CS31 Project 5 Report – Joyce Chen

**Notable Obstacles:**

For this project, one of the hardest challenges to overcome was understanding how to work with C strings. As opposed to regular strings, C strings had characters inside an array, making it so that if you wanted to have an array of C strings, you’d need a 2 dimensional array. That alone made it hard to think through how to loop through the array of C strings, find a random word, and be able to compare the C string with the probe words. Furthermore, since C strings were immutable in terms of size, I had to set some boundary cases (which thankfully, were pointed out in the project spec) for if the probe word was longer than the initialized length.

In addition, another difficult part of the project was figuring out how to keep track of characters I already compared the answer and the probe word to. For instance, if the answer word was EGRET and I inputted EERIE, the last E in EERIE should not count towards any points as there are only 2 E’s in EGRET. I figured that because I actually needed to keep track of which characters I checked, I’d need an additional array that mapped to the answer – something that contained a ‘0’ if the character was not checked yet, and a ‘1’ if it was. Eventually, I decided to use the original C strings to mark the checks, instead of creating an auxiliary array which saved up some space. Anyways it was generally just a challenge to think of different probe/answer combinations and how to take care of these cases.

**Description of design**:

For my playOneRound function, I had 3 variables global in the function – the score for golds, score for silvers, and a counter for the number of tries that the user took to get the answer. I then had a while loop that breaks when the score for golds equals the length of the answer string, since that’d mean that the user got the word right. Because the golds and silvers reset with every input, I made sure to set them back to 0 at the beginning of the loop.

Inside the while loop, I basically checked if the inputted word existed and was valid through a series of if-else statements. The last else statement executed when the input was valid, and consisted of the checks for the # of golds and # of silvers. The check is pretty simple – I just had 2 for loops to check whether each individual character in the probe word matched the answer. If they matched, I would set the character in the answer and probe word to be ‘1’ and increase the number of golds/silvers depending on whether the characters were in the same corresponding positions (userWord[i] == word[i]). In the next iterations, I’d check whether the character at “answer” is ‘1’ – if so, I’d skip the check because that’d mean the character already matched to one in the probe word. Finally, at the end of all the checks, I’d output the number of golds and silvers for that probe word and the while loop would reset.

I also had some helper functions like wordExists and isValidWord to help me check for valid probe word inputs in the playOneRoud function. The wordExists function checks whether the word that the user inputted exists in the text file; the isValidWord function checks whether the inputted word is 4-6 characters in length.

Finally, my main function consisted of the actual game. I’d prompt the user for the # of rounds and create a corresponding for-loop for the game. Some global variables within the function included the minimum, maximum, total, and average scores. Inside the for-loop (each iteration represents 1 game round), I would first call the playOneRound function with a random integer that maps to the answer in the array. At the end of one round, I would update the minimum and maximum scores, and output the cumulative average.

**Pseudocode:**

main function:

Declare words array

Call getWords function to store words from txt file into array & get total # of words

If # of words is less than 1 or greater than max word limit (9000):

Print out error message

Return 0

Declare and initialize “rounds,” “total,” “average,” “minScore,” “maxScore” variables

Prompt user for # of rounds

If # of rounds is negative

Print error message

Return 0

Incrementing from 1 → # of rounds (inclusive):

Print out round number

Print out message with # of hidden words

Initialize random integer between 0 and (# of total words) - 1

Initialize score variable by calling playOneRound

If score is less than minScore:

Set minScore to score

If score is greater than maxScore:

Set maxScore to score

If score is equal to 1:

Print out “You got it in 1 try”

Otherwise:

Print out “You got it in n tries”

Add score to total

Calculate average by dividing total with current # of rounds

Print out average, minimum, maximum message

Return 0

playOneRound function:

Check for valid arguments, if integers are negative or index is greater than max limit

Return -1

Declare constant integer “length” as size of the C string

Declare and initialize golds, silvers, count variables to 0

While the golds score is not equal to length:

Declare C string to store answer

Copy the random word from words array into answer string

Set golds and silvers scores to 0

Declare C string to store user input

Prompt user for probe word and store in C string

If length of probe word isn’t between 4-6, or if it’s not valid:

Print out error message

Otherwise, if probe word doesn’t exist in the word array:

Print out error message

Otherwise:

Incrementing across the probe word C string:

If the char in the probe word matches the char in answer:

Add 1 to golds score

Set characters in answer string & probe word to ‘1’ Incrementing across the probe word C string:

Incrementing across the answer C string:

If the chars at answer & probe word is not equal to ‘1’ and

the char at answer matches the char at probe word:

Add 1 to silvers score

Set characters in answer string & probe word to ‘1’

If the golds score is not equal to the length of the answer string

Print out golds & silvers scores onto console

Add 1 to count

Return count

wordExists function:

Incrementing from 0 → # of valid words in the array of words from word.txt:

If the input word equals word at that index in the array:

Return true

Return false

isValidWord function:

Incrementing through each character in the C string:

If the character is uppercase or not a letter:

Return false

Return true