CSCE 155 – C

Lab 8.0 – Strings

Prior to Lab

Prior to Lab

Before attending this lab:

1. Read and familiarize yourself with this handout.
2. Read Chapters 8 and 21 of the Computer Science I textbook
3. Watch Videos 8.1 thru 8.3 of the Computer Science I video series

Peer Programming Pair-Up

For students in the online section: you may complete the lab on your own if you wish or you may team up with a partner of your choosing, or, you may consult with a lab instructor to get teamed up online (via Zoom).

For students in the face-to-face section: your lab instructor will team you up with a partner.

To encourage collaboration and a team environment, labs are be structured in a *peer programming* setup. At the start of each lab, you will be randomly paired up with another student (conflicts such as absences will be dealt with by the lab instructor). One of you will be designated the *driver* and the other the *navigator*.

The navigator will be responsible for reading the instructions and telling the driver what to do next. The driver will be in charge of the keyboard and workstation. Both driver and navigator are responsible for suggesting fixes and solutions together. Neither

the navigator nor the driver is "in charge." Beyond your immediate pairing, you are encouraged to help and interact and with other pairs in the lab.

Each week you should alternate: if you were a driver last week, be a navigator next, etc. Resolve any issues (you were both drivers last week) within your pair. Ask the lab instructor to resolve issues only when you cannot come to a consensus.

Because of the peer programming setup of labs, it is absolutely essential that you complete any pre-lab activities and familiarize yourself with the handouts prior to coming to lab. Failure to do so will negatively impact your ability to collaborate and work with others which may mean that you will not be able to complete the lab.

1 Lab Objectives & Topics

At the end of this lab you should be familiar with the following

- Declare and print a string in C
- Manipulate strings in a variety of ways
- Use some basic functions from the `string.h` library

2 Background

Strings are collections of characters. In C, Strings are represented using arrays of char values terminated by a special null-terminating character, `\0`. Because they are arrays, the same precautions must be made when working with strings as with general arrays. The standard string library (`string.h`) provides many convenience functions to manipulate and use strings.

This lab will familiarize you with some of these functions. In particular, you will complete a program that implements a common children's game, horse (also known as hangman). In this game, an English word is chosen at random and its characters hidden. The player takes turns by guessing a letter; each instance (if any) of the guessed letter is revealed. If the user is able to guess the word before a certain number of guessed letters then they win. If they run out of guesses then they lose.

Most of the game mechanics have been implemented for you. However, you will need to complete the game by implementing several functions used by the game to manipulate and compare strings.

3 Activities

Clone the code for this lab from GitHub using the following URL: https://github.com/cbourke/CS1-C-Strings.

3.1 Implementing String Manipulation Functions

1. Navigate to the `src` directory and open `gameFunctions.c` in the editor of your choice.
2. There are several functions already fully implemented in this file. Your task for this lab is to implement the following four functions:
   - `initializeBlankString()` - this function should take two arguments and return nothing:
     - The first argument is an integer denoting the length of a string
     - The second argument should be a string
     - The function should alter the passed string and set all of its characters to the underscore, `_` and properly null-terminate it
   - `printWithSpaces()` - this function should take one argument and return nothing:
     - The argument should be a string
     - The function should print the contents of the string character-by-character with spaces between each one.
     - Hint: use the `strlen()` function to find the length of the passed string)
   - `revealGuessedLetter()` - this function should take three arguments and return an integer:
     - The first argument will be the solution string (that should not be altered; use the `const` modifier to ensure that it is not)
     - The second argument will be the (partially) revealed string that you will alter
     - The third argument will be a single `char`, the letter guessed by the user in the game
     - Iterate over each character in the second argument and "reveal" the letter if it matches the guessed letter
     - For example, if the first string is `"dinosaur"` and the second is `"________"` and the character passed is `a`, then the function should alter the second

string so that it becomes `"_____a__"`.

- You may assume that the strings are of equal length.
- The function should return a 1 if any letters were changed in the second string and 0 otherwise.

- `checkGuess()` - this function should take two strings as input and return an integer.
  - If the two strings are equivalent, return a 1 from the function. If they are different, return a 0.
  - Hint: make use the `strcmp()` function from the string library.

3. Navigate to the `include` directory and open `gameFunctions.h` in the editor of your choice
4. Complete the function prototypes that you implemented in `gameFunctions.c` here.
5. Compile the program using the `make` command and complete the worksheet.

4 Handin/Grader Instructions

1. Hand in your completed files:
   - `gameFunctions.c`
   - `gameFunctions.h`
   - `main.c`
   - `worksheet.md`

   through the webhandin (https://cse-apps.unl.edu/handin) using your cse login and password.
2. Even if you worked with a partner, you *both* should turn in all files.
3. Verify your program by grading yourself through the webgrader (https://cse.unl.edu/~cse155e/grade/) using the same credentials.
4. For this lab the grader will simulate playing several games (some intentionally losing, some winning). Each run may be slightly different but the output should correctly display the game being played.

5 Advanced Activity (Optional)

Currently, the game has a strict limitation on the number and length of words a user can enter in the `dictionary.txt` file. Alter the program so that it can accept any number of words and words of any length from `dictionary.txt` (hint: you'll need to dynamically allocate the memory for the array in `main.c`, among other changes in `gameFunctions.h` and `gameFunctions.c`).