Constructor & Destructor

Constructor

- Constructors are special class members which are ***called by the compiler every time*** an object of that class is instantiated.
- Constructors have the same name as the class and may be defined inside or outside the class definition.
- ***Constructors don't have return type***
- If we do not specify a constructor, C++ compiler generates a default constructor for us (expects no parameters and has an empty body).
- ***Constructors are used to initialize all class data members.***
- There are 3 types of constructors:
  - ✓ Default constructors
  - ✓ Parametrized constructors
  - ✓ Copy constructors

Default Constructors

- Default constructor is the constructor which doesn't take any argument. It has no parameters.

```
1  #include <iostream>
2  using namespace std;
3  class construct
4  {
5  public:
6      int num1, num2;
7      construct() /// Default Constructor
8      {
9          num1 = 10;
10         num2 = 20;
11     }
12 };
13 int main()
14 {
15     construct obj;
16     cout << "First Number is: " << obj.num1 << endl;
17     cout << "Second Number is: " << obj.num2 << endl;
18
19     return 0;
20 }
```

Parameterized Constructors

- 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.

Parameterized Constructors

```
1  #include <iostream>
2  using namespace std;
3  class Point
4  {
5      private:
6          int x, y;
7      public:
8          Point(int xl, int yl) /// Parameterized Constructor
9          {
10             x = xl;
11             y = yl;
12         }
13         int getX()
14         {
15             return x;
16         }
17         int getY()
18         {
19             return y;
20         }
21 };
22 int main()
23 {
24     /// Constructor called
25     Point obj = Point(20, 30); ///Explicit Call
26     Point obj(10, 15);  ///Implicit Call
27     /// Access values assigned by constructor
28     cout << "1st Value = " << obj.getX() << endl;
29     cout << "2nd Value = " << obj.getY() << endl;
30     return 0;
31 }
```

Uses of Parameterized Constructors

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

Constructor Overloading

- In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as Constructor Overloading.
- Overloaded constructors essentially have the same name (name of the class) and different number of arguments.
- A constructor is called depending upon the number and type of arguments passed.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

Constructor Overloading

```
1   #include <iostream>
2   using namespace std;
3   class prime
4   {
5       public:
6           float res;
7           /// Constructor with no parameters
8           prime()
9           {
10              res = 0;
11          }
12          /// Constructor with two parameters
13          prime(int a, int b)
14          {
15              res = a * b;
16          }
17          /// Constructor with three parameters
18          prime(int a, int b, int c)
19          {
20              res = a * b * c;
21          }
22          void disp()
23          {
24              cout<< res<< endl;
25          }
26  };
27  int main()
28  {
29      prime o; prime o2( 10, 20); prime o3( 10, 20, 30);
30      o.disp(); o2.disp(); o3.disp();
31      return 0;
32  }
```

Copy Constructor

- A copy constructor is a member function ***which initializes an object using another object of the same class.*** A copy constructor has the following general function prototype:

ClassName (const ClassName & objName);

Copy Constructor

```
4   class Point
5   {
6       private:
7           int x, y;
8       public:
9           Point(int x1, int y1)
10          {
11              x = x1;
12              y = y1;
13          }
14          Point(const Point &p2)/// Copy constructor
15          {
16              x = p2.x;
17              y = p2.y;
18          }
19
20          int getX()
21          {
22              return x;
23          }
24          int getY()
25          {
26              return y;
27          }
28  };
29  int main()
30  {
31      Point p1(10, 15); /// Normal constructor is called here
32      Point p2 = p1; /// Copy constructor is called here
33      /// Let us access values assigned by constructors
34      cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY() << endl;
35      cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY() << endl;
36      return 0;
37  }
```

Copy Constructor

```
#include <iostream>
using namespace std;
class Wall {
      private:
            double length; double height, l, h;
      public:
        // initialize variables with parameterized constructor
        Wall(double len, double hgt) {
            length = len; height = hgt;
        }
        // copy constructor with a Wall object as parameter copies data of the obj parameter
        Wall(Wall &obj) {
            l = obj.length; h = obj.height; cout<<l<<" "<<h<<endl;
        }
        void calculateArea() {
                cout<<length * height<<endl;;
        }
        void calculateArea2() {
                cout<<l * h<<endl;;
        }
};
int main() {
      // create an object of Wall class
      Wall wall1(10.5, 8.6);
      // copy contents of wall1 to wall2
      Wall wall2 = wall1;
      // print areas of wall1 and wall2
      wall1.calculateArea(); wall2.calculateArea2();
   return 0;
}
```

When is Copy Constructor Called!!!

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

Destructors

- **What is destructor?**
  Destructor is a member function which destructs or deletes an object.
- **When is destructor called?**
  A destructor function is called automatically when the object goes out of scope:
  - ✓ the function ends
  - ✓ the program ends
  - ✓ a block containing local variables ends
  - ✓ a delete operator is called

Destructors

- **How destructors are different from a normal member function?**
  Destructors have same name as the class preceded by a tilde (~). Destructors don't take any argument and don't return anything

```
1  #include <iostream>
2  using namespace std;
3  class Prime{
4      public:
5        ///Constructor
6        Prime(){
7          cout<<"Constructor is called!!"<<endl;
8        }
9        ///Destructor
10       ~Prime(){
11         cout<<"Destructor is called!!"<<endl;
12       }
13       ///Member function
14       void display(){
15         cout<<"Prime University!!"<<endl;
16       }
17 };
18 int main(){
19     ///Object created
20     Prime obj;
21     ///Member function called
22     obj.display();
23
24     return 0;
25 }
```

Thank You