

# Assignment 1: Word Matching Game

SOFT7019

February 2023

## 1 Problem outline

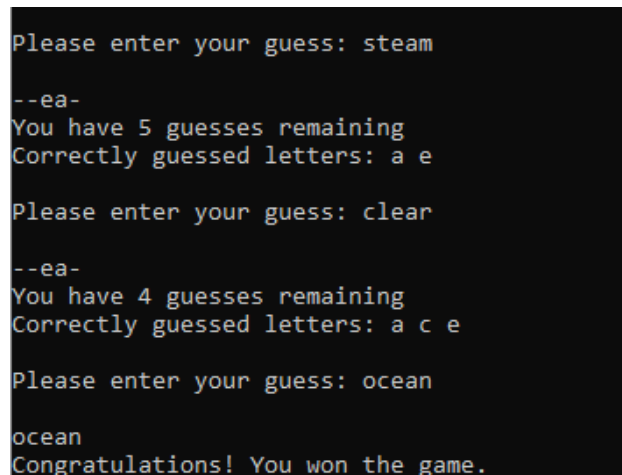
The goal of this project is to implement a C application that can simulate a word matching game. On each iteration of the game a word will be randomly selected from a list of 5 letter words. This word must be guessed by the player, they have 6 guesses to find the correct word.

If the user inputs an incorrect guess they are given information based on the letters of their incorrect word. After the user guesses the program will print a version of the true word with some clues:

- if the player's guess contains a letter from the word in the correct position it will appear in printed word
- incorrectly guessed letters will be marked with a dash

If the player's guess contains letters that are in the word but are not in the correct position they will be added to the correct letter bank.

If the player guesses the correct word they will be rewarded with a congratulatory message. If the player does not guess the correct word in the allotted 6 guesses then they are shown a message that reveals the correct word.

A screenshot of a terminal window showing a word matching game. The background is black, and the text is in a light blue/cyan monospace font. The game starts with a prompt "Please enter your guess: steam". The response is "--ea-", indicating the second and fourth letters are correct. The player is then told "You have 5 guesses remaining" and "Correctly guessed letters: a e". The next guess is "clear", resulting in "--ea-" again, with the letter bank updated to "a c e". The third guess is "ocean", which results in "ocean" being printed, indicating a full win. The final message is "Congratulations! You won the game.".

```
Please enter your guess: steam
--ea-
You have 5 guesses remaining
Correctly guessed letters: a e

Please enter your guess: clear
--ea-
You have 4 guesses remaining
Correctly guessed letters: a c e

Please enter your guess: ocean
ocean
Congratulations! You won the game.
```

Figure 1: Example game play where the user guesses the true word.

You are provided with starter code (`utils.c`) that generates an array of 5 letter words. You have to implement code that

- calls the `load_word_list()` function that is defined in `utils.c` and randomly selects a word from the word list
- asks the user to enter their guess for the word
- keeps track of the correctly guessed letters and the number of guesses
- checks if the user has guessed the word correctly

Note: In the `utils.c` file the function that loads the list of 5 letter words is

```
void load_word_list(char dictionary[DICTIONARY_SIZE][WORD_SIZE+1])
```

`DICTIONARY_SIZE` is the length of the provided list of 5 letter words and `WORD_SIZE` is set to 5. When this function is called it will populate a 2D array of size `DICTIONARY_SIZE` by `WORD_SIZE + 1` with 5 letter words. Consider what role the '+1' plays in the array dimensions.

## 2 Solution requirements

1. Select a random word: each time the game is played a random word should be selected from the word list, this will be the true word that the player is trying to guess
2. Process the player's guess: prompt the user to enter a 5 letter word. Use `getchar()` to read each of the letters in the player's guess. Store the user's current guess.
3. Check the guess:
  - if the user has guessed the correct word let them know with a congratulatory message and end the current game
  - if the user has guessed a letter in the correct position let them know by printing their guess with that letter shown
  - if the user has not guessed a letter in the correct position replace that letter in their guess with a dash (-)
  - if the user has guessed a letter that is in the true word but is not in the correct position then add it to a bank of correctly guessed letters that is shown after each player guess
  - see Figure 1 for an example of this guessing output
4. Repeat the process: the player is given 6 opportunities to guess the true word, if they do not guess the true word in that time the game ends.
5. The player can only guess valid words, each five letter word that the player guesses must appear in the list of valid words (this is the same list that the true word is randomly selected from).

## 3 Additional requirements

In addition to the basic requirements outlined in section 2, these are some additional requirements that will increase your grade.

1. Ensure that your solution works if the player enters both upper case and lower case letters

2. If the player enters something other than a letter the game should prompt the player to reenter their guess using only letters this time. This should not count as one of their 6 guesses.
3. Statistics report: if the player chooses to play multiple games in the one session then a statistics report of their game play should be generated with the following information
  - the percentage of games won
  - the average percentage of characters in the alphabet used in each game
  - a histogram that shows how many guesses it took to win each game
4. The game can be played in "hard mode", in this mode any information learned by the user must be incorporated into their subsequent guesses.
  - if a letter has been guessed in the correct position then it must be used in all further guesses in that position
  - any letters in the bank of correctly guessed letters must be used in all future guesses
  - a letter can no longer be used in a guess if it has been ruled out
  - if the player's guess does not comply with these rules they should be prompted to enter a valid guess and this mistake does not count as one of their 6 guesses

## 4 Compilation

Your application will consist of two source files

- `main.c` this is where you will write your solution code
- `utils.c` this is where the function to load the list of 5 letter words is defined

If you are using an IDE you will have to ensure that both of these files are selected during compilation. If you are using `gcc` you can compile your code with: `gcc main.c utils.c -lm -o main` to generate a `main.exe` executable.

In order for your code to execute correctly, you must be able to load the list from 5 letter words from the provided file called `word_list.txt`. This must be in the same folder as your source code and executable file.

## 5 Grading (100)

- game setup (10)
  - create a variable to store the word list - 2
  - call the `load_word_list()` function correctly - 2
  - randomly select a word from the word list - 2
  - store the true word in an array - 4
- user input processing (10)
  - read 5 letters from the user - 2
  - store the user's guess in an array - 2

- clear the input buffer between guesses - 3
  - if the user enters too few letters ask them to enter a different guess - 3
- game play (35)
  - guess processing 25
    - \* check for correctly placed letters - 3
    - \* check for incorrectly placed letters that are in the true word - 3
    - \* update the bank of correctly guessed letters - 5
    - \* check if the guess exactly matches the true word - 4
    - \* print the word (dashes replace any letters that were not guessed in the correct position, show the true letter otherwise) - 4
    - \* allow the user to guess 6 times - 2
    - \* print an appropriate message if the player fails to guess the word within 6 tries - 2
    - \* print the true word at the end of each game - 2
  - guess validation 10
    - \* check that the player's guess is valid (it is in the list of 5 letter words) - 5
    - \* optimise this checking process - 5
- hard mode (20)
  - a letter can no longer be used in a guess if it has been ruled out - 7
  - if the correct position of a letter has been found that letter must appear in that position in all future guesses - 6
  - any letters in the bank of correctly guessed letters must be used in all future guesses - 7
- additional input checks (10)
  - process both upper case and lower case guesses - 5
  - if the user enters a character other than a letter ask them to enter a different guess - 5
- statistics report (10)
  - loop so that the user can play multiple games and they can request to stop playing - 2
  - calculate the percentage win rate - 2
  - calculate the average percentage of letters from the alphabet used in each game - 2
  - print a histogram that shows the number of guesses required to complete each game - 4
- code quality (5)
  - informative code comments - 3
  - informative variable names - 2

note: do not include string.h in your code, marks will not be given for aspects of the solution that utilise string.h functions instead of basic operations.

## 6 Honour code

- it is ok to exchange ideas with colleagues
- it is not ok to share a solution or parts of a solution with colleagues
- it is not ok to copy solutions from other sources (e.g. Internet)
- we use automated plagiarism detection software to find indications of plagiarism
- plagiarism consequences range from a grade of zero marks given for the assignment and the potential consequences increase in severity if they are escalated to the University Assessment Infringements Board
- when several parties share a solution (or parts thereof), all will be penalised, regardless of who was the source and who copied