OOP class

Overloadable / Non-overloadable Operators: Following is the list of operators which can be overloaded:

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

Following is the list of operators, which can not be overloaded:

::	.*	.	?:

Assignment operators overloading in C++

- Overloading assignment operator (=) same as other operators and it can be used to create an object just like the copy constructor.

```
#include <iostream>
using namespace std;

class Distance
{
  private:
    int feet;             // 0 to infinite
    int inches;           // 0 to 12
  public:
    // required constructors
    Distance(){
      feet = 0;
      inches = 0;
    }
    Distance(int f, int i){
      feet = f;
      inches = i;
    }
    void operator=(const Distance &D )
    {
      feet = D.feet;
      inches = D.inches;
    }
```

```
 // method to display distance
     void displayDistance()
     {
       cout << "F: " << feet <<  " I:" <<  inches << endl;
     }

};
int main()
{
  Distance D1(11, 10), D2(5, 11);

  cout << "First Distance : ";
  D1.displayDistance();
  cout << "Second Distance :";
  D2.displayDistance();

  // use assignment operator
  D1 = D2;
  cout << "First Distance :";
  D1.displayDistance();

  return 0;
}
```

output

First Distance : F: 11 I:10
Second Distance :F: 5 I:11
First Distance :F: 5 I:11

Input/Output operators overloading in C++

- The stream insertion and stream extraction operators also can be overloaded to perform input and output for user-defined types like an object.
- It is important to make operator overloading function a friend of the class because it would be called without creating an object.

```
#include <iostream>
using namespace std;

class Distance
{
  private:
    int feet;             // 0 to infinite
    int inches;           // 0 to 12
  public:
    // required constructors
    Distance(){
      feet = 0;
      inches = 0;
    }
    Distance(int f, int i){
      feet = f;
      inches = i;
    }
```

```
friend ostream &operator<<( ostream &output,
                            const Distance &D )
    {
      output << "F : " << D.feet << " I : " << D.inches;
      return output;
    }

    friend istream &operator>>( istream  &input, Distance &D )
    {
      input >> D.feet >> D.inches;
      return input;
    }
};
```

```
int main()
{
  Distance D1(11, 10), D2(5, 11), D3;

  cout << "Enter the value of object : " << endl;
  cin >> D3;
  cout << "First Distance : " << D1 << endl;
  cout << "Second Distance :" << D2 << endl;
  cout << "Third Distance :" << D3 << endl;


  return 0;
}
```

Out put

Enter the value of object :
70
10
First Distance : F : 11 I : 10
Second Distance :F : 5 I : 11
Third Distance :F : 70 I : 10

Relational operators overloading

- There are various relational operators supported by C++ language like (<, >, <=, >=, ==, etc.) which can be used to compare C++ built-in data types.
- You can overload any of these operators, which can be used to compare the objects of a class.

```
#include <iostream>
using namespace std;

class Distance
{
  private:
    int feet;             // 0 to infinite
    int inches;           // 0 to 12
  public:
    // required constructors
    Distance(){
      feet = 0;
      inches = 0;
    }
    Distance(int f, int i){
      feet = f;
      inches = i;
    }
    // method to display distance
    void displayDistance()
    {
      cout << "F: " << feet << " I:" << inches
<<endl;
    }
```

```
// overloaded minus (-) operator
    Distance operator- ()
    {
      feet = -feet;
      inches = -inches;
      return Distance(feet, inches);
    }
    // overloaded < operator
    bool operator <(const Distance& d)
    {
      if(feet < d.feet)
      {
        return true;
      }
      if(feet == d.feet && inches < d.inches)
      {
        return true;
      }
      return false;
    }
};
```

```
int main()
{
  Distance D1(11, 10), D2(5, 11);

  if( D1 < D2 )
  {
    cout << "D1 is less than D2 " << endl;
  }
  else
  {
    cout << "D2 is less than D1 " << endl;
  }
  return 0;
}
```

Output

- D2 is less than D1