## Class static data members

- We can define class members static using **static** keyword. When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.
- 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.
- We can't put it in the class definition but it can be initialized outside the class.

## Example

```
#include <iostream>
Using namespace std;
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
};
// Initialize static member of class
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;
}
```

## Static Function Members

•A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator ::.

•A static member function can only access static data member, other static member functions and any other functions from outside the class.

•Static member functions have a class scope and they do not have access to the this pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

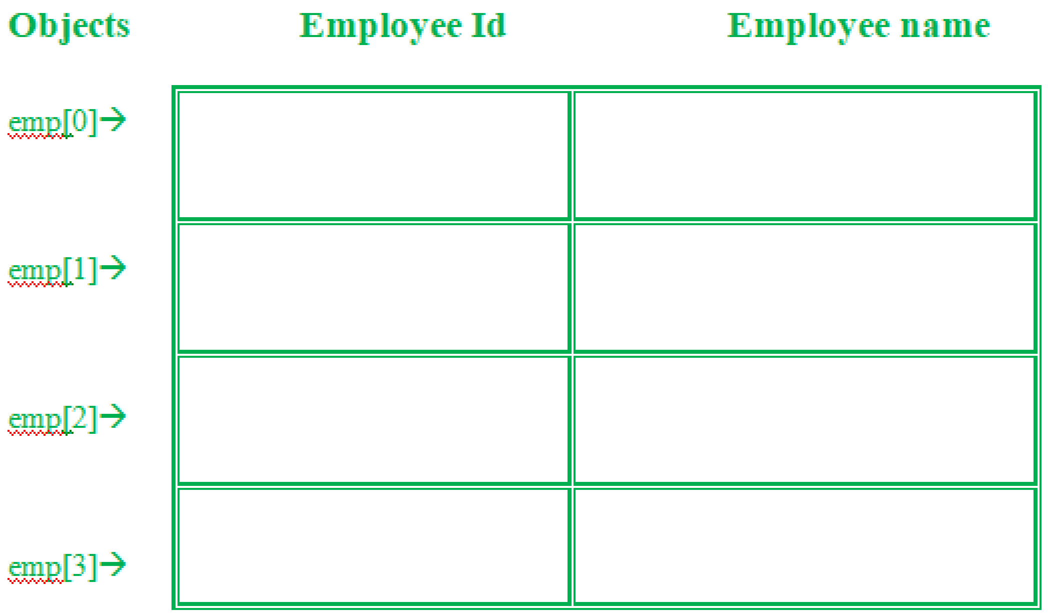
## Example

```
class Box {
public:
static int objectCount;
Box(double l = 2.0, double b = 2.0, double h = 2.0) {
cout <<"Constructor called." << endl;
 length = l; breadth = b; height = h;
objectCount++;
}
double Volume() {
 return length * breadth * height;
}
static int getCount(){
return objectCount;}
 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) {
cout << "Inital Stage Count: " << Box::getCount() << endl;
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 after creating object.
cout << "Final Stage Count: " << Box::getCount() << endl;
 return 0; }
```

## Arrays of Objects

- When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, we need to create objects.
- The Array of Objects stores *objects*. An array of a class type is also known as an array of objects.
- Syntax :

ClassName ObjectName[number of objects];



## Example

```
class Employee
{
  int id;
  char name[30];
  public:
  void getdata();        //Declaration of function
  void putdata();        //Declaration of function
};
void Employee::getdata(){      //Defining of function
  cout<<"Enter Id : ";
  cin>>id;
  cout<<"Enter Name : ";
  cin>>name;
}
void Employee::putdata(){/        /Defining of function
  cout<<id<<" ";
  cout<<name<<" ";
  cout<<endl;
}
int main(){
  Employee emp[30];
  int n, i;
  cout << "Enter Number of Employees - ";
  cin >> n;
  for(i = 0; i < n; i++)
   emp[i].getdata();
   cout << "Employee Data - " << endl;
   for(i = 0; i < n; i++)
   emp[i].putdata();
  return 0;
 }
```

**C++ String Class**

- C++ string class internally uses char array to store character but all memory management, allocation, and null termination is handled by string class itself that is why it is easy to use.
- The length of the C++ string can be changed at runtime because of dynamic allocation of memory similar to vectors.
- As string class is a container class, we can iterate over all its characters using an iterator similar to other containers like vector, set and maps, but generally, we use a simple for loop for iterating over the characters and index them using the [] operator.
  C++ string class has a lot of functions to handle string easily. Most useful of them are demonstrated in below code.

## Example

```
int main()
{
   // various constructor of string class

   // initialization by raw string
   string str1("first string");

   // initialization by another string
   string str2(str1);

   // initialization by character with number of occurrence
   string str3(5, '#');

   // initialization by part of another string
   string str4(str1, 6, 6); //    from 6th index (second parameter)
                            // 6 characters (third parameter)

   // initialization by part of another string : iterator version
   string str5(str2.begin(), str2.begin() + 5);
   cout << str1 << endl;
   cout << str2 << endl;
   cout << str3 << endl;
   cout << str4 << endl;
   cout << str5 << endl;
   //  assignment operator
   string str6 = str4;
// clear function deletes all character from string
   str4.clear();
   //  both size() and length() return length of string and
```

# Continue

```
     //  they work as synonyms
    int len = str6.length(); // Same as "len = str6.size();"
    cout << "Length of string is : " << len << endl;
    // a particular character can be accessed using at /
    // [] operator
    char ch = str6.at(2); //  Same as "ch = str6[2];"
    cout << "third character of string is : " << ch << endl;
    //  front return first character and back returns last character
    //  of string
    char ch_f = str6.front();  // Same as "ch_f = str6[0];"
    char ch_b = str6.back();   // Same as below
                          // "ch_b = str6[str6.length() - 1];"

    cout << "First char is : " << ch_f << ", Last char is : "
         << ch_b << endl;

    // c_str returns null terminated char array version of string
    const char* charstr = str6.c_str();
    printf("%s\n", charstr);
    // append add the argument string at the end
    str6.append(" extension");
    //  same as str6 += " extension"
   // another version of append, which appends part of other
  // string
    str4.append(str6, 0, 6);  // at 0th position 6 character

    cout << str6 << endl;
    cout << str4 << endl;
```

## Continue

```
if (str6.find(str4) != string::npos)
        cout << "str4 found in str6 at " << str6.find(str4)
             << " pos" << endl;
    else
        cout << "str4 not found in str6" << endl;
    //  substr(a, b) function returns a substring of b length
    //  starting from index a
    cout << str6.substr(7, 3) << endl;

    //  if second argument is not passed, string till end is
    // taken as substring
    cout << str6.substr(7) << endl;
    //  erase(a, b) deletes b characters at index a
    str6.erase(7, 4);
    cout << str6 << endl;
    //  iterator version of erase
    str6.erase(str6.begin() + 5, str6.end() - 3);
    cout << str6 << endl;

    str6 = "This is a examples";

    //  replace(a, b, str)  replaces b characters from a index by str
    str6.replace(2, 7, "ese are test");

    cout << str6 << endl;

    return 0;
}
```

# Output

```
first string
 first string
 #####
string first Length of string is : 6
 third character of string is : r
 First char is : s, Last char is : g
string
string extension
string
str4 found in str6 at 0 pos
Ext
 extension
string nsion strinion These are test examples
```