COP 3530 – Data Structures and Algorithms I

Project 1

Due: Friday, February 2nd 11:59 P.M. CST

Objective:

This project is meant to help you review concepts of programming in C while also getting used to the conventions used for programming in this course. In this project, you are asked to create the game of Hangman. You will need to be able to perform basic file input as well as to create and free arrays. You will also need to create a makefile to compile your program. Information about creating makefiles is up on eLearning and zyBooks.

Problem Description:

You will be given files in the following format:

```
n
word1
word2
word3
word4
```

The first line of the file will be a single integer value n. The integer n will denote the number of words contained within the file. Use this number to initialize an array. Each word will then appear on the next n lines. You can assume that no word is longer than 30 characters. The game will use these words as the puzzles for the game of Hangman. You will need to randomly select one of the words from array. Read "Getting a Random Number" for an explanation on how to do this.

For those of you that have never played the game of Hangman before, the idea behind the game is to be able to guess all the letters in a puzzle before a certain number of missed guesses. If a player guesses an incorrect letter, then they have one less incorrect guess that they can make. If a player guesses a correct letter, then they can make the next guess without any penalties. In the case of our game, we will allow for 7 missed guesses. With each guess, a letter for the word "HANGMAN" is added to the screen. Once the complete word is spelled out, the player loses. If the player correctly guesses all letters in the puzzle before HANGMAN is spelled, then they win and can either choose to start another game or quit the game.

Creating the Game

You are required to create three different files: hangman.c, hangman.h, and client.c. The files hangman.c and hangman.h contain the logic for your game, while the file client.c is the main entry point for your program. Upon the start of the game, your terminal should display the following:

```
Welcome to Hangman!
Please enter the name of the file containing your puzzles:
```

The player will then type the name of a file with their puzzles into the terminal. Afterward, the screen should look similar to the following:

Welcome to Hangman!
Please enter the name of the file containing your puzzles: puzzles.txt

Current Puzzle: _ _ _ _ _ _ _
Missed Guesses:
Please guess a letter:

We will use the character '_' to indicate a missing letter. "Missed Guesses" refers to the number of missed guesses that the player has and this is where you will spell out "HANGMAN". The player is being prompted to enter a possible letter. At this point, one of two things should happen. Either the player guesses a correct letter or they guess an incorrect letter. For simplicity, you may assume that the player will only guess letters and that no other inputs will be entered (this is not a safe assumption in real world applications!).

If they make a correct guess, the screen will look like this:

Current Puzzle: _ _ _ _ _ _ _
Missed Guesses:
Please guess a letter:o

Current Puzzle: _ _ _ _ o o _
Missed Guesses:
Please guess a letter:

If they make an incorrect guess, the screen will look like this:

Current Puzzle: _ _ _ _ _ _ _
Missed Guesses:
Please guess a letter:k

Current Puzzle: _ _ _ _ o o _
Missed Guesses: H
Please guess a letter:

The game will proceed in this way until either they correctly guess all letters in the puzzle or until they spell they word "HANGMAN". At which time, the game should display an appropriate message and prompt the player to either play again or load a new puzzle file (this means you can create Hangman games based on different themes if you would like):

Current Puzzle: c a r t o o n
Missed Guesses: HANGM
Please guess a letter:

You win!!
Would you like to:

1. Play a new game
2. Load a new file of puzzles

Please make a selection:

If the player chooses to load a new file, the game will prompt the player to enter a new file as they did when first running the program.

The Hangman Files

You will need to create a struct that can hold the array of puzzle words, the number of puzzles currently loaded, the current puzzle word, the guessed word to the current point in time (filled with correctly guessed letters and '_' characters), a boolean (use stdbool.h, refer to references in zyBooks for this if needed) for whether the game is over, a boolean for whether the game is won, and the number of missed guesses thus far. Note that nothing is printed from this file. We are purposefully separating the logic of Hangman from the view that gets displayed to the user. Your file should have the minimum set of functions:

- **Hangman createHangmanGame(char *puzzleFile)** – Will create the main Hangman game while loading the initial set of puzzles into the struct. You can initialize the words and to NULL and the current number of missed guesses to 0.
- **void newHangmanPuzzle(Hangman currentHangmanGame)** – Sets up a new puzzle for the player to guess. You will need to randomly select a new puzzle from the array of puzzles in the game.
- **void loadPuzzleFile(Hangman currentHangmanGame, char *puzzleFile)** – Will free the previous array for puzzles and load a new set of puzzles from the input file.
- **bool isPuzzleOver(Hangman currentHangmanGame)** – Will return whether or not the current puzzle has reached a state where either the player has won or lost.
- **bool isPuzzleSolved(Hangman currentHangmanGame)** – Will return whether or not the current puzzle has been solved.
- **char* getGuessedWord(Hangman currentHangmanGame)** – Will return the string for the current state of the word that has been guessed thus far (the string that contains both the guessed letters and remaining '_').
- **bool guessLetter(Hangman currentHangmanGame, char letterToGuess)** – Will allow the user to guess a letter and then return whether the guessed letter was part of the puzzle word. This function should update the word guessed thus far or increase the number of missed guesses depending on whether the guess was correct.

- **char* getStateOfHangman(Hangman currentHangmanGame)** – Will return the string for the current state of “HANGMAN”.
- **void freeHangmanGame(Hangman currentHangmanGame)** – Will free the memory allocated for the game of Hangman.

These are the minimum functions required by this project. Place all function signatures into hangman.h. Your struct and the implementation of your functions should be in hangman.c.

Getting a Random Number

You must select a puzzle from your array at random. In C, this can be accomplished by using the rand() function from stdlib.h. It is important to note that before using the random number generator you will need to provide the random number generator with a seed value, this can be done by using the existing time. An example of using the rand() function to pick a random number from 0 – 9 is as follows:

```
#include <stdlib.h>
int main () {
     time_t t;

     /* Intializes random number generator */
     srand((unsigned) time(&t));
     printf("%d\n", rand() % 10);

     return 0;
}
```

Note that we have to mod the result of rand by 10 to get a number within the range 0 – 9. You will be using the number of elements in your puzzles array as the mod value in the case of the Hangman game.

The Client

Your client file handles all display messages, running the game, and taking in user input. Separating the client that displays the menus to the player from the game of Hangman allows us to reuse the existing Hangman game in different programs or to change out our current view for a new one if we so choose.

Breakdown of Grades:

The following is a breakdown of how points are distributed to each part of your project:

1. hangman.h and hangman.c – properly formatted with only the correct details in each (55 points)

2. Create the client for your program. You should adequately test your program with multiple inputs. (30 points)
3. Provide a makefile that will properly compile your program. (15 points)

In addition, please add your name as a comment to the top of each file.

Submission Instructions:

Make sure that before submitting your program that you have tested it on the ssh server (ssh.cs.uwf.edu). There is information for how to do this on eLearning. For this project, submit a zip file containing all files required to run your project, these files will be named hangman.c, hangman.h, client.c, and makefile. Submit your project using the eLearning dropbox. Remember that all students who turn in their projects by the due date that receive a B or better will receive an added 0.5% on their final overall average.