# Lab Manual

On

## Computer Programming in C

As on Computer Programming I
TU

**Prepared by:**
**Er. Kiran Sharma**

TRIBHUVAN UNIVERSITY
Advanced College of Engineering and Management
C-Programming (BCA ( I/II))

# Lab-1: Introduction to C Programming

## Objectives:

- To be familiar with the basic concept of C Programming.
- To be familiar with Codeblocks
- To know the basic structure of C Program
- To run and compile the C program

## Theory:

### Basic Structure of C program:

C is a group of building blocks called functions, which is a subroutine that may include one or more statements designed to perform a specific task.

| **Documentation Section** |
|---|
| **Link Section** |
| **Definition Section** |
| **Global Declaration section** |
| **Main() function section**<br>**{**<br>**Declaration part;**<br>**Execution part;**<br>**}** |
| **Sub Program Section**<br>**Function1**<br>**Function 2** |

**Documentation section** consists of a set of comment lines giving the name of the program, author, and other details to be used later by the programmer. The compiler ignores any comment so they do not add to the file size during the time of execution. Comment lines which starts with // for single line comment OR /*….*/ for multiple line comments.

**Link section** provides instructions to the compiler to link functions from the system library.

**Definition section** defines all symbolic constants.

**Global declaration section** declares all variables used in the executable part globally.

**Main function section** is a must section and one program contains only one main. Main function section starts with the opening brace '{' and ends with closing brace '}'. It consists of a declaration and execution section. Declaration part declares all variables used in the executable part. There must be at least one statement in the executable part. Each statement ends with a semicolon except for function definitions, control statements and loops.

**Subprogram section** contains all user-defined functions that are called in the main() function.

Every C program consists of one or more modules called functions. One- of the functions must be called main () .The program will always begin by executing the main() function, which may access other functions. Any other function definitions must be defined separately, either ahead of or after main ().

The main () function can be located somewhere in the program so that the computer can determine where to start the execution. This function can be allocated anywhere in the program but the general practice is to place it as the first function for better readability.

Any function in a C program consists of valid C statements and is linked together through function calls. A function is analogous to the subroutine or a procedure in other higher-level languages. Every function in a program has a unique name and is designed to perform a specific task. Each function is defined by a block of statements, which are enclosed within a par and is treated as one single unit.

Every instruction in the C program is written as a separate statement. These statements must appear in the same order in which we wish them to be executed; unless logic of the problem demands a deliberate jump or transfer of control to a statement, which is out of sequence.

## Rules for statements:

- Generally all C statements are entered in small cases.
- Any C statement always ends with a semicolon (;).
- C has no specific rules about the position at which different parts of a statement are to be written.

**Example:-**

```
#include <stdio.h>  //header files
#include <conio.h>
void main()  //main function
{
 clrscr();    // library functiion to Clear the screen
 printf("This is first lecture in C programming"); // library function that prints the given
    string
 getch();
```

}

*The line numbering is just for the purpose of discussion. Students are advised to remember semicolon; while writing a C program to denote end of statement. Students are strictly recommended to avoid the line number while writing the program in the lab.*

- To be familiar with basic Input / Output functions.

```
1. /* a program to display  string */
2. #include<stdio.h>
3. void main()
4. {
5.         printf("Programming is fun");          //library function that displays string
6. }
```

## Discussion :

## Line 1:

The line starting with // or /*……*/ is considered as a comment and the compiler avoids this line. It describes the program but these lines are not compiled or executed.

## Line 2:

Every C program has a link section. # is called a preprocessor directive. It tells the compiler to include the file in the program which contains the definition of library functions we use in the program.

## Line 3:

Execution of any C program starts with the main( ) function. Our program will not start without a main function.

## Line 4:

Opening braces { indicates the start of block or the beginning of the function whereas the closing brace } indicates the termination of function or block. (Line 6)

## Line 5:

printf() function is defined with stdio.h which is included in the program. It takes a string as a parameter within the double quote "…." The function printf() is used to display the string. Here it will print Programming is fun on the monitor.

Every statement is ended with a semicolon ; you should now remember ; now and then.

## 1.2 Write a program as follows:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    printf("Name:   ");
    printf("Faculty:    ");          //you can display more details
    printf("College :   ");
}
```

## 1.3 To be familiar with basic Input / Output functions.

```
1.   /* a program to input and display integer number*/
2.  #include<stdio.h>
3.  void main()
4.  {
5.          int a, b, sum;
6.          printf("Enter two numbers");
7.          scanf("%d%d ",&a,&b);
8.          sum=a+b;
9.          printf("sum of %d and %d is : %d", a,b,sum);
10.         }
```

## Line 5:

This is how we declare variables in C. The keyword int is used to declare variables which will store integer type of data. Here, a, b and sum are integer variables.

## Line 6:

printf() function will display the string Enter two numbers on the monitor

### Line 7:

The function scanf() is used to input data. It (%) specifies the format of data to be read. %d specifies integer number & denoted address of. &a tells the compiler to store the integer number it read to a. Likewise, the second integer number read is stored to b.

### Line 8:

This line performs data processing, arithmetic addition. It adds the value of a and b and stores the result in a variable sum.

### Line 9:

As aforesaid printf() function is used to display messages on string. It also contains a format specifier to be printed. %d within printf() tells the compiler to print some integer variable. The variable a, b and sum are the variables to be printed respectively at corresponding %d format specifier. If a=5, b=4, the output from line 9 will be:

sum of 5 and 4 is: 9

```
printf("sum of %d and %d is : %d",a,b,sum);
```

## Write the Flowchart, algorithm and source code of the following programs.

A. WAP to calculate the product of two numbers given by the user.

B. WAP that asks the radius of the circle with the user, calculates the area of the circle and displays it.

Hint: in order to read real numbers, use float instead of int and %f instead of %d

Example:

```
float radius,area;
printf("area of circle with radius %f=%f",radius,area);
```

C. WAP to calculate the simple Interest(I) by reading Principal amount(P), time(T) and rate of Interest(R).

D. WAP to ask length, breadth and height of the cuboid and calculate the volume.

E. WAP to read temperature in degrees Celsius and convert it into Fahrenheit degrees. **[F= 1.8C + 32]**

# Lab-2: Input/Output Functions

## Objectives:

- To deal with input and output functions.

## Theory:

### Variable:

Variables are the most fundamental part of any programming language. Variable is a symbolic name which is used to store different types of data in the computer's memory. When the user gives the value to the variable, that value is stored at the same location occupied by the variable. Variables may be of integer, character, string, or floating type.

Before you can use a variable you have to declare it. As we have seen above, to do this you state its type and then give its name. The declaration does two things.

1. Tell the compiler the variable name.
2. Specifies what type of data the variable will hold.

Syntax:

datatype v1, v2, v3, …….. vn;

Where v1, v2, v3 are variable names. Variables are separated by commas. A declaration statement must end with a semicolon.

For example, int i; declares an integer variable. You can declare any number of variables of the same type with a single statement.

```
int a, b, c;
float x,y,z;
char ch;
```

declares three integers: a, b and c, three floating variables: x, y and z and one character: ch.

You have to declare all the variables that you want to use at the start of the program.

It is also possible to initialize variables using the = operator for assignment.

For example:-

```
float sum=0.0;
int bigsum=0;     char letter=`A';
```

## Data Types:

The first thing we need to know is that we can create variables to store values in. A variable is just a named area of storage that can hold a single value (numeric or character). C is very fussy about how you create variables and what you store in them. It demands that you declare the name of each variable that you are going to use and its type, before you actually try to do anything with it. Hence, data type can be defined as the storage representation and machine instruction to handle constants and variables.

There are four basic data types associated with variables:

- Primary or fundamental data type
- User- defined data type
- Derived data type
- Void type

Primary or fundamental data type:

The data type that is used without any modifier is known as the primary data type. The primary data type can be categorized as follows:

- Integral type
    - Signed Integer type
        - short int %d
        - int %d
        - long int %ld
    - Unsigned Integer type
        - short int
        - int
        - long int
- Character type
    - Signed char
    - Unsigned char
- Float type
    - Float
    - double
    - long double

## Formatted I/O Functions

### WRITING OUTPUT DATA –– THE printf () FUNCTION

Output data can be written from the computer into a standard output device using the library function printf (). This function can be used to output any combination of numerical values, single characters and strings.

Syntax: printf(control string, argl, arg2, …, argn);

Where control string refers to a string that contains formatting information, and argl, arg2 are arguments that represent the individual output data items. The argument can be written as constants, single variable or array names, or more complex expressions. A format specifier is used to control what format will be used by the printf() to print the particular variable.

Usage:
```
#include <stdio.h>
void main ()
{   char item [20] ;
    int partno ;
    float cost ;
    …
    printf ("%s %d %f", item, partno, cost);
}
```

The control string is all-important because it specifies the type of each variable in the list and how you want it printed. The way that this works is that printf scans the string from left to right and prints on the screen, or any suitable output device, any characters it encounters - except when it reaches a % character. The % character is a signal that what follows it is a specification for how the next variable in the list of variables should be printed. printf uses this information to convert and format the value that was passed to the function by the variable and then moves on to process the rest of the control string and any variables it might specify.

**ENTERING INPUT DATA – THE scanf () FUNCTION**

Input data can be entered into the computer from a standard input device by means of the C library function scanf(). This function can be used to enter any combination of numerical values, single characters and strings.

Syntax: scanf(control string, argl, arg2, …, argn)

Where control string refers to a string containing certain required formatting information, and argl, arg2… argn are argument lists that represent individual data items. The control string comprises individual groups of characters, with one character group for each data item. Each character group must begin with a percent sign (%). In this case the control string specifies how strings of characters, usually typed on the keyboard, should be converted into values and stored in the listed variables. The most obvious is that scanf has to change the values stored in the parts of computers memory that are associated with parameters (variables). The scanf function has to have the addresses of the variables rather than just their values. This means that simple variables have to be passed with a preceding &.

## Unformatted I/O Functions

Single character input function:
Syntax: variable_name= getchar()

| | |
|---|---|
| char ch; | char ch; |
| printf(“Enter a character”); | printf(“Enter a character”); |
| scanf(“%c”, &ch); | ch=getchar(); |

Single character output function:
Syntax: putchar(variable_name)

| | |
|---|---|
| printf(“The output character is:%c”, ch); | putchar(ch); |

String input function:
Syntax: gets(string variable)
String output functions:
Syntax: puts(string constant/string variable)

## Escape Sequences:

Escape sequences are used in the programming languages C and C++. These are character combinations that comprise a backslash (\) followed by some character. They give results such as getting to the next line or a tab space. They are called escape sequences because the backslash causes an "escape" from the normal way characters are interpreted by the compiler.

Common escape sequences

- \a – Bell (beep)
- \b – Backspace
- \f – Form Feed
- \n – New line
- \r – Carriage return
- \t – Horizontal tab
- \\ – Backslash
- \' – Single quotation mark
- \" – Double quotation mark

Escape sequence is a special string used to control output on a monitor.

similarly, printf(“%c”, char_var); or putchar(char_var);
can be used to display a character on the monitor.
Both the functions getchar() and putchar() reads and displays a single character.

- To write a program to read personal information and display it on the screen

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

```
void main()
{
    char name[25]; // character array to store up to 25 characters
    int roll;
    char addr[25]; // char can hold only one character at a time, so array is required
    printf(“Enter your name”);
    gets(name);    //to input a line of string like scanf(“%s”,name);
    printf(“Enter your roll”);
    scanf(“%d”,&roll);
    printf(“Enter your address: “);
    gets(addr);
    printf(“\n Your \t information: \n”);
    printf(“Name: “);
    puts(name);
    printf(“\nRoll: “);
    printf(“%d” ,roll);
    printf(“\nAddress: \n“);
    puts(addr);
    getch();
}
//REMEMBER: gets() and puts() functions can only be used with a character array.
```

## Write the Flowchart, algorithm and source code of the following programs.

1. WAP to take information of a student as input and display them suitably. Information includes name, roll no, age, phone number and email address.
2. To write a program to read a character from the keyboard and display it.

    Hint:

    %c is the format specifier for character variable

    For eg:

    ```
    char char_var;
    scanf(“%c”,&char_var);
    ```

3. WAP to separate the user given three digits number and display each separated digit in a new line.
4. WAP to convert lower characters given by the user to uppercase.

```
Hint:
'A'=65
'a'=97
97-65=32  i.e subtract 32 from the lower case letter
#include<stdio.h>
#include<conio.h>
void main()
{
    char c;
    printf("\nEnter the Lower case character");
    c=getchar();
    c-=32;  //c=c-32;
    printf("\nhere is the Uppercase of your letter=\t%c",c);  //putchar(c);
getch();
}
//WAP to convert upper case characters given by the user to the lower case. //c=c+32
```

5. WAP to calculate compound interest amount (a) when p, n, r, t are given.

   [Hint: a=P *(1 + r/n)^(nt)]  (n=number of times interest applied per time period)

6. WAP to find the maximum between two numbers using conditional operators.

# Lab-3: Control Statements (Conditional Branching)

## Objectives:

- To deal with if statements, if …..else statements, nested if, switch statements.

## Theory:

**Control statements** enable us to specify the flow of program control; ie, the order in which the instructions in a program must be executed. They make it possible to make decisions, to perform tasks repeatedly or to jump from one section of code to another.

- Decision Making Statement
- The if Statement

This is the simplest form of the branching statements. It takes an expression in parenthesis and a statement or block of statements. if the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

### if() -statement

*if(condition)*
*statement1;*

Example: following example shows whether the entered programming is positive or not

```
#include<stdio.h>
#include<conio.h>
void main( )  {
      int a;
      printf("\n Enter a number:");
      scanf("%d", &a);
      if(a>0)
       {
        printf( "\n The number %d is positive.",a);
           }
}
```

## If…else statement

The if-else statement is used to carry out a logical test and then take one of two possible actions depending on the outcome of the test (i.e., whether the outcome is true or false).

If the condition specified in the if statement evaluates to true, the statements inside the if-block are executed and then the control gets transferred to the statement immediately after the if-block. Even if the condition is false and no else-block is present, control gets transferred to the statement immediately after the if-block.

The else part is required only if a certain sequence of instructions needs to be executed if the condition evaluates to false. It is important to note that the condition is always specified in parentheses and that it is a good practice to enclose the statements in if blocks or in the else-block in braces, whether it is a single statement or a compound statement.

**Syntax of if-else statement**

```
if(condition)
{
    statement1;
    statement2;
}
else
{
    statement3;
    statement4;
}
```

The following program checks whether the entered number is positive or negative.

**Example:**

```
#include<stdio.h>
#include<conio.h>
void main( )
   {
    int a;
    printf("\n Enter a number:");
    scanf("%d", &a);
    if(a>0)
     {
      printf( "\n The number %d is positive.",a);
     }
    else
```

```
{
 printf("\n The number %d is negative.",a);
}

 }
```

## else if() ladder

```
if(condition1)
{   statement1;
    statement2;
}
else if(condition2)
{
    statement3;
    statement4;
}
else if
……..
……..
……..
……..
else
{
    Statement_n;
}
```

**Example:**

```
void main()
{
    if ( a == 2 )
    {
                printf("2\n");
    }
    else if ( a == 3 )
    {
            printf("3\n");
    }
    else if ( a == 4 )
```

```
    {
            printf("4\n");
    }
     else if ( a == 5 )
    {
            printf("5\n");
    }
}
```

## Nested else if statement:

Syntax of nested-if statements

```
if(condition1)
{
    if(condition2)
    {
            statement1;
            statement2;
    }
    else
    {
            statement3;
    }
}
else
{
    statement4;
    statement5;
}
```

**Example:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x;
    scanf("%d",&x);
    if (x < 0)
```

```
    printf("\n the no. u entered is negative");
  else
  {
    if (x == 0)
                printf("\nthe no. u entered is 0");
    else
          printf("\nthe no. u entered is positive");
  }
  getch();
}
```

## Switch statements:

A switch statement is used for multiple way selections that will branch into different code segments based on the value of a variable or expression. This expression or variable must be of integer data type or character type.

**Syntax:**

```
switch (expression)
{
    case value1:
          code segment1;
          break;
    case value2:
          code segment2;
           break;
    .
    .
    .
     case valueN:
          code segmentN;
          break;
    default:
          default code segment;
}
```

The value of this expression is either generated during program execution or read in as user input. The case whose value is the same as that of the expression is selected and executed. The optional default label is used to specify the code

segment to be executed when the value of the expression does not match with any of the case values.

The break statement is present at the end of every case. If it were not so, the execution would continue on into the code segment of the next case without even checking the case value. For example, supposing a switch statement has five cases and the value of the third case matches the value of expression. If no break statements were present at the end of the third case, all the cases after case 3 would also get executed along with case 3. If break is present only the required case is selected and executed; after which the control gets transferred to the next statement immediately after the switch statement. There is no break after default because after the default case the control will either way get transferred to the next statement immediately after switch.

**Example: program to display the day of the week**

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int day;
    printf("\nEnter the number of the day:");
    scanf("%d",&day);
    switch(day)
      {
       case 1:
            printf("Sunday");
            break;
       case 2:
            printf("Monday");
            break;
       case 3:
            printf("Tuesday");
            break;
       case 4:
            printf("Wednesday");
            break;
       case 5:
            printf("Thursday");
            break;
       case 6:
            printf("Friday");
```

```
            break;
        case 7:
            printf("Saturday");
            break;
        default:
            printf("Invalid choice");
      }

   }
```

## Write algorithm flowchart, source code and output of the following.

1. Run the following program and observe the output for different inputs:

```
# include<stdio.h>
#include<conio.h>
void main( )
{
    int x;
    printf(“Enter any number:”);
    scanf(“%d”, &x);
    if(x>5)
      printf(“%d is greater than 5\n”,x);
    if(x>10)
      printf(“%d is greater than 10\n”,x);
    if(x<100)
      printf(“%d” is less than 100\n”,x);
    getch( );
}
```

2. Write a program to check whether the given number is even or odd.
3. Write a program to input a character from the user and convert it to capital if entered in small and to small letter if it is given in capital letter. ////

   a=97  z=122 ;  A=65  Z=90

4. Write a program to input two numbers and print the larger number.
5. Write a program to input three numbers and print the largest number. ////
6. Write a program to solve the quadratic equation $Ax^2 + Bx + C = 0$. Take coefficients A, B and C from the user and check that they do not result in imaginary roots. If they do, print an appropriate error message and exit from the program.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(void)
{
        float a,b,c,desc,x1,x2,real, img;
         clrscr();
         printf("Enter the coefficients of ax62 + bx + c = 0 \n");
        scanf("%f %f %f",&a,&b,&c);
        if(a != 0)
        {
          desc  = b * b - 4 * a * c;
          if(desc >= 0)                    //real roots
            {
                x1 = (-b + sqrt(desc))/(2 * a);
                x2 = (-b - sqrt(desc))/(2 * a);
                printf("\n The real roots are : %.2f \t %.2f \n",x1,x2);
            }
          else                        //imaginary roots
        {
                real = -b/(2 * a);
                img = sqrt(-desc)/(2 * a);
                img = fabs(img);        //To find the absolute of img
                printf("\n The imaginary roots are :\n %.2f + i%.2f \n "
                  "%.2f - i%.2f",real, img,real, img);
        }
    }
    else
          printf("\n The equation is not quadratic \n Try again");
        getch();
  }
```

7. Write a program to input three sides of a triangle from the user and calculate the area of the triangle. Be sure to check that the sum of two other sides is always greater than third side.////

   Area=$\sqrt{s(s-a)(s-b)(s-c)}$
   where a,b,c are the sides of triangle and s is the semi-perimeter, s= (a+b+c)/2

8. Write a program to take numbers from 0 to 6 from the user and print Sunday if the entered number is 0, Monday if entered number is 1, Tuesday if entered number is 2 and so on. If the number does not lie between 0 and 6, print an error message.////

9. Write a program to input any number from 1 to 12 from the user. If the entered number is 1, then display January, if the number is 2 then display February and so on. Use switch-case statement.////

10. Write a program to enter any number from the user. If the number is less than 25 then, check whether the entered number is odd or even and print the appropriate message. If the number is greater than or equal to 25, then square it and display the result.////

**Use of Logical and Relational Operators**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int marks;
printf("Enter the marks of student");
scanf(%d",&marks);
if(marks<32)
    printf("Fail");
else if(marks>=32 && marks <45)
    printf("Third Division");        // {….} ar optional for single line statement
else if(marks>=45 && marks <60)
```

```
    printf("Second Division");
else
    printf("First Division");        // can check in reverse order
getch();
}
```

# Assignments:

## Write algorithm, flowcharts, source code and outputs of followings

- WAP to enter the year and find whether the given year is leap year or not.////
- WAP to find the smallest among the four entered numbers////
- WAP to read length and breadth of a room and print area and print////
    - "Auditorium" if area>2500
    - "Hall" if 500<area<=2500
    - "Big room" if 150<area<=500
    - "Small room" if area<=150
- WAP to check whether the number entered by the user is divisible by 3 but not by 7.
- Write an appropriate control structure that will examine the value of a floating point variable called temp and print one of the following messages, depending on the value assigned to the temp,
    - Ice, if the value of temp is less than 0
    - Water, if the value of temp is between 0 and 100
    - Steam, if the value of temp is more than 100
- Admission to a professional course is subject to be following conditions:
    - Marks in mathematics>=60
    - Marks in physics>=50
    - Marks in Chemistry>=50
    - Total in all three subject>=200

  Give the marks in the three subjects, WAP to process the application to list the eligible students.

- WAP that will read two numbers, display the following menu:
    - Summation
    - Sum of squares
    - Sum of cubes
    - Product

  And perform task as per users' choice (use Switch statement)

- An organization is dealing in two items, say A and B and provides the commission on the sale of these items according to the following policy:
    - Commission rate for item A is 5% up to a sale of Rs.2000. If the sale of A is above 2000, then the commission is 6% on the extra sale.
    - For B, 10% up to a sale of Rs.4000, if the sale is above 4000, commission rate is 12% on extra sale. Given the sales of both items, WAP to compute net commission.

- WAP to find the largest among the three entered numbers.

- WAP to take the option from the user for Save(S), Open(O), Exit(E) and print the corresponding message like "you want to open the file?" by using a switch case.

- WAP to take two numbers from user and one arithmetic operator(+, -, *, /, %) and perform the corresponding operation of those two numbers by using switch case.

**Note: You must submit your assignment along with your corresponding Lab Report.**

# Lab-4 & 5: CONTROL STATEMENTS(Loop)

## Objectives:

- To deal with while loop, do…while loop and for loop.
- To be familiar continue ,break

## Background Theory:

### Iterative Structure

### Loop

A loop is a segment of code that is executed repeatedly.

THE for LOOP

The for loop is used only when the number of iterations is predetermined, for example, 10 iterations or 100 iterations.

Program/Example

The general format for the for loop is

***for (initializing; continuation condition; update)***

simple or compound statement

For example,

```
for (i = 0; i < 5; i++)
{
   printf("value of i");
}
```

### Explanation

- The for loop has four components; three are given in parentheses and one in the loop body.
- All three components between the parentheses are optional.
- The initialization part is executed first and only once.
- The condition is evaluated before the loop body is executed. If the condition is false then the loop body is not executed.
- The update part is executed only after the loop body is executed and is generally used for updating the loop variables.
- The absence of a condition is taken as true.

- It is the responsibility of the programmer to make sure the condition is false after certain iterations.

## THE while LOOP

The while loop is used when you want to repeat the execution of a certain statement or a set of statements (compound statement). The while loop tests the loop condition and executes repeatedly till the condition being tested remains true. The while loop must test a condition that will eventually become false, otherwise the loop will continue forever.

## Program/Example

The general format for a while loop is

```
   while (condition)
{
   simple or compound statement (body of the loop)
}
```

**For example,**

```
i = 0;
while (i<5)
{
   printf(" the value of i is %d\n", i);
   i = i + 1;
}
```

## Explanation

- Before entering into the loop, the while condition is evaluated. If it is true then only the loop body is executed.
- Before making an iteration, the while condition is checked. If it is true then the loop body is executed.
- It is the responsibility of the programmer to ensure that the condition is false after certain iterations; otherwise, the loop will make infinite iterations and it will not terminate.
- The programmer should be aware of the final value of the looping variable. For example, in this case, the final value of the looping variable is 5.
- While writing the loop body, you have to be careful to decide whether the loop variable is updated at the start of the body or at the end of the body.

## THE do-while LOOP

It executes the loop at least once. The condition is not tested until the body of the loop has been executed once. If the condition is true after the first loop iteration, the loop continues, otherwise the loop terminates.

**Program/Example**

The general format for a do-while loop is

```
        do
{
   simple or compound statement
}   while (condition);
   For example,
i = 0;
do
{
   printf(" the value of i is %d\n", i);
   i = i + 1;
}
while (i<5);
```

## Explanation

The loop body is executed at least once.

- The condition is checked after executing the loop body once.
- If the condition is false then the loop is terminated.
- In this example, the last value of i is printed as 5.

| Syntax of while loop | Syntax of do…while loop |
|---|---|
| Initialization;<br>while(condition)<br>{<br><br>body of while loop<br>update;<br>} | Initialization;<br>do<br>{<br><br>body of while loop<br>update;<br>} while(condition); |

## THE break STATEMENT

Just like the switch statement, break is used to break any type of loop. Breaking a loop means terminating it. A break terminates the loop in which the loop body is written.

**Program/Example**

For example,

```
i = 0;
    while (1)
    {
```

```
    i = i + 1;
    printf(" the value of i is %d\n");
    if (i>5) break;
}
```

## Explanation

- The while (1) here means the while condition is always true.
- When i reaches 6, the if condition becomes true and break is executed, which terminates the loop.

## THE continue STATEMENT

The break statement breaks the entire loop, but a continue statement breaks the current iteration. After a continue statement, the control returns to the top of the loop, that is, to the test conditions. Switch doesn't have a continue statement.

## Program/Example

Suppose you want to print numbers 1 to 10 except 4 and 7. You can write:

```
for(i = 0, i < 11, i++)
{
  if ((i == 4) || (i == 7)) continue;
  printf(" the value of i is %d\n", i);
}
```

## Explanation

- If i is 1 then the if condition is not satisfied and continue is not executed. The value of i is printed as 1.
- When i is 4 then the if condition is satisfied and continue is executed.
- After executing the continue statement, the next statement, (printf), is not executed; instead, the updated part of the for statement (i++) is executed.

## Task 3.a

***Write the flowchart, algorithm and source code, output of the following programs.***

- **Write a number to print all the numbers from 1 to 100.**

  Hint: use loop as follows

```
for(x=1;x<=100;x++)
        printf("%d \t", x);
```

**Note**: in loop x=x+1, x +=1 and x++ are the same.

- **WAP to print numbers from 100 to 10 which are exactly divided by 11 but not by 7.**
- **Write a program to print the factorial of the number entered by the user. Use any loop of your choice**.7! =7*6*5*4*3*2*1

```
#include<stdio.h>
#include<conio.h>
void main( )
    {
     int i,num,fact ;
     clrscr();
     printf("Enter a number whose factorial is to be found  \n");
     scanf("%d",&num);
     if( num < 0)
           printf("\n Factorial of negative number does not exist \n");
     else
           {
             for( fact = 1;i = num; i >= 1; i--)
                  fact *= i;
             printf("\n %d ! =  %d ", num, fact);
           }
     getch();
    }
```

- **WAP to find the HCF and LCM of two given numbers.**
- **WAP to display the first 15 terms of the following series.**

```
-12, -14, -18, -26, -42 .
a=-12;
for(i=1; i<=15; i++)
        {
                print a;
                a= a- pow(2,i);
        }
```

- **WAP to calculate the value of sine series, by summing first n terms of Maclurinus' series.** (Hint: sin x = x – x3/3! + x5/5! – x7/7!+… )
- **Write a program to print Fibonacci series up to $m^{th}$ term.**

1, 1 , 2 , 3 , 5 ,8, 13,…….

Hint.

```
f1 = f2 =1
f3 = f1 +f2
f4 = f3 +f2
```

.....

```
#include<stdio.h>
#include<conio.h>
void main( )
        {
                int a = 0 ,b = 1,c,m,i;
                clrscr();
                printf("How many terms of Fibonacci sequences \n");
                scanf("%d",&m);
                //printing the sequence
                  printf("\n %d   ",b);
                  for(i = 1 ; i < m ; i++)
                    {
                        c = a + b;
                        a = b;
                        b = c;
                        printf("%d   ",c);
                }
                getch();
}
```

## LAB Task 3.b

***Write the flowchart, algorithm and source code, output of the following programs.***

- **WAP to take five digit numbers to check if the number is Palindrome or not.**

  (Hint:- 12321,25852,45654) ///

- **WAP to find prime numbers between any ranges of two numbers given by the user.**

```
 A program to print all prime numbers from n1 to n2
#include<stdio.h>
#include<conio.h>
void main( )
{
        int i,j,flag,n1,n2;
        clrscr();
        printf(“enter n1 and n2 “);
```

```
scanf("%d%d",&n1,&n2);
printf("The prime number from n1 to n2 are:\n");
for(i = n1 ; i <= n2 ; i++)
        {
         for(j = 2 ; j < i ; j++)
          if((i % j) == 0)
            {
                flag = 0;
                break;
            }
          else
            flag = 1;
        if(flag )
          printf("%3d  ",i);
        }
        getch();
}
```

- **WAP to print all Armstrong numbers between any range of two numbers given by the user. (Hint: $1^3+5^3+3^3=153$ i.e $a^3+b^3+c^3=abc$).**
- **WAP to print the following using for loops.**

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

- WAP prints this triangle .

```
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
```

- **Write a program to read numbers until user enters zero.The program should display the count of +ve and _ve numbers separately**

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

```
void main( )
  {
    float num;
    int count_p = 0,count_n = 0;
    clrscr();
   printf("Enter numbers.Zero to display the result\n");
   scanf("%f",&num);
   while( num != 0 )
    {
     if(num > 0)
            count_p ++;
     else
           count_n++;
     scanf("%f",&num);
    }
   printf("\n Count of +ve numbers = %d",count_p);
   printf("\n Count of -ve numbers = %d",count_n);
   getch();
  }
```

- **A program to calculate the sum of digits of a number**

```
#include<stdio.h>
#include<conio.h>
void main(void)
    {
     int digit,number,n,sum = 0;
     clrscr();
     printf("Enter the number  \n");
     scanf("%d",&number);
     n = number;
     do
  {
   digit = number % 10;
   number /= 10;
   sum += digit;
  }
     while(number != 0);
     printf("The sum of digits of %d is %d",n,sum);
     getch();
}
```

## Assignment:

***Write the flowchart, algorithm and source code, output of the following programs.***

- Write a program to print all the numbers from 100 to 1.
- Input a number n from the user. Then find the sum of all numbers from 1 to n and print the sum.
- Write a program to display all the even numbers from 1 to n. read n from the user.
- WAP to calculate the sum of first 10 terms of the following series using for loop
    - 1 5 9 13 ……
- Write a program to find the reverse of the number entered by the user. Then check whether the reverse is equal to the original number. If the reverse and original numbers are equal, display a message telling that the number is a palindrome
- WAP to print your name 10 times.
- WAP to print the leap year between 1800 to 3000 A.D.

  *Hint: The logic is that the year is either divisible by both 100 and 4 , OR it's only divisible by 4 not by hundred.*

- WAP to compute and display the table of number n given by the user.
- WAP to find the sum of numbers between 1111 to 4444.
- WAP to compute the given series of nth order.

  $$Y = 1 + x + x2/ 2 ! + x3/3! + x4/4!+\ldots+xn/n!$$

- WAP to find the number of and sum of all integers greater than 100 and less than 200 that are exactly divisible by 7 but not by 5
- WAP to calculate sum of first n odd integers.(ask value of n from the user)
- WAP to add the first seven terms of the following series using a for loop.

  1/1!+ 2/2!+ 3/3!+…

- WAP to print the following table.

  0 1 1 1 1
  2 0 2 2 2
  3 3 0 3 3
  4 4 4 0 4

5 5 5 5 0

- WAP to print the following table.

| | | | | |
|---|---|---|---|---|
| **1** | **2** | **3** | **4** | **5** |
| **2** | **4** | **6** | **8** | **10** |
| **3** | **6** | **9** | **12** | **15** |
| **4** | **8** | **12** | **16** | **20** |
| **5** | **10** | **15** | **20** | **25** |

- WAP to display the first 10 terms of following series.

**1 2 5 10 17 …….**

```
a=1
for(i=1; i<=10; i++)
        print a
        a=a+(2*i-1);
```

- WAP to compute the sum of the following series without using pow() function.

**$1^2 - 2^2 + 3^2 - 4^2 \ldots. + (-1)^{n+1} n^2$** . where the value of n is entered by the user.

- WAP to find the terms in the given series till the value of the term is less than 1000.

**$(1^2+2^2)/3$ , $(2^2+3^2)/4$ , $(3^2+4^2)/5$ , …….**

- WAP to find the sum of the series Sn = ∑ 1/n2, value of n is given by the user.
- WAP that generates the table of values of equationY = 2 $e^{-0.1}$t sin 0.5 t.(where t varies from 0 to 60) //you have to use exp(), sin() functions

# Lab-6: Arrays

## Objectives:

- To deal with arrays.
- To be familiar with array structures, accessing and manipulating multidimensional data.

## Background Theory:

### Arrays:

Array is defined as a homogeneous collection of similar data elements.

### Syntax :

```
datatype arrayname[size];
```

**eg.**

int A[4]

datatype arrayname[size][size];

**eg.**

float B[5][5];

An array is a data structure used to process multiple elements with the same data type when a number of such elements are known. You would use an array when, for example, you want to find out the average grades of a class based on the grades of 50 students in the class. Here you cannot define 50 variables and add their grades. This is not practical. Using an array, you can store grades of 50 students in one entity, say grades, and you can access each entity by using subscripts as grades[1], grades[2]. Thus you have to define the array of grades of the float data type and a size of 50. An array is a composite data structure; that means it had to be constructed from basic data types such as array integers.

### One or single dimensional array:

There are several forms of an array in C-one dimensional and multi-dimensional arrays. In a one dimensional array, there is a single subscript or index whose value refers to the

individual array element which ranges from 0 to n-1 where n is the size of the array. For e.g. int a[5] ; is a declaration of one dimensional array of type int. Its elements can be illustrated as

1st element 2nd 3rd 4th 5th element

a[0] a[1] a[2] a[3] a[4]

2000 2002 2004 2006 2008

The elements of an integer array a[5] are stored contiguous memory locations. It is assumed that the starting memory location is 2000. As each integer element requires 2 bytes, subsequent elements appear after a gap of 2 locations.

The general syntax of array is [i.e.declaration of an array :]

data_type array_name[size];

• data_type is the data type of array. It may be int, float, char, etc.

• array_name is the name of the array. It is the user-defined name for an array. The name of the array may be any valid identifier.

• the size of the array is the number of elements in the array. The size is mentioned within[ ]. The size must be an int constant like 10, 15, 100, etc or a constant expression like symbolic constant. For example

#define size 80

a [size] ;

The size of the array must be specified [i.e. should not be blank) except in array initialization.

int num[5] ; i.e. num is an integer array of size 5 and store 5 integer values char name[10] ; i.e. name is a char array of size 10 and it can store 10 characters.

float salary[50] ; i.e. salary is a float type array of size 50 and it can store 50 fractional numbers.

## Initialization of array:

The array is initialized like follow if we need time of declaration

data_type array_name[size] = {value1, value2, ....., valuen} ;

For eg.

1. int subject[5] = {85, 96, 40, 80, 75} ;

2. char sex[2] = {‘M’, ‘F’} ;

3. float marks[3] = {80.5, 7.0, 50.8} ;

4. int element[5] = {4, 5, 6}

In example(4), elements are five but we are assigning only three values. In this case the value to each element is assigned like following

element[0] = 4

element[1] = 5

element[2] = 6

element[3] = 0

element[4] = 0

i.e. missing element will be set to zero

## Accessing elements of array:

If we have created an array, the next thing is how we can access (read or write) the individual elements of an array. The accessing function for array is

array_name[index or subscript]

For e.g.

int a[5], b[5] ;

a[0] = 30 ; /* acceptable */

b = a ; /* not acceptable */
if (a<b)

{_ _ _ _ } /* not acceptable */

we can assign integer values to these individual element directly:

a[0] = 30 ;

a[1] = 31 ;

a[2] = 32 ;

a[3] = 33 ;

a[4] = 34 ;

A loop can be used to input and output the elements of an array.

## Examples of single dimensional array:

/* Program that reads 10 integers from keyboard and displays entered numbers in the screen*/

```
#include<stdio.h>

void main( )

 {

int a[10], i ;

clrscr( ) ;

printf (“Enter 10 numbers : \t”) ;

for (i=0 ; i<10 ; i++)

scanf(“%d”, &a[i]) ; /*array input */

printf (“\n we have entered these 10 numbers : \n”) ;

for (i=0 ; i<10 ; i++)

printf (“\ta[%d]=%d”, i, a[i] ) ; /* array output */
getch( ) ;

}
```

Output:

Enter 10 numbers : 10 30 45 23 45 68 90 78 34 32

We have entered these 10 numbers:

a[0] = 10 a[1] = 30 - - - - - - - - - - - - - - - - -a[9] = 32

```
/* Program to sort n numbers in ascending order */

void main( )

{

int num[50], i, j, n, temp,

clrscr( ) ;

printf("How many numbers are there? \t") ;

scanf("%d", &n) ;

printf("\n Enter %d numbers: \n", n) ;

for (i=0; i<n; i++)

{

for (j=i+1 ; j<n ; j++)

{

if(num[i]>num[j]) ;

{

temp = num[i] ;

num[i] = num[j] ;

num[j] = temp ;
} /* end of if */

} /*end of inner loop */

} /* end of outer loop */

printf("\n The numbers in ascending order : \n") ;

for (i=0 ; i<n ; i++)

printf("\t%d, num[i] ) ;

getch( );
```

} /* end of main */

Output:

How many numbers are there? 5

Enter 5 numbers : 12 56 3 9 17

The numbers in ascending order: 3 9 12 17 56

## Multi-dimensional arrays:

An array of arrays is called a multi-dimensional array. For example a one dimensional array of one dimensional array is called a two dimensional array. A one dimensional array of two dimensional arrays is called three dimensional arrays, etc. The two dimensional array are very useful for matrix operations.

## Declaration of two dimensional array:

Multidimensional arrays are declared in the same manner as one dimensional array except that a separate pair of square brackets are required for each subscript i.e. two dimensional array requires two pair of square bracket ; three dimensional array requires three pairs of square brackets, four dimensional array require four pair of square brackets, etc.

The general format for declaring multidimensional array is

data_type array_name [size1] [size2] ..... [sizen] ;

For example:

1. int n[5] [6] ;

2. float a[6] [7] [8] ;

3. char line [5] [6] ;

4. add [6] [7] [8] [9] ;

## Initialization of multidimensional array:

Similar to one dimensional array, multidimensional array can also be initialized. For e.g.

1.int [3] [2] = {1, 2, 3, 4, 5, 6) ;

The value is assigned like follow:

a[0] [0] = 1 ;

a[0] [1] = 2 ;

a[1] [0] = 3 ;

a[1] [1] = 4 ;

a[2] [0] = 5 ;

a[2] [1] = 6 ;

- int a[3] [2] = { {1,2}, {3,4}, {5,6} } ;

In the above example, the two values in the first inner pair of braces are assigned to the array element in the first row, the values in the second pair of braces are assigned to the array element in the second row and so on. i.e.

a[0] [0] = 1 ; a[0] [1] = 2 ;

a[1] [0] = 3 ; a[1] [1] = 4 ;

a[2] [0] = 5 ; a[2] [1] = 6 ;

- int a[3] [3] = { {1,2}, {4,5}, {7, 8, 9} } ;

In the above example, the values are less than the elements of the array. The value assign like

follow:

a[0] [0] = 1 a[0] [1] = 2 a[0] [2] = 0

a[1] [0] = 4 a[1] [1] = 5 a[1] [2] = 0

a[2] [0] = 7 a[2] [1] = 8 a[2] [] = 9

In the above example the value zero is assigned to a[0] [2] and a[1] [2] because no value is assigned to these.

- int a[ ] [3] = {12, 34, 23, 45, 56, 45} ; is perfectly acceptable.

It is important to remember that while initializing a 2-dimensional array, it is necessary to mention the second (column) size whereas first size (row) is optional.

In above example, value assign like follow

a[0] [0] = 12 a[0] [1] = 34 a[0] [2] = 23

a[1] [0] = 45 a[1] [1] = 56 a[1] [2] = 45

int a[2] [ ] = {12, 34, 23, 45, 56, 45} ;

int a[ ] [ ] = {12, 34, 23, 45, 56, 45} ;

would never work.

## Accessing elements of multidimensional array:

We can access the multidimensional array with the help of following accessing function:

array_name [index1] [index2]. . . . . [indexn]

For example:

int a[5] [6] ;

We can write the following statement

1. a[0] [2] = 5 ;

2. x = a [3] [2] ;

3. printf("%d", a[0] [0] ) ;

4. printf("%d", a[2] [2] ) ;

If we want to read all the elements of above array, we can make a loop like

for (i=0; i<=4; i++)

for (j=0; j<=5; j++)

scanf("%d", &a[i] [j] ) ;

If we want to print all the elements of above array a then the loop is

for (i=0;j<=4;i++)

for (j=0; j<=5; j++)

printf("%d", a[i] [j] ) ;

## Processing an array:

```
/* Program to read & display 2×3 matrix */

#include<stdio.h>

#include<conio.h>

void main( )

{

int matrix [2] [3], i, j ;

clrscr( ) ;

for (i=0 ; i<2 ; i++)

for (j = 0 ; j<3 ; j++)

{

printf("Enter matrix [%d] [%d] : \t", i, j) ;

scanf("%d", &matrix [i] [j] ) ;

}

printf ("\n Entered matrix is : \n") ;
for (i=0 ; i<2 ; i++)

{

for(j=0; j<3; j++)

{

printf("%d\t", matrix [i] [j]) ;

}

printf("\n");

}

getch( )
```

```
}
```

Output:

```
Enter matrix [0] [0] : 1

Enter matrix [0] [1] : 2

Enter matrix [0] [2] : 3

Enter matrix [1] [0] : 4

Enter matrix [1] [1] : 5

Enter matrix [1] [2] : 6

Enter matrix is

 1 2 3

4 5 6
```

```
/* Program to read two matrices and display their sum */

#include<stdio.h>

#include<conio.h>
void main( )

 {

int a[3] [3], b[3] [3], s[3] [3], i, j ;

clrscr( ) ;

printf("Enter first matrix : \n") ;

for (i=0 ; i<3 ; i++)

 {

for (j=0 ; j<3 ; j++)

 {

scanf("%d", &a[i] [j]) ;
```

```
            }
    }
printf("Enter second matrix : \n") ;
for (i=0 ; i<3 ; i++)
{
for (j=0 ; j<3 ; j++)
{
scanf("%d", &b[i] [j]) ;
            }
    }
for (i=0 ; i<3 ; i++)
{
            //adding the matrix
for (j=0 ; j<3 ; j++)
{
s[i] [j] = a[i] [j] + b[i] [j] ;
            }
    }
    printf("\n The sum matrix is : \n") ;
for (i=0 ; i<3, i++)
{
 for(j=0 ; j<3 ; j++)
{
 printf("\t%d", s[i] [j]) ;
```

```
}
printf("\n") ;
}
getch( ) ;
}
```

Output:

```
Enter first matrix:
1 2 3
4 5 6
7 8 9
Enter second matrix:
9 8 7
6 5 4
3 2 1
The sum matrix is
10 10 10
10 10 10
10 10 10
```

**/* Program to read two matrices and multiply them if possible */**

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int a[10] [10], b[10] [10], s[10] [10] ;
```

```
int m, n, l, p, i, j, k ;

clrscr( );

printf("Enter row of first matrix(<=10) : \t") ;

scanf("%d", &m) ;

printf("Enter column of first matrix(<=10) : \t") ;

scanf("%d", &n) ;

printf("Enter row of second matrix(<=10) : \t") ;

scanf("%d", &l) ;

printf("Enter column of second matrix(<=10) : \t") ;

scanf("%d", &p) ;

if (n!=l)
printf("Multiplication is not possible :") ;

else

{

printf("Enter the first matrix : \n") ;

for (i=0 ; i<m ; i++)

{

for (j=0 ; j<n ; j++)

{

printf("Enter a[%d] [%d] : \t", i, j) ;

scanf ("%d", &a [i] [j] ;

}

}

printf("Enter the second matrix : \n") ;
```

```
for (i=0 ; i<l ; i++)
{
for (j=0 ; j<p ; j++)
{
printf (“Enter b[%d] [%d] :\t”, i, j) ;
scanf(“%d”, &b[i] [j] );
}
}
for (i=0 ; i<m ; i++)
{
for (j=0 ; j<p ; j++)
{
s[i] [j] = 0 ;
for (k=0 ; k<n ; k++)
{
s[i] [j] = s[i][j] + a[i] [k] * b[k] [j] ; }
                    }
            }
printf (“The matrix multiplication is : \n”) ;
for (i=0 ; i<m; i++)
{
for (j=0 ; j<p ; j++)
{
printf(“%d\t”, s[i] [j],
```

```
}

printf("\n") ;

}

} /* end of else */

getch( ) ;

} /* end of main( ) */
```

Output:

```
Enter row of the first matrix (<=10) : 2

Enter column of the first matrix (<=10) : 1

Enter row of the second matrix (<=10) : 1

Enter column of the second matrix (<=10) : 2

Enter the first matrix :

Enter a[0] [0] : 2

Enter a[1] [0] : 2

Enter the second matrix :

Enter b[0] [0] : 3

Enter b[0] [1] : 3

The matrix multiplication is :

6 6

6 6
```

## Write algorithm, flowchart, source code and output of followings.

- Declare an integer array naming box1 of size 5 and initialize the array box1 with numbers 0, 11, 22, 33, 44 and declare another float array box2, which will take user input. Display the contents of both arrays simultaneously.
- WAP to output the character array in the following format.

array[0]= A

array[1]= R

array[2]= R

array[3]= A

array[4]= Y

● WAP to find the sum of 10 elements of an array. Read all elements from the user. ● WAP to find the largest element and smallest element of an array.

● WAP to sort the contents of an array (ascending and descending).

● WAP to add two matrices A and B. Display the result in matrix format. Use function. ● Write a program to multiply two matrices of orders m x n and p x q. Read orders from the user. Before multiplying, check that the matrices can actually be multiplied. If they cannot be multiplied, then display the error message "Order Mismatch ! Matrices cannot be multiplied !" ///

● Write a program to find transpose of a m*n matrix. ///

● Write a C program to read two strings and concatenate them using user defined functions(without using library functions).

● WAP in C to read a string . create a user defined function to reverse it and display it in the main() function. ///

## Assignments:

### Write algorithm, flowchart, source code and output of followings.

● WAP that will enter a line of text, store it in an array and then display it backwards. ● WAP that will examine how many characters are letters, how many are digits, how many are white space characters in a line.

● WAP that will examine each character in character type array and determine how many vowels and consonants are there.

● WAP that will examine each character in a character type array called text and print out the result by converting the lowercase letter to uppercase and vice versa. ● WAP that will examine each character in character type array called text and print result by replacing all the vowels by "*" character.

● WAP to count the frequency of characters present in a line of text and print the result on screen.

● WAP to read the "n" number of a person's age in an array and print minimum, maximum and average age.

● WAP to take co-ordinate of ten different points and find the distance between first point and last point where distance is the sum of the distance between consecutive points not a

displacement. ///

- WAP that will input the names of ten persons and display the result by printing in ascending order. ///
- WAP that accepts a sentence of words and counts number of words that a sentence has then display each words of the sentence in different lines.///
- 
- WAP to take input 3*3 matrices and create a new matrix by replacing all the elements of the previous matrix by 15 if the element of the previous matrix is less than 5. ● Write two 3*4 matrices A and B in your program and print them. And obtain matrix C=2*(A+B) and print.
- WAP to take two 1- dimensional arrays of size n and m and merge them into a single array with size n + m. And display them. ///

**Note: You must submit your assignment along with your corresponding lab Report.**

# Lab-7: Functions

## Objectives:

- To deal with different categories of function
- To be familiar with recursive function, return statement, function definition, function prototype.

## Background Theory:

### FUNCTION

### Introduction

A function is defined as a self contained block of statements that performs a particular task. This is a logical unit composed of a number of statements grouped into a single unit. It can also be defined as a section of a program performing a specific task. [Every C program can be thought of as a collection of these functions.] Each program has one or more functions. The function main( ) is always present in each program which is executed first and other functions are optional.

### Advantages of using Functions:

The advantages of using functions are as follows:

1. Generally a difficult problem is divided into subproblems and then solved. This divide and conquer technique is implemented in C through functions.
2. A program can be divided into functions, each of which performs some specific task. So, the use of C functions modularizes and divides the work of a program.
3. When some specific code is to be used more than once and at different places in the program, the use of function avoids repetition of that code.
4. The program becomes easily understandable. It becomes simple to write the program and understand what work is done by each part of the program.
5. Functions can be stored in a library and reusability can be achieved.

### Types of Functions

C program has two types of functions:

1. Library Functions
2. User defined functions

**Library Functions:**

These are the functions which are already written, compiled and placed in C Library and they are not required to be written by a programmer. The function's name, its return type, their argument number and types have been already defined. We can use these functions as required.

For example: printf(), scanf(), sqrt(), getch(), etc.

**User defined Functions:**

These are the functions which are defined by the user at the time of writing a program. The user has the choice to choose its name, return type, arguments and their types. The job of each user defined function is as defined by the user. A complex C program can be divided into a number of user defined functions.

For example:

```
#include<stdio.h>
double convert (int) ; /* function prototype */
void main( )
{
int c ;
double d ;
clrscr( );
printf ("Enter temperature in Celsius: ") ;
scanf ("%d", &c) ;
d = convert(c) ; /*Function call*/
printf ("The Fahrenheit temperature of %d C = %lf F",c, d) ;
getch( );
}
double convert (int c) /* function definition */
{
double f ;
f = 9.0*c/5.0+32.0 ;
return f ;
}
```

**#What is the main( ) function?**

The function main() is an user defined function except that the name of function is defined or fixed by the language. The return type, argument and body of the function are defined by the programmer as required. This function is executed first, when the program starts execution.

## Function Declaration or Prototype:

The function declaration or prototype is a model or blueprint of the function. If functions are used before they are defined, then function declaration or prototype is necessary. Many programmers prefer a "top-down" approach in which main appears ahead of the programmer defined function definition. Function prototypes are usually written at the beginning of a program, ahead of any user-defined function including main(). Function prototypes provide the following information to the compiler.

− The name of the function
− The type of the value returned by the function
− The number and the type of arguments that must be supplied while calling the function.

In "bottom-up" approach where user-defined functions are defined ahead of main() function, there is no need for function prototypes.

The general syntax of function prototype is

return_type function_name (type1, type2, ...., type n) ;

where,

return_type specifies the data type of the value returned by the function. A function can return values of any data type. If there is no return value, the keyword void is used.

type1, type2, ....., type n are type of arguments. Arguments are optional.

**For example:**

```
int add (int, int) ; /* int add (int a, int b) ;*/
void display (int a) ; /* void display (int); */
```

## Function definition:

A function definition is a group of statements that is executed when it is called from some points of the program.

The general syntax is

```
return_type function_name (parameter1, parameter2, ....., parameter n)
{
 - - - - -
statements ;
- - - - -
}
```

where,

- return_type is the data type specifier of data returned by the function.
- function_name is the identifier by which it will be possible to call the function.

- Parameters(as many as needed) : Each parameter consists of a data type specifier followed by an identifier like any regular variable declaration. (for eg: int x) and which acts within the function as a regular local variable. They allow passing arguments to the function when it is called. The different parameters are separated by commas.
- statements is the function's body. It is a block of statements surrounded by braces{}.
- The first line of the function definition is known as function header.

```
int addition(int a, int b) /* function header */
{
 int r;
r = a+b;
return r;
}
```

## return statement:

The return statement is used in a function to return a value to the calling function. It may also be used for immediate exit from the called function to the calling function without returning a value. This statement can appear anywhere inside the body of the function. There are two ways in which it can be used:

```
return ;
return (expression) ;
```

where return is a keyword. The first form of return statement is used to terminate the function without returning any value. In this case only the return keyword is written.

```
/* Program to understand the use of return statement */
#include<stdio.h>
void funct( int age, float ht) ;
void main()
{
int age ;
float ht ;
clrscr( );
printf ("Enter age and height :") ;
scanf ("%d%f", &age, &ht) ;
funct (age, ht) ;          // function calling 20 6
getch( );
}
void funct (int age, float ht)
{
if (age>25)
{
 printf ("Age should be less than 25\n") ;
```

```
return ;
}
if (ht<5)
{
 printf (“Height should be more than 5\n”);
return ;
}
print (“selected \n”) ;
}
```

The second form of return statement is used to terminate a function and return a value to the calling function. The value returned by the return statement may be any constant,variable, expression or even any other function call which returns a value.

For example:

```
return 1 ;
return x++;
return (x+y*z);
return (3*sum(a,b)) ;
```

## Calling a function (or A calling function):

A function can be called by specifying its name, followed by a list of arguments enclosed in parenthesis and separated by commas in the main() function. The syntax of the call if function return_type is void is:

```
 function_name (parameter name) ;
```

If function return int, float or any other type value then we have to assign the call to same type value like

```
variable = function_name(parameter) ;
```

**For example:**

```
m = mul(x, y)
```

The type of m is the same as the return type of mul function. So, we can write

printf (“%d”, mul(x, y) ); also as printf (“%d”, m) ;

## Actual and Formal Parameters:

The arguments are classified into

1. Actual Parameters
2. Formal Parameters

**Actual Parameters:**

When a function is called some parameters are written within parenthesis. These are known as the actual parameter.

For example

```
main()
{
 - - - - -
convert(c) ; actual parameter
- - - - -
}
```

**Formal parameters**

Formal Parameters are those which are written in the function header.

For example

```
double convert (int a) /* function header */
{
formal parameter
}
```

## Call by value & Call by reference ( Or Pass by value & Pass by reference)

An argument is data passed from a program to the function. In function, we can pass a variable by two ways.

1. Pass by value (or call by value)
2. Pass by reference (or call by reference)

**Function call by value:**

In this the value of actual parameter is passed to formal parameter when we call the function. But actual parameters are not changed. For eg:

```
#include<stdio.h>
void swap(int, int) ; /*function prototype*/
void main( )
{
int x, y ;
clrscr( ) ;
x = 10 ;
swap (x,y) ; /*function call by value */  // x y are actual
printf ("x = %d \n", x) ;
printf ("y=%d\n", y)
getch( ) ;
}
void swap (int a, int b) /*function definition */  // a b are formal
{
```

```
int t ;
t = a ;
a = b ;
b = t ;
}
```
Output x = 10
y = 20

**Function call by reference:**

In this type of function call, the address of variable or argument is passed to the function as argument instead of the actual value of the variable.

## A RECURSIVE FUNCTION

### Introduction

A recursive function is a function that calls itself. Some problems can be easily solved by using recursion, such as when you are dividing a problem into sub- problems with similar natures. Note that recursion is a time-consuming solution that decreases the speed of execution because of stack overheads. In recursion, there is a function call and the number of such calls is large. In each call, data is pushed into the stack and when the call is over, the data is popped from the stack. These push and pop operations are time-consuming operations. If you have the choice of iteration or recursion, it is better to choose iteration because it does not involve stack overheads. You can use recursion only for programming convenience. A sample recursive program for calculating factorials follows.

### Program

```
#include <stdio.h>
int fact(int n);
main()
{
   int k = 4,i;
   i =fact(k);       \\ A
   printf("The value of i is %d\n",i);
}
int fact(int n)
{
if(n<=0)         \\ B
   return(1);          \\ C
else
```

```
   return(n*fact(n-1)); \\ D
}
```

## Explanation

You can express factorials by using recursion as shown:
fact (5) = 5 * fact (4)
In general,
fact (N) = N * fact (N-1)
fact 5 is calculated as follows:
fact (5) = 5 * fact (4) i.e. there is call to fact (4) \\ A
   fact (4) = 4 * fact (3)
   fact (3) = 3 * fact (2)
   fact (2) = 2 * fact (1)
   fact (1) = 1 * fact (0)
   fact (0) = 1 \\ B
fact (1) = 1 * 1, that is the value of the fact(0) is substituted in
   fact (2) = 2 * 1 = 2
   fact (3) = 3 * 2 = 6
   fact (4) = 4 * 6 = 24
   fact (5) = 5 * 24 = 120 \\ C

- The operations from statements B to A are collectivelly called the winding phase, while the operations from B to C are called the unwinding phase. The winding phase should be the terminating point at some time because there is no call to function that is given by statement B; the value of the argument that equals 0 is the terminating condition. After the winding phase is over, the unwinding phase starts and finally the unwinding phase ends at statement C. In recursion, three entities are important: recursive expressions, recursive condition, and terminating condition. For example,

fact ( N) = N * fact (N-1) N > 0
= 1 N = 0

  - N * fact (N-1) indicates a recursive expression.
  - N > 0 indicates a recursive condition.
  - N = 0 indicates a terminating condition.
- You should note that the recursive expression is such that you will get a terminating condition after some time. Otherwise, the program enters into an infinite recursion and you will get a stack overflow error. Statement B indicates the terminating condition, that is, N = 0.
- The condition N > 0 indicates a recursive condition that is specified by the else statement. The recursive expression is n * fact(n-1), as given by statement D.

## Iteration

A function is called from the definition of the same function to be repeated.

1. Loop is used to do repeated tasks.
2. Recursive is a top-down approach to problem solving
3. Iteration is like a bottom-up approach.
4. In recursion, a function calls itself until some condition is satisfied.
5. In iteration, a function doesn't call to itself.
6. Problem could be defined in terms of its previous result to solve a problem using recursion.
7. It is not necessary to define a problem in terms of its previous result to solve using iteration.
8. All problems cannot be solved using recursion.
9. All problems can be solved using iteration.

| **Recursion** | **Iteration** |
|---|---|
| 1. A function is called from the definition of the same function to be repeated. | 1. Loop is used to do repeated tasks. |
| 2. Recursive is a top-down approach to problem solving | 2. Iteration is like a bottom-up approach. |
| 3. In recursion, a function calls itself until some condition is satisfied. | 3. In iteration, a function doesn't call to itself. |
| 4.Problem could be defined in terms of its previous result to solve a problem using recursion. | 4. It is not necessary to define a problem in term of its previous result to solve using iteration. |
| 5. All problems cannot be solved using recursion. | 5. All problems can be solved using iteration. |

# EXERCISES:

## Write algorithm, flowchart,source code and output of followings.

1. Create following functions:
    a. add() with return type as integers
    b. subtract() with return type as void
    c. multi( ) with return type as integer
    d. division( ) with return type as void
2. WAP that uses functions to calculate smallest among two numbers, where numbers are passed from the main() function.
3. WAP to create a user defined function to check a number prime or not.
4. WAP that uses a user defined function to calculate the HCF of two numbers. The program should calculate the LCM in the main () function.

5. Write a program to swap the content of two variables using functions.
6. WAP to create a user defined function which reverses a given number as a formal parameter. The function should return both the reversed number and sum of digits.(Hint. Use pass by reference method).
7. WAP to calculate the factorial of a given number by using a recursive function.
8. WAP to calculate the $n^{th}$ element of the Fib sequence by using a recursive function.
9. WAP to find the first 10 elements of fib sequence by using recursive function.
10. WAP that calculates the sum of digits of a given number by using a recursive function.
11. WAP to find the sum of the series $Sn = \sum 1/n^2$ using recursion, value of n is given by the user.
12. Write a program containing function power(a,b) to calculate the value of a raised to b using recursion.

# Assignments:

## Write algorithm, flowchart,source code and output of followings.

1. Write an interactive program that reads positive numbers until the user enters 0 and then sum the number divisible by 4 that lie between the ranges of 10 to 15, finally display that average of them into the main function.
2. WAP to read an integer number, count the even digit and odd digit present in the entered number. You must write a function for calculating the number, and the result must be displayed in the main function itself.
3. WAP to take a string from the user and pass it to a function. The function finds and returns the number of words in a string. In the main program, display the number of words.
4. WAP to raise the power of each element of the given matrix of order P*Q by 3. Display the resultant matrix. You have to use input, process and display functions.
5. WAP to swap the value of a pair of integers by using pass by value and pass by reference.
6. WAP that reads two numbers from the user. If both the numbers are even then find their HCF else find their LCM. The HCF and LCM should be found using two different functions. You can, however, call one of them from the other.
7. WAP that reads two different strings. Pass these to a function which reverse the second string and then appends it at the end of the first string, Print the new string from the function itself

# Lab-8 & 9: Pointers & Structures

## Objectives:

- To deal with structures its use and importance
- To deal with the pointers its use and importance

## Background Theory:

### Structure:

Structure is a collection of data items. The data item can be different types, some can be int, some can be float, some can be char and so on. The data item of the structure is called a member of the structure. In other words we can say that heterogeneous data types can be grouped to form a structure. In some languages structure is known as record. The difference between array and structure is the element of an array has the same type while the element of structure can be of different type. Another difference is that each element of an array is referred to by its position while each element of structure has a unique name.

**Defining a structure, arrays of structures, structures within structures:**

The general syntax for declaration of a structure is

```
struct name {
data_type1 member1;
data_type2 member2;
- - - - -
- - - - -
data_typen membern;
};
```

where struct is a keyword. The name is written by the programmer. The rule for writing names is the rule for identification.

```
 For example:
 struct student
{
char name[80];
int roll_no;
char branch[10];
int semester ;
};
```

## Creating structure variable:

In the above section, we have studied about how we declare structure but have not created the structure variable. Note that the space in the memory for a structure is created only when the structure variables are defined.

**Method of creating structure variable:**

I. creating structure variable at the time of declaration:

**Example:**

```
struct student
{
 char name[80];
int roll_no;
char branch[10];
int semester;
} var1, var2, var3;
```

In this example three structure variables named var1, var2, var3 are created. We can create one or more than one variable. The space is created like:

**II.**

```
 struct student { /* structure declaration */
char name[80] ;
int roll_no;
char branch[10];
int semester;
};
```

/* structure variable creation */

struct student var1, var2, var3 ;

Accessing member of structure:

The accessing concept of member is: structure_variable_name. member. The dot(.) operator is also known as a member operator or period. The dot operator has precedence and associativity is left to right.

**For example:**

++var1. mark is equivalent to ++(var1.marks)

Implies that the dot operator acts first and then unary operator

.For example:

struct bio

{ char name[80];

char address[80];

long phno;

};

struct bio b1, b2;

for accessing
b1.phno = 271280;

**Initialization of structure:**
A structure is initialized like other data types in C. The values to be initialized must appear in order as in the definition of structure within braces and separated by commas. C does not allow the initialization of individual structure members within its definition. Its syntax is
struct structure_name structure_variable = {value1, value2, value3 _ _ _ _ value..n}
**(1)**

```
 struct particular{
int, rn ;
int age ;
char sex;
};
```

We can initialize the structure at the time of variable creation like:

```
struct_particular per = {10, 22, 'm'};
```

By this 10 is assigned to m of per; 22 is assigned to age of per and m is assigned to per structure.
**(2)**

```
 struct bio
{
 int age;
int rn;
char sex;
int phno;
};
```

If we write the following statement:
struct bio b = {22, 10, 'm'}
In above example, C compiler assigns following value to each member:
age = 22 sex = m
rn = 10 phno = 0
So, the C compiler will automatically initialize zero to those members who are not initialized.
/* Create a structure named student that has name, roll, marks and remarks as members.Assume appropriate types and size of member. WAP using structure to read and display the data entered by the user */

```
# include <stdio.h>
main( )
{
struct student
{
char name[20];
int roll;
float marks;
char remark;
};
struct student s;
clrsc( );
printf("enter name: \t");
gets(s.name);
printf("In enter roll:\t");
scanf("%d", &s.roll);
printf("\n enter marks: \t");
scanf("%f", &s.marks);
printf("enter remarks p for pass or f for fail: \t")
s.remark = getch( )
printf("\n\n The student's information is \n");
printf("Student Name ItItIt Roll It Marks It Remarks");
printf("\n - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - -\n");
printf("%s\t\t%d\t%.2f\t%c", s.name, s.roll, s.marks, s.remark);
getch( );
}
```

**Output:**

enter name: Ram Singh
enter roll : 45
enter marks : 89
Enter remark p for pass or f for fail : P
The student's information is
Student Name Roll Marks Remarks
---
Ram Singh 45 89.00 P

## Array of Structure:

Like an array of int, float or char type, there may be an array of structure. In this case, the array will have individual structure as its elements.

**For example:**

```
struct employee
{
 char name[20];
int empid;
float salary;
} emp[10];
```

Where emp is an array of 10 employee structures. Each element of the array emp will contain the structure of the type employee. The another way to declare structure is

```
struct employee{
char name[20];
int empid;
float salary;
};
struct employee emp[10];
/* Create a structure named student that has name, roll, marks and remarks as members. Assume appropriate types and size of members. Use this structure to read and display records of 5 student */
main()
{
struct student
{
charname[30];
int roll;
float marks;
char remark;
};
struct student st[5];
int i;
clrscr( );
for(i=0; i<5; i++)
{
printf("\n Enter Information of student No%d\n"; i+1);
printf("Name: \t");
scanf("%s", s[i].name);
printf("\n Roll: \t");
```

```
scanf(“%d”, &s[i].roll);
printf(“\n Marks:\t”);
scanf(“%f”, &s[i].marks);
printf(“remark(p/f): \t”);
s[i].remark = getch( );
}
printf(“\n\n The Detail Information is \n”);
printf(“Student Name: \t Roll \t Marks \t Remarks”)
printf(“\n_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ \n”)
for(i=0; i<5; i++)
printf(“%s\t\t%d\t%.2f \t%c\n”.s[i].name, s[i].roll, s[i].marks, s[i].remark);
getch( );
}
```

output:

```
Enter Information of student No1
Name: Ram
Roll: 20
Marks: 89
Remarks: P
Enter Information of Student No2
Name: Shyam
Roll: 21
Marks: 46
Remarks: P
Enter Information of Student No 3
Name: Bikash
Roll: 22
Marks: 97
Remarks: P
Enter Information of Student No 4
Name: Jaya
Roll: 23
Marks: 87
Remarks: P
Enter Information of Student No 5
Name: Nisha
Roll: 24
Marks: 27
Remarks: F
The Detail Information is:
```

Student Name Roll Marks Remarks

---

Ram 20 89 P
Shyam 21 46 P
Bikash 12 97 P
Jaya 23 87 P
Nisha 24 27 F

**Initializing array of structure:**

Array structures can be initialized in the same way as a single structure.

For example:

```
Struct Book
{
 char name[20];
int pages;
float price;
};
Struct Book b[3] = {
"Programming In C", 200, 150.50,
"Let Us C", 455, 315,
"Programming with C", 558, 300.75}
```

**Structure within structure (Nested Structure):**

One structure can be nested within another structure in C. In other words, the individual members of a structure can be another structure as well.

**For example:**

/* Create a structure named date that has day, month and year as its members. Include this structure as a member in another structure named employee which has name, id, salary, as other members. Use this structure to read and display employee's name, id, dob and salary */

```
#include<stdio.h>
void main( )
{
struct date
{
int day;
int month;
int year;
};
struct employee
{
```

```
char name[20];
int id;
struct date dob;
float salary;
} emp;
printf("Name of Employee: \t");
scanf("%s", emp.name);
printf("\n ID of employee: \t");
scanf("%d", & emp.id);
printf("\n Day of Birthday: \t");
scanf("%d", &emp.dob.day);
printf("\n month of Birthday: \t");
scanf("%d", &emp.dob.month);
printf("\n Year of Birthday: \t");
scanf("%d", &emp.dob.year);
printf("salary of Employee: \t");
scanf("%d", &emp.salary);
printf("\n\n The Detail Information of Employee");
printf("\n Name \t id \t day \t month \t year \t salary");
printf("\n - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - n");
printf("%s\t%d\t%d\t%d\t%d\t%.2f",emp.name,emp.id,emp.dob.day,
emp.dob.month,
emp.dob.year, emp.salary);
}
```

**output:**

Name of Employee : Teena
Id of Employee : 2001
Day of Birthday : 10
Month of Birthday : 10
Year of Birthday : 1987
Salary of Employee : 12000
The Detail Information of employee:
Name ID Day Month Year Salary
---
Teena 2001 10 10 1987 12000.00

**Processing a Structure:**

**(PU2008)**

/* WAP to read records of 5 employee(Enter relevant fields: Name, address, salary, Id) Display the records of the top three employees which have the highest salary */

```
void main( )
{
struct employee
{
 char name[20];
char Address[20];
float salary;
int ID;
};
int, i, j ;
struct employee temp;
struct employee emp[5];
clrscr( );
 printf("enter employee Information: ");
for(i=0;i<5;i++)
{
printf("Name of Employee: \t");
scanf("%s", emp[i].name);
printf("\n ID of employee: \t");
scanf("%d", & emp[i].ID);
printf("salary of Employee: \t");
scanf("%d", &emp[i].salary);
}
for(i=0; i<4; i++)
{
for(j=i+1; j<5; j++)
{
if(emp[i].salary<emp[j].salary)
{
temp = emp[i];
 emp[i] = emp[j].;
 emp[j]= temp;
}
          }
}
printf("Information of 3 employees having highest salary: In");
printf("\nName\t\t\tAddress\t\tSalary\t\D\t\n");
```

```
printf("\n - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - \n');
for(i=0, i<3; i++)
{
printf("%s\t\t%s\t\t%.2f\t%d",emp[i].Name,emp[i].Address,emp[i].
Salary, emp[i].ID)
}
getch( )
}
```

**PU2007**

/* Create a user defined array structure student record having members physics, chemistry and mathematics. Feed the marks obtained by three students in each subjects and calculate the total marks of each student */

```
void main( )
{ struct student
{int physics;
int chemistry;
int mathematics;
int total;
};
struct student std[3]; int i;
printf("Enter the marks in physics, chemistry and mathematics of
3 student: ");
for(i=0; i<3; i++)
{ printf("Marks of students %d", i+1);
scanf("%d%d%d",         std[i].physics,         std[i].chemistry,
std[i].mathematics);
std[i].total    =    std[i].physics    +    std[i].chemistry    +
std[i].mathematics;
}
printf("Information of students: ");
printf(\n - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - \n");
printf("student\t\ttotal marks \t");
printf("\n - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - \n");
for9i=0; i<3; i++)
{
printf("\nStudent%d\t\t%d\n", i+1, s+d[i].total);
}
```

Structure declaration:
Syntax:

```
struct structure_tag
     {
       Member1;
       Member2;
     ……..
     ……
};
```

Structure variables:
Structure variable initialization:

```
struct
structure_tag
     {
       Member1;
       Member2;
     ……..
     ……
}var1,  var2  =
{val1, val2,…};
```

## Accessing structure members:

**Syntax:**

structure_variable.member

Creation and initialization of structure

```
     #include<stdio.h>
     #include<conio.h>
     struct  student
     {         char name[20];
               int rollno;
               int grade;
     };
     void main()
     {    int i;
          struct student s1={"mm",5,6};
          struct student s2;

          printf("Enter the students information \n");
     printf("Name:-");
     scanf("%s", s2.name);
     printf("Roll No:-");
```

```
scanf("%d",&s2.rollno);
printf("grade:-");
scanf("%d",&s2.grade);
     printf("\n Name  Roll No  Grade\n");
     printf("----------------------");
     printf("\n       %s                %d                %d
\n",s1.name,s1.rollno,s1.grade);
     printf("\n       %s                %d                %d
\n",s2.name,s2.rollno,s2.grade);
     getch();
}
```

## Pointers:

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is −

type *var-name;

Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations −

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

**How to Use Pointers?**

There are a few important operations, which we will do with the help of pointers very frequently.

(a) We define a pointer variable,

(b) assign the address of a variable to a pointer and

(c) finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. The following example makes use of these operations −

```
#include <stdio.h>
int main ()
{
   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */
   ip = &var;  /* store address of var in pointer variable*/
   printf("Address of var variable: %x\n", &var  );
   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );
   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );
   return 0;
}
```

When the above code is compiled and executed, it produces the following result −
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20

**NULL Pointers**
It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer.

The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program −

```
#include <stdio.h>
int main () {
  int  *ptr = NULL;
  printf("The value of ptr is : %x\n", ptr  );
  return 0;
}
```

When the above code is compiled and executed, it produces the following result −
The value of ptr is 0
In most of the operating systems, programs are not permitted to access memory at address 0 because that memory is reserved by the operating system. However, the memory address 0 has special significance; it signals that the pointer is not intended to point to an accessible memory location. But by convention, if a pointer contains the null (zero) value, it is assumed to point to nothing.
To check for a null pointer, you can use an 'if' statement as follows −

```
if(ptr)     /* succeeds if p is not null */
if(!ptr)    /* succeeds if p is null */
```

strings are actually a one-dimensional array of characters terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization then you can write the above statement as follows −

```
char greeting[] = "Hello";
```

Following is the memory presentation of the above defined string in C/C++ −

Actually, you do not place the null character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print the above mentioned string –

```
#include <stdio.h>
int main ()
{
  char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
  printf("Greeting message: %s\n", greeting );
  return 0;
}
```

When the above code is compiled and executed, it produces the following result −

*Greeting message: Hello*

C supports a wide range of functions that manipulate null-terminated strings −

| **S. N.** | **Function & Purpose** |
|---|---|
| 1 | strcpy(s1, s2);<br>Copies string s2 into string s1. |
| 2 | strcat(s1, s2);<br>Concatenates string s2 onto the end of string s1. |
| 3 | strlen(s1);<br>Returns the length of string s1. |
| 4 | strcmp(s1, s2);<br>Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2. |
| 5 | strchr(s1, ch);<br>Returns a pointer to the first occurrence of character ch in string s1. |

| | |
|---|---|
| 6 | strstr(s1, s2);<br>Returns a pointer to the first occurrence of string s2 in string s1. |

**Array and Pointers**

Array name by itself is an address or pointer. It points to the address of the first element( $0^{th}$ element of the array)

**One dimensional Array and Pointer Notation**

Declaration:

int a[5] equivalent to int *a;

Address

Values

X-> &X[0] -> x+0

X+1 -> &X[1]

X+5 -> &X[5]

*X -> X[0]

*(X+1) -> X[1]

*(X+5) -> x[5]

**Two dimensional Array**

Declaration:

int a[][20] equivalent to (*a)[20];

Address

Values

& a[i][j] -> *(a+i) +j

a[i][j] -> *(*(a+i)+ j)

## EXERCISES:

1. Create a structure Student containing name as char string, roll as integer and address as character string. Read name, roll and address for one person from the user and display the contents in following format:

   NAME: name

   ADDRESS: address

   ROLL NO: roll
2. Modify question 1 to input and display name, roll and address for N number of students, where N is any number input by the user. Input everything from main( ) and display the contents from display( ) function.
3. Create a structure Date containing three members: int dd, int mm and int yy. Create another structure Person containing four members: name, address, telephone and date of birth. For member date of birth, create an object of structure Date within Person. Using

these structures, write a program to input the records. Then, display the contents in tabular form.

4. Create a structure Employee containing name as character string, telephone as character string and salary as integer. Input records of 10 employees. Display the name, telephone and salary of the employee with highest salary, lowest salary. Also display the average salary of all 10 employees.
5. WAP to read a ‘n’ elements array to sort them in descending order. Display the second highest number using pointers.
6. WAP to read two 3×2 matrices. Add them and display the sum of two matrices using pointers.
7. WAP is a program that reads a string, passes it to the function and counts the total number of vowels present in it using pointers.
8. WAP that calls a concatenate () function. Pass two strings to the function. The function should join the two strings. The concatenated string should be displayed in the main() function using pointers.
9. WAP to enter two matrices of sizes m×n and size p×q. Multiply the two matrices and store in the third matrix if possible (using pointers).
10. WAP to store the name and age of 5 students. Use the concept of structure and pointer to structure concept to access the members of the structures.

# Lab-10: File Handling

## Objectives:

- To deal with file handling its use and importance
- To open, read , write and close files.

## Background Theory:

### File handling:

File handling is the process of creating, opening, reading, writing, and closing files in C. Files are used to store data permanently on the computer's hard disk. This allows us to access and use the data even after the program that created it has terminated.

### Why are files needed?

- When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
- If you have to enter a large amount of data, it will take a lot of time to enter them all. However, if you have a file containing all the data, you can easily access the contents of the file using a few commands in C.
- You can easily move your data from one computer to another without any changes.

### Types of Files

When dealing with files, there are two types of files you should know about:

1. Text files
2. Binary files

**1. Text files**

Text files are the normal .txt files. You can easily create text files using any simple text editors such as Notepad.

When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.

They take minimum effort to maintain, are easily readable, provide the least security and take bigger storage space.

**2. Binary files**

Binary files are mostly the .bin files in your computer.

Instead of storing data in plain text, they store it in the binary form (0's and 1's).

They can hold a higher amount of data, are not readable easily, and provide better security than text files.

## C File Operations

C file operations refer to the different possible operations that we can perform on a file in C such as:

1. Creating a new file – fopen() with attributes as “a” or “a+” or “w” or “w+”
2. Opening an existing file – fopen()
3. Reading from file – fscanf() or fgets()
4. Writing to a file – fprintf() or fputs()
5. Moving to a specific location in a file – fseek(), rewind()
6. Closing a file – fclose()

## File Pointer in C

A file pointer is a reference to a particular position in the opened file. It is used in file handling to perform all file operations such as read, write, close, etc. We use the FILE macro to declare the file pointer variable. The FILE macro is defined inside the <stdio.h> header file.

**Syntax of File Pointer**

FILE* pointer_name;

File Pointer is used in almost all the file operations in C.

## Open a File in C

For opening a file in C, the fOpen() function is used with the filename or file path along with the required access modes.

**Syntax of fopen()**

FILE* fopen(const char *file_name, const char *access_mode);

**Parameters**

- file_name: name of the file when present in the same directory as the source file. Otherwise, full path.
- access_mode: Specifies for what operation the file is being opened.

**Return Value**

- If the file is opened successfully, it returns a file pointer to it.
- If the file is not opened, then returns NULL.

**File opening modes in C**

File opening modes or access modes specify the allowed operations on the file to be opened. They are passed as an argument to the fopen() function. Some of the commonly used file access modes are listed below:

| Opening Modes | Description |
|---|---|
| r | Searches file. If the file is opened successfully, fopen( ) loads it into memory and sets up a pointer that points to the first character in it. If the file cannot be opened, fopen( ) returns NULL. |
| rb | Open for reading in binary mode. If the file does not exist, fopen( ) returns NULL. |
| w | Open for reading in text mode. If the file exists, its contents are overwritten. If the file doesn't exist, a new file is created. Returns NULL, if unable to open the file. |
| wb | Open for writing in binary mode. If the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| a | Searches file. If the file is opened successfully, fopen( ) loads it into memory and sets up a pointer that points to the last character in it. It opens only in the append mode. If the file doesn't exist, a new file is created. Returns NULL, if unable to open the file. |
| ab | Open for append in binary mode. Data is added to the end of the file. If the file does not exist, it will be created. |
| r+ | Searches file. It is opened successfully and fopen( ) loads it into memory and sets up a pointer that points to the first character in it. Returns NULL, if unable to open the file. |
| rb+ | Open for both reading and writing in binary mode. If the file does not exist, fopen( ) returns NULL. |

| | |
|---|---|
| w+ | Searches file. If the file exists, its contents are overwritten. If the file doesn't exist a new file is created. Returns NULL, if unable to open the file. |
| wb+ | Open for both reading and writing in binary mode. If the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| a+ | Searches file. If the file is opened successfully, fopen( ) loads it into memory and sets up a pointer that points to the last character in it. It opens the file in both reading and append mode. If the file doesn't exist, a new file is created. Returns NULL, if unable to open the file. |
| ab+ | Open for both reading and appending in binary mode. If the file does not exist, it will be created. |

As given above, if you want to perform operations on a binary file, then you have to append 'b' at the end. For example, instead of "w", you have to use "wb", instead of "a+" you have to use "a+b".

**Example of Opening a File**

```
// C Program to illustrate file opening
#include <stdio.h>
#include <stdlib.h>

int main()
{
        // file pointer variable to store the value returned by
        // fopen
        FILE* fptr;

        // opening the file in read mode
        fptr = fopen("filename.txt", "r");

        // checking if the file is opened successfully
        if (fptr == NULL) {
                printf("The file is not opened. The program will "
                        "now exit.");
                exit(0);
```

```
    }

    return 0;
}
```

**Output**

```
The file is not opened. The program will now exit.
```

The file is not opened because it does not exist in the source directory. But the fopen() function is also capable of creating a file if it does not exist. It is shown below.

## Create a File in C

The fopen() function can not only open a file but also can create a file if it does not exist already. For that, we have to use the modes that allow the creation of a file if not found such as w, w+, wb, wb+, a, a+, ab, and ab+.

```
FILE *fptr;
fptr = fopen("filename.txt", "w");
```

Program to create a file.

```
// C Program to create a file
#include <stdio.h>
#include <stdlib.h>

int main()
{
    // file pointer
    FILE* fptr;

    // creating file using fopen() access mode "w"
    fptr = fopen("file.txt", "w");

    // checking if the file is created
    if (fptr == NULL) {
        printf("The file is not opened. The program will "
               "exit now");
        exit(0);
    }
    else {
```

```
            printf("The file is created Successfully.");
        }

        return 0;
}
```

## Reading From a File

The file read operation in C can be performed using functions fscanf() or fgets(). Both the functions performed the same operations as that of scanf and gets but with an additional parameter, the file pointer. There are also other functions we can use to read from a file. Such functions are listed below:

| Function | Description |
|---|---|
| fscanf() | Use formatted string and variable arguments list to take input from a file. |
| fgets() | Input the whole line from the file. |
| fgetc() | Reads a single character from the file. |
| fgetw() | Read a number from a file. |
| fread() | Reads the specified bytes of data from a binary file. |

So, it depends on you if you want to read the file line by line or character by character.

**Example:**

```
FILE * fptr;
fptr = fopen(“fileName.txt”, “r”);
fscanf(fptr, "%s %s %s %d", str1, str2, str3, &year);
char c = fgetc(fptr);
```

The getc() and some other file reading functions return EOF (End Of File) when they reach the end of the file while reading. EOF indicates the end of the file and its value is implementation-defined.

Note: One thing to note here is that after reading a particular part of the file, the file pointer will be automatically moved to the end of the last read character.

## Write to a File

The file write operations can be performed by the functions fprintf() and fputs() with similarities to read operations. C programming also provides some other functions that can be used to write data to a file such as:

| Function | Description |
|---|---|
| fprintf() | Similar to printf(), this function uses formatted string and variable arguments list to print output to the file. |
| fputs() | Prints the whole line in the file and a newline at the end. |
| fputc() | Prints a single character into the file. |
| fputw() | Prints a number to the file. |
| fwrite() | These functions write the specified amount of bytes to the binary file. |

**Example:**

```
FILE *fptr ;
fptr = fopen(“fileName.txt”, “w”);
fprintf(fptr, "%s %s %s %d", "We", "are", "in", 2012);
fputc("a", fptr);
```

## Closing a File

The fclose() function is used to close the file. After successful file operations, you must always close a file to remove it from the memory.

**Syntax of fclose()**

```
fclose(file_pointer);
```

where the file_pointer is the pointer to the opened file.

**Example:**

```
FILE *fptr ;
fptr= fopen(“fileName.txt”, “w”);
---------- Some file Operations -------
fclose(fptr);
```

**Example 1: Program to Create a File, Write in it, And Close the File**

```
// C program to Open a File,
// Write in it, And Close the File
#include <stdio.h>
#include <string.h>

int main()
{
     // Declare the file pointer
     FILE* filePointer;

     // Get the data to be written in file
     char dataToBeWritten[50] = "GeeksforGeeks-A Computer "
                                "Science Portal for Geeks";

     // Open the existing file GfgTest.c using fopen()
     // in write mode using "w" attribute
     filePointer = fopen("GfgTest.c", "w");

     // Check if this filePointer is null
     // which maybe if the file does not exist
     if (filePointer == NULL) {
          printf("GfgTest.c file failed to open.");
     }
     else {

          printf("The file is now opened.\n");

          // Write the dataToBeWritten into the file
          if (strlen(dataToBeWritten) > 0) {

               // writing in the file using fputs()
               fputs(dataToBeWritten, filePointer);
               fputs("\n", filePointer);
          }

          // Closing the file using fclose()
          fclose(filePointer);

          printf("Data successfully written in file "
               "GfgTest.c\n");
```

```
        printf("The file is now closed.");
    }

    return 0;
}
```

**Output**

*The file is now opened.*
*Data successfully written in file GfgTest.c*
*The file is now closed.*

This program will create a file named GfgTest.c in the same directory as the source file which will contain the following text: “GeeksforGeeks-A Computer Science Portal for Geeks”.

**Example 2: Program to Open a File, Read from it, And Close the File**

// C program to Open a File,
// Read from it, And Close the File

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

int main()
{

    // Declare the file pointer
    FILE* filePointer;

    // Declare the variable for the data to be read from
    // file
    char dataToBeRead[50];

    // Open the existing file GfgTest.c using fopen()
    // in read mode using "r" attribute
    filePointer = fopen("GfgTest.c", "r");

    // Check if this filePointer is null
    // which maybe if the file does not exist
    if (filePointer == NULL) {
        printf("GfgTest.c file failed to open.");
    }
    else {
```

```
            printf("The file is now opened.\n");

            // Read the dataToBeRead from the file
            // using fgets() method
            while (fgets(dataToBeRead, 50, filePointer)
                 != NULL) {

                 // Print the dataToBeRead
                 printf("%s", dataToBeRead);
            }

            // Closing the file using fclose()
            fclose(filePointer);

            printf(
                 "Data successfully read from file GfgTest.c\n");
            printf("The file is now closed.");
       }
       return 0;
}
```

**Output**

```
The file is now opened.
GeeksforGeeks-A Computer Science Portal for Geeks
Data successfully read from file GfgTest.c
The file is now closed.
```

This program reads the text from the file named GfgTest.c which we created in the previous example and prints it in the console.

## Read and Write in a Binary File

Till now, we have only discussed text file operations. The operations on a binary file are similar to text file operations with little difference.

**Opening a Binary File**

To open a file in binary mode, we use the rb, rb+, ab, ab+, wb, and wb+ access mode in the fopen() function. We also use the .bin file extension in the binary filename.

Example
fptr = fopen("filename.bin", "rb");

**Write to a Binary File**
We use fwrite() function to write data to a binary file. The data is written to the binary file in the form of bits (0's and 1's).
Syntax of fwrite()
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *file_pointer);
Parameters:

- ptr: pointer to the block of memory to be written.
- size: size of each element to be written (in bytes).
- nmemb: number of elements.
- file_pointer: FILE pointer to the output file stream.

Return Value:

- Number of objects written.

## Example: Program to write to a Binary file using fwrite()

```
// C program to write to a Binary file using fwrite()
#include <stdio.h>
#include <stdlib.h>
struct threeNum {
     int n1, n2, n3;
};
int main()
{
     int n;
     // Structure variable declared here.
     struct threeNum num;
     FILE* fptr;
     if ((fptr = fopen("C:\\program.bin", "wb")) == NULL) {
          printf("Error! opening file");
          // If file pointer will return NULL
          // Program will exit.
          exit(1);
     }
     int flag = 0;
     // else it will return a pointer to the file.
     for (n = 1; n < 5; ++n) {
          num.n1 = n;
          num.n2 = 5 * n;
          num.n3 = 5 * n + 1;
```

```
        flag = fwrite(&num, sizeof(struct threeNum), 1,
                      fptr);
    }

    // checking if the data is written
    if (!flag) {
        printf("Write Operation Failure");
    }
    else {
        printf("Write Operation Successful");
    }

    fclose(fptr);

    return 0;
}
```

**Output**

*Write Operation Successful*

**Reading from Binary File**

The fread() function can be used to read data from a binary file in C. The data is read from the file in the same form as it is stored i.e. binary form.

Syntax of fread()

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *file_pointer);

Parameters:

- ptr: pointer to the block of memory to read.
- size: the size of each element to read(in bytes).
- nmemb: number of elements.
- file_pointer: FILE pointer to the input file stream.

Return Value:

- Number of objects written.

## Example: Program to Read from a binary file using fread()

```
// C Program to Read from a binary file using fread()
#include <stdio.h>
#include <stdlib.h>
struct threeNum {
    int n1, n2, n3;
};
int main()
```

```
{
      int n;
      struct threeNum num;
      FILE* fptr;
      if ((fptr = fopen("C:\\program.bin", "rb")) == NULL) {
            printf("Error! opening file");
            // If file pointer will return NULL
            // Program will exit.
            exit(1);
      }
      // else it will return a pointer to the file.
      for (n = 1; n < 5; ++n) {
            fread(&num, sizeof(struct threeNum), 1, fptr);
            printf("n1: %d\tn2: %d\tn3: %d\n", num.n1, num.n2,
                  num.n3);
      }
      fclose(fptr);

      return 0;
}
```

**Output**

```
n1: 1   n2: 5   n3: 6
n1: 2   n2: 10  n3: 11
n1: 3   n2: 15  n3: 16
n1: 4   n2: 20  n3: 21
```

## fseek() in C

If we have multiple records inside a file and need to access a particular record that is at a specific position, we need to loop through all the records before it to get the record. Doing this will waste a lot of memory and operational time. To reduce memory consumption and operational time we can use fseek() which provides an easier way to get to the required data. fseek() function in C seeks the cursor to the given record in the file.

**Syntax for fseek()**

int fseek(FILE *ptr, long int offset, int pos);

**Example of fseek()**

// C Program to demonstrate the use of fseek() in C

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

```
int main()
{
     FILE* fp;
     fp = fopen("test.txt", "r");

     // Moving pointer to end
     fseek(fp, 0, SEEK_END);

     // Printing position of pointer
     printf("%ld", ftell(fp));

     return 0;
}
```

**Output**

*81*

## rewind() in C

The rewind() function is used to bring the file pointer to the beginning of the file. It can be used in place of fseek() when you want the file pointer at the start.

**Syntax of rewind()**

rewind (file_pointer);

**Example**

// C program to illustrate the use of rewind

```
#include <stdio.h>

int main()
{
     FILE* fptr;
     fptr = fopen("file.txt", "w+");
     fprintf(fptr, "Geeks for Geeks\n");

     // using rewind()
```

```
    rewind(fptr);

    // reading from file
    char buf[50];
    fscanf(fptr, "%[^\n]s", buf);

    printf("%s", buf);

    return 0;
}
```

**Output**

*Geeks for Geeks*

## More Functions for C File Operations

The following table lists some more functions that can be used to perform file operations or assist in performing them.

| Functions | Description |
|---|---|
| fopen() | It is used to create a file or to open a file. |
| fclose() | It is used to close a file. |
| fgets() | It is used to read a file. |
| fprintf() | It is used to write blocks of data into a file. |
| fscanf() | It is used to read blocks of data from a file. |
| getc() | It is used to read a single character to a file. |
| putc() | It is used to write a single character to a file. |
| fseek() | It is used to set the position of a file pointer to a mentioned location. |

| | |
|---|---|
| ftell() | It is used to return the current position of a file pointer. |
| rewind() | It is used to set the file pointer to the beginning of a file. |
| putw() | It is used to write an integer to a file. |
| getw() | It is used to read an integer from a file. |

## Exercise:

1. Write a program to create a new file and write a string to it.
2. Write a program to copy the contents of one file to another file.
3. Write a program to read a line of text from a file and print it to the console.
4. Write a program to input a string and write it to a file named string1.txt. read the information from the file and count the total no words present in it and display it.
5. Create a structure Employee with members name, id and salary as integer. Read the data for 5 employees and write it to a file named emp.txt. Sort the information according to the salary in ascending order and write the sorted information to the file named sort.txt.
6. Create a structure named student with its members as name, roll and age. Read the data from the user until the user enters ‘n’ or ‘N’. Write the information in the file named “student.dat”. Input a name from the user, search it in the file and display all information of that student if found.