Assignment 3 – Hangman Report Template

Mandar Patil

CSE 13S – Spring 24

k

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.

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

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 26 valid single characters, ranging from the beginning of the alphabet, 'a' to '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 track whether a letter has been guessed or not. Keeping track of the number of times a letter is guessed or the order of guesses is unnecessary for this game.
- 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 a data structure like an array or a set to keep track of guesses. For ease of printing in alphabetical order, a set would be more suitable.
- Based on your previous response, how can we check if a letter has already been guessed. [2] To check if a letter has already been guessed, we can simply check if it exists in our set of guesses.

[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.

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. chr() and ord() functions in Python are used to convert characters to their ASCII integer representations and vice versa.
- 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';
```

- 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' && return 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.
- What does it mean for a string to be null terminated? Are strings null terminated by default? A string is null terminated when it is ended by a null character (''). Strings are not null terminated by default.
- What happens when a program that is looking for a null terminator and does not find it? If a program looking for a null terminator doesn't find it, it might continue reading memory beyond the intended string, potentially causing undefined behavior or segmentation faults.

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.

Ensure that the program correctly handles valid and invalid inputs from the user. Test the functionality of the game loop, including guessing, tracking correct guesses, and updating the display of the hidden word. Verify that the program correctly ends the game when the word is guessed or the maximum number of incorrect guesses is reached. Utilize the provided input file (tester.txt) and compare the output of the program against the expected output files (expected_win.txt and expected_lose.txt).

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 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:

[3] Do not hard code any numeric values.

[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.

`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].

1. compile the program using a C compiler 2. Run the compiled executable file 3. Follow the prompts to guess letters and uncover the hidden word

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. Data Structures: Word List: Predefined list of words for selection. Hidden Word: String representing the word to guess. Guessed Letters Set: Stores guessed letters to avoid duplicates.

Algorithms: initialization: Selects a random word and initializes the guessed letters set. Display: Renders the hidden word with placeholders for unrevealed letters. Input Validation: Validates single-character input and checks for previous guesses. Update Display: Updates the displayed word based on correct guesses. Win Condition: Checks for a complete word guess. Lose Condition: Determines if maximum incorrect guesses are reached.

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?

1. Initialize the Hangman game: - Get the secret phrase from command-line arguments. - Initialize the hidden phrase with underscores. - Check if the secret phrase is valid.

2. Initialize game variables: - Initialize wrong_attempts to 0. - Initialize arrays to track incorrect and all guesses.

3. Start the main game loop: - While there are underscores in the hidden phrase and wrong attempts are less than the losing mistake threshold: a. Print the current hangman art and game status. b. Print eliminated characters. c. Read a letter from the user. d. Check if the guessed letter is valid. e. If valid: - If the letter is in the secret phrase, reveal it in the hidden phrase. - Otherwise, increment wrong_attempts and mark the guess as incorrect. f. Mark the guess as made.

4. After the loop ends: - Print final hangman art and game status. - Print eliminated characters. - Print the game result based on whether all characters have been guessed correctly.

5. End the game.

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.

[5] This is my footnote.

- 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.**

initialize-game(): Inputs: None. Outputs: Randomly selected word. Purpose: Selects a random word from a predefined list to be guessed by the player. display-hidden-word(): Inputs: Hidden word, guessed letters. Outputs: None. Purpose: Displays the current state of the hidden word, replacing unrevealed letters with placeholders. validate-input(): Inputs: User input. Outputs: Validity of input. Purpose: Checks if the user input is a valid single character. update-display(): Inputs: Hidden word, guessed letters, current guess. Outputs: Updated display. Purpose: Updates the display of the hidden word with correctly guessed letters. check-win(): Inputs: Hidden word, guessed letters. Outputs: Win condition. Purpose: Checks if the player has guessed all the letters in the word, resulting in a win. check-lose(): Inputs: Incorrect guesses, maximum allowed attempts. Outputs: Lose condition. Purpose: Checks if the player has exceeded the maximum number of incorrect guesses, resulting in a loss.

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.