Lab: Functions in C++

Goals: By the end of this lab, you should be comfortable with writing functions in C++.

Overview: For large programs, you need to have procedural abstraction in your code. When you hear the term procedural abstraction, this simply means the use of modules. In C++, modules are called one of several names:

- Functions
- Procedures
- Sub Algorithms
- Methods (in OOP)
- Members (in OOP)

We really won't focus on writing functions in classes (OOP), but instead, we'll put the functions right above the main function.

All Functions: No matter what kind of function you write, it will have to specify it's *return type*, it's *name*, and any *parameters* that are passed to it. Let's go through a scenario:

You'll play the role of the function named "average2Nums". Think about this: if someone tells you to average two numbers, will you return them any information? Sure! You return them the result. Ask yourself what data type the result is and that's your return type. So here, our return type would be a float or a double. I've already told you the function's name is "average2Nums" - so that's easy. Finally, if you're this function, do you need any information to do your job? Yes - you need two numbers to average. Let's look at the first part of the function:

```
float average2Nums (int num1, int num2)
```

What this says is that if anyone ever calls average2Nums to do some work, they need to pass average2Nums two numbers that are ints, and they should expect to get back a float! The first number that is passed to this function goes into num1, and the second number goes into num2.

Now it's time to calculate the average. We can do this much as you imagine:

```
    float result;
result = (num1 / num2);
```

Now that the method has the result, it should return it to whoever called the method:

```
return result;
```

If we put all of this together, you'll get something like this:

```
float average2Nums (int num1, int num2) {
 float result;
 result = (num1 / num2);
 return result;
}
```

Just a note: if you ever have a function that returns nothing (i.e. a procedure), you return void.

Getting the Function to Work:

To get the function to do its job, you call it by its name; in this case the name is "average2Nums". For things to work correctly, you need to:

1. Pass it the correct number of parameters
2. Pass them in the correct order
3. Pass the correct data types

In this case, average2Nums needs two ints for it to work correctly. So starting out, you might *guess* something like this:

```
int userNum1 = 5;
int userNum2 = 6;

average2Num2 (userNum1, userNum2);
```

The above line calls the function average2Nums and passes it userNum1 (which goes into num1) and userNum2 (which goes into num2). average2Nums will then do its job (it should return us the answer of 5.5). Where does the 5.5 go? It goes back to the main algorithm! The problem lies in the fact that we didn't do anything with the return from that function! Here's a better example:

```
int userNum1 = 5;
int userNum2 = 6;
float average;

average = average2Num2 (userNum1, userNum2);
cout << average << endl;
```

```
// Here's another way to do it on one line
cout << average2Nums (userNum1, userNum2) << endl;
```

In this case, you can see we print it out, and keep a copy in a variable named average (yes, I know I could have printed out average instead of calling the function again - I just did this as an example). If you ever call a function that returns void, then you don't have to store what the function is returning (it's returning "nothing")

Our final code:

Now that we've designed the function and have seen how it's called from the main algorithm, we can put it all together. Here's our final code:

```
#include <iostream.h>

float average2Nums (int num1, int num2) {
    float result;
    result = (num1 / num2);
    return result;
}

void main ( ) {
     int userNum1 = 5;
  int userNum2 = 6;
  float average;

    average = average2Num2 (userNum1, userNum2);
    cout << average2Nums (userNum1, userNum2) << endl; // prints out 5.5
    cout <<  average;                          // prints out 5.5
}
```

When looking at this code, you have to start at the word "main" - not the function. In this case, we pass to variables (userNum1 and userNum2) to the function average2Nums, and it returns us the value 5.5. And that's it! I would recommend creating your own functions. Here are some ideas:

- Write a function that finds the max of two numbers
- Write a function that returns a letter grade (char) when passed a number (i.e. a 74 is a 'C')
- Write a function that prints a menu (this would have a void return type)
- Write a function that finds the cube of a number

The list goes on. After you write these functions, try calling them and making

sure they work!

Advanced Topic: function prototypes

When calling any kind of function in C/C++, you should always place the function being called above the one that calls it; we have to do this because of the way that the code is being compiled. If you ever have the desire to call a function below the current code, you'll need to create a function prototype. This is really easy to do. All you need to do is write out the function header followed by a semi-colon. This is best explained with an example, so let's rewrite the code from above with the function below the main:

```
#include <iostream.h>
// Here's the prototype of the function.  It's required because
//  average2Nums is defined below where it's being called from.
float average2Nums (int num1, int num2);

void main ( ) {
    int userNum1 = 5;
  int userNum2 = 6;
  float average;

    average = average2Num2 (userNum1, userNum2);
    cout << average2Nums (userNum1, userNum2) << endl; // prints out 5.5
    cout <<  average;                                  // prints out 5.5
}

float average2Nums (int num1, int num2) {
    float result;
    result = (num1 / num2);
    return result;
}
```