Shallow Copy and Deep Copy

Example of Shallow Copy vs. Deep Copy

When defining an overloaded assignment operator or a copy constructor, if your code simply copies the contents of member variables from one object to the other that is known as a shallow copy. The default assignment operator and the default copy constructor perform shallow copies. If there are no pointers or dynamically allocated data involved, this works fine. If some member variable names a dynamic array (or points to some other dynamic structure), then you normally do not want a shallow copy. Instead, you want to create a copy of what each member variable is pointing to, so that you get a separate but identical copy, as illustrated in Invitem Class. This is called a deep copy and is what we normally do when overloading the assignment operator or defining a copy constructor.

Destructor

The destructor of a class is a member function of a class that is called automatically when an object of the class goes out of scope. Among other things, this means that if an object of the class type is a local variable for a function, then the destructor is automatically called as the last action before the function call ends. Destructors are used to eliminate any dynamically allocated variables that have been created by the object so that the memory occupied by these dynamic variables is returned to the freestore manager for reuse. Destructors may perform other clean-up tasks as well. The name of a destructor must consist of the tilde symbol, ~, followed by the name of the class.

Copy Constructors

A copy constructor is a constructor that has one parameter that is of the same type as the class. The one parameter must be a call-by-reference parameter, and normally the parameter is preceded by the const parameter modifier, so it is a constant parameter. In all other respects a copy constructor is defined in the same way as any other constructor and can be used just like other constructors.

A copy constructor should be defined so that the object being initialized becomes a complete, independent copy of its argument.

A copy constructor can be used just like any other constructor. A copy constructor is also called automatically in certain other situations. Roughly speaking, whenever C++ needs to make a copy of an object, it automatically calls the copy constructor. In particular, the copy constructor is called automatically in three circumstances:

1. When a class object is being declared and is initialized by another object of the same type given in parentheses. (This is the case of using the copy constructor like any other constructor.)

2. When a function returns a value of the class type. 3. Whenever an argument of the class type is “plugged in” for a call-by-value parameter. In this case, the copy constructor defines what is meant by “plugging in.”

If you do not define a copy constructor for a class, C++ will automatically generate a copy constructor for you. However, this default copy constructor simply copies the contents of member variables and does not work correctly for classes with pointers or dynamic data in their member variables. Thus, if your class member variables involve pointers, dynamic arrays, or other dynamic data, you should define a copy constructor for the class.

Copy Constructor: A copy constructor is a constructor that has one call-by-reference parameter that is of the same type as the class. The one parameter must be a call-by-reference parameter; normally, the parameter is also a constant parameter—that is, it is preceded by the const parameter modifier. The copy constructor for a class is called automatically whenever a function returns a value of the class type. The copy constructor is also called automatically whenever an argument is plugged in for a call-by-value parameter of the class type. A copy constructor can also be used in the same ways as other constructors. Any class that uses pointers and the new operator should have a copy constructor.

\*C++ makes a distinction between initialization (the three cases where the copy constructor is called) and assignment. Initialization uses the copy constructor to create a new object; the assignment opera-tor takes an existing object and modifies it so that it is an identical copy (in all but location) of the right-hand side of the assignment.

The Big Three

The copy constructor, the = assignment operator, and the destructor are called the big three because experts say that if you need any of them, you need all three. If any of these is missing, the compiler will create it, but the created item might not behave as you want. Thus, it pays to define them yourself. The copy constructor and overloaded = assignment operator that the compiler generates for you will work fine if all member variables are of predefined types such as int and double. For any class that uses pointers and the new operator, it is safest to define your own copy constructor, overloaded =, and a destructor.