HANGMAN DEVELOPER DOCUMENTATION

USED LIBRARIES:

#INCLUDE <STDIO.H>: STANDARD INPUT AND OUTPUT, USED FOR PRINTING TO THE SCREEN & SCANNING FOR USER INPUT.

#INCLUDE <STDLIB.H>: USED FOR MEMORY ALLOCATION FUNCTIONS, LIKE MALLOC.

#INCLUDE<TIME.H>: FOR RANDOM GENERATION PURPOSES.

#INCLUDE <STRING.H>: USED FOR STRINGS FUNCTION, LIKE STRCPY () AND STRCMP ()

# DEFINE DIRECTIVE:

#DEFINE MAX_LINES 3000: THIS VALUE IS PASSED TO THE MULTIDIMENSIONAL ARRAY, IT REFERS TO THE MAXIMUM LINES

#DEFINE MAX_LEN 20: THIS VALUE IS PASSED TO THE ARRAY, IT REFERS TO THE MAXIMUM WORD LENGTH

THESE VALUES USED FOR THE **CHOOSE_DIFFICULTY ()** FUNCTION

#DEFINE HARD 2: IF THE USER INPUT IS 2 -> FUNCTION RETURNS HARD.

#DEFINE EASY 1: IF THE USER INPUT IS 1 -> FUNCTION RETURNS EASY.

#DEFINE INVALID 0: IF THE USER ENTERED AN INVALID INPUT -> FUNCTION RETURNS INVALID.

FUNCTIONS INTERFACE

VOID GET_RANDOM_WORD ()

- Function purpose:
  - As the name suggests, this function will randomly choose a word by a length given by the user.
  - Copy the word to a pointer, then use it.

- Function parameters:
  - This function receives a linked list head (node *head)
  - pointer (word)
  - Integer value -> Word length (word_len)

- Function behavior:
  1- Declare a counter -> this counter counts how many words by the given length exists in the file
  It loops through the words in the file and increment the counter when it finds a word with same length.

  2- Using srand (time (NULL)) & rand () -> the program randomly chooses a number within the range of number of words of the given length, in other words number less than the counter. I named it as (selected).

  3- Declared another counter (cntr)

  4- loop through the words in the List, if a word length equals the length given by the user, increment the counter (cntr). Then if the (cntr) equals the (selected) -> "this is how I managed to get a random number"
  used strcpy () to copy that certain word to the pointer, then use the word.

VOID PLAY_GAME ()

- This function is only for testing purposes.
- It prints the generated word to the screen

VOID PRINT ()

- Function purpose:
  - Prints a string to the screen
- Function parameters:
  - string[ ]
- Function behavior:
  - Assign the length of the passed string to an integer variable
  - For loop to loop through the string and prints the character with spaces (_ _ _ _)

FUNCTIONS INTERFACE

VOID UPDATE_WORD ()

- Function purpose:
  - Update the word with all occurrences of the entered character
- Function parameters:
  - string [ ] -> (the word)
  - character letter -> (the guessed letter)
  - string [ ] -> (current guessed word)
- Function behavior:
  - Assign the length of the passed string (the word) to an integer variable
  - For loop -> this loop is to check if the letter is in the word by checking if each letter in the word equals the passed letter, if yes -> update the letter with the index i.

VOID GENERATE_DASHES ()

- Function purpose:
  - Generate dashes as much as the number of letters in generated word
- Function parameters:
  - Integer value (length)
  - string [ ] (current guessed word) -> in this step the user didn't guess yet.
- Function behavior:
  - 2 for loops:
    - ➢ First for loop to fill the string (current guessed word) with dashes
    - ➢ Second loop is to print the string with spaces -> if the passed length was (4) this will be printed to the screen _ _ _ _

FUNCTIONS INTERFACE

INT CHOOSE_DIFFICULTY()

- Function purpose:
  - ask the user for difficulty "EASY" or "HARD".
  - returns number of misses allowed per one game.

- Function parameters:
  - None

- Function behavior:
  - Declare an integer value (difficulty).
  - Ask the user to choose the difficulty, 1 -> EASY 2-> HARD.
  - Assign the user input to the integer variable .

  - If the difficulty equals 1:
    - The function returns EASY -> EASY is defined as 1.

  - If the difficulty equals 2:
    - The function returns HARD -> HARD is defined as 2.

  - Otherwise (else) :
    - Detect if the user entered an invalid input, then let the user choose again.

INT ALREADY_GUESSED ()

- Function purpose:
  - This function is to check if the letter entered (guessed letter) is in the word. // it is the same of function is_in_word, but I kept both of them for code readability
- Function parameters:
  - Character letter
  - string [ ]
- Function behavior:
  - Assign the length of the string to an integer variable.
  - Declare a counter .
  - Using for loop -> loop through the string .
    - If the string contains the letter .
      - Increment the counter.
  - Check if the counter is more than (1), then the string contains the word, the function returns (1).
  - Otherwise, the function returns (0).

FUNCTIONS INTERFACE

INT GET_WORD_LENGTH ()

- Function purpose:
  - To check if the user entered a valid number
- Function parameters:
  - none
- Function behavior:
  - Ask the user to enter a length
  - If the length is a valid number .i.e., 3 – 16 -> the function returns the length
  - Otherwise, the function asks the user to enter the length again

---

VOID RUN_GAME () // RECEIVES LINKED LIST HEAD

- Declared an integer value (word_length)

- Ask the user for the length of the word they want to use -> the value is assigned with the variable (word_length)
- Declare a string (word[20]) -> stores maximum 20 letters
- Using the function (get_random_word ()) -> the function loops through the list (List_of_words), generates a random word, then copy it to the string (word[20])

- Create a memory allocated array (*current_guessed_letter)
  - It allocates memory for as many characters in the word + the terminating zero ‘ \0’.

- Declare an integer variable (number_of_misses)

- Create a memory allocated array (*missed_letters)
  - It allocates memory to store the missed letters entered by the user
  - when the user misses a guess, the number of misses is incremented and the array (missed_letters) size is also incremented so it can store the required number of letters only

- the program asks the user for difficulty level using the function choose_difficulty():
  - if the user entered an invalid input -> the programs ask them for difficulty again

- using the function (generate_dashes()) -> the function will generate dashes as many as the word length

- the program asks the user to guess a letter
  - assign the entered letter to a string pointer (*guessed letter)
  - then only use the first character of the string guessed_letter[0] to avoid multiple input from the user and ignore the rest of the input

CONTINUE VOID RUN_GAME ()

- WHILE LOOP -> while the guessed letter is not (' ~ ') :
    - using the function (is_in_word())
        - if the letter is in the word -> using the function update() replace that letter in its place, then print the string. The allocated string contains dashes
    - else:
        - If the letter is not in the word:
            - Check if the user already guessed that letter using the function (already_guessed())
                - if yes, the program notifies the user that they already missed that letter. (Does not count as a miss)
            - else -> that count as a miss
                - add the letter to the memory allocated array (missed_letters)
                - increment the number of misses
                - print the number of misses and the missed letters to the screen
                - then check if the user has more chances or not
                    - if the user chose EASY and missed 10 letters the player loses, otherwise the program continues
                    - or if the user chose HARD and missed 5 letters then the player loses, otherwise program continues

        // while loop ends here

    - using strcmp() -> if the word (word[20]) is same as the (current_guessed_letters) // happens after every guessed letter
        - that means the user guessed all the letters
        - player wins -> print winning message to the screen

- after finishing the game, release the string and the missed letters arrays to be able to use them again using free().

INT MAIN ()

FILE HANDLING CREATE_LIST()

- Opening the file
- Check if file == NULL
  - A message appears to the user -> “error occurred with the file“

- Declare word[20] variable to store the words
- Initialized the head with function init_list()
- Scan the file, using push_back function inserts all the words in the list
- Close file
- Return the head

WELCOME MESSAGE:

- Print a welcoming message to the screen (welcome to hangman).

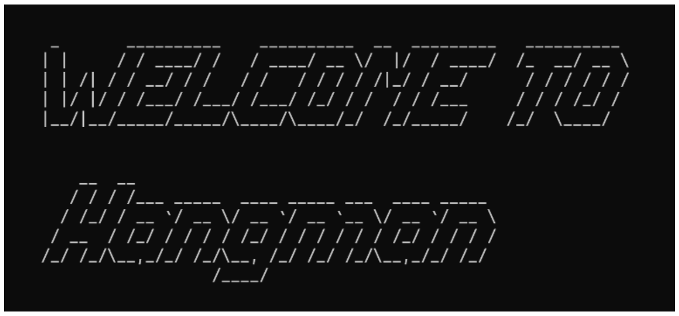
DO{} – WHILE() LOOP

- This loop is to assure the ability of playing again
  - Declared an integer value (resume)
  - Run the game
  - At the end of the game, the program askes the user if they wish to play again , if the player entered (1), the game continues otherwise the program ends.

- Delete the list

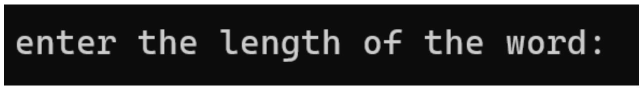
TESTING

WELCOMING MESSAGE ON THE SCREEN :



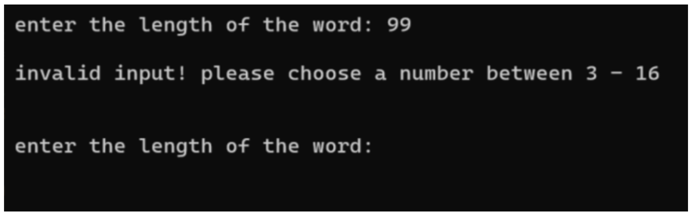
```
 _       ____________    __________  __  _________   __________
| |     / / ____/ /   / ____/ __ \/  |/  / ____/  /_  __/ __ \
| | /| / / __/ / /   / /   / / / / /|_/ / __/      / / / / / /
| |/ |/ / /___/ /___/ /___/ /_/ / /  / / /___     / / / /_/ /
|__/|__/_____/_____/\____/\____/_/  /_/_____/    /_/  \____/


    __  __
   / / / /___ _____  ____ _____ ___  ____ _____
  / /_/ / __ `/ __ \/ __ `/ __ `__ \/ __ `/ __ \
 / __  / /_/ / / / / /_/ / / / / / / /_/ / / / /
/_/ /_/\__,_/_/ /_/\__, /_/ /_/ /_/\__,_/_/ /_/
                  /____/
```

ASKING THE USER FOR THE LENGTH OF THE WORD :



```
enter the length of the word:
```

- if the user entered an invalid input, a notification shows up then let the user choose again:

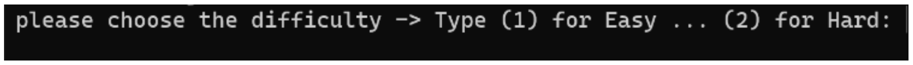


```
enter the length of the word: 99

invalid input! please choose a number between 3 - 16


enter the length of the word:
```

- if the user entered a valid number, then the program continues.

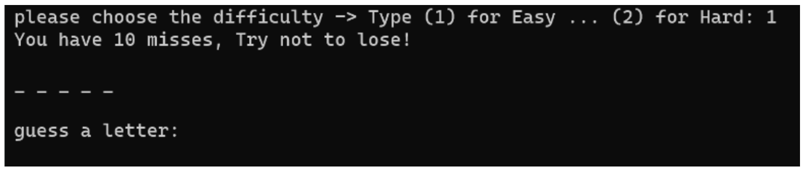
TESTING

ASKING THE USER FOR DIFFICULTY LEVEL:



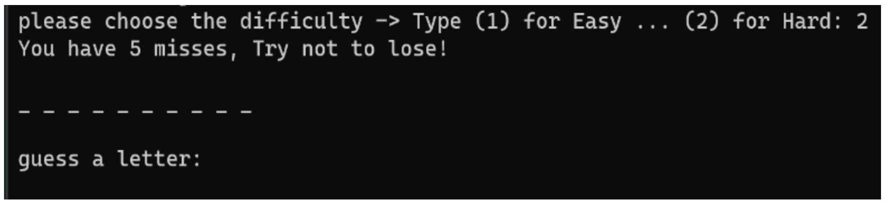
```
please choose the difficulty -> Type (1) for Easy ... (2) for Hard:
```

- ❖ if the user typed 1, the game starts, the user has 10 chances.



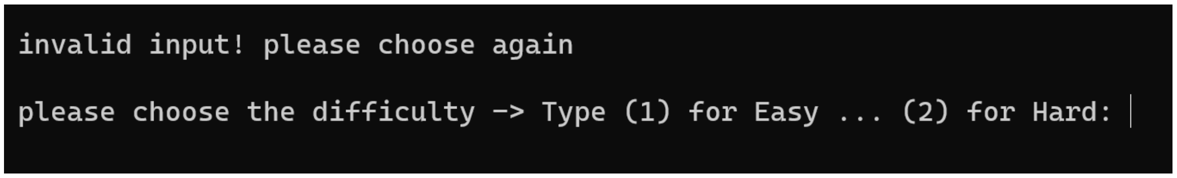
```
please choose the difficulty -> Type (1) for Easy ... (2) for Hard: 1
You have 10 misses, Try not to lose!

- - - - -

guess a letter:
```

- ❖ if the user typed 2, the game starts, the user has 5 chances.



```
please choose the difficulty -> Type (1) for Easy ... (2) for Hard: 2
You have 5 misses, Try not to lose!

- - - - - - - - - - -

guess a letter:
```

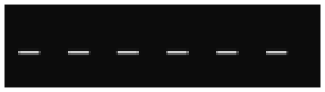
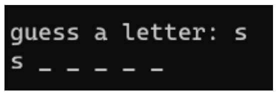
- ❖ Otherwise, it is an invalid input the program will ask the user again

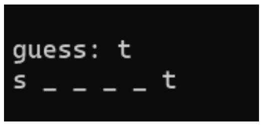


```
invalid input! please choose again

please choose the difficulty -> Type (1) for Easy ... (2) for Hard:
```

TESTING

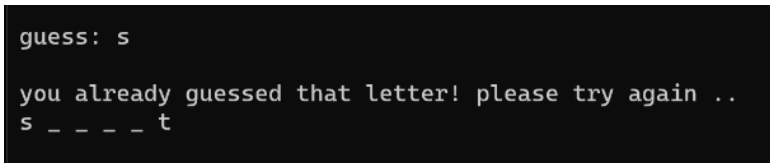
PLAYING THE GAME:

- For example (the word is spirit)
  - Once the game started, dashes will be generated:
    - 
      ```
      _ _ _ _ _ _
      ```
  - If the user guessed letter s, letter s is in the word (spirit), so the first dash will be replaced by letter s.
    - 
      ```
      guess a letter: s
      s _ _ _ _ _
      ```
    - Let's try letter t, letter s will stay at place, then the last dash is updated with letter t.



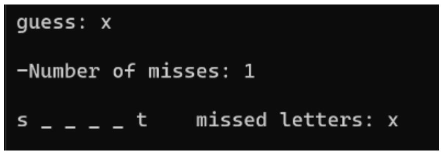
```
guess: t
s _ _ _ _ t
```

  - If the user entered a letter that they already guessed: //that Does not count as a miss

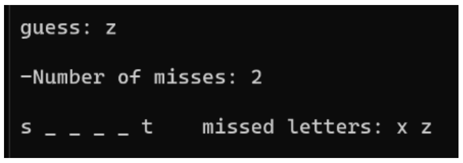


```
guess: s

you already guessed that letter! please try again ..
s _ _ _ _ t
```

  - If the user incorrectly guessed a letter: number of misses will start counting and the missed letters appears

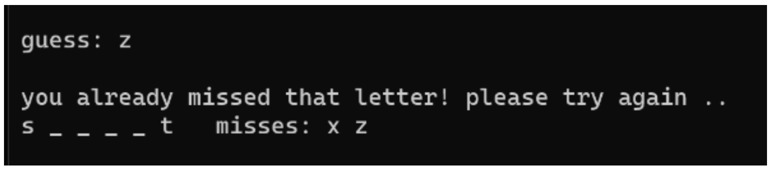


```
guess: x

-Number of misses: 1

s _ _ _ _ t     missed letters: x
```



```
guess: z

-Number of misses: 2

s _ _ _ _ t     missed letters: x z
```

  - If the user incorrectly guessed a letter that they already missed: doesn't count as a miss

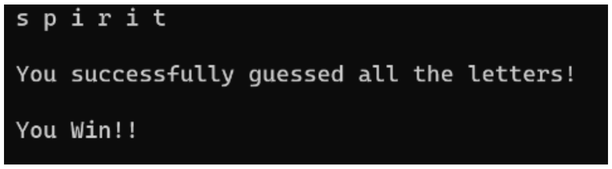


```
guess: z

you already missed that letter! please try again ..
s _ _ _ _ t    misses: x z
```

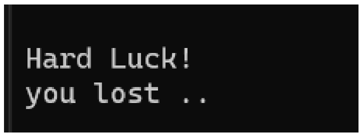
TESTING

CONTINUE PLAYING THE GAME:

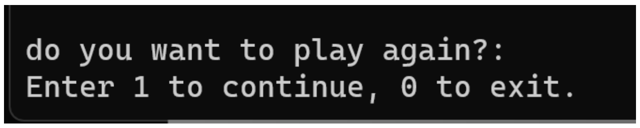
- If the user successfully guessed all the letters:


```
s p i r i t

You successfully guessed all the letters!

You Win!!
```

- Otherwise, if the user ran out of lives, they lose


```
Hard Luck!
you lost ..
```

- Finally, ask the user if they want to play again:


```
do you want to play again?:
Enter 1 to continue, 0 to exit.
```

  - If the user entered 1:
    - The game starts again with a new word .
  - If the user entered 0:
    - The game is over.