Passing by reference vs. value, and CONST

Comp Sci 1570 Introduction to C++

Missouri S&T | Computer Science

Outline

Outline

Benefits of functions

- **Organization** – As programs grow in complexity, having all the code live inside the main() function becomes increasingly complicated. Divide complicated tasks into smaller, simpler ones, and drastically reduce 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 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 is super-useful for making other people's code accessible (such as everything in the standard library).

Outline

When to use functions

- Code that appears 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.
- Code that has a discrete 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.
- A function should generally perform one (and only one) task.
- When a function becomes too long, too complicated, or hard to understand, it should be split into multiple sub-functions. This is called refactoring.

Outline

Simplicity of purpose

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

- Reading inputs from the user
- Calculating a value from the inputs
- Printing the calculated value

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).

Outline

Outline

Outline

Pass by value

Pass by value

- The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function.
- changes made to the parameter inside the function have no effect on the argument.
- By default, C++ uses call by value to pass arguments.
- This method copies the actual value of an argument into the formal parameter of the function.
- changes made to the parameter inside the function have no effect on the argument.

Outline

Pass by reference

- This method copies the reference of an argument into the formal parameter.
- Inside the function, the reference is used to access the actual argument used in the call.
- This means that changes made to the parameter affect the argument.
- When a variable is passed by reference, what is passed is no longer a copy, but the variable itself, the variable identified by the function parameter, becomes somehow associated with the argument passed to the function, and any modification on their corresponding local variables within the function are reflected in the variables passed as arguments in the call.

Outline

Pass by reference versus value

Pass By Value

- The local parameters are copies of the original arguments passed in
- Changes made in the function to these variables do not affect originals

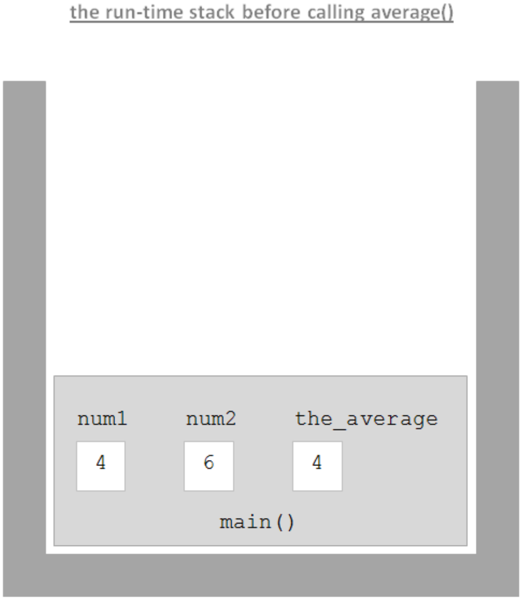
Pass By Reference

- The local parameters are references to the storage locations of the original arguments passed in.
- Changes to these variables in the function will affect the originals
- No copy is made, so overhead of copying (time, storage) is saved

Outline

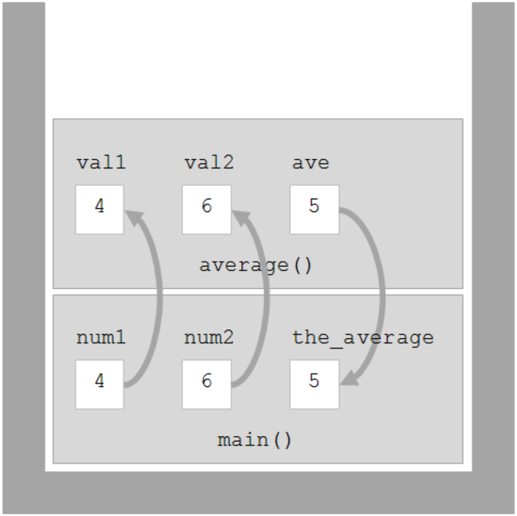
Outline

See code

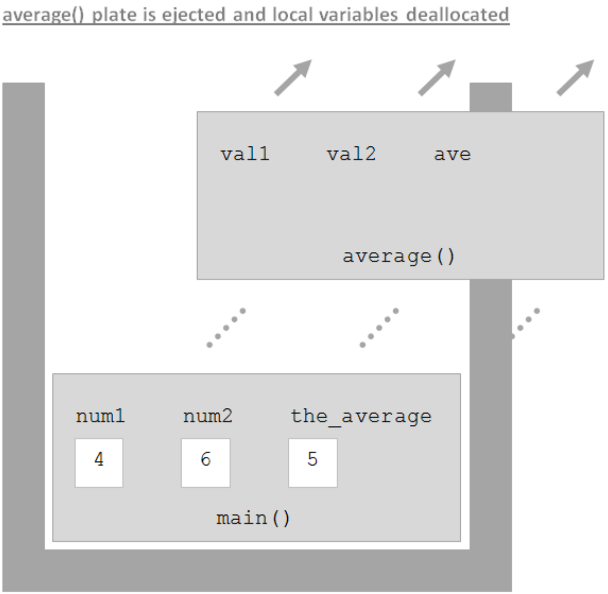

the run-time stack before calling average()
num1
4
num2
6
the_average
4
main()

See code

the run-time stack after calling average()

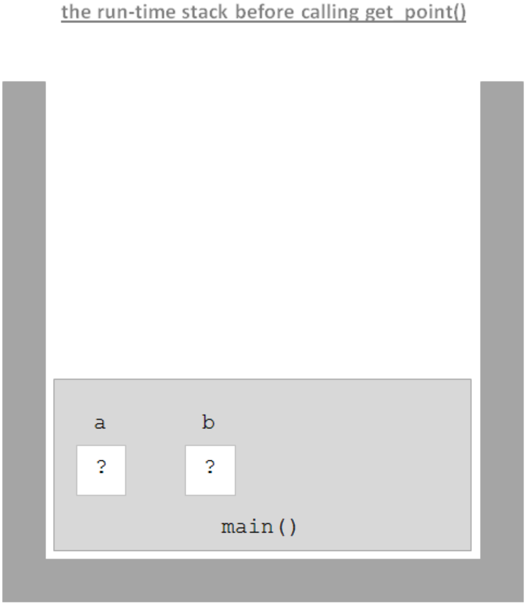


See code


average() plate is ejected and local variables deallocated
val1
val2
ave
average()
num1
num2
the_average
4
6
5
main()

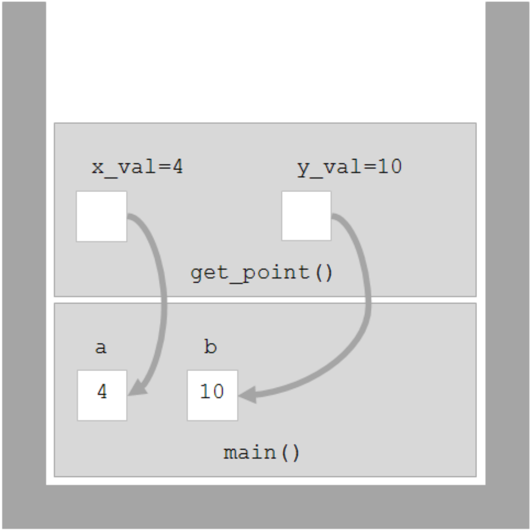
Outline

See code

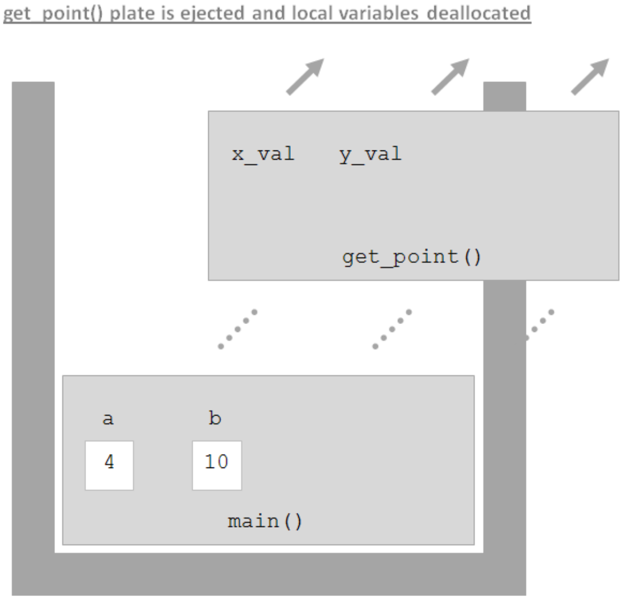


See code

the run-time stack after calling get_point()



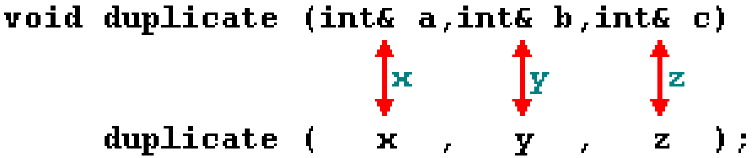
See code



get_point() plate is ejected and local variables deallocated
x_val
y_val
get_point()
a
b
4
10
main()

Outline

see code



```
void duplicate (int& a,int& b,int& c)

     duplicate (   x  ,   y  ,   z  );
```

- a, b, and c become aliases of the arguments passed on the function call (x, y, and z) and any change on a within the function is actually modifying variable x outside the function.
- Any change on b modifies y, and any change on c modifies z.
- That is why when, in the example, function duplicate modifies the values of variables a, b, and c, the values of x, y, and z are affected.

Advantages of passing by reference:

- References allow a function to change the value of the argument, which is sometimes useful. Otherwise, const references can be used to guarantee the function won't change the argument.
- Because a copy of the argument is not made, pass by reference is fast, even when used with large structs or classes.
- References can be used to return multiple values from a function (via out parameters).
- References must be initialized, so there's no worry about null values.

Disadvantages of passing by reference:

- It can be hard to tell whether a parameter passed by non-const reference is meant to be input, output, or both. Judicious use of const and a naming suffix for out variables can help.
- It's impossible to tell from the function call whether the argument may change. An argument passed by value and passed by reference looks the same. We can only tell whether an argument is passed by value or reference by looking at the function declaration. This can lead to situations where the programmer does not realize a function will change the value of the argument.

Use cases

When to use pass by reference:

- When passing big structs or classes (use const if read-only).
- When you need the function to modify an argument.

When not to use pass by reference:

- When passing small fundamental types (can use pass by value).
- When passing built-in arrays (use pass by address).