Assignment 2 – Hangman

Aiden Trager

CSE 13S – Fall 2023

Purpose

This assignment is going to help my skills in C by teaching me the basics of strings and string manipulation. It will also give me a greater understanding of more complex programs in C.

Audience for this section: Pretend that you are working in industry, and write this paragraph for your boss. You are answering the basic question, "What does this thing do?". This section can be short. A single paragraph is okay.

Do not just copy the assignment PDF to complete this section, use your own words.

How to Use the Program

1. In order to run the program first make the program and then run it. These are what the commands should look like in your terminal:

   ```
   make hangman
   ```

2. Run the Program: After making, you can run the program by providing the secret phrase as a command-line argument. If the secret phrase contains spaces or special characters, be sure to enclose it in quotes. Here's an example of how to run the program:

   ```
   ./hangman "secret word or phrase"
   ```

   Replace `"this is the secret phrase"` with your actual secret phrase. The program will start, and you can begin playing.

3. Gameplay: The Hangman game will start. You'll see the initial state of the word to guess represented by underscores and dashes for spaces. You'll also see an empty list of eliminated letters.

4. Make a Guess: The program will prompt you to guess a letter. Enter a lowercase letter from 'a' to 'z', a space, a hyphen, or an apostrophe. Invalid characters and previously guessed letters won't count as guesses.

5. Correct Guesses: If your guess is correct, the program will update the word to guess, revealing the correct letters.

6. Incorrect Guesses: If your guess is incorrect, the program will add the letter to the list of eliminated letters. Incorrect guesses are shown in the eliminated list.

7. Winning: If you correctly guess all the letters in the secret phrase, you win! The program will reveal the entire phrase and tell you that you won. It will then exit the game.

8. Losing: If you reach the maximum number of mistakes (as defined by `LOSING_MISTAKE`), you lose. The program will reveal the secret phrase and inform you that you've lost.

Program Design

The program consists of the following main components:

1. `main` Function: This is the entry point of the program and is responsible for running the Hangman game. It handles user input, game logic, and outcomes.
2. Data Structures: The program uses several data structures to manage information during gameplay. These data structures include arrays and strings to store and manipulate data.
3. Supporting Functions: The program relies on a set of functions defined in `hangman_helpers.h` to perform tasks like validating the secret phrase, printing the game state, and reading user input.

Data Structures

My program employs several data structures to manage and manipulate information during gameplay. These data structures have been carefully chosen to efficiently handle the requirements of the game. Here's a detailed description of the data structures used:

1. `secret` (String): This data structure stores the secret phrase provided by the user. It is of type string (`const char*`) and is chosen for its flexibility in accommodating variable-length phrases. The program ensures that the secret phrase contains only lowercase letters, spaces, hyphens, and apostrophes.
2. `eliminated` (String): The `eliminated` string is used to store letters that the player has guessed, which are not present in the secret phrase. This helps in tracking incorrect guesses. The string is dynamically updated as the player guesses letters.
3. `correct_guesses` (String): This string keeps track of the correct guesses made by the player. It stores letters that are present in the secret phrase. As the player makes correct guesses, the `correct_guesses` string is updated to reflect these letters.
4. `guessed` (String): The `guessed` string stores all the letters that the player has guessed, whether correct or incorrect. It helps prevent the player from guessing the same letter multiple times.
5. `mistakes` (int): An integer variable that counts the number of incorrect guesses. It's crucial for determining the game's outcome.
6. `remaining_letters` (size_t): A variable that tracks the number of letters left to guess in the secret phrase. It helps determine when the player has won.

Algorithms

The Hangman program relies on one key algorithm to facilitate gameplay. Here is pseudocode illustrating the algorithm:

Game Loop:

```
InitializeGame() // Initialize the game state, including the secret phrase, eliminated
letters, correct guesses, mistakes, and other variables.
while (mistakes < LOSING_MISTAKE) {
    PrintGameState() // Print the current game state.
    guess = PromptForGuess() // Prompt the player to guess a letter.
    if (IsValidGuess(guess) && !IsLetterGuessed(guess)) {
        if (IsLetterInSecret(guess)) {
            UpdateCorrectGuesses(guess)
        } else {
            UpdateEliminated(guess)
            IncrementMistakes()
        }
```

```
        if (IsPlayerWin()) {
            PrintWinMessage() // Player wins, end the game.
            return
        }
    }
}
PrintLossMessage() // Player loses, end the game.
```

This algorithm drives the core functionality of the Hangman program and ensures an engaging user experience while adhering to the game's rules and logic.

Function Descriptions

Here's an explanation of each function in the Hangman program, including inputs, outputs, purpose, and decision-making processes:

1. `is_lowercase_letter(char c)`
    - Inputs: `c` (a character)
    - Outputs: `true` if the character is a lowercase letter, `false` otherwise
    - Purpose: This function checks whether a given character is a lowercase letter. It's used to validate characters in the secret phrase.
    - Decision: None. Included in the assignment.
2. `validate_secret(const char *secret)`
    - Inputs: `secret` (a string)
    - Outputs: `true` if the secret is valid, `false` otherwise
    - Purpose: This function validates the secret phrase provided by the user. It checks if the phrase contains only lowercase letters, spaces, hyphens, and apostrophes, and if it's within the character limit.
    - Decision: None. Included in the assignment.
3. `string_contains_character(const char *s, char c)`
    - Inputs: `s` (a string), `c` (a character)
    - Outputs: `true` if the character is found in the string, `false` otherwise
    - Purpose: This function checks whether a given character exists in a given string.
    - Decision: None. Included in the assignment.
4. `read_letter(void)`
    - Inputs: None (reads from standard input)
    - Outputs: The character input by the user
    - Purpose: This function reads a character from the user, ensuring that only the first character entered is considered.
    - Decision: None. Included in the assignment.
5. `print_game_state(const char *phrase, const char *eliminated, int mistakes, const char *correct_gues`
    - Inputs: `phrase` (secret phrase), `eliminated` (letters eliminated), `mistakes` (number of mistakes), `correct_guesses` (letters correctly guessed)

- Outputs: None (prints game state to the console)
- Purpose: This function is responsible for displaying the current game state, including the gallows, the partially revealed secret phrase, eliminated letters, and any correct guesses made.
- Decision: I created this function to provide a representation of the game state while not having to write it all in my main loop.

Results

Currently still working on the program. The results will be here soon.