Object oriented Programming with C++

Constructors and Destructors

By

Nilesh Dalvi

Lecturer, Patkar-Varde College.

Constructor

- A constructor is a 'special' member function whose task is to initialize the objects of its class.
- It is special because its name is same as the class.
- It is called constructor because it constructs the value of data member.
- Constructor is invoked whenever an object of its associated class is created.

Constructor

```
class add
{
      int m, n ;
   public :
      add (void) ;
      ------
};
add :: add (void)
{
   m = 0; n = 0;
}
```

- add a ;
- Creates the object 'a' of types add and initializes its data members m and n to zero.
- There is no need to write any statement to invoke the constructor function.
- A constructor that accepts no parameter is called as a default constructor .

Constructor

Characteristics of Constructor:

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and they cannot return values.
- They cannot be inherited, though a derived class can call the base class constructor.
- Like other C++ functions, Constructors can have default arguments.
- Constructors can not be virtual.

Types of Constructor:

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor
4. Dynamic Constructor

Default Constructor

```
#include<iostream>
using namespace std;

class Cube
{
public:
    int side;
    Cube()
    {
        side=10;
    }
};

int main()
{
    Cube c;
    cout << c.side;
    return 0;
}
```

- Default constructor is the constructor which doesn't take any argument.
- It has no parameter.
- It is also called as zero-argument constructor.

Parameterized Constructor

- It is also possible to create constructor with arguments and such constructors are called as parameterized constructors or constructor with arguments.
- For such constructors, it is necessary to pass values to the constructor when object is created.

Parameterized Constructor

```
class area
{
    int length,breadth;
  public:
    area(int l,int b)//parameterized constructor.
    {
        length = l;
        breadth = b;
    }
    void display()
    {
        cout << "Length of rectangle is:" << length << endl;
        cout << "Breadth of rectangle is: " << breadth << endl;
        cout << "Area of rectangle is: " << length*breadth;
    }
};
```

Parameterized Constructor

- When a constructor has been parameterized,
- **area a**; // may not work.
- We must pass initial value as arguments to the constructor function when an object is declared. This can be done by two ways:
  - By calling the constructor explicitly.
  - By calling the constructor implicitly.
- The following declaration illustrates above method:

```
area a = area (5, 6);    // explicit call
area a (5, 6);           // implicit call
```

Parameterized Constructor

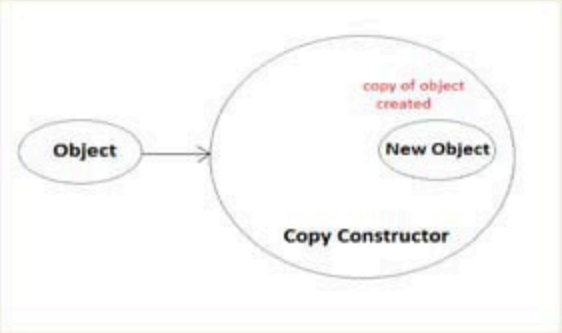
```
int main()
{
    area a(8,7);
    a.display();

    area c(10,5);
    c.display();

    return 0;
}
```

Constructors with default arguments:

```
#include<iostream>
#include<cmath>
using namespace std;

class power
{
    int num;
    int power;
    int ans;
  public :
    power (int n = 9, int p = 3); // declaration of constructor
                                  //with default arguments

    void show ()
    {
       cout <<"\n"<<num <<" raise to "<<power <<" is " <<ans;
    }
};

power :: power (int n,int p )
{
    num = n;
    power = p;
    ans = pow (n, p);
}

int main( )
{
    power p1, p2(5);
    p1.show ();
    p2.show ();
    return  0;
}
```

Copy Constructor

- The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously.
- The copy constructor is used to initialize one object from another of same type.



Copy Constructor

- Syntax:

```
class  abc
{
      public:
            abc (abc &);
};
```

Copy Constructor

- Referencing operator (&) is used to define referencing variable.
- Ref. variable prepares an alternative (alias) name for previously defined variable.
- For Example:

```
int  qty = 10;      // declared  and initialized.
int  & qt = qty;  // alias variable.
```

- Any change made in one of the variable causes change in both the variable.

```
qt = qt*2;       // contents of qt and qty will be 20.
```

Copy Constructor

```
// Class definition
class TimeSlt
{
    int Da, Mo, Ye;
  public:
    void Disply();
    TimeSlt(){}
    TimeSlt(int D1,int M1,int Y1); // Constructor with parameters
    TimeSlt(TimeSlt &); // Copy constructor
};
void TimeSlt::Display()
{
    cout<<Da<<" "<<Mo<<" "<<Ye<<endl;
}
TimeSlt::TimeSlt(int D1,int M1,int Y1)
{
    Da = D1;
    Mo = M1;
    Ye = Y1;
}
TimeSlt::TimeSlt(TimeSlt &tmp)
{
    cout<<"Data copied"<<endl;
    Da = tmp.Da;
    Mo = tmp.Mo;
    Ye = tmp.Ye;
}
```

```
int main()
{
    TimeSlt T1(13,8,1990);
    T1.Display();
    TimeSlt T2(T1);
    T2.Dispaly();
    TimeSlt T3 = T1;
    T3.Dispaly();
    TimeSlt T4;
    T4 = T1;
    T4.Dispaly();
    return 0;
}
```

Copy Constructor

```
TimeSlt  T2 (T1);
```

- Object T2 is created with one object(T1), the copy constructor is invoked and data members are initialized.

```
TimeSlt T3 = T1;
```

- Object T3 is created with assignment with object T1; copy constructor invoked, compilers copies all the members of object T1 to destination object T3 .

```
TimeSlt T4;
T4 = T1;
```

- Copy constructor is not executed, member-to-member of T1 are copied to object T4 , assignment statement assigns the values.

Overloading Constructor

- Like functions, it is also possible to overload constructors.
- In previous examples, we declared single constructors without arguments and with all arguments.
- A class can contain more than one constructor. This is known as constructor overloading.
- All constructors are defined with the same name as the class.
- All the constructors contain different number of arguments.
- Depending upon number of arguments, the compiler executes appropriate constructor.

Overloading Constructor

```
#include<iostream>
using namespace std;

class perimeter
{
    int l, b, p;
   public:
    perimeter ()
    {
        cout << "\n Enter the value of l and b";
        cin >> l >> b;
    }
    perimeter (int a)
    {
        l = b = a;
    }
    perimeter (int l1, int b1)
    {
        l = l1;
        b = b1;
    }
    perimeter (perimeter &peri)
    {
        l = peri.l;
        b = peri.b;
    }
    void calculate (void);
};
```

Overloading Constructor

```
void perimeter :: calculate (void)
{
    p = 2* (l + b)
    cout << p;
}
int main ()
{

    perimeter obj, obj1 (2), obj2 (2, 3);

    cout<<"\n perimeter of rectangle is ";
    obj. Calculate ();

    cout<<"\n perimeter of square is ";
    obj1.calculate ();

    cout<<"\n perimeter of rectangle is ";
    obj2.calculate ();

    cout<<"\n perimeter of rectangle is ";
    perimeter obj3 (obj2);
    obj3.calculate ();

    return 0;
}
```

Dynamic Constructor

- The constructors can also be used to allocate memory while creating objects.
- This will enable the system to allocate the right amount of memory for each object when the objects are not of the same size.
- Allocation of memory to objects at the time of their construction is known as dynamic construction of objects.
- The memory is created with the help of the new operator.

Dynamic Constructor

```
#include <iostream>
#include <string>
using namespace std;

class str
{
        char *name;
        int len;
    public:
        str()
        {
            len = 0;
            name = new char[len + 1];
        }
        str(char *s)
        {
            len = strlen(s);
            name = new char[len + 1];
            strcpy(name,s);
        }
        void show()
        {
            cout<<"NAME IS:->"<<name<<endl;
        }
        void join(str a, str b);
};
```

Dynamic Constructor

```
void str :: join(str a, str b)
{
    len = a.len + b.len;
    name = new char[len + 1];
    strcpy(name, a.name);
    strcat(name, b.name);
}

int main()
{
    char *first="OOPS";
    str n1(first),n2("WITH"),n3("C++"),n4,n5;

    n4.join(n1,n2);
    n5.join(n4,n3);
    n1.show();
    n2.show();
    n3.show();
    n4.show();
    n5.show();
    return 0;
}
```

Destructor

- Destructor is also a 'special' member function like constructor.
- Destructors destroy the class objects created by constructors.
- The destructors have the same name as their class, preceded by a tilde (~).
- It is not possible to define more than one destructor.
- For example : ~Circle();

Destructor

- The destructor is only one way to destroy the object.
- They cannot be overloaded.
- A destructor neither requires any argument nor returns any value.
- It is automatically called when object goes out of scope.
- Destructor releases memory space occupied by the objects.

Destructor

```
#include<iostream>
using namespace std;

class Circle //specify a class
{
    private :
        double radius; //class data members
    public:
        Circle() //default constructor
        {
            radius = 0;
        }
        Circle(double r) //parameterized constructor
        {
            radius = r;
        }
        Circle(Circle &t) //copy constructor
        {
            radius = t.radius;
        }
        void setRadius(double r); //function to set data

        double getArea();

        ~Circle() //destructor
        {}
};
```

Destructor

```
void Circle :: setRadius(double r) //function to set data
{
      radius = r;
}
double Circle :: getArea()
{
    return 3.14 * radius * radius;
}
int main()
{
    Circle c1; //defalut constructor invoked
    Circle c2(2.5); //parmeterized constructor invoked
    Circle c3(c2); //copy constructor invoked
    cout << c1.getArea()<<endl;
    cout << c2.getArea()<<endl;
    cout << c3.getArea()<<endl;
    c1.setRadius (1.2);
    cout << c1.getArea()<<endl;
    return 0;
}
```

Const Member function

- The member functions of class can be declared as constant using `const` keyword.
- The constant function cannot modify any data in the class.
- The `const` keyword is suffixed to the function declaration as well as in function definition.
- If these functions attempt to change the data, compiler will generate an error message.

Const Member function

```
#include <iostream>
using namespace std;

class ABC
{
      int c;
public:
      void show(int a, int b) const
      {
            c = a + b;
            cout << " a + b ::" <<c;
      }
};

int main()
{
      const ABC x;
      x.show (5, 7);
      return 0;
}
```

Const Member function

```
#include <iostream>
using namespace std;

class ABC
{
      int c;
public:
      void show(int a, int b) const
      {
            //c =  a + b;
            cout << " a + b ::" <<a + b;
      }
};

int main()
{
      const ABC x;
      x.show (5, 7);
      return 0;
}
```

Const Objects

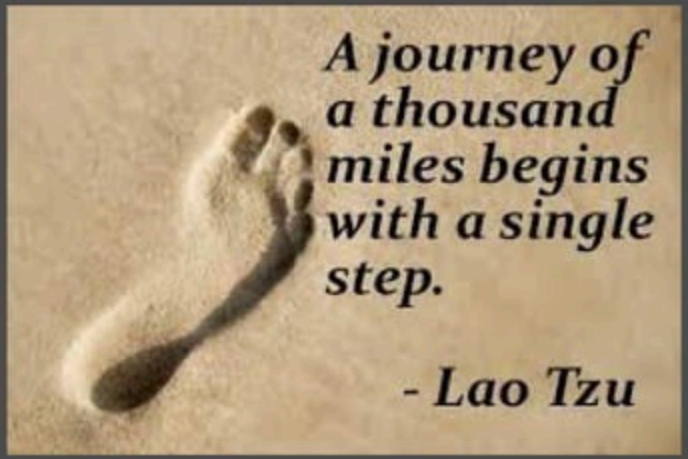
- Like constant member function, we can make the object constant by the keyword const.
- Only constructor can initialize data member variables of constant objects.
- The data member of constant objects can only be read and any effect to alter values of variables will generate an error.
- The data member of constant objects are also called as a read-only data member.
- The const-objects can access only const-functions.
- If constant objects tries to invoke a non-member function, an error message will be displayed.

Const Objects

```
#include <iostream>
using namespace std;

class ABC
{
	int a;
public:
	ABC(int m)
	{
		a = m;
	}
	void show() const
	{
		cout << "a = " << a <<endl;
	}
};
```

Const Objects

```
int main()
{
        const ABC x(5);
        x.show ();
        return 0;
}
```



To be continued.....