Department of Computer Science and Engineering
Institute of Technical Education & Research, SOA, Deemed to be University

MINOR ASSIGNMENT-004-I
String and String Tokenization
UNIX Systems Programming (CSE 3041)

Assignment Objectives:

Working with command line arguments, ~~process environment, shell metacharacters,~~ and string tokenization using `strtok()` & `strtok_r()` string handling library functions.

Instruction to Students (If any):

This assignment is designed to give practice with strings, string processing, array of pointers to strings and string library functions. Students are required to create their own programs to solve each and every question/problem as per the specification of the given question/problem to meet the basic requirement of Unix systems programming. **Students are required to write the output/ ~~paste the output screen~~ shots onto their laboratory record after each question.**

** NOT TO SOLVED IN GROUP **.

Programming/ Output Based Questions:

1. Consider the given 'C' line for main function **`int main(int argc, char *argv[])`**. Here, the **`argc`** parameter contains the number of command-line tokens or arguments, and **`argv`** is an array of pointers to the command-line tokens. The **`argv`** is an example of an argument array. Write a complete C code with command-line arguments to main and check/verify the outputs.

   (a) Print the number of command-line arguments you have passed

   (b) Run the code **`$ ./a.out ITER SOA IBCS`**, and print all the command-line tokens.

   (c) Run the code **`$ ./a.out "12 34 56"`**, and display the total number of arguments.[Hint: grouping of tokens]

   (d) Run the code **`$ ./a.out ITER SOA IBCS ``23 45 67''`**, and display the total number of arguments.

   (e) Run the code **`$ ./a.out ./a.out 12\34\56`**, and display the total number of arguments.

   (f) Run the code **`$ ./a.out ./a.out '12\34\56'`**, and display the total number of arguments.

   (g) Run the code **`$ ./a.out ./a.out ``12\34\56''`**, and display the total number of arguments.

   (h) Run the code **`$ ./a.out ./a.out 12  14  45  66`**, and display the total number of arguments. Multiple blank spaces are given inbetween numbers.

   (i) Write the below by putting enter at end of each line

```
./a.out 12\
34\
56\
78
```

   [**NOTE::**] A backslash at the end of a line causes the line to be continued. It is a way to present a verylong line to the shell.

   (j) [redacted]

(k) [redacted]

2. Write a 'C' program that uses the C library function **strtok()** to split a string into tokens and also display the number of tokens. Type the command **man 3 strtok** to get the description of the library function. The **strtok()** functions return a pointer to the next token, or NULL if there are no more tokens.[**NOTE::** The first call to **strtok** is different from subsequent calls. On the first call, pass the address of the string to parse as the first argument. On subsequent calls for parsing the same string, pass a NULL for the first argument. The second argument to **strtok**, is a string of allowed token delimiters]. Each successive call to **strtok** returns the start of the next token and inserts a '\0' at the end of the token being returned. The **strtok** function returns NULL when it reaches the end of the string to be parsed. Sample inputted string: (1) ITER-IBCS-SOA-IDS-SUM-CSE (ii) iter ibcs soa ids sum

3. The below given code snippet demonstrate the use of multiple delimiters with **strtok**, the second argument is a C string with the list of delimiters in it. Run the sample code to get the desired tokens as output.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

int main()
{
    char str[] ="ITER:IBCS;SOA:Pot*Hot";
    char* token;
    token=strtok(str,":;*");
    while (token!=NULL){
        printf("Token=%s\n",token);
        token=strtok(NULL,":;*");
    }
    return 0;
}
```

4. Now, change the string to **char str[ ] ="*ITER:IBCS;SOA:Pot*Hot:"**, and test the output for tokens. You will be getting the conclusion; "delimiter bytes at the start or end of the string are ignored".

5. For example, given the string **TOC;;PLC,USP;**, use successive calls to **strtok()** to get the token of strings "TOC", "PLC", and "USP", and then a NULL pointer.

6. In command line you have to type **`./a.out 'CSE/CSIT//EEE/EC//MECH//CIVIL:MBA:MBBS' ':' '//'`** arguments. You are required to write a 'C' code to tokenize the command-line arguments. The first command-line argument(i.e argv[1]) specifies the string to be parsed. The second argument specifies the delimiter byte(s) to be used to separate that string into "major" tokens. The third argument specifies the delimiter byte(s) to be used to separate the "major" tokens into subtokens.

```
Token 1: CSE/CSIT//EEE/EC//MECH//CIVIL
   Subtoken: CSE CSIT EEE EC MECH CIVIL
Token 2 MBA
   Subtoken: MBA
Token 3 MBBS
   Subtoken: MBBS
```

7. Write a 'C' code using **`strtok()`** to determine the average number of words per line. Refer **Program 2.3** of your text book.

8. The **`strtok_r()`** function is a reentrant version **`strtok()`**. The **`strtok_r()`** function behaves similarly to **`strtok()`** except for an additional parameter. The **`saveptr`** argument is a pointer to a **`char`** variable that is used internally by **`strtok_r()`** in order to maintain context between successive calls that parse the same string. On the first call to **`strtok_r()`**, **`str`** should point to the string to be parsed, and the value of **`saveptr`** is ignored. In subsequent calls, **`str`** should be `NULL`, and **`saveptr`** should be unchanged since the previous call. A sample demo code is given for your reference as;

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char str[] = "lesson-plan-usp-DOS";
    char *token;
    char *last;
    token = strtok_r(str, "-", &last);
    while (token!=NULL) {
        printf("Token: %s\n", token);
        token = strtok_r(NULL, "-", &last);
    }
    return (0);
}
```

Write your 'C' code using **`strtok_r()`** to determine the average number of words per line. Refer **Program 2.4** of your text book.

9. Write the C code for question numbers 3, 4, 5,and 6 using **`strtok_r()`** library function. The function prototype for **`strtok()`**, and **`strtok_r()`** given as follows;

```c
#include <string.h>

char *strtok(char *str, const char *delim);

char *strtok_r(char *str, const char *delim, char **saveptr);
```

```
The strtok() and strtok_r() functions return a pointer to the
   next token, or NULL if there are no more tokens.
```

[redacted]