Homework 3, TCSS 333 Spring 2016
Due: Friday, May 13, 9:00 a.m.

OBJECTIVE

The objective of this assignment is to give you more practice with using functions, strings, string arrays, pointers, and dynamic allocation of data.

ASSIGNMENT SUBMISSION

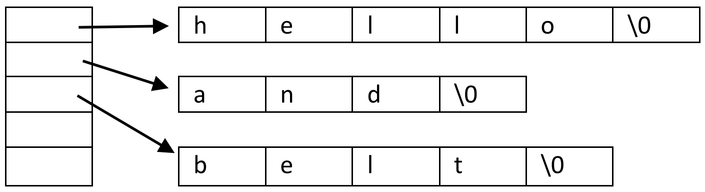
To get credit for this assignment, you must

- ✓ write a program in C using functions and dynamic memory allocations (80%)
- ✓ provide function interface comments (10%)
- ✓ write a test plan (10%)
- ✓ submit your files through Canvas exactly as instructed (naming, compatibility, etc.)
- ✓ submit your assignment on time

PROBLEM STATEMENT

Write a program that allows the user to spell check words. The program will receive the name of a text file containing the dictionary and the word to spell check, both entered by the user as command line arguments (explained below).

Before trying to spell check the word, your program is to make sure that the valid number of arguments is entered. If an incorrect number of arguments is entered, the program prints an error message to the screen and terminates. If the correct number of arguments is entered, the program compares the user's word to the dictionary entries stored in the character pointer array. The code that iterates over a typical Unix dictionary file and reads in one word at a time is provided in Canvas. You need to use this code to store each word that the code reads into a character pointer array where each dictionary entry takes up exactly as much space as the length of the word, plus the null character (this is the first place where dynamic memory allocations should be used). For example, after reading in all the dictionary words, the dictionary array should look similar to this picture:



If the program finds the exact match for the entered word, the program reports it found the exact match and finishes. If no exact match is found, then the program starts looking for suitable replacements and collects them into a character pointer array set up in the same exact fashion as the dictionary array (this is the second place where dynamic memory allocations should be used). The words collected into the dictionary array must follow the rules described below. Finally, the program reports top 10 possible replacements to the user (note that the program may actually find more or less than 10 suggestions; if there are less than 10 suggestion, then the program reports all of them).

The replacement suggestions are to be collected and organized as follows:

1. all dictionary words of the same length as the user's word where only one letter substitution anywhere in the word would create a dictionary word, e.g. if a user enters `manies`, then the top replacement suggestions would include `manias` and `mantes`
2. all dictionary words that are longer by one letter only from the user's word and where the user's word constitutes the first letters of the dictionary word, e.g. if a user enters `belo`, then replacement suggestions would include `below` (right after `bell` and `belt` generated by the prior bullet's rule)

3. all dictionary words that are shorter by one letter only from the user's word and where the dictionary word constitutes the first letters of the user's word, e.g. if a user enters `bel`, then replacement suggestions would include `be`
4. if the user's word is longer than 5 characters, all dictionary words of the same length as the original where two letter substitution anywhere in the user's word would create a dictionary word, e.g. if a user enters `manies`, then replacement suggestions include `movies` and `varies`
5. all dictionary words in which the first length-1 letters of the user's word appear in the beginning of the dictionary's word and the user's length-1 substring constitutes at least 50% of the dictionary's word, e.g. if a user enters `belo`, then replacement suggestions include `belch` and `belle`
6. all dictionary words in which the first length-1 letters of the user's word appear anywhere in the dictionary's word and the user's length-1 substring constitutes at least 50% of the dictionary's word, e.g. if a user enters `belo`, then replacement suggestions include `Nobel` and `libel`

These are sample runs of the program:

```
---
./a.out  dictionary.txt  hello
hello found
./a.out  dictionary.txt
Wrong number of arguments
./a.out  dictionary.txt  belo
Suggestions:
bell
belt
below
belay
belch
belie
belle
bells
belly
belts

---
```

Program Specs

- Your program is to be contained in a single c file named `<netid>_hw3.c`
- Your program has to follow basic stylistic features, such as proper indentation (use whitespaces, not tabs), meaningful variable names, etc.
- Your program must use functions and dynamically allocated string arrays
- You are not allowed to use global variables
- **Your program must compile in gcc gnu 90 – programs that do not compile will receive a grade of 0**
- Your program should include the following comments:
    - Your name at the top
    - Comments explaining your logic
    - Function comments, as specified in the *function comments* section below
    - If your program does not run exactly as shown above, explain at the top how to run your program – you will not receive full credit but at least you will receive some credit rather than none

Command-line arguments

Command-line arguments are discussed in your textbook in chapter 3. However, better examples are provided at:

- http://www.tutorialspoint.com/cprogramming/c_command_line_arguments.htm
- http://www.thegeekstuff.com/2013/01/c-argc-argv/

Note that your program will receive the input as 2 string arguments.

Function Comments

In case you have not heard of design by contract, the general idea behind design by contract is that the software designer defines precise and verifiable interface specifications for software components. In case of functions, this translates to a function header that specifies the function's purpose, as well as documented preconditions and postconditions for a function. These comments should be written next to function prototypes but in this case, you need to write them by function definitions.

Preconditions focus on arguments/parameters passed to functions – any assumptions that a function does regarding the state of the parameter need to be written as assertions. Postconditions focus on the effect the function holds on the state of computation, so it should describe, as assertions, the value return statement and the state of parameters passed by reference. In addition, write a comment by each parameter to denote a data flow of each parameter: in, out, or in/out. A flow of parameter is *in*, if it is passed by value or as a const reference and a function only uses it for its internal processing. A flow of parameter is *out*, if it is passed by reference and a function only uses it to replace its value. A flow of parameter is *in/out*, if it is passed by reference and a function uses its contents before replacing them with new values.

Some Examples

```
// Prints a header to standard output
// pre:   none
// post: none
void printMessage() {
       puts("some code that prints something");
}

// Prints parameter values to standard output
// pre:   n1 assigned, n2 assigned
// post:  none
void printMessage(/* in */ int n1, /*in*/ int n2) {
       printf("%d %d\n", n1, n2)
}

// Swaps parameter values
// pre:   n1 and n2 assigned
// post:  *n1 == *n2@entry and *n2 == *n1@entry
void swap(/*inout*/ int* n1, /*inout*/ int* n2) {
       int temp = *n1;
       *n1 = *n2;
       *n2 = temp;
}

// Fills array parameter with data from the user and returns array count of even
// numbers
// pre:   array allocated, size == array length
// post:  array filled with user data, count of even numbers returned
int fillArray( /*out*/ int array[], /*in*/ int size) {
       int i, count = 0;
       printf("enter %d integer values: ", size);
       for (i = 0; i < size; i++) {
              scanf("%d", &array[i]);
              if (!(array[i] % 2))
                     count++;
       }
       return count;
}
```

This last example does not follow good design practices as one should not mix returning via the return statement and returning via function parameters at the same time. If you are only to return one value, use value-return functions. If you need to return multiple items, use pass-by-reference for all of them.

Extra Credit (15%)

Open-ended – extend the program in some fashion (look into other string matching algorithms) – explain in comments at the top of your code.

Program Submission

On or before the due date, use the link posted in Canvas next to Homework 3 to submit your test plan and your C code. Make sure you know how to do that before the due date since late assignments will not be accepted. Valid documentation file formats are: pdf, jpg, gif, png. Valid program format: a single file .c