Assignment 2
Hangman

by Jess Srinivas and Ben Grant
CSE 13S, Fall 2023
Document version 3 (changes in Section 4)

Due Thursday February 1st, 2024
Draft Due Tuesday, January 30th, 2024

1 Introduction

Many of you have played the classic word game, Hangman. Despite its morbid themes, hangman is sometimes played as a tool to teach vocabulary. In this assignment, you will be implementing the game in C.

2 Strings and Arrays

Hangman is a word game, and to create a word game, you must first have a representation for a word. You may recall that a `char` in C is actually just a signed 8 bit integer [1]. To use that integer as a character, we use an ASCII table (which is only defined for positive numbers). A string is just an array of characters. To allow you to complete this assignment, we need to explain all of these in depth, and provide you tools to work with them.

2.1 Arrays

Arrays and memory are a topic that we will cover in a lot greater detail soon in this class, but we will cover the basics. Arrays in C work by storing data in sequential order in memory. When you write a line like, `int arr[3];` the space that the data is stored in is set aside when the program is compiled, meaning that you can't increase the size of an array.

Arrays can also implicitly convert to pointers. In this case, you will end up with a pointer to the first element of the array. However, it's still important to keep in mind that arrays and pointers *are not the same thing*. One thing that demonstrates the difference is the `sizeof` operator, which tells you how much space a variable takes up in bytes. `sizeof` an array is the size of each element multiplied by the number of elements, whereas `sizeof` a pointer is a constant for each CPU architecture (8 on modern computers that use 64-bit processors).

2.2 Strings

Strings in C are just arrays of characters, followed by a null byte (a character whose decimal value is 0). The null byte is widely used by library functions to indicate the length of the string, so omitting a null terminator will likely lead to errors. Fortunately, the compiler automatically inserts null bytes for string literals. You can index into and iterate over a string just like an array. There is also a library for interacting with strings, `<string.h>`. For more information, see the `man` pages on this library.

[1] This is the case for your virtual machine, but may be different on other computers: they are not guaranteed to be signed or 8 bits. The important part is that you can treat them as numbers.

3 Your task

3.1 Workflow

1. Complete and submit your design doc by Tuesday, January 30th.
2. Implement functions in the template file, `hangman_helpers.c`
3. Test those functions. You should do that by creating the file `test_helpers.c`
4. Complete your `main` function in `hangman.c`
5. Test the full functionality of the code with a script you write called `test_functionality.sh`. You can use the text files `win.txt` and `lose.txt` to help you.
6. Fix any issues with your design draft.
7. Submit your final commit ID by Thursday February 1st

3.2 Starter Files

The files we will provide to you to help you with this are as follows:

- `Makefile`: To help you build and run your code
- `Report_template.zip`: For you to upload to `https://overleaf.com` and modify to make your draft and final report.
- `hangman_helpers.h`: This is a header file with a few strings defined for consistency in grading, and a few function definitions that you need to fill out.
- `hangman_helpers.c`: This file is a template for how you should fill out all the rest of your functions.
- `lose.txt` A game in which the player loses. This game was invoked by using `./hangman "don't go in empty-handed"`. You should make sure that when you run your code with the same inputs, it matches this output.
- `win.txt` A game in which the player wins. This game was invoked by using `./hangman zyxwvut-srqponmlkjihgfedcba`. You should make sure that when you run your code with the same inputs, it matches this output.
- `tester.txt` A list of player guesses. The player guessed the same thing for both `win.txt` and `lose.txt`

3.3 Required functions

These functions must appear in `hangman_helpers.c`

`bool string_contains_character(const char* s, char c)`

Checks if the string `s` contains the character `c`. If it does, it returns `true`, otherwise, it returns `false`. You may assume that `s` is a properly null terminated string, less or equal to than 256 characters.

`char read_letter(void)`

Prompts the user for a guess and reads in one letter (or any other character) from `stdin`. It then returns it. You may assume that the user only inputs one character followed by a newline.

`bool is_lowercase_letter(char c)`

Checks if the character `c` is a lowercase letter. If it is not, it returns false. You may not use any functions from `ctype.h` in this function, or anywhere else in your code.

`bool validate_secret(const char* secret)`

Takes in a string called `secret` and checks if it is a valid hangman secret. If it is not, it will print the first invalid character in the form `invalid character: 'X'`, where `X` is the character that is invalid and return false. It must match the format below (including the new line). In this case, the offending character was a zero. The error message must print to `stdout`. The secret can be up to 256 characters. If the secret is too long, it will print `the secret phrase is over 256 characters`. If the secret is valid, it prints nothing and returns true.

```
invalid character: '0'
the secret phrase must contain only lowercase letters, spaces, hyphens, and apostrophes
```

Recall that a valid secret word is all lowercase, but may include spaces, apostrophes, and hyphens. The argument here is `const` so we can't modify the secret on accident when we check if its valid.

3.4 Assignment specifics

For this assignment, the hangman game that you create will have a few modifications. For one, the secret can be multiple words, and can include apostrophes, spaces, and hyphens. A secret must be provided, and at most 256. The secret will be inputted by the user before the game starts as a command line argument. If these special characters are present, they will be shown at the start of the game, and every time the guesses are updated. If only special characters are present, the player has already won. The player will guess only one letter at a time. You do not need to handle errors if they chose to show more than that. If the player does not enter a secret (which is different from giving an empty string), or if the user enters the wrong number of arguments in any other way, print the following error.

```
wrong number of arguments
usage: ./hangman <secret word or phrase>
if the secret is multiple words, you must quote it
```

We have also provided you with a few strings in `hangman_helpers.h`. One of these strings clears the screen, and it must be what you use to clear the screen as well. There is also an array of ascii art for hangman. You should print one of these strings every time a player makes a guess. Use the number of mistakes the player has made as an index for this array. You must also print these exactly. Along with that, there is a definition of `LOSING MISTAKE`. You should use that to determine when the game is over. For example, if `LOSING_MISTAKE` is 3, the player should lose immediately after making their third incorrect guess.

In the beginning of the game, you will print an empty gallows, a line for the phrase, and a list of eliminated letters. You will also print a new line and a prompt to have the player guess a letter.

```
    Phrase: ___'_ __ __ _____-______
Eliminated:


    Guess a letter:
```

In this example, the phrase is `don't go in empty-handed`, and the gallows is not shown. Note how the colons after "Phrase" and "Eliminated" align. Your output must match ours, and this is a good start to doing so.

If the player loses, you will print the following

```
    Phrase: d__'_ g_ i_ e____-ha_ded
Eliminated: bcfjkl

You lose! The secret phrase was: don't go in empty-handed
```

In this example, the player guessed letters in reverse alphabetical order, starting at `l`. The gallows is not shown in the PDF for clarity. Note that the eliminated letters print in alphabetical order. This is how the eliminated letters should always print. If a player guesses any letter that they have already guessed, we should prompt the player for another letter without reprinting the gallows and phrase.

If the player wins, you will reprint the gallows and phrase, along with a phrase telling the player they won.

```
    Phrase: don't go in empty-handed
Eliminated:

You win! The secret phrase was: don't go in empty-handed
```

Again, the gallows is not shown in this PDF. In this case, the player managed to guess the entire phrase without making any mistakes.

3.5 Makefile

In this assignment, you will be modifying the given Makefile slightly. You must make the following changes:

- Create a target called `tests`. This must run the targets `test_functionality` and `test_helpers`
- Create a target called `test_functionality`. This must run the shell script you create, `test_functionality.sh`
- Create a target called `test_helpers`. This must compile **and run** the file `test_helpers.c`

3.6 Submission

These are the files we require from you:

- `Makefile`: To help you build and run your code. You may only add to this file. When you submit this file, it must include targets for the tester files you write.
- `design.pdf`: Instructions can be found in the template
- `hangman_helpers.h`: This is a header file with a few strings defined for consistency in grading, and a few function definitions that you need to fill out. You may add functions to this file, but may not modify anything that is currently there.
- `hangman_helpers.c`: This file is a template for how you should fill out all the rest of your functions. You may NOT use any functions from `ctype.h` in this file.
- `hangman.c`: This should contain anything a main function that you use to play hangman. You must use all four of the functions provided in `hangman_helpers.h`
- `test_helpers.c`: This must thoroughly test all the functions in `hangman_helpers.c`
- `test_functionality.sh`: This file must ensure basic correct functionality of your code by matching your binary's functionality to the given test cases.
- `win.txt`, `lose.txt`, and `tester.txt`: These should be submitted unmodified.

4 Revisions

Version 1 Original.

Version 2 Slight clarifications on losing and testing code.

Version 3 Clarification on running the code and errors.