Tutorial Link https://codequotient.com/tutorials/Shallow Copy and Deep Copy/5b50a0ec9c655618ac6c0d61

TUTORIAL

Shallow Copy and Deep Copy

Chapter

1. Shallow Copy and Deep Copy

A shallow copy of an object copies all of the member field values. This works well if the fields are values, but may not be what you want for fields that point to dynamically allocated memory. The pointer will be copied. but the memory it points to will not be copied -- the field in both the original object and the copy will then point to the same dynamically allocated memory, which is not usually what you want. The default copy constructor and assignment operator make shallow copies.

A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. To make a deep copy, you must write a copy constructor and overload the assignment operator, otherwise the copy will point to the original, with disasterous consequences. To understand in more details, let us consider the below example: -

This example class is ShallowCQ. This class contains only one integer pointer as private data member as shown below:

```
private:
    int * x;
```

The constructor will create a memory location in a heap and copy the passed in value m to the heap content. This code is shown below:

```
ShallowCQ (int m)
{
    x = new int;
    *x = m;
}
```

The Get and Set functions are used to get the heap memory content value and Set the heap memory content respectively. Below is the code that sets and gets the integer heap memory value:

```
int GetX() const
{
    return *x;
}
void SetX(int m)
{
    *x = m;
}
```

Finally, there is a function to print the heap content value in the console window. The function is shown below:

```
void PrintX()
{
    cout << "Int X=" << *x << endl;
}
```

At present ShallowCQ has a constructor that creates a heap memory and in the destructor we clear the memory created as shown in the below code:

```
~ShallowCQ()
{
    delete x;
}
```

Let us write the main function as below for the above class: -

```
1 int main()
```
C++

```
{
  ShallowCQ ob1(55);
  ShallowCQ ob2 = ob1;        //default copy constructor called.
  ob1.PrintX();
  ob2.PrintX();
  ob1.SetX(77);
  ob1.PrintX();
  ob2.PrintX();

  return 0;
}

```

Output is

```
Int X=55
Int X=55
Int X=77
Int X=77
```

Surprisingly, we modified the data member of the object ob1 only. Then, Why the changes are reflected on both the objects? This is what called **shallow copy** induced by the compiler provided default constructor.

When object ob1 is created, the memory to store an integer is allocated in the heap. Let us assume the heap memory location address is 0x1000. This address is what stored in the x. Remember x is an integer pointer. The value stored in the pointer variable x is the address 0x1000 and the content of the address 0x1000 is value 55. In the example, we want to deal with the content of the address 0x1000 we use the pointer de-referencing like *x. The compiler provided copy constructor copies the address stored in the ob1(x) to ob2 (x). After the copy, both pointers in ob1 and ob2 points to the same object. So changing the 0x1000 through ob1.SetX(77) is reflected back in the ob2.

Now you got how the result is printing 77 for both the objects ob1 and ob2.

How do we avoid the above-shown problem? We should perform the **deep copy** by implementing our own copy constructor. So a user defined copy constructor is required to avoid the problem of shallow copy. Below is the copy constructor:

```
ShallowCQ(const ShallowCQ &obj)
{
    x = new int;        // Create new memory for new objects hence performing deep copy.
    *x = obj.GetX();
}
```

```C++
int main()
{
  ShallowCQ ob1(55);
  ShallowCQ ob2 = ob1;        //default copy constructor called.
  ob1.PrintX();
  ob2.PrintX();
  ob1.SetX(77);
  ob1.PrintX();
  ob2.PrintX();

  return 0;
}

```

Once we inject this copy constructor to the ShallowCQ class, the x pointer in the object ob2 will not point to the same heap location 0x1000. The statement x = new int; will create the new heap location and then copies the value of obj content to new heap location. The output of the program, after introducing our own copy constructor is shown below:

```
Int X=55
Int X=55
Int X=77
Int X=55
```

If we want to assign an object to other with assignment operator, we have to write an overloaded assignment operator for deep copy, otherwise results will be similar to above. For the above class we can overload = (assignment) operator as below: -

```
void operator=(const ShallowCQ &ob)
{
        this->x = new int;
    *x = ob.GetX();
        // Create new memory for new objects hence performing deep
copy.
}
```

So, if an object has pointers to dynamically allocated memory, and the dynamically allocated memory needs to be copied when the original object is copied, then a deep copy is required.