Functions (1)

Creating your own functions
Goals

- By the end of this lesson you will be able to:
  - Write and use your own functions
  - Understand the difference between passing by reference and passing by value
Function background

• We’ve been using pre-defined functions that are part of the C++ library to do various things

• We will concentrate on regular functions for now where that function does something (e.g. getline), evaluates to a value (e.g. pow), or both

• The main body of your program is actually a function too – it is a special function where your program starts executing
Structured programming

- Structured programming is a programming paradigm in which programs are broken up into smaller pieces
- Smaller pieces include loops, blocks, and functions
- Functions may also be called methods, subroutines, procedures, routines, or callable units
- Aside from the most basic elements of C++ (e.g. arithmetic, if/for/while) a structured C++ program essentially consists of a sequence of function calls
Why write your own functions?

- Code that repeatedly needs to be executed can be copied/pasted or placed in a loop
- It may be beneficial to place this code in a function
- It makes sense to write a function if it has a clearly defined purpose
  - Input is an integer, output is a Boolean for if it is a prime
  - Input is a string, output is the string in reverse
  - Input is two variables, output is that those two variables swap
- Once the function is written it can be tested, debugged if necessary, and then saved for future use for that and other programs
A function is written outside of the main body and has the following syntax:

define_datatype function_ident(param1, param2, ...)
{
    statement1;
    statement2;
    ...

    return expr; // if a non-void function
}
Syntax for writing functions

- Function names adhere to normal identifier rules
- There can be as many parameters as needed, even zero
  - Each parameter is specified by both a data type and identifier
- The data type for the function is also called the **return type** and indicates what type of value the function will return upon completion
- The function may not need to return a value, in which case the data type of the function can be **void** and no **return** statement is required
- The braces for the function are required, even if there is only one statement in the function
- You may call functions from within other functions
Function prototypes

- Typically we place the functions *after* the main body, however the compiler processes the program in a top-to-bottom manner.

- To tell the compiler to expect a function after we actually wish to call it, we use a function prototype before the main body as follows:

```plaintext
datatype fcnname(param1datatype, param2datatype, ...);
```
Local variables

- Values that you pass to a function are accepted into the function by making local copies only available to that function.
- Variables that you declare inside functions can only be accessed within that function and do not keep their values when the function is called again.
- When you pass something to a function and change that value in the function, the value is only changed for that function.
Function comments

- Commenting each function is necessary so others can understand it easier and you can view it later and recall what it does.
- Each function should have a comment that:
  - Describes the purpose of the function
  - Provides details about the specific input that is to be provided
  - Provides details about the output that the function provides
  - Any restrictions (e.g. do not use this function with negative values)
- Additional code within the function may be commented if warranted (e.g. particularly complicated loops or logic)
Passing by reference

- When you pass a value to a function it’s called **passing by value** – a local copy is made and when the function is over that value is effectively “destroyed”
- If you wish to manipulate the values in the function and have them stay changed everywhere (even main) you must pass by reference by placing an ampersand & after the datatype in the parameter list of both the function and function prototype
- You should only pass by reference when you need to – your default should be to pass by value
Top down/bottom up

- Given a problem you think about how to implement a solution
- You may think about the problem from two perspectives:
  - Top-down – write a program starting at the logical beginning, gaining insight into the functions that may be needed, then writing them later
  - Bottom-up – Think of the functions needed, write them, then put them together into one larger program
- Pros/cons: You need good intuition and/or it may be difficult to think about bottom-up; starting with top-down may lead to significant rewriting of the code
Exploration

- Write your own functions that do the same thing that many built-in library functions already do, such as `hypot` from `cmath` and `isdigit` from `cctype`.
- Write a function that will take a string and return that string with alternating upper/lower case letters.
- Write a function that will take an integer representing a value and an integer representing a number of factors – return `true` if the value has at least the specified number of factors.
- Write a void function using pass-by-reference that will take two variable values and swap them (do not call this function `swap`, as that already exists in the `std` namespace).