C++ Overloading Operator

- **C++ overloads the built-in operators for the primitive data types**

  <u>Example:</u>
  ```
   int x,y;
     x=10+5;
     y=10;
  ```
- **Overload class type**

  <u>Example:</u>
  ```
  Employee e1(…);
  Employee e2, e3;
  e3=e1+e2; //compiler error unless you overload + operator for type employee
  ```
  <u>Example:</u>

  Assume A and B are programmer defined classes
  ```
      A a;
      B b;
      a=b;
  ```
- **Overloading operator restrictions**
  - Cannot overload operators as they applied to built-in data-types
    - Can't overload (+) if it is used in the following statement:

      int y=x+5;
  - Must respect the original "functional" template of the operators.
    - Cant convert unary to binary
  - Cannot change operators procedence
    - Can't make + operator a higher precedence
  - C++ allows to overload any operator except class member operator
    - dot operator (.)
    - Resolution operator (::)
    - Conditional operator (?)
    - Class member deference operator
- **Overloading an operator**
  - Create a method to overload an operator
  - Declaration
    - Inside class (public section)
    - Syntax
      - Return type operator op (arguments);
        - operator-key word
        - op: +,-,/,*, ….
  - Definition (implementation)
    - Inside or outside of class
    - Syntax

      Returned type classname::operator op (arguments) {…}

Example:

```
Test s;
cin>>s;
cout<<s;

Class Test
{
        int a,b;
        public:
        Test()
        {
                a=0;
                b=0;
        }
        Test(int x, int y)
        {
                a=x;
                b=y;
        }
        //overload >>
        //friend allows you to accept object outside of class
        friend istream &operator >>(istream &in, Test &o)
        //overload <<
        friend ostream &operator << (ostream &out, Test &o)
};
//Define outside of class
ostream &operator << (ostream &out, test &o)
{
        cout<<"Values of a & b: "<<endl;
        cout<< o.a<<"   "<<o.b<<endl;
        return out;
}

istream &operator >>(istream &in, Test &o)
{
        cout<<"Enter a and b"<<endl;
        cin>>o.a>>o.b;
        return in;
}

//In Main
int main()
{
        Test s;
        cin>>s; //calls overloading
        cout<<s //calls overloaded;
        return 0;
}
```

- **Overloading binary operator with programmer defined class**

```
Class Test
{
        int a,b;
        public:
        Test()
        {
                a=0;
                b=0;
        }
        Test(int x, int y)
        {
                a=x;
                b=y;
        }
        Test operator + (Test &x);
};
Test::Test Operator +(Test &x)
{
        Test result;
        result.a=a+x.a;
        result.b=b+x.b
        return result;
}

Test cc;
Test aa(2,3);
Test bb(7,5);
cc=aa+bb;
```

- **Overloading binary operator with programmer defined class and built in data type**

<u>Problem</u>:

```
        Test s1, s2;
        s2=s1+10;
        s2=10+s1;
        //In the class test
        public:
        friend Test operator + (test &x, int )
        friend Test operator + (int, test &x)
        //Outside of the class test
        Test::operator +(Test &x, int y)
        {
                Test temp;
                temp.a=x.a+y;
                temp.b=x.b+y;
                Return temp
        }
```

```
Test::operator +(int y,Test &x)
{
        Test temp;
        temp.a=x.a+y;
        temp.b=x.b+y;
        return temp;
}
```

- **Overloading Unary Operator**

<u>Problem</u>:

```
Test s2, s1(2,5);
s2=s1++;
//or
s2=++s1;
//In the class Test
Test operator ++(); //prefix
Test operator ++(int); //postfix
//Outside of the class Test
Test Test::operator ++()
{
        Test temp;
        a++; //or ++a;
        b++ //or ++a;
        temp.a=a;
        temp.b=b;
        return temp;
}
Test Test::operator++ (int )
{
        return Test (a++, b++);
}
```

An alternative code for Test::Test operator ++()

- Previous

```
Test::Test operator ++();
{
        Test temp;
        a++; //or ++a;
        b++ //or ++a;
        temp.a=a;
        temp.b=b;
        return temp;
}
```

- Alternative

```
Test::Test operator ++()
{
        ++a;
        ++b;
        return *this;
    }
```
- Note: This is a pointer reference to an operator calling the object function

An alternative code for Test::operator ++(int)

- Previous

```
    Test::Test operator++ (int )
    {
        return  Test(a++, b++);
    }
```
- Alternate

```
    Test::Test operator ++(int)
    {
        Test temp=*this;
        ++a;
        ++b;
        return temp;
    }
```

- **Overloading the relational operators**
  - >,</<=,>=,!=,==

Example:

```
    Class Test
    { …
        bool operator > (Test &);
    };
    bool Test::operator >(Test &o) //if (s1>s2)
    {
        if ((a>o.a)&&(b>o.b))
            return true;
        else
            return false;
    }
```

- **Type conversion**

Conversion	Routine in Destination	Routine in source
Basic to basic (float to int)	Built in	Built in
Basic to class (int to obj)	Constructor	
Class to Basic (obj to int)		Operator function
Class to class (obj to otherObj)	Constructor	Operator function

  - Example: Class to basic and basic to class
    - Metric system vs English system

```
const float MTF=3.280833;
Class Es
{
        int feet;
        int inches;
        public:
                Es(int f, float i)
                {
                        feet=f;
                        inches=i;
                }
                //basic to class
                Es(float m) //m is a metric value
                {
                        float fi=MTF *m;
                        feet=fi;
                        inches=12*(fi-feet)
                }
                //class to basic
                operator float()
                {
                        float ff=inches/12;
                        ff+=feet;
                        return ff/MTF;
}
//In Main
Es e(2,3.0);
float y;
y=e; //class to basic
```

```
          e=y; //basic to class
```

- Example: Class to class - Polar to Cartesian

```
Polar p;
Cartesian c;
p=c;
//or
c=p;

Class Cartesian
{
          double x;
          double y;
          public:
                    Cartesian()
                    {x=0;y=0;}
                    Cartesian(doubly x, double y)
                    {
                              this.x=x;
                              this.y=y;
                    }
                    //added constructor
                    Cartesian(Polar p)
                    {
                              double r=P.getRadius();
                              double a=p.getAngle();
                              x=r*cos(a;)
                              y=r*cos(a);
                    }
};
Class Polar
{
          double radius;
          double angle;
          public:
                    Polar()
                    {
                              radius=0;
                              angle=0;
                    }
                    Polar (double r, double a)
                    {
                              radius=r;
                              angle=a;
                    }
```

```
        operator Cartesian()
        {
            double x=Radius*cos(angle);
            double y=radius*sin(angle);
            return Cartesian(x,y);
        }
};

//In the main
Polar p(10,.5);
Cartesian c;
c=p;
```