2.6 — Why functions are useful, and how to use them effectively

ALEX[1] APRIL 15, 2022

Now that we've covered what functions are and some of their basic capabilities, let's take a closer look at why they're useful.

Why use functions?

New programmers often ask, "Can't we just put all the code inside the *main* function?" For simple programs, you absolutely can. However, functions provide a number of benefits that make them extremely useful in programs of non-trivial length or complexity.

- Organization -- As programs grow in complexity, having all the code live inside the main() function becomes increasingly complicated. A function is almost like a mini-program that we can write separately from the main program, without having to think about the rest of the program while we write it. This allows us to reduce a complicated program into smaller, more manageable chunks, which reduces the overall complexity of our program.
- Reusability -- Once a function is written, it can be called multiple times from within the program. This avoids duplicated code ("Don't Repeat Yourself") and minimizes the probability of copy/paste errors. Functions can also be shared with other programs, reducing the amount of code that has to be written from scratch (and retested) each time.
- Testing -- Because functions reduce code redundancy, there's less code to test in the first place. Also because functions are self-contained, once we've tested a function to ensure it works, we don't need to test it again unless we change it. This reduces the amount of code we have to test at one time, making it much easier to find bugs (or avoid them in the first place).
- Extensibility -- When we need to extend our program to handle a case it didn't handle before, functions allow us to make the change in one place and have that change take effect every time the function is called.
- Abstraction -- In order to use a function, you only need to know its name, inputs, outputs, and where it lives. You don't need to know how it works, or what other code it's dependent upon to use it. This lowers the amount of knowledge required to use other people's code (including everything in the standard library).

Although it doesn't look like it, every time you use operator<< or operator>> to do input or output, you're using a function provided by the standard library that meets all of the above criteria.

Effectively using functions

One of the biggest challenges new programmers encounter (besides learning the language) is understanding when and how to use functions effectively. Here are a few basic guidelines for writing functions:

- Groups of statements that appear more than once in a program should generally be made into a function. For example, if we're reading input from the user multiple times in the same way, that's a great candidate for a function. If we output something in the same way in multiple places, that's also a great candidate for a function.
- Code that has a well-defined set of inputs and outputs is a good candidate for a function, (particularly if it is complicated). For example, if we have a list of items that we want to sort, the code to do the sorting would make a great function, even if it's only done once. The input is the unsorted list, and the output is the sorted list. Another good prospective function would be code that simulates the roll of a 6-sided dice. Your current program might only use that in one place, but if you turn it into a function, it's ready to be reused if you later extend your program or in a future program.
- A function should generally perform one (and only one) task.
- When a function becomes too long, too complicated, or hard to understand, it can be split into multiple sub-functions. This is called **refactoring**. We talk more about refactoring in lesson 3.10 -- Finding issues before they become problems[2].

Typically, when learning C++, you will write a lot of programs that involve 3 subtasks:

1. Reading inputs from the user
2. Calculating a value from the inputs
3. Printing the calculated value

For trivial programs (e.g. less than 20 lines of code), some or all of these can be done in function *main*. However, for longer programs (or just for practice) each of these is a good candidate for an individual function.

New programmers often combine calculating a value and printing the calculated value into a single function. However, this violates the "one task" rule of thumb for functions. A function that calculates a value should return the value to the caller and let the caller decide what to do with the calculated value (such as call another function to print the value).

Next lesson
2.7 Forward declarations and definitions

3

Back to table of contents

4

Previous lesson
2.5 Introduction to local scope

5

6

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

Leave a comment...

Name*

Email*

Find a mistake? Leave a comment!

Avatars from https://gravatar.com/[9] are connected to your provided email address.

Notify me about replies:

POST COMMENT

80 COMMENTS Newest

South
April 26, 2022 1:28 am

Not sure if the way I think isn't normal but functions I think are the best way to go. A dice roll function can be used by a game to calculate random damage to the player or mob but can also be used in any other program where a random number is desired.

I see functions as modules to be plugged in as needed and accessed. I see coding in general as a group of modules working together through linking code in the main to produce a result.

6 Reply

> **Alex** Author
>
> Reply to South April 26, 2022 9:37 am
>
> Exactly. If you're smart about how you design your "modules" they will be reusable across many different programs. This is maximized by separating the logic of your program (which is specific to your program) from the reusuable components (functions and types that can be used in any program).
>
> 10 Reply

KKW

December 24, 2021 6:45 am

so like the brawl star figures, they were made using functions and if else so they will attack when the attack button is pressed, right? and the COC troops keep attacking and don't attack when walking is also because of the use of functions and if else, right?

3 Reply

> **Ankesh**
>
> Reply to KKW August 29, 2022 3:52 am
>
> yeah you are right
>
> 0 Reply

> **MWAR**
>
> Reply to KKW March 3, 2022 7:54 pm
>
> Not sure what you're referring to. But keep it up! Learning by relating to things you already know is one of the best ways to retain information! I use the ADEPT method : https://betterexplained.com/articles/adept-method/
>
> 2 Reply

Jean Peter T. Paredes
August 26, 2021 6:23 pm

"Statements that appear more than once in a program should generally be made into a function. For example, if we're reading input from the user multiple times in the same way, that's a great candidate for a function. If we output something in the same way multiple times, that's also a great candidate for a function."

Would like to add that

Statements that appeared once BUT has a high probability of being used multiple times in the future should generally be made into a function
just like the get user input from our previous lessons

10 Reply

Nitin
August 22, 2021 9:27 pm

isn't it okay to use "using namespace std;" than using every time std: with in-build functions.

1 Reply

Alex Author
Reply to Nitin August 25, 2021 12:44 pm

This question is addressed in lesson https://www.learncpp.com/cpp-tutorial/using-declarations-and-using-directives/

9 Reply

Ni3
Reply to Alex August 26, 2021 1:38 am

Okay thank you sir

4 Reply

lil pump
February 20, 2021 7:36 am

[C++]
#include iostream

```
int main(){
  std::cout << "Hello world" << std::endl;
}
```

[/C++]

0 Reply

seaque

Reply to lil pump June 1, 2021 6:40 am

the correct use is

```
#include iostream

int main(){
   std::cout << "Hello world" << std::endl;
}
```

0 Reply

Vinorosso

Reply to seaque June 21, 2021 2:03 am

The code should be like this tho:

```
#include <iostream>

int main()
{
    std::cout << "Hello world\n";

    return 0;
}
```

0 Reply

seaque

Reply to seaque June 1, 2021 6:58 am

sorry, i meant (code) (/code)

change the parenthesis with []

1 Reply

RHOULAN DHAMAR WANTO

November 21, 2020 11:12 pm

please help Alex
please help me with this

(Equilateral Triangle validation and perimeter) Implement the following two functions:
// Returns true if all the sides of the triangle
// are same.
bool isValid(double side1, double side2, double side3)
// Returns the perimeter of an equilateral triangle.
double perimeter(double side1)
The formula for computing the perimeter is perimeter = 3 * side. Write a test program that reads three sides for a triangle and computes the perimeter if the input is valid. Otherwise, display that the input is invalid.

0 Reply

Anon

Reply to RHOULAN DHAMAR WANTO December 17, 2020 10:56 am

```
#include <iostream>

bool isValid(double side1, double side2, double side3)
{
    //if side1 == side2 then they have the same length, and if side2 and side 3 have the same length,
    //it follows that side1 == side2 == side3 and therefore it is equilateral
    return (side1 == side2) && (side2 == side3);
}

double perimeter(double side1)
{
    return side1 * 3;
}

int main()
{
    //you could refactor a way to take the input for all 3 sides as a function
```

```
    std::cout << "Please enter side1 of your triangle:";
    double side1{};
    std::cin >> side1;
    std::cout << "Please enter side2 of your triangle:";
    double side2{};
    std::cin >> side2;
    std::cout << "Please enter side3 of your triangle:";
    double side3{};
    std::cin >> side3;
    if (isValid(side1, side2, side3))
    {
        std::cout << "The perimeter of an equilateral triangle with side length " << side1 << " is " << perimeter(side1) << ".";
    } else
    {
        std::cout << "The triangle is not equilateral.";
    }
    return 0;
}
```

2 Reply

John

November 10, 2020 6:13 am

for practice purposes:

```cpp
#include <iostream>

int get_value()
{
    int v{};

    std::cout << "Enter a Number: ";
    std::cin >> v;

    return v;
}

char get_operator()
{
    char o{};

    std::cout << "Enter operator: ";
    std::cin >> o;

    return o;
}

int main()
{
    int a{ get_value() };
    char op{ get_operator() };
    int b{ get_value() };

    switch (op)
    {
        case '*':
            std::cout << a * b << "\n";
            break;
        case '/':
            std::cout << a / b << "\n";
            break;
        case '+':
            std::cout << a + b << "\n";
            break;
        case '-':
            std::cout << a - b << "\n";
            break;
    }

    return 0;
}
```

4 Reply

> **lettuceMan**
>
> Reply to John February 14, 2022 7:37 am
>
> nice
>
> 2 Reply

Mathari

May 9, 2020 9:00 pm

Hello
Are there any tutorials or exercises accompanying each lesson?

0 Reply

> ### José
>
> Reply to Mathari May 19, 2020 6:46 am
>
> If you search on the internet, maybe you'll find something
>
> 0 Reply

> ### nascardriver Sub-admin
>
> Reply to Mathari May 10, 2020 3:23 am
>
> Not for each lesson. There's a quiz every now and then.
>
> 0 Reply

you’re using a function provided by the standard library

May 5, 2020 8:43 am

```
1 "Although it doesn’t look like it, every time you use operator<< or operator>> to
  do input or output, you’re using a function provided by the standard library that
  meets all of the above criteria."
```

I found this part very interesting. Can we reach in this tutorial to a point where we can start writing user-defined functions with such characteristic you mentioned in the quote?
I mean I hadn't had any idea std::cout, could be a function!

0 Reply

> ### nascardriver Sub-admin
>
> Reply to you’re using a function provided by the standard library May 6, 2020 6:40 am
>
> `std::cout` is a variable, the function is `<<`. We show how to write those functions in the chapter about operator overloading.
>
> 1 Reply

1 Reply

ColdCoffee

April 7, 2020 11:02 am

"A function should generally perform one (and only one) task."
If function always performs one and only one task then why do you use "generally" ?

0 Reply

Gabe

Reply to ColdCoffee April 9, 2020 1:32 pm

The idea is to use a function that only solves a single computational. No more. If you have a function that not only computes the sum of two parameters passed as arguments, but also prints out the result in the same function, that only shows that you can separate the two as their own functions. Functions should be short and sweet and binds to the name of the function.

1 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/finding-issues-before-they-become-problems/
3. https://www.learncpp.com/cpp-tutorial/forward-declarations/
4. https://www.learncpp.com/
5. https://www.learncpp.com/cpp-tutorial/introduction-to-local-scope/
6. https://www.learncpp.com/why-functions-are-useful-and-how-to-use-them-effectively/
7. https://www.learncpp.com/cpp-tutorial/for-each-loops/
8. https://www.learncpp.com/cpp-tutorial/null-pointers/
9. https://gravatar.com/
10. https://www.ezoic.com/what-is-ezoic/