PROGRAMMING ASSIGNMENT 1 CPSC 1020

PRACTICE WITH C

HOW MANY WORDS

DUE: OCTOBER 9, 2019, midnight

Learning Objectives:

This assignment is designed to provide practice with the following:

- Working with structs
- Allocating memory
- C-style file pointer
- Formatting output
- Working with multiple files
- Command-line arguments
- Reading data from files

Overview:

For this assignment you are going to write a simple word game. Your program will read and store a set of data, produce a 2D matrix of characters, print instructions for the player on how to play, and determine the amount of time it took the player to find a set of words.

You will be required to write several functions for this project. I realize this seems like a lot, and you would be correct. However, you should get in the habit of writing functions that do very specific task, rather than large functions that do multiple task. Developing functions that do small specific task not only make your code clean and readable, your functions will likely be more reusable than large multitask functions.

I will put an explanation of each of these functions in the functions.h file.

```
void readData(FILE*, matrixInfo_t*);
void printDirections(matrixInfo_t*);
void printMatrix(matrixInfo_t*);
int readUserInput(matrixInfo_t*);
int checkWords(char*, matrixInfo_t*);
void allocateLetters(matrixInfo_t*);
void allocateWords(matrixInfo_t*);
void readLetters(FILE*, matrixInfo_t*);
void readWords(FILE*, matrixInfo_t*);
void checkArguments(int);
void free1DMemory(matrixInfo_t*);
void free2DMemory(matrixInfo_t*);
```

PROGRAMMING ASSIGNMENT 1 CPSC 1020

PRACTICE WITH C

HOW MANY WORDS

DUE: OCTOBER 9, 2019, midnight

Below is a sample data file with an explanation next to the data. You can use this data to test your program. I will test your program with this data, as well as several other test cases.

```
3 3          This means the gameboard will be a 3 X 3
s h o h u t i g h   These are letter that will occupy the gameboard
12           This number represents the # of possible words found on the board.
shout
out
hi
high
to
hut
gut
his
thus
is
shut
hug
```

These are the words I found in the gameboard. This does not mean these are the only words, they are the choices I have included in the answer.

1. You must provide the following files:
    a. driver.c
    b. functions.c
    c. makefile – your make file must provide following
        i. make run
        ii. make clean
    d. Readme – later in this document

    I will provide the functions.h file with an explanation of each function.
2. Currently the struct in the .h file does not use typedef. You will need to change this to use typedef to allow the use **matrixInfo_t** when declaring a variable of the the struct
3. You will, of course, need to use command line arguments that consist of two things:
    a. an executable
    b. the name of the data file.

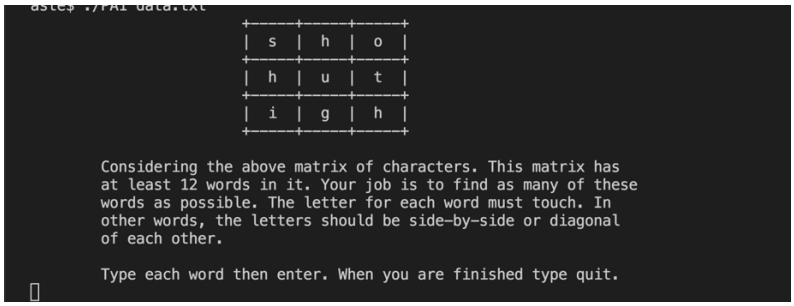
PROGRAMMING ASSIGNMENT 1 CPSC 1020

PRACTICE WITH C

HOW MANY WORDS

DUE: OCTOBER 9, 2019, midnight

Below is a screenshot of the game after the data has been read in and the appropriate functions called to start the game.:



The player will type a word then enter. Your game will check if the word is in the list of words provided. The game board should be displayed after each iteration of a word being entered. This will ensure the user can see the matrix at all times. The game will maintain a count of the number of correct and unique words the player found. When the player believe she has found all words, she will enter "quit". The game will display a message showing the number of correct and unique words found. "Full disclosure, when running my version of the game I realized I did not check if I guessed the same word twice. You are required to perform this check."

Your matrix **must** look like the matrix above. You **must** use a series of "–" as well "+" and "|" to create the boxed outline of the gameboard. At a minimal, your message must be the same as pictured above. You may be creative with your message; however, you must have at least the above.

Points will be deducted for not following direction.

4. You must dynamically allocate the memory for the characters that go in the game. This must be a 2D matrix.
5. You will also provide a timer to determine the amount of time it took the user to find as many words as possible. This will require you to look up the time_t data type and time() function. You will need to read the time just prior to calling the readUserInput function and read the time after this function has returned. This should give you appropriate information to calculate the number of seconds it took for the player to find the words. The number should be displayed in minutes and seconds. Ex. If it took the player 80 seconds to find the words, at a minimal display the following:

   The amount of time for you to find 12 words was 1 min - 20 sec.

6. Although the example given uses a 3X3 matrix you are to write your code such that any size matrix could be read from an input file. Ex. I could provide an input file that uses a 5X8. Your

PROGRAMMING ASSIGNMENT 1 CPSC 1020

PRACTICE WITH C

HOW MANY WORDS

DUE: OCTOBER 9, 2019, midnight

program should be able to handle any size of matrix and any number of words. All input files will follow the same format as the format given above.

7. Remember for each allocation of memory there must be a call to free. There are two functions that you will need to complete that should be called to free memory.
    a. void free1DMemory(matrixInfo_t*);
    b. void free2DMemory(matrixInfo_t*);
8. Your main should have a minimal amount of code it it. It should:
    a. call checkArguments
    b. create and open a file pointer, which will be used to read the data file
    c. create any needed local variable to pass to functions
    d. call the function necessary to start the process of reading the data
    e. call the function to print the matrix
    f. call the function to print the directions for the game
    g. get the start time
    h. call the function to start reading the user input (read until the user enters quit)
    i. get the end time
    j. print the user time and number of words found. (See example output above.)

Extra Credit Option 1
For each of the following 2 Extra Credit opportunities, the basic game requirements will stay the same. You may choose one of the 2 EC to complete.
5 points
After displaying the final time and # of words found. Add some type of loop that ask the user if they want to pay again with the same set of data to try to beat their original time. After each game, continue to ask until the player says no. You will need to keep track of their time and number of words found. Determine the best game by determining the largest number of words found in the shortest amount of time. Display a message with this information to the player.

Extra Credit Option 2
10 points

In **addition** to giving the above EC option to the user, give the player a **second** option of playing again with a different set of data of the same size. After each game continue to ask until the player says no, keeping track of the time and number of words found. Display the best game by determining the largest number of words found in the shortest amount of time. Display the message with this information to the player.

Formatting, Compiling, and Hand-in Requirements:

PROGRAMMING ASSIGNMENT 1 CPSC 1020

PRACTICE WITH C

HOW MANY WORDS

DUE: OCTOBER 9, 2019, midnight

Tar your file **PA1.tar.gz** and submit the file through **hand-in to the folder PA1.** If you are doing the EC you must submit a tarred file called **PA1EC.tar.gz** through **hand-in to the folder PA1EC.** **PLEASE SUBMIT TO ONLY ONE HAND-IN FOLDER.** You should submit the following files.

- driver.c
- functions.c
- functions.h
- makefile
- Readme that has the following information:
  - problems you encountered with this assignment
  - how you solved the problems
  - your thoughts about the assignment.
- With the exception of the makefile, you should have a header in each of your files that contains the following information. If you neglect to place a header in a file, you will receive a 5-point deduction.

```
/*************************/
 *Your name                *
 *CPSC1020 Summer 2017     *
 *UserName:                *
 *Instructor:  Dr. Yvon Feaster  *
/*************************/
```

- Your program should compile with no warnings and no errors on the School of Computing servers. There will be a substantial point deduction if your program has compile errors. Ifyou made a substantial effort to complete the program, but you have compile errors, you will receive no more than 30 on the assignment. If the program compiles but has warnings, there will be a minimal of 10-point reduction. (The more warnings you have the higher the deductions.)
- You should use meaningful variable names throughout the program.
- Your code should be well documented. (comments – see example below)
- There should be no line of code longer than 80 characters.
- You should use proper and consistent indention.

Failure to do any of the above items will result in a deduction of points for each offense.

Here are some guide lines for documenting the code in this assignment.

PROGRAMMING ASSIGNMENT 1 CPSC 1020

PRACTICE WITH C

HOW MANY WORDS

DUE: OCTOBER 9, 2019, midnight

Before each function**, in the functions.h file**, you should have a detailed description of what the overall function does. To borrow from another student's code, here is an example of overall function description.

```
/* Parameters: img - image_t pointer array holding the image data for
 *                   each of the input files
 * Return:       output - image_t struct containing output image data
 * This function averages every pixels rbg values from each of the
 * input images and puts those averages into a single output image
 */
```

You are required to have this type of comment block before each function.

Also, if you include comments in the body of the function (and you should) they should be placed above the line of code not beside the code.

Example:

Good

```
//This is a comment
if(something)
{
    do something;
}
```

Bad

```
if(something) //This is a comment
{
    do something;
}
```