Constructors in C++

NOTE: Constructors should always be public and they must have the same name as the class name.

Types of constructors:

1) **Default constructor** = the constructor that has no parameters.

   If we don't define explicitly any constructor than C++ will automatically generate a default constructor with an empty body. But if we define explicitly a constructor (with or without parameters), then C++ does not create a default constructor. Notice that a default constructor can be defined by the programmer.

   Is called default constructor because is the constructor that is called when we create an object.

Ex:

- Create the default constructor

```
MyClass()
{
  //here you give values to the private fields
}
```

- Use the constructor

```
MyClass mc1;                    //both lines call the default constructor with no parameter

MyClass *pmc1 = new myClass();
```

Care: if you define a parameterized constructor (but you do not define a constructor with no parameter, also called default constructor) than the 2 lines above will not compile since C++ does not generate a default constructor in the case already a constructor exist.

2) **Parameterized constructor** = a constructor that has parameters, used to initialize some or all private fields.

If we define a parameterized constructor, then C++ will not create anymore a default constructor, so a statement like MyClass mc1; will not compile anymore, unless we actually create also a constructor with no parameters, which is called default constructor.

Ex:

- Create the parameterized constructor, for example with 2 parameters:

```
MyClass(int a, int b)
{
        //here I use the parameters a,… to initialize private fields
}
```

- Use the constructor

```
MyClass mc1(3,4);                  //implicit call of the parameterized constructor
MyClass mc1 = MyClass(3,4);        //explicit call of the parameterized constructor
MyClass *mc2 = new MyClass(3,4); //explicit call of the parameterized constructor with pointers
```

Notice that in both cases we did 2 approaches to call the constructor, one by using objects, the other by using a pointer to an object. One creates objects on the stack (is faster but the limits of the stack are fairly low), the other creates them dynamically on the heap slower, but the limits are higher, depend on the machine capacity).

You may have more than one constructor, one default (with no parameter), one or more parameterized constructors. This is called constructor overloading.

3) **Copy constructor** = a constructor that initializes an object using another object of the same class.

If the user doesn't define a copy constructor, then by default C++ will create a copy constructor with a body that does member-wise copy between objects. In general, the default copy constructor is enough, unless an object has pointers or any runtime allocation of resources like file handle, network connections…

Default constructor does only a shallow copy

Deep copy is possible only with user defined copy constructors. In this case we make sure that pointers (or references) of copied object point to new memory location.

Ex:

- Create a copy constructor:

```
MyClass(const MyClass &old_obj)
{
    //use the private fields of obj to initialize the private fields of this instance
    //Ex: a = obj.a; ….
}
```

- Call a copy constructor:

```
MyClass mc1;            //calls the default constructor, which must be defined!
MyClass mc2 = mc1;      //calls implicitly the copy constructor!!
MyClass mc2 (mc1);      //calls explicitly the copy constructor!!
myMethod(mc1);          //calls the copy constructor if and only if
                        //the myMethod is not using references/pointers (pass by value)
```

A common misconception is to mix copy constructor with the assignment operator. See the following example:

```
MyClass mc1, mc2;               //calls the default constructor, which must be defined!
MyClass mc3 = mc1;      //calls implicitly the copy constructor!!

mc3 = mc2;              //the assignment operator is called

                        //equivalent to mc3.operator=(mc2)
```

For more information on assignment operator in C++ lookup the information available on Wikipedia.

A copy constructor is called:

a) When an object of the class is returned by value.
b) When an object of the class is passed (to a function) by value as an argument.
c) When an object is created/constructed based on another object of the same class.
d) When the compiler generates a temporary object.

Obs: not in all cases mentioned above the copy constructor is called because of the optimization that C++ compilers are allowed to do, an example being RVO = return value optimization, which allows the compiler to avoid constructing a temporary object for the return object, but this may have unintended side effects. RVO should be deactivated in C++11. For more details see shaharmike.com/cpp/rvo and shaharmike.com/cpp/move-semantics

4) **Delegating constructors**: in c++11 you may delegate constructors, which is similar to chaining constructors in Java.

Ex:

```
MyClass(int a, int b)
{
        //assume here you are using a and b to set certain member variables
}
MyClass(int a, int b, int c) : MyClass(a,b)          //so we call initially the constructor with 2 parameters
{
        //here I already used the first constructor so all I have to do is
        //use c to set maybe other member variables
}
```

Links with more information:

geeksforgeeks.org/constructors-c

ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.cbclx01/cplr375.htm

wikipedia