Object Oriented Programming Using Cpp

Topic:

Polymorphism

Chapter 9: Polymorphism

9.1 Introduction

Polymorphism is made of two greek words "Poly" and "Morph". Poly means many and morph means several forms. So, formally polymorphism can be defined as below:

Definition: The process in which various forms of a single function can be defined and utilized by different objects to perform similar types of operation. The process of polymorphism is categorized into following hierarchy:

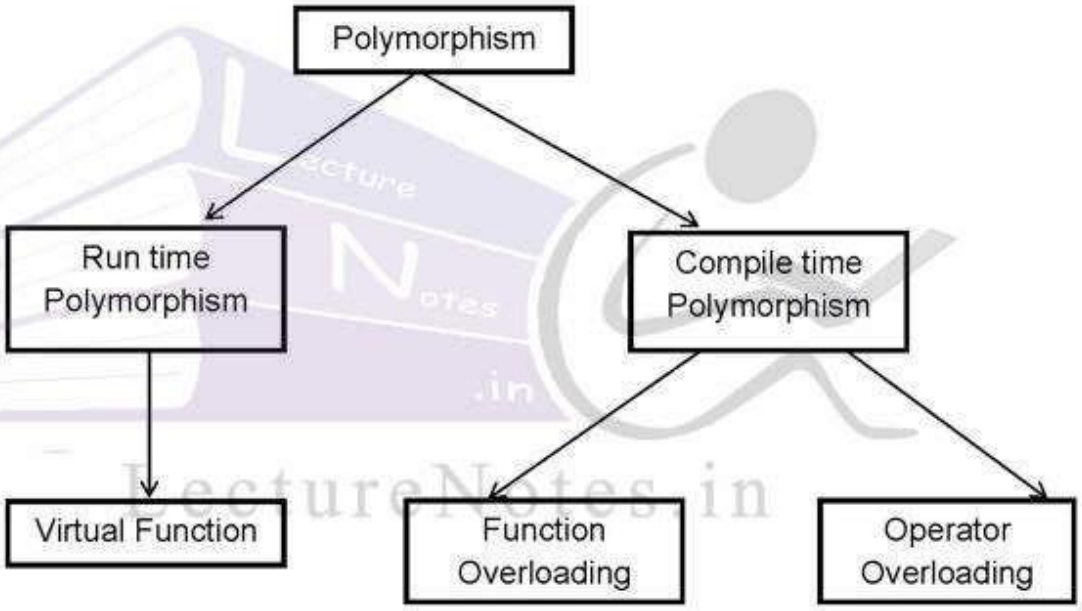


Fig 9.1: Polymorphism

9.2 Compile Time Polymorphism

9.2.1 Function Overloading

C++ allows defining multiple functions with the same name without any problem. The concept is known as function overloading. However, this practice should not be used frequently because it brings some problem also and if the rules for overloading will not be used properly then an ambiguity will arise for the system i.e. the system won't be able to decide that which function should be called because all the functions have the same name.

To overload functions, following rules must be followed strictly:

- The functions can be differentiated in terms of argument type.
- In case, if argument type is same, the functions can be differentiated in terms of number of arguments.
- In case, if both type and number of arguments are same then the functions can be made different by changing the sequence of the arguments.
- If none of the above three rules are satisfied then the functions can't be overloaded.

Program 57: Write a C++ program to explain function overloading.

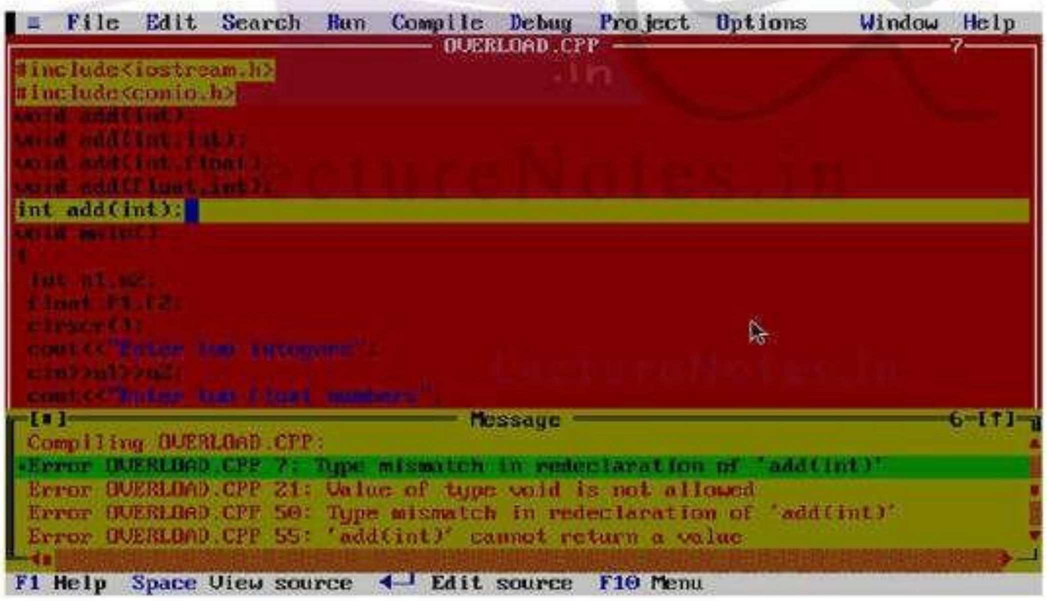
Code:

```
#include<iostream.h>

#include<conio.h>

void add(int);

void add(int,int);

void add(int,float);

void add(float,int);

int add(int);                  // Here only return type is changed, rest is same.

void main()

{

        int n1,n2;

        float f1,f2;

        clrscr();

        cout<<"Enter two integers";

        cin>>n1>>n2;

        cout<<"Enter two float numbers";

        cin>>f1>>f2;

        add(n1);
        add(n1,n2);
        add(n1,f1);
        add(f2,n2);
        int z=add(n1);        //Call to the add function with int return type.
        cout<<z;
        getch();
}
void add(int a)
{
        intnum=5, s1=0;
        s1=num+a;
        cout<<"Sum is "<<s1<<endl;
}
void add(int b, int c)
{
        int s2=0;
        s2=b+c;
        cout<<"Sum is "<<s2<<endl;
}
void add(int d, float x)
{
        float s3=0.0;
        s3=d+x;
        cout<<"Sum is "<<s3<<endl;
}
void add(float y, int e)
{
```

```
            float s4=0.0;

            s4=y+e;

            cout<<"Sum is "<<s4<<endl;

        }

        int add(int p)

        {

            int q=5;

            int s5=0;

            s5=p+q;

        return(s5);

}
```

O/P:

At present the program gives the following error due to the ambiguity cause during function call of 1st add() and 5th add() cause these two functions are not being differentiated by the system successfully as their only return type is different.

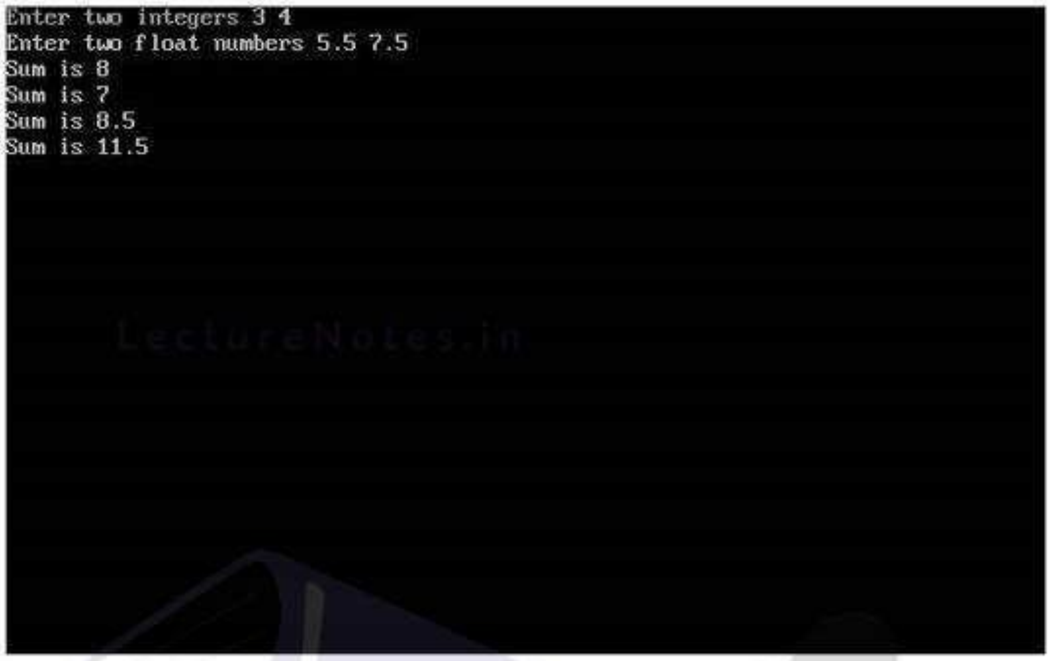


```
#include<iostream.h>
#include<conio.h>
void add(int);
void add(int,int);
void add(int,float);
void add(float,int);
int add(int);
void main()
{
 int n1,n2;
 float f1,f2;
 clrscr();
 cout<<"Enter two integers";
 cin>>n1>>n2;
 cout<<"Enter two float numbers";
```

```
Compiling OVERLOAD.CPP:
•Error OVERLOAD.CPP 7: Type mismatch in redeclaration of 'add(int)'
 Error OVERLOAD.CPP 21: Value of type void is not allowed
 Error OVERLOAD.CPP 50: Type mismatch in redeclaration of 'add(int)'
 Error OVERLOAD.CPP 55: 'add(int)' cannot return a value
```

≡ File Edit Search Run Compile Debug Project Options Window Help
OVERLOAD.CPP
7
Message
6
F1 Help Space View source ◄┘ Edit source F10 Menu

Now, if the 5th function is removed from the program we will get following output:



```
Enter two integers 3 4
Enter two float numbers 5.5 7.5
Sum is 8
Sum is 7
Sum is 8.5
Sum is 11.5
```

Note: return type is not considered as a rule for overloading functions because function can return only when it is executed and a function can execute only if it is called successfully. The main problem arises during function call. So, it is very important to differentiate the functions in terms of arguments.

9.2.2 Operator Overloading

It is an important and useful feature of C++. Formally, it can be defined as the capability to relate the existing operator with a member function so that the resulting operator function may be used with the objects of its class without changing the nature of the operator. Operator overloading is performed with the help of a keyword **"operator"**.This keyword defines a new action or operation for the existing operator.

Syntax:

```
    Return_type        operator      <operator_symbol> (arguments)
    {



    }
```

Operator overloading is of following two types:

i. Unary operator overloading: When unary operator is overloaded, the process is known as unary operator overloading. It can be done in following two ways:

- ➢ Using member function
- ➢ Using friend function

9.2.2.1 Unary operator overloading using member function

In this method, the unary operator is overloaded using a member function of a class. The required one operand is provided by the system using the calling object, in this method. Due to this reason the operator function does not requires any argument.

Syntax:

```
class        <class_name>
{
        Data members;
        public:
                return_type  operator      <operator_symbol>()
                {


                }
};
```

In the above syntax, the operator function is defined inside the class. However, it may be defined outside the class as per the requirement. The syntax is same in this case as we discussed while defining a member function outside the class.

Program 58: Write a program in C++ to overload unary operator ++ using member function method.

Code:

```
#include<iostream.h>
#include<conio.h>
class NUM
{
        int n1,n2,n3;
```

```
public:
        NUM(intx,inty,int z)
        {
                n1=x;
                n2=y;
                n3=z;
        }
        void show();
        void operator ++();
};
void NUM::show()
{
        cout<<"n1="<<n1<<endl<<"n2="<<n2<<endl<<"n3="<<n3<<endl;
}
void NUM::operator ++()
{
        n1=n1+5;
        n2=n2+10;
        n3=n3+15;
}
void main()
{
        clrscr();
        NUM obj(10,20,30);
        cout<<"The entered numbers are as below:"<<endl;
        obj.show();
        obj.operator ++();          //Member function call
```

```
            cout<<"After execution of operator function:"<<endl;
            obj.show();
            getch();
    }
```

O/P:

```
    The entered numbers are as below:
    n1=10
    n2=20
    n3=30
    After execution of operator function
    n1=15
    n2=30
    n3=45
```

Note: Refer Kamthane book page no 309, Program no: 8.4 for another program.

9.2.2.2 Unary operator overloading using friend function

In this method, the unary operator is overloaded using a friend function. In this case, the required one operand is provided by the user. Due to this reason the operator function requires one argument. Since friend function is involved, the argument must be of class type i.e. an object is used as argument.

Syntax:

```
    class       <class_name>
    {
            Data members;
            public:
                    friend return_type  operator   <operator_symbol>(classname);
    };
```

```
Return_type        operator      <operator_symbol> (classname   obj)
{


}
```

In the above syntax, the operator function is defined outside the class as friend function is used.

Program 59: Write a program in C++ to overload unary operator ++ using friend function method.

Code:

```
#include<iostream.h>
#include<conio.h>
class NUM
{
        int n1,n2,n3;
public:
        NUM(intx,inty,int z)
        {
                n1=x;
                n2=y;
                n3=z;
        }
        void show();
        friend  void operator ++(NUM);
};
void NUM::show()
{
        cout<<"n1="<<n1<<endl<<"n2="<<n2<<endl<<"n3="<<n3<<endl;
}
```

```
void    operator ++(NUM    o1)

{

 o.1n1=o1.n1+5;

 o1.n2=o1.n2+10;

 o1.n3=o1.n3+15;

cout<<"n1="<<o1.n1<<endl<<"n2="<<o1.n2<<endl<<"n3="<<o1.n3<<endl;

}

void main()

{

        clrscr();

        NUM obj(10,20,30);

        cout<<"The entered numbers are as below:"<<endl;

        obj.show();

        cout<<"After execution of operator function:"<<endl;

        operator ++(obj);    //friend function call

        getch();

}
```

O/P:

```
The entered numbers are as below:

n1=10

n2=20

n3=30

After execution of operator function

n1=15

n2=30

n3=45
```

Note:

Operator function with return

Like normal functions, it is also possible to return from an operator function. One important thing to remember here is that, operator function always returns object. This is because operator overloading is only for objects.

Program 60: Write a program in C++ to overload unary operator ++ using friend function method and operator function must return.

Code:

```
#include<iostream.h>

#include<conio.h>

class NUM

{

        int n1,n2,n3;

public:

        NUM(intx,inty,int z)

        {

                n1=x;

                n2=y;

                n3=z;

        }

        void show();

        friend  NUM operator ++(NUM);

};

void NUM::show()

{

        cout<<"n1="<<n1<<endl<<"n2="<<n2<<endl<<"n3="<<n3<<endl;

}
```

```
NUM  operator ++(NUM    o1)

{

  o.1n1=o1.n1+5;

  o1.n2=o1.n2+10;

  o1.n3=o1.n3+15;

  return(o1);

}

void main()

{

        clrscr();

        NUM obj(10,20,30);

        cout<<"The entered numbers are as below:"<<endl;

        obj.show();

        cout<<"After execution of operator function:"<<endl;

        NUM o2;

        o2=operator ++(obj);       //friend function call

        o2.show();

        getch();

}
```

O/P:

```
The entered numbers are as below:

n1=10

n2=20

n3=30

After execution of operator function

n1=15

n2=30
```

```
n3=45
```

ii. Binary operator overloading: When binary operator is overloaded, the process is known as binary operator overloading. It can be done in following two ways:

- Using member function
- Using friend function

9.2.2.3 Binary operator overloading using member function

In this method, the binary operator is overloaded using a member function of a class. Since binary operator is used, two operands are required. One operand is provided by the system using the calling object and another operand is provided by the user. Due to this reason the operator function requires one argument. The argument may be of basic type or class type.

Syntax:

```
class        <class_name>
{
        Data members;
        public:
                return_type   operator      <operator_symbol>(arg)
                {

                }
};
```

In the above syntax, the operator function is defined inside the class. However, it may be defined outside the class as per the requirement. The syntax is same in this case as we discussed while defining a member function outside the class.

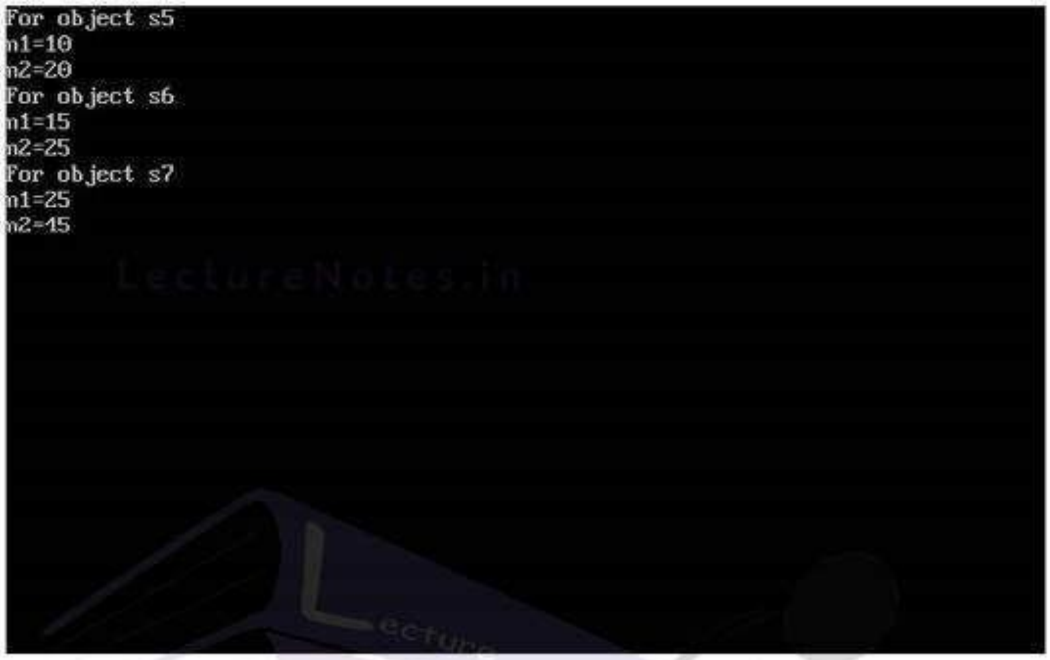
Program 61: Write a program in C++ to overload binary operator '+' using member function method.

Code:

```
#include<iostream.h>

#include<conio.h>

class sample

{

        int n1,n2;
public:
        sample()        //Zero argument or default constructor
        {
        }
        sample(intx,int y)
        {
                n1=x;
                n2=y;
        }
        void show();
        sample operator +(int);
        sample operator +(sample);
};
void sample::show()
{
        cout<<"n1="<<n1<<endl<<"n2="<<n2<<endl;
}
sample sample::operator +(int x)
{
        sample s1;
        s1.n1=n1+x;
        s1.n2=n2+x;
        return(s1);
}
```

```
sample sample::operator +(sample s3)
{
        sample s4;
        s4.n1=n1+s3.n1;
        s4.n2=n2+s3.n2;
        return(s4);
}
void main()
{
        clrscr();
        sample s5(10,20);
        cout<<"For object s5"<<endl;
        s5.show();
        sample s6;
        s6=s5.operator +(5);       //Explicit call to first operator function
        cout<<"For object s6"<<endl;
        s6.show();
        sample s7;
        s7=s6.operator +(s5);    //Explicit call to second operator function
        cout<<"For object s7"<<endl;
        s7.show();
        getch();
}
```

O/P:



```
For object s5
n1=10
n2=20
For object s6
n1=15
n2=25
For object s7
n1=25
n2=45
```

Explanation: In the above program, class sample contains two integer data members n1 and n2 along with a do nothing constructor. It also contains a parameterized constructor (to initialize the data members), member functions show() and two operator functions. First operator function contains a basic data type as argument whereas second operator function contains a class type as argument. The member functions are defined outside the class. In the main function, object s5 is declared with 10 and 20 as arguments which executes the parameterized constructor. After that data members for object s5 is displayed using show(). In 1st call to the operator function, first operator function is called as integer is used as argument. The operator function is called using object s5 i.e. this operand is provided by the system whereas the second operand, x, is provided by the user. The operator function gets executed and the result is returned using object s1. This object is received by another object s6 inside main function. Using member function show(), the data members for s6 is displayed. Same process is repeated during 2nd call to the operator function.

Note:The statements**s6=s5.operator +(5);**and**s7=s6.operator +(s5);**are the explicit call to operator function.. Same can be done using implicit call as mentioned below:

```
s6=s5 + 5;      //Implicit call to first operator function
s7=s6 + s5;     //Implicit call to second operator function
```

However, this is not suitable way of calling as it hides the implementation method of binary operator overloading.

9.2.2.4 Binary operator overloading using friend function

In this method, the binary operator is overloaded using a friendfunction. Since binary operator is used, two operands are required. Since friend function is used, both operands are provided by the user. Due to this reason the operator function requires two arguments. The argument may be provided using any one of the three following ways:

- Classtype, classtype
- Classtype, basic
- Basic, classtype

Syntax:

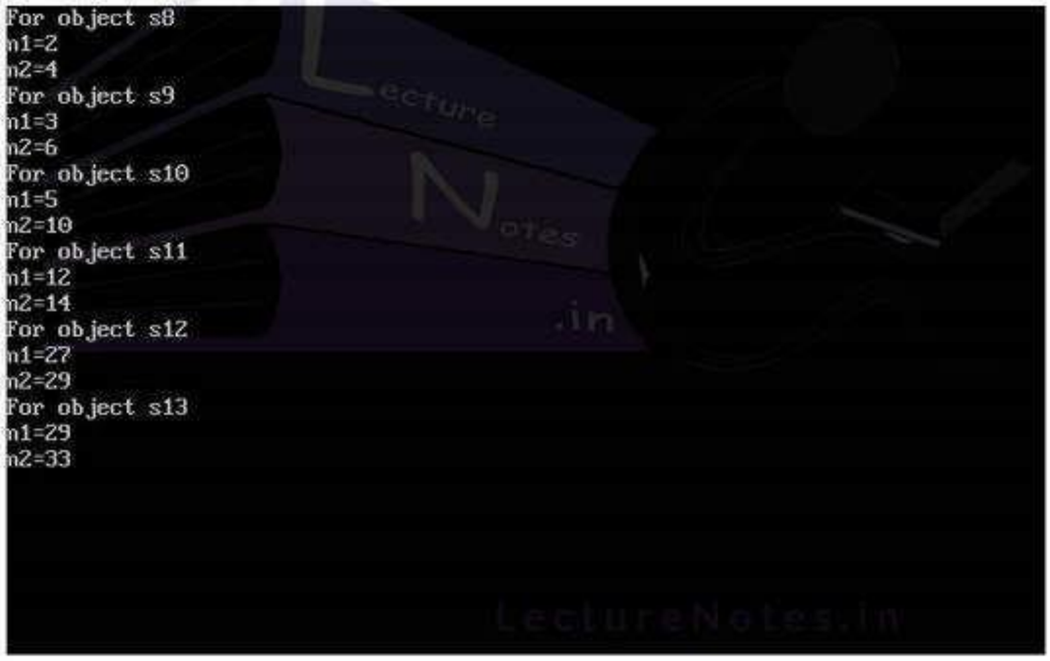
```
class          <class_name>
{
        Data members;
public:
        friend return_type  operator      <operator_symbol>(arg);
};
return_type  operator      <operator_symbol>(arg)
{

}
```

Program 62: Write a program in C++ to overload binary operator '+' using friend function method.

Code:

```
#include<iostream.h>

#include<conio.h>

class sample

{

            int n1,n2;

        public:

            sample()
            {
            }
            sample(intx,int y)
            {
                    n1=x;
                    n2=y;
            }
            void show();
            friend sample operator +(sample,sample);
            friend sample operator +(sample,int);
            friend sample operator +(int,sample);
};
void sample::show()
{
        cout<<"n1="<<n1<<endl<<"n2="<<n2<<endl;
}
sample operator +(sample s1,sample s2)
{
        sample s3;
        s3.n1=s1.n1+s2.n1;
        s3.n2=s1.n2+s2.n2;
        return(s3);
}
sample operator +(sample s4,int x)
{
        sample s5;
        s5.n1=s4.n1+x;
        s5.n2=s4.n2+x;
        return(s5);
}
sample operator +(inty,sample s6)
{
        sample s7;
        s7.n1=y+s6.n1;
        s7.n2=y+s6.n2;
        return(s7);
}
void main()
{
        clrscr();
        sample s8(2,4),s9(3,6);
        cout<<"For object s8"<<endl;
        s8.show();
        cout<<"For object s9"<<endl;
        s9.show();
        sample s10;
        s10=operator +(s8,s9);    //Call to 1st operator function
        cout<<"For object s10"<<endl;
        s10.show();
        sample s11;
        s11=operator +(s8,10);    //Call to 2nd operator function
        cout<<"For object s11"<<endl;
        s11.show();
        sample s12;
        s12=operator +(15,s11);  //Call to 3rd operator function
        cout<<"For object s12"<<endl;
        s12.show();
        sample s13;
        s13=operator +(s8,s12);  //Call to 1st operator function
        cout<<"For object s13"<<endl;
        s13.show();
        getch();
    }
```

O/P:



```
For object s8
n1=2
n2=4
For object s9
n1=3
n2=6
For object s10
n1=5
n2=10
For object s11
n1=12
n2=14
For object s12
n1=27
n2=29
For object s13
n1=29
n2=33
```

Explanation: In the above program, class sample contains two integer data members n1 and n2 along with a do nothing constructor. It also contains a parameterized constructor (to initialize the data members), member functions show() and two operator functions which are friend functions. First operator function contains two objects as arguments, second operator function contains an object and one integer as arguments. Similarly, 3rd operator function contains 1st argument as integer and 2nd argument as object.

In the main function, objects s8 and s9 are declared with 2, 4and 3, 6 as arguments which executes the parameterized constructor twice. After that data members for objects s8 and s9 are displayed using show(). In 1st call to the operator function, first operator function is called as two objects are used as arguments. The operator function gets executed and the result is returned using object s3. This object is received by another object s10 inside main function. Using member function show(), the data members for s10 is displayed. Same process is repeated during other function calls to the operator function further in the program.

Note: The statement**s10=operator +(s8,s9);**is the explicit call to operator function. Same can be done using implicit call as mentioned below:

S10=s8 + s9; //Implicit call to first operator function

The same technique can be applied everywhere when an operator function is called.However, this is not suitable way of calling as it hides the implementation method of binary operator overloading.

9.2.2.5 Rules to remember for operator overloading

- Operator overloading does not follow commutative property **i.e. s11=s8+10; and s11=10+s8; are not same.**
- In case of unary operator overloading, member function method uses operand provided by the system. Due to this operator function does not have any argument. On the other hand, friend function method uses the operand provided by the user due to which the operator function must have one argument. **The argument must be of class type.**
- In case of binary operator overloading, member function method uses one operand provided by the system and another operand provided by the user. **Due to this the operator function has one argument which may be of class type or basic type.** On the other hand, friend function method uses both operands provided by the user. Due to this the operator function must have two arguments in **class,class or class,basic or basic,class** manner.
- Operator overloading does not change the original nature of the object used. That's why it is better to perform operation similar to the operator used, in the operator function.
- Following operators can't be overloaded: **sizeof(), dot (.) , scope resolution operator (: :) , conditional operator (? :)**
- Following operators can't be overloaded using friend function: **Assignment operator, Function call operator (), Subscript operator [ ], class member access operator (→)**