Explaining copy constructor example

A copy constructor is used for many things such as when I need to use pointers or dynamically allocate memory for an object. But looking at this example at `tutorialpoint.com` :

```
#include <iostream>

using namespace std;

class Line
{
public:
  int getLength( void );
  Line( int len );             // simple constructor
  Line( const Line &obj);  // copy constructor
  ~Line();                     // destructor

private:
  int *ptr;
};

// Member functions definitions including constructor
Line::Line(int len)
{
cout << "Normal constructor allocating ptr" << endl;
// allocate memory for the pointer;
ptr = new int;
*ptr = len;
}

Line::Line(const Line &obj)
{
cout << "Copy constructor allocating ptr." << endl;
ptr = new int;
*ptr = *obj.ptr; // copy the value
}

Line::~Line(void)
{
cout << "Freeing memory!" << endl;
delete ptr;
}
int Line::getLength( void )
{
return *ptr;
}

void display(Line obj)
{
   cout << "Length of line : " << obj.getLength() <<endl;
}

// Main function for the program
int main( )
{
   Line line(10);

   display(line);

  return 0;
}
```

the result is :

```
Normal constructor allocating ptr
Copy constructor allocating ptr.
Length of line : 10
Freeing memory!
Freeing memory!
```

and when I commented out (the copy constructor) and the code inside destructor I got the same results:

```
Normal constructor allocating ptr
Length of line : 10
```

So what is the difference between using the copy constructor here or not? Also why does "Freeing Memory!" occur twice?

c++ oop copy-constructor

edited Feb 6 '13 at 21:27
Keith Pinson
4,001 ●5 ●35 ●78

asked Aug 23 '12 at 20:51
Omar
367 ●1 ●5 ●22

2 Look at the results. In the first example, you're allocating two different ints and freeing them both. In the second, you allocate one and free it twice. Not good. – chris Aug 23 '12 at 20:54

its not freed in the second example, i just forgot to comment the "cout<<"freeing memory!" statement inside constructor , so its actually not freed – Omar Aug 23 '12 at 21:02

Then you've allocated something that you haven't freed, which is a memory leak if it starts getting more complex than just creating one and ending the program. – chris Aug 23 '12 at 21:04

i know that @chris but i just want to understand the concept and know what is the difference here between using constructor or not using it?.. – Omar Aug 23 '12 at 21:08

1 You'll find some great references on it via the Rule of Three, which could be argued as either the Rule of Five, the Rule of Four and a Half, or the Rule of Zero nowadays. An example for zero would be to use `std::unique_ptr<int>` instead of `int *`. Then there's no need for worrying about the destructor or copy constructor (or assignment operator, or move constructor/assignment operator). – chris Aug 23 '12 at 21:14

4 Answers

The argument to the `display()` function is passed by value, so the compiler calls the copy constructor to create it. When the class defines its copy constructor you get the correct semantics: the copy constructor makes a copy, and that copy has its own memory to hold the length. When you remove the copy constructor, the compiler generates one for you, and the copy that gets passed to `display()` has the same pointer as the original. When that copy gets destroyed it deletes the memory that ptr points to. When the original gets destroyed it deletes the same memory again (which happens to have no visible effects here). That's definitely not what you want to have happen, which is why you need to define a copy constructor. As @Joe says: inside the destructor, print the value of `ptr' to see this more clearly.

answered Aug 23 '12 at 21:10
Pete Becker
48.2k ●3 ●35 ●94

Print the address of the memory being freed.

I believe you will find the compiler generated the constructor for you, did a value copy of the contents, including the pointer, and you are double-freeing the pointer and just getting lucky that the runtime isn't complaining about it.

The compiler generated copy constructor is still being called - nothing has changed in that regard, you just aren't printing anything from it since you didn't write it.

answered Aug 23 '12 at 20:54
Joe
2,593 ●9 ●12

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 run time allocation of resource like file handle, a network connection..

answered Oct 16 '14 at 10:34
Raja
11 ●1

A constructor of some type T of the form

```
T (const & T);
```

The single argument must be a const reference to an existing object of same type Creates a duplicate of the existing object Used whenever a copy of an object is needed Including arguments to functions, results returned from functions Problems with pointer-based arrays in C++:– No range checking. Cannot be compared meaningfully with == No array assignment (array names are const pointers). If array passed to a function, size must be passed as a separate argument.

answered Jan 15 '14 at 9:37
Abdul Rahman Sajjad
1