11.1 — Introduction to function overloading

ALEX[1] DECEMBER 28, 2023

Consider the following function:

```
int add(int x, int y)
{
    return x + y;
}
```

This trivial function adds two integers and returns an integer result. However, what if we also want a function that can add two floating point numbers? This `add()` function is not suitable, as any floating point parameters would be converted to integers, causing the floating point arguments to lose their fractional values.

One way to work around this issue is to define multiple functions with slightly different names:

```
int addInteger(int x, int y)
{
    return x + y;
}

double addDouble(double x, double y)
{
    return x + y;
}
```

However, for best effect, this requires that you define a consistent function naming standard for similar functions that have parameters of different types, remember the names of these functions, and actually call the correct one.

And then what happens when we want to have a similar function that adds 3 integers instead of 2? Managing unique names for each function quickly becomes burdensome.

Introduction to function overloading

Fortunately, C++ has an elegant solution to handle such cases. **Function overloading** allows us to create multiple functions with the same name, so long as each identically named function has different parameter types (or the functions can be otherwise differentiated). Each function sharing a name (in the same scope) is called an **overloaded function** (sometimes called an **overload** for short).

To overload our `add()` function, we can simply declare another `add()` function that takes double parameters:

```
double add(double x, double y)
{
    return x + y;
}
```

We now have two versions of `add()` in the same scope:

```
int add(int x, int y) // integer version
{
    return x + y;
}

double add(double x, double y) // floating point version
{
    return x + y;
}

int main()
{
    return 0;
}
```

The above program will compile. Although you might expect these functions to result in a naming conflict, that is not the case here. Because the parameter types of these functions differ, the compiler is able to differentiate these functions, and will treat them as separate functions that just happen to share a name.

Key insight

Functions can be overloaded so long as each overloaded function can be differentiated by the compiler. If an overloaded function can not be differentiated, a compile error will result.

Related content

Operators can also be overloaded in a similar manner. We'll discuss operator overloading in 21.1 -- Introduction to operator overloading (https://www.learncpp.com/cpp-tutorial/introduction-to-operator-overloading/)[2].

Introduction to overload resolution

Additionally, when a function call is made to a function that has been overloaded, the compiler will try to match the function call to the appropriate overload based on the arguments used in the function call. This is called **overload resolution**.

Here's a simple example demonstrating this:

```cpp
#include <iostream>

int add(int x, int y)
{
    return x + y;
}

double add(double x, double y)
{
    return x + y;
}

int main()
{
    std::cout << add(1, 2); // calls add(int, int)
    std::cout << '\n';
    std::cout << add(1.2, 3.4); // calls add(double, double)

    return 0;
}
```

The above program compiles and produces the result:

```
3
4.6
```

When we provide integer arguments in the call to `add(1, 2)`, the compiler will determine that we're trying to call `add(int, int)`. And when we provide floating point arguments in the call to `add(1.2, 3.4)`, the compiler will determine that we're trying to call `add(double, double)`.

Making it compile

In order for a program using overloaded functions to compile, two things have to be true:

1. Each overloaded function has to be differentiated from the others. We discuss how functions can be differentiated in lesson 11.2 -- Function overload differentiation[3].
2. Each call to an overloaded function has to resolve to an overloaded function. We discuss how the compiler matches function calls to overloaded functions in lesson 11.3 -- Function overload resolution and ambiguous matches[4].

If an overloaded function is not differentiated, or if a function call to an overloaded function can not be resolved to an overloaded function, then a compile error will result.

In the next lesson, we'll explore how overloaded functions can be differentiated from each other. Then, in the following lesson, we'll explore how the compiler resolves function calls to overloaded functions.

Conclusion

Function overloading provides a great way to reduce the complexity of your program by reducing the number of function names you need to remember. It can and should be used liberally.

Best practice

Use function overloading to make your program simpler.

Next lesson
11.2 Function overload differentiation

3

Back to table of contents

5

Previous lesson
10.x Chapter 10 summary and quiz

6

7

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[10] are connected to your provided email address.

186 COMMENTS

Newest

PooperShooter
December 27, 2024 4:24 pm PST

How badly does overloading a function impact performance whether it be compile time or what have u? Id imagine it would hinder performance a bit if the compiler has to determine which type of function to use either int or double in this case

1 Reply

Alex Author

Reply to PooperShooter[11] January 3, 2025 5:27 pm PST

It slows your compile time by a trivial amount, and has no runtime impact. It's not something to worry about.

12 Reply

Sadaam Rooble Awil
November 4, 2024 1:31 pm PST

Good well

0 Reply

r.kh
June 19, 2024 3:18 pm PDT

Hi
Should I learn all chapters to start learning the oop programming?

0 Reply

raj
Reply to r.kh [12] July 19, 2024 10:49 pm PDT

yes offcourse you have to clear the basic.

0 Reply

Alex Author
Reply to r.kh [12] June 25, 2024 1:44 pm PDT

The chapters are designed to be read sequentially, so yes.

8 Reply

deCppNovice
January 17, 2024 11:46 pm PST

3 chapters or 36 more lessons before OOP.
Excitement is an understatement!
When i get to OOP i will need learning buddies of 5 people. We can talk about learncpp lessons and solve lots of problems

10 Reply

Mello
Reply to deCppNovice [13] June 29, 2024 12:52 am PDT

Hey! If a spot is still open i want to join in too.
I will send my discord once you confirm

Last edited 11 months ago by Mello

0 Reply

groovyest

Reply to deCppNovice [13] May 18, 2024 7:28 am PDT

wanna count me in?

0 Reply

deCppNovice

Reply to groovyest [14] May 18, 2024 7:38 am PDT

Add me as a friend on Discord! Invite expires in 1 week: https://discord.com/invite/zjWJvZWq

0 Reply

Raskolnikov

Reply to deCppNovice [15] October 27, 2024 4:48 am PDT

yo can you send new link?

0 Reply

shashwatrd123@gmail.com

Reply to deCppNovice [15] June 26, 2024 11:12 am PDT

I am neewbie,can i also join you..I wish we could learn together?

0 Reply

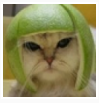
Krishnakumar

June 29, 2023 2:41 pm PDT

>Because operators in C++ are just functions

Whaaat?!! Didn't know this!

0 Reply



Alex Author

Reply to Krishnakumar [16] July 1, 2023 2:12 pm PDT

This is a misstatement on my part. The implementation of built-in operators is an implementation-specific detail (though they effectively behave like functions for the most part, but there are a few minor differences, like you can not call them explicitly by name). Overloaded operators are functions. I corrected the text.

3 Reply

Bryan
February 10, 2023 10:50 pm PST

Maybe it should be made clear that function/operator overloading and templating are part of the compile-time polymorphism concept

0 Reply

Alex Author
Reply to Bryan [17] February 11, 2023 8:02 pm PST

I'm hesitant to introduce the term "polymorphism" here. Although static polymorphism is a definitely a thing, the term "polymorphism" in C++ generally refers to runtime polymorphism. If we say a class is polymorphic, we mean it has at least one virtual function, and thus is eligible for virtual function resolution. Is there some specific reason you think this is important nomenclature to mention here?

I think it would be better to talk about this in the introduction to virtual functions, since at that point there's value in making the distinction between compile time vs runtime resolution. Here, it's just extra vocab.

11 Reply

Krishnakumar
Reply to Alex [18] June 29, 2023 2:46 pm PDT

Wow. So much to learn!

1 Reply

XnneHang
November 14, 2022 10:40 pm PST

It's the first time that you say that something can be use and should be use.about function overloading

3 Reply

Alex Author
Reply to XnneHang [19] November 18, 2022 8:59 pm PST

What do you mean?

1 Reply

yellowEmu
Reply to Alex [20] January 5, 2023 6:24 am PST

I think they are referring to the C++ course symptom of "hey, look at this feature! but never use it"

26 Reply

NordicPeace

Reply to yellowEmu [21] December 9, 2024 8:44 pm PST

yup

0 Reply

David Brixton

August 28, 2022 12:42 am PDT

I am missing out a major thing. How would I avoid duplicate code while doing function overloading? Like suppose the function in question is a 100 line function and I have no desire to break it down in parts.

Last edited 2 years ago by David Brixton

0 Reply

Alex Author

Reply to David Brixton [22] September 7, 2022 10:26 am PDT

Templates is one way. Another way is to make one overloaded function call another overloaded function.

If you don't want to break things down into parts, then you'll probably end up with more duplicate code than you would otherwise.

1 Reply

Eldinur

Reply to David Brixton [22] August 31, 2022 1:06 pm PDT

You could just use templates.. which he covers later! But for simple functions I'd say function overloading might really be the most practical choice!

0 Reply

Arthur See Clarke

August 23, 2022 1:14 pm PDT

I vote for overloaded functions to be called Schrödinger's Functions.

13 Reply

Valhalla

August 23, 2022 7:06 am PDT

So two functions with same name but no parameters cant be overloaded ?
something like this:-

```
double add();
int add();
```

1 Reply

Kevin

Reply to Valhalla [23] July 13, 2024 8:39 pm PDT

In C++, function overloading is based on the function signature, which includes the function name and the parameter types, but not the return type. Therefore, two functions with the same name and no parameters cannot be overloaded based solely on their return type.

0 Reply

Alex Author

Reply to Valhalla [23] August 25, 2022 7:16 pm PDT

Correct. These functions only differ by return type and return type isn't a valid differentiator for overloading.

2 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/introduction-to-operator-overloading/
3. https://www.learncpp.com/cpp-tutorial/function-overload-differentiation/
4. https://www.learncpp.com/cpp-tutorial/function-overload-resolution-and-ambiguous-matches/
5. https://www.learncpp.com/
6. https://www.learncpp.com/cpp-tutorial/chapter-10-summary-and-quiz/
7. https://www.learncpp.com/introduction-to-function-overloading/
8. https://www.learncpp.com/cpp-tutorial/inline-functions-and-variables/
9. https://www.learncpp.com/cpp-tutorial/default-arguments/
10. https://gravatar.com/
11. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-605795
12. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-598597
13. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-592511
14. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-597237
15. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-597240
16. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-582857
17. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-577345
18. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-577386
19. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-574687
20. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-574758

21. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-575862
22. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-572511
23. https://www.learncpp.com/cpp-tutorial/introduction-to-function-overloading/#comment-572333
24. https://g.ezoic.net/privacy/learncpp.com