

This course has already ended.

The latest instance of the course can be found at: [Object oriented programming with C++: 2023 Autumn](#)

## 4 Operator Overloading¶

- You can download the template for the programming tasks of the module as a [zip](#) file from [this link](#).

Contents

- 4 Operator Overloading
  - 4.1 Programming tasks

C++ Primer Chapter 14 (Operator overloading)

The standard C (or C++) operators can be redefined through operator overloading. This can be useful with some classes, where e.g. the typical arithmetic operators have an intuitive meaning. It is not advised to do to this in counter-intuitive way, however, because it would only make the program very hard to understand.

Any of the operators can be overloaded, but below we take a simple example of an [euclidean vector](#) which should be easy to understand with a high school background. For an euclidian vector there are well-understood definitions for most operators. Below example implements class *GeomVector* (to distinguish from the vector container), with addition ( [+](#) ) and scalar multiplication ( [\\*](#) ) operators, as well as the [<<](#) operator for output. It would be easy to extend the class to support other operations, such as cross product, magnitude, and so on.

```
1#include <iostream>
2
3class GeomVector {
4public:
5    GeomVector(double x, double y, double z) : x_(x),y_(y),z_(z) { }
6
7    // this is member function of GeomVector
8    GeomVector operator+(const GeomVector& a);
9
10   // these are external functions, but have access to private parts,
11   // because they are declared as 'friend'.
12   friend GeomVector operator*(double a, const GeomVector& b);
13   friend std::ostream& operator<<(std::ostream& out,
14                                   const GeomVector& a);
15
16private:
17    double x_,y_,z_;
18};
19
20GeomVector GeomVector::operator+(const GeomVector& a) {
21    return GeomVector(a.x_ + x_, a.y_ + y_, a.z_ + z_);
22}
23
24GeomVector operator*(double a, const GeomVector& b) {
25    return GeomVector(a * b.x_, a * b.y_, a * b.z_);
26}
27
28std::ostream &operator<<(std::ostream& out, const GeomVector& a) {
29    out << "(" << a.x_ << ", " << a.y_ << ", " << a.z_ << ")";
30    return out;
31}
32
33int main() {
34    GeomVector vecA(2,3,4), vecB(1,1,1), vecC(0.1,0,0);
35    GeomVector vecD(0,0,0), vecE(0,0,0);
36
37    vecD = vecA + vecB; // uses operator+
38    vecE = 5 * vecB; // uses operator*
39
40    // uses operator<<
41    std::cout << "vecA: " << vecA << std::endl
42              << "vecD: " << vecD << std::endl
43              << "vecE: " << vecE << std::endl;
44}
```

*GeomVector* is always initialized with three parameters, for the three dimensions of the vector (this example is fixed to three dimensions, although it would be rather straight forward to generalize it). Initialization is done using an *initializer list*.


Line 8 declares a function for overloading the [+](#) operator. Such operators can be defined in two ways: as *member functions* of the class, in which case binary operators have only one function parameter, and the other part of the binary operator being the object itself (as done with the [+](#) operator), or as global *external functions* in which case binary operators have two function parameters (as done with the [\\*](#) operator). The actual definition of the functions start on line 20. The [+](#) overloading function simply returns a new instance of *GeomVector* that contains a sum of the object itself and the other vector given as argument *a*.

Scalar multiplication is declared on line 12, and the function definition starts from line 24. This variant is not a member of *GeomVector* class, but a global function. However, with the **friend** keyword we give this external function access to the private data of *GeomVector* objects.

This variant of overloading allows mixing different types on both sides of a binary operator. Here we have a *double* multiplier for a *GeomVector* type. As with function overloading, we can have multiple versions of overloading the same operator for different types: we could add support for integer scalar multiplication, or multiplication of two *GeomVectors*, e.g., for dot product.

An useful mechanism is to overload output and input stream operators [<<](#) and [>>](#). This is also done as a global function, as declared on line 13, and definition starting from line 28. When this operator is defined, the objects of *GeomVector* type can be directly used as part of input and output streams. The *main* function shows this in use starting on line 41. General handling of input/output stream objects is discussed later, but for now it is ok to apply this example, if you want to support I/O overloading in your own classes.

The *main* function creates five vectors with given initial values, and then tests the different operators. From the output we see that this apparently works.

Point of interest iconOperator overloading further details


previous | next

### 4.1 Programming tasks¶

Points **10 / 10** My submissions **1**

Deadline Friday, 8 October 2021, 19:59

To be submitted alone

 This course has been archived (Saturday, 17 December 2022, 19:59).

#### Trolls and dragons

**Objective:** Learning to prepare class to support stream output by overloading the [<<](#) operator.

Below are slightly modified implementations of the **Troll** and **Dragon** classes that were shown above. Enhance the classes so that they support the [<<](#) operator for output streams, and the main function following the class definitions compiles.

When Troll name is “Erkki” and hitpoints are 5, the output format should be:

Troll Erkki with 5 HP

When Dragon name is “Mirja” and hitpoints are 35, the output format should be:

Dragon Mirja with 35 HP

When the exercise works properly, it should output

Trolls are: Troll Diiba with 10 HP and Troll Urkki with 15 HP  
Dragons are: Dragon Rhaegal with 50 HP and Dragon Viserion with 55 HP

 **creature.hpp**

Choose File No file chosen

 **creature.cpp**

Choose File No file chosen

 **troll.hpp**

Choose File No file chosen

 **troll.cpp**

Choose File No file chosen

 **dragon.hpp**

Choose File No file chosen

 **dragon.cpp**


Choose File No file chosen

Submit

Points **20 / 20** My submissions **1**

Deadline Friday, 8 October 2021, 19:59

To be submitted alone

 This course has been archived (Saturday, 17 December 2022, 19:59).

#### Overload operators

**Objective:** Learning to overload arithmetic and relational operators.

In this exercise, we continue with the class [GeomVector](#) that was used as an example before. Your task is to create more functionality to the class by overloading the [/](#), [<](#), [>](#), [==](#) and [!=](#) operators. [GeomVector](#) objects are equal if all their components are equal ([x](#), [y](#) and [z](#)). [<](#) and [>](#) should compare the vector by length (use the [Length](#) function).

To access the *GeomVector* itself, you can use [this](#), which is a pointer to the object itself. So, to get the length of the vector being compared to another, you’d do [this->Length\(\)](#).

 **geomvector.hpp**

Choose File No file chosen

 **geomvector.cpp**

Choose File No file chosen

Submit