Dylan Dunsheath

HW3

CS-288-006

Professor Dale

**Problem 1**

This initially seemed like a rather daunting problem/task to complete but it turned out to be rather simple:

In the main function, I started off by adding two parameters: int argc and char \*argv[] as I wanted to get the files via command-line arguments by the user. I made sure EXACTLY 3 arguments were passed, the executable file of the c program and then the two matrices that could be generated from the create\_matrix.c file. After I made sure that the appropriate number of arguments were passed, I used a c function found from unistd.h that allows me to test if a file exists with the F\_OK parameter (when it is equal to -1, it doesn’t exist). I did this for both files; where if either one or both meet this criteria, I outputted a message to the terminal and exit out of the program. Otherwise, I opened both binary files with argv[1] and argv[2] with “rb” since binary files require ‘b’ for binary. I also made sure that the files weren’t NULL and that the files actually could be open. I declared three different integers as well, one for matrix one, one for matrix two, and the third for both matrixes after verifying the dimensions are fine (they match and don’t exceed 100). I read the dimensions of both matrices with fread, given that the syntax for fread was fread(void \* buffer, size\_t size, size\_t count, FILE \* stream), according to <https://www.geeksforgeeks.org/fread-function-in-c/>. After getting the dimensions, that was when I checked the equality of n1 and n2 for matrix 1 and 2 and that neither the values exceed 100. If any of them did, I outputted the appropriate message to the screen, closed the files, and closed/exit out of the program. Otherwise, I initialized the value of n to n1 (wouldn’t matter if it was n1 or n2 since we realized at this point, they are the same). I then read the elements using fread (which also got the dimensions), which were read as a long integer to ensure there was no overflow. I then closed these 2 files and began calculating the product of the 3D matrices; which were all long integers. After the values of the result\_matrix was calculated, I opened a file called “result.bin” which used ‘wb’ for writing and binary. Similarly to the two matrices that were randomly created with create\_matrix.c, I wrote the dimensions (although I didn’t read them, this step is required) to the result matrix, I wrote the elements as a long as well, and then I output the matrices via a function called print\_matrix, which took a size ‘n’ and a matrix. I did this for all the files (matrix 1, matrix 2, and result\_matrix). Outputting essentially required three for-loops.

Note: I was required to change the syntax of create\_matrix.c to use a long int element rather than a int element to prevent overflow.

I then did a total of 3 loops simultaneously (since we are working for the )

**Problem 2**

This program was rather quite simple to achieve the expected results. I used three different preprocessor directives:

* stdio.h: Required for input and output (along with snprintf). Snprintf is a function that allows us to format and store a string into a buffer and ensures there is no buffer overflow
* String.h: With this directive, it allowed me to use strcmp and strstr to both:
  + Strcmp: This is used when traversing so we can exclude both the CURRENT and PARENT directory; otherwise we may run into an infinite loop or incorrect/unintended behavior. Doing so will allow us to provide meaningful and correct info.
  + Strstr: Given two arguments: string and substring, we can check if the substring exists within the stirng (or in our case, if it is a txt file) so we can process and count the lines in these files
* Dirent.h: Allows us to read/close a directory, declare ‘struct dirent \*entry’, verify what the entry is [is it a subdirectory or of a specific file type], and allows us to actually ignore the current and subdirectory when traversing.

When starting off the program, I declared an array called ‘directory’ with a size of 1001; which holds 1000 elements/characters PLUS the \0 or null terminator. I then prompted the user to enter a string which handles whitespace with “%[^\n]” and ensures there isn’t more than 1000 characters in the array with an if-condition. I also ensured that the length of the input was NOT 0 by checking if it was then letting the user know it is invalid input. I also checked if the directory existed. I then declared an integer called total\_lines and called traverse\_directory which will traverse the current directory and subdirectories. The output after it is called is then outputted to the user.

Within the traverse\_directory function which takes a const character array called “path” as an argument, which will allow us to traverse through the path given and count the total lines in each txt file. I then started off this particular function with declaring DIR \*dir and struct dirent \*entry. DIR \*dir is a pointer of type DIR which will hold information about an opened directory and the ‘entry’ is a pointer that’s named entry of type struct dirent that will check if the entry we are on is a directory/file (or if we are done when it is NULL). If it’s Directory, we declare a char array called sub\_path of the max length = 1001 (but we exclude the current and parent directory) that constructs the path of the subdirectory where we concatenate the parent directory called ‘path’ and the name of the current entry (entry->d\_name). This is stored in sub\_path. We then recursively call traverse\_directory with the sub\_path that returns the total count of lines within that directory.

* Otherwise, it’s a file where we verify it has a .txt extension. We then use file\_path which will construct the path of the current directory with the txt file and recursively calls count\_lines\_in\_file with the path to get the total lines of that file.

The function count\_lines\_in\_file also has a const char but it is called filename. In here, we open the given file that was passed from traverse\_directory with the fopen function. We then verify if the file is NULL and exit if this is the case. My method of counting the lines was using two vriables: count of type int and ch of type char. In a while loop, I did while((ch = fetc(file)) != EOF) { … } which continuously gets input from a file until we reach the end of the file. Everyimt we come across ]\n’ at the end of the line and into a new line, we increment the counter. We then close the file and return the count within the file and output it.