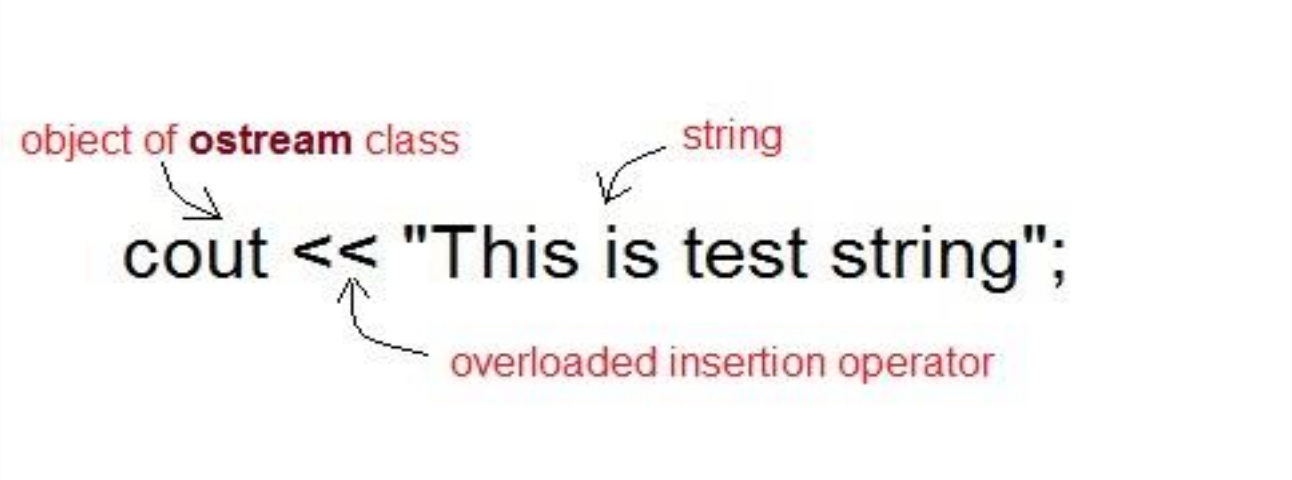
Operator Overloading

Contents

- Operator Overloading
- Overloading Unary Operator
- Overloading Binary Operator
- Rules for overloading operator
- Type Conversion

Operator Overloading

- It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it.
- Overloaded operator is used to perform operation on user-defined data type.



How to do

Keyword Operator to be overloaded

```
ReturnType classname :: Operator OperatorSymbol (argument list)
{
        \\ Function body
}
```

- The return type comes first which is followed by keyword **operator**, followed by operator sign,i.e., the operator you want to overload like: +, <, ++ etc. and finally the arguments is passed. Then, inside the body of you want perform the task you want when this operator function is called.
- This operator function is called when, the operator(sign) operates on the object of that class class_name.

How we can implement

- Operator overloading can be done by implementing a function which can be :
  - Member Function
  - Friend Function
- Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, then Operator overloading function must be a non-member function.
- Operator overloading function can be made friend function if it needs access to the private and protected members of class.

Points to be remember

- Precedence and Associatively of an operator cannot be changed.
- Arity (numbers of Operands) cannot be changed. Unary operator remains unary, binary remains binary etc.
- No new operators can be created, only existing operators can be overloaded.
- Operator overloading cannot be used to change the way operator works on built-in types. Operator overloading only allows to redefine the meaning of operator for user-defined types.
- There are two operators assignment operator(=) and address operator(&) which does not need to be overloaded. Because these two operators are already overloaded in C++ library.

Operators that are not overloaded

- scope operator - **::**
- Sizeof
- member selector - **.**
- member pointer selector - ***
- ternary operator - **?:**

Overloading Unary Operator

- The unary operators 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--.
- The unary operators operate on a single operand and following are the examples of Unary operators:
  - The increment (++) and decrement (--) operators.
  - The unary minus (-) operator.
  - The logical not (!) operator.

Program on overloading unary operator

```
#include<iostream>
using namespace std;
class uopoverload
{
    int a,b,c;
   public:
       void getvalue()
    {
            cout<<"Enter the Two Numbers:";
            cin>>a>>b;
    }
```

```
void operator++(int)
    {
            a=++a;
            b=++b;
    }

    void operator--()
    {
            a=--a;
            b=--b;
    }
void display()
    {
            cout<<a<<"\t"<<b<<endl;
    }
};
```

```
int main()
{

    uopoverload obj;
    obj.getvalue();
    ++obj;
    cout<<"Increment Complex Number\n";
    obj.display();
    ++obj;
    cout<<"Decrement Complex
Number\n";
    obj.display();
    return 0;
}
```

Overloading Binary Operator

- The binary operators take two arguments and following are the examples of Binary operators. You use binary operators very frequently like addition (+) operator, subtraction (-) operator and division (/) operator.
- We can overload binary operator:
  - With Member Functions
  - With Friend Functions

Overloading with member function

- If overloading as a member function ,binary operator requires one argument.
- The argument contains the value of object, which is to the right of operator.

```
operator(num o2);
Where, num is a class name and o2 is an object.
To call function operator the statement is as follows:
o3=o1+o2;
```

OR

```
o3=o1.operator +(o2);
```

Program on Overloading Binary Operator

```
#include <iostream>

using namespace std;

class num
{
   int a=10;
public:
   num operator+(num x)
    {
       num y;
       y.a=a+x.a;
       return(y);
    }
    void display()
     {
        cout<<"Value is"<<a;
     }

};
```

```
int main()
  {
      num n,p,q;
      n=p+q;
      n.display();
      return 0;
  }
```

Overloading with Friend Functions

- If overloading as a friend function ,binary operator requires two argument
- Friend functions are useful when we require performing an operation with operand of two different types.
- Friend function can be called without using an object.

```
o3=o1+o2;
o3=operator+(o1,o2);
```

Program on operator overloading using friend function

```
#include <iostream>
using namespace std;
class fload
 {

  public:
   int i,j;
   fload()
    {
        i=0;
        j=0;
   }
   void getvalue()
    {
        cout<<"enter values";
        cin>>i>>j;
    }
   void display()
       {
           cout<<i<<j;
           }
friend fload operator+(fload,int);
 };

 fload operator+(fload f,int a)
    {
        fload k;
        k.i=a+f.i;
        k.j=a+f.j;
        return k;;
    }
    int main()
    {
        fload f1;
        f1.getvalue();
        fload f2=f1+20;
        f2.display();
        return 0;
    }
```

Type conversion

- The C++ compiler has a way to deal with expressions that contains variables or constants which have different data types.
- So if we have two different types of variables/constants in an expression, they can be converted to one single type (either by the compiler or explicitly by the programmer).
- This process is known as type conversion in C++.

Implicit Conversion

- Implicit Type Conversion is performed by the compiler on its own when it encounters a mixed expression in the program.
- This is also known as automatic conversion as it is done automatically by the compiler without the programmer's assistance.
- In this type of conversion, all operands are converted upto the type of the largest operand in the expression. This is also called as type promotion.
- Thus, the result of the expression is also of the type of the largest operand.

Explicit Conversion

- When the programmer explicitly changes the data type of an operand or an expression, then this is called as explicit type conversion.
- It is also known as type casting.