Introduction

Functions in programming are very similar to functions in math. They take in an input and map it to an output in a predictable way. The functions you are used to seeing in math take in number(s) and return number(s) that are the result of a series of operations on their input. These functions can easily be translated into code. Consider the very simple function $y = mx + b$. A simple problem this equation can solve is:

Given the slope and intercept of a line and a value x, calculate the corresponding value of y.

We could very easily write a small program for this as follows:

```
1  #include<iostream>
2
3  using namespace std;
4
5  int main(){
6      int m, b, x;
7      cout << "Input m, b, x" << endl;
8      cin >> m >> b >> x;
9
10     int y;
11     y = m * x + b;
12
13     cout << y << endl;
14
15     return 0;
16 }
```

This is simple and will get the trick done; however, as your programs grow larger and what you want to calculate gets more complex it will make your code infinitely more readable, simpler, and reusable if you break the program into smaller functions (as we'll see below). So how can we write this as a separate function?

Functions absolutely MUST have the following 4 things:

1. return type
2. name
3. list of parameters the function takes
4. body with return statement

Three of these items are easily visible in the function you are most familiar with so far, main. The "**int**" on line 5 is the return type of main, "main" is the name of main, and line 14 showcases the return statement. The list of parameters that main takes is not as clear. Notice how main has a pair of parantheses with nothing between them. This is because main (as we have written it) does not take any parameters.

Our function for calculating y returns the integer y that is on the line given by m and b at the given point x, so like main our function has the return type **int**. Unlike for main, we can choose any name

we want for our function, provided it does not start with a number, contain any restricted characters, or match any reserved words. We will call our new function calcY. In order to calculate y, we need to know what values of m, b, and x to use. These our the parameters that our function will take. Lastly, we must include a statement at the end of our function that returns an **int**. (Ideally, we would like this int to be the correct value for y, but the compiler does not care.)

Putting it all together, we get something like the following:

```
1  #include<iostream>
2
3  using namespace std;
4
5  int calcY(int slope, int intercept, int x){
6      int y;
7      y = slope * x + intercept;
8      return y;
9  }
10
11 int main(){
12     int m, b, x;
13     cout << "Input m, b, x" << endl;
14     cin >> m >> b >> x;
15
16     int y_value;
17     // y_value = calcY(int m, int b, int x);     Very Bad!
18     y_value = calcY(m, b, x);
19     // cout << y << endl;                        Very Bad!
20     cout << y_value << endl;
21     return 0;
22 }
```

There are a couple things worth noting in this small example. First, look at line 5. Notice how each of the parameters have their type before their name. Where else have we seen variables preceded by their type? Variable declaration. The similarity is no accident. Every single time we make a call to calcY it gets its own variables slope, intercept, and x. In order for the computer to make the appropriate space to store these variables, we need to tell it what kinds of variables they are. (As you will see in the future, this also helps the computer call the right function if you have multiple functions with the same name.) Notice too how the variables slope and intercept have a different name in calcY than main, while x has the same name in caclY and main. Both of these cases are fine, because the variables in calcY belong to calcY and the ones in main belong to main. This is a very important distinction, and the reason why lines 17 and 19 are so bad. main has no variable named y, so trying to print out the value of y does not work. In order to get the value of y from calcY into main we must return it from calcY and store it into a variable in main. Likewise, main already has the variables m, b, and x so we cannot declare them again.

Conceptual Questions

1. Suppose we add a line after line 7 with the statement "x++;" and a line after line 20 with the statement "cout << x << endl;". Is the value printed by this new line the same or one greater than the value inputted for x on line 14?

2. Can we replace lines 18-20 with just "cout << calcY(m, b, x) << endl;" without affecting our output?

3. Can we replace lines 6-8 with "return $slope * x + intercept$;" without affecting our program?

Answers

1. The same! Always, always, always remember that a function owns its own variables. When we change x in calcY, it will not affect x in main.

2. Yes! We can think of it this way: when the program runs, it will take the function call and replace it with the value the function returns. So a statement like "cout << calcY(m, b, x)*2 << endl;" is also valid syntax and the output will be twice the value of calcY(m, b, x).

3. Yes! The value to the right of the return statement will always be calculated before it is returned.

The previous example is a little bit silly, in that all our auxilary function does is something that could always just be done in one line wherever necessary. It's not clear why separating things into separate functions is useful. Hopefully the following excercise illustrates the value and power of functions.

Exercise 1

Your goal is to write a program that will take in the orders of three separate customers. For each, you will first read in their name and then an integer from 1-5 representing their order. Once you have done this for all 3 customers, print out their name and order in REVERSE ORDER, substituting the number for the following:

1. Hamburger
2. Hot Dog
3. Chili Cheese Fries
4. Veggie Wrap
5. Soda

Skeleton code has been provided below. You can assume that whoever uses your program will only input a string for the customer's name and a value between 1 and 5 for their order.

```
#include<iostream>
#include<string>

using namespace std;

_______ getOrderAsString(____ number){
    |
    |
    |
    |
    |
    |
    |
    |
    |
    |
}

int main(){
    string name1, name2, name3;
    int order1, order2, order3;

    cout << "What is your name?" << endl;
    cin >> name1;
    cout << "What is your order?" << endl;
    cin >> order1;

    cout << "What is your name?" << endl;
    cin >> name2;
    cout << "What is your order?" << endl;
    cin >> order2;

    cout << "What is your name?" << endl;
    cin >> name3;
    cout << "What is your order?" << endl;
    cin >> order3;


    // YOUR CODE HERE
    |
    |
    |
    |
    |
    return 0;
}
```

Now suppose you did not have to print the orders out in reverse. How could you do this with a while loop and no external function?

A Few Notes

Above we have only seen examples of functions that return **int**s (excluding the exercise). This is not the only return type a function can have. In HW5 you had to implement a function with a return type of **bool**. In the exercise, getOrderAsString had a return type of string. From this, it's easy to see that you can write functions with return types of any of the variable types you've already learned. There are two other possibilities that you may have not seen yet. The first is *void*. This is a function that does not return anything. (While it may seem silly now, the utility of void functions will be more apparent when you learn about classes and passing-by-reference.) The second is the possibility to return custom classes. This is not something that you will worry about until possibly the end of 103, but more likely in 104.

Review Questions

1. What are the four components of a function definition?
2. Suppose you are given an implementation of the following function:
   double radiansToDegrees(double radians);
   Write a line of code to convert a variable rad to degrees and store the result.
3. Suppose you wanted to call a function whose sole purpose was to print to the terminal. What return type would you give it?
   Hint: Think about what information if any this function would need to send back to main.