Operator Overloading

- C++ provides a keyword for operators known as ***operator***
- Every operator is associated with an operator function that defines it behavior.
- Operator function definition is predefined for built-in types cannot be redefined.

Operator expression	**Operator function call**
a+b	operator+(a,b)
a=b	operator=(a,b)
c=a+b	operator=(c , operator+(a,b))

- A non member operator function can be
  - Global function
  - Friend function
- C++ allows to define the own meaning for the standard c++ built-in operators when they are applied to class types is known as operator overloading.
- An overloaded operator is called an operator function.

- Binary operator

  MyType a,b;

  Syntax:

  MyType operator+(const MyType& ,const MyType&){} // global function

  friend MyType operator+(const MyType& ,const MyType&){} // friend function

- Unary operator

  Syntax:

  MyType operator++(const MyType&){} // global function

  friend MyType operator++(const MyType&){} // friend function

Note: parameters may be constant or pass by values and return type may be reference or value

Overload Binary Operator

Example : Binary Addition Operator '+' .

Pre-defined type: compiler will provide the add logic depends on the type of operands

User-defined type: Compiler does not provide , user has to provide the code

Type	Operator +	Description	
integer	int x=10, y=20 z=x + y =30	Integer addition	Overloaded
float	float x=2.34, y=4.56 z= x + y = 2.34 + 4.56 = 6.90	Float addition- Adding mantissa and exponent separately	
string	string s1="hello" ,s2="welcome" string s3=s1+s2="hello"+ "welcome" = hellowelcome	String concatenation	
complex (user-defined)	c1 = 2 + i3 , c2 = 5 + i2 C = c1 + c2 = 2 + i3 + 5 + i2 = 7 + i5	Complex addition –adding real part and imaginary part separately.	

Syntax:

```
return-type operator+(argument-list){
        // logic
}
```

Example : Operator overloading

Operator+ is a non-member friend function of class complex

```
class complex{
        double real,img;
        public:
           complex(double r =0.0,double i=0.0){
                 real=r;img=i;
            }
            void display(){
                 cout<<real<<" "<< showpos<<img<<"i"<< noshowpos<<endl;
            }
          friend complex operator+(const complex&,const complex&);
};
```

To display sign of operand

Not to display sign of operand

c1+c2 changes to call operator+(c1,c2)

```
complex operator+ (const complex& c1,const complex& c2)
{
                complex r;
                r.real=c1.real+c2.real;
                r.img=c2.img+c2.img;
                return r;
}
```

```
int main(){
    complex c1(2.1,3.4),c2(4.3,5.1);
    complex r=c1+c2;
    cout<<"complex number 1: ";
    c1.display();
    cout<<"complex number 2: ";
    c2.display();
    cout<<"addition of complex number 1 and 2 : ";
    r.display();
    return 0;
}
```

Output:

```
complex number 1: 2.1 +3.4i
complex number 2: 4.3 +5.1i
addition of complex number 1 and 2 : 6.4 +8.5i
```

Example:

Operator+ is a member function(known as operator function) of class complex

```
class complex{
        double real,img;
        public:
           complex(double r =0,double i=0){
                 real=r;img=i;
           }
           complex operator+(const complex& c){
                 this->real=real+c.real;
                 this->img=img+c.img;
                 return *this;
           }
           void display(){
                cout<<real<<" "<< showpos<<img<<"i"<< noshowpos<< endl;
          }
};
```

c1+c2 changes to call c1.operator+(c2)

```
int main(){
   complex c1(2.1,3.4),c2(4.3,5.1);
   complex r=c1+c2;
   cout<<"complex number 1: ";
   c1.display();
   cout<<"complex number 2: ";
   c2.display();
   cout<<"addition of complex number 1 and 2 : ";
   r.display();
   return 0;
}
```

Output:

```
complex number 1: 2.1 +3.4i
complex number 2: 4.3 +5.1i
addition of complex number 1 and 2 : 6.4 +8.5i
```

Overloading stream Insertion and Extraction Operator

```
class Date
{
    int da,mo,yr;
  public:
    Date(int m=0, int d=0, int y=0) {
        mo = m; da = d; yr = y;
    }
    //insertion operator(<<)
    friend ostream& operator<<(ostream& os, const Date& dt);
    //extraction operator(>>)
    friend istream& operator>>(istream& is,Date&);
};
ostream& operator<<(ostream& os, const Date& dt){
    os << dt.da << '/' << dt.mo << '/' << dt.yr;
    return os;
}
istream& operator>>(istream& is, Date& dt){
    is>>dt.da>>dt.mo>>dt.yr;
    return is;
}
```

```
int main(){
    Date dt1(10, 9, 2023);
    cout << dt1<<endl;
    Date dt2;
    cin>>dt2;
    cout<<dt2;
    return 0;
}
```

Overload Unary Operator

Operator++(pre and post) is a member function(known as operator function) of class A

```
class A {
    int data;
    public:
        A(int x): data(x){}
        A operator++(int){   //post-increment
            A t(data);
            data+=3;
            return t;
        }
        A& operator++(){     //pre-increment
            data+=2;
            return *this;
        }
        void display(){
            cout<<"data="<<data<<endl;
        }
};
```

obj++ changes to call obj.operator++(int)

++obj changes to call obj.operator++()

```
int main(){
        A obj(10);
        obj.display();
        cout<<"pre-increment display"<<endl;
        A obj2=obj++;
        obj2.display();obj.display();
        cout<<"post increment display"<<endl;
        obj2=++obj;
        obj2.display();obj.display();
        return 0;
}
```

Output

```
data=10
post-increment display
data=10
data=13
pre increment display
data=15
data=15
```

Operator++(pre and post) is a non-member friend function of class A

```
class A {
    int data;
    public:
        A(int x): data(x){}
        friend A operator++(A&,int);
        friend A& operator++(A&);
        void display(){
            cout<<"data="<<data<<endl;
        }
};
A operator++(A& obj,int){
    A t(obj.data);
    obj.data+=3;
    return t;
}
A& operator++(A& obj){
    obj.data+=2;
    return obj;
}
```

obj++ changes to call operator++(&obj,int)

++obj changes to call operator++(&obj)

```
int main(){
        A obj(10);
        obj.display();
        cout<<"pre-increment display"<<endl;
        A obj2=obj++;
        obj2.display();obj.display();
        cout<<"post increment display"<<endl;
        obj2=++obj;
        obj2.display();obj.display();
        return 0;
}
```

```
data=10
post-increment display
data=10
data=13
pre increment display
data=15
data=15
```

Rules for operator overloading

- No new operator is defined for overloading like **operator**()** or **operator<>()**
- Intrinsic properties of the overload operator cannot be changes like **arity, precedence, and associativity**.
- To differentiate between unary prefix and postfix operators use
  - For prefix use MyType& operator++(MyType& s1)
  - For postfix use MyType operator++(MyType& s1,int);
- The following operators cannot be overloaded
  - :: (Scope resolution operator)
  - . (member access operator)
  - .* (member access through pointer)
  - ?: (ternary operator)
  - sizeof (sizeof operator)
- For a **member operator function**, invoking object is passed implicitly as the left operand but the right operand is passed explicitly
- For a **non-member operator function(global/friend)** operands are need to passed explicitly.