INF1002 Programming Fundamentals – C Lab3

OBJECTIVES:

1. Understand and apply pointer concepts.
2. Understand and apply the relationship between pointers and arrays in C.

EXERCISE-1: POINTER SYNTAX

Assume that the following lines of code have been executed.

```
int *zPtr; /* zPtr will reference array z */
int *aPtr = NULL;
void *sPtr = NULL;
int number, i;
int z[ 5 ] = { 1, 2, 3, 4, 5 };
zPtr = z;
sPtr = z;
```

Find the error in each of the following fragments of code to be executed after the code above. Write a simple programme to check your answers by correcting the error and compiling it.

a)

```
/* use a pointer to get the first value of array */
number = zPtr;
```

b)

```
/* assign array element 2 (the value 3) to number */
number = *zPtr[ 2 ];
```

c)

```
/* print the entire array z */
for ( i = 0; i <= 5; i++ ) {
        printf( "%d ", zPtr[ i ] );
}
```

EXERCISE-2: POINTER BASICS

For each of the following, write a single statement that performs the task indicated. Assume that long integer variables value1 and value2 have been defined and that value1 has been initialised to 200,000.

a) Define the variable ***lPtr*** to be a pointer to a variable of type long.
b) Assign the address of variable value1 to pointer variable ***lPtr***.
c) Print the value of the variable pointed to by ***lPtr***.
d) Assign the value of the variable pointed to by ***lPtr*** to variable value2.
e) Print the value of value2.
f) Print the address of value1.
g) Print the address stored in ***lPtr***. Is the value printed the same as the address of value1?

Check your answers by writing a program that performs all the above steps.

Lab Assignment:

To help you better practice, you need to perform a set of tasks in one auto-grading system, Gradescope in LMS/xSiTe. Gradescope will provide you immediate feedback of your program, such that you will know the issue of your program.

In this lab, you need to finish one task in Gradescope system. Gradescope system will provide you immediate feedback about the correctness of your program. You can view the feedback of your system by clicking the failed case. Gradescope adopts a test cases-based approach to check your program. Your score is depending on the number of test cases that you program can pass. Note that to train you to have a good programming practice, you must write your program strictly according to the requirement of the tasks, including your input and output format. If there is any difference (even one space), your program will fail on the test cases. SO, TRAIN YOURSELF TO BE AN EXACT THINKER!

To help you practice, you are allowed to do multiple attempts for each task. Enjoy your learning!

Task: GUESS WORD

Task Description:

Write a program called ***guessWord*** that plays a two-player word-guessing game using a similar procedure to the *guessInteger* program from C Lab1. The game will proceed as follows:

1. Player 1 will be asked to enter a word of up 12 letters. The word should contain only the lower-case English letters from 'a' to 'z', and no punctuation marks or digits.
    a. If Player 1 enters a word with upper case letters, the program should change them to lower case.
    b. If Player 1 enters a word with punctuation marks or digits, he or she should be asked to enter another word.
    c. The program does not need to check that the word is a "real" word (i.e. in a dictionary).
2. Player 2 (who again has not been watching Player 1) will be asked to guess one letter at a time.
    a. Player 2 has maximum 7 guesses.
    b. At the beginning of each round, the program will output a row of characters containing one underscore for every letter in the word to be guessed. If Player 2 has previously guessed a letter that is in the word, the underscore will be replaced by that letter.
    c. Player 2 will enter one letter. If he or she enters an upper-case letter, the program will convert it to lower case. If he or she enters a punctuation mark or digit, he or she will be considered to have made an incorrect guess.
    d. If the letter is not in the word, the number of incorrect guesses will be incremented.
    e. If the letter is in the word, every position in the word in which that letter occurs will be revealed at the start of the next round.
3. The game ends when either Player 2 has guessed all the letters of the word, or when Player 2 has made seven incorrect guesses.

Note:

- Use macros (*#define*) to represent all constants, so that it is easy to change things like the number of guesses allowed, the highest number allowed, and so on.
- Don't use *sys.argv[]* for user inputs. Use other functions such as *scanf(), fgets(), fgetc(),* etc.,

- You can use the *ctype.h* and *string.h* libraries for manipulating characters and strings.
- You may find it useful to write "helper" functions that perform tasks like checking that Player 1's string is valid, whether and where Player 2 has correctly guessed a letter, and so on.
- There is no white space in the print after the colons (:).
- All print statements terminate with a new line (\n).
- When only one guess is remaining, use "guess" instead of "guesses".
- Include comment to your code that explains what each section of it does.

Some sample output is shown below, with the user input shown in red:
Example – 1:

```
Player 1, enter a word of no more than 12 letters:
Topsy-turvy
Sorry, the word must contain only English letters.
Player 1, enter a word of no more than 12 letters:
Cat
Player 2 has so far guessed:

_ _ _
Player 2, you have 7 guesses remaining. Enter your next guess:
e
Player 2 has so far guessed:

_ _ _
Player 2, you have 6 guesses remaining. Enter your next guess:
a
Player 2 has so far guessed:
_ a _
Player 2, you have 6 guesses remaining. Enter your next guess:
c
Player 2 has so far guessed:
c a _
Player 2, you have 6 guesses remaining. Enter your next guess:
t
Player 2 has so far guessed:
c a t
Player 2 wins.
```

Example – 2:

```
Player 1, enter a word of no more than 12 letters:
computer
Player 2 has so far guessed:
_ _ _ _ _ _ _ _
Player 2, you have 7 guesses remaining. Enter your next guess
a
Player 2 has so far guessed:
_ _ _ _ _ _ _ _
Player 2, you have 6 guesses remaining. Enter your next guess:
b
Player 2 has so far guessed:
_ _ _ _ _ _ _ _
Player 2, you have 5 guesses remaining. Enter your next guess:
d
Player 2 has so far guessed:
_ _ _ _ _ _ _ _
Player 2, you have 4 guesses remaining. Enter your next guess:
e
Player 2 has so far guessed:
_ _ _ _ _ _ e _
Player 2, you have 4 guesses remaining. Enter your next guess:
f
Player 2 has so far guessed:
_ _ _ _ _ _ e _
Player 2, you have 3 guesses remaining. Enter your next guess:
g
Player 2 has so far guessed:
_ _ _ _ _ _ e _
Player 2, you have 2 guesses remaining. Enter your next guess:
h
Player 2 has so far guessed:
_ _ _ _ _ _ e _
Player 2, you have 1 guess remaining. Enter your next guess:
i
Player 2 has so far guessed:
_ _ _ _ _ _ e _
Player 1 wins.
```

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file "*guessWord.c*" from the "C\Lab\Lab3" xSiTe folder.
2. Do not change file name or function names.
3. Add your code to the function "*main()*" and other required functions.
4. Don't use *sys.argv[]* for user inputs. Use other functions such as *scanf(), fgets(), fgetc(),* etc.,
5. Drag and drop your locally tested code to Gradescope. You can submit and auto graded any number of times. Last submission counts.
6. There are 5 test cases and maximum score for this task is 5.
7. Autograder will be timed out after 10 minutes.