Lecture 11

Object-Oriented Programming VII

Copy constructor

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

Contents

- Copy Constructor (13.1)

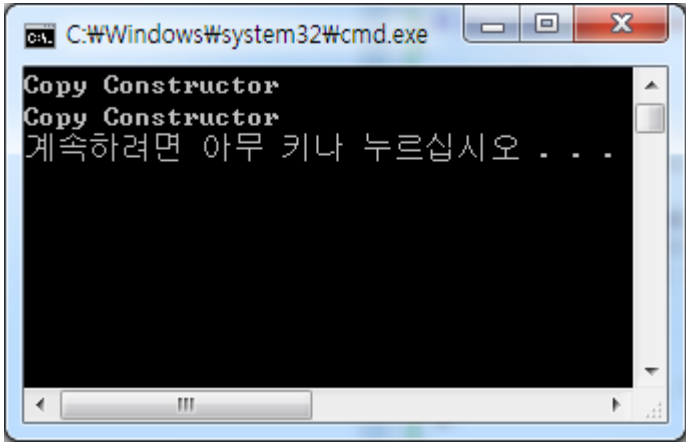
What is Copy Constructor ?

- A member function which is automatically called whenever a copy of a class object is created.
- The copy constructor takes a single parameter, the reference (usually `const`) to the object from which the copy is made.

```
#include <iostream>

class Box {
public :
   Box(const Box& b)
   : width(b.width), height(b.height), length(b.length) {}

private :
   double width, height, length;
};

void main() {
   Box box(1,2,3);

   Box box_1(box);     // copy constructor is called.
}
```

When Copy Constructor is Called?

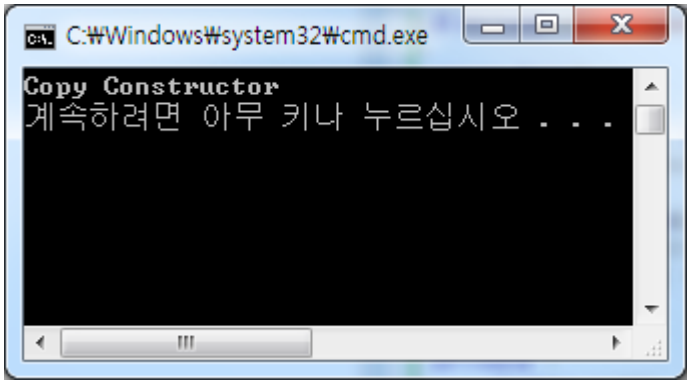
- When creating objects




```
#include <iostream>

class Box {
public :
   Box(double w, double h, double l)
   : width(w), height(h), length(l) {}

   Box(const Box& b)
   : width(b.width), height(b.height), length(b.length) {
      std::cout << "Copy Constructor" << std::endl;
   }

private :
   double width, height, length;
};

void main() {
   Box box(1,2,3);

   Box box_1(box);              // Copy constructor is called.
   Box box_3; box_3 = box;      // Copy constructor is not called.
                                // Assignment operator is called instead.
   Box box_2 = box;             // Copy constructor is called.
}
```

When Copy Constructor is Called?

- When calling a function (call-by-value)




```
#include <iostream>

class Box {
public :
   Box(double w, double h, double l)
   : width(w), height(h), length(l) {}

   Box(const Box& b)
   : width(b.width), height(b.height), length(b.length) {
      std::cout << "Copy Constructor" << std::endl;
   }

private :
   double width, height, length;
};

void func(Box box) { // copy constructor is called
   // ...
}

void main() {
   Box box1(1,2,3);
   func(box1);
}
```

Default Copy Constructor

- If we do not define the copy constructor, the compiler automatically creates one for us.
- You need to note that the default copy constructor makes member-wise copy only at the top-level.

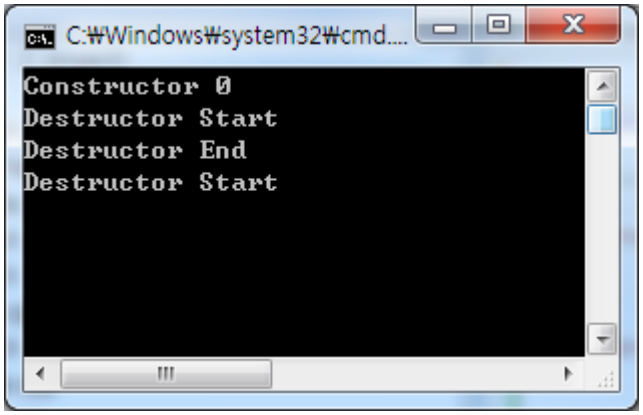
```
#include <iostream>

class Array {
public :
   Array(std::size_t num) : size(num) { ptr = new int[num]; }
   ~Array() { if(ptr != NULL) delete [] ptr; }

   Array(const Array& array) : ptr(array.ptr), size(array.size) {}

   int *          ptr;
   std::size_t    size;
};
```

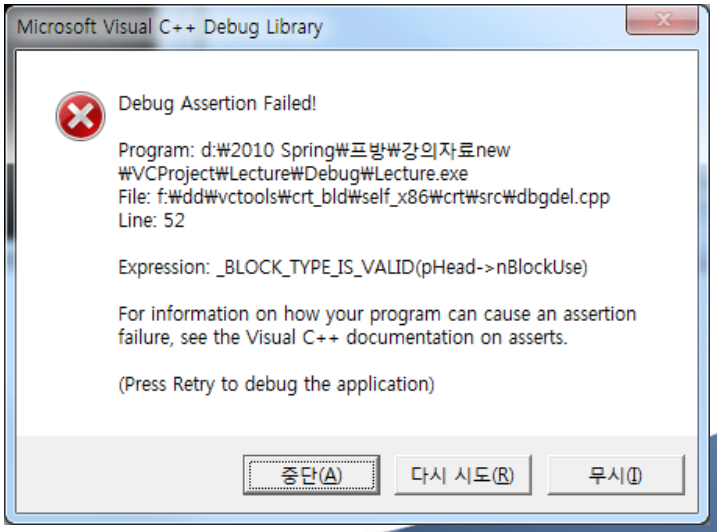
Default copy constructor created automatically by the compiler

You May Have to Define Your Own Copy Constructor

- Default copy constructor can be problematic.

```
#include <iostream>

class Array {
public :
   Array(std::size_t num) : size(num) {
      std::cout << "Constructor 0" << std::endl;
      ptr = new int[num];
   }
   ~Array() {
      std::cout << "Destructor Start" << std::endl;
      if(ptr != NULL) delete [] ptr;
      std::cout << "Destructor End" << std::endl;
   }

   int *            ptr;
   std::size_t      size;
};

void f() {
   Array array(5);
   Array array0(array);
}

void main() {
   f();
}
```

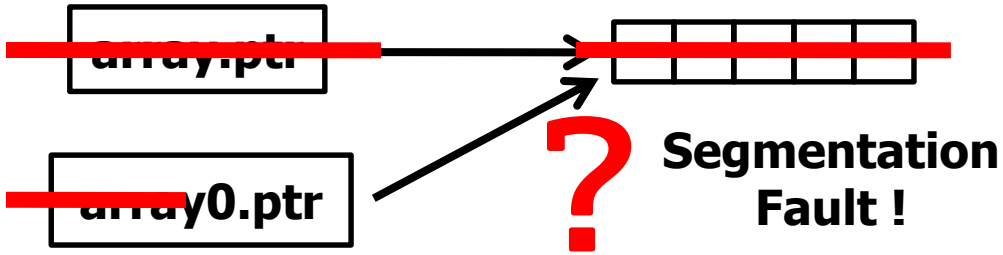




You May Have to Define Your Own Copy Constructor

- Default copy constructor can be problematic.

```
#include <iostream>

class Array {
public :
   Array(std::size_t num) : size(num) {
      std::cout << "Constructor 0" << std::endl;
      ptr = new int[num];
   }
   ~Array() {
      std::cout << "Destructor Start" << std::endl;
      if(ptr != NULL) delete [] ptr;
      std::cout << "Destructor End" << std::endl;
   }

   int *            ptr;
   std::size_t      size;
};

void f() {
   Array array(5);
   Array array0(array);
}

void main() {
   f();
}
```



How to Define Your Own Copy Constructor?

```
#include <iostream>

class Array {
public :
   Array(std::size_t num) : size(num) {
      std::cout << "Constructor 0" << std::endl;
      ptr = new int[num];
   }
   Array(const Array& arr) : size(arr.size) {
      std::cout << "Copy Constructor" << std::endl;
      ptr = new int[size];
      for(std::size_t i=0;i<size;++i)
         ptr[i] = arr.ptr[i];
   }
   ~Array() {
      std::cout << "Destructor Start" << std::endl;
      if(ptr != NULL) delete [] ptr;
      std::cout << "Destructor End" << std::endl;
   }

   int *         ptr;
   std::size_t   size;
};

void f() {
   Array array(5);
   Array array0(array);
}

void main() {
   f();
}
```