Hangman Report

Purpose

The purpose of this program is to replicate the classic game of hangman. Using the terminal, one player can choose a word that needs to be guessed, and the other tries to guess the word letter by letter. This can be played for fun or competitively.

Questions

Please answer the following questions before you start coding. They will help guide you through the assignment. To make the grader's life easier, please do not remove the questions, and simply put your answers below the text of each question.
To fill in the answers and edit this file, you can upload the provided zip file to overleaf by pressing [New project] and then [Upload project].

Guesses:
One of the most common questions in the past has been the best way to keep track of which letters have already been guessed. To help you out with this, here are some questions (that you must answer) that may lead you to the best solution.

How many valid single character guesses are there? What are they?

There are only 26 valid single character guesses, which are the 26 letters in lowercase. This is because capital letters aren't counted, and spaces, dashes, and apostrophes are not counted.

Do we need to keep track of how many times something is guessed? Do you need to keep track of the order in which the user makes guesses?

All we have to keep track of is whether a letter is already guessed or not. The order that the user makes guesses does not matter because they will be displayed in alphabetical order anyways.

What data type can we use to keep track of guesses? Keep in mind your answer to the previous questions. Make sure the data type you chose is easily printable in alphabetical order.

To keep track of the guesses, I can make a string of all the lowercase letters, and another list of all the guessed letters. Each time a letter is guessed, I copy it into the guessed letters list. Then when I need to print the guessed letters in alphabetical order, I iterate through the string of all letters then check if they are in the guessed letters string.

Based on your previous response, how can we check if a letter has already been guessed.

We see if it is in the guessed letters string.

Strings and characters:

Python has the functions \texttt{chr()} and \texttt{ord()}. Describe what these functions do. If you are not already familiar with these functions, do some research into them.

These functions convert between character representations and their ascii equivalents. chr() takes an int and returns the ascii char associated with that decimal value. ord() does the opposite, taking in a char and returning the decimal equivalence in the ascii table.

Below is some python code. Finish the C code below so it has the same effect.

```
    x = 'q'
    print(ord(x))
```

C Code:

```
    char x = 'q';
       printf("%d\n", x);
```

Without using \texttt{ctype.h} or \textbf{any} numeric values, write C code for \texttt{is_uppercase_letter()}. It should return false if the parameter is not uppercase, and true if it is.

```
    #include <stdbool.h>
    char is_uppercase_letter(char x){
       if (x < 91 && x > 64){
                return true;
       }
       return false;
    }
```

What is a string in C? Based on that definition, give an example of something that you would assume is a string that C will not allow.

A string is a list of chars that end in \0. I would assume that ‘a’ is a string, or any single character, just like in python, but this is not the case in C because there is no \0.

What does it mean for a string to be null terminated? Are strings null terminated by default?

A null terminated string means that it ends in \0, which is a special character not shown to the user. They are null terminated by default.

What happens when a program that is looking for a null terminator and does not find it.

If other functions or libraries act on that string, it will likely cause errors because they expect to find it.

In this assignment, you are given a macro called \texttt{CLEAR_SCREEN}. What is it's data type? How and when do you use it?

Clearing the screen is a command in c and we use it before each turn by simply printing out the macro.

Testing:

List what you will do to test your code. Make sure this is comprehensive. Remember that you will not be getting a reference binary.

To test my code, I will test both the main hangman function and the individual functions in hangman_helpers. It will be a similar format to the previous two assignments. For the hangman.c file, I will test various inputs, such as those with bad secret words that should give an error message, and also the results of a player winning or losing. For the functions in hangman_helpers, I will check the return values for various inputs where I know the answer, and make sure they are working according to the spec.

Purpose

Using this program is pretty simple. First, make sure you have hangman.c, hangman_helpers.c, hangman_helpers.h, and the Makefile. Compile hangman.c using the make command and the program should compile. Let’s say that there are two people playing the game. The first player will type ./hangman <secret word>, where <secret word> is the word that the other player will try to guess. This runs the program. This secret word must be of lowercase letters, hyphens, or apostrophes. If it involves a space, you must use quotes to A gallow will appear and so will the number of letters in the phrase represented as “_” for each letter. The list of eliminated letters will also be there. The program will prompt the guesser to guess a letter, to which they will type in a lowercase letter. If they type an uppercase letter, a number, or anything else that is not a lowercase letter, it will simply ask them to guess another letter. If they guess an already guessed letter, it will do the same. After each guess, the screen will display the gallow, the correct letters that the user has guessed in their proper position, and the eliminated letters. The guesser is allowed up to 5 mistakes. If they guess a 6th wrong letter, the game ends, and the word is revealed. If they guess all of the correct letters of the secret word, the game ends and the secret word is revealed.

Program Design

This program is divided into four different files. hangman.c is the file that will be compiled and ran. This is where the main function is, and actually goes through the steps of calling different functions to make the program work. These functions, which I will describe later, are held in hangman_helpers.c. In hangman.c, I will include a file called hangman_helpers.h, which basically just contains the function definitions, but not the bodies. hangman_helpers.h also contains some variables and information, such as the gallow drawing, the code to clear the screen, and some include statements for <stdio.h>, <stdbool.h>, <string.h> and <stdlib.h>. This file is connected to hangman_helpers.c via the Makefile, which ties everything together. These four files allow the program to run properly.

I have quite a few variables in my program so here I will list the data structures I will use for those variables. I will copy the user inputted variable, argv[1], into a variable called secret_word, which will be a string. To get the length of secret_word, I will use an int variable secret_word_length. I figured that the easiest way to represent the letters that the user guessed right and wrong were through strings, one of right letters guessed, and one of wrong letters guessed. Then I could check if a user inputted a letter that was in one of the two strings using the string_contains_character function. I also will have integer variables of the number of right and wrong letters guessed. That way, I can check if the number of right letters guessed is equal to secret_word_length, in which case the guesser would win. And I could also check if the number of wrong letters guessed was equal to 6, in which case the guesser would lose.

I will have a fair number of algorithms and functions that help make my program work. Beginning with the string_contains_character function, which takes in two arguments, I will use the strchr function from <string.h> to check if the inputted string contains the inputted character. If it does, all I know is that the return of it would not be NULL. So if strchr(string, character) does not yield NULL, then return true, because it means that the string did indeed contain that character. If not, return false. For the read_letter function, all I have to do is declare a variable I called g. I will prompt the user to guess a letter, and using the scanf function, I will save the inputted letter in char g, and return that value. For validate_secret, I will first check if the length of the secret phrase is greater than the max number of characters, 256, by using the strlen function. If it is, I will tell the user that the length of the secret phrase is over 256 characters, and return false. If it isn't, I will make a string of legitimate characters, which include all lowercase letters of the alphabet, as well as a space, a hyphen, and an apostrophe. I will then iterate over the secret word passed into the function, and make an if statement. This if statement says that if one of the characters was not in the string of legitimate characters, then print the invalid character and return false. If that does not happen, then the function returns true. The last function in hangman_helpers.h is is_lowercase, in which I will make a string of all lowercase letters, and use the strchr function again to see if the inputted character is in the list of lowercase characters. If it does not return NULL, meaning that it is there, return true. Else, return false.

The hangman.c function will begin with some include statements, then goes into the main function. First, I will check if the number of arguments inputted in the command line is1 by checking if argc is 2. If it is, I will tell the user that they must insert only one argument and return one. Then I will move on to defining variables that I will use in my code. I will copy argv[1], the word the user inputted, into a string variable called secret_word, and check if it is a valid secret word using the validate_secret function. If it returns false, the main function returns 1 and ends. If not, I will continue defining variables. Moving on, I will create a string called guessed_wrong_letters, which will be 6. I will also make a string of guessed_right_letters, which will have the length of the secret_word_length. I will then make two integer variables, which are the number of right and wrong letters guessed. Next, I will make a string of the alphabet in all lowercase letters for reference.

The first algorithmic part of this file is putting spaces, hyphens, and apostrophes into the guessed right letters list, because those are already given to the user. By iterating over the secret word, I can check if each character was a space, an apostrophe, or a hyphen, and if it is, I can include that in the guessed right number of letters string, and increase the num_of_guessed_right_letters by one. Next up, I enter the while loop. This keeps going until I return 0 and the program ends. I will first clear the screen, then print the gallow using arts[num_of_guessed_wrong_letters]. Next up, I will print the word “phrase”, and iterate over the secret_word to see if each character in secret_word is in the guessed right letters string. If so, I will print it out, and if not, I will print an underscore to tell the user that they have not guessed that part of the secret_word yet.

Next up comes the eliminated letters. Because this has to be in alphabetical order, I will iterate through my alphabet string and see if each character is in the guessed_wrong_letters string, and if it is, I will print it out. That way, all of the guessed wrong letters will be in alphabetical order. Next comes reading input from the user. I will first make an int variable called repeat and set it to 0. I will then say that while repeat is equal to 0, try to read the letter with the read_letter function, and make sure that the letter is not in the guessed_right_letters or guessed_wrong_letters strings. In addition, make sure that the guessed letter is lowercase. If all of that is true, I will set repeat to 1 which will break out of the loop, and if not, the loop will run again and the program will prompt the user once again. Once the user finally inputs a valid character, I check if it is in the secret_word string. If it is, I will add it to the string f guessed_right_letters, and increase num_of_guessed_right_letters by 1. Then I use the string_contains_character function to check if the guessed letter is in the secret_word, and if it is not, I will add it to the list of wrong letters guessed and increase num_of_guessed_wrong_letters by 1.

To close, I have to check to see if the game is over. If the num_of_guessed_right_letters is the same as the secret_word_length, I will first clear the screen, then print the gallow corresponding to the number of mistakes that the user made. Then I will print the "phrase: " and simply print out the secret_word variable. Then I will print out the eliminated letters in the same way as before: by iterating through the alphabet and seeing if each character is in the guessed_wrong_letters string, and if it is, then printing it out. Then I will print two newlines and announce that the player has won! If the player lost, meaning that the num_of_guessed_wrong_letters is 6, I will do a similar thing. I will clear the screen, print the last gallow showing a fully hanging man, then print out the phrase that the user could get so far. Again, I will do this the same as before: by iterating through the secret_word string and checking if each character is in the guessed_right_letters string, and printing it out if it is. If not, I will print an underscore. To print the eliminated letters, again I will do the same method as listed above to print all of the guessed wrong letters (there should be 6 now) in alphabetical order. I will then print two newlines then give a message that the user lost. Finally, I will return 0 to end the main function and the program.

Pseudocode/Function Descriptions

All of the functions can be found in the hangman_helpers.c file. The string_contains_character function takes in two parameters, a constant string and a char. It should return true if the string contains the char in it and false if it does not. The purpose of this function is to make it easier for the rest of the program to check if a string, whether it be the secret word, or the list of right or wrong guessed letters, contains a specific character in it. This was a very simple and straightforward function.

The read_letter function does not take in any parameters. In the function, it prompts the user to guess a letter, and this guess, which is a character, is the output of the function. The purpose of this function is to make an easy way for the user to guess a letter. Again, it is very simple and straightforward.

The validate_secret function is a tad bit more complicated, but still nothing too crazy. It takes in a string called secret, and returns whether or not it is a valid secret word in the form of a bool. The implementation of this is further explained in the algorithms section. The purpose of this function is to check if the command line argument secret_word is a valid word, meaning it only contains lowercase letters, spaces, hyphens, and apostrophes. Without it, the user could enter numbers or other characters that would make it much harder to guess and certainly unrealistic, given that hangman is a word-guessing game.

The final function is_lowercase_letter takes in a character, and checks if it is lowercase or not. If it is, it returns true. If not, it returns false. I simply make a string of lowercase letters and check if the inputted character is in that string. If it is, return true. If not, return false. The purpose of this function is to make sure that the user inputs a lowercase letter when guessing the secret_word, so that they don’t waste a guess if they guess the right letter but in uppercase. Again, a very easy straightforward function.