MP2: Functions

Objectives:

- Implement functions to modularize a program.
- Use integer operations to manipulate decimal integer values.

Tasks:

1. Implement a function that counts the number of decimal digits in an unsigned integer value.
2. Implement a function that prints the English word equivalent of a single decimal digit, e.g., "two" for 2.
3. Implement a function that prints all the decimal digits of a positive integer as English words.

Task 0: Getting Started

In CLion, create a **New Project** (under the File menu). The type is "C Executable" and the Language Standard is "C90". Use **mp2** as the last part of the project location (replacing "untitled"). CLion will create a file named **main.c**, and will put a "Hello, World!" print statement in the main function.

We're going to replace this code with the pre-written code provided for this exercise. There should be a simple way to do this in CLion, but the best I could find is:

1. Create a new C project in CLion (C executable, C90 standard). For the purposes of explanation, we'll assume you named this project **mp2**.
2. Download the file **mp2.c** that is included as part of the assignment. Put it into the folder that was created for the next project. (E.g., c:\Users\Greg\CLionProjects\mp2) You should see the file "appear" in the list of files in your project.
3. Open the **CMakeLists.txt** file in the project. In the last line, replace **main.c** with **mp2.c**. (You are telling the build system to compile m2.c to create the executable.) Save the file.
4. Right-click on the project name, and select "Reload CMake Project".
5. Finally, get rid of the **main.c** file that was created for the project. Right-click on that file name and select "Delete".

Now you have a project that only contains the **mp2.c** file. This is your starting point – you will be editing this file to implement the functions.

Compile and run the program. It should prompt you to enter a positive decimal integer, and then it should print the following output. (It doesn't matter what number you enter; the output will be the same for any number.)[1] There are three functions defined by the program. In the file that I give you,

[1] If it echoes your input twice, remember the trick from last time. (1) Go to Help -> Find Action -> Registry… (2) Uncheck the box next to **run.processes.with.pty**.

these are “dummy” versions of these functions. They are implemented, but they don’t do what we want yet.

```
Enter a positive decimal integer: 123
The number has 0 digits.
The 1's digit is NOT DONE
NOT DONE
```

This is clearly not the way the program is supposed to work. The dummy functions provide these placeholder results, and you’ll fix each problem in the following tasks.

Task 1: Count the number of digits

First, implement the **numDigits** function, which counts the number of digits in the decimal representation of an integer. You may assume that the integer passed in as an argument is zero or greater.

This function is called in the first printf statement in main:

```
printf("The number has %d digits.\n", numDigits(num));
```

Note how the function call is used as an expression (value) in the call to printf.

The stub implementation of numDigits is given below the main function:

```
int numDigits(int val) {
    return 0;   /* not implemented -- return 0 */
}
```

This is why the program above said that the number had zero digits. Fill in the implementation of the function, compile it, run it, and test it with several different numbers to make sure the digit count is correct.

Hint: “Count the digits” is equivalent to “how many times can I divide the number by 10 before it becomes zero?” (What if the number is zero?)

When your **numDigits** function works, the example that we ran before will look like this:

```
Enter a positive decimal integer: 123
The number has 3 digits.
The 1’s digit is NOT DONE
NOT DONE
```

Task 2: Print the text version of a digit

Next, implement the **printDigit** function, which prints the English word corresponding to a number between 0 and 9. In other words, if the argument is 0, print “zero ”. If the argument is 7, print “seven ”. Print a space after the word.

Note how the main function extracts the one's digit from the number and passes it to the printDigit function:

```
printDigit(num % 10);
```

The one's digit of a decimal number is the remainder whenever the number is divided by 10. That's what the modulo operator is doing. This will be helpful in Task 3.

You will likely implement this function using a long `if...else if...else if...else` statement. If you know how to use a switch statement, you can. But you're not required or expected to know how to use switch in the class, so the if-statement approach is just fine. (If you're interested, the switch statement is described on page 253 of *C Primer Plus*.)

When your **printDigit** function works, the example that we ran before will look like this:

```
Enter a positive decimal integer: 123
The number has 3 digits.
The 1's digit is three
NOT DONE
```

Task 3: Print all digits as words

Finally, implement the **allDigits** function. This takes in an integer and prints all of the digits of the decimal representation of the integer as words. You will definitely want to call **printDigit** as part of your implementation, and you're welcome to call **numDigits**, if you want.

How can you "look at" the digits to know what they are? We've already done the one's digit above, and here's a hint:

- The 10's digit is the one's digit of (number / 10).
- The 100's digit is the one's digit of (number / 100).
- The 1000's digit is the one's digit of (number / 1000).
- See a pattern? The 10^n's digit is the one's digit of (number / 10^n). I'm not suggesting that you use some exponentiation operator here, because there is none. I'm simply demonstrating a useful pattern that might be exploited.
- Please DO NOT use the pow() function to raise 10 to a power. The pow() function works for floating-point numbers. Use integer operations only. (Hint: Divide the number by 10 until you get the desired digit in the one's place, then use modulo (%) to get only the one's digit.)

When your **allDigits** function works, the example that we ran before will look like this:

```
Enter a positive decimal integer: 123
The number has 3 digits.
The 1's digit is three
one two three
```

Submission and Grading

Submit your source code (**mp2.c**) to the Moodle MP2 assignment. Do not submit your executable program, just the source code.

Grading:

- 70 pts: **numDigits** works correctly
- 20 pts: **printDigit** works correctly
- 10 pts: **allDigits** works correctly