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

This program contains two integer arrays, an unsigned long value, three integer values, and one character value.

- `guessed`: This array is of size equal to the length of the secret phrase. It hold either a 1 or 0 (true or false) at each index to represent whether the corresponding letter in the phrase at the matching index has been guess. This array is used to check if the user has made a repeat guess and if the game has been won.
- `eliminated`: This array is of size 26. Each index corresponds to a letter in the alphabet with index 0 being 'a' and index 25 being 'z'. The array holds either a 1 or 0 (true or false) at each index to represent whether that letter has been incorrectly guessed. This array is used to print out which letters have been eliminated in alphabetical order. It is also used to check whether a repeat guess has been made. I chose to keep track of the eliminated letters in this way because it allowed me to easily print them out in alphabetical order.
- `len`: This is a variable of type unsigned long. It holds the length of the secret phrase. I chose to use a variable of type unsigned long because the command strlen() returns and unsigned long and changing this to an int value would loose precision. I use this value to iterate over the list of guessed letters and iterate over the phrase itself.
- `mistake`: This integer variable keeps count of the number of mistakes the user has made. It is used to check if the user has lost the game and it is used to determine which game art to print to the screen.
- `elim_index`: This integer variable is used as the index value for the `eliminated` array. When a player makes a guess, this variable recieves the players guess - 97 in order to get the corresponding index value for the letter the player guessed in the `eliminated` array.
- `valid_guess` This integer variable represents whether the user made a valid guess or not. It stores either a 0 or 1 (true or false). It is used to determine whether or not to continue prompting for input.
- `guess` This variable of type character is used to store the user's guess. The user's guess is vital to the program and is used repeatedly.

Algorithms

```
game loop
   loop while mistakes < LOSING_MISTAKE and not all_letters_guessed(guessed, length of phrase)
   print_screen(mistakes, guessed, len, argv[1], eliminated)
       do
           do
               print "Guess a letter: "
               guess = read_letter
           while guess is not a lowercase letter
           if string argv[1] does not contain character user guessed then
               set elim_index to users guess - 97
               if eliminated[elim_index] true then
                   valid_guess = false
                   loop again
               else
                   eliminated[elim_index] = true
                   add a mistake
                   valid_guess = true
           else
               valid_guess = true
               for i, 0 to < len
                   if guess == argv[1][i] then
                       if guessed[i] true
                           vaild_guess = false
                           loop again
                       else
                           guessed[i] = tru
```

```
    while valid guess false
```

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