Object-Oriented Programming

Lesson 7

Copy constructor in complex objects Static const members

Upcasting and the copy-constructor

```
class base
{   char* name;
    void copy(char* str);
public:
    base(char* str = 0);
    base (const base& r);
    virtual ~base();
};
```

```
base::base(char* str)
{   copy (str); }

base::base(const base& r)
{   copy (r.name); }

base::~base()
{   delete [] name; }
```

```
void base::copy(char* str)
{  if (str)
   {
      name = new char [strlen(str)+1];
      strcpy_s (name, strlen(str)+1 , str);
   }
   else
      name = 0;
}
```

Upcasting and the copy-constructor

```
class derived : public base
{  int index;
public:
   derived(char* s = 0, int i = 0);
   derived (const derived& r);
   ~derived() {};
};
```

```
derived::derived (char* s, int i)
       : base (s)
{ index = i; }
```

```
derived::derived (const derived& r)
       : base(r)
{ index = r.index; }
```

Remember to properly call (in the constructor initializer list) the base-class copy-constructor whenever you write your own copy-constructor!
If you forget, then the default constructor will automatically be called for the base-class!

Composition and the copy-constructor

```
class A
{  char* name;
public:
   A(char* str = 0);
   A (const A& r);
   virtual ~A();
};
```

```
class B
{
      A object;
public:
      B (char* str);
      B (const B& r);
};
```

Class B includes an object of type A.

Composition and the copy-constructor

When implementing the copy-constructor of the class B remember to properly call (in the constructor initializer list) the copy-constructor of the class A!
If you forget, then the default constructor of the class A will automatically be called!

```
B::B (const B& r) : object(r.object) {}
```

The copy-constructor in complex objects

If you do not implement the copy-constructor in the derived class, the compiler then synthesizes the copy-constructor (since that is one of the four functions it will synthesize, along with the default constructor – if you don’t create any constructors – and the destructor) by calling the base-class copy-constructor, copy-constructors for subobjects and making a bitwise copy of the remaining data members.

Static keyword

- static variables and objects
- static members (data and functions)
- visibility control

Static data members

There is a single piece of storage for a static data member, regardless of how many objects of that class you create. All objects share the same static storage space for that data member.

Access control functions in the same way as for non-static data members.

The static data members can be accessed:

1) as non-static data members;
2) using their complete name, including the scope resolution operator :: and the class name, for example:

```
obj.static_var
pobj->static_var
my_class::static_var
```

Static data members

Because static data has a single piece of storage regardless of how many objects are created, that storage must be defined in a single place.

The linker will report an error if a static data member is declared but not defined.

The definition must occur outside the class (no inlining is allowed), and only one definition is allowed. Thus, it is common to put it in the implementation file (*.cpp) for the class.

```
class my_class
{
        static int s_n;
        //…
};
```

```
int my_class::s_n = 10;
```

The type has to be repeated in the my_class.cpp file!

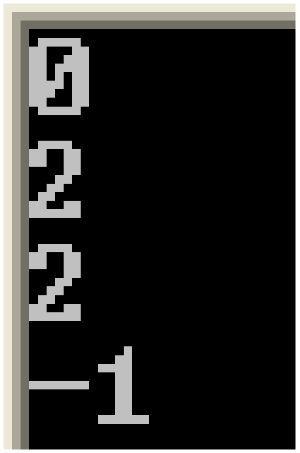
Static data members

```
class CStatic
{
public:
   static int s_nInstances;
   CStatic();
   virtual ~CStatic();
};
```

```
int CStatic::s_nInstances = 0;

CStatic::CStatic()
{    s_nInstances++;   }
CStatic::~CStatic()
{    s_nInstances--;   }
```

```
void main()
{
   {
         cout << CStatic::s_nInstances << endl;
         CStatic s1, s2;
         cout << CStatic::s_nInstances << endl;
         CStatic s3 = s2;
         cout << s2.s_nInstances << endl;
   }
         cout << CStatic::s_nInstances << endl;
}
```



```
0
2
2
-1
```

Static const data members

A static data member can be const.

There will be a single piece of storage for a static const data member, to be shared by all the class instances, and that piece of storage, once initialized, cannot be modified.

```
class my_class
{
   static int const A;
   static const int B;
public:
   //...
};

int const my_classe::A = 1;
const int my_classe::B = 2;
```

Bibliography

Bruce Eckel, Thinking in C++, 2nd edition, MindView, Inc., 2003

=> Chapters 10, 14