BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJ.)
CS F111 Computer Programming
LAB SESSION #12
(Strings, Command Line Arguments, Introduction to Dynamic Memory)

Create a directory "**lab12**" inside "**myprogs**" directory for all your programs in this week's lab.

Strings

1. Implement a function convertLower() to convert a given string into one that contains entirely lower case letters. The function performs the conversion on the string received as its argument. First, implement your function using ASCII codes only. Next, use the library function tolower() to do the job. Remember to include the header <ctype.h>. You can learn more about this function by reading the manual pages.

   [Note: There are a suite of useful library functions like **tolower()** and **toupper()** available in the <ctype.h> library. Get to know them by typing **man isalpha** at the shell prompt. You should be familiar with the first 13 functions – starting from **isalnum()** and ending with **isblank()**, whose descriptions are also provided in the manual page, so you can use them in your programs when needed.]

2. Write a program to accept an English sentence and an English word from the user (using scanf() only), and then to check whether the word exists in the sentence or not. If it occurs, the program should print out how many times the word occurs in the sentence and then remove all the occurrences of the word from the sentence. Try this with and without using any string processing functions.

   Following is a sample for the same:

hi hello how are you hello hi hi	No. of occurrences of hi is 2 output string is: hello how are you hello

Command-line arguments (source: tutorialspoint.com)

It is possible to pass some values from the command line to your C programs when they are executed. These values are called **command-line arguments** and many times they are important for your program especially when you want to control your program from outside instead of hard-coding those values inside the code.

The command-line arguments are handled using the main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array that points to each argument passed to the program. Essentially argv is an array of character arrays (or strings). Following is a simple example that checks if there is any argument supplied from the command line and take action accordingly –

```
#include <stdio.h>

int main( int argc, char *argv[] )  {
   if( argc == 2 ) {
      printf("The argument supplied is %s\n", argv[1]);
   }
   else if( argc > 2 ) {
      printf("Too many arguments supplied.\n");
```

```
   }
   else {
      printf("One argument expected.\n");
   }
}
```

When the above code is compiled and executed with single argument, it produces the following result.

```
$./a.out testing
The argument supplied is testing
```

When the above code is compiled and executed with a two arguments, it produces the following result.

```
$./a.out testing1 testing2
Too many arguments supplied.
```

When the above code is compiled and executed without passing any argument, it produces the following result.

```
$./a.out
One argument expected
```

It should be noted that **argv[0]** holds the name of the program itself and **argv[1]** is a pointer to the first command-line argument supplied, and **argv[n]** is the **n**th argument. If no arguments are supplied, **argc** will be one, and if you pass one argument then **argc** is set at 2.

You pass all the command line arguments separated by a space, but if argument itself has a space then you can pass such arguments by putting them inside double quotes "" or single quotes ''. Let us re-write above example once again where we will print program name and we also pass a command line argument by putting inside double quotes −

```
#include <stdio.h>

int main( int argc, char *argv[] )  {

   printf("Program name %s\n", argv[0]);

   if( argc == 2 ) {
      printf("The argument supplied is %s\n", argv[1]);
   }
   else if( argc > 2 ) {
      printf("Too many arguments supplied.\n");
   }
   else {
      printf("One argument expected.\n");
   }
}
```

When the above code is compiled and executed with a single argument separated by space but inside double quotes, it produces the following result.

```
$./a.out "testing1 testing2"

Program name ./a.out
The argument supplied is testing1 testing2
```

Exercise: Execute the following program by giving three integers as command-line arguments. **[Note:** atoi() is a C library function that converts a string argument to an integer value. **End of Note]**

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    int a, b, c;

    a = atoi(argv[1]);
    b = atoi(argv[2]);
    c = atoi(argv[3]);
    printf("\n %d",a+b+c);
    return 0;
}
```

Introductory Programs: Dynamic memory Allocation

1. Create a function fun1() that takes an array of integers and its size as input and swaps the first and the last element. fun1() should be called from the main() function. The array that will be passed into the function fun1() must be created in the main() function. fun1() **should not return anything, but main() should see the elements swapped.**

   So, create an integer array of size 10 in the main() function, print it, then pass it to fun1(), and print the array again after fun1() returns. Use (a) static array; (b) dynamically allocated array.

2. Write a program that takes two arrays of the same size as input. The size of the arrays is not known till runtime and has to be taken from the user. The program then takes the elements of the arrays from the user as input too. Finally, the program prints the dot product of the two arrays on the console. [Taking dot product of the arrays is same as multiplying the arrays element wise and then taking the sum of all the products]

   Note that you need to declare all the arrays dynamically and allocate them on the heap for the implementation of the above question, no static arrays are allowed.

   Sample Output

   ```
   Enter the size: 3
   Enter the elements of the first array
   2 5 10
   Enter the elements of the second array
   4 10 8
   The dot product of the arrays is 138.
   ```
   [Hint: 2*4 + 5*10 + 10*8 = 138]

Additional Practice Exercises

Q1. Write a C program that receives two strings S1 and S2 as input. Then it compares S1 and S2, and then copy the two strings in a new string S3 in alphabetical order. If both the strings are equal then either S1 or S2 is copied into S3. Few sample I/P and O/P instance is given below:

S1 = Programming S2 = Computer
S3 = Computer Programming

S1 = Computer S2 = Computer
S3 = Computer

S1 = Computer S2 = Program
S3 = Computer Program

Note: To achieve the task, you are not allowed to use any string processing functions.

Q2. Write a program to create a 2D array (**say new2D**) that contains 2 rows. The first row should store an array of integers and the second row should store an array of float values. Take the number of elements in the first array and the second array as input from command line arguments. Then fill the 2D array with values taken from the user as input using scanf. Then Print the 2D array.

[Hint: Create a double-pointer of the type void (with the name **new2D)**. Then allocate memory to it for size 2. This creates an array of size 2, where each location can store address of another variable (or an array). Then use appropriate typecasting to allocate an integer array in the first location of new2D and a float array in the second location of new2D, using the respective sizes taken from the user as input.
End of Hint]