INF1002 Programming Fundamentals
C Lab2

OBJECTIVES:

1. Write functions to implement some features.
2. Understand and apply array concepts.
3. Understand C strings and use the standard string library for manipulating strings.

EXERCISE-1: ARRAY EXPRESSIONS

Suppose that the following declarations and assignments have been made in a C program:

```
int a[4] = { -1, 2, 10, 7 };
int b[4];
for (int i = 0; i < 4; i++)
    b[i] = a[3 - i];
```

Write a program to print the values for each of the following expressions.

a) a[3]
b) b[3]
c) b[a[1]]

EXERCISE-2: CHARACTER AND STRING EXPRESSIONS

Suppose that the following declarations and assignments have been made in a C program:

```
char *a = "abcdef";
char b[7];
strcpy(b, a);
for (int i = 0; i < 3; i++)
    b[i] = b[i] + 1;
b[3] = '\0';
```

Write a program to print the values for each of the following expressions?

a) a[0]
b) b[0]
c) b[4]
d) strlen(a)
e) strlen(b)
f) strcmp(a, b)

EXERCISE - 3: CHARACTERS AND STRINGS

Write a program that asks the user to type in a sentence of up to 255 characters in length. Your program will then divide the sentence into its individual words (indicated by a space character or punctuation mark) and print them line by line. Each line will include the word and the number of characters in it, as shown below:

```
Enter a sentence, up to 255 characters:
The cat sat on the mat.

The  3
cat  3
sat  3
on   2
the  3
mat  3
```

For an extra exercise, choose a secret "magic word" to be recognised by your program. If the sentence contains the magic word, add the text "You said the magic word!" at the end of the table.

Hints:

- Use *`fgets(buffer, n, stdin)`* to read a line of text that includes spaces, where buffer is the array into which you want to place the characters and n is the maximum number of characters to read.
- You can use the functions defined in *`ctype.h`* to test whether a character is a letter, a space, or a punctuation mark.

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: TINY GREP

Task Description:

Unix systems provide a utility known as grep that searches the lines of a file for a given pattern of characters. (The pattern is known as a regular expression, and the name grep comes from "global regular expression print"). We won't learn how to use files until Week 12, so we'll just search a string entered at the keyboard. We'll also support only very simple patterns.

Write a program, called *`tinyGrep.c`*, that performs as follows:

1. The program asks the user to enter a line of text of up to 255 characters.
2. The program asks the user to enter a pattern (a string), also of up to 255 characters.

3. The program asks the user whether the match should be case-sensitive or case-insensitive.
4. The program outputs whether the pattern occurs anywhere in the line of text, and, if it does, the index of the string at which the first instance of the pattern occurs.

The rules for patterns are as follows:

- Any English letter matches itself. If the match is case-sensitive, lower-case letters match only lower-case letters, and upper-case letters match only upper-case letters. If the match is case-insensitive, lower-case letters match upper-case letters, and vice versa.
- A dot (.) matches any character.
- An underscore (_) matches any form of whitespace (i.e. any character for which `isspace()` returns a true value).
- All other characters match only themselves.

The following table shows some examples.

Text	Pattern	Case-Sensitive	Output
`The cat sat on the mat.`	`cat`	N	`Matches at position 4.`
`The cat sat on the mat.`	`rat`	N	`No match.`
`The cat sat on the mat.`	`at`	N	`Matches at position 5`
`The cat sat on the mat.`	`.at`	N	`Matches at position 4.`
`The cat sat on the mat.`	`the`	N	`Matches at position 0.`
`The cat sat on the mat.`	`the`	Y	`Matches at position 15.`
`The cat sat on the mat.`	`...`	Y	`Matches at position 0.`
`"Hello," said the cat.`	`,`	N	`Matches at position 6.`

You might like to proceed as follows:

- Write a main program that reads the strings and case-sensitivity information as described above.
- Don't use `sys.argv[]` for user inputs. Use other functions such as `scanf()`, `fgets()`, `fgetc()`, etc.,
- Ignoring the pattern-matching rules for now, write a loop that simply searches the input text for an occurrence of the pattern string using `strncmp()` or similar function.
- Replace `strncmp()` with a new function that matches case-sensitively or case-insensitively depending on the user's answer to this question.
- There is no white space in the print after the colons (:).
- All print statements terminate with a new line (\n).
- Modify your new function to handle dot and underscore characters according to the rules above.

Sample Program inputs and outputs:
Example - 1:

```
Enter a line of text (up to 255 characters):
The cat sat on the mat.
Enter a pattern (up to 255 characters):
cat
Should the match be case-sensitive? (Y/N):
N
Matches at position 4.
```

Example - 2:

```
Enter a line of text (up to 255 characters):
The cat sat on the mat.
Enter a pattern (up to 255 characters):
rat
Should the match be case-sensitive? (Y/N):
N
No match.
```

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file "*tinyGrep.c*" from the "C\Lab Assignments\Lab2" 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 10 test cases and maximum score for this task is 5.
7. Autograder will be timed out after 10 minutes.