CSCE 155 - C

Lab 05 - Functions

Dr. Chris Bourke

Prior to Lab

Before attending this lab:

1. Read and familiarize yourself with this handout.
2. Review the lecture notes on conditionals, or the following free textbook resources:
   - http://en.wikibooks.org/wiki/C_Programming/Procedures_and_functions
   - http://www.cs.cf.ac.uk/Dave/C/node8.html#SECTION00800000000000000000

Peer Programming Pair-Up

To encourage collaboration and a team environment, labs will be structured in a *pair programming* setup. At the start of each lab, you will be randomly paired up with another student (conflicts such as absences will be dealt with by the lab instructor). One of you will be designated the *driver* and the other the *navigator*.

The navigator will be responsible for reading the instructions and telling the driver what to do next. The driver will be in charge of the keyboard and workstation. Both driver and navigator are responsible for suggesting fixes and solutions together. Neither the navigator nor the driver is "in charge." Beyond your immediate pairing, you are encouraged to help and interact and with other pairs in the lab.

Each week you should alternate: if you were a driver last week, be a navigator next, etc. Resolve any issues (you were both drivers last week) within your pair. Ask the lab instructor to resolve issues only when you cannot come to a consensus.

Because of the peer programming setup of labs, it is absolutely essential that you complete any pre-lab activities and familiarize yourself with the handouts prior to coming to lab. Failure to do so will negatively impact your ability to collaborate and work with others which may mean that you will not be able to complete the lab.

1 Lab Objectives & Topics

At the end of this lab you should be familiar with the following

- Know how to write and use functions
- Know the difference between a function prototype and function definition

2 Background

Most programming languages allow you to define and use functions (or methods). Functions are reusable blocks of code that can be packaged as a single unit. A function can be specified to inputs (parameters, arguments) and return an output. Defining functions has several advantages. First, it facilitates code reuse. Rather than cutting and pasting the same block of code, it can be encapsulated into a function and reused by calling the function anytime it needs to be executed.

Second, functions facilitate *procedural abstraction*. Often times, we don't care or need to worry about the implementation details of a certain algorithm or procedure. By encapsulating the details in a function, we only need to know how to use it (what inputs to provide it and what output we can expect from it) and not need to worry about how it computes its result. For example, up to now you've been using the standard math library's function to compute the square root of a number x, but you haven't had to worry about the details of how this computation actually takes place.

When defining a function, it is necessary to declare its *signature*. The signature of a function includes:

- The function's identifier – its name
- The return type – the type of variable the function returns)
- The parameter list – the number of parameters the function takes (also called its *arity*) along with the types

2.1 Functions in C

In C, functions must be declared before they can be defined (much like variables). Functions are declared by defining prototypes: a declaration of the function's signature followed by a semicolon. Prototypes are usually placed before the main function or in separate header files. Functions are defined by repeating the function signature and adding a block of code that specifies the instructions that will be executed when the function is invoked.

3 Activities

We have provided partially completed programs for each of the following activities. Clone the lab's code from GitHub using the following URL: `https://github.com/cbourke/CSCE155-C-Lab05`.

3.1 Using Functions

The file `orderStatistic.c` contains code necessary to find the i-th order statistic of a collection of numbers. The i-th order statistic corresponds to the i-th element in a sorted list. For example, the 4th order statistic of the list [5, 99, 23, 14, 6] is 23; the sorted list is [5, 6, 14, 23, 99], and 23 is the 4th element. Some special cases:

- The 1-th order statistic is the *minimum* element
- The n-th order statistic is the *maximum* element
- The $\frac{n}{2}$-th order statistic is the *median* element

The program converts command line arguments into i and an array of integers. It then sorts the array and outputs the i-th element. It does so by calling a series of functions.

Instructions

1. Complete question 1 on your worksheet.
2. Open the `orderStatistic.c` source file and study the code (do not make any changes). In particular, read the prototypes for each function and their documentation to learn what each one does.
3. Compile and run the program with the following input values:
   `./[nameOfTheProgram] 4 99 23 76 100 8 3 0 1 72 104 1000 12 18 14`
4. Answer the relevant questions on your worksheet.

3.2 Writing A Function

Open the `sine.c` source file. This is a skeleton program similar to a previous lab exercise that uses a Taylor series to compute $\sin x$. Modify the program so that the code that computes $\sin x$ is encapsulated into a function:

1. Write a function prototype for this function: what is its return type? What are its parameters? Note: a function named `sin` has already been defined in the standard math library; you will not be able to use this identifier (just as you can't have two variables with the same name).
2. Write the function definition by moving the code from the `main` function to your function.
3. Modify the code in the `main` function to use your new function.
4. Compile and test your program.

3.3 Writing More Functions

Open the `physicsCalc.c` source file. This is a skeleton program that you'll complete using your own functions. This program will prompt the user for the following input:

- The distance an object traveled from point 1 to point 2 (variable distance)
- The time required to travel distance (variable time)
- The object's initial velocity (variable initVelocity)
- The mass of the object (variable mass)

You will need to write a series of functions that will calculate the force of the object using the instructions below. Each function should be independent from the others (i.e., the force function shouldn't call the acceleration function, and the acceleration function shouldn't call the velocity function).

1. Write a function that calculates the average velocity, using the equation

$$v = \frac{\text{distance}}{\text{time}}$$

2. Write a function that calculates the average acceleration using the equation

$$a = \frac{v - v_0}{\text{time}}$$

   where v is the current velocity and v_0 is the initial velocity. For your program, use the average velocity as the current velocity.

3. Write another function that calculates the force of an object using the equation

$$F = ma$$

   where m is the mass of the object and a is the acceleration.

4. Add a print statement to print your result to the screen.
5. Answer question 3 on the worksheet and demonstrate your program to the lab instructor.

4 Advanced Activity (Optional)

Write a few more functions in the `orderStatistics.c` program. First, write a function that gets the *median* of a list of elements. However, the function should *not* make changes to the array. Instead, write additional functions that create a *copy* of the array and sorts the copy so that it can find the median element.