CS 161C++
Functions

Functions are a way to break your program into smaller pieces. This allows multiple people to work on the same project, makes it easier to understand and maintain, and lets you reuse a method of solving a particular problem.

What is a function

Another term that is often used for a function is a routine. If you think of your personal life, you use routines on a daily basis. For example, this morning I made coffee.

```
Coffee Routine:
    inputs:

        Water
        Coffee beans

    Process:

        Grind up beans (multiple ways to do this)
        Make coffee
            could cold brew, steep, drip, …
    Outputs:

        Cup of coffee
```

Any time I want to have some coffee, I simply take the inputs, follow the process, and I get the output.

The same thing works for a computer routine or function. It has some sort of inputs, has a process, and then has some results.

Function Format

The general form of a function is:

```
    return_type function_name ( parameter_list )
    {

        code to do the processing

        return statement

    }
```

Lets consider each part separately. First, the return_type. This tells the computer what sort of thing the function will return. It could be an integer, a floating point number, a character, a string, a boolean value, or some user defined type. Alternately, there might be nothing returned (for example a function that simply outputs the result on the console); functions that return nothing have a return type of **void**.

Now consider the function_name; this is simply an indentifier that a programmer uses to refer to this function when they want to call it (use it) later. It follows the same rules as other identifiers, such as variable names. It starts with a letter and then can include letters, numerals, or the underscore. It is important that function names are chosen so that the user can readily understand what the function is going to do. Names such as do_it or x or b7, probably will not help someone later working with the program to understand it.

The parameter list is a comma-separated list of the inputs to the function. These are variable names that will be used in the function when the processing code needs to access that input.

The code to do the processing is what we have been using all term. It uses the input parameter names when it needs to access one of the inputs, may define local variables, and does whatever the programmer wants to have done.

Finally the return statement is used to return the results of the function, if any. If the return_type is not void, then the return statement must have an argument that is of the same type as the return_type. If the return_type is void, then the return statement does not have an argument.

Examples

First, lets consider our coffee process:

Return type = beverage
Function name = brewCoffee
Parameters or inputs = water, coffee beans

```
beverage  brewCoffee( ingredient water, ingredient coffee_beans)
{
        grounds = grind coffee_beans:
        hot_water = heat water;
        coffee = grounds + hot_water;
        return coffee;

}
```

Lets look at a simple function that adds two integers and returns their sum:

Return type = integer
Function name = adder (or anything else you want to use)
Parameters or inputs = the integers to be added, lets call them number1 and number2

```
int adder(int number1, int number2)
{
        int sum;

        sum = number1 + number2;

        return sum;

}
```

Now look at how we could use the adder function:

```
int main()
{
        int value = adder(3, 4);

        cout << value;

        return 0;
}
```

When the computer is executing this program, it comes to the assignment statement setting value. It sees that a function is being called. It stores the value 3 in the parameter number1 and the number 4 in the parameter number2. Then it jumps to the location in memory where this function is located and executes its processing code. When it is done with the processing, the return statement causes processing to return to the calling program and the result in the variable sum is then copied into the variable value in the main program.

How about a function that gets an integer from the console and returns that to the calling program.

Return type = integer
Function name = getInteger
Parameters = none, this function reads from the console and has no input from the calling program

```
int getInteger()
{
	int value;
	cout << "Enter an integer" << endl;
	cin >> value;
	return value;
}
```

Now lets use this in our program. Note that we are also using the adder method from above.

```
int main()
{
	int num1 = getInteger();
	int num2 = getInteger();

	cout << "The sum of your numbers is " << adder(num1, num2) << endl;

	return 0;

}
```

The computer executes this program. When it gets to setting num1, it calls getInteger, reads an integer from the console, returns it. The returned value is stored in num1. The same process is used to get and store a value in num2. Then the cout statement calls the function adder and outputs the result.

To complete this program, lets move the cout statement into a function of its own.

Return type = void (nothing is returned)
Function name = displaySum
Parameters = the integer to be output, lets call it sum

```
void displaySum(int sum)
{
	cout << "The sum of your numbers is " << sum << endl;
	return;

}
```

Note that there is no argument on the return statement in this function. This is because it has a return type of void.

Now our complete program looks like:

```
#include <iostream> // for cin and cout

using namespace std;

    //define getInteger
int getInteger()
{
    int value;
    cout << "Enter an integer" << endl;
    cin >> value;
    return value;
}

    //define adder
int adder(int number1, int number2)
{
    int sum;
    sum = number1 + number2;
    return sum;
}

    //define displaySum
void displaySum(int sum)
{
    cout << "The sum of your numbers is " << sum << endl;
    return;
}

int main()
{
        // define variables
    int num1 = getInteger();
    int num2 = getInteger();

    result = adder(num1, num2);

    displaySum(result);

    return 0;
}
```