ICT1002 – LAB – WEEK 9

Functions, Arrays, and Strings

1. 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.

2. EXERCISES FOR WEEK 9 LAB

WEEK_9_LAB_EXE_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]]

WEEK_9_LAB_EXE_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)

WEEK_9_LAB_EXE_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.

Submit your tested source code for Exercises 1-3 to repl.it by 11:59PM on 8 Nov 2020.

3. WEEK_9_LAB_ASSIGNMENT: TINYGREP

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`, 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.`
`"Hello," said the cat.`	`he_c`	`N`	`Matches at position 15.`

You might like to proceed as follows:

1. Write a main program that reads the strings and case-sensitivity information as described above.
2. 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.
3. Replace `strncmp()` with a new function that matches case-sensitively or case-insensitively depending on the user's answer to this question.
4. Modify your new function to handle dot and underscore characters according to the rules above.

Other approaches are possible.

Submit your tested source code to repl.it by 11:59PM on 8 Nov 2021.