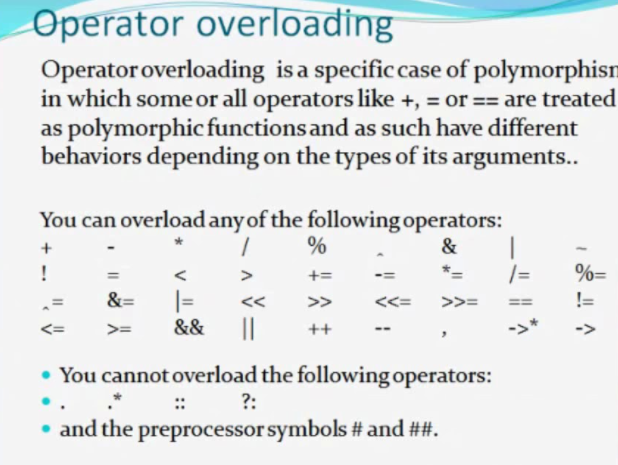
Operator Overloading

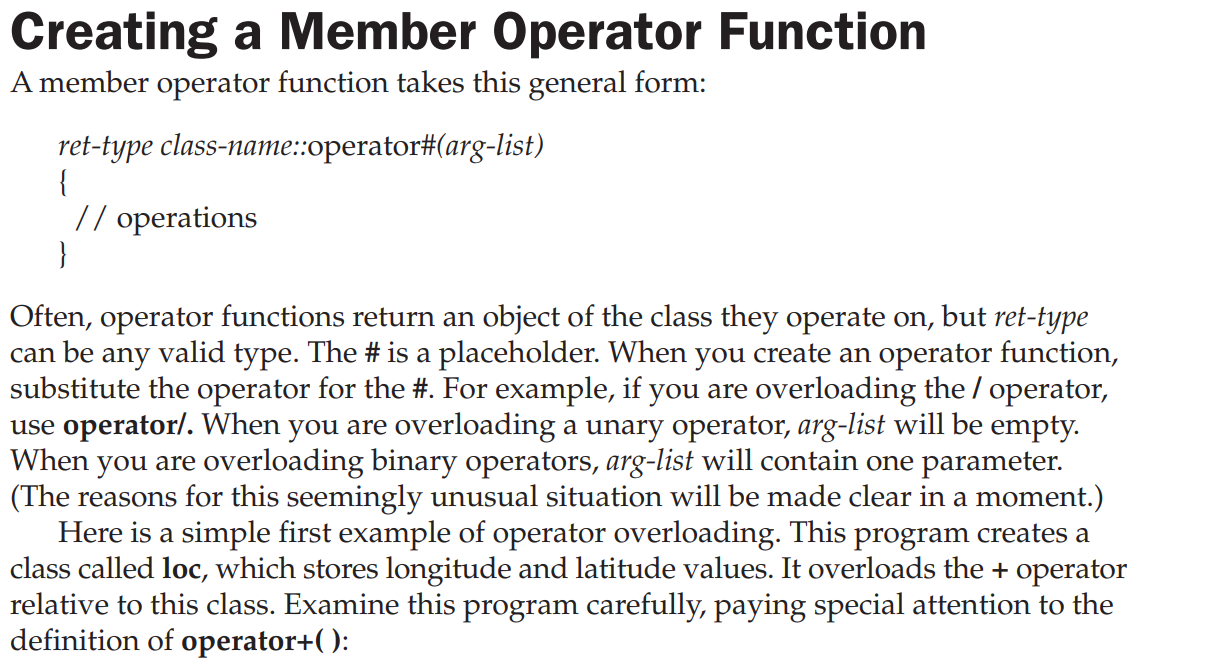
The ability to overload operators is one of C++'s most powerful features. It allows the full integration of new class types into the programming environment.

After overloading the appropriate operators, you can use objects in expressions in just the same way that you use C++'s built-in data types. Operator overloading also forms the basis of C++'s approach to I/O.

You overload operators by creating **operator functions**. An operator function defines the operations that the overloaded operator will perform relative to the class upon which it will work. An operator function is created using the keyword **operator**.

**Operator functions can be either members or nonmembers of a class**. **Nonmember operator functions are almost always friend functions of the class,** however. The way operator functions are written differs between member and nonmember functions. Therefore, each will be examined separately, beginning with member operator functions.





#include <iostream>

using namespace std;

class loc {

int longitude, latitude;

public:

loc() {}

loc(int lg, int lt) {

longitude = lg;

latitude = lt;

}

void show() {

cout << longitude << " ";

cout << latitude << "\n";

}

loc operator+(loc op2); //operator function//

};

// Overload + for loc.

loc loc::operator+(loc op2)

{

loc temp;

temp.longitude = op2.longitude + longitude;

temp.latitude = op2.latitude + latitude;

return temp;

}

int main()

{

loc ob1(10, 20), ob2( 5, 30);

ob1.show(); // displays 10 20

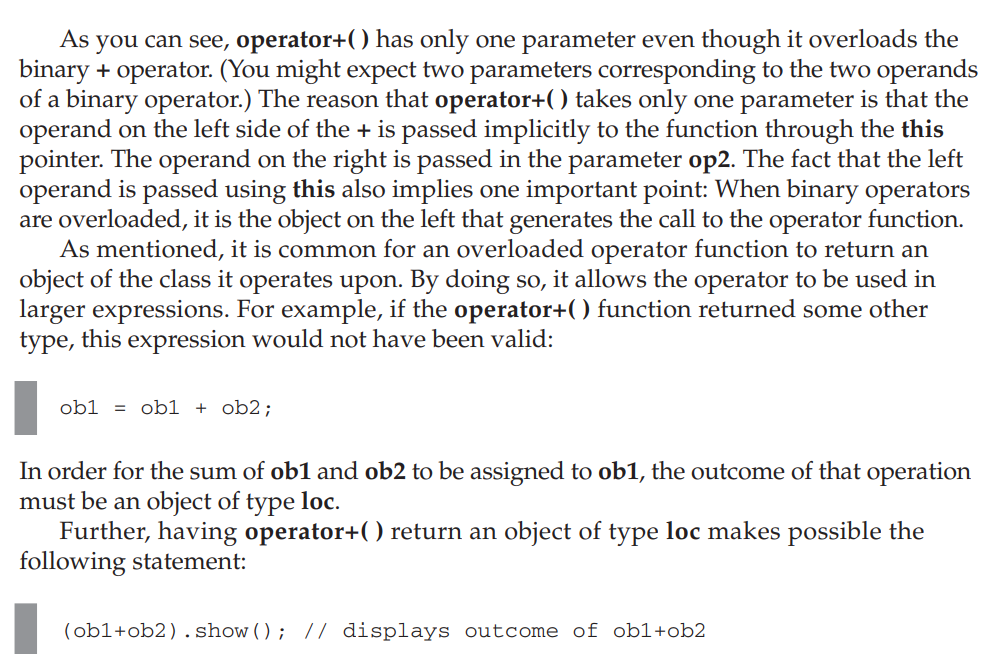
ob2.show(); // displays 5 30

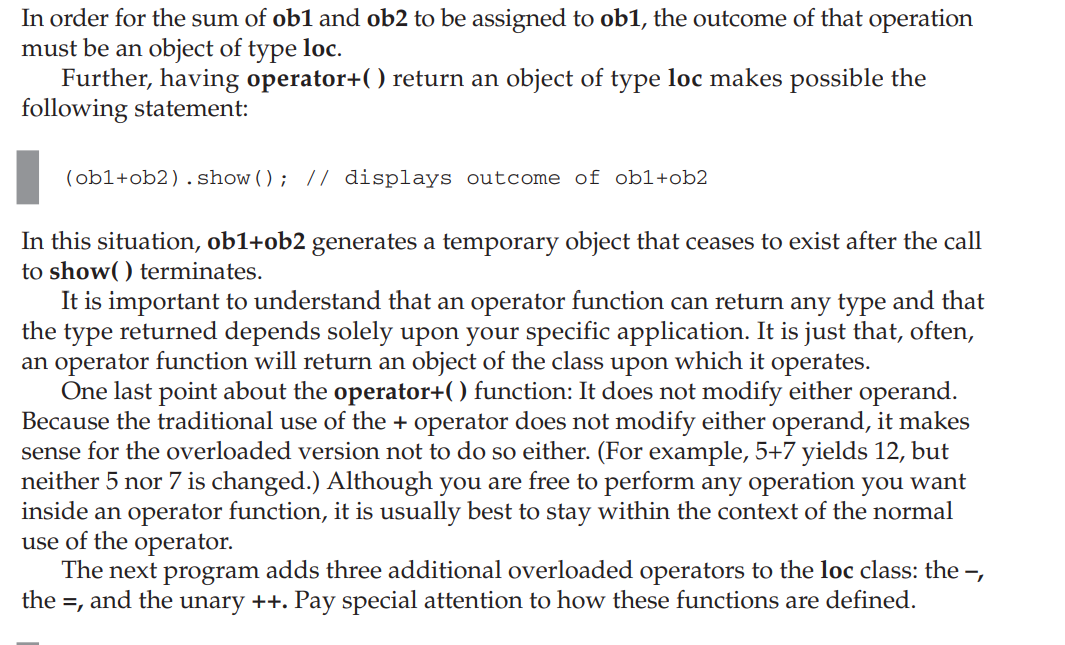
ob1 = ob1 + ob2;

ob1.show(); // displays 15 50

return 0;

}





#include <iostream>

using namespace std;

class loc {

int longitude, latitude;public:

loc() {} // needed to construct temporaries

loc(int lg, int lt) {

longitude = lg;

latitude = lt;

}

void show() {

cout << longitude << " ";

cout << latitude << "\n";

}

loc operator+(loc op2);

loc operator-(loc op2);

loc operator=(loc op2);

loc operator++();

};

// Overload + for loc.

loc loc::operator+(loc op2)

{

loc temp;

temp.longitude = op2.longitude + longitude;

temp.latitude = op2.latitude + latitude;

return temp;

}

// Overload - for loc.

loc loc::operator-(loc op2)

{

loc temp;

// notice order of operands

temp.longitude = longitude - op2.longitude;

temp.latitude = latitude - op2.latitude;

return temp;

}

// Overload asignment for loc.

loc loc::operator=(loc op2)

{

longitude = op2.longitude;

latitude = op2.latitude;

return \*this; // i.e., return object that generated call

}

// Overload prefix ++ for loc.

loc loc::operator++()

{

longitude++;

latitude++;

return \*this;

}

int main()

{

loc ob1(10, 20), ob2( 5, 30), ob3(90, 90);

ob1.show();

ob2.show();

++ob1;

ob1.show(); // displays 11 21

ob2 = ++ob1;

ob1.show(); // displays 12 22

ob2.show(); // displays 12 22

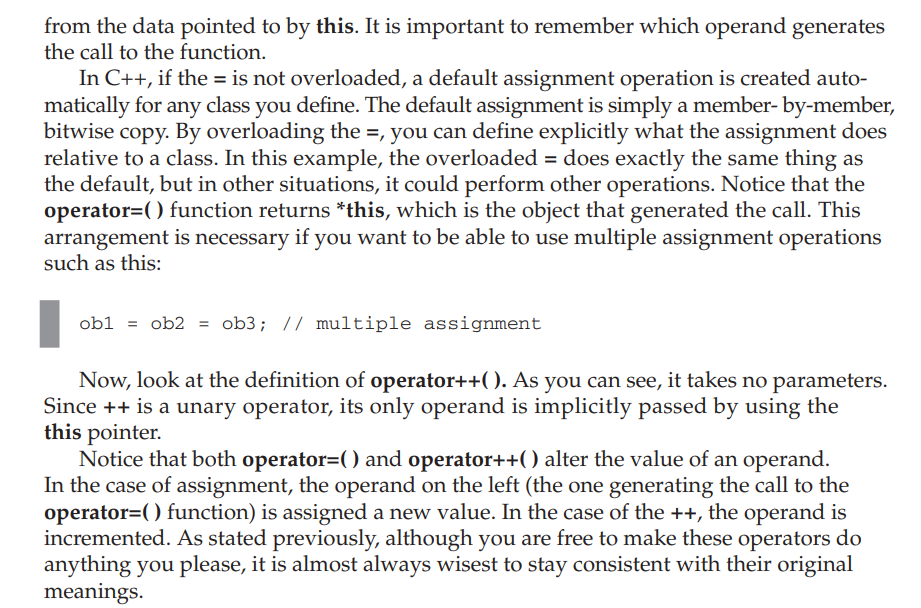
ob1 = ob2 = ob3; // multiple assignment

ob1.show(); // displays 90 90

ob2.show(); // displays 90 90

return 0;

}



Types of Operator Overloading in C++

**Operator Overloading:**

*C++ provides a special function to change the current functionality of some operators within its class which is often called as operator overloading. Operator Overloading is the method by which we can change the function of some specific operators to do some different task.*

This can be done by declaring the function, its syntax is, 

Return\_Type classname :: operator op(Argument list)

{

Function Body

}

In the above syntax Return\_Type is value type to be returned to another object, operator op is the function where the operator is a keyword and op is the operator to be overloaded.

Operator function must be either non-static (member function) or friend function.  
Operator Overloading can be done by using **three approaches**, they are 

1. Overloading unary operator.
2. Overloading binary operator.
3. Overloading binary operator using a friend function.

Below are some criteria/rules to define the operator function: 

* In case of a non-static function, the binary operator should have only one argument and unary should not have an argument.
* In the case of a friend function, the binary operator should have only two argument and unary should have only one argument.
* All the class member object should be public if operator overloading is implemented.
* Operators that cannot be overloaded are **.** **.\*** **::** **?:**
* Operator cannot be used to overload when declaring that function as friend function **=** **()** **[]** **->**.

Refer this, for more rules of Operator Overloading

1. **Overloading Unary Operator**: Let us consider to overload (-) unary operator. In unary operator function, no arguments should be passed. It works only with one class objects. It is a overloading of an operator operating on a single operand.  
   **Example:**   
   Assume that class Distance takes two member object i.e. feet and inches, create a function by which Distance object should decrement the value of feet and inches by 1 (having single operand of Distance Type).

|  |
| --- |
| // C++ program to show unary operator overloading  #include <iostream>    using namespace std;    class Distance {  public:        // Member Object      int feet, inch;        // Constructor to initialize the object's value      Distance(int f, int i)      {          this->feet = f;          this->inch = i;      }        // Overloading(-) operator to perform decrement      // operation of Distance object      void operator-()      {          feet--;          inch--;          cout << "\nFeet & Inches(Decrement): " << feet << "'" << inch;      }  };    // Driver Code  int main()  {      // Declare and Initialize the constructor      Distance d1(8, 9);        // Use (-) unary operator by single operand      -d1;      return 0;  } |

**Output:**

Feet & Inches(Decrement): 7'8

1. In the above program, it shows that no argument is passed and no return\_type value is returned, because unary operator works on a single operand. (-) operator change the functionality to its member function.  
   **Note:** d2 = -d1 will not work, because operator-() does not return any value.
2. **Overloading Binary Operator**: In binary operator overloading function, there should be one argument to be passed. It is overloading of an operator operating on two operands.  
   Let’s take the same example of class Distance, but this time, add two distance objects.

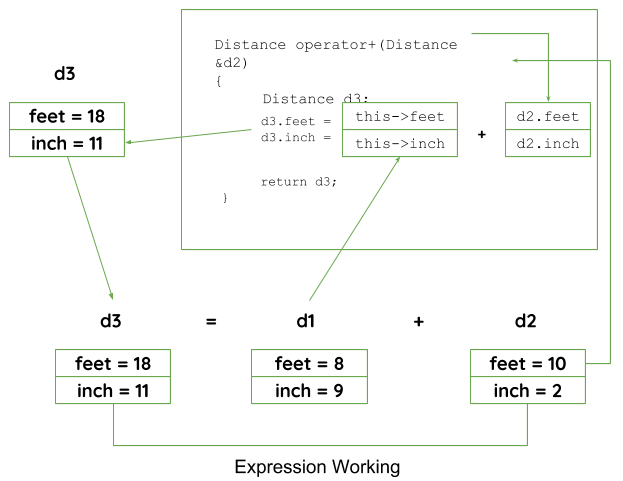
* CPP

|  |
| --- |
| // C++ program to show binary operator overloading  #include <iostream>    using namespace std;    class Distance {  public:      // Member Object      int feet, inch;      // No Parameter Constructor      Distance()      {          this->feet = 0;          this->inch = 0;      }        // Constructor to initialize the object's value      // Parameterized Constructor      Distance(int f, int i)      {          this->feet = f;          this->inch = i;      }        // Overloading (+) operator to perform addition of      // two distance object      Distance operator+(Distance& d2) // Call by reference      {          // Create an object to return          Distance d3;            // Perform addition of feet and inches          d3.feet = this->feet + d2.feet;          d3.inch = this->inch + d2.inch;            // Return the resulting object          return d3;      }  };    // Driver Code  int main()  {      // Declaring and Initializing first object      Distance d1(8, 9);        // Declaring and Initializing second object      Distance d2(10, 2);        // Declaring third object      Distance d3;        // Use overloaded operator      d3 = d1 + d2;        // Display the result      cout << "\nTotal Feet & Inches: " << d3.feet << "'" << d3.inch;      return 0;  } |

**Output:**

Total Feet & Inches: 18'11

1. Here in the above program,   
   *See Line no. 26*, Distance operator+(Distance &d2), here return type of function is distance and it uses call by references to pass an argument.   
   *See Line no. 49*, d3 = d1 + d2; here, d1 calls the operator function of its class object and takes d2 as a parameter, by which operator function return object and the result will reflect in the d3 object.  
   Pictorial View of working of Binary Operator:



* **Overloading Binary Operator using a Friend function**: In this approach, the operator overloading function must precede with friend keyword, and declare a function class scope. Keeping in mind, friend operator function takes two parameters in a binary operator, varies one parameter in a unary operator. All the working and implementation would same as binary operator function except this function will be implemented outside of the class scope.  
  Let’s take the same example using the friend function.

|  |
| --- |
| // C++ program to show binary operator overloading  #include <iostream>    using namespace std;    class Distance {  public:        // Member Object      int feet, inch;        // No Parameter Constructor      Distance()      {          this->feet = 0;          this->inch = 0;      }        // Constructor to initialize the object's value      // Parameterized Constructor      Distance(int f, int i)      {          this->feet = f;          this->inch = i;      }        // Declaring friend function using friend keyword      friend Distance operator+(Distance&, Distance&);  };    // Implementing friend function with two parameters  Distance operator+(Distance& d1, Distance& d2) // Call by reference  {      // Create an object to return      Distance d3;        // Perform addition of feet and inches      d3.feet = d1.feet + d2.feet;      d3.inch = d1.inch + d2.inch;        // Return the resulting object      return d3;  }    // Driver Code  int main()  {      // Declaring and Initializing first object      Distance d1(8, 9);        // Declaring and Initializing second object      Distance d2(10, 2);        // Declaring third object      Distance d3;        // Use overloaded operator      d3 = d1 + d2;        // Display the result      cout << "\nTotal Feet & Inches: " << d3.feet << "'" << d3.inch;      return 0;  } |

**Output:**

Total Feet & Inches: 18'11

1. Here in the above program, operator function is implemented outside of class scope by declaring that function as the friend function.

In these ways, an operator can be overloaded to perform certain tasks by changing the functionality of operators.

Operator Overloading Using a Friend Function

You can overload an operator for a class by using a nonmember function, which is usually a friend of the class. Since a friend function is not a member of the class, it does not have a this pointer. Therefore, an overloaded friend operator function is passed the operands explicitly. This means that a friend function that overloads a binary operator has two parameters, and a friend function that overloads a unary operator has one parameter. When overloading a binary operator using a friend function, the left operand is passed in the first parameter and the right operand is passed in the second parameter. In this program, the operator+( ) function is made into a friend:

#include <iostream>

using namespace std;

class loc {

int longitude, latitude;

public:loc() {} // needed to construct temporaries

loc(int lg, int lt) {

longitude = lg;

latitude = lt;

}

void show() {

cout << longitude << " ";

cout << latitude << "\n";

}

friend loc operator+(loc op1, loc op2); // now a friend

loc operator-(loc op2);

loc operator=(loc op2);

loc operator++();

};

// Now, + is overloaded using friend function.

loc operator+(loc op1, loc op2)

{

loc temp;

temp.longitude = op1.longitude + op2.longitude;

temp.latitude = op1.latitude + op2.latitude;

return temp;

}

// Overload - for loc.

loc loc::operator-(loc op2)

{

loc temp;

// notice order of operands

temp.longitude = longitude - op2.longitude;

temp.latitude = latitude - op2.latitude;

return temp;

}

// Overload assignment for loc.

loc loc::operator=(loc op2)

{

longitude = op2.longitude;

latitude = op2.latitude;

return \*this; // i.e., return object that generated call

}

// Overload ++ for loc.

loc loc::operator++()

{

longitude++;

latitude++;

return \*this;

}

int main()

{

loc ob1(10, 20), ob2( 5, 30);

ob1 = ob1 + ob2;

ob1.show();

return 0;

}