Assignment 2 – Hangman Report Template

Airi Kokuryo

CSE 13S – Winter 24

Purpose

The purpose of this report is to have an overall view of how the hangman game would be coded. It starts with the critical questions towards the game design, and it will also include important codes in C that could be used in the programming of this game. At last, this report will explain the code designs for creating the hangman game in C, including the psudocode and function descriptions.

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 52 valid single character guesses. This is the single character guesses based on alphabets using string.

- 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 keep track of how many times something is guessed, because this will decide when the game will end. For example, if the number of the wrong guesses go over the limit, it should be game over. The order in which the user makes guesses should not matter, because we only need to check if the input is correct or wrong.

- 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] I would keep track of the guesses using arrays. This would keep track of all character input, and could be rearranged in alphabetical order if we use the loop and another array.

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

[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.

new_input is the letter that has been input at this moment, and input_list is the list for all past inputs. "length" here would be the amount of times you can get wrong and the length of the correct word added together, which would be the maximum length of tries.

```
for(int i=0; i<length; i++){
    if(new_input == input_list[i]){
        printf("This word has already been input");
    }
}
```

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.

  In Python, chr() function turns the ASCII code into corresponding character. ord() is the opposite, and takes the alphabet to turn into the integer representing its Unicode code point.

- 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);
  ```

- 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 A = 'A';
  char Z = 'Z';
  char is_uppercase_letter(char x){
      if((int)A<=x && x<=(int)Z){
          //if the input is upper case
          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.

  String in C is an array of characters that is terminated with null. However, while NULL may be considered a string, C does not allow this to be in the array. It must be in the form of "\0".

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

  This means that the string will be considered a string until it hits the null. The strings null are terminated by default, even if you don't input it manually.

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

  If it does not find the null terminator, it may cause buffer overflows, and keep on reading over the memory, which may lead to security issues and unintended behaviors.

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

[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].

* Tests:

1. The input should be an alphabet
2. The input should be only one letter of alphabet
3. The final output should be either you win, or you lose
4. Arrays should not go over the given length

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[1][2][3].

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.

- First will be the user input of the secret
- The alphabets will be extracted into an array from this secret, and if it has no alphabet, the player wins automatically

[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.

- This array will be used to change the secret's alphabet parts into "_" so the player can not see
- The game will start by printing out the explanations and the phrase
- If the user input matches one of the alphabets in the array, this alphabet will be erased from the array
- If the user input does not match the alphabets in the array, it will add 1 to the integer called "mistake", which counts up the times player has mistaken
- If the array becomes all null, the player wins
- If the "mistake" equals the LOSING_MISTAKE, which is the decided number for allowed mistakes, the player loses
- If neither happens, this will loop around until the player wins or loses
- It will printout the result, and the game ends

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 string_contains_character(const char* s, char c)

- The input: char*, char
- The output: bool
- Purpose: This function checks if the string (char*) contains the character (char). If it does, it returns true.

2. char read_letter(void)

- The input: void
- The outputs: char
- Purpose: This function reads the user input and returns it.

3. bool is_lowercase_letter(char c)

- The input: char c
- The output: bool
- Purpose: This function checks if the given character is lowercase. If it is not a lowercase, it returns false.

4. bool validate_secret(const char* secret)

- The input: const char*
- The output: bool
- Purpose: This function checks if the given character is a valid hangman secret.

5. int get_alphabet(const char* secret)

- The input: const char*
- The output: int &array_alphabet
- Purpose: This function checks for alphabets and put it in an array called array_alphabet.

6. int revise_string(const char* secret, char* array_alphabet)

- The input: const char* secret, char* array_alphabet
- The output: int &hidden_string
- Purpose: This function checks for the alphabets in the array_alphabet and changes them to "_", putting it into another array called hidden_string.

7. int delete_alphabet(char* array_alphabet, char c)

- The input: char* array_alphabet, char c
- The output: int &array_alphabet
- Purpose: This function checks for the alphabet given by c and deletes it from array_alphabet if it's found

8. int returned_correct(char* revised_char, char input, char* array)

- The input: char* revised_char, char input, char* array_alphabet
- The output: int &char
- Purpose: Delete the correct alphabet from array_alphabet.

9. bool hangman_game()

- The input: const char*, char, *array, *char
- The output: bool
- Purpose: This function runs the game itself, and decides if the player wins or not.
- pseudocode:

- Print out the statement for starting the game (only the first time)
- run get_alphabet and revise_string
- All of the psudocode under here would be inside a do-while loop
- run revise_string
- print the game statement
- run char read_letter function
- using the given input, run string_contains_char function

- if the function returns false, add the char to the mistake list. Add one to integer called "mistake".
- else if returned true, run returned_correct function and delete_alphabet function.
- at last, check if "mistake" equals LOSING_MISTAKE, and if array_alphabet is all null. If either one is true, get out of the loop by returning true for winning, and false for losing.

• I wanted to make the main game as simple as I could, so I used more outer functions to do the complicated operations.

10. Main function

• Run the user input and the hangman_game function

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.