Sam Dinkelman

11/17/2018

CIS 247

Assignment Report 2

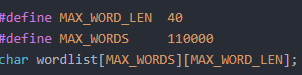
Introduction:

The purpose of this assignment was to create a program that calculates the hamming distance between a word entered by the user and a wordlist supplied. If we find the word in the wordlist we don’t have to suggest a word but if the word entered is not in the wordlist we suggest the word with the lowest hamming distance.

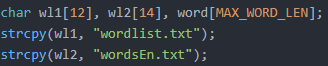
Process:

To begin this assignment I worked through the first two steps on the assignment page which were to check whether the name of the file containing the wordlist was provided on the command line. The second step was to open the file and read the words and add them to a list with one word per line removing the ‘/r’ and ‘/n’ characters at the end of each line.

At the top of the file I define several global variables which were provided to us in the description of the assignment.



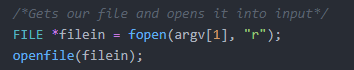
To start I created two character arrays of the length of the two wordlist files and a third character array that will store the word we enter. Then I copy the strings of the two names of the files into the character arrays for both



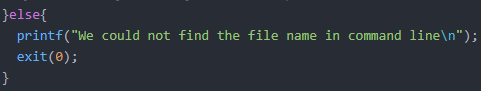
Next I perform a check to compare the strings entered on the command line to the character arrays we just filled with the wordlist file name.



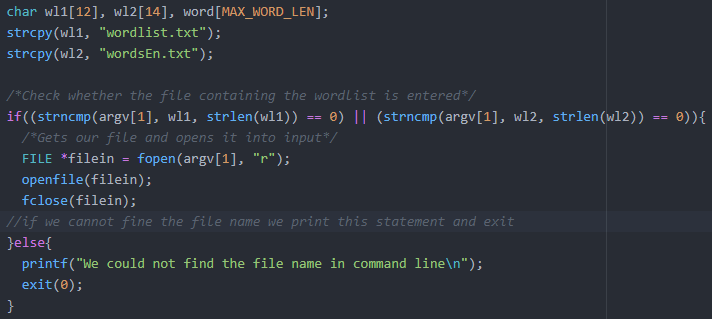
This will make sure that the file will only open if we find one of the two wordlist files we were supplied with since below this check I open the file using fopen with the string supplied on the command line. Then I pass it to the function which will parse the file out into the wordlist array. I also make sure that after I pass the file off to the function I close the file so that it doesn’t get changed in any way or mess with my program.



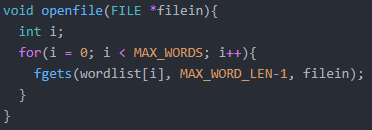
Also if this check doesn’t return any of the file names of the wordlists which we compared to the character arrays it will print that it couldn’t find the file and exit.



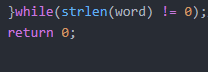
Here is this part of my code in full:



In my openfile function I take in the file that was opened in main using fopen and parse it out into the wordlist using fgets stripping off the newline character at the end of each line. I do this by having the length of the line fgets grab be equal to the MAX\_WORD\_LENGTH-1 which will strip off the newline character.

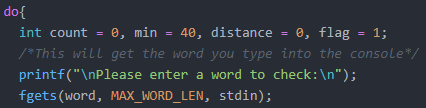


After this I have my file opened and parsed out and then closed again. So I have my full wordlist in the wordlist array. Back down in main I start the bulk of my program which exists in a do while loop which functions very well for this program. Basically my do while loop says do the main bulk as long as the word entered by the user isn’t zero, if it is zero the do while loop will not run and the program will exit. Here is that line at the end of the do while loop:

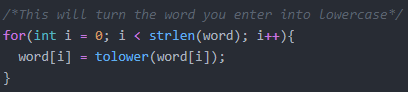


At the top of this do while loop I initialize some variables for count which will count the number of words we suggest, for minimum which will get the distance as long as the distance is less than 40. I also have distance and a flag, the flag ensures that when a word is entered that is not in the wordlist we suggest words.

After this we prompt the user to enter a word to check and fgets that word from standard input into the word character array we created earlier.



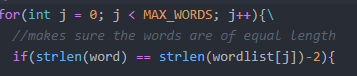
So now that we have the word we turn it into a lowercase word so that we can easily compare it to the wordlist. To accomplish this I use the standard library function tolower which converts words with uppercase characters to lowercase characters.



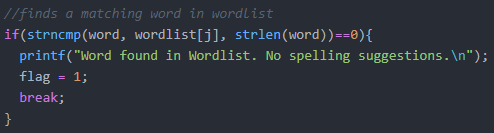
After this we move on to print the word entered by the user in order to clarify if they entered the word that they desired to enter. I also strip off the newline character of the word entered using the string span standard library function which will delimit the ‘\n’ character at the end of the word entered by the user.



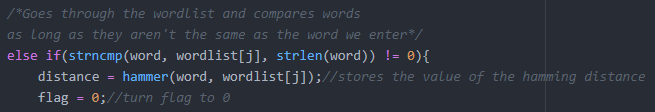
Next up we begin our loop which will run through the wordlist, the first if statement makes sure that the length of the word entered is equal to the length of the word in the wordlist so that we only run compares on words of the same length. We subtract two from the length of the wordlist word in order to get rid of the return character and the newline character.



Below this if statement I do a string compare to compare the word entered to the words of the same length in the wordlist and if the word entered is found to be in the wordlist I don’t suggest any word and set the flag to 1 so that the suggestion doesn't run. I also break the program so that it doesn’t run more than once.



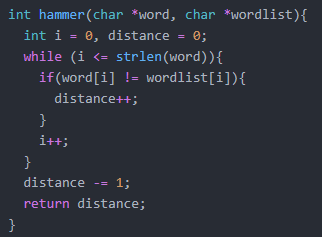
If this last check is unsuccessful and we don’t find the word entered in the wordlist we will get to this else if statement that says if we don’t find the word in the wordlist then we call our hamming distance function which is named hammer. We pass it the word entered by the user and the words that are the same length in the wordlist. We set the flag to zero for use later in the program.



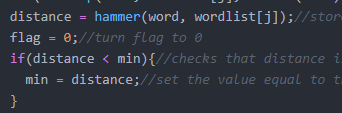
So up in our hammer function we get the two character arrays word and wordlist using pointers to their values in main. Then we create two integers one as an index integer and one to hold the distance we calculate between the two words.

Then we start a while loop that says when the index integer is less than the length of the word entered by the user then compare the letters of the two words and if they are not equal we increase the distance by one. Below that we increase the index integer in order to go through both the word and the letters.

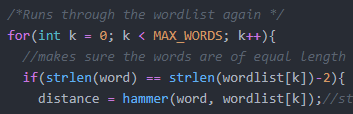
For some reason I was always returning a distance with one more than the actual distance between the two words so I subtract one from the distance and return it. I believe this was because when I passed to the hammer function it tacked on another newline character or possibly because made the while loop as less than or equal to the string length of the word. I could have fixed that and tested but subtracting one from the distance solved the issue.



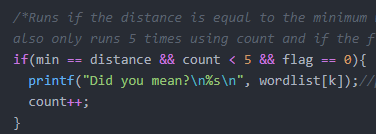
Back down in main we get the value of distance and store it into an integer called distance. Then we set our flag equal to zero which means that we found words with the same length that don’t match the word entered by the user, we will use this later to suggest words. Then we say if the distance is less than the min which we set to forty then the min is equal to the distance. This checks to make sure that we don’t get a hamming distance of greater than 40.



Now I will run through the wordlist again and check that the word entered and the word in the wordlist are equal in length and if they are call my hamming distance function hammer again. This will pass the word entered and all the words of the same length in the wordlist to the hamming distance function and return the distance of all these words.



After this I have another if statement that says if the distance we just found when we run through the wordlist a second time to calculate the hamming distance is equal to the hamming distance we found before. And the count is less than 5 and the flag is equal to zero then suggest a word from the wordlist. We increase the count so that we can suggest five words. Basically if the distance we found before is equal to the distance we find the second time through the wordlist we suggest those words, so we only suggest the words with the lowest hamming distance.

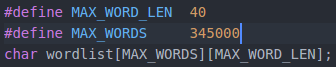


This concludes my process, when we are finished with this if statement the values from the beginning, including the flag, min, and distance all get reset to their original values.

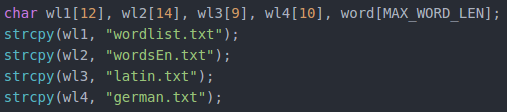
After I completed my program I started working on the extra credit assignments. I began with trying to create an alternate language wordlist and test it against it. I first tried German which worked pretty well but the special characters would print strange on the console. Also the wordlist I found was around 2 million words which made it hard to search through and my program would quit after around 500,000 words.

I finally found a wordlist that worked for German that had around 340,000 words and had to first convert them all to lowercase since some had uppercase as the first character. So I used a program I wrote last summer during our C bootcamp at Whatcom to convert the characters from uppercase to lowercase.

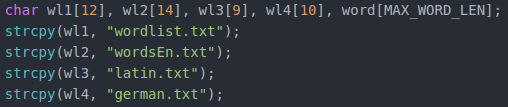
To implement this wordlist I only had to write two additional lines of code and increase my MAX\_WORDS global variable to accommodate the longer wordlist.



Here I added the german.txt text file name to the names of the text files we can use for this program



Once I had tried German I decided to try an easier wordlist, I tried Latin next since it has no special characters. I found a Latin wordlist online and only had to slightly modify my code to get it to work, the lines I added are here:

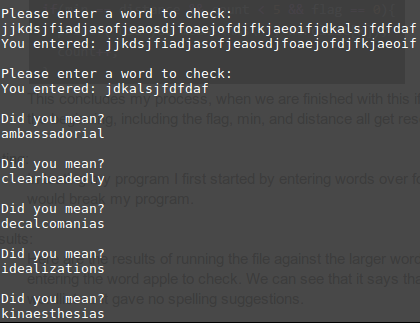


And then below I added to the string compare to accommodate for these new word lists.

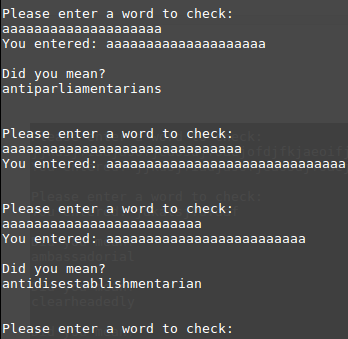


Testing:

In testing my program I first started by entering words over forty characters to see if it would break my program.

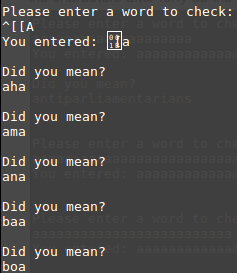


As you can see from the results my program will cut off at forty characters and accept the characters after forty as the string to check and will suggest words based on that.

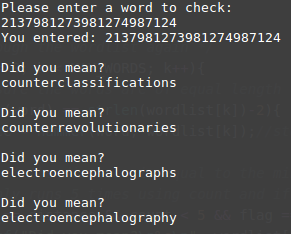


Als it testing I entered words of length 20, 30, and then 25. I found that it would suggest a word for 20 and 25 characters but after 25 it wouldn’t suggest any words. I believe this is because it couldn’t find any words for that word length, since the two strings have to be the same length it wouldn’t suggest any.

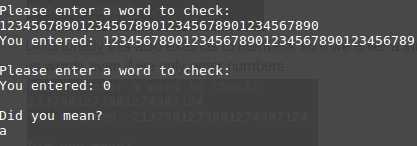
I also found that if I enter a character that isn’t valid it will still suggest words as you can see below, this shouldn’t be happening but I’m not scrubbing input from the user so it’s to be expected that invalid characters can be inputted.



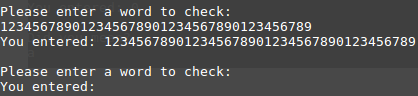
Unfortunetly this also extends to numbers so if we enter a string of numbers it suggest us words even if we only enter numbers.



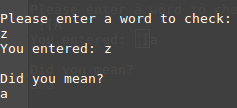
Further testing indicated that my program was actually only checking up to 39 characters and would take the last character entered after the 39th character and assume that the 40th character was our string that we entered and then make suggestions based on that one character.



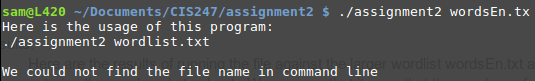
Some more strange behavior I found was that if I entered a 39 character string as numbers my program would think that the user hadn’t entered anything and exit the program. This is likely because I enter the full length of the word that we can enter and when it doesn’t find any words with that length it doesn’t suggest any and goes to the next line. But the fgets has overflowed a newline character or something and enters that into the next check which exits the program.



Testing if it will suggest words at the beginning if we input a character from the end works too since we run through the whole wordlist twice, once to find the lowest hamming distance and again to suggest words with the lowest hamming distance

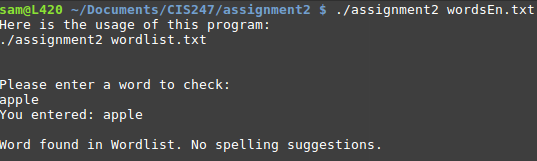


Entering a filename that is not one of the two wordlists supported will tell you that we could not find the filename in the command line

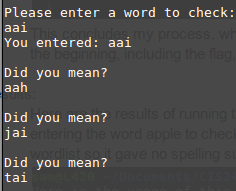


Results:

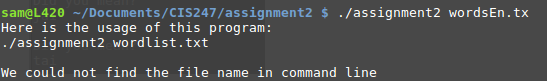
Here are the results of running the file against the larger wordlist wordsEn.txt and entering the word apple to check. We can see that it says that the word was found in the wordlist so it gave no spelling suggestions.



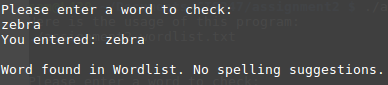
If I enter a word that isn’t in the wordlist it will suggest words, as you can see below.

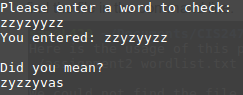


If I enter a file name that isn’t one of the two supplied my program will state that it couldn’t find the file in the command line.

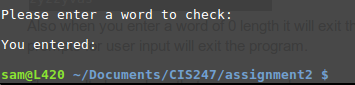


It will go all the way to the bottom of the wordlist



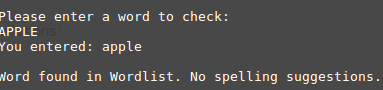


Also when you enter a word of 0 length it will exit the program so pressing enter when it prompts for user input will exit the program.



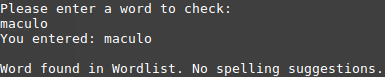
My program will also keep prompting for user input until a word of 0 length is entered.

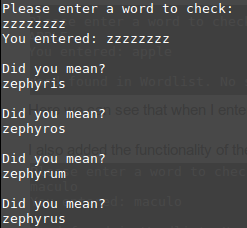
My program also turns all the uppercase to lowercase as I will now demonstrate:



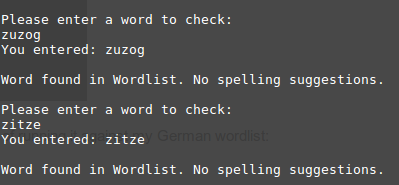
Here we can see that when I enter APPLE all uppercase it will turn it to lowercase.

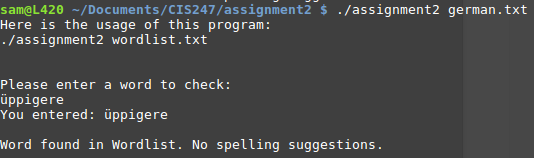
I also added the functionality of the Latin wordlist as I will demonstrate:

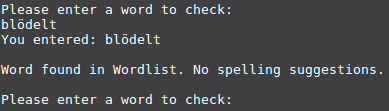




Here are also the results of running it against my German wordlist:







The results here show that my program has accomplished the intent of this assignment.

Conclusions:

Overall this assignment was a very good exercise in practical problem solving. I really enjoyed the intent of this assignment and really enjoyed creating this assignment. I learned a lot about programming to a strict set of guidelines, also about creating bounds to which the assignment must operate. If I had more time I would have liked to do some input scrubbing so that only letters would be accepted into the program. Stripping off newline and return characters taught me a lot about reading in strings and comparing them to input. Also accepting input from users and then using that input to operate the program was something we hadn’t done before and I enjoyed it quite a bit. This assignment really taught me a lot about solving a practical problem using programming.

References and Acknowledgements:

I worked on this assignment alone and only seeked assistance when stuck for more than a day. I referred to the C K&R book when looking for standard library functions to implement in my program.