CS-112
Object Oriented Programming
(3+1)
Prerequisites:
Programming Fundamentals

Syed Muhammad Rafi
Lecturer, Department of Software Engineering
Faculty of Engineering Science and Technology,
Ziauddin University

More Concepts about Data Member and Member Function

Lec-6

- Create a class for counting number of objects created and destroyed within various block using constructor and destructor. Create 3 objects in main and then 1 object in new block within the main and 1 more object after the block. Examine the output.

```
#include<iostream>
using namespace std;
int count=1;
class countobjects
{

public:
countobjects()
{
cout<<"object created"<<count++<<endl;
}
~countobjects()
{
cout<<"object destroyed"<<count--<<endl;
}
};
```

```
int main()
{

countobjects a;
countobjects b;
countobjects c;
}
```

```
object created1
object created2
object created3
object destroyed4
object destroyed3
object destroyed2
```

```
#include<iostream>
using namespace std;
int count=1;
class countobjects
{

public:
countobjects()
{
cout<<"object created"<<++count<<endl;
}
~countobjects()
{
cout<<"object destroyed"<<--count<<endl;
}
};
```

```
int main()
{

countobjects a;
countobjects b;
countobjects c;
}
```

```
object created2
object created3
object created4
object destroyed3
object destroyed2
object destroyed1
```

Objects as Function Parameters

- Objects can also be passed as parameters to member functions.
- The method of passing objects to a functions as parameters is same as passing other simple variables.

- Write a class **Travel** that has the attributes of kilometers and hours. A constructor with no parameter initializes both data parameters to 0.

A member function get() inputs the values and show() display the values.

It has a member function add() that takes an object of type Travel, to add the kilometers and hours of calling object and the parameter and returns an object with added values.

```
#include<iostream>
using namespace std;
class Travel
{
   private:
int km,hr;
   public:
      Travel()
      {
         km=hr=0;

      }
      void get()
      {
         cout<<"Enter Kilometers traveled...";
         cin>>km;
         cout<<"Enter hours Traveled...";
         cin>>hr;
      }
void show()
      {
         cout<<"You Traveled "<<km<<" km in "<<hr<<"hours"<<endl;

      }

void add(Travel p)  //parameter object
      {
         Travel t;    //temporay object
         t.km=km+p.km;
          t.hr=hr+p.hr;
//t.km & p.km  are data members  //temporary object t
//p.km and p.hr are the data members of parameter object p
//without dot are data members of calling object my

         cout<<"Total Travelling is "<<t.km<<" kilometers in "<<t.hr<<"hours"<<endl;

      }
};
int main()
{
   Travel my,your;
   my.get();
   my.show();
   your.get();
   your.show();
   my.add(your);
*/ add() recieve the content of  your object in parameter p value of p represents the value of your object/*
}
```

```
Enter Kilometers traveled...50
Enter hours Traveled...5
You Traveled 50 km in 5hours
Enter Kilometers traveled...60
Enter hours Traveled...6
You Traveled 60 km in 6hours
Total Travelling is 110 kilometers in 11hours
```

How Above Program Works

- The above program declares two objects of class **Travel** and inputs data in both objects.
- The **add()** member function accepts an object of type Travel as parameter.
- It adds the values of data members of the parameter objects and the values of calling object's data members and displays the result.
- The working of member function add() is as follows:

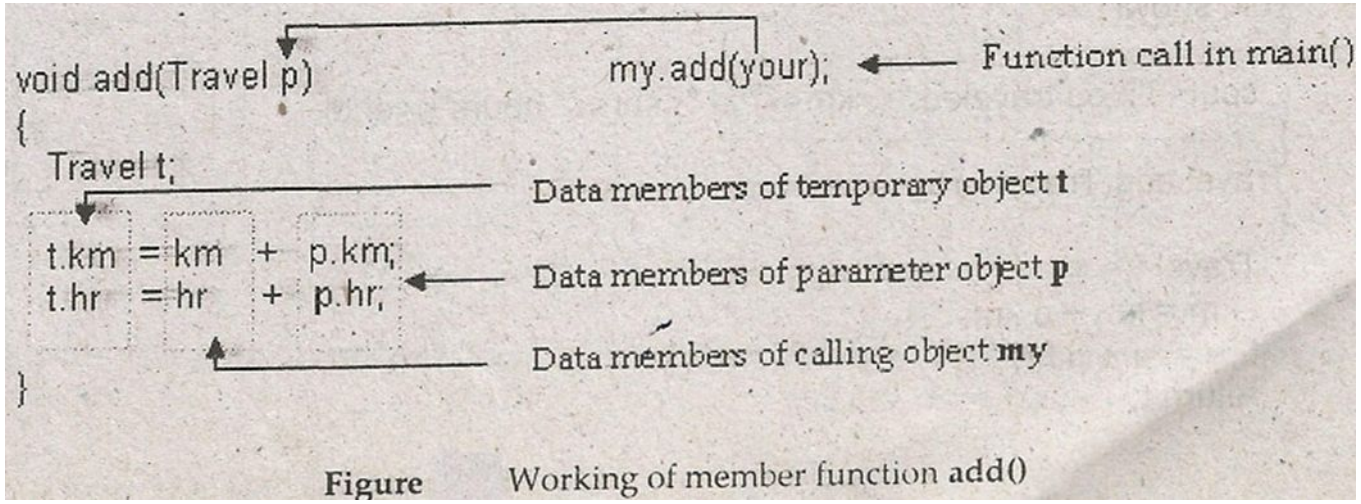

Figure Working of member function add()

Explanation

- The above figure shows that **add()** function receives the contents of **your** object in parameter **p**.
- It means that the value of p represents the values of your object.
- The function adds both values and store the result in temporary object **t** using the following statements:

t.km=km+p.km;

t.hr=hr+p.hr;

- In above statements **t.km** and **t.hr** are the data members of temporary object **t**, **p.km** and **p.hr** are the data members of parameter object **p**.
- The data members without dot operator are the data members of calling object **my**.
- It means that any data member that is not preceded by object name in a member function represents the data member of **calling object.**

Static Data Member

- A type of data member that is shared among all object class is known as **static data member**.
- The static data member is defined in the class with **static** keyword.
- When a data member is defined as static ,only one variable is created in memory even if there are many objects of that class.

Syntax

The syntax for Declaration

```
static data_type member_name;
```

Defining the static data member

It should be defined outside of the class following this syntax:

```
data_type class_name :: member_name =value;
```

Characteristic of Static Data Member

- The characteristic of static data member are same as normal static variable.
- It is visible only in the class in which it is defined but its life start when the program starts its execution.
- The lifetime ends when the entire program is terminated.
- It is normally used to share some data among all objects of a particular class.

Difference Between Normal & Static Data Member

- The main difference between normal data member and static data member is that each object has its own variable of normal data member.
- On the other hand, static data member is shared among all objects of the class.
- Only one memory location is created for static data member that is shared among all objects.

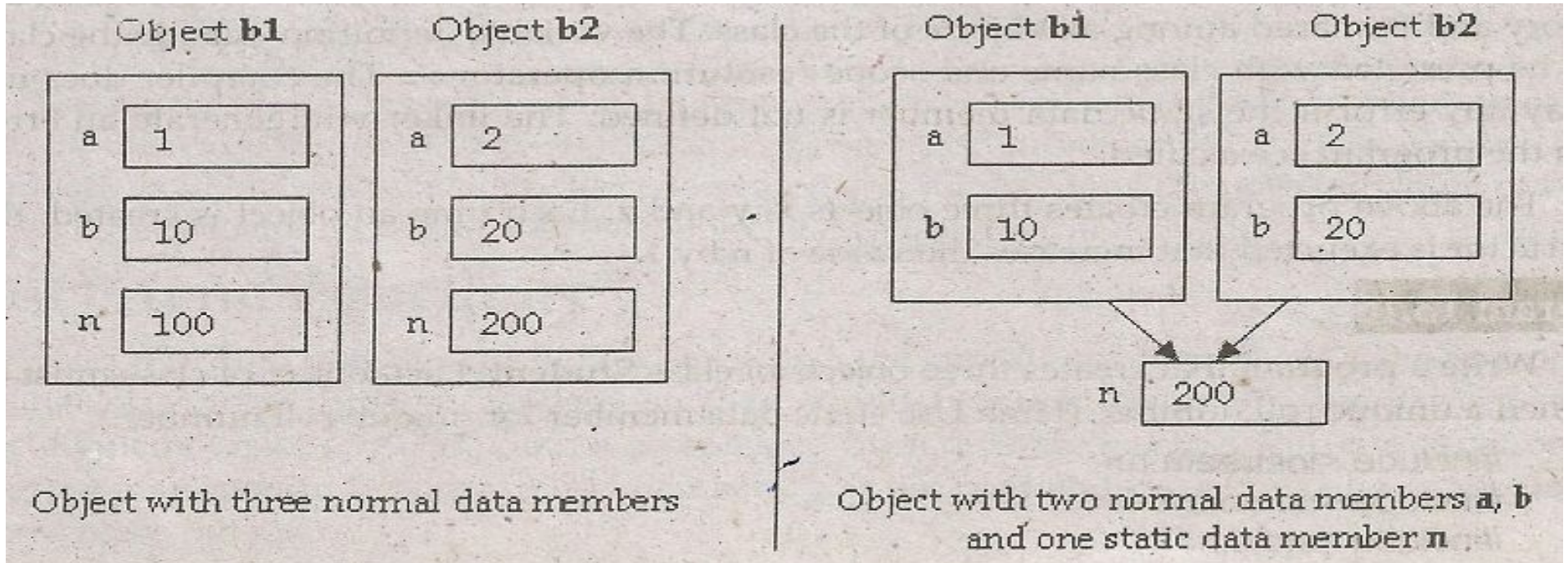

Figure Difference between normal data member and static data member

- Write a program that counts the number of objects created of a particular class.

```
#include <iostream>

using namespace std;

class yahoo
{
private:
   static int n;
public:
   yahoo()
   {
      n++;


   }
   void show()
   {
      cout<<"You  have Created " <<n<< " Objects so
far"<<endl;
   }
};
```

```
int yahoo::n=0;        // defining variable
int main()
{
   yahoo x,y;
   x.show();
   yahoo z;
   x.show();
}
```

```
You  have Created 2 Objects so far
You  have Created 3 Objects so far

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

How Above Program Works

- The above program declares a static data member **n** to count the number of objects that have been created.
- The following statement defines the variable:

int yahoo::n=0;

- The above statement defines the variable and initializes it to 0 value.
- The variable is defined outside the class because it is not the part of any object.
- It is created only once in the memory and is shared among all the objects of the class.
- The variable definition outside the class must be preceded with class name and scope **resolution operator ::** .
- the above program creates three objects **x, y** and **z**. Each time an object is created, the constructor is executed that increase the value of **n** by 1.

Static Member function

- A Type of member function that can be accessed without any object of the class is called **static function.**
- Normally, a member function of any class cannot be accessed without creating an object of that class. In some situations, a member function has to be executed without referencing any object.
- A static member function is a special member function, which is used to access only static data members.
- Any other normal data member cannot be accessed through static member function.
- Just like static data member, static member function is also a class function; it is not associated with any class object.

- The static data member of a class are not created for each object.
- The class creates only one data member for all objects.
- The static data member is defined when the program is executed.
- The static member function can be used to access a static data member.

Syntax

The syntax for Declaration

```
class_name:: function_name(parameter);
```

```
#include <iostream>

using namespace std;

class Demo
{
	private:
		//static data members
		static int X;
		static int Y;

	public:
	//static member function
	static void  Print()
	{
		cout <<"Value of X: " << X << endl;
		cout <<"Value of Y: " << Y << endl;
	}
};
```

```
//static data members initializations
int Demo :: X =10;
int Demo :: Y =20;


int main()
{
	Demo OB;
	//accessing class name with object name
	cout<<"Printing through object name:"<<endl;
	OB.Print();

	//accessing class name with class name
	cout<<"Printing through class name:"<<endl;
	Demo::Print();

	return 0;
}
```

```
Printing through object name:
Value of X: 10
Value of Y: 20
Printing through class name:
Value of X: 10
Value of Y: 20
```

How Above Program Works

- In above program X and Y are two static data members and print() is a **static member function**.
- According to the rule of static in C++, only static member function can access static data members.
- Non-static data member can never be accessed through static member functions.

- Write a program that describe the use of static function, such that it displays thee result witout creating an object in main.

```
#include <iostream>
using namespace std;
class Test
{
private:
    static int n;
public:
    static void show()
    {
        cout<<"n="<<n<<endl;
    }
};
int Test::n=10;
int main()
{
    Test::show();
}
```

```
n=10
```

How above program works

- The above program declares a class Test with a static data member **n**.
- The following statement defines the data member with an initial value of 10.

```
Int Test::n=10;
```

- The program also declares a static member function **show()** that displays the value of **n**.
- The program calls the static member function without creating an object of class as follows

```
Test::show();
```

Write a program that counts the number of objects created of a particular class. The program must be able to display the result even if no object is created so far.

```
#include <iostream>
using namespace std;

class yahoo{
private:
   static int n;
public:
   yahoo()
   {
      n++;
   }
   static void show()
   {
      cout<<"You have created"<<n<<"objects so
far."<<endl;
   }
};
```

```
int yahoo::n=0;
int main()
{
   yahoo::show();
   yahoo x,y;
   x.show();
   yahoo z;
   x.show();

}
```

```
You have created0objects so far.
You have created2objects so far.
You have created3objects so far.
```

Friend Functions

- A type of function that is allowed to access the private and protected members of a particular class from outside the class is called **friend function.**
- Normally, the private and protected members of a class cannot be accessed from outside the class.
- In some situations, a program may requires to access these members. The use of friends function allows the users to access these members.
- A functions that is declared in a class with **friend** keyword becomes the friend function of that class.
- It enables that function to access the private and protected members of that class.

- Write a program which declares two classes A and B. In class A declare a private integer a and public member function A() which has a=10 in its body. Declare a friend function which accepts two objects of both classes as parameter. In class B declare a private integer b and public member function A() which has b=20 in its body. Declare a friend function which accepts two objects of both classes as parameter. Now a member function show with both the classes as objects is declared and in its body integer r is declared which sums the values of a and b and displays value of a, b & a+b individually. In main create objects and display values.

```
#include <iostream>
using namespace std;
class B;
class A
{
private:
   int a;
public:
   A()
   {
      a=10;
   }
   friend void show(A,B);
};
class B
{
private:
   int b;
public:
   B()
{
   b=20;
}

friend void show (A,B);
};
void show(A x,B y)
{
int r;
r=x.a +y.b;
cout<<"The values of class A
object="<<x.a<<endl;
cout<<"The values of class B
object="<<y.b<<endl;
cout<<"The sum of both values
="<<r<<endl;
}
int main()
{
   A obj1;
   B obj2;
   show(obj1,obj2);
}
```

```
The values of class A object=10
The values of class B object=20
The sum of both values =30
```

How Above Program Works?

- The above classes declares two classes A and B.
- Each class contains one data member.
- The program declares a separate function **show()** that accepts the objects of both classes and display the sum of data members in these objects.
- The function must be a friend function of both classes in order to perform this task. The program declare the following statement in both classes.
- Friend void show(A,B);
- The above statement declares tells the compiler that the function is the friend function of the classes. It allowed to access the private and protected data members of these classes.

- The program also specifies the prototype of the class B before class A as follows:

Class B;

- The above declaration is important to specify because a class cannot be referenced before it has been declared.
- The class B is referenced in the declaration of friend function **show()** in class A.
- Therefore, the above class declaration is necessary. Otherwise the compiler will generate an error.

Friend classes

- A type of class all of whose member functions are allowed to access the private and protected members of a particular class is called **friend class.**
- Normally, the private and protected members of any class cannot be accessed from outside the class.
- In some situations, a program may require to access these members.
- The use of friends classes allows a class to access these members of another class.
- A class that is declared in another class with **friend** keyword becomes the friend of that class.

- Write a program which declares two classes A and B. In class A declare a member function A()which declare a= 10 and b=20. Also declare class B as friend in class A. In Class B declare two member functions show A() and show B().Both functions accepts an object of class A as parameter and displays the value of one of its data member.

```
using namespace
std;
class A
{

private:
   int a,b;
public:
   A()
   {
      a=10;
      b=20;
   }
   friend class B;
};

class B
{
public:
   void showA(A obj)
   {
      cout<<"The value of
a:"<<obj.a<<endl;
   }
   void showB(A obj)
   {
      cout<<"The value of
b:"<<obj.b<<endl;
   }
};

int main()
{
   A x;
   B y;
   y.showA(x);
   y.showB(x);
}
```

```
The value of a:10
The value of b:20
```

End of lecture