Manoj Vasa

10/17/2016

Code Reading MW

Reading #14

1) I traced some numbers through the getstops method to understand what was being assigned to the tabstops array. I used two numbers where:

Iteration 1: \*cp =2 iteration 2: \*cp = 4

I passed the string \thisisatest\nnewline\b

2) The first statement in this do while loop checks if there’s anything in the file at all. After establishing that the first thing in the file is not a null character the rest of the do while loop is executed. The code is written so that the file could be read at least once and establish its contents before parsing through the characters in the file.

3) This line of code is an infinite for loop. This is used when no starting or ending conditions can be specified for the loop. These loops are usually exited with a return, exit, or other conditions within the loop. In this case the loop is being exited by either the condition that is checking if I is less than or equal to zero or greater than or equal to 256 or by the other three if statements. Furthermore, there are multiple instances where the program is being exited. All these conditions can’t be stated in the for statement, so we use a for loop here.

4) The optind variable is used to skip over a number of argv entries. The line argv += optind skips over to the next argument. This then reduces the number of arguments in the count so argc is decremented by how many arguments were skipped over in the argv array which is optind.

5) When exiting a function, using exit with a parameter of 1 indicates that program exited with an error. However, if a parameter of 0 was used, it’d indicate that the program exited normally and had no errors.

6) The switch (case) statements in this code don’t have breaks since the break would just get out of the switch (case) statements completely. This code, however, needs to process every character before leaving the process. For this reason, a continue is used to go towards the next iteration or in this case the next character of the file.

7) The default case can be anywhere in the switch (case) statements and no error will be caused. This is because every case is checked before the default case. This chronology leads the default case to be the chosen case only when all the other cases aren’t right.

8) This while statement is checking to make sure that the right amount of spaces was placed at the beginning of each line. At the first iteration of the loop the column will be zero indicating that this process occurs on new lines. When I set tabstops[0] to be 3 and tested the number of spaces that were placed, I found that three spaces were placed. When I tried 4 for the offset, four spaces were placed. The same is true for any tabstop offset specified by tabstop[0]. This condition makes sure the amount of spaces in tabstop[0] was placed at the beginning of the line.

Code Reading Reflection

1. I feel had very moderate c knowledge before reading this code. I understood the control structures used but some minor statements eluded my comprehension. With some time, I was able to understand more of the lines of these programs. Understanding what value of a pointer is being used and how variable values go across functions were difficult to understand.

* After reading on your own (in class): %20
* After Reviewing the code with a classmate (in class): %28
* After tracing the code with a classmate (in class): %50
* At the completion of the assignment: %100
* Easiest first: I started by reading the do while loop in the main method since it was easier to read. Then I approached the getstops method with what knowledge I gathered from the main method.
* Tracing: I traced some values through the tabstops array to see how theyd be used throughout the program. I later traced string inputs.

1. Two major things I had trouble understanding were the calculation i = i \* 10 + \*cp++ - ‘0’ and the use of infinite for loops. This hindered my progress as I took some time search about the use of infinite for loops in my cprogramming.com source. I traced data to understand the calculations being done to i. this took some time to understand as I was unfamiliar with some c idioms such as subtracting by ‘0’. I later learned what this meant and made further progress with my reading. I also had trouble with the while condition (column – 1) % tabstops[0] != tabstops[0]-1. After some data tracing, I understood what was happening here.
2. Syntax
   * At first I was confused of the opposite update of the argc and argv.
   * I didn’t understand why argc was being decremented while argv was being incremented

Code: argc--, argv++

1. Semantics
   * The code made use of an infinite for loop that had no ending condition

Code: for (;;)

* + I was confused as to how this loop would be exited.

1. Style: none
2. Documentation
   * Some calculations and assignments in this code seemed rather cryptic without the proper documentation

Code: i = i \* 10 + \*cp++ - ‘0’

* + This code was hard to understand as it was not backed up by any documentation

1. I talked to Aaron Loomis about the code. We examined the tabstop array to understand what its purpose is. This opened up our understanding of the rest of the code as we approached the while statement (column – 1) % tabstops[0] != tabstops[0] -1. After discussing these two pieces of the code we had a better understanding of the flow and structure of the code.
2. I continued using the tutorialspoint resource I used to learn c originally. Another resource I used was cprogramming.com
3. I didn’t run any of the code. I traced the code with some test data but didn’t get around to actually running the code.
4. This code felt rather cryptic at my first reading until it became clear line after line.