

CC-112L

Programming Fundamentals

Laboratory 10

Strings and Formatted I/O

Version: 1.0.0

Release Date: 23-09-2022

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

- Learning Objectives
- Required Resources
- General Instructions
- Background and Overview
 - C Strings
 - String Manipulation
 - Character Handling
 - Formatted I/O Functions
- Activities
 - Pre-Lab Activity
 - Fundamentals of Strings and Characters
 - Initializing char Arrays
 - Reading a string with scanf
 - NULL Terminated
 - Character Handling Library
 - String Conversion Functions
 - Task 01: String Conversion
 - Task 02: Change case
 - In-Lab Activity
 - Standard I/O Library Functions
 - String Manipulation Functions
 - Comparison Functions
 - Streams
 - Formatting Output with printf
 - Printing Integers
 - Printing Floating-point numbers
 - printf format flags
 - Printing Literals and Escape sequence
 - Task 01: String Comparison
 - Task 02: Largest & Smallest Word
 - Task 03: Temperature
 - Task 04: Find Error
 - Post-Lab Activity
 - Task 01: Vigenère Cipher
- Submissions
- Evaluations Metric
- References and Additional Material
- Lab Time and Activity Simulation Log

Learning Objectives:

- Character Handling Library
- String Conversion Functions
- Standard Input/Output Library Functions
- String Manipulation Functions
- Comparison Functions
- Streams
- Formatted Output with printf
- printf Format Flags
- Printing Literals and Escape Sequences

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Teacher Assistants	Usman Ali	bitf19m007@pucit.edu.pk
	Saad Rahman	bsef19m021@pucit.edu.pk

Background and Overview:

C Strings:

A string is a series of characters treated as a single unit. A string may include letters, digits and various special characters such as +, -, *, / and \$. String literals, or string constants, are written in double quotation marks.

String Manipulation:

String Manipulation is a class of problems where a user is asked to process a given string and use/change its data.

Character Handling:

Character handling in C is done by declaring arrays and moving characters in and out of them 'by hand'.

Formatted Input/Output Functions:

Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user. These types of I/O functions can help to display the output to the user in different formats using the format specifiers. These I/O supports all data types like int, float, char, and many more.

Activities:

Pre-Lab Activities:

Fundamentals of Strings and Characters:

Initializing char Arrays:

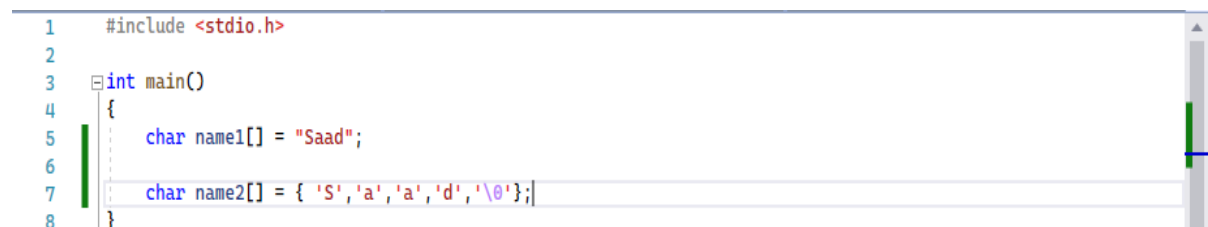
A string is a series of characters treated as a single unit. You can initialize character array as:

```
char name[] = "Saad";
```

The definition creates a 5-element array name containing the modifiable characters 'S', 'a', 'a', 'd' and '\0'.

The name array can also be defined as:

```
char name[] = {'S', 'a', 'a', 'd', '\0'};
```



```
1  #include <stdio.h>
2
3  int main()
4  {
5      char name1[] = "Saad";
6
7      char name2[] = {'S', 'a', 'a', 'd', '\0'};
8  }
```

Fig 01. (Initializing char Array)

Reading a string with scanf:

Function scanf can read a string and store it in a char array. Assume we have a char array word containing 10 elements. You can read a string into the array with:

```
scanf("%9s", array);
```

The field width 9 in the preceding statement ensures that scanf reads a maximum of 9 characters, saving the last array element for the string's terminating null character.



```
2  #include <stdio.h>
3
4  int main()
5  {
6      char array[10];
7      scanf("%9s", array);
8
9  }
```

Fig 02. (Reading a string with scanf)

NULL Terminated:

Every string must end with the null character ('\0'). Printing a "string" that does not contain a terminating null character is a logic error.

```

2 | #include <stdio.h>
3 |
4 | int main()
5 | {
6 |     char array1[] = {'L','o','g','i','c',' ','E','r','r','o','r','\0'};
7 |     printf("%s\n", array1);
8 |
9 |     char array2[] = { 'L','o','g','i','c',' ','E','r','r','o','r'};
10 |    printf("%s", array2);
11 | }

```

Fig 03. (NULL Terminator)

In the above program, null terminator is added at the end of array1. In array2, there is no null character which results in logical error.



Fig 04. (NULL Terminator)

Character-Handling Library:

The character-handling library (<ctype.h>) contains functions that test and manipulate character data. Each function receives an unsigned char (represented as an int) or EOF as an argument.

Following are some character-handling library functions:

Prototype	Description
isblank(c)	Returns a true value if c is a blank character that separates words in a line of text; otherwise, it returns 0 (false).
isdigit(c)	Returns a true value if c is a digit; otherwise, it returns 0 (false).
isalpha(c)	Returns a true value if c is a letter; otherwise, it returns 0 (false).
isalnum(c)	Returns a true value if c is a digit or a letter; otherwise, it returns 0 (false).
isxdigit(c)	Returns a true value if c is a hexadecimal digit character; otherwise, it returns 0 (false).
islower(c)	Returns a true value if c is a lowercase letter; otherwise, it returns 0 (false).
isupper(c)	Returns a true value if c is an uppercase letter; otherwise, it returns 0 (false).
tolower(c)	If c is an uppercase letter, tolower returns c as a lowercase letter; otherwise, it returns the argument unchanged.
toupper(c)	If c is a lowercase letter, toupper returns c as an uppercase letter; otherwise, it returns the argument unchanged.
isspace(c)	Returns a true value if c is a whitespace character—newline ('\n'), space (' '), form feed ('\f'), carriage return ('\r'), horizontal tab ('\t') or vertical tab ('\v')—otherwise, it returns 0 (false).
isctrl(c)	Returns a true value if c is a control character—horizontal tab ('\t'), vertical tab ('\v'), form feed ('\f'), alert ('\a'), backspace ('\b'), carriage return ('\r'), newline ('\n') and others—otherwise, it returns 0 (false).
ispunct(c)	Returns a true value if c is a printing character other than a space, a digit, or a letter—such as \$, #, (,), [,], {, }, :, ; or %—otherwise, it returns 0 (false).
isprint(c)	Returns a true value if c is a printing character (i.e., a character that's visible on the screen) including a space; otherwise, it returns 0 (false).
Isgraph(c)	Returns a true value if c is a printing character other than a space; otherwise, it returns 0 (false).

The following code demonstrates isalpha function.

```
#include <stdio.h>
#include <ctype.h>
int main()
{
    char alphabet = 'a', number='1';

    //Checking if a character is an alphabet or not
    if (isalpha(alphabet))
        printf("The character is an alphabet\n\n");
    else
        printf("The character is not an alphabet\n\n");

    if (isalpha(number))
        printf("The character is an alphabet\n\n");
    else
        printf("The character is not an alphabet\n\n");
}
```

Fig. 05 (isalpha function)

Output of the above code is as follows.



```
Microsoft Visual Studio Debug Console
The character is an alphabet
The character is not an alphabet
```

Fig. 06 (isalpha function)

The following code converts a lowercase letter to uppercase using toupper function.

```
#include <stdio.h>
#include <ctype.h>
int main()
{
    char alphabet = 'a';
    printf("Character before conversion: %c\n\n", alphabet);
    //Convert to uppercase
    alphabet = toupper(alphabet);
    printf("Character before conversion: %c\n\n", alphabet);
}
```

Fig. 07 (toupper function)

Output of the above code is as follows.



```
Microsoft Visual Studio Debug Console
Character before conversion: a
Character before conversion: A
```

Fig. 08 (toupper function)

String Conversion Functions:

String conversion functions convert strings of digits to integer and floating-point values. These functions are from the general utilities library <stdlib.h> The following table summarizes the string-conversion functions.

Prototype	Description
-----------	-------------

double strtod (const char *nPtr, char **endPtr)	Converts string nPtr to double.
long strtol (const char *nPtr, char **endPtr, int base)	Converts string nPtr to long.
unsigned long strtoul (const char *nPtr, char **endPtr, int base)	Converts string nPtr to unsigned long.

The use of strtod function is shown in the following example.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char* string = "85.7 percent";
    char* stringPtr = NULL;

    double d = strtod(string, &stringPtr);
    printf("Double Value: %f\n", d);
    printf("String: %s\n", stringPtr);
}
```

Fig. 09 (strtod function)

Output of the above program is as follows:



```
Microsoft Visual Studio Debug Console
Double Value: 85.700000
String: percent
```

Fig. 10 (strtod function)

Task 01: String Conversion**[Estimated 30 minutes / 20 marks]**

Write a C program which:

- Initialize a string.
- If all the characters in the string are digits, then convert it into long.
- Display the output on the Console.

String	Output
12345764	Long value: 12345764
This is a string	String conversion not possible
1234abc2	String conversion not possible

Task 02: Change case**[Estimated 30 minutes / 20 marks]**

Write a C program which:

- Ask the user to input elements of a character array.
- Check if all the characters are alphabets and space.
- Change the case of characters which are vowels alphabets (Uppercase to Lowercase vice versa).

String	Output
This is an Umbrella	This is An UmbrEllA
Alphabet	Alphabet
1234abc2	Case conversion not possible

In-Lab Activities:**Standard I/O Library Functions:**

Prototype	Description
int getchar(void)	Returns the next character from the standard input as an integer.
char *fgets(char *s, int n, FILE *stream)	Reads characters from the specified stream into the array s until a newline or end-of-file character is encountered.
int putchar(int c)	Prints the character stored in c and returns it as an integer.
int puts(const char *s)	Prints the string s followed by a newline character.
int sprintf(char *s, const char *format, ...)	Equivalent to printf, but the output is stored in the array s instead of printed on the screen.
int sscanf(char *s, const char *format, ...)	Equivalent to scanf, but the input is read from the array s rather than from the keyboard.

The fgets and putchar functions are used in the following code to read a line of text and display it on the Console.

```
#include <stdio.h>
int main()
{
    char array[20];
    printf("Enter some Text: ");
    fgets(array, 20, stdin);

    printf("Output: ");

    for (int i = 0; i < 20; i++)
    {
        if (array[i] == '\0')
            break;
        putchar(array[i]);
    }
}
```

Fig 11. (fgets and putchar function)

The output of the above code is:



```
Microsoft Visual Studio Debug Console
Enter some Text: BSEF19M021
Output: BSEF19M021
```

Fig 12. (fgets and putchar function)

Following example demonstrates the use of sprintf function. The program inputs an integer and double value and to be formatted and printed to character array.

```

#include <stdio.h>
#define SIZE 80

int main(void) {
    int x = 0;
    double y = 0.0;

    puts("Enter an integer and a double:");
    scanf("%d%lf", &x, &y);
    char s[SIZE] = { '\0' };
    sprintf(s, "integer:%d\ndouble:%7.2f", x, y);
    printf("The formatted output stored in array s is:\n%s\n", s);
}

```

Fig 13. (sprintf function)

The output of the above code is:

```

Microsoft Visual Studio Debug Console
Enter an integer and a double:
33
22.98
The formatted output stored in array s is:
integer: 33
double: 22.98

```

Fig 14. (sprintf function)

String-Manipulation Functions:

The string-manipulation functions of string handling library **<string.h>** are summarized in the following table.

Prototype	Description
char *strcpy(char *s1, const char *s2)	Copies string s2 into array s1 and returns s1.
char *strncpy(char *s1, const char *s2, size_t n)	Copies at most n characters of string s2 into array s1 and returns s1.
char *strcat(char *s1, const char *s2)	Appends string s2 to array s1 and returns s1. String s2's first character overwrites s1's terminating null character.
char *strncat(char *s1, const char *s2, size_t n)	Appends at most n characters of string s2 to arrays s1 and returns s1. String s2's first character overwrites s1's terminating null character.

Function `strcat` appends its second argument string to the string in its char array first argument, replacing the first argument's null ('`\0`') character. You must ensure that the array used to store the first string is large enough to store the first string, the second string and the terminating null character copied from the second string. Function `strncat` appends a specified number of characters from the second string to the first string and adds a terminating '`\0`'. The following code demonstrates function `strcat` and function `strncat`.

```

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

int main(void) {
    char s1[20] = "Roll No: "; // initialize char array s1
    char s2[30] = "BSEF19M021"; // initialize char array s2
    char s3[40] = ""; // initialize char array s3 to empty

    printf("s1 = %s\ns2 = %s\n", s1, s2);

    // concatenate s2 to s1
    printf("strcat(s1, s2) = %s\n", strcat(s1, s2));

    // concatenate first 5 characters of s1 to s3
    printf("strncat(s3, s1, 5) = %s\n", strncat(s3, s1, 5));
    // concatenate s1 to s3
    printf("strcat(s3, s1) = %s\n", strcat(s3, s1));
}

```

Fig 13. (String-Manipulation function)

The output of the above program is:



```

Microsoft Visual Studio Debug Console
s1 = Roll No:
s2 = BSEF19M021
strcat(s1, s2) = Roll No:BSEF19M021
strncat(s3, s1, 5) = Roll
strcat(s3, s1) = Roll Roll No:BSEF19M021

```

Fig 14. (String-Manipulation function)

Comparison Functions:

The string handling library <string.h> comparison functions are demonstrated below.

Prototype	Description
int strcmp(const char *s1, const char *s2)	Compares the string s1 with the string s2. The function returns 0, less than 0 or greater than 0 if s1 is equal to, less than or greater than s2, respectively.
int strncmp(const char *s1, const char *s2, size_t n)	Compares up to n characters of the string s1 with the string s2. The function returns 0, less than 0 or greater than 0 if s1 is equal to, less than or greater than s2, respectively

Function strcmp performs character by character comparison of its two string arguments. The function returns:

- 0 if strings are equal
- a negative value if the first string is less than the second string
- a positive value if the first string is greater than the second string

Function strncmp is equivalent to strcmp but compares up to a specified number of characters.

```

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

int main(void) {
    char* s1 = "BSEF19M021";
    char* s2 = "BSEF19M021";
    char* s3 = "BSEF20M021"; // initialize char pointer

    printf("s1 = %s\ns2 = %s\ns3 = %s\n\n%s%2d\n%s%2d\n%s%2d\n\n", s1, s2, s3,
           "strcmp(s1, s2) = ", strcmp(s1, s2),
           "strcmp(s1, s3) = ", strcmp(s1, s3),
           "strcmp(s3, s1) = ", strcmp(s3, s1));
    printf("%s%2d\n%s%2d\n%s%2d\n",
           "strncmp(s1, s2, 4) = ", strncmp(s1, s2, 4),
           "strncmp(s1, s3, 4) = ", strncmp(s1, s3, 4),
           "strncmp(s3, s1, 4) = ", strncmp(s3, s1, 4));
}

```

Fig 15. (Comparison Functions)

The output of the above program is:



```

Microsoft Visual Studio Debug Console
s1 = BSEF19M021
s2 = BSEF19M021
s3 = BSEF20M021

strcmp(s1, s2) = 0
strcmp(s1, s3) = -1
strcmp(s3, s1) = 1

strncmp(s1, s2, 4) = 0
strncmp(s1, s3, 4) = 0
strncmp(s3, s1, 4) = 0

```

Fig 16. (Comparison Functions)

Streams:

Input and output are performed with sequences of bytes called streams:

- In input operations, the bytes flow into main memory from a device, such as a keyboard, a solid-state drive, a network connection, and so on.
- In output operations, bytes flow from main memory to a device, such as a computer's screen, a printer, a solid-state drive, a network connection, and so on.

Formatting Output with printf:

Throughout the course, you've seen various printf output formatting features. Every printf call contains a format control string that describes the output format. The format control string consists of conversion specifiers, flags, field widths, precisions, and literal characters. Together with the percent sign (%), these form conversion specifications. Function printf can perform the following formatting capabilities:

- Rounding floating-point values to an indicated number of decimal places
- Aligning columns of numbers at their decimal points
- Right-aligning and left-aligning outputs
- Inserting literal characters at precise locations in a line of output
- Representing floating-point numbers in exponential format
- Representing unsigned integers in octal and hexadecimal format
- Displaying data with fixed-size field widths and precisions

Printing Integers:

An integer is a whole number, such as 776, 0 or -52. Integer values are displayed in one of several formats described by the following integer conversion specifiers.

Conversion Specifier	Description
d	Display as a signed decimal integer.
i	Display as a signed decimal integer.
O	Display as an unsigned octal integer.
u	Display as an unsigned decimal integer.
X or x	Display as an unsigned hexadecimal integer.
h, l or ll	These length modifiers are placed before any integer conversion specifier to indicate that the value to display is a short, long or long long integer.

Following code prints an integer using each integer conversion specifier.

```
#include <stdio.h>

int main(void) {
    printf("%d\n", 455);
    printf("%i\n", 455); // i same as d in printf
    printf("%d\n", +455); // plus sign does not print
    printf("%d\n", -455); // minus sign prints
    printf("%hd\n", 32000); // print as type short
    printf("%ld\n", 2000000000L); // print as type long
    printf("%o\n", 455); // octal
    printf("%u\n", 455);
    printf("%u\n", -455);
    printf("%x\n", 455); // hexadecimal with lowercase letters
    printf("%X\n", 455); // hexadecimal with uppercase letters
}
```

Fig 17. (Printing integers)

The output of the above code is:



```
Microsoft Visual Studio Debug Console
455
455
455
-455
32000
2000000000
707
455
4294966841
1c7
1C7
```

Fig 18. (Printing integers)

Printing Floating-point numbers:

Floating-point values contain a decimal point, as in 33.5, -657.93. Floating-point values are displayed using the conversion specifiers summarized below.

Conversion Specifier	Description
----------------------	-------------

E or e	Display a floating-point value in exponential notation.
F or f	Display a floating-point value in fixed-point notation.
G or g	Display a floating-point value in either the fixed-point form or the exponential form, based on the value's magnitude.
L	Place this length modifier before any floating-point conversion specifier to indicate that a long double floating-point value should be displayed.

Following code prints a decimal number using each floating-point conversion specifier.

```
#include <stdio.h>

int main(void) {
    printf("%e\n", 1234567.89);
    printf("%e\n", +1234567.89); // plus does not print
    printf("%e\n", -1234567.89); // minus prints
    printf("%E\n", 1234567.89);
    printf("%f\n", 1234567.89); // six digits to right of decimal point
    printf("%g\n", 1234567.89); // prints with lowercase e
    printf("%G\n", 1234567.89); // prints with uppercase E
}
```

Fig 19. (Printing floating-point numbers)

The output of the above code is:



```
Microsoft Visual Studio Debug Console
1.234568e+06
1.234568e+06
-1.234568e+06
1.234568E+06
1234567.890000
1.23457e+06
1.23457e+06
1.23457E+06
```

Fig 20. (Printing floating-point numbers)

printf Format Flags:

Function printf also provides flags to supplement its output formatting capabilities. The following table summarizes the five flags you can use in format control strings.

Flag	Description
-	Left-align the output within the specified field.
+	Display a plus sign preceding positive values and a minus sign preceding negative values.
Space	Print a space before a positive value not printed with the + flag.
#	Prefix 0 to the output value when used with the octal conversion specifier o. Prefix 0x or 0X to the output value when used with the hexadecimal conversion specifiers x or X.
0	Pad a field with leading zeros.

Following code demonstrates the use of each flag.

```
#include <stdio.h>

int main(void) {

    printf("%10s%10d%10c%10f\n", "hello", 7, 'a', 1.23);
    printf("%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23); //Left Aligning (- flag)

    printf("%d\n%d\n", 786, -786);
    printf("%+d\n%+d\n", 786, -786); //Displaying Sign (+ flag)

    printf("% d\n% d\n", 999, -999); // Displaying space (space flag)

    printf("%+09d\n", 999);
    printf("%09d\n", 999); //Padding zeroes (0 flag)
}
```

Fig 21. (printf Flags)

The output of the above code is:



```
hello      7      a 1.230000
hello     7      a 1.230000

786
-786
+786
-786

999
-999

+00000999
000000999
```

Fig 22. (printf Flags)

Printing Literals and Escape Sequence:

An escape sequence is represented by a backslash (\), followed by a particular escape character. The following table lists the escape sequences and the actions they cause.

Escape Sequence	Description
\' (single quote)	Output the single quote (') character.
\" (double quote)	Output the double quote (") character.
\? (question mark)	Output the question mark (?) character.
\\ (backslash)	Output the backslash (\) character.
\a (alert or bell)	Cause an audible (bell) or visual alert.
\b (backspace)	Move the cursor back one position on the current line.
\f (new page or form feed)	Move the cursor to the next logical page's start.
\n (newline)	Move the cursor to the beginning of the next line.
\r (carriage return)	Move the cursor to the beginning of the current line.
\t (horizontal tab)	Move the cursor to the next horizontal tab position.
\t (vertical tab)	Move the cursor to the next vertical tab position.

Task 01: String Comparison**[20 minutes / 20 mark]**

Write a C program that:

- Take two strings as an input from user
- Take input of the number of characters to be compared
- Compare the strings up to specified number
- Then display the result on Console

Input	Output
Saad Osman 4	First 4 characters of strings are equal.
Saad Saad 3	First 3 characters of strings are equal.

Task 02: Largest & Smallest Word**[30 minutes / 40 marks]**

Create a C program that

- Take a string as an input from the user
- Find the largest and smallest word in the string using character handling library
- Display the words on the Console

Input	Output
This is a string	Largest Word: string Smallest Word: a
This is a test	Largest Words: This, test Smallest Word: a

Task 03: Temperature**[30 minutes / 30 marks]**

Create a C Program that:

- Inputs Fahrenheit integer temperature (0-212) from user
- Converts Fahrenheit temperature to floating-point Celsius temperature with 3-digit precision
- Display the output in two right-aligned columns of 10 characters each
- Precede the Celsius temperatures by a sign for both positive and negative values

Formula:

$$\text{Celsius} = 5.0 / 9.0 * (\text{Fahrenheit} - 32)$$

Task 04: Find Error**[20 minutes / 20 marks]**

Find and correct the error in the following program:

```
#include <stdio.h>
int main(void) {

    printf("%s\n", 'Happy Birthday');
    printf("%c\n", 'Hello');
    printf("%c\n", "This is a string");
    printf("" % s", "Bon Voyage");
    char day[] = "Sunday";
    printf("%s\n", day[3]);
    puts('Enter your name: ');
    printf(% f, 123.456);
    printf("%s%s\n", 'O', 'K');
    char s[10];
    scanf("%c", s[7]);

}
```

Fig 23. (In-Lab Task)

Post-Lab Activities:**Task 01: Vigenère Cipher****[Estimated 90 minutes / 100 marks]**

Vigenère Cipher is a method of encrypting alphabetic text.

Create a C program that:

- Inputs a String from the user
- Inputs a keyword from a user
- Validates the length of keyword must be less than the string using comparison function
- Encrypt the string using Vigenère Cipher table
- Decrypt the encrypted string by using Vigenère Cipher
- Use string handling library
- Display the Encrypted and Decrypted string on Console
- Right Align the Decrypted string

Vigenère Cipher Table:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Fig 24. (Post-Lab Task)

Input	Output
MY NAME THIS IS A MESSAGE	Keyword is bigger than string
THIS IS A MESSAGE KEYWORD	Encrypted: DLGO WJ D WIQOOXH Decrypted: THIS IS A MESSAGE
TEXT KEY	Encrypted: DIVD Decrypted: TEXT

Submit two “.c” files named your “Roll No” on Google Classroom.

Submissions:

- For In-Lab Activity:
 - Save the files on your PC.
 - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
 - Submit the .c file on Google Classroom and name it to your roll no.

Evaluations Metric:

- All the lab tasks will be evaluated offline by TA's
- **Division of Pre-Lab marks:** **[40 marks]**
 - Task 01: String Conversion [20 marks]
 - Task 02: Change case [20 marks]
- **Division of In-Lab marks:** **[110 marks]**
 - Task 01: String Comparison [20 marks]
 - Task 02: Largest & Smallest word [40 marks]
 - Task 03: Temperature [30 marks]
 - Task 04: Find Error [20 marks]
- **Division of Post-Lab marks:** **[100 marks]**
 - Task 01: Vigenère Cipher [100 marks]

References and Additional Material:

- C Strings
<https://www.programiz.com/c-programming/c-strings>
- C Input/Output
<https://www.programiz.com/c-programming/c-input-output>
- Vigenère Cipher
https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher

Lab Time Activity Simulation Log:

- Slot – 01 – 00:00 – 00:15: Class Settlement
- Slot – 02 – 00:15 – 00:30: In-Lab Task
- Slot – 03 – 00:30 – 00:45: In-Lab Task
- Slot – 04 – 00:45 – 01:00: In-Lab Task
- Slot – 05 – 01:00 – 01:15: In-Lab Task
- Slot – 06 – 01:15 – 01:30: In-Lab Task
- Slot – 07 – 01:30 – 01:45: In-Lab Task
- Slot – 08 – 01:45 – 02:00: In-Lab Task
- Slot – 09 – 02:00 – 02:15: In-Lab Task
- Slot – 10 – 02:15 – 02:30: In-Lab Task
- Slot – 11 – 02:30 – 02:45: Evaluation of Lab Tasks
- Slot – 12 – 02:45 – 03:00: Discussion on Post-Lab Task