# Assignment 2 – Hangman

Sebastian Avila

CSE 13S – Fall 2023

## Purpose

This program implements the hangman game. This game consists of a player guessing letters in order to figure out a secret word. For each wrong guess a body part of a man is drawn hanging from a gallows. The man's body consists of a head, two arms, two legs, and a spine; a total of six incorrect guesses. The program draws ASCII art to represent the man and the gallows.

## How to Use the Program

With the files hangman_helpers.h, hangman_helpers.c, hangman.c, and Makefile in the same directory, use command `make` to compile the program. After doing this, use command `./hangman word` to run the program. This will run the game with the secret word as "word". To change the secret word replace "word" in the command with your desired word. If you wish to have more than one word or wish to include apostrophes, spaces, or hyphens in the secret word, then wrap your secret word with double quotes ("").

## Program Design

This program will first check for the correct number of arguments: 2. It hen passes the second argument to a function which validates the argument as a correct phrase to be used for the game. It then creates an array equal to the length of the string of argv[1] with each values set to 0. A mistakes coounter is initialized and set to 0. An array is created with length of 26 with every value initialized to 0. This array represents whether each letter of the alphabet was guessed. A integer variable is initialized and set to 0 to be used later as an index value for the previous array. A character variable is set to the value ' ' to represent the user's guess and an integer variable is set to 0 to represent if the guess was valid. The program then begins a while loop that continues while the mistakes counter is less than the constant value LOSING_MISTAKE and a function that checks whether all the letters have been guessed returns false. A function is called to print the game to the screen. Then a do while loop begins. Within this loop is another do while loop that prompts, and retrieves a character from the user and continues while the function that checks whether the user's input was a lowercase letter returns false. After exiting that loop, the program checks whether the guess is within the secret phrase. If it is not, the program checks whether it has been guessed already. If it has been guessed already, the program loops again. If it has not been guessed then it is added to the array of eliminated letters and a mistake is added. If the guess is within the secret phrase and has not been guessed already, its guessed value in the array is set to 1. Once the program exits the game loop, it checks whether mistakes is equal to or greater than LOSING_MISTAKES. If it is, it prints the losing prompt, if it is not it prints the winning prompt.

### Data Structures

Describe your data structures here. Data structures include things like arrays, enums, strings, and structs. You should also mention why you chose the data structures that you did.

## Algorithms

For the algorithms section, you want to show psuedocode. Psuedocode should not be able to run on any computer, but should be written in plain english so that someone who knows nothing about computer science can follow along. This section should include sections of code you can give a name to, like ”game loop” or ”check if letter already guessed”.

Want to show some pseudocode? Use the framed verbatim text shown above.

```
bubble sort algorithm
    loop from i = 0 to n - 1
        loop from j = i + 1 to n - 1
            if a[i] > a[j] then
                swap a[i] and a[j]
```

## Function Descriptions

### bool string_contains_character(const char *s, char *c)

This function takes two parameters: a constant character string pointer s and a character c. It returns either a 1 or 0 (true or false). Its purpose is to determine whether the character c is in the character string s.

```
bool string_contains_character(const char *s, char *c)
    while (s does not point to end of string)
        if c equal to s then
            return true
        move s to next character
    return false
```

### bool is_lowercase_letter(char c)

This function takes one parameter: a character c. It returns either true (1) or false (0). Its purpose is to determine whether c is a lowercase letter or not.

```
bool is_lowercase_letter(char c)
    if ASCII value of a <= c or c <= ASCII value of z then
        return true
    return false
```

### bool validate_secret(const char *secret)

This function takes one parameter: a constant charace pointer secret. It returns either true (1) or false (0). Its purpose is to check whether the secret phrase entered by the user meets the required guidelines.

```
bool validate_secret(const char *secret)
    if length of secret > max length of secret phrase then
        print "the secret phrase is over 256 characters"
        return false
    while secret does not point to end of string
       if secret not lowercase letter and secret not a space, hyphen, or apostrophe then
            print "invalid character: {secret}"
            print "he secret phrase must contain only lowercase letters, spaces, hyphens, and
                apostrophes"
            return false
        move secret pointer to next position
    return true
```

**char read_letter(void)**

This function takes no parameters and returns a character. Its purpose is to read a character from standard input and return it. The function checks whether the characters input is the new line character because when a user submits an entry the program receives the users entry followed by a newline character. This means that if the program looks for input again it will receive a newline character rather than waiting for the user's entry.

```
char read_letter(void)
    initialize variable c to space character
    do
        set c to user's input
        if c == new line character then
            loop again
        else
            return c
    while true
```

**bool all_letters_guessed(int guessed[ ], unsigned long len)**

This function takes two parameters: an integer array guessed and an unsigned lon len. Guessed is an array of size equal to the length of the secret phrase. It stores either a 0 or 1 for each index to represent whether that index in the secret phrase has been guessed (1) or not (0). Len is the length of the secret phrase. The purpose of this function is to check whether each value in the array guessed is true.

```
bool all_letters_guessed(int guessed[ ], unsigned long len)
    for i, 0 to < len
        if not guessed then
            return false
    return true
```

**void print_screen(int mistakes, int guessed[ ], unsigned long len, const char *secret, int eliminated[ ])**

This function takes 5 parameters: an integer mistakes, an integer array guessed, an unsigned long len, a constant character pointer secret, and an integer array eliminated. It returns nothing. The integer mistakes represents how many mistakes the player has made. Guessed is an array of size equal to the length of the secret phrase. It stores either a 0 or 1 for each index to represent whether that index in the secret phrase has been guessed (1) or not (0). Len is the length of the secret phrase. Secret is a pointer to the secret phrase. It is constant so it can't be changed accidentally. Elimintated is an array equal to the length of the alphabet with index 0 being a and index 25 being z. It holds either a 0 or 1 at each index to represent whether this letter has been guessed incorrectly.

```
void print_screen(int mistakes, int guessed[ ], unsigned long len, const char *secret, int
     eliminated[ ])
    clear the screen
    print game art according to number of mistakes
    print "Phrase: "
    for i, 0 to < len
        if guessed[i] = true then
            print character secrets pointing at
        else
            print "_"
        move secret pointer to next character
        print new line
        print "Eliminated: "
        for i, 0 to 25
            if eliminted[i] = true
                print character of ascii value i + 97
        print two new line
```

## Results