Copy Constructor in C++

We have discussed [introduction to Constructors in C++](http://quiz.geeksforgeeks.org/constructors-c/). In this post, copy constructor is discussed.

**What is a copy constructor?**  
A copy constructor is a member function which initializes an object using another object of the same class. A copy constructor has the following general function prototype:

ClassName (const ClassName &old\_obj);

Following is a simple example of copy constructor.

|  |
| --- |
| #include<iostream>  using namespace std;    class Point  {  private:      int x, y;  public:      Point(int x1, int y1) { x = x1; y = y1; }        // Copy constructor      Point(const Point &p2) {x = p2.x; y = p2.y; }        int getX()            {  return x; }      int getY()            {  return y; }  };    int main()  {      Point p1(10, 15); // Normal constructor is called here      Point p2 = p1; // Copy constructor is called here        // Let us access values assigned by constructors      cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();      cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();        return 0;  } |

Output:

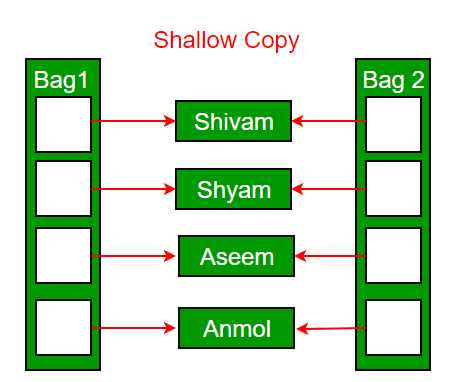
p1.x = 10, p1.y = 15

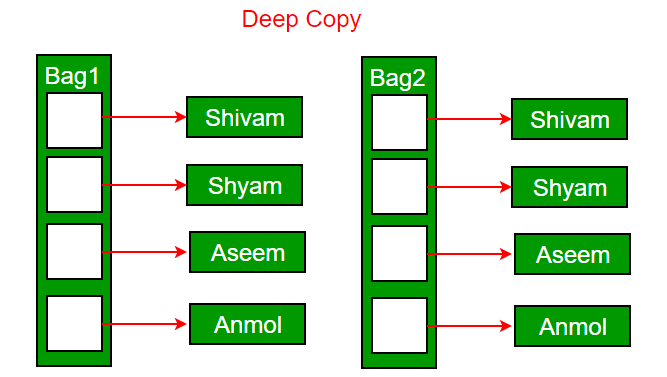
p2.x = 10, p2.y = 15

**When is copy constructor called?**  
In C++, a Copy Constructor may be called in following cases:  
1. When an object of the class is returned by value.  
2. When an object of the class is passed (to a function) by value as an argument.  
3. When an object is constructed based on another object of the same class.  
4. When the compiler generates a temporary object.

It is, however, not guaranteed that a copy constructor will be called in all these cases, because the C++ Standard allows the compiler to optimize the copy away in certain cases, one example is the [return value optimization (sometimes referred to as RVO)](http://en.wikipedia.org/wiki/Return_value_optimization).  
Source: <https://www.geeksforgeeks.org/g-fact-13/>

**When is user-defined copy constructor needed?**  
If we don’t define our own copy constructor, the C++ compiler creates a default copy constructor for each class which does a member-wise copy between objects. The compiler created copy constructor works fine in general. We need to define our own copy constructor only if an object has pointers or any runtime allocation of the resource like file handle, a network connection..etc.

***Default constructor does only shallow copy.***  
[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/copy-constructor.png)

***Deep copy is possible only with user defined copy constructor.*** In user defined copy constructor, we make sure that pointers (or references) of copied object point to new memory locations.  
[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/copy-constructor1.png)

**Copy constructor vs Assignment Operator**  
Which of the following two statements call copy constructor and which one calls assignment operator?

|  |
| --- |
| MyClass t1, t2;  MyClass t3 = t1;  // ----> (1)  t2 = t1;          // -----> (2) |

Copy constructor is called when a new object is created from an existing object, as a copy of the existing object. Assignment operator is called when an already initialized object is assigned a new value from another existing object. In the above example (1) calls copy constructor and (2) calls assignment operator. See [this](https://www.geeksforgeeks.org/copy-constructor-vs-assignment-operator-in-c/)for more details.

**Write an example class where copy constructor is needed?**  
Following is a complete C++ program to demonstrate use of Copy constructor. In the following String class, we must write copy constructor.

|  |
| --- |
| #include<iostream>  #include<cstring>  using namespace std;    class String  {  private:      char \*s;      int size;  public:      String(const char \*str = NULL); // constructor      ~String() { delete [] s;  }// destructor      String(const String&); // copy constructor      void print() { cout << s << endl; } // Function to print string      void change(const char \*);  // Function to change  };    String::String(const char \*str)  {      size = strlen(str);      s = new char[size+1];      strcpy(s, str);  }    void String::change(const char \*str)  {      delete [] s;      size = strlen(str);      s = new char[size+1];      strcpy(s, str);  }    String::String(const String& old\_str)  {      size = old\_str.size;      s = new char[size+1];      strcpy(s, old\_str.s);  }    int main()  {      String str1("GeeksQuiz");      String str2 = str1;        str1.print(); // what is printed ?      str2.print();        str2.change("GeeksforGeeks");        str1.print(); // what is printed now ?      str2.print();      return 0;  } |

Output:

GeeksQuiz

GeeksQuiz

GeeksQuiz

GeeksforGeeks

**What would be the problem if we remove copy constructor from above code?**  
If we remove copy constructor from the above program, we don’t get the expected output. The changes made to str2 reflect in str1 as well which is never expected.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| #include<iostream>  #include<cstring>  using namespace std;    class String  {  private:      char \*s;      int size;  public:      String(const char \*str = NULL); // constructor      ~String() { delete [] s;  }// destructor      void print() { cout << s << endl; }      void change(const char \*);  // Function to change  };    String::String(const char \*str)  {      size = strlen(str);      s = new char[size+1];      strcpy(s, str);  }    void String::change(const char \*str)  {      delete [] s;      size = strlen(str);      s = new char[size+1];      strcpy(s, str);  }    int main()  {      String str1("GeeksQuiz");      String str2 = str1;        str1.print(); // what is printed ?      str2.print();        str2.change("GeeksforGeeks");        str1.print(); // what is printed now ?      str2.print();      return 0;  } |

Output:

GeeksQuiz

GeeksQuiz

GeeksforGeeks

GeeksforGeeks

**Can we make copy constructor private?**  
Yes, a copy constructor can be made private. When we make a copy constructor private in a class, objects of that class become non-copyable. This is particularly useful when our class has pointers or dynamically allocated resources. In such situations, we can either write our own copy constructor like above String example or make a private copy constructor so that users get compiler errors rather than surprises at runtime.

**Why argument to a copy constructor must be passed as a reference?**  
A copy constructor is called when an object is passed by value. Copy constructor itself is a function. So if we pass an argument by value in a copy constructor, a call to copy constructor would be made to call copy constructor which becomes a non-terminating chain of calls. Therefore compiler doesn’t allow parameters to be passed by value.

# Preventing Object Copy in C++ (3 Different Ways)

Many times, user wants that an instance of a C++ class should not be copied at all. So, the question is how do we achieve this ?

There are three ways to achieve this :

1. **Keeping the Copy Constructor and Copy assignment operator as private in the class.**

Below is the C++ implementation to illustrate how this can be done.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| #include <iostream>  using namespace std;    class Base {      int x;  public:      Base()    {  }      Base(int y): x(y)     {  }  private:        // Copy constructor      Base(const Base& obj) : x(obj.x) {   }        // copy assignment operator      Base& operator=(const Base& tmp\_obj)      {          x = tmp\_obj.x;          return \*this;      }  };    int main()  {      Base b1(10);      Base b2(b1); // calls copy constructor      b2 = b1; // calls copy assignment operator      return 0;  } |

NOTE: This code does not compile as we cannot copy the object of this class and hence it will show this error.

prog.cpp: In function 'int main()':

prog.cpp:18:2: error: 'Base::Base(const Base&)' is private

Base(const Base &obj) : x(obj.x) //Copy constructor

^

prog.cpp:33:12: error: within this context

Base b2(b1); // Calls copy constructor

^

prog.cpp:22:8: error: 'Base& Base::operator=(const Base&)' is private

Base& operator = (const Base& tmp\_obj) // copy assignment operator

^

prog.cpp:35:5: error: within this context

b2 = b1; // calls copy assignment operator

^

1. **Inherit a Dummy class with a private copy constructor and a private copy assignment operator.**

Below is the C++ implementation to illustrate how this can be done.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| #include <iostream>  using namespace std;    class Dummy {  public:      Dummy() {  }  private:      Dummy(const Dummy& temp\_obj)  {   }      Dummy& operator=(const Dummy& temp\_obj)   {   }  };    class Base : public Dummy {      int x;  public:      Base()  {   }      Base(int y) : x(y)  {  }  };    int main()  {      Base b1(10);      Base b2(b1); // Calls copy constructor      b2 = b1; // Calls copy assignment operator      return 0;  } |

prog.cpp: In function 'int main()':

prog.cpp:12:5: error:

'Dummy::Dummy(const Dummy&)' is private

Dummy(const Dummy &temp\_obj)

^

prog.cpp:22:7: error: within this context

class Base: public Dummy

^

prog.cpp:16:12: error:

'Dummy& Dummy::operator=(const Dummy&)' is private

Dummy& operator = (const Dummy &temp\_obj)

^

prog.cpp:22:7: error: within this context

class Base: public Dummy

NOTE: This code does not compile as we cannot copy the object of this class and hence it will show this error.

1. **Using Deleted copy constructor and copy assignment operator**: Above two ways are quite complex, C++11 has come up with a simpler solution i.e. just delete the copy constructor and assignment operator.

Below is the C++ implementation to illustrate :

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // CPP program to demonstrate use Delete copy  // constructor and delete assignment operator  #include <iostream>  using namespace std;    class Base {      int x;  public:      Base()   {     }      Base(int y) : x(y) {    }      Base(const Base& temp\_obj) = delete;      Base& operator=(const Base& temp\_obj) = delete;  };    int main()  {      Base b1(10);      Base b2(b1); // Calls copy constructor      b2 = b1; // Calls copy assignment operator      return 0;  } |

prog.cpp: In function 'int main()':

prog.cpp:24:15: error: use of deleted function

'Base::Base(const Base&)'

Base b2(b1); // Calls copy constructor

^

prog.cpp:16:5: note: declared here

Base(const Base &temp\_obj) = delete;

^

prog.cpp:26:8: error: use of deleted function

'Base& Base::operator=(const Base&)'

b2 = b1; // Calls copy assignment operator

^

prog.cpp:17:11: note: declared here

Base& operator = (const Base &temp\_obj) = delete;

^

NOTE: This code does not work as we cannot copy the object of this class and hence it will show this error.