|  |  |  |
| --- | --- | --- |
| **LAB101 Assignment** | **Type:** | **Long Assignment** |
| **Code:** | **C.L.P0006** |
| **LOC:** | **150** |
| **Slot(s):** | **3** |

**Title**

Hangman.

**Background Context**

An airline reservation system (ARS) is part of the so-called passenger service systems (PSS), which are applications supporting the direct contact with the passenger.

ARS eventually evolved into the computer reservations system (CRS). A computer reservation system is used for the reservations of a particular airline and interfaces with a global distribution system (GDS) which supports travel agencies and other distribution channels in making reservations for most major airlines in a single system.

**Program Specifications**

You are given a puzzle with blanks (representing letters) and you guess a letter that might be in the puzzle. If it is, then each occurrence of the letter is placed on the board. If not, you get a point against you (which is usually drawing one portion of a stick-figure.) If you complete the puzzle before too many wrong guesses (this usually depends on how many parts there are to the stick-figure being drawn), then you win. If not, you lose. For our game, we won't draw any stick figures, but we'll simply keep track of how many incorrect guesses the user has made. In order to win, the user must uncover the phrase before they get 5 incorrect guesses.

For this game, all of the words to be guessed will have uppercase letters only. All of the possible puzzles will be read in from a file into a two-dimensional character array (array of strings). The code for this will be given to you in a separate file. You will have to call the given function only once in your program. From there, each time the user wants to play the game, you'll pick a different random word stored in the array of strings as the word for the puzzle. Note: All words are guaranteed to be 29 characters or fewer and the file is guaranteed to have 1000 words or fewer..

***Function details:***

At the beginning of the program, you should prompt the user to enter the name of the input file with all of the words for the game.

Then, you should ask the user if they want to play. If they do, choose a random puzzle, and present it to them with all the letters hidden represented by underscores and ask them what letter they want to choose. Separate each slot with a blank. Here is an example:

*You currently have 0 incorrect guesses.*

*Here is your puzzle:*

*\_ \_ \_ \_*

*Please enter your guess.*

After the user enters their guess, if the letter they guessed is in the puzzle and hasn't been guessed before, print out the following message:

*Congratulations, you guessed a letter in the puzzle!*

If the letter is NOT in the puzzle, print out the following message:

*Sorry, that letter is NOT in the puzzle.*

If the user has ALREADY guessed that letter, print out the following message:

*Sorry, you have guessed that letter already.*

*Now it counts as a miss.*

Once you've printed this out, continue to the next turn, prompting the user with the board again as described above.

If the user has 5 incorrect guesses, instead of reprompting them with the board, print out the following message:

*Sorry, you have made 5 incorrect guesses, you lose.*

*The correct word was \_\_\_\_.*

where \_\_\_\_ is the word for the round.

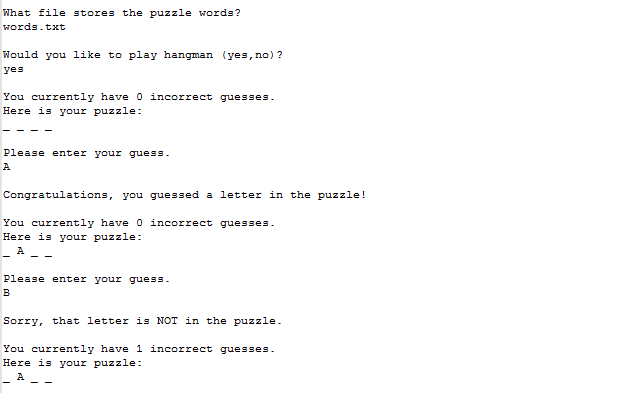
If the user fills in the entire word with the letters they guess before 5 incorrect guesses, print out the following message:

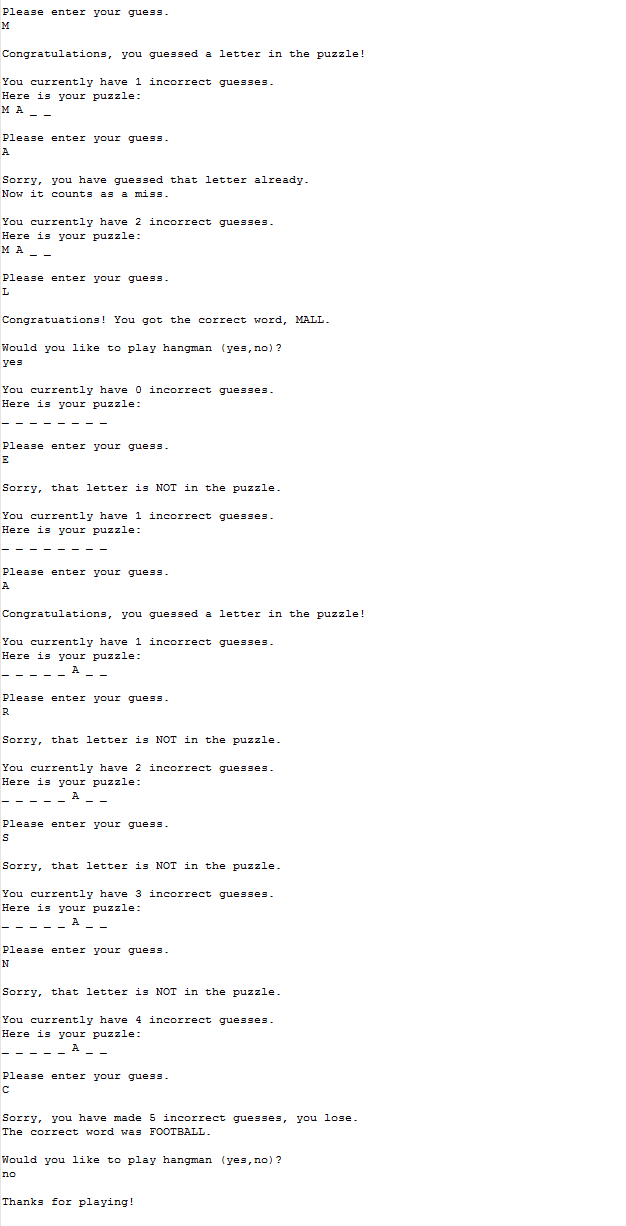
*Congratuations! You got the correct word, \_\_\_\_.*

where \_\_\_\_ is the word for the round.

After the round is over, prompt the user if they want to play again. If so, repeat the process described here.

***Expectation of User interface:***





**Guidelines**

Store two strings for the program: one which is the correct word, and the other which is what gets printed out to the user. For example, if the correct word was MALL, store the two following strings:

MALL

\_\_\_\_

When the user guesses a letter, read it into a string and just access index 0 of that string to find the character they guessed. (This will make dealing with whitespace easier.)

Scan through the real word, checking to see if any of the letters equals the guessed letter. If so, replace the corresponding character (at the same index) in the other string. So, for example if A were guessed, the two strings would then look like:

MALL

\_A\_\_

To keep track of which letters have been guessed and which ones haven't, store an integer array of size 26 and initialize it to all 0s, indicating that no letter had been guessed. When a letter is chosen, go to the appropriate index and add 1 to the value stored there. For example, if the letter chosen is stored in guess[0], and the name of the array is chosen, we would do the following:

chosen[(int)(guess[0] – 'A')]++;

Since the ascii values are contiguous for capital letters, guess[0] – 'A' will equal a value from 0 to 25. It will equal the index which stores whether or not that letter has been guessed.

To detect if the puzzle has been guessed, just check if the two separate strings are equal.

To detect if the user has lost, just keep a single integer variable that stores the number of incorrect guesses and check if that equals 5 (which should be stored in a constant.)

To print out the word for the user, write a function which prints out one letter from the string followed by a space, repeated throughout the word, so that

\_ A \_ \_

gets printed instead of

\_A\_\_