Homework 3 - MPCS 51040

(last modified: April 11, 2017)

Issued: MondayApril 10, 2017

1 General Instructions

1.1 Compiling

Your code must compile with `gcc -std=c11 -Wall -pedantic`. There should be no warnings or errors when compiling with gcc (as installed on `linux.cs.uchicago.edu`).

1.2 Handing in

To hand in your homework, you need to commit all requested files (with correct filenames!) to your personal git repository.

Make a subdirectory called '`homework/hw3`' and place your files under that directory. Don't forget to commit and push your files! You can check on `http://mit.cs.uchicago.edu` to make sure all files were committed to the repository correctly.

The deadline for this homework is Monday April 17, 2017. To grade the homework, the contents of your repository at exactly the deadline will be considered. Changes made after the deadline are not taken into account.

1.3 Code samples

This document, and any file you might need to complete the homework can be found in the git repository
`https://mit.cs.uchicago.edu/mpcs51040-spr-17/mpcs51040-spr-17/`.

1.4 Grading

Your code will be graded based on the following points (in order of descending importance):

- Correctness of the C code: there should be no compiler errors or warnings when compiling as described in 1.1. There should be no memory leaks or other problems (such as those detected by valgrind).
- Correctness of the solution. Your code should implement the required functionality, as specified in this document.
- Code documentation. Properly documented code will help understand and grade your work.

- Code quality: your code should be easy to read and follow accepted good practices (avoid code duplication, use functions to structure your program, ...). This includes writing portable code (which will work on both 32 bit and 64 bit systems).
- Efficiency: your code should not use more resources (time or space) than needed.

2 Assignment

2.1 Palindrome

Write a function which checks if a given sentence is a palindrome.

> A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward or forward. Allowances may be made for adjustments to capital letters, punctuation, and word dividers.

When determining if something is a palindrome, ignore any non-alphanumeric character. Case does not matter.

So, the following is a palindrome:

```
This is an example, elpm axena sisiht.
```

Your program should read a line of input from the keyboard, and write 'PALINDROME' or 'NOT PALINDROME' (exactly matching case) to the output in response. The program should terminate when an empty line is entered. **Do not write anything else to the screen.** You can assume that no line will be longer than 1024 characters. Since the input will be line based (i.e. the program only considers the input once return is pressed), an empty line is a line for which *no alphanumeric characters were entered before hitting return*. This means that a line consisting only out of non-alphanumeric characters (such as ',' or spaces), followed by return, should also terminate the program.

We want the code to check if a string is a palindrome to be reusable, so we are going to put it in a separate `.c` file (with matching header). This will also allow easy testing of the function. The header (`palindrome.h` – **which should not be modified**) is already provided. It is up to you to write `palindrome.c` and `palindrome_driver.c` (the latter will contain your main function and the code to read from the keyboard).

Commit `palindrome.c`, `palindrome_driver.c` and a `Makefile` for building your program (program name `palindrome_driver`) to `homework/hw3/ex3`. Make sure to accurately express the dependencies in your Makefile, including headers! Also remember you'll have to combine both `.c` files into a single executable, otherwise you'll see linker errors...

2.2 Hangman Game

The task is to implement the well known 'hangman' game. You can find a terse description of the game here: `https://en.wikipedia.org/wiki/Hangman_(game)`.

For our version of the game, there are two players: a computer player, and a human player. The computer player starts by picking a random english word. The objective of the game is to guess which word was picked, by repeatedly guessing a letter. If the letter is part of the word, the positions in the word that have contain that letter are revealed. If not, another guess is tried. After a certain number of guesses, if the word has not been found yet, the game ends and the player lost. In our version of the game, **a player should be allowed 8 guesses**.

How to pick a random word? Luckily, on unix (and unix-like systems, such as Ubuntu), there is a file of newline-delimited dictionary words: `/usr/share/dict/words`. To keep things simple, **you should ignore any word containing any character other than a–z** (*ignoring case*). This makes the game easier, since otherwise one might end up with a word like 'éclair's'...

A skeleton for the program has been provided, and can be found in the homework directory of the class repository.

Tasks:

- Write a function called `pickword` which will read `/usr/share/dict/words` and returns a randomly chosen word which only consists out of letters of the alphabet. *Your function should return the word to lowercase before returning it.*
- Write a function which shows the current state of the gallows.

  The function should be called `showdiagram` and should take the current number of incorrect guesses, and output the diagram accordingly. What exactly this function outputs to the screen is up to you, but the output should progress towards a complete picture as the number of incorrect guesses approaches the limit. See figure 1 for an example. Your function should be able to output 9 different 'states': from 0 wrong guesses all the way up to 8 wrong guesses.
- Write a function which shows the guesses, incorrect guesses, and letter positions for the secret word. Call the function `showguesses`.
- Your game should accept upper and lower case letters as input, but ignore the case when matching guesses to secret word letters.
- You should not accept letters that have been provided earlier, i.e. if a letter is reused, print a warning and let the player choose again.

Some advice:

- Be careful with newline characters ('`\n`'). Depending on which function is used to read from the keyboard (or a file), the string returned might or might not include a newline character.
- Use valgrind to ensure your code does not have any detectable problems.
- There is a very simple algorithm for picking an item of a collection of n items (or a line out of n lines) without having to go over the collection twice or without having to know in advance how many items the collection contains...
- Make sure to 'seed' the random number generator using the `srand` function. The manpage gives an example of how to obtain a suitable seed based on the current time.
- *You are not allowed to modify the skeleton of the program. For example, you cannot change the prototypes of any of the predeclared functions.*
- Make sure to size your variables appropriately so that your program can handle even the longest words from the wordfile. It is safe to assume that no word in that file will exceed 255 characters.

Commit `hangman.c` and a `Makefile` for building your program (the program name needs to be `hangman`) to `homework/hw3/hangman`, and typing '`make`' should be enough to build your program.

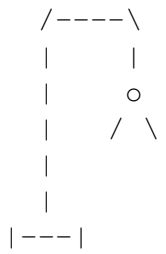

Figure 1: ASCII art example