Sdffgdgh

Sdfghfgs

Hfgh

Fg

**Increment ++ and Decrement -- Operator Overloading in C++ Programming**

In this example, you'll learn to overload increment ++ and decrement -- operators in C++.

To understand this example, you should have the knowledge of the following [C++ programming](https://www.programiz.com/cpp-programming) topics:

* [C++ Classes and Objects](https://www.programiz.com/cpp-programming/object-class)
* [C++ Constructors](https://www.programiz.com/cpp-programming/constructors)
* [C++ Operator Overloading](https://www.programiz.com/cpp-programming/operator-overloading)

In this tutorial, increment ++ and decrements -- operator are overloaded in best possible way, i.e., increase the value of a data member by 1 if ++ operator operates on an object and decrease value of data member by 1 if -- operator is used.

**Example 1: Prefix ++ Increment Operator Overloading with no return type**

#include <iostream>

using namespace std;

class Check

{

private:

int i;

public:

Check(): i(0) { }

void operator ++()

{ ++i; }

void Display()

{ cout << "i=" << i << endl; }

};

int main()

{

Check obj;

// Displays the value of data member i for object obj

obj.Display();

// Invokes operator function void operator ++( )

++obj;

// Displays the value of data member i for object obj

obj.Display();

return 0;

}

**Output**

i=0

i=1

Initially when the object obj is declared, the value of data member i for object obj is 0 (constructor initializes i to 0).

When ++ operator is operated on obj, operator function void operator++( ) is invoked which increases the value of data member i to 1.

This program is not complete in the sense that, you cannot used code:

obj1 = ++obj;

It is because the return type of operator function in above program is void.

Here is the little modification of above program so that you can use code obj1 = ++obj.

**Example 2: Prefix Increment ++ operator overloading with return type**

#include <iostream>

using namespace std;

class Check

{

private:

int i;

public:

Check(): i(0) { }

// Return type is Check

Check operator ++()

{

Check temp;

++i;

temp.i = i;

return temp;

}

void Display()

{ cout << "i = " << i << endl; }

};

int main()

{

Check obj, obj1;

obj.Display();

obj1.Display();

obj1 = ++obj;

obj.Display();

obj1.Display();

return 0;

}

**Output**

i = 0

i = 0

i = 1

i = 1

This program is similar to the one above.

The only difference is that, the return type of operator function is Check in this case which allows to use both codes ++obj; obj1 = ++obj;. It is because, temp returned from operator function is stored in object obj.

Since, the return type of operator function is Check, you can also assign the value of obj to another object.

Notice that, = (assignment operator) does not need to be overloaded because this operator is already overloaded in C++ library.

**Example 3: Postfix Increment ++ Operator Overloading**

Overloading of increment operator up to this point is only true if it is used in prefix form.

This is the modification of above program to make this work both for prefix form and postfix form.

#include <iostream>

using namespace std;

class Check

{

private:

int i;

public:

Check(): i(0) { }

Check operator ++ ()

{

Check temp;

temp.i = ++i;

return temp;

}

// Notice int inside barcket which indicates postfix increment.

Check operator ++ (int)

{

Check temp;

temp.i = i++;

return temp;

}

void Display()

{ cout << "i = "<< i <<endl; }

};

int main()

{

Check obj, obj1;

obj.Display();

obj1.Display();

// Operator function is called, only then value of obj is assigned to obj1

obj1 = ++obj;

obj.Display();

obj1.Display();

// Assigns value of obj to obj1, only then operator function is called.

obj1 = obj++;

obj.Display();

obj1.Display();

return 0;

}

**Output**

i = 0

i = 0

i = 1

i = 1

i = 2

i = 1

When increment operator is overloaded in prefix form; Check operator ++ () is called but, when increment operator is overloaded in postfix form; Check operator ++ (int) is invoked.

Notice, the int inside bracket. This int gives information to the compiler that it is the postfix version of operator.

Don't confuse this int doesn't indicate integer.

**Example 4: Operator Overloading of Decrement -- Operator**

Decrement operator can be overloaded in similar way as increment operator.

#include <iostream>

using namespace std;

class Check

{

private:

int i;

public:

Check(): i(3) { }

Check operator -- ()

{

Check temp;

temp.i = --i;

return temp;

}

// Notice int inside barcket which indicates postfix decrement.

Check operator -- (int)

{

Check temp;

temp.i = i--;

return temp;

}

void Display()

{ cout << "i = "<< i <<endl; }

};

int main()

{

Check obj, obj1;

obj.Display();

obj1.Display();

// Operator function is called, only then value of obj is assigned to obj1

obj1 = --obj;

obj.Display();

obj1.Display();

// Assigns value of obj to obj1, only then operator function is called.

obj1 = obj--;

obj.Display();

obj1.Display();

return 0;

}

**Output**

i = 3

i = 3

i = 2

i = 2

i = 1

i = 2

Also, unary operators like: !, ~ etc can be overloaded in similar manner.