Assignment 3 – Hangman Report Template

Mira Saini

CSE 13S – Spring 24

1 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.

This program replicates the popular "Hangman" game. When the program is run, the user is prompted for a secret passphrase/the word/phrase they the other user(s) will be trying to guess. The board is built which includes a text art of the gallows, an eliminated section that contains the bad guesses, the phrase hidden by a certain number of underscores, hyphens, apostrophes, and spaces to denote each character. The user is asked to guess a singular, lower-case letter which then if the letter does exist in the phrase, the board replaces the underscore(s) with the guessed letter; if the letter does not exist in the phrase/word, then the board updates the text art of the gallows to begin showing more of the "hanged man" and the letter is added to the eliminated section. If the user guesses incorrectly six times, then the game ends and the user loses. If the user guesses a letter that has already been guessed, nothing happens and the program prompts the user to guess again. If the user guesses all of the letters correctly, then the user wins and a winner message is printed.

2 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].

2.1 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 a total of 26 valid guesses which include all lowercase letters a-z.

- 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?

  We need to make sure that no lowercase letter is guessed more than once, but we don't need to keep track of the specific number. As long as we note down that this character has already been guessed once, we don't need to keep track of the order in which the user makes the guesses. We do need to add incorrect guesses to an eliminated array which then will alphabetize the incorrect guesses. Though, a general array for all guesses does not need to be in alphabetical order.

- 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]

  We can use two arrays to keep track of all of our guesses and for our eliminated guesses. Then we can easily print the eliminated array in alphabetical order and then use the guesses array to make sure that the user hasn't already guessed that specific char.

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

We can check if a letter has already been guessed by using a guesses array that keeps track of all of our guesses. We can then use a loop to loop through the guesses array and return true if the current guess inputted by the user is equivalent to any of the elements in the guesses array.

2.2 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.

  The chr() function in python converts an integer into its ASCII character representation. The ord() function in python converts an ASCII character into its 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", (int)x); //use casting
  ```

- 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){
  if(x >= 'A'&& x <= 'Z'){
      return true;
  }else{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 in C is an array of characters that ends with the terminating character '\0'. If I didn't know this, I would assume that I could change the size of the array after initializing it to a constant length, similar to how you can change string lengths in Python by reassigning the variable with a different string. I would also assume that I would be creating a string in C by setting the length of the array to the number of elements that I plan to have in the array, but this wouldn't make it a string because there wouldn't be any room left for the terminating character. I would have to add an extra space at the end of the array for the terminating character in order for that array to be considered a string.

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

[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.

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

  For a string to be a string and not just an array of chars, it must be null terminated, meaning the array must end with a '\0'. Strings are not null terminated by default in C and therefore you must add the '\0' at the end of the array to indicate that its a string. Initializing an array with an extra element for the '\0' will turn it into a string.

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

  If a program is unable to locate the null terminator, unexpected behavior can occur in the rest of the program. The array is not declared as a string and is just an array of character literals. The program could potentially read past the end of the array and bring random data into the program making the outputs incorrect.

2.3 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, but you do have a file of inputs (`tester.txt`), and two output files (`expected_win.txt` and `expected_lose.txt`). See the Testing section of the assignment PDF for how to use them.

To test my program, I can create test scripts "test_something.sh," make those scripts executable using

```
chmod +x <test\_something.sh>",
```

and then run those tests with

```
./<test_something.sh>.
```

The expected_win.txt and expected_lose.txt files were already provided to me for two different inputs ("zyxwvutsrqponmlkjihgfedcba" and "don't go in empty-handed"). If I wanted to test other inputs, I would have to create additional files similar to these two that print out what the expected output should be for those specific inputs. For example, if I wanted to test the input "abc!", I would have to create a file for what the expected output would look like, which would be: "invalid character: '!' the secret phrase must contain only lowercase letters, spaces, hyphens, and apostrophes"
I would place the expected output in my expected output file and then create a test script to compare the expected output file with a different file that contains the output of my program. To do this, I could write a script that takes an input, runs it through my program and places the output into a file. Then I could use diff to compare that file with the expected output file. If diff returns 0, then no differences were found and the test passed. Else, the program outputs are different and the test failed. If I wanted to see the outputs of the expected output file and the file that contained my program's output, I could do:

```
echo "$(<expected_input_abcexlaimpt.txt>)"
echo "$(<my_program_output.txt>)"
```

which would print both file outputs so I could compare what's happening.

To test the input "abc!," my test script could look something like this:

```
./hangman "abc!" < tester.txt > bad.txt
if diff -w bad.txt expected_input_abcexlaimpt.txt; then
  echo "passed"
else
  echo "failed"
  echo "$(<bad.txt>)"
  echo "$(<expected_input_abcexlaimpt.txt>)"
fi
```

[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.

If I wanted to test other inputs, I could replace the "abc!" with my new input and create a different expected output text file and compare my program output with the expected output text file in the same way.

I could test to make sure that my program handles inputs with over 256 characters correctly by printing an error message and that it proceeds if the input is exactly 256 characters. I could test that my program handles repeated guesses correctly. I could confirm that my will not proceed if my secret phrase has other invalid characters. I could make sure that my program arranges my eliminated array in alphabetical order.

3 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 run this program, simply compile the program with the command "make" and then run the program with the command "./hangman ¡secret phrase¿" which will start the game. The ¡secret phrase¿ should be the string that the user will be trying to guess. Then the user will be prompted to guess single lowercase characters until the user either guesses all of the chars correctly or guesses 6 incorrect chars.

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.[5]

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[1][2][3].

4 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.

This program will be implementing four separate arrays:

1. char phrase[257]: This array will hold the underscore, hyphen, space, and apostrophe characters that will be printed each time after the user guesses a letter.

2. char eliminated[6]: This array will hold all of the eliminated chars that the user has previously guessed. This array is alphabetized after every incorrect guess before being printed.

3. char guesses[27]: This array will hold all of the guesses that the user has previously made (incorrect

[5]This is my footnote.

and correct). We will use this array to ensure that the program doesn't process repeated guesses.

4. char secret[257]: This array holds the user's secret phrase. We will compare the elements of this array to the elements of the phrase array when we want to replace chars in the phrase array with the actual chars in the secret array when the user guesses correctly.

This program will also have three separate int counter variables:

1. int eliminate_counter: The variable will keep count of how many chars are added to the eliminated array. If this variable reaches 6, then the program exits and the player loses.

2. int correct_guessed: This variable keeps track of how many chars have been correctly guessed. We will compare this to our next variable called int length_of.

3. int length_of: This variable denotes how many lowercase chars are in the secret phrase. If this variable equals the int correct_guessed variable, then the program exits and the user wins as they have correctly guessed all of the chars in the secret.

To give a top-down description of my code, I begin by initializing my three counter variables, a variable called breaks which will be used to break out of the program if the user enters an invalid secret, and three of my arrays (not including my secret array), all outside of my main function. Inside main, I declare the secret array and then copy the second command (secret phrase) into the secret, making it a string with a terminating char. Then I use a while loop (while(1)) that checks if the secret is valid using my is_valid_secret function and if so, will set up the board including the gallows, printed phrase and eliminated statements. If the function is invalid, I break from the loop and assign my var breaks to 0. Outside of the while loop, I do two if conditions , where one checks that breaks equals 0 and the length of the secret is less than or equal to 256 chars which means that the phrase is invalid because it contained non lowercase, space, hyphen or apostrophe chars. My second if statement checks if breaks equals 0 and the secret is more than 256 chars, which then just exits the program (I print the error message that its valid due to having more than 256 chars in my is_valid_secret function). If the phrase seems to be valid, I move on to receiving guesses from the user. Inside another while loop that checks if the eliminate_counter is less than 6, I call my read_letter function and then check that my guess is a lowercase letter. If it is, I then check if that guess is already in my guesses array using a loop. If it isn't, I add it to my guesses array and I go on to check if the secret actually contains that character using my string_contains_character function. If the guess has already been guessed, I just use a continue statement to get another guess. Using my string_contains_character function, it it returns true, I use a loop to replace the element(s) in my phrase array with the guessed char and then I print the gallows, the phrase array and the eliminated array. Then I check if the correct_guessed and length_of variables are equal, meaning that the user has guessed everything. If it is, I print the win message otherwise the while loop repeats and asks for another char. If the function returns false, I add the guess to my eliminated_array and increment my eliminated_counter by 1. Then I use my sort function to sort my eliminated array before printing it out. I first print the updated gallows, the phrase array, and then the eliminated array. Then I use a continue statement to get the next guess. Once the while loop exits, meaning six incorrect chars have been guessed, I print the lose message and that's the end of the program!

4.1 Overall 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?

```
//header files up here
//declare my global variables + global arrays
//phrase, eliminated,guesses array
//length_pf, eliminate_counter, correct_guessed, breaks int variables

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

```
    char secret[257]; //initialize array
    strcpy(secret, argv[1])
    while loop{
        is is_valid_secret(secret) == true{
            //print gallows
            //print phrase
            for i in secret{
                if secret[i] == '/''{print "'"}
                else if secret[i] == '-'{print "-"}
                else if secret[i] == ' '{print " "}
                else{print "_"} //means lowercase char
            }

        if is_valid_secret(secret) == false && strlen(secret)<= 256{
            print error indicate that message has invalid chars
        }
        if is_valid_secret(secret) == true && str(secret) > 256{
            //just exit because function prints that the secret is too big
        }
while(eliminate_counter < 6){
    char guess = read_letter()
    //check that the guess is a lowercase char
    guessIndex = -1;
    for num in guesses{
        //if statement to check if num == guesses[num]
        //if so, set guessIndex = num and break
    }
    if (guessIndex != -1){continue}//means that the guess has already been guessed
    //if it does equal -1 just add the guess to the guesses array
    if string_contains_character(secret, guess) == true{
        //print the gallows
        for p in secret{
            if secret[p] == guess{
                phrase[p] == guess
                //increment correct_guessed
            }
        }
        //print update phrase array
        //print eliminated array
    }
    if string_contains_charater(secret, guess) == false{
        //add the guess to the eliminated array
        //update eliminate_counter
        //use a for loop to print out the phrase array
        //use a for loop to print out the eliminated array
    }
//here check if the correct_guessed == length_of, if so then the player wins
}
else{continue} // if guess is not lowercase, just ask for new guess

        }
if eliminate_counter == 6{//print that the player lost and the secret phrase}
    }
}
```

4.2 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.**

1. bool is_lowercase_letter(char c): This function will take in a char and assess if the function is a lowercase letter. The function uses an if statement to test if the char is more than or equal to the ASCII 'a' value which is really just an integer and if the char is less than or equal to the ASCII 'z.' If the char is in between those integer representations of those ASCII chars, then the function returns true. Otherwise, the function returns false.

2. bool is_valid_secret(const char *secret): This function assess if the secret phrase entered by the user in the command line is valid. For the phrase to be valid, it's chars must be either a hyphen, space, apostrophe, or between 'a' and 'z'. The phrase must also be 256 chars or less. The function initializes a counter var to 0 at the beginning. This counter will be used to figure out if the phrase is 256 chars or less. A for loop loops through the chars in the secret phrase. In the loop, the counter is updated and the char is compared to all of the valid chars (hyphen, space, apostrophe, or between 'a' and 'z'). If the char is a hyphen, space, or apostrophe, a continue is used to move on to the next char. If the char is not either of those, it is checked to see if it is a lowercase letter using the is_lowercase_letter function. If true, the loop moves to the next char. If false, then the function prints invalid char and returns false. If all of the chars are valid, then the function tests to see if the phrase is 256 chars or less. If so, it returns true otherwise it prints that the phrase is over 256 chars and returns false.

Here is some pseudo code for this function:

```
bool is_valid_secret(const char *secret){

//valid secret is 256 chars or less and all lowercase

int counter = 0;

for letter in secret{

    counter ++

    //add the hyphen, apostrophe, and space

    if letter ==  -  {continue}

    if letter ==     {continue}

    if letter ==         {continue}

    if bool is_lowercase_letter(letter) == true{
```

```
    continue

        }

    else{

        printf( invalid  character:  \%  s  , letter)

        return false

        }

} //end of for loop

// if statement is not in loop

if counter<=256{

    return True

}

else{

    printf( the  secret phrase is over 256 characters)

    return false

}

} //end of function
```

3. bool string_contains_character(const char *s, char c): This function will return true if the guess (char c) is found in the secret phrase (const char *s). This function uses a for loop to loop through the secret phrase and if it finds that the guess equals an element in the secret, it returns true. Otherwise, if it loops through the entire string and does not find any equivalent chars, it returns false.

4. char read_letter(void): This function retrieves the user's single char lowercase letter guess. It prompts the user for a letter and then uses scanf to assign the input to a variable that is then returned back to main.

5. void sort(char arr[], int n): This function is used to sort my eliminated array each time a new char is added to the array. This function is called anytime the string_contains_character function returns false. This functions uses a bubble sort algorithm that compares two chars at a time and rearranges them as it loops n number of times. n is equaled to the size of the char arr[] and the char arr[] is the array that is being sorted, which in this case is the eliminated array.

References

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

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

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