Assignment 2

Pedram Pasandide

Due Date: 27 July

You have been hired as a programmer by a company that specializes in online gaming. The company is developing a multiplayer word-guessing game where players have to guess a secret word within a limited number of attempts. Something like Hangman. The server generates a random secret word at the start of each game and sends it to the players.

The random word is picked from a dictionary, `words_alpha.txt`, including more than 370k words (Reference). This file contains only [[:alpha:]] words (words that only have letters, no numbers or symbols). Other files you need to work with are:

`main.c`: The main program file that includes the necessary header files and uses the functions `readWords` and `getRandomWordWithMinLength` from `readFile_randWorld.c`. It initializes the `words` string array, reads words from "words_alpha.txt", and selects a random word from the array based on the minimum letter that should be in the word (`MIN_RAND_WORD_LENGTH` which is defined in `constants.h`).

`constants.h`: The header file that defines all used constants, including `MAX_ATTEMPTS` and `MIN_RAND_WORD_LENGTH`. `MAX_ATTEMPTS` is the maximum attempts allowed in Hangman game. You will not change anything here!

`readFile_randWorld.h`: The header file containing **function declarations** for reading words from a file, selecting a random word, and freeing allocated memory. You do not know how this reading words from the text file works. You just need to use it! You will not change anything here!

`readFile_randWorld.c`: The implementation file containing the definitions for the functions declared in `readFile_randWorld.h`. It includes `constants.h` for using the defined constants. Again you won't need to know how it works, because it is called in the `main.c` and you get the output there. You just need to use the outputs in `main.c`. You will not change anything here!

To compile these files, make sure they are all in the same directory. In the Terminal enter the following command:

```
gcc -g main.c readFile_randWord.c -o main
```

Don't forget you need to write the Makefile for this assignment and this is just an example. This command makes the executable file named `main`, by linking two source code `main.c` and `readFile_randWord.c`. The flag `-g` is used in case you want to see the intermediate results using GDB or LLDB. To run the program, simply do `./main`. My suggestion is put some breakpoints different parts of the code, and see how it works before we start!

a. (**8 points**). Your task is to write a C program that implements the core logic for the word-guessing game. The server informs the player about the length of the secret word using ”-” character to represent each letter, the same as Hangman game! The player repeatedly guesses a letter of the word within a limited number of attempts (`MAX_ATTEMPTS` equal to 5 is defined in `constants.h`). After each guess, the server should provide feedback to the player, revealing the correctly guessed letters and their positions. If the player guesses the word correctly within the given attempts, they win the game; otherwise, they lose.

Tips about the algorithm: in the `main.c` take out the following section from comment, it should be executable line like:

```
// ############################################################
// ###################### Part A ##########################
// ############################################################
// Call the word guessing game
bool gameResult = playWordGuessingGame(randomWord);

if (gameResult) {
 printf("Congratulations! You guessed the word correctly: %s
    \n", randomWord);
} else {
 printf("Out of attempts. The word was: %s\n", randomWord);
}
```

Here the function `playWordGuessingGame` is called and it returns a `boolean` which is `true` or `false`. If the user was able to guess the all the letter, the function must return in `true`. You must implement the function `playWordGuessingGame` in `functions.c` and declare the definitions in `functions.h`. The general format of your `functions.c` will be like:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#include "constants.h"
#include "functions.h"

bool playWordGuessingGame(const char *randomWord)
{
 <some of your code if necessary>

 while (attempts < MAX_ATTEMPTS)
```

```
    {
     <here use scanf(" %c", &guess) to recieve input from user>

     <you can use strcmp function to compare two string!>

     <if all letters were guessed here use return true;>
    }
    <if the run reaches here it means after MAX_ATTEMPTS...>
    <the letters are not found, so use return false;>
   }
```

`"constants.h"` is included at the top, to make sure we can have access to values defined in this file. Like the format in `main.c`, the only input here is `randomWord` as a constant, which in your code, you need to compare it with the guessed letters and also you need the length of this random word to be shown to user! What I see in the terminal must be like:

```
The server has generated a random word. You have 5 attempts to guess the word.
Attempt 1: l
- - - - - - - - - - - l - - -
Attempt 1: s
- - - - - s s - - - - l - s -
Attempt 1: q
- - - - - s s - - - - l - s -
Incorrect guess. Try again.
Attempt 2: e
- - - - e s s - - - - l - s -
Attempt 2: a
- - - - e s s - - - a l - s -
Attempt 2: g
- - - - e s s - - - a l - s -
Incorrect guess. Try again.
Attempt 3: f
- - - f e s s - - - a l - s -
Attempt 3: c
c - - f e s s - - - a l - s -
Attempt 3: m
c - - f e s s - - - a l - s m
Attempt 3: n
c - n f e s s - - n a l - s m
```

```
Attempt 3: t
c - n f e s s - - n a l - s m
Incorrect guess. Try again.
Attempt 4: p
c - n f e s s - - n a l - s m
Incorrect guess. Try again.
Attempt 5: o
c o n f e s s - o n a l - s m
Attempt 5: i
c o n f e s s i o n a l i s m
Congratulations! You guessed the word correctly: confessionalism
```

Warning! You will only submit `functions.c` and `functions.h`. For the other files I use the default versions you have received. DO NOT change anything in other files. So far the only changes you have done in `main.c` are: **1.** uncommenting the section **Part A**. **2.** Including `"functions.h"` at the top of `main.c`.

b. (**12 points**). This part we want to make the guessing more automatic. Before the last possible attempt (`MAX_ATTEMPTS - 1`), the program must compare the letters that so far are guessed, with the whole dictionary (the array `words`), and see what potentially could be the random word. Your `playWordGuessingGameAutomatic` function must be inside `functions.c` and the declration must be in `functions.h`. This function in `functions.c` must have the following format, so I can call it with this format from `main.c`.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#include "constants.h"
#include "functions.h"

bool playWordGuessingGame(const char *randomWord)
{
 <your code for function part A>
}


// numSuggestion: is the maximum number of possible word that
```

```
     can be printed
// I have set this to 50 to make sure the Terminal is not
   gonna be messy
// numWords: is the number of words in words array
// words is the array of string
// with each row one word and the columns are each string
   length

bool playWordGuessingGameAutomatic(const char *randomWord,
   char **words, int numWords, int numSuggestion)
{
 <the same algorithm in Part A>
 < until: attempts == MAX_ATTEMPTS - 1>
 <your code goes here>
}
```

What I must see in the Terminal when the the automatic version starts:

```
The server has generated a random word. You have 5 attempts to guess the word.
Automatic Version.

The server has generated a random word. You have 5 attempts to guess the word.
Attempt 1: er
- e - - - e - - e - - - - - -
Attempt 1: - e - - - e r - e - - - - - -
Attempt 1: o
- e - - - e r - e - - - - - -
Incorrect guess. Try again.
Attempt 2: d
- e - - - e r - e d d - - -
Attempt 2: q
- e - - - e r - e d d - - -
Incorrect guess. Try again.
Attempt 3: q
- e - - - e r - e d d - - -
Incorrect guess. Try again.
Attempt 4: q
- e - - - e r - e d d - - -
Incorrect guess. Try again.
```

```
The number of possible words = 1
Here is up to the first 50 possible words:

featherbedding,

Attempt 5:
```

Here after the 4th attempt, I have `- e - - - e r - e d d - - -` saved as a string (`guessed`). If `numWords` is the total number of words, from zero to `i < numWords` I will check which check which word has the same number of letters. You can used `strlen(words[i])` to get the length of each word. Then I compare each letter of `guessed` with `words[i][j]` to make sure if they have the same order and position. In this example, the terminal is telling me after:

```
Attempt 4: q
- e - - - e r - e d d - - -
Incorrect guess. Try again.
```

There is only one word between over 370k which has the same format, and it is `featherbedding`. If I keep typing the rest of letter which are not guessed yet, I must see in the terminal:

```
Attempt 5: f e a t h e r b e d d i n g
Congratulations! You guessed the word correctly: featherbedding
```

Here the 3rd change you will make in the `main.c` is uncommenting the Part B, which finds a new random word, and calls the function `playWordGuessingGameAutomatic`.

```c
// ###################################################
// ####################### Part B ###########################
// ###################################################
printf("\n\nAutomatic Version.\n");
printf("\nThe server has generated a random word. You have %d
    attempts to guess the word.\n", MAX_ATTEMPTS);

// Call the automatic version of the word guessing game
int numSuggestion =50;
bool gameResult_auto = playWordGuessingGameAutomatic(
   randomWord, words, numWords, numSuggestion);

if (gameResult_auto) {
    printf("Congratulations! You guessed the word correctly: %s\
       n", randomWord);
  } else {
    printf("Out of attempts. The word was: %s\n", randomWord);
  }
```

Submit On Avenue to Learn following files:

1. `functions.c`
2. `functions.h`
3. `Makefile`
4. `ReadME.pdf` a short detailed description how your Makefile works to connect these files, and mentioning how I can COMPILE and RUN your program (**5 points**).

Objectives: In this assignment, you will learn a perfect structure for code by splitting your code into .h and .c files. Another thing to consider is how much easier it is to crack a password that includes only numbers or alphabets. That's why, sometimes, you may need to pick a password for your account that also includes symbols!