Lecture 19

Functors

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

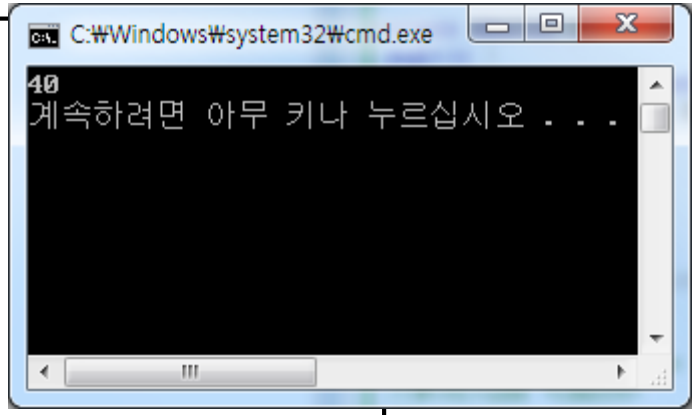
Contents

- Call Operator and Function Object (14.8.1)

Call Operator

- A **call operator** can be overloaded for the class.

```
class AbsInt {
public :
   int operator()(int val) {
      return val < 0 ? - val : val;
   }
};

void main() {
   AbsInt absint;
   std::cout << absint(-40) << std::endl;
}
```



Function Object (Functor)

- Even though `AbsInt` is a class and not a function, we can make a "call" on an object of `AbsInt`.

- Class objects which can be used with the call operator are referred to as **function objects** or **functors**.
  - They are objects that act like functions

```
class AbsInt {
public :
   int operator()(int val) {
      return val < 0 ? - val : val;
   }
};

void main() {
   AbsInt absint;
   std::cout << absint(-40) << std::endl;
}
```

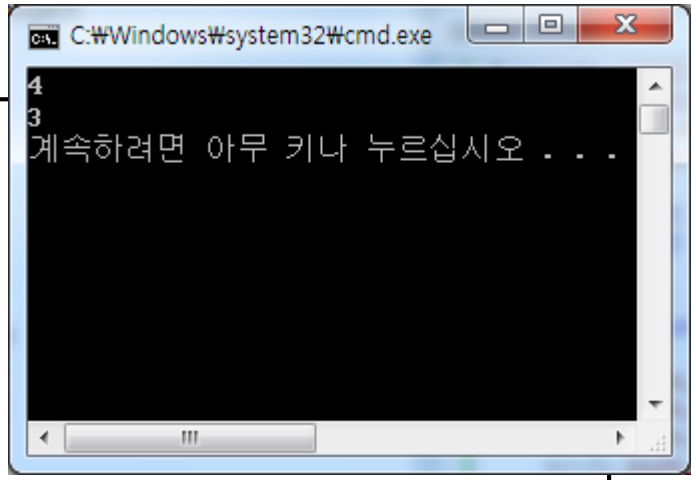
Example Usage of Functors

`count_if(first, last, pred)`

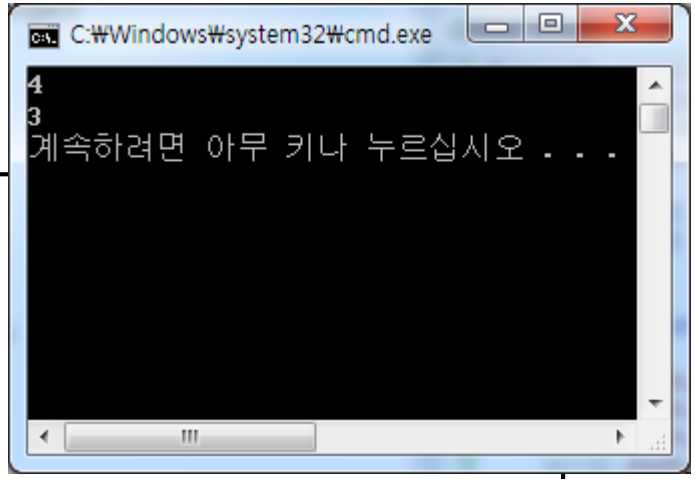
- A function in std algorithm
  - Returns the number of elements in the range `[first,last)` for which the predicate **`pred`** is true.
  - **`pred`** can be a function or function object.

With Functions

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

bool GT3(const std::string& s) {
   return s.size() >= 3;
}
bool GT6(const std::string& s) {
   return s.size() >= 6;
}

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");

   std::cout << count_if(words.begin(), words.end(), GT3) << std::endl;
   std::cout << count_if(words.begin(), words.end(), GT6) << std::endl;
}
```



With Functors

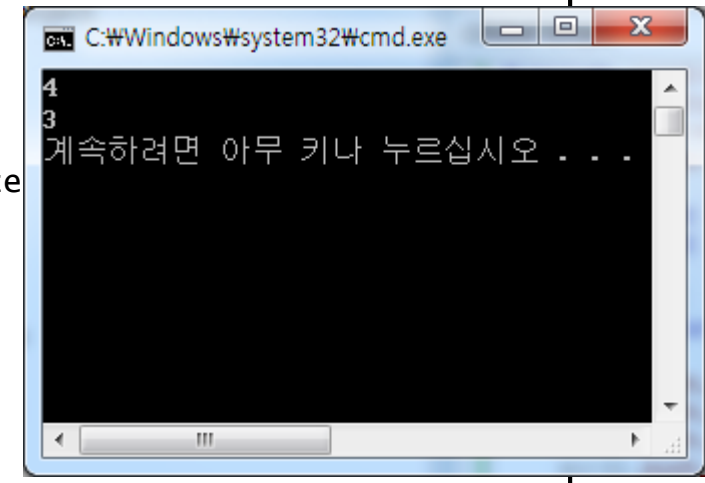



```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

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;
   GT_cls a(3), b(6);
   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");

   std::cout << count_if(words.begin(), words.end(), a) << std::endl;
   std::cout << count_if(words.begin(), words.end(), b) << std::endl;
}
```

With Functors

- Instead of creating an object, the functor can be instantiated with the constructor call.
  - This is in fact more recommended.

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

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

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");

   std::cout << count_if(words.begin(), words.end(), GT_cls(3)) << std::endl;
   std::cout << count_if(words.begin(), words.end(), GT_cls(6)) << std::endl;
}
```



Functors can be more flexible than functions

- A functor can be used with a variable argument.
  - Each of such a functor corresponds to a different version of the function.
  - This is the feature which cannot be enjoyed with normal functions.

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

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

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");

   for(size_t i=3;i<10;++i)
      std::cout << count_if(words.begin(), words.end(), GT_cls(i)) << std::endl;
}
```