**RULE OF THREE (aka “The Big Three”) LINKS**

**RULE OF THREE:**

<http://en.wikipedia.org/wiki/Rule_of_three_(C%2B%2B_programming)>

1. DESTRUCTOR:

<http://en.wikipedia.org/wiki/Destructor_(computer_science)>

1. COPY CONSTRUCTOR:

By default, only the pointer gets copied, not the object it points to.

<http://en.wikipedia.org/wiki/Copy_constructor> <http://www.codersource.net/cpp_copy_constructors.html>  
<http://en.wikipedia.org/wiki/Object_copy> (diagram)

1. OVERLOADED ASSIGNMENT OPERATOR

<http://en.wikipedia.org/wiki/Assignment_operator_in_C%2B%2B>

**Other Perspectives on the Rule of Three:**

The Law of the Big Two – Using RAII (Resource Acquisition Is Initialization)

<http://www.artima.com/cppsource/bigtwo.html>

**Why Use Copy Constructors**

Source: <http://www.codersource.net/cpp_copy_constructors.html>

Also see: <http://en.wikipedia.org/wiki/Object_copy> for diagrams of these concepts\

Shallow / Deep copy: <http://www.poppastring.com/CShallowCopyAndDeepCopy.aspx> note: graphic only. C# ref not c++

**A Copy constructor is :**

* a constructor function with the same name as the class
* used to make [**deep copy**](file:///S:\IN%20Faculty_Student\David%20Russell\2015%20Fall%20-%20PROG%202100%20-%20Student\2-C++%20Lessons\Week%2007%20-%20Big%20Three\why%20copy%20constructors.docx#shallow) of objects.

There are 3 important places where a copy constructor is called.

* When an object is created from another object of the same type
* When an object is passed by value as a parameter to a function
* When an object is returned from a function

If a copy constructor is not defined in a class, the compiler itself defines one. This will ensure a shallow copy. If the class does not have pointer variables with dynamically allocated memory, then one need not worry about defining a copy constructor. It can be left to the compiler's discretion.

But if the class has pointer variables and has some dynamic memory allocations, then it is a must to have a copy constructor.

For ex:  
      class A   //Without copy constructor  
      {  
           private:  
           int x;  
           public:  
           A() {A = 10;}  
           ~A() {}  
      }

      class B    //With copy constructor  
      {  
           private:  
           char \*name; //dynamically allocated member – in this case a dynamic array  
           public:  
           B()  
           {  
           name = new char[20];  
           }  
           ~B()  
           {  
           delete name[];  
           }  
     //Copy constructor  
           B(const B &b)  
           {  
           name = new char[20];  
           strcpy(name, b.name);  
           }  
      };

Let us imagine if you don't have a copy constructor for the class B. At the first place, if an object is created from some existing object, we cannot be sure that the memory is allocated. Also, if the memory is deleted in destructor, the delete operator might be called twice for the same memory location.

**Definitions:** Source: Savitch

**Shallow Copy:**

If your code simply copies the contents of the 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.

**Deep Copy:**

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. This is called a deep copy and is what we normally do when overloading the assignment operator or defining a copy constructor.

**An Overloaded Assignment Operator:**

To avoid the corruption of dynamically allocated members and other issues during an assignment (e.g. classVar = classVar), you need to overload the assignment operator (=) so that it will produce a completely independent copy of the thing on the right-hand side of the assignment. In other words, deep copies require overloaded assignment operators. Note that the overloaded operation must be a member, and not a friend of the class.

**A Copy Constructor is :**

* a constructor function with the same name as the class
* used to make deep copies of objects.

There are 3 important places where a copy constructor is called.

* When an object is created from another object of the same type
* When an object is passed by value as a parameter to a function
* When an object is returned from a function