**Friend function**

#include<iostream>

using namespace std;

class B;

class A{

int x;

public:

void setval(int a){

x = a;

}

public:

void getVal(){

cout << " value of A : " << x;

}

friend void addVal(A, B);

};

class B{

int y;

public:

void setval(int b){

y = b;

}

void getVal(){

cout << " value of B : " << y;

}

friend void addVal(A, B);

};

void addVal(A a, B b){

cout << "Sum of two values : " << a.x + b.y;

}

int main(){

A a;

B b;

a.setval(100);

b.setval(200);

a.getVal();

b.getVal();

addVal(a, b);

}

/\*

value of A : 100 value of B : 200Sum of two values : 300

\*/

**Unary operator overloading**

// overloading unary operator

#include<iostream>

using namespace std;

class A{

int a, b, c;

public:

void setValue(int x, int y, int z);

void operator-(); // bcz nothing to return

void getValue();

};

void A :: setValue(int x, int y, int z){

a = x;

b = y;

c = z;

}

void A :: operator-(){

a = -a;

b = -b;

c = -c;

}

void A :: getValue(){

cout << "Value of a : " << a;

cout << " Value of b : " << b;

cout << " Value of c : " << c;

}

int main(){

A a;

a.setValue(10, -25, 30);

a.getValue(); // showing value

cout << " " << endl;

-a; // operator overloading

a.getValue();

return 0;

}

/\*

Value of a : 10 Value of b : -25 Value of c : 30

Value of a : -10 Value of b : 25 Value of c : -30

\*/

**Binary operator overloading**

#include<iostream>

using namespace std;

class A{

int a, b;

public:

A(){ // default parameter

a = 0;

b = 0;

}

A(int x, int y){

a = x;

b = y;

}

A operator+(A a); // bcz we know binary operator overloading demands atleast one object

A operator=(A a);

void getVal();

};

//<return type> <class name> :: <function name><symbol>(<OBJ obj>)

A A :: operator+(A n){

A temp;

temp.a = a + n.a;

temp.b = b + n.b;

return temp;

}

A A :: operator=(A n){

a = n.a;

b = n.b;

return \*this;

}

void A :: getVal(){

cout << " value of a : " << a;

cout << " value of b : " << b;

}

int main(){

A a1, a2, a3;

a1 = A(100, 200);

a2 = A(50, 150);

a3 = a1 + a2;

a1.getVal();

cout << " " << endl;

a2.getVal();

cout << " " << endl;

a3.getVal();

cout << " " << endl;

// assignment operator

cout << "Assignment operator" << endl;

a1 = a2;

a1.getVal();

return 0;

}

/\* value of a : 100 value of b : 200

value of a : 50 value of b : 150

value of a : 150 value of b : 350

Assignment operator

value of a : 50 value of b : 150 \*/

**operator overloading with friend function**

#include<iostream>

using namespace std;

class A{

int a;

int b;

public:

A(){

a = 0;

b = 0;

}

A(int x, int y){

a = x;

b = y;

}

// Friend

friend A operator+(A obj, A obj1);

// General binary overload

A operator-(A obj);

// getter

void getVal();

};

// Friend : we dont need RESOLUTION OPERATOR ( like <A A :: <fun name>> ), bcz a friend function is not a member of a class

A operator+(A obj, A obj1){ // need two parameters to pass both values explicitly

A temp;

temp.a = obj1.a + obj.a;

temp.b = obj1.b + obj.b;

return temp;

}

// General overload

A A :: operator-(A obj){

A temp;

temp.a = a - obj.a;

temp.b = b - obj.b;

return temp;

}

// getter method

void A :: getVal(){

cout << " Value of A : " << a;

cout << " Value of B : " << b;

}

int main(){

A a1, a2, a3, a4;

a1 = A(10, 20);

a2 = A(30, 40);

a1.getVal();

cout<<" " << endl;

a2.getVal();

cout<< " " << endl;

a3 = a1 + a2;

a3.getVal();

cout<< " " << endl;

a4 = a1 - a2;

a4.getVal();

return 0;

}

/\*

Value of A : 10 Value of B : 20

Value of A : 30 Value of B : 40

Value of A : 40 Value of B : 60

Value of A : -20 Value of B : -20\*/

// C++ program to implement class called string, overload +=, +, = operators that appends, concatenation, assignments respectively

#include<iostream>

using namespace std;

class String{

string sen;

public:

String(){

sen = " ";

}

String(string x){

sen = x;

}

String operator+=(String str); // appends

String operator+(String str); // concatenation

String operator=(String str); // assignments

void getString(){

cout << " String is : " << sen << endl;

}

};

String String :: operator+=(String str){

String temp1;

temp1.sen = sen + str.sen;

return temp1;

}

String String :: operator+(String str){

String temp;

temp.sen = sen + str.sen;

return temp;

}

String String :: operator=(String str){

sen = str.sen;

return \*this;

}

int main(){

String a1, a2, a3, a4, a5;

a1 = String("I");

a2 = String("am");

a3 = String("Programmer");

cout << "strings" << endl;

a1.getString();

a2.getString();

a3.getString();

cout << "Appending" << endl;

a4 = a1 += a2 += a3;

a4.getString();

cout << " Concatinating " << endl;

a5 = a1 + a3;

a5.getString();

cout << " Assignings " << endl;

a3 = a1;

a3.getString();

return 0;

}

/\*

strings

String is : I

String is : am

String is : Programmer

Appending

String is : IamProgrammer

Concatinating

String is : IProgrammer

Assignings

String is : I

\*/