Polymorphism in C++

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Polymorphism means having multiple forms of one thing. It can be done in two types:

1) At Compile Time
2) At Run Time

In Compile time it can be done it two ways:

1) Function Overloading
2) Operator Overloading

In Run rime it can be done with the help of virtual functions.

Compile Time Polymorphism:

When the selection of appropriate function or operator at Compile Time , is called Early Binding or Static Binding or static linking. Also known as Compile Time Polymorphism.

Run Time Polymorphism:

When the selection of appropriate function or operator at Run Time , is called Late Binding or Dynamic Binding. Also known as Run Time Polymorphism.

Function Overloading in C++

If any class having multiple functions with the same name but different parameters then they are said to be function overloading.

In inheritance, polymorphism is done, by method overriding, when both super and sub class have member function with same declaration but different definition.

There are two ways to overload a function:

1) By changing number of arguments.
2) By having different types of arguments.

By changing number of arguments

```
Class A
{
public:
int sum(int x, int y)
{
cout<<x + y<<endl;
}
int sum(int x)
{
cout<<x<<endl;
}
}
```

```
int sum(int x, int y, int z)
{
cout<<x + y + z<<endl;
}
};

void main()
 {
A  b;
b.sum(3,4);
b.sum(5,3,9);
b.sum(5);
}
```

Output:

7

17

5

By having different data types of arguments

```
Class A
{
Public:
int sum(int x, int y)
{
cout<<x + y<<endl;
}
double sum(double x)
{
cout<<x<<endl;
}
}
```

```
float sum(float x, float y)
{
cout<<x + y<<endl;
}
};

void main()
 {
A  b;
b.sum(3,4);
b.sum(5.9);
b.sum(5.34F, 3.45F);
}
```

Output:

7

5.9

8.79

Function with Default Arguments

When we declared a function with default parameters then it is said to be Function with default arguments .

C++ allows us to call a function without specifying all its arguments.

In such cases, the function assigns a default value to the parameter which does not have a matching argument in the function call.

Rules :

Only the last argument must be given default value. You cannot have a default argument followed by non default argument.

1)	sum(int x, int y);	//	valid
2)	sum(int x, int y = 0);	//	valid
3)	sum(int x = 0, int y);	//	not valid
4)	sum(int x, int y=0, int z);	//	not valid

```
class A
{
public:
int sum(int x, int y=9)
{
cout<<x + y<<endl;
}
void main()
 {
A  b;
b.sum(3,4);
b.sum(5);
}
```

Output:

7

14

Operator Overloading in C++

The meaning of operators are already defined and fixed for basic types like: int, float, double etc in C++ language. For example: If you want to add two integers then, + operator is used. But, for user-defined types(like: objects), you can define the meaning of operator, i.e, you can redefine the way that operator works. For example: If there are two objects of a class that contain string as its data member, you can use + operator to concatenate two strings. Suppose, instead of strings if that class contains integer data member, then you can use + operator to add integers.

This feature in C++ programming that allows programmer to redefine the meaning of operator when they operate on class objects is known as operator overloading.

Overloaded Operators:

-	*	/	%	^	
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

These operator can’t be overloaded:

1) Class members access operators(. , *)
2) Scope Resolution Operator (::)
3) Size Operator (sizeof)
4) Conditional Operator (?:):

The reason behind this that these operators takes names (class name) as their operand instead of values, as in the case with other normal operators.

Defining Operator Overloading :

- To define an additional task to an operator, we must specifywhat it means in relation to th class to which the operator is applied. This is done with the help of a special function, called operator function, which described the task.

- Syntax:
- Return_type class_name :: operator op(parameter)
- {
- ……
- …… // Function Body
- }

- Here op is the operator being overloaded & operator is a keyword.

Operator functions must be either member functions(non static) or friend function.

A basic difference between them is that a friend function will have only one argument for unary operator & two for binary operator.

While a member function has no argument for unary & one argument for binary operator.

This is because the object used to invoke the member function is passed implicitly and therefore is available for member function.

This is not the case with friend function. Arguments may be passed either by value or by reference.

Overloaded Unary Operator

The unary operators operate on a single operand.

They operate on the object for which they were called and normally, this operator appears on the left side of the object, as in !obj, -obj, and ++obj but sometime they can be used as postfix as well like obj++ or obj--.

Following are the examples of Unary operators:

1) The unary minus (-) operator.
2) The logical not (!) operator.
3) The increment (++) and decrement (--) operators.

Unary minus operator:

```
class A
{
   int a,b,c;
public:

void getdata (int x, int y, int z)
{
   a=x;        b=y;        c=z;
}

void display()
{
   cout<<"a:" <<"\t"<<a<<endl;
   cout<<"b:" <<"\t"<<b<<endl;
   cout<<"c:" <<"\t"<<c<<endl;
}
```

```
void operator - ()     // Operator - () takes no argument. It changes the
      sign of data member of the object S. Sine this function is a member
      function of the same class. It can directly access the member of the
      object which activated it.
{
   a=-a;       b=-b;   c=-c;
}
};

int main()
{
   A m;
   m.getdata(10,20,30);
   m.display();
   -m ;   // Activates operator – function
   m.display();
}
```

Output:

10,	**20,**	**30**
-10	**-20**	**-30**

Overloaded binary Operator

The binary operators take two arguments.

Following are the examples of Unary operators:

1) Addition (+) operator
2) Subtraction (-) operator
3) Division (/) operator.

Binary (+) operator:

```
class A
{
   int a, b;
public:
   void getdata(int x, int y)
   {
      a = x;
      b = y;
   }
   void display()
   {
      cout<<a<<"+"<<b<<"i"<<endl;
   }
   A operator+ ( A ob)
   {
   A t;
   t.a= a+ob.a;
   t.b= b+ob.b;
   return t;
   }
```

```
A operator- ( A ob)
   {
   A t;
   t.a= a-ob.a;
   t.b= b-ob.b;
   return t;
   }
};
int main()
{
   A obj1,obj2,result,result1;
   obj1.getdata(50,30);
   obj2.getdata(10,20);
   result=obj1+obj2;
cout<<"Input values are\n";
   obj1.display();
   cout<<"Result are:\n";
   result. display();
}
```

Output:

Input values are:

50 + 30i

10 + 20i

Result are:

60 + 50i

40 + 10i

Overloaded binary Operator using Friend Function:

As already discussed, friend functions may be used in the place of member functions for overloading a binary operator, the only difference between that a friend function requires two arguments to be explicitly passed to it, while a member function requires only one.

Syntax:

```
friend Num operator +(Num a, Num b); //  Declaration

Num operator +(Num a, Num b)
{
return Num ((a . x  + b . x), (a . y + b . y));//   Definition
}
```

Binary (+) operator using friend function:

```
class s
{
public:
int i,j;

s()
{
   i=j=0;
}
void disp()
{
   cout << i <<" " << j;
}
void getdata()
{
   cout<<"Enter the values of i & j\n";
   cin>>i>>j;
}
friend s operator+(int,s);
};

s operator+(int a, s s1)
{
  s k;
  k.i = a+s1.i;
  k.j = a+s1.j;
  return k;
}

int main()
{
s s2;
s2.getdata();
s s3 = 10+s2;
s3.disp();
}
```

Output:

```
Enter the values of i & j :
4     5

14  15
```

Rules for Overloading Operators

In C++, following are the general rules for operator overloading.

1) Only built-in operators can be overloaded. New operators can not be created.
2) Precedence and associativity of the operators cannot be changed.
3) Overloaded operators cannot have default arguments except the function call operator () which can have default arguments.
4) Operators cannot be overloaded for built in types only. At least one operand must be used defined type.
5) Assignment (=), subscript ([]), function call ("()"), and member selection (->) operators must be defined as member functions.
6) Except the operators specified in point 6, all other operators can be either member functions or a non member functions.
7) Some operators like (assignment)=, (address)& and comma (,) are by default overloaded