The process of defining additional meaning of operator is known as operator overloading. It enables an operator to perform differently depending on the type of operands.

Syntax of operator overloading:

Return type operator op()

{

Function body;

}

#include<iostream.h>

#include<conio.h>

class complex {

int a, b;

public:

void getvalue() {

cout << "Enter the value of Complex Numbers a,b:";

cin >> a>>b;

}

complex operator+(complex ob) {

complex t;

t.a = a + ob.a;

t.b = b + ob.b;

return (t);

}

complex operator-(complex ob) {

complex t;

t.a = a - ob.a;

t.b = b - ob.b;

return (t);

}

void display() {

cout << a << "+" << b << "i" << "\n";

}

};

void main() {

clrscr();

complex obj1, obj2, result, result1;

obj1.getvalue();

obj2.getvalue();

result = obj1 + obj2;

result1 = obj1 - obj2;

cout << "Input Values:\n";

obj1.display();

obj2.display();

cout << "Result:";

result.display();

result1.display();

getch();

}

#include <iostream>

using namespace std;

class Test

{

private:

int count;

public:

Test()

{

Count=5;

}

void operator ++()

{

count = count+1;

}

void Display() { cout<<"Count: "<<count; }

};

int main()

{

Test t;

// this calls "function void operator ++()" function

++t;

t.Display();

return 0;

}