LECTURE-8

Static Data Member

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

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- 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.
- It is visible only with in the class but its life time is the entire program. Static variables are normally used to maintain values common to the entire class. For example, a static data member can be used as a counter that records the occurrence of all the objects.

int item:: count; // definition of static data member

Note that 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. The following are some key points about static data members in OOP:

- Shared Across Instances: Unlike regular member variables, which have separate copies for each instance of the class, a static data member is shared by all instances of that class. Changes made to the static member affect all objects of the class.
- Declared with static: To declare a static data member in a class, the static keyword is used before the member's declaration. This keyword signifies that the variable belongs to the class itself rather than to any specific object.

- Initialization: Static data members must be defined outside the class, typically in the global scope or within a source file, to allocate memory for them. However, they can also be initialized inside the class, but they still need to be defined outside the class to allocate memory space.
- Accessing Static Members: Static data members can be accessed without creating an object of the class. They are accessed using the class name followed by the scope resolution operator ::.

```
#include<iostream>
using namespace std;
class item
{
	static int count; //count is static
	int number;
	public:
	void getdata(int a)
	{
		number=a;
		count++;
	}
	void getcount()
	{
		cout<<"count:";
		cout<<count<<endl;
	}
};
int item::count; //count defined
main()
{
	item a,b,c;
	a.getcount();
	b.getcount();
	c.getcount();
	a.getdata(2);
	b.getdata(3);
	c.getdata(4);
	cout<<"after reading data : "<<endl;
	a.getcount();
	b.getcount();
	c.getcount();
}
```

The static Variable count is initialized to Zero when the objects created. The count is incremented whenever the data is read into an object. Since the data is read into objects three times the variable count is incremented three times. Because there is only one copy of count shared by all the three objects, all the three output statements cause the value 3 to be displayed.

Static Member Function

A member function that is declared static has following properties:

- A static function can have access to only other static members declared in the same class.
- A static member function can be called using the class name as follows: class - name:: function - name;

```
#include<iostream>
using namespace std;
class test
{
       int code;
       static int count; // static member variable
       public:
       void set(void)
              { code=++count;}
       void showcode(void)
              { cout<<"object member : "<<code<<endl;}
       static void showcount(void)
              { cout<<"count="<<count<<endl; }
};
int test:: count;
main()
{
       test t1,t2,t3;
       t1.set();      t2.set();
       test::showcount ( );
       t3.set();
       test:: showcount( );
       t1.showcode();
       t2.showcode();
       t3.showcode();
}
```

Objects As Function Arguments

Like any other data type, an object may be used as A function argument. This can come in two ways

- A copy of the entire object is passed to the function.
- Only the address of the object is transferred to the function

The **first** method is called pass-by-value. Since a copy of the object is passed to the function, any change 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 functions will reflect in the actual object. The pass by reference method is more efficient since it requires to pass only the address of the object and not the entire object.

function_name(object_name);

```
#include <bits/stdc++.h>
using namespace std;
class Example {
public:
	int a;
	void add(Example E)
	{
		a = a + E.a;
	}
};
main()
{
	Example E1, E2;
	E1.a = 50;
	E2.a = 100;
	cout << "Initial Values \n";
	cout << "Value of object 1: " << E1.a
		<< "\n& object 2: " << E2.a
		<< "\n\n";
	E2.add(E1);
	Cout<<E.a<<E.a;}
```

Example: In this Example there is a class which has an integer variable 'a' and a function 'add' which takes an object as argument. The function is called by one object and takes another as an argument. Inside the function, the integer value of the argument object is added to that on which the 'add' function is called. In this method, we can pass objects as an argument and alter them.

```
#include<iostream>
using namespace std;
class time
{
	int hours;
	int minutes;
	public:
	void gettime(int h, int m)
	{
		hours=h;
		minutes=m;
	}
	void puttime()
	{
		cout<< hours<<"hours and:";
		cout<<minutes<<"minutes:"<<endl;
	}
	void sum(time,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;
}
main()
{
	time T1,T2,T3;
	T1.gettime(2,45);
	T2.gettime(3,30);
	T3.sum(T1,T2);
	cout<<"T1=";
	T1.puttime( );	cout<<"T2=";
	T2.puttime( );	cout<<"T3=";
	T3.puttime( );
}
```

Example: A program to demonstrate passing objects by value to a member function of the same class.

```
#include<iostream>
using namespace std;
class weight {
  int kilogram;
  int gram;
  public:
    void getdata ();
    void putdata ();
    void sum_weight (weight,weight) ;
} ;
void weight :: getdata() {
    cout<<"/nKilograms:";
    cin>>kilogram;
    cout<<"Grams:";
    cin>>gram;
}
void weight :: putdata () {
  cout<<kilogram<<" Kgs. and"<<gram<<" gros.\n";
}
void weight :: sum_weight(weight w1,weight w2) {
  gram = w1.gram + w2.gram;
  kilogram=gram/1000;
  gram=gram%1000;
  kilogram+=w1.kilogram+w2.kilogram;
}
main () {
 weight w1,w2 ,w3;
   cout<<"Enter weight in kilograms and grams\n";
   cout<<"\n Enter weight #1" ;
   w1.getdata();
   cout<<" \n Enter weight #2" ;
   w2.getdata();
   w3.sum_weight(w1,w2);
   cout<<"/n Weight #1 = ";
   w1.putdata();
   cout<<"Weight #2 = ";
   w2.putdata();
   cout<<"Total Weight = ";
   w3.putdata();
}
```

Example

```
#include <iostream>
using namespace std;
class Demo {
private:
    int a;
public:
    void set(int x)
    {
        a = x;
    }
    void sum(Demo ob1, Demo ob2)
    {
        a = ob1.a + ob2.a;
    }
    void print()
    {
        cout << "Value of A :  " << a << endl;
    }
};
main()
{
    Demo d1;
    Demo d2;
    Demo d3;
    d1.set(10);
    d2.set(20);
    d3.sum(d1, d2);
    d1.print();
    d2.print();
    d3.print();
}
```