Assignment 2 – Hangman Report

Jackson Dawson

CSE 13S – Winter 24

Purpose

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.

This is a playable hangman game! It allows the user to enter a secret word that then must be guessed. The user is given 6 mistakes until they lose, and they win by guessing the entire word. It utilizes strings and arrays in C in order to efficiently organize and execute the game mechanics. It features ASCII visual art, as well as responsive and informative messaging for the user.

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?

  I believe there are 52 valid single character guesses. 26 for each letter of the alphabet in lowercase and 26 for each letter in the alphabet in uppercase.

- 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?

  I don't think so. When something is guessed more than once, all we need is a record that the guess has already been submitted. The order shouldn't be necessary since we will be alphabetizing the guesses anyway.

- 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. [1]

  An array would be a fine choice, but it would require a function to alphabetize the letters for each print. After giving it a bit of thought, this would not be any worse than any other reasonable implementation. I'm thinking an array of size 26.

[1] Your answer should not involve rearranging the old guesses when a new one is made.

- Based on your previous response, how can we check if a letter has already been guessed. [2]

At most the length of our list of previous guesses will be 26, so I would feel fine looking through the entire list each time to verify.

Strings and characters

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

  ”chr” takes an integer argument, and returns the corresponding Unicode character. ”ord” takes a unicode character and returns the corresponding integer representation.

- Below is some python code. Finish the C code below so it has the same effect. [3]

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

  C Code:

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

- Without using `ctype.h` or **any** numeric values, write C code for `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){
    return x >= 'A' && x <= 'Z';
}
```

- 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 an array of ”char” types, with an ending null character. An array defined as ”char string[3] = ’s’,’t’,’r’ might be confused as a string, but is simply an array containing three characters.

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

  The array which contains each character in the string must finish with the character ”(backslash)0”. This signifies that it is the end of the string. Strings are null-terminated by default.

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

  The result of a missing null terminator can be unpredictable. A few common issues include buffer overflow, in which the program reads or writes past the intended end of the array. It can cause infinite loops, or ”segmentation faults” as described in class. This means that the program is trying to access system memory that is not permitted.

- In this assignment, you are given a macro called `CLEAR_SCREEN`. What is it’s data type? How and when do you use it?

  ‘CLEAR_SCREEN‘ is a string literal; that is, it’s essentially ‘const char[]‘. When this macro is printed in the terminal, it sends an ”ANSI Escape Sequence” to the terminal in order to clear the screen, similar to how typing ”clear” into the shell will clear your shell. You can invoke this macro by printing it within your C program. [1]

[2]The answer to this should be 1-2 lines of code. Keep it simple. If you struggle to do this, investigate different solutions to the previous questions that make this one easier.
[3]Do not hard code any numeric values.

Testing

List what you will do to test your code. Make sure this is comprehensive. [4] Remember that you will not be getting a reference binary [5].

I intend to test the output of my binary given the edge cases that I can think of. I will try to input non-valid chars as guesses, such as special characters and numbers. I will make sure that the user does in fact win when they submit a word with only valid special characters. I will test a long-game win condition, as well as various lose-conditions. Additionally, I will test the functionality of my helper functions that are used within the program, such as 'is_lowercase_letter' by testing for invalid inputs as well as a variety of valid letters (upper case and lower case).

How to Use the Program

Audience: Write this section for the user of your program. You are answering the basic question, "How do I use this thing?". Don't copy the assignment exactly; explain this in your own words. This section will be longer for a more complicated program and shorter for a less complicated program. You should show how to compile and run your program. You should also describe any optional flags or inputs that your program uses, and what they do.

To show "code font" text within a paragraph, you can use `\lstinline{}`, which will look like this: `text`.

For a code block, use `\begin{lstlisting}` and `\end{lstlisting}`, which will look like this:

```
Here is some code in lstlisting.
```

And if you want a box around the code text, then use `\begin{lstlisting}[frame=single]` and `\end{ lstlisting}`

which will look like this:

```
Here is some framed code (lstlisting) text.
```

Want to make a footnote? Here's how.[6]

Do you need to cite a reference? You do that by putting the reference in the file `bibtex.bib`, and then you cite your reference like this[2][3][4].

First format the program using:

```
make format
```

Then, build the program using:

```
make
```

You're ready to run the program! Execute it with:[7]

```
./hangman <secret word>
```

Great! You will be shown the level of your progress and eliminated characters at the beginning of each new turn. Each turn, you are permitted to guess one letter, lower or upper case. If you guess something other than a letter, you will be prompted to guess again. If your guess is part of the secret, your word will fill in the appropriate spaces. If your guess is not, your guess will be added to the "eliminated" section, and your man will gain some features! Oh no! If you win, the game will end. If you reach 6 eliminated words before winning, you will lose. Upon either of these conditions, the game will terminate.

When you're done, run the following to clean up:

```
make clean
```

[4]This question is a whole lot more vague than it has been the last few assignments. Continue to answer it with the same level of detail and thought.

[5]The output of your binary is not the only thing you should be testing!

[6]This is my footnote.

[7]Make sure to insert your own secret word instead of "secret word"

You're all done!

Program Design

Audience: Write this section for someone who will maintain your program. In industry you maintain your own programs, and so your audience could be future you! List the main data structures and the main algorithms. You are answering the basic question, "How is this thing organized so that I can have a chance of fixing it?". This section will be longer for a more complicated program and shorter for a less complicated program.

Pseudocode

Give the reader a top down description of your code! How will you break it down? What features will your code have? How will you implement each function.

This first function is my "main" function. It will handle the initialization for the game (setting up required data structures and counters), and then it will jump into a while loop that serves as the game loop. In this current implementation/draft, I have elected to use an array of size 26 to store the eliminated guesses, so that I can ensure easy and efficient insertion in order of each new addition to the list. I have functions planned to handle these needs. Each new turn will clear the screen, print the updated game state using a few functions, and then check whether it should execute a winning, losing, or continuing condition. It is run with an argument in the command line, which is the "secret word" that the user must figure out. Once it has completed its game, it returns 0 to signify a successful run.

```
int main(int argc, char *argv[]) {

    // check for correct usage
    if (argc != 2) {
        printf("Usage: ./hangman <word>\n");
        return 1;
    }

    // Set up game
    char *secret = argv[1];
    char guess[strlen(secret)];

    char eliminated[26];
    initializeEliminated(eliminated);
    int mistakes = 0;

  // game loop
  while (true) {
    // clear the screen
    printf("%s", CLEAR_SCREEN);

    // print the current hangman art
    printf("%s\n", arts[mistakes]);

    // print "Phrase: <phrase with redacted unguessed letters>"
    printSecret(); // TODO

    // print "Eliminated: <list of eliminated letters in alphabetical order>"
    printEliminated(); // TODO

    if (mistakes == LOSING_MISTAKE) {
      printf("%s %s", "You lose! The secret word was:", secret);
      break;
    } else if (isWinner(secret, guess)) {
      printf("%s %s", "You win! The secret word was:", secret);
      break;
    } else {
      // prompt user for next guess
      handleGuess(secret, guess, eliminated); // TODO
    }
  }
  return 0;
}
```

Next is a function that checks whether a function is lowercase or not. It takes one character as an argument, and returns true or false depending on the argument given.

```
bool is_lowercase_letter(char c) {
  return c >= 'a' && c <= 'z';
}
```

Next is a function that checks whether a function is uppercase or not. It takes one character as an argument, and returns true or false depending on the argument given.

```
bool is_uppercase_letter(char c) {
  return c >= 'A' && c <= 'Z';
}
```

This function validates the secret passed in at runtime. It checks to make sure that only valid characters have been used in the secret message. It takes the secret as a parameter, and returns true if the secret is valid, and false if it is not.

```
bool validate_secret(const char *secret) {
  for (int i = 0; i < strlen(secret); i++) {
    if (!is_lowercase_letter(secret[i])
        && !is_uppercase_letter(secret[i])
        && secret[i] != ' '
        && secret[i] != '-'
        && secret[i] != '\'') {
      return false;
    }
  }
  return true;
}
```

This function has two parameters- a string and a character. It checks if the string contains the character, and returns true if so. If not, it returns false.

```
bool string_contains_character(const char *s, char c) {
  for (int i = 0; i < strlen(s); i++) {
    if (s[i] == c) {
      return true;
    }
  }
  return false;
}
```

This function reads an input from the user, and returns it. It is not complete, as I need to make sure that it handles invalid inputs.

```
// TODO: Needs specifications and validations?
char read_letter(void) {
  char c;
  scanf(" %c", &c);
  return c;
}
```

This function takes in the secret string and the updated guess progress that the user has made. It checks if they are the same (i.e. the user has guessed the secret), and returns true if so. Otherwise, it returns false.

```
bool isWinner(const char *secret, const char *guess) {
  if (strcmp(secret, guess) == 0) {
    return true;
  }
  return false;
}
```

This function takes in the secret string and the guess that the user is up to. It checks to see if the user has guessed any of the letters, and prints them if so. Otherwise, it prints an underscore.

```
void printSecret(const char *secret, const char *guess) {
  for (int i = 0; i < strlen(secret); i++) {
    if (string_contains_character(punctuation, secret[i])) {
      printf("%c", secret[i]);
    } else if (string_contains_character(guess, secret[i])) {
      printf("%c", secret[i]);
    } else {
      printf("_");
    }
  }
  printf("\n");
}
```

This function initializes the aforementioned eliminated array with 0's in each place to represent unused slots. it takes an array of size 26 to fill as a parameter.

```
void initializeEliminated(char eliminated[]) {
    for (int i = 0; i < 26; i++) {
        eliminated[i] = 0;
    }
}
```

This function will eventually look through every element in the eliminated array (the one parameter), check if it has a letter in it, and print that letter if so.

```
// TODO: Fit alphabetized requirement
void printEliminated(const char *eliminated) {
  printf("Eliminated: ");
  for (int i = 0; i < strlen(eliminated); i++) {
    printf("%c ", eliminated[i]);
  }
  printf("\n");
}
```

This function will take one character as its parameter, and add it to the eliminated array in the correct spot by evaluating it using ASCII codes.

```
void eliminateGuess() {
  // TODO
  printf("Eliminate\n");
}
```

This function is to be called during the ”continue” condition in the game loop. It will take the secret, the guess progress, and the eliminated array as parameters. It will prompt the user to submit a new letter to guess, and re-prompt if it is invalid. If the guess is in the secret, it will add the letter to the current guess, and continue. If it is not, the guess will be eliminated using eliminateGuess.

```
void handleGuess(const char *secret, char *guess, char *eliminated) {
  printf("Guess a letter: ");
  char c = read_letter();
  if (string_contains_character(eliminated, c) || string_contains_character(guess, c)) {
    handleGuess(secret, guess, eliminated);
  } else if (string_contains_character(secret, c)) {
    for (int i = 0; i < strlen(secret); i++) {
      if (secret[i] == c) {
        guess[i] = c;
      }
    }
  } else {
    eliminateGuess();
  }
}
```

Function Descriptions

For each function in your program, you will need to explain your thought process. This means doing the following

- The inputs of every function (even if it's not a parameter)
- The outputs of every function (even if it's not the return value)
- The purpose of each function, a brief description about a sentence long.
- For more complicated functions, include pseudocode that describes how the function works
- For more complicated functions, also include a description of your decision making process; why you chose to use any data structures or control flows that you did.

Do not simply use your code to describe this. This section should be readable to a person with little to no code knowledge. **DO NOT JUST PUT THE FUNCTION SIGNATURES HERE. MORE EXPLANATION IS REQUIRED.**

References

[1] Wikipedia contributors. Ansi escape code — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/ANSI_escape_code, 2023. [Online; accessed 29-January-2023].

[2] Wikipedia contributors. C (programming language) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/C_(programming_language), 2023. [Online; accessed 20-April-2023].

[3] Robert Mecklenburg. *Managing Projects with GNU Make, 3rd ed.* O'Reilly, Cambridge, Mass., 2005.

[4] Walter R. Tschinkel. Just scoring points. *The Chronicle of Higher Education*, 53(32):B13, 2007.