Nick Tansino

Prof. Hervé

CSC 412 - 0001

10 October 2020

Prog03 Report:

Prog03:

For this assignment, I planned to separate my tasks into smaller functions, considering the fact that I would repeat these tasks multiple times throughout the duration of my program. The most prominent functions in my program (besides the main()), were the read_image() and find_matches() functions. The former returns a struct (defined as "Image"), and the latter returns a char* (defined as "matchArray"). The read_image() function allocates and fills a given 2D array with the contents of either an image file (.img) or a pattern file (.pat), while the find_matches function parses through the contents of the '.img' file (now stored in an "Image" struct) in search of the desired pattern.

Prog03 (find_matches):

My find_matches() function takes in two parameters, both being instances of an "Image" struct. One is used to represent the '.pat' file data (pattern), and the other is used for the '.img' file data (img). The first thing I do upon entering my find_matches() function is initialize a counter (defined as count) to be '0'. I then define a 'char*' named matchArray that will store the number of matches (count) and the row(s)/column(s) in which the match(es) were found. I then begin to parse through the image in search of the pattern. To do this, I used a nested 'for' loop that respectively started from zero and ended at "img.numRows/Cols - pattern.numRows/Cols".

This allows the ‘img’ struct to be parsed until it can no longer accommodate for the dimensions of the pattern. Inside of this nested ‘for’ loop, I initialized an integer, ‘match’ to be ‘1’, which would assume a match until a discrepancy was found between two compared characters.

Following initialization of the match variable, I then created a nested ‘for’ loop that would compare the pattern struct to the image struct character-by-character, so long as ‘match’ remains as the value ‘1’. Within this ‘for’ loop is a conditional statement that is only reached if a discrepancy exists between the compared characters. If this conditional statement is reached, the value of ‘match’ is changed to 0, thus exiting the second nested for loop, and shifting the search over by one column, and eventually by row as well. If the conditional statement is not reached, then a match has been found. Outside of this nested ‘for’ loop lies a conditional statement that handles this case. The match counter is then incremented by one, and the top-left row/column numbers are added into matchArray. This process is repeated until the image has been searched entirely. The match counter is then added to the end of matchArray, and the array is then returned to the main function.

Difficulties:

The biggest challenge that I faced was getting the data in matchArray to return properly. I had a lot of trouble figuring out how to store the data as I was going without overwriting the existing data that was already stored in matchArray. Other challenges I had faced were trying to get the ‘.img’ files to open. This was because I was originally using just the filename in attempting to open the file, rather than the path to the file. I was able to solve this issue after consulting Prof. Hervé. Due to the amount of time that I spent on the C program, I was not able to complete the Bash program in time.

Limitations:

The biggest limitation that my program exudes is the output, in that it does not produce any. Though the pattern matching does work (proven through testing), as well as all of my other functions, I was not able to figure out my prior issue before the deadline was reached. One other limitation that I am aware of in my program is the fact that it does not check the file extension, and therefore may cause a “Segmentation Fault” if given the path to an invalid file format.