Nathan Tung
Com Sci 31 (Lec 2)
November 12, 2011

CS31 Project 5 Report

a. The biggest obstacle in this project was deciding how to structure the program, particularly because no global variables were allowed. Every function had to operate independently based on parameters alone. In the end, I coded the main function to do practically everything (except for accepting the trial word and comparing it to the mystery word) in a rather sequential basis.

   The manageOneRound function was also slightly overwhelming, so I coded two extra functions (boolean-function invalidWord and int-function charsInCommon) to aid the process. At first, when the code prompted the user for the trial word, anything greater than 6 characters would cause the program to crash. To fix this, I changed the maximum allowed characters to 100, including the zero bit (which is enough, according to the professor and project specs); but when stored, anything beyond the maximum allowed word length would be cut off. This way, the program would run and continue prompting for the trial word without errors.

   Another problem I had was comparing the C string characters of the trial word with the mystery word. The solution was to do an embedded for-loop to check for any identical characters. If any matched, the character in the trial word's C string (a copy of the real array, so as to preserve the original content) would be changed to a 'Z' to prevent repeat counting; then the inner loop would break to check the next character. Lastly, I had to determine how to loop the prompt for trial words. I settled with a do-while loop that would stop once the trial word matched the mystery word, simplifying the program to a clean but functional structure.

b. The main function initiates the C string wordList to hold up to 10000 C strings of 7 characters each (including the zero bit), according to global constants. The function loadWords is then called to fill up the wordList array. The user is then prompted for the number of rounds to play. If no words are loaded or the number of rounds to play is not positive, an error is given. Counters of minimum, maximum, and average tries are then initialized to zero; zero is an impossible number of tries, so we know that no rounds have been played yet. Counters for both single and accumulated numbers of tries are also initialized to zero. A for-loop then runs for the specified amount of rounds. Each time, a random integer (seeded earlier) less than the wordList length is generated (and later used to determine the mystery word). The number of tries is set to the value given when manageOneRound is called. The minimum, maximum, and average number of tries are recalculated after each iteration of the for-loop and presented to the user.

Nathan Tung
Com Sci 31 (Lec 2)
November 12, 2011

The function for manageOneRound first checks for errors. Then it initializes the number of tries to zero and determines the mystery word based on the parameter-passed index and array. A C string is initialized for trial words. Then a do-while loop (satisfied only when the trial word matches the mystery word) prompts the user for the trial word and checks if it is valid – that is, if it has between 4 to 6 characters long and contains only lowercase alpha characters. If there is no validity error, the amount of letters in common between the words is calculated and printed out. The number of tries to successfully guess the mystery word is returned to the main function.

These functions (briefly summarized below) are called in the program. The specs provide information on the implementation of loadWords.

main():

- Initiate the word list with proper length
- Call loadWords to fill up the word list
  - If no words are loaded, output error and terminate program
- Initiate and prompt user for number of rounds
  - If number of rounds is not positive, output error and terminate program
- Initiate minimum, maximum, and average number of tries to zero
- Seed rand() with the system time
- Initiate single and accumulative number of tries to zero
- For each round:
  - Initiate random index for the mystery word and pass to manageOneRound
  - Set number of tries to the integer returned by manageOneRound
  - If both minimum and maximum tries are zero (meaning the first round)
    - Set both variables to number of tries
  - If number of tries is greater than the maximum
    - Update the maximum
  - If number of tries is less than the minimum
    - Update the minimum
  - Add number of tries to the accumulative number of tries variable
  - Calculate the average number of tries with two decimal points
  - Output the average, minimum, and maximum number of tries

manageOneRound():

- If word list has no words or index of mystery word is out of bounds
  - Terminate the program
- Initialize number of tries to zero
- Initialize mystery word C string based on the index passed in parameter
- Initialize guess word C string to hold 100 characters (including zero bit)

Nathan Tung
Com Sci 31 (Lec 2)
November 12, 2011

Do-While loop (until guess word equals mystery word):
    Prompt user for guess word
    Increment the number of tries
    If guess word returns true for invalidWord()
        Output error message and continue next loop iteration
    For each element in word list:
        If guess word does not match any
            Output error message and continue loop
    If guess word and mystery word are not identical
        Output number of common characters through charsInCommon()
Return number of tries taken to guess mystery word

charsInCommon():
Initialize a temporary C string to hold 100 characters (including zero bit)
Copy guess word in parameter to temporary C string (to preserve original content)
Initialize count for common characters to zero
For each character in mystery word before zero bit
    If it matches any character in guess word before zero bit
        Increment count
        Set that character in guess word to 'Z' to prevent double counting
        Break inner for-loop to check next character in mystery word
Return total number of common characters

invalidWord():
For each character in parameter word before zero bit
    If any character is not alpha or is uppercase, return true
If the parameter word is less than 4 or greater than 6 in length, return true
Otherwise, return false