ia is integer array
// vec is std::vector<int>
// lst is std::list<int>
```

Generic Algorithm + Functor

```
template<typename T, typename F>
std::size_t count_if(T begin, T end, F& func) {
    std::size_t num = 0;
    for(T it = begin ; it != end ; it++) {
        if(func(*it) == true)
            ++num;
    }
    return num;
}
```

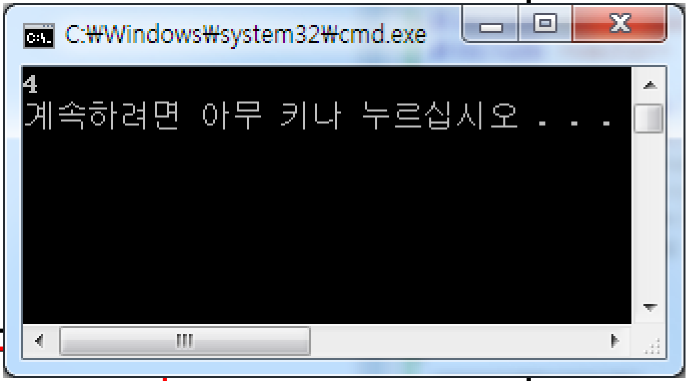
```
class GT_cls {
public :
   GT_cls(std::size_t b) : bound(b) {}
   bool operator()(const std::string& s) { return s.size() >= bound; }

private :
   std::size_t bound;
};

void main() {
   std::vector<std::string> words;

   words.push_back("Programming"); words.push_back("Methodology");
   words.push_back("is");          words.push_back("easy");
   words.push_back("or");          words.push_back("not easy");

   cout << count_if(words.begin(), words.end(), GT_cls(3)) << endl;
}
```



```
4
계속하려면 아무 키나 누르십시오 . . .
```

Generic Algorithm + Functor

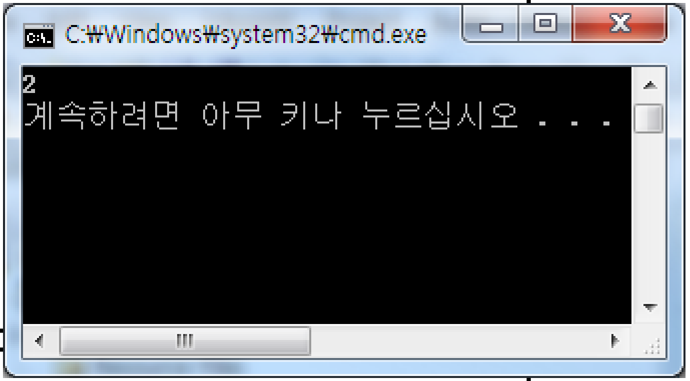
```
template<typename T, typename F>
std::size_t count_if(T begin, T end, F& func) {
    std::size_t num = 0;
    for(T it = begin ; it != end ; it++) {
        if(func(*it) == true)
            ++num;
    }
    return num;
}
```

```
class GT_integer {
public :
   GT_integer(int b) : bound(b) {}
   bool operator()(const int x) { return x >= bound; }

private :
   int bound;
};

void main() {
   int ia[5] = { 0,1,2,3,4 };

   cout << count_if(ia, ia+5, GT_integer(3)) << endl;
}
```



```
2
계속하려면 아무 키나 누르십시오 . . .
```

Generic Algorithms in STL

- You can find a number of useful generic algorithms in Standard Template Library (STL).
  - `find, count, copy, replace, remove, sort, …`
  - http://www.cplusplus.com/reference/algorithm/