C++ programming constructors

C++ constructors are *special* member functions which are created when the object is created or defined and its task is to initialize the object of its class. It is called **constructor** because it constructs the values of data members of the class.
A constructor has the same name as the class and it doesn't have any return type. It is invoked whenever an object of its associated class is created.
When a class is instantiated, even if we don't declare a constructor, compiler automatically creates one for the program. This compiler created constructor is called **default constructor**.
A constructor is defined as following

```
/*.....class with constructor..........*/
class class_name
{
    .........
public:
    class_name(); //constructor declared or constructor prototype
    .........
};
class_name :: class_name() //constructor defined
{
      //constructor function body
}
```

Types of c++ constructors

- Default Constructor
- Parameterized Constructor
- Copy Constructor

Default constructor

If no constructor is defined in the class then the compiler automatically creates one for the program. This constructor which is created by the compiler when there is no user defined constructor and which doesn't take any parameters is called **default constructor**.

Format of default constructor

```
/*.....format of default constructor..........*/
 class class_name
 {
        .........
         public:
             class_name() { }; //default constructor
             .........
 };
```

Parameterized constructor

To put it up simply, the constructor that can take arguments are called parameterized constructor.

In practical programs, we often need to initialize the various data elements of the different object with different values when they are created. This can be achieved by passing the arguments to the constructor functions when the object is created.
Following sample program will highlight the concept of parameterized constructor

```
 /*.....A program to find area of rectangle .......... */
#include<iostream>
using namespace std;
class ABC
{
   private:
     int length,breadth,x;
   public:
     ABC (int a,int b) //parameterized constructor to initialize l and b
     {
         length = a;
         breadth = b;
      }
      int area( ) //function to find area
      {
         x = length * breadth;
         return x;
      }
      void display( ) //function to display the area
      {
          cout << "Area = " << x << endl;
      }
};

int main()
{
    ABC c(2,4);  //initializing the data members of object 'c' implicitly
    c.area();
    c.display();
    ABC c1= ABC(4,4);  // initializing the data members of object 'c' explicitly
    c1.area();
    c1.display();
    return 0;
 }   //end of program
```

Output

```
Area = 8
Area = 16
```

Note: Remember that constructor is always defined and declared in public section of the class and we can't refer to their addresses.

Copy Constructor

Generally in a constructor the object of its own class can't be passed as a value parameter. But the classes own object can be passed as a reference parameter.Such constructor having reference to the object of its own class is known as copy constructor.

Moreover, it creates a new object as a copy of an existing object. For the classes which do not have a copy constructor defined by the user, compiler itself creates a copy constructor for each class known as default copy constructor.

Example to illustrate the concept of copy constructor

```
/*.....A program to highlight the concept of copy constructor.......... */
#include<iostream>
using namespace std;
class example
{
    private:
           int x;
    public:
        example (int a) //parameterized constructor to initialize l and b
        {
             x = a;
        }
        example( example &b) //copy constructor with reference object argument
        {
            x = b.x;
        }
        int display( ) //function to display
        {
             return x;
        }
};

int main()
{
        example c1(2);    //initializing the data members of object 'c' implicitly
        example c2(c1);   //copy constructor called
        example c3 = c1;
        example c4 = c2;
        cout << "example c1 = " << c1.display() << endl;
        cout << "example c2 = " << c2.display() << endl;
        cout << "example c3 = " << c3.display() << endl;
        cout << "example c4 = " << c4.display() << endl;
        return 0;
}   //end of program
```

Output

```
example c1 = 2
example c2 = 2
example c3 = 2
example c4 = 2
```

Note: In copy constructor passing an argument by value is not possible.

Destructor in c++ programming

C++ destructor is a special member function that is executed automatically when an object is destroyed that has been created by the constructor. C++ destructors are used to de-allocate the memory that has been allocated for the object by the constructor.
Its syntax is same as constructor except the fact that it is preceded by the tilde sign.

```
~class_name() { };   //syntax of destructor
```

Structure of C++ destructors

```
/*...syntax of destructor....*/
 class class_name
 {
      public:
         class_name();    //constructor.
        ~class_name();    //destructor.
}
```

Unlike constructor a destructor neither takes any arguments nor does it returns value. And destructor can’t be overloaded.

C++ Destructor Example

```
/*.....A program to highlight the concept of destructor.......... */
#include <iostream>
using namespace std;
class ABC
{
    public:
        ABC () //constructor defined
       {
            cout << "Hey look I am in constructor" << endl;
       }
       ~ABC() //destructor defined
       {
             cout << "Hey look I am in destructor" << endl;
       }
};

int main()
{
     ABC cc1; //constructor is called
     cout << "function main is terminating...." << endl;
     /*....object cc1 goes out of scope ,now destructor is being called...*/
     return 0;
}  //end of program
```

Output

```
Hey look I am in constructor
function main is terminating....
Hey look I am in destructor
```

Explanation

In the above program, when constructor is called *“Hey look I am in constructor”* is printed then following it *“Function main is terminating…..”* is printed but after that the object `cc1` that was created before goes out of scope and to de-allocate the memory consumed by `cc1` destructor is called and *“Hey I am in destructor”* is printed.

Note: Remember that more than one destructor can’t be used in a program. Only single destructor is allowed.

Program for Tower of Hanoi

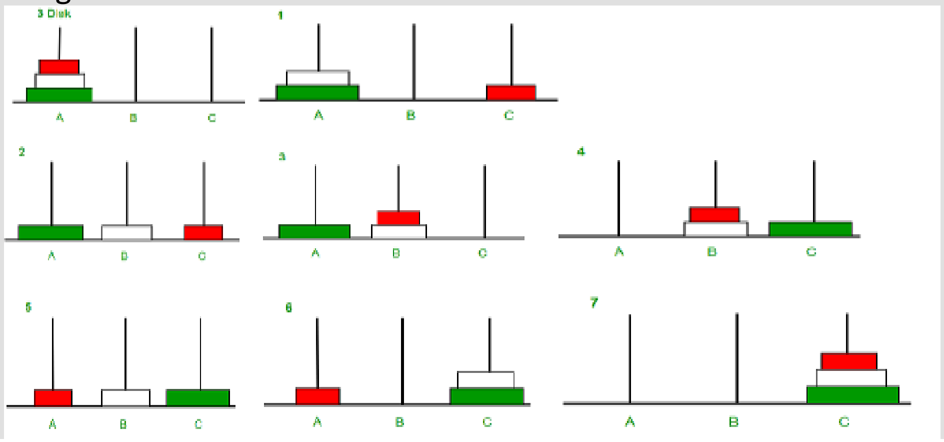
Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:
1) Only one disk can be moved at a time.
2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3) No disk may be placed on top of a smaller disk.
Approach :

```
Take an example for 2 disks :
Let rod 1 = 'A', rod 2 = 'B', rod 3 = 'C'.

Step 1 : Shift first disk from 'A' to 'B'.
Step 2 : Shift second disk from 'A' to 'C'.
Step 3 : Shift first disk from 'B' to 'C'.

The pattern here is :
Shift 'n-1' disks from 'A' to 'B'.
Shift last disk from 'A' to 'C'.
Shift 'n-1' disks from 'B' to 'C'.

Image illustration for 3 disks :
```



Examples:

```
Input : 2
Output : Disk 1 moved from A to B
         Disk 2 moved from A to C
         Disk 1 moved from B to C

Input : 3
Output : Disk 1 moved from A to C
         Disk 2 moved from A to B
         Disk 1 moved from C to B
         Disk 3 moved from A to C
         Disk 1 moved from B to A
```

```
          Disk 2 moved from B to C
          Disk 1 moved from A to C
```

```
// C++ recursive function to
// solve tower of hanoi puzzle
#include <iostream>
using namespace std;

void towerOfHanoi(int n, char from_rod,
                    char to_rod, char aux_rod)
{
    if (n == 1)
    {
        cout << "Move disk 1 from rod " << from_rod <<
                                " to rod " << to_rod<<endl;
        return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    cout << "Move disk " << n << " from rod " << from_rod <<
                                " to rod " << to_rod << endl;
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}

int main()
{
    int n = 4; // Number of disks
    towerOfHanoi(n, 'A', 'B', 'C'); // A, B and C are names of rods
    return 0;
}
```

Output:

```
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
Move disk 4 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 2 from rod C to rod A
Move disk 1 from rod B to rod A
Move disk 3 from rod C to rod B
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
```

For n disks, total $2^n – 1$ moves are required.

eg: For 4 disks $2^4 – 1 = 15$ moves are required.

For n disks, total 2^{n-1} function calls are made.

eg: For 4 disks $2^{4-1} = 8$ function calls are made.