Programming Notes 4: Functions

Always remember, I will provide examples of the notes in the example program.

1. Intro to Functions

Functions are arguably (ok, definitely) the most essential part of programming. You will be using them for more or less everything, and it is important to know how to write and use them well.

2. Function Theory

There are several basic characteristics that represent good function design: first, a function should accomplish one and only one task. The function might do that thing many times, or in different situations, but a function should only be designed to solve one problem. However, the scope of that problem is up to you, as is where you want to breaks smaller parts of that task into more functions. However, if you have completely separate types of operations, say data processing and console output in the same function, you're doing it wrong*. For example, the main function has the task of executing your program. Then, a function you write might display a menu system, and another might execute one of the options, and another might do something specific within that option, etc. Second, functions should demonstrate modularity and reusability. Now, a function is essentially a reusable block of code, so if it was only designed to work in one situation, for example, that would mostly defeat the purpose of having the function at all. Pretty self-explanatory. Finally, functions should, of course, meet specification. This basically means that the function should have a specification of what it should do, and it should do that. Also pretty self-explanatory.

*Not objectively wrong, per se, but separating things like processing and IO shows good design practice, as it makes debugging much easier.

3. Function declarations

Now that you know what functions do, you may be wondering how to use them in actual programming. Well, there are several important aspects of functions that you need to know in order to use them effectively. The basic structure of a function you've already seen in the main function of your previous programs. A function is essentially a block of code (a scope) that can be called from other code, receiving and outputting data.

To declare a function, there are two things you have to do. First, there is the function prototype, which goes at the top of your program between your global variables and your main function. The prototype is technically unnecessary, but it allows functions to be used anywhere in your program. If you can't, it can lead to a lot of headaches. The structure is:

```
<data type> functionName(<data type> param1, <data type> param2);
```

There are several components to this statement: first, the initial data type describes what type of variable the function will return. More on that under "Using functions." If you don't want your function to return any data, you can write "void" here instead of a data type like "int" or "double." Second comes your function identifier, which we have gone over previously. Then you have your function's parameters enclosed in parenthesis. Parameters describe the data that your function will receive when it is called by other code. To add a parameter, first type the data type of the parameter, for example “int” or “char,” and then the identifier of the parameter. To add more parameters, simply add a comma and repeat the process. The parameters that you add in the prototype and declaration of the function are called formal parameters, as they define what will be used in the function. Finally, cap off the statement with a semicolon. Examples:

```
void printCharacters(int numCharacters, char character);

double calcRoot(int a, int b, int c);
```

4. Writing functions

One you have written your function prototype, you can move on to the actual function. Now, to declare the implementation of a function, simply paste your prototype—minus the semicolon—below your main function, and add a pair of curly brackets. Now you can write the code within your function.

You can use your function parameters as variables within your function, as they will hold the data passed into your function when you call it (more on that in a sec). Once you’ve finished writing the code within your function, if you want to return data (remember the data type you defined the function as), you must put a “return” statement at the end of your function, or somewhere where it will always be run. After typing return, type either a literal value or more often a variable of the data type you specified the function will return. Example – a simplified quadratic formula function:

```
double calcRoot(int a, int b, int c) {

        double x;

        x = (-b + sqrt(b*b – 4*a*c)) / 2*a;

        return x;

}
```

5. Using functions

So, finally, you need to learn how to use this function you created within your other code. Doing this is called calling the function, and to do so, you first—if the function has a return type—identify a variable that will hold the value returned from the function, then set it equal to the function name, and finally add your parameters. These parameters are called actual parameters, as they represent the actual data that you pass to the function. To add the parameters, simply enclose literals or variables of the correct data types within parenthesis after your function identifier. Note that these do not have to have anything to do with the actual parameters other than matching their data types—the variables are completely separate and can be named whatever you want. For example:

```
int a = 5;

int bValue = 3;

double oneRoot = calcRoot(a, bValue, 12);
```