Static Member Data
&
Static Member Function

Static Data

We use the static keyword to define the static data member or static member function inside and outside of the class.

The syntax of the static data members is given as follows –

static data_type data_member_name;

```
#include <iostream>
using namespace std;
class Stu {
   private:
   int rollNo;
  int marks;
   public:
   static int b;
   Stu() {
      b++;
   }
   void getdata() {
      cout << "Enter roll number:
"<<endl;
      cin >> rollNo;
     cout << "Enter marks: "<<endl;
      cin >> marks;
   }
   void putdata() {
      cout<<"Roll Number = "<< rollNo
<<endl;
      cout<<"Marks = "<< marks <<endl;

   }
};
int Stu::b = 0;
int main(void) {
   Stu s1;
   s1.getdata();
   s1.putdata();
   Student s2;
   s2.getdata();
   s2.putdata();
   Student s3;
   s3.getdata();
   s3.putdata();
   cout << "Total
objects created = " <<
Student::b << endl;
   return 0;

}
```

Static Member Function

- A static member function can access only the names of static members, enumerators, and nested types of the class in which it is declared.

- A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created, if no other initialization is present.

```
#include <iostream>
 using namespace std;int Box::objectCount = 0;
int main(void) {
   Box Box1(3.3, 1.2, 1.5);    // Declare box1
   Box Box2(8.5, 6.0, 2.0);    // Declare box2
cout << "Total objects: " << Box::objectCount << endl;
   return 0;
class Box {
   public:
      static int objectCount;
        // Constructor definition
      Box(double l = 2.0, double b = 2.0, double h = 2.0) {
         cout <<"Constructor called." << endl;
         length = l;
         breadth = b;
         height = h;
           // Increase every time object is created
         objectCount++;
      }
      double Volume() {
         return length * breadth * height;
      }
         private:
      double length;     // Length of a box
      double breadth;    // Breadth of a box
      double height;     // Height of a box
};}
```

```
int Box::objectCount = 0;


int main(void) {
   Box Box1(3.3, 1.2, 1.5);    // Declare box1
   Box Box2(8.5, 6.0, 2.0);    // Declare box2


   // Print total number of objects.
   cout << "Total objects: " << Box::objectCount << endl;


   return 0;
```

Friend Function

Friend Class A friend class can access private and protected members of other class in which it is declared as friend.

Friend Function Like friend class, a friend function can be given a special grant to access private and protected members. A friend function can be:

a) A member of another class

b) A global function

Friend function

Following are some important points about friend functions and classes:
1) Friends should be used only for limited purpose. too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.
2) Friendship is not mutual. If class A is a friend of B, then B doesn’t become a friend of A automatically.
3) Friendship is not inherited (See this for more details)
4) The concept of friends is not there in Java.
A simple and complete C++ program to demonstrate friend Class

Constant Member Function

The const member functions are the functions which are declared as constant in the program. The object called by these functions cannot be modified.

Here is the syntax of const member function in C++ language,

datatype function_name const();

```
class Demo {
   int val;
   public:
   Demo(int x = 0) {      val = x;   }
   int getValue() const {      return val;   }};
int main() {
   const Demo d(28);
   Demo d1(8);
```

This Pointer

- Each object gets its own copy of the data member.
- All access the same function definition as present in the code segment. Meaning each object gets its own copy of data members and all objects share single copy of member functions.
- The ‘this’ pointer is passed as a hidden argument to all nonstatic member function calls and is available as a local variable within the body of all nonstatic functions.
- ‘this’ pointer is a constant pointer that holds the memory address of the current object.
- ‘this’ pointer is not available in static member functions as static member functions can be called without any object (with class name).

Characteristics of this pointer

- built-in pointer
- Points to objects or contains the address of the object
- Data member can be accessed
-

Nested Class

A nested class is a class that is declared in another class. The nested class is also a member variable of the enclosing class and has the same access rights as the other members.

```
#include<iostream>
using namespace std;
class A {
   public:
   class B {
      private:
      int num;
      public:
      void getdata(int n) {
         num = n;
      }
      void putdata() {
         cout<<"The number is "<<num;
      }
   };};
```