

OOPs Concepts in C++

Object oriented programming is a way of solving complex problems by breaking them into smaller problems using objects. Before Object Oriented Programming (commonly referred as OOP), programs were written in procedural language, they were nothing but a long list of instructions. On the other hand, the OOP is all about creating objects that can interact with each other, this makes it easier to develop programs in OOP as we can understand the relationship between them.

Object Oriented Programming (OOP)

In Object oriented programming we write programs using classes and objects utilizing features of OOPs such as abstraction, encapsulation, inheritance and polymorphism.

Class and Objects

A class is like a blueprint of data member and functions and object is an instance of class. For example, lets say we have a class **Car** which has data members (variables) such as speed, weight, price and functions such as gearChange(), slowDown(), brake() etc. Now lets say I create a object of this class named FordFigo which uses these data members and functions and give them its own values. Similarly we can create as many objects as we want using the blueprint(class).

```
//Class name is Car
class Car
{
    //Data members
    char name[20];
    int speed;
    int weight;

public:
    //Functions
    void brake(){
    }
    void slowDown(){
    }
};

int main()
{
    //ford is an object
    Car ford;
}
```

Abstraction

Abstraction is a process of hiding irrelevant details from user. For example, When you send an sms you just type the message, select the contact and click send, the phone shows you that the message has been sent, what actually happens in background when you click send is hidden from you as it is not relevant to you.

Encapsulation

Encapsulation is a process of combining data and function into a single unit like capsule. This is to avoid the access of private data members from outside the class. To achieve encapsulation, we make all data members of class private and create public functions, using them we can get the values from these data members or set the value to these data members.

Inheritance

Inheritance is a feature using which an object of child class acquires the properties of parent class.

```
#include <iostream>
using namespace std;
class ParentClass {
    //data member
public:
    int var1 =100;
};
class ChildClass: public ParentClass {
public:
    int var2 = 500;
};
int main(void) {
    ChildClass obj;
}
```

Now this object obj can use the properties (such as variable var1) of ParentClass.

Polymorphism

Function overloading and Operator overloading are examples of polymorphism. Polymorphism is a feature using which an object behaves differently in different situation. In function overloading we can have more than one function with same name but different numbers, type or sequence of arguments.

Polymorphism Example

```
#include <iostream>
using namespace std;
class Sum {
public:
    int add(int num1,int num2){
        return num1 + num2;
    }
    int add(int num1, int num2, int num3){
        return num1 + num2 + num3;
    }
};
int main(void) {
    //Object of class Sum
    Sum obj;

    //This will call the second add function
}
```

```
cout<<obj.add(10, 20, 30)<<endl;  
  
//This will call the first add function  
cout<<obj.add(11, 22);  
return 0;  
}
```

Output:

```
60  
33
```

Constructors in C++

Constructor is a special member function of a class that initializes the object of the class. Constructor name is same as class name and it doesn't have a return type. Lets take a simple example to understand the working of constructor.

Simple Example: How to use constructor in C++

Read the comments in the following program to understand each part of the program.

```
#include <iostream>
using namespace std;
class constructorDemo{
public:
    int num;
    char ch;
    /* This is a default constructor of the
     * class, do note that it's name is same as
     * class name and it doesn't have return type.
     */
    constructorDemo() {
        num = 100; ch = 'A';
    }
};
int main(){
    /* This is how we create the object of class,
     * I have given the object name as obj, you can
     * give any name, just remember the syntax:
     * class_name object_name;
     */
    constructorDemo obj;

    /* This is how we access data members using object
     * we are just checking that the value we have
     * initialized in constructor are reflecting or not.
     */
    cout<<"num: "<<obj.num<<endl;
    cout<<"ch: "<<obj.ch;
    return 0;
}
```

Output:

```
num: 100
ch: A
```

Constructor vs Member function

Now that we know what is constructor, lets discuss how a constructor is different from member function of the class.

1) Constructor doesn't have a return type. Member function has a return type.

2) Constructor is automatically called when we create the object of the class. Member function needs to be called explicitly using object of class.

3) When we do not create any constructor in our class, C++ compiler generates a default constructor and insert it into our code. The same does not apply to member functions. This is how a compiler generated default constructor looks:

```
class XYZ
{
    ....
    XYZ()
    {
        //Empty no code
    }
};
```

Types of Constructor in C++

There are two types of constructor in C++. 1) Default constructor 2) Parameterized constructor

1) Default Constructor

A default constructor doesn't have any arguments (or parameters)

```
#include <iostream>
using namespace std;
class Website{
public:
    //Default constructor
    Website() {
        cout<<"Welcome to BeginnersBook"<<endl;
    }
};
int main(void){
    /*creating two objects of class Website.
    * This means that the default constructor
    * should have been invoked twice.
    */
    Website obj1;
    Website obj2;
    return 0;
}
```

Output:

```
Welcome to BeginnersBook
Welcome to BeginnersBook
```

When you don't specify any constructor in the class, a default constructor with no code (empty body) would be inserted into your code by compiler.

2) Parameterized Constructor

Constructors with parameters are known as Parameterized constructors. These type of constructor allows us to pass arguments while object creation. Lets see how they look:

Lets say class name is XYZ

Default constructor:

```
XYZ() {  
  
}  
....  
XYZ obj;  
....
```

Parameterized Constructor:

```
XYZ(int a, int b) {  
  
}  
...  
XYZ obj(10, 20);
```

Example:

```
#include <iostream>  
using namespace std;  
class Add{  
public:  
    //Parameterized constructor  
    Add(int num1, int num2) {  
        cout<<(num1+num2)<<endl;  
    }  
};  
int main(void){  
    /* One way of creating object. Also  
    * known as implicit call to the  
    * constructor  
    */  
    Add obj1(10, 20);  
    /* Another way of creating object. This  
    * is known as explicit calling the  
    * constructor.  
    */  
    Add obj2 = Add(50, 60);  
    return 0;  
}
```

Output:

```
30  
110
```

Destructors in C++

A destructor is a special member function that works just opposite to constructor, unlike constructors that are used for initializing an object, destructors destroy (or delete) the object.

Syntax of Destructor

```
~class_name()  
{  
    //Some code  
}
```

Similar to constructor, the destructor name should exactly match with the class name. A destructor declaration should always begin with the tilde(~) symbol as shown in the syntax above.

When does the destructor get called?

A destructor is **automatically called** when:

- 1) The program finished execution.
- 2) When a scope (the { } parenthesis) containing local variable ends.
- 3) When you call the delete operator.

Destructor Example

```
#include <iostream>  
using namespace std;  
class HelloWorld{  
public:  
    //Constructor  
    HelloWorld(){  
        cout<<"Constructor is called"<<endl;  
    }  
    //Destructor  
    ~HelloWorld(){  
        cout<<"Destructor is called"<<endl;  
    }  
    //Member function  
    void display(){  
        cout<<"Hello World!"<<endl;  
    }  
};  
int main(){  
    //Object created  
    HelloWorld obj;  
    //Member function called  
    obj.display();  
    return 0;  
}
```

Output:

```
Constructor is called  
Hello World!  
Destructor is called
```

Destructor rules

- 1) Name should begin with tilde sign(~) and must match class name.
- 2) There cannot be more than one destructor in a class.
- 3) Unlike constructors that can have parameters, destructors do not allow any parameter.
- 4) They do not have any return type, just like constructors.
- 5) When you do not specify any destructor in a class, compiler generates a default destructor and inserts it into your code.