



MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

CSE 1061
PROBLEM SOLVING USING
COMPUTERS - LAB MANUAL

FIRST YEAR
COMMON TO ALL BRANCHES
(2018 CURRICULUM)

(March 2021 batch – MS Teams – Online Lab)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MANIPAL INSTITUTE OF TECHNOLOGY
MAHE, MANIPAL
KARNATAKA - 576104

CONTENTS

LAB NO.	TITLE	PAGE NO.	REMARKS
	COURSE OBJECTIVES AND OUTCOMES	i	
	EVALUATION PLAN	i	
	INSTRUCTIONS TO THE STUDENTS	ii-iii	
	SAMPLE LAB OBSERVATION NOTE PREPARATION	iv-v	
1	SIMPLE C PROGRAMS	1	
2	BRANCHING CONTROL STRUCTURES	6	
3	LOOPING CONTROL STRUCTURES-WHILE & DO LOOPS	10	
4	LOOPING CONTROL STRUCTURES- FOR LOOPS	13	
5	1D ARRAYS	15	
6	2D ARRAYS	18	
7	STRINGS	20	
8	MODULAR PROGRAMMING - FUNCTIONS	22	
*	MODULAR PROGRAMMING-RECURSIVE FUNCTIONS, STRUCTURES, POINTERS	25	SELF STUDY LAB
9	LAB EXAM (MS TEAMS SUBMISSION)		
	REFERENCES	30	
	C LANGUAGE QUICK REFERENCE	31	

Course Objectives

- To develop the programming skills using basics of C language
- To write algorithms and draw flowchart for various problems
- To write, compile and debug programs in C language

Course Outcomes

At the end of this course, students will be able to

- Represent and manipulate data to store it in a computer using C syntax.
- Develop, execute, and document C programs.

Evaluation plan

- Internal Assessment Marks: 80%
 - ✓ Students must work out the C programs in code blocks only.
 - ✓ Students shall submit the lab code along with snapshot of results obtained under every program (PDF) **as per the schedule**. Eg: At the end of lab 2, both lab 1 & lab 2 programs should be submitted in a single MS word file. While submission students should print their names in every program through a print statement.
 - ✓ Maximum of 5M will be awarded for each lab submission, ie. 10M for lab 1&2. Marks will be reduced if students miss out programs.
 - ✓ During lab 2, 4, 6 & 8 **MCQ/ Descriptive** related to earlier completed lab practice will be asked in MS forms with 1M each.
 - ✓ Total of 20M will be awarded for every 2 labs. ie, 10 M for write up and 10 M for MCQ/lab assessment question. The **C program code** should be submitted for lab assessment question. The name of file should be like **regno_lab1.c Eg: 20100345_lab1.c**
 - ✓ In this way there will be 4 such evaluation with total of 80M.
- End semester assessment of 2-hour duration: 20%
 - ✓ A complete C program using functions will be asked. The given program must be worked out in code blocks and compiled. The **C program code** should be submitted. The name of file should be like **regno.c Eg: 20100345.c**

Note:

1. Students shall do all the programming exercises in CODE BLOCKS only. The URL for downloading and procedure is **given below**.
2. An additional lab exercise on Structures / Pointers (self-study) is kept at the end for student's reference.

Procedure for CODE BLOCKS downloading and installing.

Code:Blocks is a free C/C++ and Fortran IDE built to meet the most demanding needs of its users. It is designed to be very extensible and fully configurable.

**URL: <https://www.codeblocks.org/downloads/binaries/>
<https://www.codeblocks.org/downloads/>**

In this URL you may be able to download for the following operating systems.

- **Windows XP / Vista / 7 / 8.x / 10**
- **Linux 32 and 64-bit**
- **Mac OS X**

Please look into the below YouTube link which details how to download and install code blocks for windows 10.

YouTube: <https://www.youtube.com/watch?v=GWJqsmiR2I>

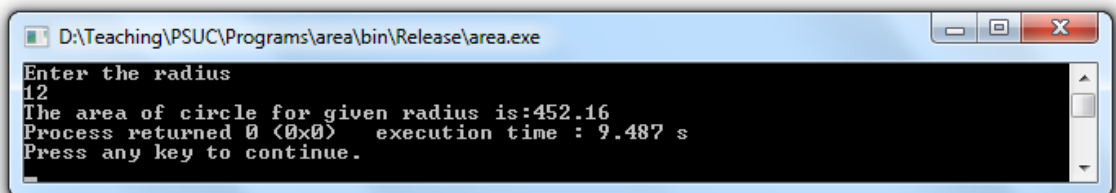
Sample Lab Observation Note Preparation

Title: SIMPLE C PROGRAMS

1. Program to find area of the circle. (Hint: $\text{Area}=3.14*r*r$)

```
//program to find area of circle
#include<stdio.h>
int main()
{
    int radius;
    float area;
    printf("Enter the radius\n");
    scanf("%d", &radius);
    area=3.14*radius*radius;
    printf("My name is abcd"); →Name should be printed in every prog.
    printf("The area of circle for given radius is: %f", area);
    return 0;
}
```

Sample input and output: Screen shot of result – must be included.



The screenshot shows a Windows command prompt window with the title bar "D:\Teaching\PSUC\Programs\area\bin\Release\area.exe". The window contains the following text:

```
Enter the radius
12
The area of circle for given radius is:452.16
Process returned 0 (0x0)   execution time : 9.487 s
Press any key to continue.
```

LAB NO.: 1

SIMPLE C PROGRAMS

Objectives

In this lab, student will be able to:

1. Write C programs.
2. Compile and execute C programs.
3. Debug and trace the programs.

Code: Blocks Integrated Development Environment

Code: Blocks has a C editor and compiler. It allows us to create and test our programs. Code: Blocks creates Workspace to keep track of the project that is being used. A project is a collection of one or more source files. Source files are the files that contain the source code for the problem.

Let's look at C program implementation in steps by writing, storing, compiling and executing a sample program

- Create a directory with section followed by roll number (to be unique); e.g. A21.
- As per the instructions given by the lab teacher, create *InchToCm.c* program.
 - Open a new notepad file and type the given program
- Save the file with name and extension as "*InchToCm.c*" into the respective directory created.

Sample Program (*InchToCm.c*):

```
// InchToCm.c
#include <stdio.h>
int main()
{
    float centimeters, inches;

    printf( "This program converts inches to centimeters"\n);
    printf( "Enter a number");
    scanf("%f", &inches);
```

```

        centimeters = inches * 2.54;
        printf("%f inches is equivalent to %f centimeters\n", inches, centimeters);
        return 0;
    } // end main

```

- Run the program as per the instructions given by the lab teacher.
 - Compile the saved program and run it either by using keyboard short cuts or through the menu.

PROGRAM STRUCTURE AND PARTS

Comments

The first line of the file is:

```
// InchToCm.c
```

This line is a comment. Let's add a comment above the name of the program that contains the student's name.

- Edit the file. Add the student's name on the top line so that the first two lines of the file now look like:

```
// student's name
```

```
// InchToCm.c
```

Comments tell people reading the program something about the program. The compiler ignores these lines.

Preprocessor Directives

After the initial comments, the student should be able to see the following lines:

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

This is called a preprocessor directive. It tells the compiler to do something. Preprocessor directives always start with a # sign. The preprocessor directive includes the information in the file *stdio.h*. as part of the program. Most of the programs will almost always have at least one include file. These header files are stored in a library that shall be learnt more in the subsequent labs.

The function main ()

The next non-blank line *int main ()* gives the name of a function. There must be exactly one function named *main* in each C program, and *main* is the function where program execution starts when the program begins running. The *int* before *main ()* indicates the function is returning an integer value and also to indicate empty argument list to the function. Essentially functions are units of C code that do a particular task. Large programs will have many functions just as large organizations have many functions. Small programs, like smaller organizations, have fewer functions. The parentheses following the words *int main* contains a list of arguments to the function. In the case of this function, there are no *arguments*. Arguments to functions tell the function what objects to use in performing its task. The curly braces ({} on the next line and on the last line { }) of the program determine the beginning and ending of the function.

Variable Declarations

The line after the opening curly brace, *float centimeters, inches;* is called a variable declaration. This line tells the compiler to reserve two places in memory with adequate size for a real number (the *float* keyword indicates the variable as a real number). The memory locations will have the names *inches* and *centimeters* associated with them. The programs often have many different variables of many different types.

EXECUTABLE STATEMENTS

Output and Input

The statements following the variable declaration up to the closing curly brace are executable statements. The executable statements are statements that will be executed when the program run. *printf ()* statement tells the compiler to generate instructions that will display information on the screen when the program run, and *scanf ()* statement reads information from the keyboard when the program run.

Format Specifiers	Description	Example
%d	%d is used to print the value of integer variable. We can also use %i to print integer value. %d and %i have same meaning	int v; scanf("%d",&v); printf("value is %d", v);
%f	%f is used to print the value of floating point variable in a decimal form.	float v; scanf("%f",&v); printf("value is %f",v);
%c	%c is used to print the value of character variable.	char ch; scanf("%c",&v); printf("value is %c",ch)
%s	%s is used to print the string	char name[20]; scanf("%s",v); printf("value is %s",name)

Assignment Statements

The statement *centimeters = inches * 2.54;* is an assignment statement. It calculates what is on the right hand side of the equation (in this case *inches * 2.54*) and stores it in the memory location that has the name specified on the left hand side of the equation (in this case, *centimeters*). So *centimeters = inches * 2.54* takes whatever was read into the memory location *inches*, multiplies it by 2.54, and stores the result in *centimeters*. The next statement outputs the result of the calculation.

Return Statement

The last statement of this program, *return 0;* returns the program control back to the operating system. The value 0 indicates that the program ended normally. The last line of every main function written should be *return 0;*

Syntax

Syntax is the way that a language must be phrased in order for it to be understandable. The general form of a C program is given below:

```
// program name
// other comments like what program does and student's name
# include <appropriate files>
int main()
{
    Variable declarations;
    Executable statements;
} // end main
```

Lab Exercises

1. Write a C program to add two integers a and b read through the keyboard. Display the result using third variable sum
2. Write a C program to find the sum, difference, product and quotient of 2 numbers.
3. Write a C program to print the ASCII value of a character
4. Write a C program to display the size of the data type int, char, float, double, long int and long double using size of () operator.
5. Input P, N and R to compute simple and compound interest. [Hint: $SI = PNR/100$, $CI = P(1+R/100)^N - P$]
6. Input radius to find the volume and surface area of a sphere. [Hint: volume = $(4\pi r^3)/3$, Area= $4\pi r^2$]
7. Convert the given temperature in Fahrenheit to Centigrade. [Hint: $C=5/9(F-32)$]
8. Write a C program to evaluate the following expression for the values a = 30, b=10, c=5, d=15
 - (i) $(a + b) * c / d$
 - (ii) $((a + b) * c) / d$
 - (iii) $a + (b * c) / d$
 - (iv) $(a + b) * (c / d)$

LAB NO.: 2

BRANCHING CONTROL STRUCTURES

Objectives:

In this lab, student will be able to do C programs using

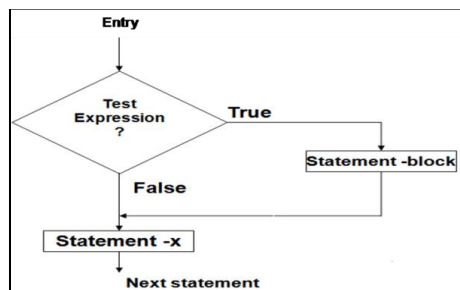
1. **simple *if*** statement
2. ***if-else*** statement
3. ***switch-case*** statement

Introduction:

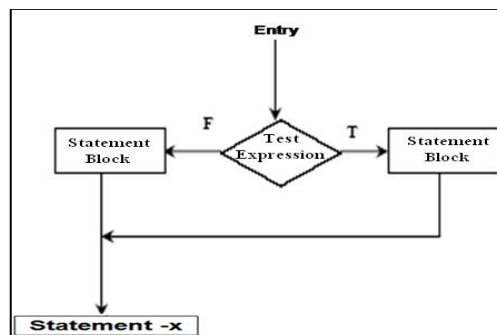
- A control structure refers to the way in which the programmer specifies the order of execution of the instructions

C decision making and branching statements flow control actions:

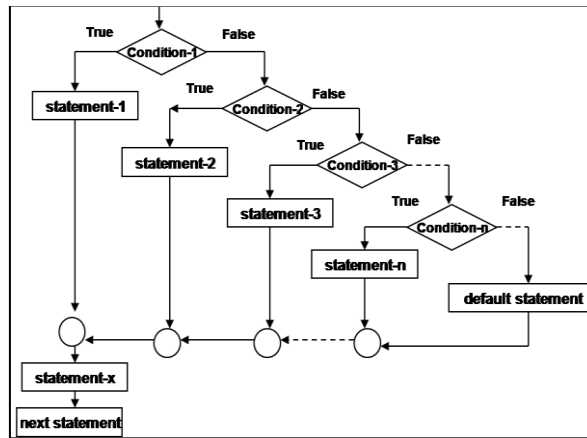
Simple if statement:



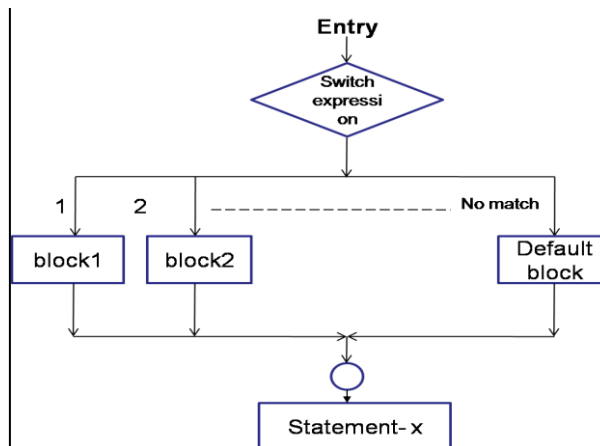
If - else statement:



Else - if ladder:



Switch statement:



Solved Exercise

C program to compute all the roots of a quadratic equation

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

```
#include<math.h>
```

```
int main() {
```

```
int a,b,c;
```

```
float root1, root2, re, im, disc;
```

```
scanf("%d,%d,%d",&a,&b,&c);
```

```

disc=b*b-4*a*c;
if (disc<0) // first if condition
{
printf("imaginary roots\n");
re= - b / (2*a);
im = pow(fabs(disc),0.5)/(2*a);
printf("%f +i %f",re,im);
printf("%f -i %f",re,im);
//printf("%f, re", "+ i", im);
//cout<<re<<"-i"<<im;
}
else if (disc==0){ //2nd else-if condition
printf("real & equal roots");
re=-b / (2*a);
printf("Roots are %f",re);
}
else{ /*disc > 0- otherwise part with else*/
printf("real & distinct roots");
printf("Roots are");
root1=(-b + sqrt(disc))/(2*a);
root2=(-b - sqrt(disc))/(2*a);
printf("%f and %f",root1,root2);
}
return 0;
}

```

Lab Exercises

With the help of various branching control constructs like *if*, *if-else* and *switch* case statements,

Write C programs to do the following:

- 1) Check whether the given number is odd or even
- 2) Find the largest among given 3 numbers
- 3) Swap two numbers without using third variable.
- 4) Compute all the roots of a quadratic equation using *switch case* statement.

[Hint: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$]

- 5) Write a program that will read the value of x and evaluate the following function

$$Y = \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{for } x = 0 \\ -1 & \text{for } x < 0 \end{cases}$$

Use else if statements & Print the result ('Y' value).

6. Find the smallest among three numbers using conditional operator.

LAB NO.: 3

LOOPING CONTROL STRUCTURES-WHILE & DO LOOPS

Objectives:

In this lab, student will be able to:

1. Write and execute C programs using 'while' statement
2. Write and execute C programs using 'do-while' statement
3. To learn to use break and continue statements in while and do while loop statements.

Introduction:

- Iterative (repetitive) control structures are used to repeat certain statements for a specified number of times.
- The statements are executed as long as the condition is true
- These types of control structures are also called as loop control structures
- Three kinds of loop control structures are:
 - while
 - do-while
 - for

C looping control structures:

While loop:

```
while (test condition)
{
    body of the loop
}
```

Do-while loop:

```
do
{
    body of the loop
}
while (test condition);
```

Solved Exercise

[Understand the working of looping with this illustrative example for finding sum of natural numbers up to 100 using *while* and *do-while* statements]

Using *do-while*

```
#include<stdio.h>
int main()
{
    int n;
    int sum;
    sum=0; //initialize sum
    n=1;
    do
    {
        sum = sum + counter;
        counter = counter +1;
    } while (counter < 100);
    printf(“%d”,sum);
    return 0;}
```

Using *while*

```
#include<stdio.h>
int main( )
{
    int n;
    int sum;
    sum=0; //initialize sum
    n=1;
    while (n<100)
    {
        sum = sum + n;
        n = n +1;
    }
```



```
printf("%d",sum);  
return 0; }
```

Lab Exercises

Write C programs to do the following with the help of iterative (looping) control structures such as *while* and *do-while* statements,

1. Reverse a given number and check if it is a palindrome or not. (use while loop).
[Ex: 1234, reverse= $4 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 = 4321$]
2. Generate prime numbers between 2 given limits.(use while loop)
3. Check if the sum of the cubes of all digits of an inputted number equals the number itself (Armstrong Number). (use while loop)
4. Write a program using do-while loop to read the numbers until -1 is encountered. Also count the number of prime numbers and composite numbers entered by user. [Hint: 1 is neither prime nor composite]
5. Check whether the given number is strong or not.

[Hint: Positive number whose sum of the factorial of its digits is equal to the number itself] Ex: $145 = 1! + 4! + 5! = 1 + 24 + 120 = 145$ is a strong number.

6. Write a program to demonstrate use of break and continue statements in while and do-while loops.

LAB NO.: 4

LOOPING CONTROL STRUCTURES- FOR LOOPS

Objectives:

In this lab, student will be able to:

- Write and execute C programs using 'for' statement
- To learn to use break and continue statements in for loop statements.

Introduction:

- For loop statements are used to repeat certain statements for a specified number of times.
- The statements are executed as long as the execution condition is true
- These types of control structures are also called as loop control structures

For loop:

```
for (initialization; test condition; increment/decrement)
{
    body of the loop
}
```

Lab Exercises

With the help of *for loop* statements,

1. Generate the multiplication table for '*n*' numbers up to '*k*' terms (using nested for loops).

```
[ Hint: 1  2  3  4  5  ....  k
        2  4  6  8 10  ....2*k
        .....
        n..... n*k ]
```

2. Generate Floyd's triangle using natural numbers for a given limit N. (using for loops)

[Hint: Floyd's triangle is a right angled-triangle using the natural numbers]

Ex: Input: N = 4

Output:

1
2 3
4 5 6
7 8 9 10

3. Evaluate the sine series, $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$ to n terms.

4. Check whether a given number is perfect or not.

[Hint: Sum of all positive divisors of a given number excluding the given number is

equal to the number] Ex: $28 = 1 + 2 + 4 + 7 + 14 = 28$ is a perfect number

5. Find out the generic root of any number.

[Hint: Generic root is the sum of digits of a number until a single digit is obtained.]

Ex: Generic root of 456 is $4 + 5 + 6 = 15 = 1 + 5 = 6$

6. Write a program to demonstrate use of break and continue statements in *for* loop

LAB NO.: 5

1D ARRAYS

Objectives:

In this lab, student will be able to:

- Write and execute programs on 1Dimensional arrays

Introduction to 1D Arrays

1 Dimensional Array

Definition:

- An array is a group of related data items that share a common name.
- The array elements are placed in a contiguous memory location.
- A particular value in an array is indicated by writing an integer number called index number or subscript in square brackets after the array name. The least value that an index can take in array is 0.

Array Declaration:

data-type name [size];

- ✓ where data-type is a valid data type (like int, float,char...)
- ✓ name is a valid identifier
- ✓ size specifies how many elements the array has to contain
- ✓ size field is always enclosed in square brackets [] and takes static values.

Total size of 1D array:

The Total memory that can be allocated to 1D array is computed as:

Total size =size *(sizeof(data_type));

where, *size* is number of elements in 1-D array

data_type is basic data type.

Sizeof() is an unary operator which returns the size of expression or data type in bytes.

For example, to represent a set of 5 numbers by an array variable **Arr**, the declaration the variable **Arr** is

int Arr[5];

Solved Exercise

Sample Program to read n elements into a 1D array and print it:

```
#include<stdio.h>
int main()
{
    int a[10], i, n;
    printf("enter no of elements");
    scanf("%d",&n);
    printf("enter n values\n");
    for(i=0;i<n;i++) // input 1D array
        scanf("%d",&a[i]);
    printf("\nNumbers entered are:\n");
    for(i=0;i<n;i++) // output 1D array
        printf("%d",a[i]);
    return 0;
}
```

Output:

Enter no. of elements

3

Enter n values

9

11

13

Numbers entered are:

9

11

13

Lab Exercises

With the knowledge of 1D arrays, Write C programs to do the following:

1. Find the largest and smallest element in a 1D array.
2. Print all the prime numbers in a given 1D array.
3. Arrange the given elements in a 1D array in ascending and descending order using bubble sort method. [Hint: use switch case (as case 'a' and case 'd') to specify the order].
4. Insert an element into a 1D array by getting an element and the position from the user.
5. Search the position of the number that is entered by the user and delete that number from the array and display the resultant array elements.

LAB NO.: 6

2D ARRAYS

Objectives:

In this lab, student will be able to:

- Write and execute programs on 2D dimensional arrays

Introduction to 2 Dimensional Arrays

- It is an ordered table of homogeneous elements.
- It can be imagined as a two dimensional table made of elements, all of them of a same uniform data type.
- It is generally referred to as matrix, of some rows and some columns. It is also called as a two-subscripted variable.

For example

```
int marks[5][3];  
float matrix[3][3];  
char page[25][80];
```

- The first example tells that marks is a 2-D array of 5 rows and 3 columns.
- The second example tells that matrix is a 2-D array of 3 rows and 3 columns.
- Similarly, the third example tells that page is a 2-D array of 25 rows and 80 columns.

Solved Exercise:

```
#include<stdio.h>  
int main()  
{  
int i,j,m,n,a[100][100];  
printf("enter dimension of matrix");  
scanf("%d %d",&m,&n);  
printf("enter the elements");  
for(i=0;i<m;i++) // input 2D array using 2 for loops  
{
```

```

for(j=0;j<n;j++)
    scanf("%d",&a[i][j]);
}
for(i=0;i<m;i++) // output 2D array with 2 for loops
{
for(j=0;j<n;j++)
printf("%d\t",a[i][j]);
printf("\n");
}
return 0;
}

```

Lab Exercises

With the knowledge of 2D arrays,

Write C programs to do the following:

1. Find whether a given matrix is symmetric or not. [Hint: $A = A^T$]
2. Find the trace and norm of a given square matrix.
[Hint: Trace= sum of principal diagonal elements
Norm= SQRT (sum of squares of the individual elements of an array)]
3. Perform matrix multiplication.
4. To interchange the primary and secondary diagonal elements in the given Matrix.
5. Interchange any two Rows & Columns in the given Matrix.
6. Search for an element in a given matrix and count the number of its occurrences.

LAB NO.: 7

STRINGS

Objectives:

In this lab, student will be able to:

1. Declare, initialize, read and write a string
2. Write C programs with and without string handling functions to manipulate the given string

Introduction

- A string is an array of characters.
- Any group of characters (except double quote sign) defined between double quotations is a constant string.
- Character strings are often used to build meaningful and readable programs.

The common operations performed on strings are

- Reading and writing strings
- Combining strings together
- Copying one string to another
- Comparing strings to another
- Extracting a portion of a string etc.

Declaration

Syntax: ***char string_name[size];***

- The size determines the number of characters in the string_name.

Solved Exercise

Program to read and display a string

```
#include<stdio.h>
int main()
{
    const int MAX = 80; //max characters in string
    char str[MAX];      //string variable str
```

```

printf("Enter a string: ");
scanf("%s",str);
//put string in str
Printf("You entered: %s\n" str); //display string from str
return 0;}

```

Lab Exercises

With the brief introduction and knowledge on strings,

Write C programs without using STRING-HANDLING functions for the following:

1. Count the number of words in a sentence.
2. Input a string and toggle the case of every character in the input string.
Ex: INPUT: aBcDe
 OUTPUT: AbCdE
3. Check whether the given string is a palindrome or not.
4. Arrange 'n' names in alphabetical order (hint: use string handling function-*strcpy*)
5. Delete a word from the given sentence.
Ex: INPUT: I AM STUDYING IN MIT
 TO BE DELETED: STUDYING
OUTPUT: I AM IN MIT

LAB NO.: 8

MODULAR PROGRAMMING -FUNCTIONS

Objectives:

In this lab, student will be able to:

1. Understand modularization and its importance
2. Define and invoke a function
3. Analyze the flow of control in a program involving function call
4. Write programs using functions

Introduction

- A **function** is a set of instructions to carry out a particular task.
- Using functions programs can be structured in a **more modular** way.

Function definition and call

```
// FUNCTION DEFINITION
Return type  Function name  Parameter List
void DisplayMessage(void)
{
    cout << "Hello from function DisplayMessage\n";
}

int main()
{
    cout << "Hello from main";
    DisplayMessage(); // FUNCTION CALL
    cout << "Back in function main again.\n";
    return 0;
}
```

Solved Exercise

1. Program for explaining concept of multiple functions

```
#include<stdio.h>
void First (void){
    printf("I am now inside function First\n");
}
void Second (void){
    printf("I am now inside function Second\n");
    First();
    printf("Back to Second\n");
}
int main (){
    printf("I am starting in function main\n");
    First ();
    printf("Back to main function \n");
    Second ();
    printf("Back to main function \n");
    return 0;
}
```

Lab Exercises

With the knowledge of modularization, function definition, function call etc., write C programs which implement simple functions.

Write C programs as specified below:

1. Write a function **Fact** to find the factorial of a given number. Using this function, compute **NCR** in the main function.
2. Write a function **Largest** to find the maximum of a given list of numbers. Also write a main program to read N numbers and find the largest among them using this function.
3. Write a function **IsPalin** to check whether the given string is a palindrome or not. Write a main function to test this function.

4. Write a function **CornerSum** which takes as a parameter, no. of rows and no. of columns of a matrix and returns the sum of the elements in the four corners of the matrix. Write a main function to test the function.

SELF STUDY LAB

MODULAR PROGRAMMING-RECURSIVE FUNCTIONS, STRUCTURES, POINTERS

Objectives:

In this lab, student will be able to:

1. To Learn the concept of recursion and to write recursive functions
2. Declare and initialize pointer variable
3. Write basic operations and programs using structures
4. Access a variable through its pointer

Introduction:

- A recursive function is a function that invokes/calls itself directly or indirectly.
- A Pointer is a memory location or a variable which stores the address of another variable in memory
- A structure in the C programming language (and many derivatives) is a composite data type (or record) declaration that defines a physically grouped list of variables to be placed under one name in a block of memory.
- A file is a place on disc where group of related data is stored.

Steps to Design a Recursive Algorithm

- **Base case:**
 - for a small value of n , it can be solved directly
- **Recursive case(s)**
 - Smaller versions of the same problem
- Algorithmic steps:
 - Identify the base case and provide a solution to it
 - Reduce the problem to smaller versions of itself
 - Move towards the base case using smaller versions

Solved Exercise

Program to explain the concept of recursive functions

```
#include<stdio.h>
long factorial (long a) {
    if (a ==0) //base case
        return (1);
    return (a * factorial (a-1));
}
int main () {
    long number;
    printf("Please type a number: ");
    scanf("%d",&number);
    printf("%d factorial is %ld",number, factorial (number));
    return 0;
}
```

Declaring and initializing pointers:

Syntax:

```
data_type * pt_name;
```

This tells the compiler 3 things about the pt_name:

- The asterisk (*) tells the variable pt_name is a pointer variable.
- pt_name needs a memory location.
- pt_name points to a variable of type data_type

Solved Exercise:

```
#include<stdio.h>
int main()
{
    int var1 = 11; //two integer variables
    int var2 = 22;
```

```

    int *ptr;           //pointer to integer
    ptr = &var1;        //pointer points to var1
    printf("%d",*ptr);   //print contents of pointer (11)
    ptr = &var2;        //pointer points to var2
    printf("%d",*ptr);   //print contents of pointer (22)
    return 0;
}

```

Declaration and initialization of structures:

Déclaration :

```

struct student
{
    int rollno, age;
    char name[20];
} s1, s2, s3;

```

Initialisation :

```

int main( ){
    struct
    {
        int rollno;
        int age;
    }stud={20, 21 };
    ...
    ...
    return 0;
}

```

Solved Exercise:

```

#include<stdio.h>
struct Book{           //Structure Definition
    char title[20];
    char author[15];
    int pages;
    float price;
}

```



```

};
int main( ){
struct Book b[10];
int i,j;
printf("Input values");
for (i=0;i<10;i++){
scanf("%s %s %d %f",b[i].title,b[i].author,&b[i].pages,&b[i].price);
}
for (j=0;j<10;j++){
printf("title %s\n author %s\n pages %d\n
price%f\n",b[j].title,b[j].author,b[j].pages,b[j].price);
}
return 0;
}

```

Lab Exercises

With the knowledge of recursive functions, Structures and pointers
Write C programs as specified below:

1. Write a recursive function, **GCD** to find the GCD of two numbers. Write a main program which reads 2 numbers and finds the GCD of the numbers using the specified function. Ex: GCD of 9, 24 is 3.
2. Write a recursive function **FIB** to generate n^{th} Fibonacci term. Write a main program to print first N Fibonacci terms using function FIB.

[Hint: Fibonacci series is 0, 1, 1, 2, 3, 5, 8 ...]

3. Find the length of the string using pointers.
4. Find the maximum number in the input integer array using pointers.
5. Create a student record with name, rollno, marks of 3 subjects (m1, m2, m3). Compute the average of marks for 3 students and display the names of the students in ascending order of their average marks.

6. Create an employee record with emp-no, name, age, date-of-joining (year), and salary. If there is 20% hike on salary per annum, compute the retirement year of each employee and the salary at that time. [standard age of retirement is 55]

REFERENCES

1. E. Balaguruswamy, “Computing Fundamentals & C Programming”, Tata McGraw Hill, 2008.
2. E. Balaguruswamy, “Object Oriented Programming with C”, Tata McGraw Hill, 2nd Edition 2007.
3. Yashavant Kanetkar, “Let Us C”, 10th Edition, BPB Publications, 2010.
4. Dr. B. S. Grewal, “Numerical Methods in Engineering & Science”, Khanna publishers, 9th Edition, ISBN: 978-81-7409-248-9, 2010, 8th reprint, 2013.
5. Rudra Pratap, “Getting started with MATLAB – A Quick Introduction for Scientists and Engineers”, Oxford University Press, ISBN-13:978-0-19-806919-5, 2013.

C LANGUAGE QUICK REFERENCE

PREPROCESSOR

```
// Comment to end of line
/* Multi-line comment */

#include <stdio.h>    // Insert standard header file
#include "myfile.h"   // Insert file in current directory
#define X some text   // Replace X with some text
#define F(a,b) a+b     // Replace F(1,2) with 1+2
#define X \
    some text         // Line continuation
#undef X              // Remove definition
#ifdef X              // Conditional compilation (#ifdef X)
#else                 // Optional (#ifndef X or #if !defined(X))
#endif               // Required after #if, #ifdef
```

LITERALS

```
255, 0377, 0xff      // Integers (decimal, octal, hex)
2147463647L, 0x7fffffffL // Long (32-bit) integers
123.0, 1.23e2         // double (real) numbers
'a', '\141', '\x61'    // Character (literal, octal, hex)
'\n', '\\', '\'', '\"', // Newline, backslash, single quote, double quote
"string\n"            // Array of characters ending with newline and \0
"hello" "world"       // Concatenated strings
true, false           // bool constants 1 and 0
```

DECLARATIONS

```
int x;                // Declare x to be an integer (value undefined)
int x=255;             // Declare and initialize x to 255
short s; long l;       // Usually 16 or 32 bit integer (int may be either)
char c= 'a';           // Usually 8 bit character
unsigned char u=255; signed char m=-1; // char might be either
unsigned long x=0xffffffffL; // short, int, long are signed
float f; double d;     // Single or double precision real (never unsigned)
```

```

bool b=true;           // true or false, may also use int (1 or 0)
int a, b, c;           // Multiple declarations
int a[10];              // Array of 10 ints (a[0] through a[9])
int a[]={0,1,2};        // Initialized array (or a[3]={0,1,2}; )
int a[2][3]={ {1,2,3},{4,5,6}}; // Array of array of ints
char s[]="hello";       // String (6 elements including '\0')
int* p;                 // p is a pointer to (address of) int
char* s="hello";        // s points to unnamed array containing "hello"
void* p=NULL;           // Address of untyped memory (NULL is 0)
int& r=x;               // r is a reference to (alias of) int x
enum weekend {SAT, SUN}; // weekend is a type with values SAT and SUN
enum weekend day;        // day is a variable of type weekend
enum weekend {SAT=0,SUN=1}; // Explicit representation as int
enum {SAT,SUN} day;     // Anonymous enum
typedef String char*;   // String s; means char* s;
const int c=3;          // Constants must be initialized, cannot assign
const int* p=a;         // Contents of p (elements of a) are constant
int* const p=a;         // p (but not contents) are constant
const int* const p=a;   // Both p and its contents are constant
const int& cr=x;        // cr cannot be assigned to change x

```

STORAGE CLASSES

```

int x;                 // Auto (memory exists only while in scope)
static int x;          // Global lifetime even if local scope
extern int x;          // Information only, declared elsewhere

```

STATEMENTS

```

x=y;                  // Every expression is a statement
int x;                // Declarations are statements
;                     // Empty statement
{                     // A block is a single statement
    int x;            // Scope of x is from declaration to end of
block
    a;                // In C, declarations must precede statements

```

```

}
if (x) a;           // If x is true (not 0), evaluate a
else if (y) b;      // If not x and y (optional, may be repeated)
else c;             // If not x and not y (optional)
while (x) a;        // Repeat 0 or more times while x is true
for (x; y; z) a;     // Equivalent to: x; while(y) {a; z;}
do a; while (x);     // Equivalent to: a; while(x) a;
switch (x) {         // x must be int
    case X1: a;      // If x == X1 (must be a const), jump here
    case X2: b;      // Else if x == X2, jump here
    default: c;      // Else jump here (optional)
}
break;              // Jump out of while, do, for loop, or switch
continue;           // Jump to bottom of while, do, or for loop
return x;           // Return x from function to caller
try { a; }
catch (T t) { b; }   // If a throws T, then jump here
catch (...) { c; }   // If a throws something else, jump here

```

FUNCTIONS

```

int f(int x, int);    // f is a function taking 2 ints and returning int
void f();             // f is a procedure taking no arguments
void f(int a=0);      // f() is equivalent to f(0)
f();                  // Default return type is int
inline f();           // Optimize for speed
f() { statements; }   // Function definition (must be global)

```

Function parameters and return values may be of any type. A function must either be declared or defined before it is used. It may be declared first and defined later. Every program consists of a set of global variable declarations and a set of function definitions (possibly in separate files), one of which must be:

```

int main() { statements... }    or
int main(int argc, char* argv[]) { statements... }

```

argv is an array of argc strings from the command line. By convention, main returns status 0 if successful, 1 or higher for errors.

EXPRESSIONS

Operators are grouped by precedence, highest first. Unary operators and assignment evaluate right to left. All others are left to right. Precedence does not affect order of evaluation which is undefined. There are no runtime checks for arrays out of bounds, invalid pointers etc.

T::X	// Name X defined in class T
N::X	// Name X defined in namespace N
::X	// Global name X
t.x	// Member x of struct or class t
p → x	// Member x of struct or class pointed to by p
a[i]	// i'th element of array a
f(x, y)	// Call to function f with arguments x and y
T(x, y)	// Object of class T initialized with x and y
x++	// Add 1 to x, evaluates to original x (postfix)
x--	// Subtract 1 from x, evaluates to original x
sizeof x	// Number of bytes used to represent object x
sizeof(T)	// Number of bytes to represent type T
++x	// Add 1 to x, evaluates to new value (prefix)
--x	// Subtract 1 from x, evaluates to new value
~x	// Bitwise complement of x
!x	// true if x is 0, else false (1 or 0 in C)
-x	// Unary minus
+x	// Unary plus (default)
&x	// Address of x
p	// Contents of address p (&x equals x)
x * y	// Multiply
x / y	// Divide (integers round toward 0)
x % y	// Modulo (result has sign of x)
x + y	// Add, or &x[y]
x - y	// Subtract, or number of elements from *x to *y
x << y	// x shifted y bits to left (x * pow(2, y))
x >> y	// x shifted y bits to right (x / pow(2, y))
x < y	// Less than

<code>x <= y</code>	// Less than or equal to
<code>x > y</code>	// Greater than
<code>x >= y</code>	// Greater than or equal to
<code>x == y</code>	// Equals
<code>x != y</code>	// Not equals
<code>x & y</code>	// Bitwise and (3 & 6 is 2)
<code>x ^ y</code>	// Bitwise exclusive or (3 ^ 6 is 5)
<code>x y</code>	// Bitwise or (3 6 is 7)
<code>x && y</code>	// x and then y (evaluates y only if x (not 0))
<code>x r</code>	// x or else y (evaluates y only if x is false(0))
<code>x = y</code>	// Assign y to x, returns new value of x
<code>x += y</code>	// <code>x = x + y</code> , also <code>-=</code> <code>*=</code> <code>/=</code> <code><<=</code> <code>>>=</code> <code>&=</code> <code> =</code> <code>^=</code>
<code>x ? y : z</code>	// y if x is true (nonzero), else z
<code>throw x</code>	// Throw exception, aborts if not caught
<code>x, y</code>	// evaluates x and y, returns y (seldom used)

STDIO.H.H, STDIO.H

<code>cin >> x >> y;</code>	// Read words x and y (any type) from stdin
<code>cout << "x=" << 3 << endl;</code>	// Write line to stdout
<code>cerr << x << y << flush;</code>	// Write to stderr and flush
<code>c = cin.get();</code>	// <code>c = getchar();</code>
<code>cin.get(c);</code>	// Read char
<code>cin.getline(s, n, '\n');</code>	// Read line into char s[n] to '\n', (default)
<code>if (cin)</code>	// Good state (not EOY)?
	// To read/write any type T:

STRING (Variable sized character array)

<code>string s1, s2= "hello";</code>	// Create strings
<code>s1.size(), s2.size();</code>	// Number of characters: 0, 5
<code>s1 += s2 + ' ' + "world";</code>	// Concatenation
<code>s1 == "hello world";</code>	// Comparison, also <code><</code> , <code>></code> , <code>!=</code> , etc.
<code>s1[0];</code>	// 'h'
<code>s1.substr(m, n);</code>	// Substring of size n starting at s1[m]

<code>sl.c_str();</code>	<code>// Convert to const char*</code>
<code>getline(cin, s);</code>	<code>// Read line ending in '\n'</code>
<code>asin(x); acos(x); atan(x);</code>	<code>// Inverses</code>
<code>atan2(y, x);</code>	<code>// atan(y/x)</code>
<code>sinh(x); cosh(x); tanh(x);</code>	<code>// Hyperbolic</code>
<code>exp(x); log(x); log10(x);</code>	<code>// e to the x, log base e, log base 10</code>
<code>pow(x, y); sqrt(x);</code>	<code>// x to the y, square root</code>
<code>ceil(x); floor(x);</code>	<code>// Round up or down (as a double)</code>
<code>fabs(x); fmod(x, y);</code>	<code>// Absolute value, x mod y</code>
