CONSTRUCTORS DESTRUCTORS IN C++

BY: MS. GEETANJALI BHOLA

OUTLINE

- Constructor
- Default Constructor
- Parameterized Constructor
- Copy Constructor
- Destructor
- Virtual Destructor// LATER IN INHERITANCE

Constructor

- A Constructor is a member function of a class.
- It is mainly used to initialize the objects of the class.
- It has the same name as the class.
- When an object is created, the constructor is automatically called.
- It is a special kind of member function of a class.
- IT HAS NO RETURN TYPE NOT EVEN VOID

Difference Between Constructor and Other Member Functions:

1. The Constructor has the same name as the class name.
2. The Constructor is called when an object of the class is created.
3. A Constructor does not have a return type.
4. When a constructor is not specified, the compiler generates a default constructor which does nothing.

3 types of constructors:

- Default Constructor
- Parameterized Constructor
- Copy constructor

Default Constructor

- A Default constructor is a type of constructor which doesn’t take any argument and has no parameters.

Even if we do not define any constructor explicitly, the compiler will automatically provide a default constructor implicitly.

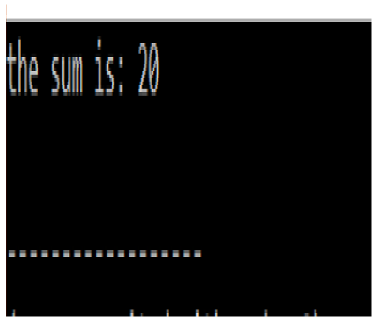
```
// Cpp program to illustrate the
// concept of Constructors
#include <iostream>
using namespace std;

class construct {
public:
	int a, b;

	// Default Constructor
	construct()
	{
		a = 10;
		b = 20;
	}
};

int main()
{
	// Default constructor called automatically
	// when the object is created
	construct c;
	cout << "a: " << c.a << endl
		<< "b: " << c.b;
	return 1;
}
```

```
#include <iostream>
using namespace std;
class test {
public:
int y, z;
test()
{
y = 7;
z = 13;
}
};
int main()
{
test a;
cout <<"the sum is: "<< a.y+a.z;
return 1;
}
```



```
the sum is: 20



------------------
```

PARAMETERIZED CONSTRUCTOR

- Passing of parameters to the constructor is possible. This is done to initialize the value using these passed parameters. This type of constructor is called a parameterized constructor.
  The constructor is defined as follows:

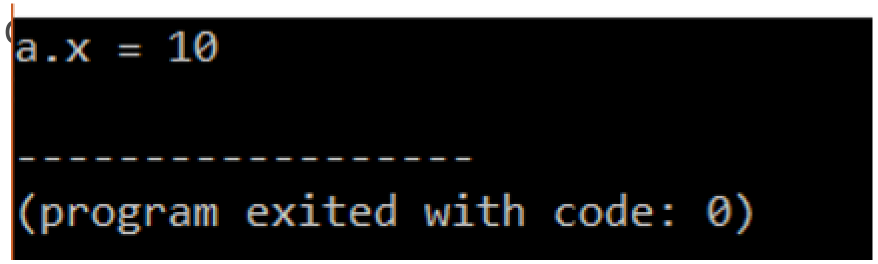
```
test(int x1)
{
x = x1;
}
```

There is a parameter that is passed to the constructor.
The value is passed when the object is created in the main function as shown below.
Inside the main function, we create an object of class test and pass the value of the variable.

```
test t(10);
```

- It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

```
#include <iostream>
using namespace std;
class test {
public:
int x;
test(int x1)
{
x = x1;
}
int getX()
{
return x;
}
};
int main()
{
test a(10);
cout << "a.x = " << a.getX() ;
return 0;
}
```



```
a.x = 10

-------------------
(program exited with code: 0)
```

```
// CPP program to illustrate
// parameterized constructors
#include <iostream>
using namespace std;
class Point {
private:
		int x, y;

public:
		// Parameterized Constructor
		Point(int x1, int y1)
		{
				x = x1;
				y = y1;
		}

		int getX()
		{
				return x;
		}
		int getY()
		{
				return y;
		}
};

int main()
{
		// Constructor called
		Point p1(10, 15);

		// Access values assigned by constructor
		cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();

		return 0;
}
```

Output:

```
p1.x = 10, p1.y = 15
```

When an object is declared in a parameterized constructor, the initial values have to be passed as arguments to the constructor function.

The normal way of object declaration may not work.

The constructors can be called explicitly or implicitly.

```
Example e = Example(0, 50); // Explicit call
Example e(0, 50); // Implicit call
```

Uses of Parameterized constructor:

1.It is used to initialize the various data elements of different objects with different values when they are created.
2.It is used to overload constructors.
Can we have more than one constructors in a class?
Yes, It is called Constructor Overloading.

Copy Constructor

- A Copy Constructor is a Constructor which initializes an object of a class using another object of the same class.
- NOTE: Whenever we define one or more non-default constructors(with parameters) for a class, a default constructor(without parameters) should also be explicitly defined as the compiler will not provide a default constructor in this case. However, it is not necessary but it’s considered to be the best practice to always define a default constructor.

```
// Illustration
#include "iostream"
using namespace std;

class point {
private:
double x, y;

public:
// Non-default Constructor & default Constructor
point (double px, double py) {
                x = px, y = py;
}
};

int main(void) {
point b = point(5, 6);
}
```

When copies of objects are made

- A copy constructor has the following general function prototype:

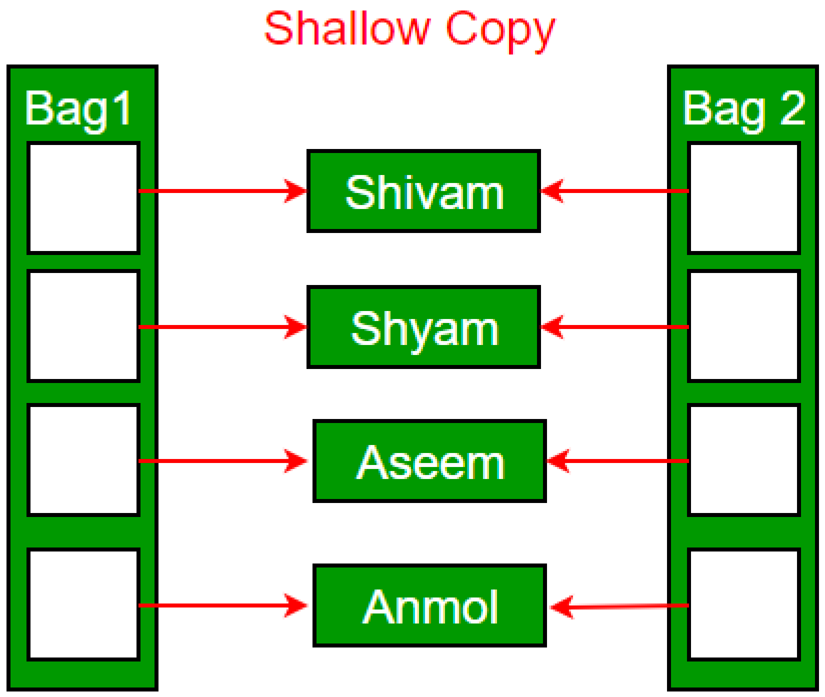
```
ClassName (const ClassName &old_obj);
```

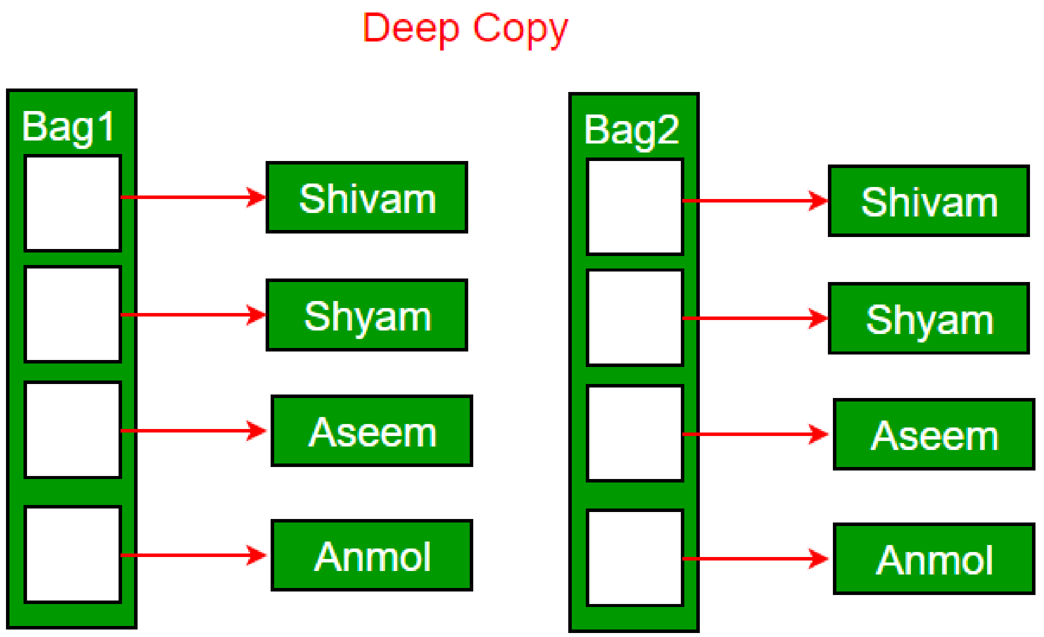
In C++, a Copy Constructor may be called in following cases:
1. When an object of the class is returned by value.
2. When an object of the class is passed (to a function) by value as an argument.
3. When an object is constructed based on another object of the same class.
4. When the compiler generates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

- **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.

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.





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.

- **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.

```
#include<iostream>
using namespace std;
class test
{
private:
int x;
public:
test(int x1)
{
x = x1;
}
test(const test &t2)
{
x = t2.x;
}
int getX()
{
return x;
}
};
int main()
{
test t1(7); // Normal constructor is called here
test t2 = t1; // Copy constructor is called here
cout << "t1.x = " << t1.getX();
cout << "nt2.x = " << t2.getX();
return 0;
}
```

- Why copy constructor argument should be const in C++?

we create our own copy constructor, we pass an object by reference and we generally pass it as a const reference.
One reason for passing const reference is, we should use const in C++ wherever possible so that objects are not accidentally modified.

PARAMETERIZED AS DEFAULT

```
include<iostream>
using namespace std;
class test
{
public:
 int x;
public:
//test(){cout<<" \n default called ";}
test(int x1=0)
{
cout<<"\n parameterized called \n";x = x1;}
};

int main()
{test t2;
test t1(7); // Normal constructor is called here
cout<<"  t1.x "<<t1.x;
cout<<"  t2.x "<<t2.x;
return 0;
}
```

```
parameterized called

parameterized called
 t1.x 7  t2.x 0
```

CAN A constructor be defined in the private section of a class?

```
include<iostream>
using namespace std;
class test
{
private:
            int x;
public:

test()
      {cout<<" \n default called ";
      }

test(int x1)
      {
      cout<<"\n parameterized called \n";x = x1;
      }

test(const test &t2,int i)
      {
      cout<<" \n \n copy constructor for "<< i
      <<"\n";x = t2.x;
      }
test ret_fun(test t3)
      {
      test temp(t3,3);
      temp.x = temp.x +20;
      return temp;
      }
int getX()
      {
      return x;
      }
};
int main()
      {
      test t1(7); // Normal constructor is called here
      test t2(t1,1); // Copy constructor is called here
      cout << "t1.x = " << t1.getX()<<" \n ";
      cout << "t2.x = " << t2.getX();
      test one1;
      test t4(30);
      test result_new(one1.ret_fun(t4),2);
      cout<<"\n \n final result    "<<result_new.getX();
      return 0;
      }
```

```
 parameterized called


 copy constructor for 1
t1.x = 7
 nt2.x = 7
 default called
 parameterized called


 copy constructor for 3


 copy constructor for 2


 final result    50
```