```
// Overload ++ when used as prefix and postfix

#include <iostream.h>
class Count {
   private:
    int value;
   public:
    // Constructor to initialize count to 5
    Count() : value(5) {}
    // Overload ++ when used as prefix
    void operator ++ () {
        ++value;
    }
    // Overload ++ when used as postfix
    void operator ++ (int) {
        value++;
    }
    void display() {
        cout << "Count: " << value << endl;
    }
};

int main() {
    Count count1;
    // Call the "void operator ++ (int)" function
    count1++;
    count1.display();
    // Call the "void operator ++ ()" function
    ++count1;
    count1.display();
    return 0;
}
```

o/p: if - - is used
Count: 4
Count: 3 , if + + is used
Count: 6
Count: 7

The Example 2 works when ++ is used as both prefix and postfix. However, it doesn't work if we try to do something like this:

```
Count count1, result;
// Error
result = ++count1;
```

This is because the return type of our operator function is void. We can solve this problem by making Count as the return type of the operator function.

```
// Return Value from Operator Function (++ Operator)
#include <iostream.h>
class Count {
   private:
    int value;
   public
       :
    // Constructor to initialize count to 5
    Count() : value(5) {}
    // Overload ++ when used as prefix
    Count operator ++ () {
        Count temp;
        // Here, value is the value attribute of the calling object
        temp.value = ++value;
        return temp;
    }

    // Overload ++ when used as postfix
    Count operator ++ (int) {
        Count temp;
        // Here, value is the value attribute of the calling object
        temp.value = value++;
        return temp;
    }

    void display() {
        cout << "Count: " << value << endl;
    }
};

int main() {
    Count count1, result;
    // Call the "Count operator ++ ()" function
    result = ++count1;
    result.display();
    // Call the "Count operator ++ (int)" function
    result = count1++;
    result.display();
    return 0;
}
o/p for , + +
Count: 6
Count: 6 , for - - o/p
Count: 4
Count: 4
```

```
// example on ++
#include <iostream.h>
class Test
{
   private:
      int num;
   public:
       Test(): num(8){}
       void operator ++()         {
          num = num+2;
       }
       void Print() {
           cout<<"The Count is: "<<num;
       }
};
int main()
{
    Test tt;
    ++tt;  // calling of a function "void operator ++()"
    tt.Print();
    return 0;
}
```

o/p 10

```
// =
#include <iostream.h>
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

// ()
#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;
```

```
    }

    // overload function call
    Distance operator()(int a, int b, int c) {
      Distance D;

      // just put random calculation
      D.feet = a + c + 10;
      D.inches = b + c + 100 ;
      return D;
    }

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

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

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

  D2 = D1(10, 10, 10); // invoke operator()
  cout << "Second Distance :";
  D2.displayDistance();

  return 0;
}
output:
First Distance : F: 11 I:10
Second Distance :F: 30 I:120

// <
#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 < 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  ,
if > , D1 is less than D2

//>> ,<<
#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;
}
output:
Enter the value of object :
20
40
First Distance : F : 11 I : 10
Second Distance :F : 5 I : 11
Third Distance :F : 20 I : 40

//[ ]
```

```
#include <iostream.h>
const int SIZE = 10;

class safearay {
  private:
    int arr[SIZE];

  public:
    safearay() {
      register int i;
      for(i = 0; i < SIZE; i++) {
        arr[i] = i;
      }
    }

    int &operator[](int i) {
      if( i > SIZE ) {
        cout << "Index out of bounds" <<endl;
        // return first element.
        return arr[0];
      }

      return arr[i];
    }
};

int main() {
  safearay A;

  cout << "Value of A[2] : " << A[2] <<endl;
  cout << "Value of A[5] : " << A[5]<<endl;
  cout << "Value of A[12] : " << A[12]<<endl;

  return 0;
}
output:
Value of A[2] : 2
Value of A[5] : 5
Value of A[12] : Index out of bounds
0

//
```