

LABORATORY ASSIGNMENTS-003

UNIX Systems Programming (CSE 3041)

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

Practice Assignment/ Minor Assignment::

The syntax of **grep** is **\$ grep option pattern filename(s)**, searches the named files or the standard input and prints each line contains an instance of the **pattern**. Students are required to try various options such as -n, -v, -i, and -y etc. of grep from the **man** page.

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) Run the code **\$./a.out # shell comments**, and display the total number of arguments. The metacharacter # is used for shell comments
- (k) Run the code **\$./a.out 12 \$HOME 34**, and display the total number of arguments.
- (l) Putting any Shell command in between backquotes ('... ') executes the command.

Syntax: To put any Shell **command** in between backquotes **vname= `commandName`**.

Example::

- (i) **\$ vname= `pwd`**, then **\$ echo \$vname** will display the current working directory.

(ii) check the output

```
DATE=`date`  
echo "Current Date: $DATE"
```

Now, Run the code `$./a.out 12 `pwd` 34`.

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;

```
#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;

```
#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.

10. The **system()** function passes the `command` parameter to a command processor for execution. It behaves as if a child process were created with `fork` and the child process invoked `sh` with `exec1`.

SYNOPSIS

```
#include <stdlib.h>

int system(const char *command);

/*
(1) A zero termination status generally indicates successful
    completion.
(2) The value returned is -1 on error. If system could not
    fork a child or get the termination status.
(3) If command is NULL, the system function always returns a
    nonzero value to mean that a command language
    interpreter is available.
(4) If command is not NULL, system returns the termination
    status of the command language interpreter after the
    execution of command.
*/
```

Design choice based C code to run different shell commands, when the command will be entered from the *stdin* with 'yes' choice. If 'no' choice is given the program will terminate.

11. Write a C program to output the contents of the current environment list using the external variable `environ` that points to the process environment list when the process begin executing.
12. Write a C program to get the value of the environment variables; `PWD`, `HOME`, `LOGNAME`, `USER`, `PATH`, and `COLORTERM` etc. using **getenv()** function.

SYNOPSIS

```
#include <stdlib.h>

char *getenv(const char *name);

/*
(1) The getenv function returns NULL if the variable does not
    have a value.
(2) If the variable has a value, getenv returns a pointer to
    the string containing that value.
*/
```

NOTE:: Be careful about calling **getenv** more than once without copying the first return string into a buffer. Some implementations of `getenv` use a static buffer for the return strings and overwrite the buffer on each call.