COSC 301: Operating Systems
Fall 2019
H1: Hangperson in C
Due: 17 September 2019, 11pm

Overview

There's just one problem for this homework: implement a program to play Hangperson in C. Yes, the game you know and love (and perhaps implemented in Python in COSC 101?) will be brought to life in the C programming language.

To get started, click on the Github classroom link on Moodle to create a repo in your own account, then clone the repo on falcon.cs.colgate.edu. (You will, of course, need to ssh onto birds.cs.colgate.edu to do the cloning step).

If you're unsure how to play Hangperson, please talk to me or a fellow student and play a practice game or two. It's pretty important that you have a clear image in mind for how the game works before moving on to writing the code.

Please work in groups of 2 for this homework (and all subsequent homeworks). For working together, I'd recommend using the "Live Share" feature of Visual Studio Code, or "Teletype" in Atom. Both of these tools enable two users to work on code simultaneously (like a Google Doc). Please ask if you want help getting set up with either of these tools.

Learning Goals

By the end of this homework you should be able to:

1. Work effectively with C strings and arrays, including mutating arrays that are passed into functions.
2. Using built-in C functions for getting input (fgets), printing to the console (printf), working with strings and characters (strlen, isalpha, toupper).
3. Work effectively with C control structures including for loops and if statements.

Details

This homework has been structured in a way so that you can focus on writing 7 specific functions to handle certain aspects of game play and get practice with C strings, arrays, control structures, and C library functions.

The 7 functions to write are described in some detail below. Skeletons for each function exist in the file `hangperson.c`; all the code you write should go in that file. To get a sense for the overall structure of the game, it is *strongly* recommended that you first read the function `one_game` (in hangperson.c), which is the function called from `main` (in main.c) to get one game

started. The basic flow of `one_game` is as follows:

Create and initialize arrays to store game state.
While the player has not won and has not lost:
    Print the gallows and the current game state
    Ask for the next guess
    If the guess has already been made, say so
    Otherwise, update the game state with the new guess

Some details on how you should handle the game state are given below, as well as some other details for the game. Note that the secret word is printed (from within main.c) before a new game is started; this is to help with debugging your code. Once you've gotten everything working, you could comment this line out. Note also that if you'd like to change the set of words used for the game, you're welcome to do so (just modify the appropriate array in main).

Here are details on the 7 functions you'll need to write:

Function 1: `initialize_game_state`

This function should, not surprisingly, initialize the game state at the beginning of a new game. The function signature is:

```
void initialize_game_state(const char word[], char game_state[], bool
already_guessed[26])
```

The function should accept the secret word (a string), the game state (an array of char, which is the same length as the secret word), and an array indicating which letters have already been guessed as parameters. It should initialize both the `game_state` and `already_guessed` arrays and return nothing.

The `game_state` array should be initially set to the character '_' (underscore) to indicate that the letter at a given position in the word has not yet been correctly guessed. As correct guesses are made for letters in the word (later in the game), these underscores will be changed to the correct letter at the given position. You can either use the built-in `memset` function to initialize the array, or use a for loop. Beware that the `game_state` array is a "plain" array of char; it is *not* intended to be a regular C string.

The `already_guessed` array is an array of bool indicating whether a letter has been previously guessed or not (notice that it is of length 26, so that we can keep track of whether any letter in the English alphabet has been previously guessed). You should initialize the array to all false, again using either the built-in `memset` function or a for loop.

Note that for the `already_guessed` array, the true/false value at index 0 indicates whether 'A'

has been previously guessed, index 1 indicates whether 'B' has been previously guessed, and so forth.

Function 2: `update_game_state`

This function should accept the current guess (which can be assumed to be an upper-case letter), the secret word (a C string), and the game state, and modify the game state to account for the current guess. The function should return true if the guess was correct (i.e., it is a letter in the secret word), and false otherwise.

The game state, again, should hold either an underscore ('_') or a letter in each position of the array, depending whether the player has correctly guessed a letter or not. For example, if the secret word is "SNOW", the player has just guessed 'S', and the current game state is ['_', '_', '_', '_'], this function should update the game state to be ['S', '_', '_', '_']. Note that if a letter appears more than once in the secret word, all relevant positions in the game state must be updated.

Here is the function signature for `update_game_state`:

```
bool update_game_state(char guess, const char word[], char game_state[])
```

Function 3: `print_game_state`

This function should print out two things: the current game state, and any letters that have been previously guessed. The function signature is:

```
void print_game_state(const char word[], char game_state[], bool already_guessed[26])
```

You can see the game trace at the end of this description for examples for how your output might look. The details are up to you, but they should at least vaguely resemble the output shown below.

Note that for printing the letters that have been previously guessed, index 0 in the array represents whether 'A' has been previously guessed, index 1 represents whether 'B' has been previously guessed, and so on. You have the index (and the true/false value at the index) but you will need to print the corresponding letter (if it has been previously guessed). Remember that the char type in C is basically just an integer, so 'A' actually is treated as the integer value 65 by C. (The specific value doesn't really matter here, but 'A' is indeed 65.) Use that fact to be able to print out the right letters. Perhaps obviously, you should not print any letters that have *not* been previously guessed.

Function 4: `get_guess`

This function should repeatedly ask for a new letter to guess from the player until the player enters a single letter. The function should return the letter *in upper case*. The function signature is as follows:

```
char get_guess(void)
```

A few notes on this function:

- I'd highly recommend that you use the C standard library `fgets` function for getting keyboard input as well as input from the dictionary file. There are examples for how to use `fgets` in the tutorial chapter of the 301 book of C (https://jsommers.github.io/cbook/tutorial.html).
- Your function should be able to handle the case when a player enters a non-letter or some other bogus guess. The program should not crash, an appropriate error message should be shown, and the game should continue. The function should only return when a single alphabetic character has been entered. Don't write this function recursively, or you will seriously hurt my feelings.
- Note that the `isalpha` built-in function can be used to test whether a single character is alphabetic.
- You should handle the case in your program if a user explicitly types a magic key-sequence to indicate "end of file" (EOF). `fgets` will return NULL on eof (or you can use the C standard library feof function to test whether stdin has reached end of file). You can type EOF on the keyboard by using ctrl+d (control key plus little d). If your program receives EOF, you should just exit the game. There is a built-in C function `exit(int)` that you can use to exit the program. It takes an integer "status" as a parameter, which can just be 0.
- Beware that when you read a line from a file using `fgets`, the last character in the line is pretty much always a newline character (\n). You'll need to be careful to strip this character out *before* testing whether there is one character and that it is alphabetic.
- Finally, note that the `toupper` function can be used to convert a single letter to uppercase.

Function 5: `won`

This function should accept the secret word and the game state as parameters, and return true if the player has won the game, and false otherwise. For this function, you should simply go through each letter in the secret word and check that that letter is also in the same position of the game state. If so, the player has guessed all the letters. (Alternatively, you can check that there are no remaining '_' (underscore characters) in the game state.)

```
bool won(const char word[], char game_state[])
```

Function 6: `lost`

This function should accept the current number of incorrect guesses and return true if the player has lost and false otherwise. A player loses when they make 7 missed guesses (i.e., 6 misses is ok, but on the 7th miss the player loses). Note that there is a constant integer declared `MAX_MISSES` which is equivalent to 7.

```
bool lost(int incorrect_guesses)
```

Function 7: `previous_guess`

Lastly, the `previous_guess` function should accept the current guess (a single character, in uppercase) and check whether the player has already guessed that letter. If so, the function should return true. If the player has not previously guessed the letter, the function should (1) modify the `already_guessed` array to indicate that the current letter has now been guessed, and (2) return false. The function signature is:

```
bool previous_guess(char guess, bool already_guessed[26])
```

Note that to check and/or modify the already_guessed array, you'll need to compute the index corresponding to a given letter (i.e., 'A' is index 0, 'B' is index 1, etc.). Again, remember that 'A' is treated as an integer by C. Use that fact (and a very simple arithmetic expression) to obtain the correct index to work with.

Testing

To compile your code, you can just type "make" at the shell. Two programs will be produced, if compilation is successful: `hangperson` and `hangtests`. The hangperson program can be invoked to play a game of `hangperson` (see example game play, below). The `hangtests` program can be invoked to run a series of tests on the functions you've written. Please refer to the code in `hangtests.c` for details on what the tests actually do. When you first get started, all the tests should fail, as shown below:

```
$ ./hangtests
test_initialize_game_state()
        Game state: (should be initialized to _ for every letter to guess)
                0 & ? ? ?       ... not ok
        Already guessed (everything should be initialized to false)
                11/1 13/1 19/1 21/1 ... not ok
test_update_game_state()
        Updating game state, guessing 'T' for word 'TRABANT';
        expecting T to be at indexes 0 and 6 in updated state:  return val should be
```

```
true but was not; T should be at index 0 but was not; T should be at index 6 but was
not;
        Updating game state, guessing 'A' for word 'TRABANT';
        expecting T to be at indexes 2 and 4 in updated state:  return val should be
true but was not; A should be at index 2 but was not; A should be at index 4 but was
not;
        Updating game state, guessing 'Z' for word 'TRABANT';
        expecting game state to be unchanged: not ok
test_won()
        won("TEST", "TEST") != true;  ... not ok
test_lost()
        lost(7) != true;  ... not ok
test_previous_guess()
        A was set in previous_guesses array but previous_guess returned false
        B was set in previous_guesses array but previous_guess returned false
        C was set in previous_guesses array but previous_guess returned false
        D was set in previous_guesses array but previous_guess returned false
        E was set in previous_guesses array but previous_guess returned false
        F was set in previous_guesses array but previous_guess returned false
        G was set in previous_guesses array but previous_guess returned false
        H was set in previous_guesses array but previous_guess returned false
        I was set in previous_guesses array but previous_guess returned false
        J was set in previous_guesses array but previous_guess returned false
        K was set in previous_guesses array but previous_guess returned false
        L was set in previous_guesses array but previous_guess returned false
        M was set in previous_guesses array but previous_guess returned false
        N was set in previous_guesses array but previous_guess returned false
        O was set in previous_guesses array but previous_guess returned false
        P was set in previous_guesses array but previous_guess returned false
        Q was set in previous_guesses array but previous_guess returned false
        R was set in previous_guesses array but previous_guess returned false
        S was set in previous_guesses array but previous_guess returned false
        T was set in previous_guesses array but previous_guess returned false
        U was set in previous_guesses array but previous_guess returned false
        V was set in previous_guesses array but previous_guess returned false
        W was set in previous_guesses array but previous_guess returned false
        X was set in previous_guesses array but previous_guess returned false
        Y was set in previous_guesses array but previous_guess returned false
        Z was set in previous_guesses array but previous_guess returned false
         ... not ok
```

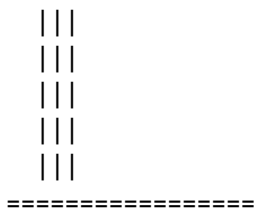
Submission

When you're done you should (1) answer questions in the README.md, (2) document in your README.md who worked together on this homework, and (3) commit your work to git and push to Github.

Example game play

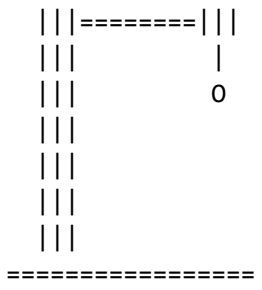
Below is an example transcript of game play (specifically, of a losing game). Your messages to the user can be anything you like; there are no specific requirements on how you print information.

```
$ ./hangperson
Number of words 4
Secret word is ICE


       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
     ==================



_ _ _
Already guessed:

What is your guess? e
Good guess.
Missed: 0


       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
     ==================



_ _ E
Already guessed: E

What is your guess? o
Bad guess, fool.
Missed: 1


       |||========|||
       |||         |
```



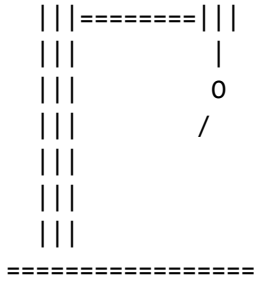
_ _ E
Already guessed: EO

What is your guess? m
Bad guess, fool.
Missed: 2



_ _ E
Already guessed: EMO

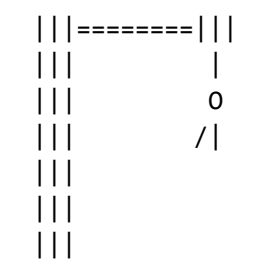
What is your guess? p
Bad guess, fool.
Missed: 3



_ _ E
Already guessed: EMOP

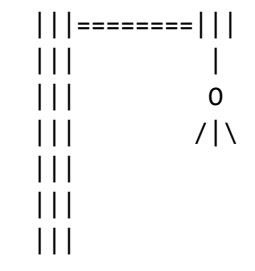
What is your guess? z

Bad guess, fool.
Missed: 4



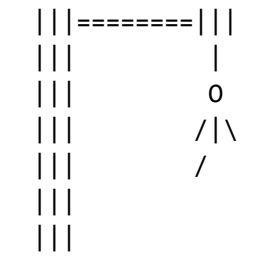
_ _ E
Already guessed: EMOPZ

What is your guess? a
Bad guess, fool.
Missed: 5



_ _ E
Already guessed: AEMOPZ

What is your guess? v
Bad guess, fool.
Missed: 6



```
_ _ E
Already guessed: AEMOPVZ

What is your guess? x
Bad guess, fool.
Missed: 7
You lost and made stick-person sad...

$ ./hangperson
Number of words 4
Secret word is SNOW


       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
     ==================



_ _ _ _
Already guessed:

What is your guess? s
Good guess.
Missed: 0


       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
     ==================


S _ _ _
Already guessed: S

What is your guess? n
Good guess.
```

```
Missed: 0


       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
     =================


S N _ _
Already guessed: NS

What is your guess? o
Good guess.
Missed: 0


       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
     =================


S N O _
Already guessed: NOS

What is your guess? w
Good guess.
Missed: 0
Congratulations!  You saved stick-person!

$ ./hangperson
Number of words 4
Secret word is FROZEN


       |||========|||
       |||
       |||
       |||
```

```
       |||
       |||
       |||
     =================
```

```
_ _ _ _ _ _
Already guessed:

What is your guess? abc
Seriously.  Just one letter, please
What is your guess? 3
C'mon, fool, letters only.
What is your guess? f
Good guess.
Missed: 0
```

```
       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
     =================
```

```
F _ _ _ _ _
Already guessed: F

What is your guess? f
You already guessed that.
```

```
       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
     =================
```

```
F _ _ _ _ _
Already guessed: F
```

```
What is your guess? r
Good guess.
Missed: 0


       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
     ==================


F R _ _ _ _
Already guessed: FR

What is your guess? o
Good guess.
Missed: 0


       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
     ==================


F R O _ _ _
Already guessed: FOR

What is your guess? z
Good guess.
Missed: 0


       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
```

```
     =================


F R O Z _ _
Already guessed: FORZ

What is your guess? e
Good guess.
Missed: 0


       |||========|||
       |||
       |||
       |||
       |||
       |||
       |||
     =================


F R O Z E _
Already guessed: EFORZ

What is your guess? n
Good guess.
Missed: 0
Congratulations!  You saved stick-person!
```