CS19003 : Programming and Data Structures Laboratory

Assignment 6 : Pointers, Dynamic Allocations, Multi-Dimensional Arrays and Strings in C

Date: 08-Jun-2021

Problem Statement:

There were two letters found in the place where the two murders happened. These are very crucial evidences and may indicate several vital clues in the investigation process. However, Detective Bakshi and Ajit had a hard time in understanding the letter. Though both of these were written in English, but there were many spelling errors and incomplete wordings in it. Moreover, some of the words are not readable due to the fact that the ink of fountain pen used to write these got washed out before the team of investigators got hold of the letters. Mr. Bakshi told Ajit to involve some linguist who can help in enhancing the readability of the letter. They primarily want two things – (i) identify the correct English words present in the letter and (ii) in case of spelling mistakes or incomplete wordings, suggest words in close proximity from a dictionary of given words.

Can you write a C-program that performs this spelling check from a dictionary of words and may suggest suitable options available from the dictionary? In particular, your program will do the following:

- Take from user as input the number of words in dictionary (`noWords`) and the maximum length of an word in the dictionary (`maxWordLen`). With the help of these parameters, *dynamically allocate the memory* for the dictionary of words as a two-dimensional array (i.e. array of strings), defined as `char **dictionary`.
- Then, take from user all the (`noWords`) words (one by one each in a new line) and store into dictionary memory. All words are to be entered in alphabetic (lexicographic) order and in lowercase.
- Then, take from user a word, i.e. a string (defined as `char *word`) in lowercase, to check for its spelling and suggestions (in case of mismatch) floated from dictionary.
- If the given `word` can be found inside the `dictionary`, then report the presence of the same.
- In case the given `word` *cannot* be found inside the `dictionary`, you need to showcase some suggestive words from dictionary as per the following rules:

  Rule-1: If the given `word` is a substring of any word present in the `dictionary` or vice versa, and the lengths of these two strings are within 2 of one another.

  Hint: A string is a substring of another one if all of its letters appear contiguously (in the same order) in the second string. For example, "bound" is a substring of "homebound" and "ire" is a substring of "firefly", but "car" is NOT a substring of "camera" because the letters 'm' and 'e' are in between the letters 'a' and 'r' in "camera". To enable this rule, you may first write a function, `int substring ( char *shortstr , char *longstr )`, which returns 1 (true) if the first string, `shortstr`, is a substring of the second string, `longstr`; and 0 (false) otherwise.

  Rule-2: If the given `word` is a subsequence of any word present in the `dictionary` or vice versa, and the lengths of the two strings are within 2 of one another. (This would get rid of the "car", "camera" case, for example.)

  Hint: A string is a subsequence of another string if all the letters in the first string appear in the second string, in the same ordering, but not necessarily contiguously. For example, "car" is a subsequence of "camera", since the letters 'c', 'a', and 'r' appear in that order (but not contiguously) in "camera". Similarly, "hikes" is a subsequence of "chickens", but "tale" is NOT a subsequence of "wallet" because the letter 't' occurs AFTER the letter 'a' in "wallet". To enable this rule, you may first write a function, `int subsequence ( char *shortstr , char *longstr )`, which returns 1 (true) if the first string, `shortstr`, is a subsequence of the second string, `longstr`; and 0 (false) otherwise.

  Rule-3: If the given `word` is a permutation of any word present in the `dictionary` or vice versa. (Only try this test if the two strings are of the same length.)

  Hint: A permutation of letters is simply a different ordering of those letters. For example, "care" is a permutation of "race" and "spill" is a permutation of "pills", but 'apple" is NOT a permutation of "pale" because the letter 'p' appears only once instead of twice in "pale". A natural consequence of this definition is that two strings can only be permutations of one another if they are both the same length. To enable this rule, you may first write a function, `int permutation ( char *string1 , char *string2 )`,

which returns 1 (true) if both the strings, `string1` and `string2` are permutations of each other; and 0 (false) otherwise.

Rule-4: If the given `word` differs from any word present in the `dictionary` in less than 3 characters. (Only try this test if the two strings are of the same length.)

Hint: When we are comparing two strings of equal length, we can define a term called *match-score* which is simply the number of corresponding letters in which the two strings disagree. For example, the match-score between "drink" and "blank" is 3 because the first three letters do not match. The match-score between "marker" and "master" is 2 because third letter ('r' vs. 's') and forth letter ('k' vs. 't') do not match. To enable this rule, you may first write a function, `int matchscore ( char *string1 , char *string2 )`, which returns score, i.e. number of mismatches. Here, the two strings `string1` and `string2` must be of the same length and should only contain lowercase letters.

- Print all the suggestions (words) from `dictionary` that qualify the above rules in an alphabetic / lexicographical order (one by one each in a new line).
- Finally prompt the user to enter [Y/N] (meaning yes/no) so that (s)he can continue searching for another word inside the same `dictionary`. If user selects 'yes' option, repeat the word searching process and produce appropriate suggestions (as per rule); otherwise exit from the program.

Note: You may need to write your own function, `int stringLength ( char *string )`, to determine the length of `string` and any other assistive string functions by yourself. **Usage of string library functions from ⟨string.h⟩ is NOT allowed!**

Example Execution Details:

```
++ Welcome to Mini Spell Checker ++

-- Load Dictionary Words --
30 15

append
back
batch
beach
beech
belch
bench
bitch
branch
bunch
butch
dependence
dependent
happen
indecent
indeed
independence
independent
institute
institution
leech
march
match
milch
mooch
mulch
munch
situation
tech
yield

-- Dictionary of Words Loaded --

** Enter Your Word: institute

** GREAT! Your word "institute" is in Dictionary!

-- Would You Like to Enter Another Word? [Y/N]: Y
```

```
** Enter Your Word: btech

** SORRY! Your word "btech" is NOT in Dictionary!
** Some Possible Suggestions:
batch
beach
beech
belch
bench
bitch
bunch
butch
leech
tech

-- Would You Like to Enter Another Word? [Y/N]: Y

** Enter Your Word: mtech

** SORRY! Your word "mtech" is NOT in Dictionary!
** Some Possible Suggestions:
beech
leech
march
match
milch
mooch
mulch
munch
tech

-- Would You Like to Enter Another Word? [Y/N]: Y

** Enter Your Word: independ

** SORRY! Your word "independ" is NOT in Dictionary!
** Some Possible Suggestions:
indecent
indeed

-- Would You Like to Enter Another Word? [Y/N]: Y

** Enter Your Word: independant

** SORRY! Your word "independant" is NOT in Dictionary!
** Some Possible Suggestions:
independent

-- Would You Like to Enter Another Word? [Y/N]: Y

** Enter Your Word: happend

** SORRY! Your word "happend" is NOT in Dictionary!
** Some Possible Suggestions:
append
happen

-- Would You Like to Enter Another Word? [Y/N]: Y

** Enter Your Word: situation

** GREAT! Your word "situation" is in Dictionary!

-- Would You Like to Enter Another Word? [Y/N]: N

++ Thank You for using Mini Spell-Checker ++
```

Submit a single C source file. Do not use global/static variables and string library functions.