Static Data Members

A data member of a class can be qualified as static. The properties of a static member variable are similar to that of a C static variable. A static member variable has certain special characteristics. These are:

1. It is initialized to zero when the first object of its class is created. No other initialization is permitted.
2. Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
3. It is visible only within the class, but its lifetime is the entire program.

Static variables arc normally used to maintain values common to the entire class.

```
#include <iostream.h>
Class item
{
        static int count;
        int number;
Public:
        void getdata(int a, int b)
        {
                number a;
                count ++;
        }
        void getcount (void)
        {
                Cout<<”count is”<<count;
        }
};
int item:: count;

int main()
{
        item a, b, c;
        a. getcount();
        b. getcount()
        c. getcount();

        a. getdata(100);
        b. gedata(200);
        c. getdata(300);

        cout<<"After reading data”<<"\n";
        a.getcount();  // display count
        b.geteount();
        c.getcount();
        return 0;
}
```

The output of the Program would be:

```
count 0
count 0
count 0
After reading data
count: 3
count: 3
```

the type and scope of each static member variable must be defined outside the class definition. This is necessary because the static data members are stored separately rather than as a part, of an object. Since they are associated with the class itself rather than with any class objects they are also known as class variables.

Static variables are like non-inline member functions as they are declared in a class declaration and defined in the source file. While defining a static variable, some initial value can also be assigned to the variable, the initial value 10 is assigned as.

```
int Item::count = 10;
```

Static member function

Like static member variable, we can also have static member functions. A member function that is declared static has the following properties:

1. A static function can have access to only ether static members (functions or variables) declared in the same class,
2. A static member function can be called using the clase name (Instead of its objects) us follows:

```
class_name :: function-name;
```

The static function showcount() displays the number of objects created till that moment. A count of number of objects created is maintained by the static variable count, The function showcode() displays the code number of each object.

```
#include<iostream.h>
class test
{
        int code;
        Static int count;
Public:
        Void setcode()
        {
                Code=++count;
        }
        Void showcode()
        {
                Cout<<"object no"<<code;
        }
        Static void showcount()
        {
                Cout<<"count is"<<count;
        }
};
Int main()
{
        Test t1, t2;
        t1.setcode();
        t2.setcode();
        test::showcount;
        test t3;
        t3.setcode();
        test::showcount;
        tl .showcode();
        t2.showcode () ;
        t3.showcode() ;
return (0);
}
```

Output:

```
count is 2
count is
object no: 1
object no: 2
object no; 3
```

Objects as Function Arguments

Like any other data type, an object may be used as a function argument. This can be done in following ways:

1. A copy of the entire object is passed to the function (call by value).
2. Only the address of the object is transferred to the function (call by reference).
3. Call by pointer.

The first method is called pass-by-value Since of copy of the object in passed to the function, any changes made to the object inside the function do not affect the object used to call the function- The second method is called pass by reference. When an address of the object is passed, the called function works directly on the actual object used in the call. This means that any changes made to the object inside the function will reflect in the actual object. The pass-by reference method is more efficient since it requires passing only the address of the object and not the entire object.

```
#include<iostream.h>

#include<conio.h>

class time
{
        int hours;
        int minutes;
public:
        void gettime(int h, int m)
{
        hours=h;
        minutes=m;
}
void display()
{
        cout<<”hours:”<<hours;
        cout<<”minutes:”<<minutes;
}
void sum(time,time);
};  // end of class time

void time::sum(time t1, time t2)
{
        minutes=t1.minutes+ t2.minutes;
        hours= minutes/60;
        minutes= minutes%60;
        hours=hours+t1.hours+t2.hours;
}


void main()
{
        time t1, t2, t3;
        t1.gettime(2,60);
        t2.gettime(3,0);
        t3.sum(t1,t2);
        cout<<”t1=”; t1.display();
        cout<<”t2=”; t2.display();
        cout<<”t3=”; t3.display();
        return 0;
}
```
Output:
```
hours:2 minutes:60
hours:3 minutes:0
hours:5 minutes:60
```

Example(call by value, reference and pointer)
```
#include<iostream.h>
#include<conio.h>
class A
{
public:
        int a;
public:
        void set(A a, int x)//call by value
        {
                cout<<"--call by value--";
                a.a=x;
                cout<<"\n"<<"value is :"<<a.a<<"\n";
        }
        void set(int x, A &a)//call by Reference
        {
                cout<<"--call by reference--";
                a.a=x;
                cout<<"\n"<<"value is :"<<a.a<<"\n";
        }
```

```
          void set(A *a, int x)//call by pointer
          {
                    cout<<"--call by pointer--";
                    a->a=x;
                    cout<<"\n"<<"value is :"<<a->a<<"\n";
          }
};
void main()
{
          clrscr();
          A a;
          a.set(a,5);
          cout<<"--value after call by value--"<<a.a<<"\n";
          a.set(10,a);
          cout<<"--value after call by referecne--"<<a.a<<"\n";
          a.set(&a,12);
          cout<<"--value after call by poniter--"<<a.a;
getch();
}
```

a.set(&a,12)
invokes the appropriate function. This time the address of object is passed to the function definition, which is stored in another pointer variable in that definition. In the example, the address of object is stored in pointer variable ‘a’ which is of type ‘class A’. The change in data member through this pointer will reflected in the original copy of object ‘a’

Function Returning Objects
A function cannot only receive objects as arguments but also can return them. The example illustrates how an object can be created within a function j and returned to another function.
Example:

```
#include<iostream.h>
#include<conio.h>
class A
{
          int a;
          int b;
public:
          void set(int x, int y)
          {
                    a=x;
                    b=y;
          }
          A sum(A a1,A a2)
          {
                    A a3;
                    a3.a=a1.a+a2.a;
                    a3.b=a1.b+a2.b;
                    return a3;
          }
          void display()
          {
                    cout<<a<<"\n"<<b<<"\n";
          }
};
void main()
{
          clrscr();
          A a,b,c;
          a.set(10,20);
          cout<<"value of object a"<<"\n";
          a.display();
          b.set(30,40);
          cout<<"value of object b"<<"\n";
          b.display();
          cout<<"total Sum:";
          c=c.sum(a,b);
          c.display();
getch();
}
```