Difference between operator and function in C++?

I could use some help understanding the following in C++, particularly the difference between an operator and a function:

- What is an operator?
- What is a function?
- What is the difference between them?
- Is a user-defined `operator+()` a *function* or an *operator*?
- Can an operator operate on operands at compile-time? Do they always operate at compile time? (like `sizeof()` in C++)

c++ operators user-defined-functions

edited Jan 16 '11 at 16:47
Chris Smith
9,949 ● 7 ● 46 ● 73

asked Jan 16 '11 at 16:32
Nawaz
226k ● 74 ● 495 ● 704

8 Answers

An *operator* is a symbol like `+`, `-`, `+=` and so forth (see 13.5). They don't carry a meaning. During semantic analysis, the meaning of an operator is determined.

A function is a constructor, destructor, conversion function (that looks like `operator type()`) or operator function (function template specialization and instantiation can yield these in turn).

An *operator function* is something that *implements* an operator (see 13.5). An example is `operator+`. These are functions in all respects, and the only difference to "usual" functions is that they may be called implicitly and they have a funny name.

Some operators have a built-in meaning, that can be changed by the programmer. One refers to the built-in meaning of an operator simply by saying *built-in operator* (see 5/3). However, if such an operator is applied on operands for which a built-in meaning is defined, changing that meaning is only allowed for a few cases (these are assignment, address-of and the comma operator, see 13.5/6).

edited Jan 17 '11 at 11:18

answered Jan 17 '11 at 10:55
Johannes Schaub - litb
373k ● 86 ● 724 ● 1096

> What is an operator?

An operator is a symbol that is use in expressions.
Examples are: + - * / etc

On built-in types there operations are well defined and unchangable.
For user defined types the operators can be defined as syntactic sugar for a function/method call

```
Array a;
a = b + c; // a.operator=(b.operator+(c));
```

> What is a function?

We use the term function/method interchangeably most of the time. The only difference is that a method is associated with an instance of a class object. Otherwise they are the same. They provide a way of grouping a set of instructions together.

> What is the difference between them?

The action of an operator on a built-in type is defined by the compiler.
The action of an operator on a user defined type is a function call.

> Is a user-defined operator+() a function or an operator?

Its a function (or a method). Use of an operator on a user defined type is syntactic sugar for a function call. They are still refereed to as operators though in normal conversation.

> Can an operator operate on operands at compile-time?

For built-in types yes. The compiler has extensive ability to optimize there usage.
For user defined types. It can perform optimizations on the operators just like other functions which may lead to there being eliminated, but the code is not executed at compile time.

> Do they always operate at compile time? (like sizeof() in C++)

No. sizeof() is relatively unique.

Edit:

To show that operator in user defined class behave just like functions here is an example of using mem_fun_ref

```
#include <vector>
#include <algorithm>
#include <memory>
#include <functional>

class X
{
    public:
        // Non standard operators.
        // Because std::mem_fun_ref has a known weakness in that it can
        // not be used with methods that take parameters be reference.
        //
        // The principle is the same though. That the operator+ can be
        // used anywhere that the add() method can be used.

        X& operator+(X* rhs)  { return *this;}
        X& add(X* rhs)        { return *this;}
};

typedef X& (X::*MEMF)(X* rhs);

int main()
{
    MEMF    p1  = &X::add;
    MEMF    p2  = &X::operator+;

    X               value;
    std::vector<X>  data;

    std::for_each(data.begin(),
                  data.end(),
                  std::bind2nd(std::mem_fun_ref(&X::operator+),&value));
}
```

edited Jan 25 '11 at 5:18
LeopardSkinPillBoxHat
19.7k ●8 ●58 ●104

answered Jan 16 '11 at 18:11
Loki Astari
175k ●52 ●248 ●441

+1 for defining operator much better than I did. – Billy ONeal Jan 16 '11 at 18:15

There is no meaningful difference between operators and functions, except that operators have a different syntax. Primitive operators however are not functions.

answered Jan 16 '11 at 16:44
Puppy
117k ●22 ●181 ●375

> What is operator?

An operator is generally an operation performed on a variable given some form of punctuation. For example, the default behavior of `operator+` between two integers is to add them.

> What is function?

A function is a subroutine -- a reuseable block of code.

> What is the difference between them?

Nothing, as far as user code is concerned, except for syntax. Note that if you override `operator||`, `operator&&`, or (to a lesser extent) `operator,`, you change the semantics of the built in operator semantics. In the case of `&&` and `||`, you make the operation which is normally short circuiting into an operation which is not. In the case of the comma, you would need to ensure that you evaluate the arguments left to right, as the comma operator normally behaves in this way.

> Is user-defined operator+() a function or operator?

Neither. It is a user defined operator overload. A function name cannot start with the keyword `operator`, and an operator is simply the actual punctuation mark used to invoke the operator overload, i.e. `+` or `-`. **EDIT:** Note that while technically speaking it is not a function, it does have the semantics of a function call, as demonstrated in @Martin York's excellent answer.

> Can operator operate on operands at compile-time? Do they always operate at compile time? (like sizeof() in C++)

No, `sizeof` cannot be overloaded. If you want some form of compile time operation done, you need to use something like template metaprogramming. Note that if the compiler is able to do the calculation at compile time it may elide the call into your overloaded operator, of course.

edited May 23 at 12:17
Community ♦
1 ● 1

answered Jan 16 '11 at 16:55
Billy ONeal
67k ● 29 ● 231 ● 458

I disagree with your definition `Is user-defined operator+() a function or operator?`. The overloaded operator for a user class is a function/method. The use of the operator is just syntactic sugar for a function/method call. a+b is actually: a.operator+(b). The method `operator+` behaves just the same as any other method in a class and you can even take its address just like any other method (like add()). – Loki Astari Jan 16 '11 at 18:02

@Martin York: It does have the semantics of a function call, but that does not mean it is a function. You can't call an operator overload with `std::mem_fun_ref`, for example. – Billy ONeal Jan 16 '11 at 18:14

Potentially you can. The problem is not with the overloaded operator but the fact that mem_fun_ref can not take methods that have a reference parameter (well knowing failing of mem_fun_ref). You can take the address of X::operator+() just like any other method and use it just like you could another method. – Loki Astari Jan 16 '11 at 18:38

See my updated answer. – Loki Astari Jan 16 '11 at 18:42

@Martin: I was not aware you could call it like that. – Billy ONeal Jan 16 '11 at 18:47

There is no huge difference between functions and operators. You can think of an using operator, e.g., 'a+b', as a shortcut to the function operator+(a,b) which is defined for the types of a and b. Of course, operators on primitive types (like integers) and a few other exceptions are not necessarily defined like this.

Thus, to answer a few of your specific questions:

Is user-defined operator+() a function or operator?

A function that implements an operator.

Can operator operate on operands at compile-time? Do they always operate at compile time?

Since it is a function, it operates at run time, but in some cases compiler optimizations can do work at compile time for certain operators. I'm not 100% sure why you're asking this, so perhaps there is something I'm not aware of here.

answered Jan 16 '11 at 16:51
RandomGuy
753 ● 3 ● 10 ● 21

In C++ you can override what the symbols +, -, ==, etc. do when applied to class instances. By defining the "operator+" method in class A, you are telling the compiler what to do with code like:

```
A a, b, c;
c = a + b; // the + here actually calls a.operator+(b)
```

It's also a function or more precisely an instance method, in the sense that it's something that gets called.

EDIT: see also http://en.wikipedia.org/wiki/Operator_overloading and http://en.wikibooks.org/wiki/C++_Programming/Operators/Operator_Overloading

answered Jan 16 '11 at 16:36
sinelaw
11.7k ● 1 ● 31 ● 64

There are only two minor differences between functions and operators.

1. Operators *can* be used in two ways (`x+y` or `operator+(a,b)`).
2. Operators *must* have same number of parameters as the built-in one (`operator==` must have exactly two params). The exception from this rule is function call `operator()` which can be overloaded with any number of any parameters.

answered Jan 16 '11 at 18:09

Xirdus
1,089 ● 3 ● 16 ● 33

Here are some differences between an operator and a function:

1. An operator does not push its parameters onto the stack, but a function pushes its parameters onto the stack.
2. The compiler knows about the operation of the operators, but is not aware of the output of the function. Said a different way, the action of the operator is defined at compilation time and that of a function is defined at the runtime.

edited May 17 '12 at 13:36

Mat
149k ● 25 ● 272 ● 307

answered May 17 '12 at 4:23

Manish Kumar
1 ● 1

Both your points are wrong in general with C++ - operator overloading comes into play. – Mat May 17 '12 at 13:37