Use of Copy Constructors in C++

Rajeev Srivastava

Abstract

The basic idea of that article is that how to create new object from an existing one. It will also explain the syntax of copy constructor for e.g. logic behind using const parameter, and shall also put light on the conditions in which copy constructor are made and the conditions in which copy constructor should be avoided. It also explain the difference between copy constructor and assignment operator.

The *copy constructor* lets you create a new object from an existing one by initialization. A copy constructor of a class A is a nontemplate constructor in which the first parameter is of type A&, const A&, volatile A&, or const volatile A&, and the rest of its parameters (if there are any) have default values.

If you do not declare a copy constructor for a class A, the compiler will implicitly declare one for you, which will be an inline public member.

The implicitly declared copy constructor of a class A will have the form A::A(const A&) if the following are true:

- The direct and virtual bases of A have copy constructors whose first parameters have been qualified with **const** or **const volatile**
- The nonstatic class type or array of class type data members of A have copy constructors whose first parameters have been qualified with **const** or **const volatile**

If the above are not true for a class A, the compiler will implicitly declare a copy constructor with the form A::A(A&).

The compiler cannot allow a program in which the compiler must implicitly define a copy constructor for a class A and one or more of the following are true:

- Class A has a nonstatic data member of a type which has an inaccessible or ambiguous copy constructor.
- Class A is derived from a class which has an inaccessible or ambiguous copy constructor.

The compiler will implicitly define an implicitly declared constructor of a class A if you initialize an object of type A or an object derived from class A.

An implicitly defined copy constructor will copy the bases and members of an object in the same order that a constructor would initialize the bases and members of the object.

Copy constructor syntax

The copy constructor takes a reference to a const parameter. It is const to guarantee that the copy constructor doesn't change it, and it is a reference because a value parameter would require making a copy, which would invoke the copy constructor, which would make a copy of its parameter, which would invoke the copy constructor, which ...

Lecturer (IT Department) IMS, Dehradun

```
//=== file Point.h ============================================
class Point {
    public:
        . . .
        Point(const Point& p);    // copy constructor
        . . .
//=== file Point.cpp ==========================================
. . .
Point::Point(const Point& p) {
    x = p.x;
    y = p.y;
}
    . . .
//=== file my_program.cpp =====================================
. . .
Point p;            // calls default constructor
Point s = p;        // calls copy constructor.
p = s;              // assignment, not copy constructor.
```

Here is an example of a copy constructor for the Point class, which doesn't really need one because the default copy constructor's action of copying fields would work fine, but it shows how it works.

Difference between copy constructor and assignment

A copy constructor is used to initialize a *newly declared* variable from an existing variable. This makes a deep copy like assignment, but it is somewhat simpler:

1. There is no need to test to see if it is being initialized from itself.

- A variable is declared which is *initialized from another object*, eg,
  - `Person q("Rajeev");  // constructor is used to build q.`
  - `Person r(p);  // copy constructor is used to build r.`
  - `Person p = q;  // copy constructor is used to initialize in declaration.`

    `p = q;  // Assignment operator, no constructor or copy constructor.`
- A value parameter is initialized from its corresponding argument.

  `f(p);  // copy constructor initializes formal value parameter.`
- An object is returned by a function.

2. There is no need to clean up (eg, delete) an existing value (there is none).

3. A reference to itself is not returned.

When copies of object are made

A *copy constructor* is called whenever a new variable is created from an object. This happens in the following cases (but not in assignment).

C++ calls a *copy constructor* to make a copy of an object in each of the above cases. If there is no copy constructor defined for the class, C++ uses the default copy constructor which copies each field, ie, makes a *shallow copy*.

Don't write a copy constructor if shallow copies are ok

If the object has no pointers to dynamically allocated memory, a shallow copy is probably sufficient. Therefore the default copy constructor, default assignment operator, and default destructor are ok and you don't need to write your own.

If you need a copy constructor ,you also need a destructor and operator=

If you need a copy constructor, it's because you need something like a deep copy, or some other management of resources. Thus is is almost certain that you will need a destructor and override the assignment operator.

Conclusion

Copy constructor should be used when one wants to create object from existing one of same the class or to return object from function or when it is required to pass object as parameter during call of function. Now the case is, when we try to initialize one object from existing one by using assignment operator then copy constructor will not be called.

Reference

Books

Herbert Schildt	: The complete reference C++
Robert Lafore	: Object Oriented Programming in Turbo C++
Ivor Horton's	: Beginning C++ the complete language
Tony Gaddis	: Starting out with C++
Deitel & Deitel	: How to program C++
Savitch	: Problem solving with C++
Dale, Weems, Headington	: Programming in C++

Websites

www.csa.iisc.ernet.in/resources/documentation/tutorials

www.linux-delhi.org

www.cs.wnstl.edu

www.cplus.about.com

www.icee.ru