1806ICT Assignment – Milestone 2

This assignment milestone is worth 30% of your final grade and is due for submission by 11:59pm on Friday, 6th October. You must do this assignment milestone by yourself (including writing ALL the C code) and the normal Griffith University rules for plagiarism apply.

Part 1:

In this part of the assignment, you will build a C program that bends the rules of a game to beat its human opponent. The rules of the game are:

1. One player chooses a secret word, then writes out the number of dashes equal to the word length.
2. The other player begins guessing letters. Whenever they guess a letter contained in the hidden word, the first player reveals each instance of that letter in the word. Otherwise, the guess is wrong.
3. The game ends either when all the letters in the word have been revealed or when the guesser has run out of guesses.

Fundamental to the game is the fact the first player accurately represents the word they have chosen. That way, when the other players guess letters, they can reveal whether that letter is in the word. But what happens if the player doesn't do this? Suppose that you are playing the game and it's your turn to choose a word, which we'll assume is of length four. Rather than committing to a secret word, you instead compile a list of every four-letter word in the English language. For simplicity, let's assume that English only has a few four-letter words, all of which are reprinted here:

ALLY BETA COOL DEAL DICE FLEW GOOD HOPE IBEX

Now, suppose that your opponent guesses the letter 'E.' You now need to tell your opponent which letters in the word you've "picked" are E's. Of course, you haven't picked a word, and so you have multiple options about where you reveal the E's. Here's the above word list, with E's highlighted in each word:

ALLY BETA COOL DEAL DICE FLEW GOOD HOPE IBEX

Every word in the word list falls into one of four "word categories:"

- ----, containing the word ALLY, COOL, and GOOD.
- -E--, containing BETA and DEAL.
- --E-, containing FLEW and IBEX.
- ---E, containing DICE and HOPE.

Since the letters you reveal have to correspond to some word in the word list, you can choose to reveal any one of the above four categories. There are many ways to pick which category to reveal – perhaps you want to steer your opponent toward a smaller category with more obscure words, or toward a larger category in the hopes of keeping your options open. If we adopt the larger category approach and always choose the largest of the remaining word categories you would pick the category ----. This reduces your word list down to ALLY COOL GOOD and since you didn't reveal any letters, you would tell your opponent that their guess was wrong.

Your program should do the following:

1. Accept the title of the dictionary file, the word length and the number of guesses from the command line.
2. Read the file dictionary.txt to get the words (of the correct length) to be used for the game. To make it easier, do not include words with duplicate letters.
3. Play the game as described below:
    1. Print out how many guesses the user has remaining, along with any letters the player has guessed and the current blanked-out version of the word and the number of possible words remaining.
    2. Prompt the user for a single letter guess, re-prompting until the user enters a letter that they have not guessed yet. Make sure that the input is exactly one character long and that it's a letter of the alphabet.
    3. Partition the words in the dictionary into groups by word category.
    4. Choose a word category and remove all words from the word list that aren't in that category. If the word category doesn't contain any copies of the letter, subtract a

remaining guess from the user.

5. If the player has run out of guesses, pick a word from the word list and display it as the word that the computer initially "chose."
6. If the player correctly guesses the word, congratulate them.

A sample run of the program could be:

```
./a.out dictionary.txt 6 10
______  Enter letter: a
Guess 1/10, Words Left 3595, Letters used = a
______  Enter letter: 4
4 is not a letter
______  Enter letter: e
Guess 2/10, Words Left 1154, Letters used = ae
____e_  Enter letter: e
e already used
____e_  Enter letter: i
Guess 3/10, Words Left 643, Letters used = aei
____e_  Enter letter: o
Guess 4/10, Words Left 259, Letters used = aeio
_o__e_  Enter letter: u
Guess 5/10, Words Left 209, Letters used = aeiou
_o__e_  Enter letter: b
Guess 6/10, Words Left 175, Letters used = aeioub
_o__e_  Enter letter: g
Guess 7/10, Words Left 151, Letters used = aeioubg
_o__e_  Enter letter: d
Guess 8/10, Words Left 79, Letters used = aeioubgd
_o__e_  Enter letter: r
Guess 9/10, Words Left 32, Letters used = aeioubgdr
_o__er  Enter letter: t
Guess 10/10, Words Left 18, Letters used = aeioubgdrt
Word was confer
```

Part 2:

This C program accepts numbers on the command line and finds all possible assignments of upper-case letters to each digit of numbers so that the resultant words are in an upper-case version of *dictionary.txt*. Note that, in a valid solution, a letter can only have one numeric value in the range 0 .. 9 associated with it and a digit can only be assigned to one letter.

Given r distinct digits in the sequence of numbers there 26! / (26 – r)! possible permutations of assigning letters to numbers.

Some examples are:

Input: ./a.out 9567 1085 10652.

Output: Found 68013 solutions, CPU Time = 0.428983, Dictionary Look Ups = 5410962
Possible Permutations = 6.2990928e+10, Actual Completed Permutations = 3494100 (0.005547%)

Note: A sample solution where D = 7, E = 5, M = 1, N = 6, O = 0, R = 8, S = 9 and Y = 2 is: SEND MORE MONEY.

Input: ./a.out 67432 704 8046 97364 173546
Output: Found 973 solutions, CPU Time = 0.638096, Dictionary Look Ups = 8210204
Possible Permutations = 1.9275224e+13, Actual Completed Permutations = 178398 (0.000001%)

Note: A sample solution where A = 7, E = 6, F = 8, H = 2, I = 0, N = 1, R = 4, T = 3, U = 5 and W = 9 is: EARTH, AIR, FIRE, WATER and NATURE.

Input: ./a.out 127503 502351 3947539 46578 4623971

Output: Found 0 solutions, CPU Time = 0.695170, Dictionary Look Ups = 8149500
Possible Permutations = 1.9275224e+13, Actual Completed Permutations = 0 (0.000000%)

Input: ./a.out 12345 54321

Output: Found 104 solutions, CPU Time = 0.757589, Dictionary Look Ups = 7899225
Possible Permutations = 2.8405900e+07, Actual Completed Permutations = 7893600 (27.788593%)

Input: ./a.out 0110

Output: Found 12 solutions, CPU Time = 0.000126, Dictionary Look Ups = 650
Possible Permutations = 3.8004924e+07, Actual Completed Permutations = 650 (0.001710%)

Marking

This assignment is worth 30% of your final grade. The assignment will be marked out of 100 and marks will be allocated as follows:

1. Quality of C code (e.g. efficiency, indentation, appropriate naming conventions, use of functions, use of appropriate statements, …) : **20 marks**
2. Implementation of Part 1 : **50 marks** (The sample program has 10 functions (including main()), 83 semi-colons and 148 lines of code)
3. Implementation of Part 2 : **30 marks** (The sample program has 11 functions (including main()), 167 semi-colons and 230 lines of code)

Please note that all submitted assignments will be analysed by a plagiarism detector that is specifically designed for assignment submissions containing C source code.