Assignment 2 – Hangman

Johann Lee

CSE 13S – Fall 2023

Purpose

The purpose of this program is to run the guessing game of Hangman in C code.

How to Use the Program

To use this program it must be first compiled using "make format hangman.c" in inclusion with the make file in order to compile a hangman executable. In order to run this executable use the command "./hangman 'word'". 'word' in this case is the chosen hidden word for the game of Hangman. After which the game will load in. Then the player can type in a letter and press enter to submit their guess. This will either cause a part of the man to appear on the gallows or the letter will appear in the hidden word section if it matches any letters. This continues until either the entire stick figure becomes hung or the player fully guesses the word.

Program Design

Data Structures

Common data structures throughout the main helper functions include if statements and booleans. The reason for this is because of the presence of mainly boolean functions that check for true or false.
Within the main function is a large do-while loop to hold the container of the game while tracking the total mistakes the player has made. Furthermore, are a varied amount of if and for loops which keep track of printing the graphics of the hangman and reading player input.

Algorithms

These are pseudocode for the helper functions:

```
bool string contains a given character (char s, char c)
    if char c not in char s returns not NULL
        return True
    else
        return False
```

```
read input letter
    create char input variable
    prints prompt to input letter
    set input to getchar

    while the input is not newline
        prompt for another input
```

```
    return input
```

```
if char c is lowercase
    if char c is greater than or equal to ASCII 'a' and less then
    or equal to ASCII 'z'
        return True
    else
        return False
```

```
If the secret word is valid
    create size variable length and set it equal to strlen of secret
    if this length is greater than 256
        print character limit error and return false
    for all of secret string
        if its not lowercase or present in punctuation
            print character input error
            return false
    return True
```

The pseudocode here represents the helper functions and how they are set up. The next set of pseudocode shows the basic layout of the main function.

```
main function
    takes in # of words and input string on file run
        if # of inputs is greater than 2
            print quotation error
            return 1
        if validate function is false
            return 1

        create mistake counter
        create int holds character length of secret phrase
        phrase array creation
        eliminated array creation
        copy secret phrase onto phrase
        create char player guess
        create pointer at phrase
        create second eliminated array
        create pointer at second eliminated

        replace every letter in phrase with underscore

        while mistake counter less than MAX MISTAKES
            if phrase matches secret
                    break out loop
            run clear
            print art based on mistake number
            print phrase
            print eliminated letters
            for i =0 to 25
                convert eliminated array into letters and print onto eliminated letters
```

```
            run guess letter and make sure lowercase and not in phrase or eliminated
            if not rerun guess letter

            run string contains character on guess input
            if true
                run through string with pointer and replace
                phrase array '-' if letter matches on string
            if false
                increase mistake counter
                tick eliminated array position of letter from 0 to 1


        run clear
        print art
        if mistake equal to six or greater
            print phrase
            print eliminated letters with same for loop as before
            print lose message
        else
            print phrase
            print eliminated letter with same for loop as before
            print win message
```

Function Descriptions

Here are the function descriptions:
bool string_contains_character

- Target string, character being compared
- returns true if the character is present within the string, returns false if the character is not present within the string
- The purpose of this function is to check if the Target string contains the comparison character.
- Pseudocode is shown above

char read_letter(void)

- No direct inputs, prompts player to input character
- returns the player input character
- The purpose of this function is to give a quick method of reading a one character input from the player
- Pseudocode is shown above

bool is_lowercase_letter(char c)

- target character
- return true if the target character has an ASCII value between lowercase a and lowercase z else it returns false
- The purpose of this function is to check if the target character is a lowercase letter
- Pseudocode is shown above

bool validate_secret(const char* secret)

- target string
- returns true if the string is less than 256 characters in length and it contains only lowercase letters or the permitted punctuation else it returns false
- The purpose of this function is to check if the input string by the player to set up the game of hangman is within the rules of this program
- Pseudocode is shown above

Results

My code compiles as needed according to the autograder. I have attached some pictures down below that show the process I went through for bug fixing. Regarding my lacking points during this project I would say that efficiency of my code was something that could definitely be worked on. There is a plethora of more efficient methods for my code that I am sure I could have improved had I had more time.

```
      [Summary ()]
  5.0/5.0   | Files                  | passed
  3.0/3.0   | clang-format           | passed
  2.0/2.0   | Makefile               | passed
  18.75/25.0 | Functionality: Helpers | errored
  0.0/40.0  | Functionality: Hangman | errored
  Estimated total: ~28.75/75
Cleaning up project directory and file based variables
ERROR: Job failed: exit code 1
```

Figure 1: This is the result of the first iteration of my program.

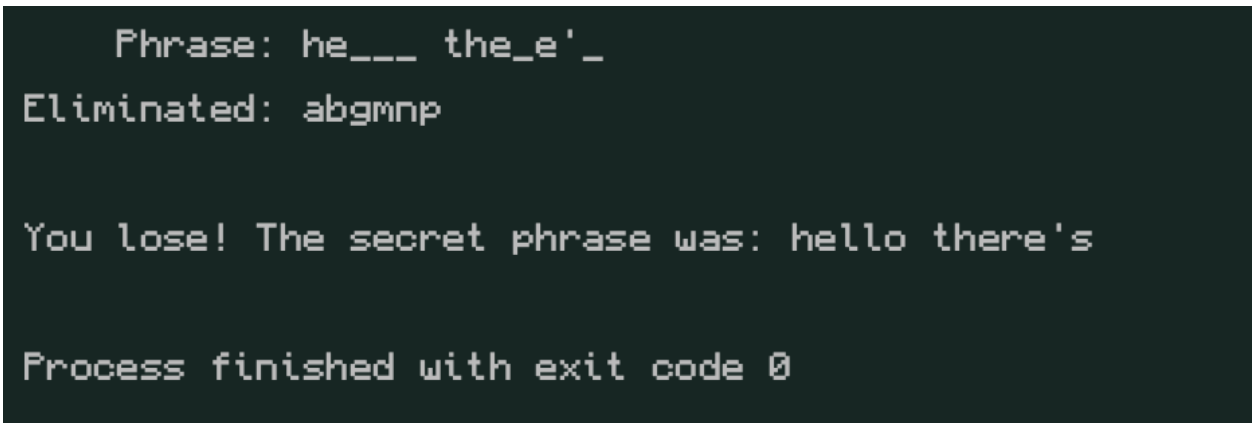

Figure 2: The output was printed through an IDE after concluding the project for simplicity's sake. It can be seen that there are some issues with formatting. There is also no alphabetical sorting of the eliminated letters. The next main issue was during commit 088b9b79 which resulted in uppercase letters crashing the program from an illegal instruction error. This issue was not able to be saved in a picture due to the IDE not replicating the same environment as the VM.

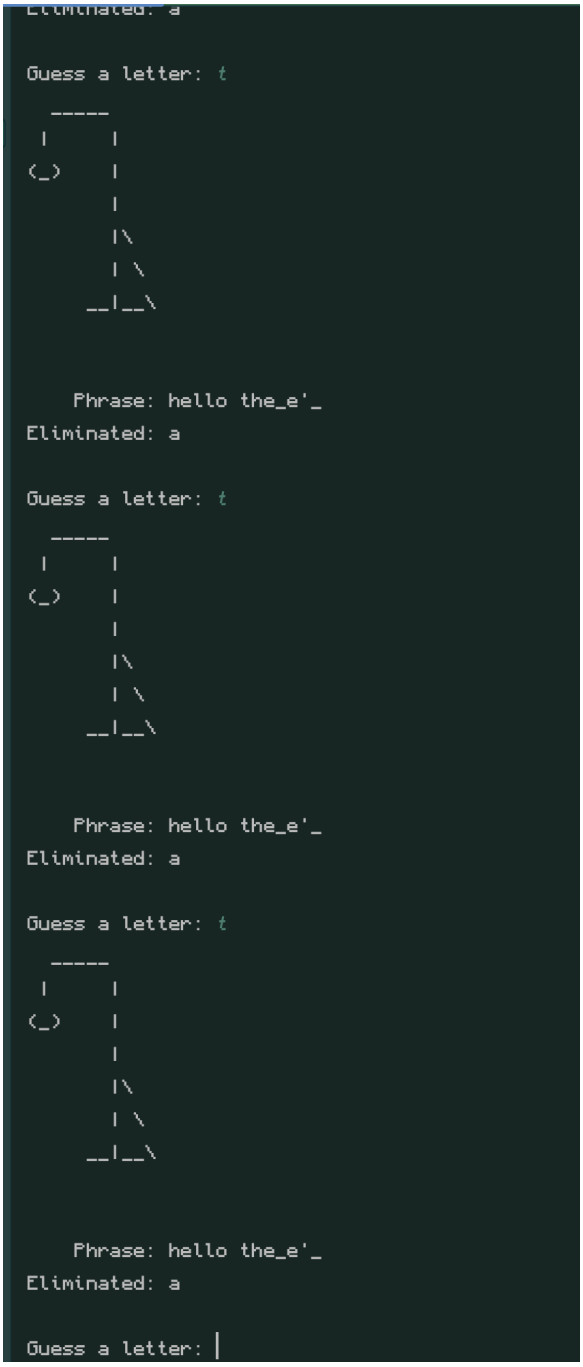

Figure 3: This was after fixing the illegal instruction error. The final issue was to reprint the read_letter prompt even if the character had already been either eliminated or put into the phrase section. This was solved by adding parameters to the while loop that encased the read_letter function forcing it to reprint the prompt even if the letter had already been used. This implemented with the same logic that I had with reprinting prompts if the letter given was an uppercase.

```
do {
    guess = read_letter();
} while (!is_lowercase_letter(guess) || string_contains_character(phrase, guess)
         || string_contains_character(alph, guess));
```

Figure 4: This is the code mentioned in the previous figure regarding parameters. The previous loop had only included the is_lowercase_letter prompt which re-prompted if an uppercase letter was given. Using this same logic I made my code re-prompt even if the letter guess had either already been guessed or if it was already eliminated. I am sure that there is a more logical method to do this, however, this was the only solution I could come up with given that I already had logic implemented that had a similar function.