

About the Book: is a fundamental and general purpose language. The knowledge of C programming language is essential to learn the advanced programming languages like C++, Java and C#. The book has written like a class lecture notes, in such a way that it is useful for both slow learner and fast learner using simple English language with numerous solved programming examples and exercises. The book has been written to meet the requirement of undergraduate university exams. The topics included are: Introduction to C, Branching and Looping, Arrays, Strings, Functions, Structures, Files, Pointers, Dynamic Memory Allocation and Introduction to Data Structures like Stacks, Queues, Linked List, Trees and Abstract Data Type. The book also contains good number of programming examples, lab manual, viva questions, question bank for practice, old question papers and model question papers. The book also contains model questions for placement interviews.

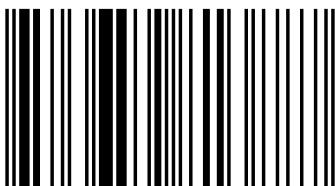
Programming in C (Revised Edition)



Thyagaraju Gowda



Dr. Thyagaraju GS Gowda, currently working as a Professor and HOD in the Department of Computer Science Engineering, Shri Dharmasthala Manjunatheshwara Institute of Technology, Ujire, Karnataka, India-574240 has more than 16 years of teaching and research experience. He has a passion of writing books on computing technology and philosophy.



978-3-659-81388-7

Gowda

# Programming in C ( Revised )

Second Edition

LAP  
LAMBERT  
Academic Publishing

**Thyagaraju Gowda**

**Programming in C ( Revised )**



**Thyagaraju Gowda**

**Programming in C ( Revised )**

**Second Edition**

**LAP LAMBERT Academic Publishing**

## **Impressum / Imprint**

Bibliografische Information der Deutschen Nationalbibliothek: Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Alle in diesem Buch genannten Marken und Produktnamen unterliegen warenzeichen-, marken- oder patentrechtlichem Schutz bzw. sind Warenzeichen oder eingetragene Warenzeichen der jeweiligen Inhaber. Die Wiedergabe von Marken, Produktnamen, Gebrauchsnamen, Handelsnamen, Warenbezeichnungen u.s.w. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutzgesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Bibliographic information published by the Deutsche Nationalbibliothek: The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <http://dnb.d-nb.de>.

Any brand names and product names mentioned in this book are subject to trademark, brand or patent protection and are trademarks or registered trademarks of their respective holders. The use of brand names, product names, common names, trade names, product descriptions etc. even without a particular marking in this work is in no way to be construed to mean that such names may be regarded as unrestricted in respect of trademark and brand protection legislation and could thus be used by anyone.

Coverbild / Cover image: [www.ingimage.com](http://www.ingimage.com)

Verlag / Publisher:

LAP LAMBERT Academic Publishing  
ist ein Imprint der / is a trademark of  
OmniScriptum GmbH & Co. KG  
Bahnhofstraße 28, 66111 Saarbrücken, Deutschland / Germany  
Email: [info@lap-publishing.com](mailto:info@lap-publishing.com)

Herstellung: siehe letzte Seite /

Printed at: see last page

**ISBN: 978-3-659-81388-7**

Copyright © 2016 OmniScriptum GmbH & Co. KG

Alle Rechte vorbehalten. / All rights reserved. Saarbrücken 2016

# **Programming in C**

**(Revised 2<sup>nd</sup> Edition)**

**Dr. Thyagaraju Gowda, M.Sc,M.Tech,Ph.D,LMIETE**

Department of Computer Science Engineering,  
Shri Dharamasthala Manjunatheshwara Institute Of Technology,  
Ujire -574240, Belthangady Taluk ,Dakshina Kannada,Karnataka ,India

**Dedicated to**

*My Daughter*

**Palguni GT**

## Preface to the Second Edition

I am extremely happy to place in the hands of our esteem readers the second edition of my text book on ***Programming in C***. Programming in C plays a most fundamental role to all branches of engineering and particularly in the field of Computer and Information Science Engineering. In almost all the placement tests of IT companies, the basic knowledge of C program is expected from any student. It has become indispensable to all engineering students to master the concepts of C program and data structures.

This book is written by aiming the first year students of Engineering / any undergraduate students and for placement interview. The book is presented in such a way that any student with no prior knowledge of programming can follow and understand the concepts easily.

The book is organized into Five Modules. In Module1, introduction to C language is presented. In this chapter the fundamental concepts like algorithm, flowchart, pseudo code and basic concepts of C program is discussed along with programming examples. In Module2, different branching and looping statements like if, if else, nested if else, cascaded if else, switch statement, ternary operators, go to statement, for loop, while loop, do while loop, break statement and continue statement is discussed with programming examples. In Module3 the concepts like functions, types of Argument passing, Arrays, Types of Arrays, and Strings are presented. In Module4 the concepts like basics of structures and file management is presented. Finally in Module5 the concepts like Pointers, Preprocessors and Introduction to Data structures like Primitive Data types, Stacks, Queues, Linked Lists and Trees are discussed with programming examples. The book also includes model questions for placement interview.

**Dr.Thyagaraju Gowda**

# Acknowledgements

It is my proud privilege to express my sincere acknowledgement to our beloved President **Dr. D.Veerendra Heggde** and Secretary **Prof. Dr.B.Yashoverma** for their consistent motivation, support and proper guidance to write this book.

My sincere thanks to our beloved Principal **Dr.Suresh.K** for his consistent motivation and guidance. My sincere thanks to all teaching and non-teaching staff of CSE Department, Deans,HOD,administrative staff and students of SDMIT- Ujire and my friends for motivation and sincere encouragement they provided, without which, I could not have progressed this work.

I would like to thank all my first year students of (*specially Anudeep M Rao and Annie Varshitha*) SDMIT who gave a feedback and suggestions about the way of presenting the contents of the book.

Also I take this opportunity to thank *my daughter Palguni GT , my wife , my mother and my friends for their* invaluable moral support, wishes and blessings to write this book. **I also thankful to Dr.Latha for reviewing and coauthoring (section like Computer Programming Lab Manual) .**

**Dr.Thyagaraju Gowda**

# Syllabus

## MODULE 1

**INTRODUCTION TO C LANGUAGE:** Pseudo code solution to problem, Basic concepts of a C program, Declaration, Assignment & Print statement, Data Types, Operators and expressions, Programming examples and exercises.

## MODULE 2

**BRANCHING AND LOOPING:** Two way selection (if, if-else, nested if-else, cascaded if-else), switch statement, ternary operator? goto, Loops (For, do-while, while) in C, break and continue, Programming examples and exercises.

## MODULE 3

### ARRAYS, STRINGS AND FUNCTIONS:

**ARRAYS AND STRINGS:** Using an array, Using arrays with Functions, Multi-Dimensional arrays. String: Declaring, Initializing, Printing and reading strings, Strings manipulation functions, Strings input and output functions, Arrays of strings, Programming examples and exercises.

**FUNCTIONS:** Functions in C, Argument Passing – call by value, call by reference, Functions and program structure, Location of functions, void and parameter less Functions, Recursion, Programming examples and exercises.

## MODULE 4

**STRUCTURES AND FILE MANAGEMENT:** Basics of structures, Structures and Functions, Array of structures, Structure Data types, Type definition, Defining, opening and closing of files, Input and output operations, Programming examples and exercises.

## **MODULE 5**

**POINTERS AND PREPROCESSORS:** Pointers and address, Pointers and functions (call by references) arguments, Pointers and arrays, Address arithmetic, Character pointer and functions, Pointers to pointer ,Initialization of pointers arrays, Dynamic memory allocations methods, Introduction to Preprocessors, Complier control Directives, Programming examples and exercises.

**INTRODUCTION TO DATA STRUCTURES:** Primitive and non-primitive data types, Abstract Data types, Definition and applications of Stacks, Queues, Linked Lists and Trees.

# **Contents**

<b>Preface to the Second Edition</b>	3
<b>Acknowledgements</b>	4
<b>Syllabus</b>	5
<b>Contents</b>	7

<b>Module 1: Introduction to C Language</b>	<b>11</b>
---	-----------

<b>1.1</b>	Introduction -----	11
<b>1.2</b>	Algorithm and Flowchart-----	12
<b>1.3</b>	Pseudocode Solution to Problem-----	20
<b>1.4</b>	Basic Concepts of a C program-----	24
<b>1.5</b>	Alphabet ,Tokens, Data Types, Variables and Constants-----	33
<b>1.6</b>	Declaration,Assignment and Print statement-----	43
<b>1.7</b>	Operators and Expressions-----	55
<b>1.8</b>	Programming Examples-----	96
<b>1.9</b>	Exercises-----	115

<b>Module 2: Branching and Looping</b>	<b>126</b>
--	------------

<b>2.1</b>	Introduction-----	126
<b>2.2</b>	Selection or Branching Statements-----	127
<b>2.2.1</b>	Conditional Branching Statements -----	127
<b>2.2.1.1</b>	Two way selection statements -----	127
<b>2.2.1.2</b>	Types of Two Way Selection statements-----	128
<b>2.2.1.3</b>	Switch statement -----	134
<b>2.2.1.4</b>	Ternary operator (? : )-----	137
<b>2.2.2</b>	Unconditional Branching Statements -----	139

<b>2.3</b>	<b>Looping Statements-----</b>	<b>142</b>
------------	--------------------------------	------------

<b>2.3.1</b>	<b>Types of Looping Statements</b>	<b>143</b>
<b>2.3.2</b>	Types of Loops supported in C (While , do While and for)-----	144
<b>2.3.3</b>	Jumps in Loops ( break and continue)-----	155
<b>2.4</b>	Programming Examples -----	161
<b>2.5</b>	Exercises-----	197

**Module 3: Arrays, Strings and Functions** 201

<b>3.1 Arrays -----</b>	<b>201</b>
<b>3.1.1 Introduction to Arrays-----</b>	<b>201</b>
<b>3.1.2 Using an Array -----</b>	<b>201</b>
<b>3.1.3 Single Dimensional Arrays-----</b>	<b>203</b>
<b>3.1.4 Arrays of Characters-----</b>	<b>205</b>
<b>3.1.5 Expressions as subscript-----</b>	<b>206</b>
<b>3.1.6 Parallel Arrays-----</b>	<b>206</b>
<b>3.1.7 Using the constant as the array size-----</b>	<b>206</b>
<b>3.1.8 Searching and Sorting-----</b>	<b>207</b>
<b>3.1.9 Multidimensional Arrays -----</b>	<b>214</b>
<b>3.2 Strings -----</b>	<b>219</b>
<b>3.2.1 Introduction-----</b>	<b>219</b>
<b>3.2.2 Declaring , Initializing , Printing and Reading Strings-----</b>	<b>219</b>
<b>3.2.3 Strings Manipulation Functions from Standard library-----</b>	<b>222</b>
<b>3.2.4 String input/output functions : gets and puts-----</b>	<b>230</b>
<b>3.2.5 Arrays of Strings-----</b>	<b>232</b>
<b>3.3 Functions -----</b>	<b>233</b>
<b>3.3.1 Introduction-to Functions in C-----</b>	<b>233</b>
<b>3.3.2 Elements of User defined functions -----</b>	<b>236</b>
<b>3.3.3 Functions and Program Structure-----</b>	<b>241</b>
<b>3.3.4 Location of functions-----</b>	<b>243</b>
<b>3.3.5 Categories of User defined functions-----</b>	<b>246</b>
<b>3.3.6 Argument Passing – Call by value -----</b>	<b>249</b>
<b>3.3.7 Recursion -----</b>	<b>252</b>
<b>3.3.8 Using arrays with functions -----</b>	<b>254</b>
<b>3.4 Programming Examples -----</b>	<b>274</b>
<b>3.5 Exercises-----</b>	<b>329</b>

**Module 4: Structures and File Management** 334

<b>4.1 Structures -----</b>	<b>334</b>
<b>4.1.1 Basics of Structures-----</b>	<b>334</b>
<b>4.1.2 Nested Structures-----</b>	<b>339</b>
<b>4.1.3 Arrays of Structures -----</b>	<b>341</b>

4.1.4	Structures and Function -----	342
4.1.5	Type Definition -----	345
4.1.6	Structure data type using <i>typedef</i> -----	347
<b>4.2</b>	<b>File Management -----</b>	<b>349</b>
4.2.1	Introduction -----	349
4.2.2	Files :Defining , Opening and Closing of Files-----	350
4.2.3	Input and Output Operations-----	353
4.3	Programming Examples -----	366
4.4	Exercises-----	384

## **Module 5:Pointers, Preprocessors and Introduction to Data Structures 386**

<b>5.1</b>	<b>Pointers</b>	<b>386</b>
5.1.1	Pointers and address -----	386
5.1.2	Pointers and functions arguments-----	393
5.1.3	Pointers and arrays-----	394
5.1.4	Pointers (address) arithmetic-----	398
5.1.5	Character pointer -----	400
5.1.6	Pointers to Pointer -----	401
5.1.7	Dynamic memory allocation -----	402
<b>5.2</b>	<b>Preprocessors</b>	<b>405</b>
5.2.1	Introduction -----	405
5.2.2	Preprocessor directives -----	406
<b>5.3</b>	<b>Introduction to Data structures</b>	<b>413</b>
5.3.1	Primitive and Non Primitive Data Structures-----	413
5.3.2	Stack and its Applications-----	414
5.3.3	Queues and its Applications -----	417
5.3.4	Linked Lists and its Applications -----	419
5.3.5	Trees and its Applications-----	421
5.3.6	Abstract Data Types -----	423
5.4	Programming Examples -----	427
5.5	Exercises-----	433

## **6. Old Question Papers 436**

<b>7. Basic Mathematical Formulae and Definitions</b>	<b>462</b>
<b>8. Computer Programming Lab Manual</b>	<b>473</b>
<b>9. References</b>	<b>571</b>
<b>10. Appendix</b>	<b>573</b>
<b>Additional Programming Examples</b>	<b>584</b>
<b>Model Questions for Placement</b>	<b>616</b>

## Module 1: Introduction to C Language

**Syllabus:** *Introduction, Algorithms and Flowchart, Pseudocode Solution to Problem, Basic Concepts of a C program, Alphabets, Tokens, Data Types, Variables and Constants, Declaration, Assignment and Print statement, Operators and Expressions, Programming Examples and Exercises.*

**1.1 Introduction:** C was designed and developed by Dennis Ritchie at Bell Laboratories in 1972. C programming language is derived from earlier languages B, BCPL (Basic Combined Programming Language) and CPL (Combined Programming Language). C language is a general purpose and high level structured programming language. It is an inspiration for the development of other languages. Some of the main features (main characteristics) of C languages are discussed below:

- 1. Simplicity:** C has good collection of inbuilt functions, keywords and data types. The keywords and library functions available in C resembles common English words thus it helps to improve the clarity and simplicity of the program. Using C language the complex programs can be written in a simplex easier way.
- 2. Modularity:** C language is a structured programming (sometimes known as modular programming) language that enforces a logical structure on the program being written to make it more efficient and easier to understand and modify. The C programming language allows dividing the program in to small modules or blocks and one block differs from other so that the reader can understand the program easily. This feature helps to increase the ability to understand the program.
- 3. Portability:** The ability to install the software in different platform is called portability. ‘C’ language offers highest degree of portability i.e., percentage of changes to be made to the source code are at minimum when the software is to be loaded in another platform.

- 4. Extendibility:** Ability to extend the existing software by adding new features is called as extendibility. In C language program, new features can be added at any time by the programmer. So C language program is extendable.
- 5. Speed:** ‘C’ is also called as middle level language. Due to this the programs written in ‘C’ language run at the greater speeds as compared to the program of other higher programming languages. C language has both the merits of high level and middle level language and because of this feature it is mainly used in developing system software.
- 6. Case Sensitive:** C language is a case sensitive language and it can differentiate the character either in upper case or lower case. All types of words (keywords or user defined words) in C language are case sensitive.

C is a middle level language. It has features of both low level language as well as high level language. C is closely related to lower level language such as “Assembly Language”. It is easier to write assembly codes in C programming. It is more user friendly as compared to previous high level languages such as BCPL, Pascal and other programming languages. C is a more powerful language as it provides wide variety of data types, functions, control statements and looping statements. It provides wide variety of bit manipulation operators to manage data at bit level. Also C provides efficient use of pointers and dynamic memory management mechanism.

## 1.2 Algorithm and Flow Chart

**1.2.1 Algorithm:** Algorithm is a step by step procedural description of the programming logic where each step is numbered in a hierarchical order. In algorithm, number of steps must be finite and every step must be complete and error free. The steps must specify the input and output as per the requirement of the user.

### **Characteristics of Algorithm:**

- a. **Finiteness:** An algorithm must terminate after a finite number of steps and further each step must be executable in finite amount of time.
- b. **Definiteness:** Each steps in algorithm must be precisely defined, the action to be carried out must be rigorously and unambiguously specified for each case.
- c. **Inputs:** An algorithm must have zero or more but only finite number of inputs.
- d. **Output:** An algorithm must have one or more outputs.
- e. **Effectiveness:** An algorithm should be effective. This means that each of the operation performed in an algorithm must be sufficiently basic that it can, in principle be done exactly and in a finite length of time, using pencil or pen and paper. It may be noted that the finiteness condition is a special case of effectiveness. If a sequence of steps is not finite, then it cannot be effective also.

### **Examples: Write an algorithm for the following**

- 1. To read the percentage marks of the student and display PASS or FAIL.**

#### **Algorithm:**

Step 0: Start  
Step 1: Read the percentage of Marks obtained by the student  
Step 2: if (Marks>35)  
          Print PASS  
          else  
          Print FAIL  
Step 3: Stop

- 2. To determine whether the given number is odd or even.**

#### **Algorithm:**

Step 0: Start  
Step 1: Read the number say n.  
Step 2: if (n%2==0)  
          Print EVEN

```
    else  
        Print ODD  
    Step 3: Stop
```

### 3. To compute the Volume and Surface area of the Sphere.

#### Algorithm:

```
Step 0: Start  
Step 1: Read the radius of the sphere say r.  
Step 2: Define the Constant PI = 3.14159  
Step 3: Compute the Volume V =  $4/3 * \text{PI} * (r * r * r)$   
Step 4: Compute the Surface Area S =  $4 * \text{PI} * (r * r)$   
Step 5: Stop
```

### 4. To find the sum of first n natural numbers.

#### Algorithm:

```
Step 0: Start  
Step 1: Initialize sum = 0  
Step 2: Read the value of n  
Step 3: Set counter i = 1  
Step 4: Compute the sum = sum + i;  
Step 5: Increment i = i+1  
Step 6: Repeat step 4 and 5 until i<=n  
Step 7: Print Sum  
Step 8: Stop
```

### 5. To calculate the factorial of a given number.

#### Algorithm:

```
Step 0: Start  
Step 1: Initialize product, p=1 as special case ( $0!=1$ )  
Step 2: Read the value of n whose factorial is to be calculated  
Step 3: Set counter i = 1  
Step 4: Compute the product p = p*i;  
Step 5: Increment i = i+1  
Step 6: Repeat step 4 and 5 until i<=n  
Step 7: Print the product p as factorial of n  
Step 8: Stop
```

**6. Algorithm to swap the contents of two variables using third variable.**

**Algorithm:**

Step 0: Start  
Step 1: Read a, b;  
Step 2: [Exchange a & b]  
    temp =a;  
    a =b;  
    b=temp;  
Step 3: [Output the result]  
    Display a b.  
Step 4: Stop.

**7. Algorithm to check whether a given number is positive or negative.**

**Algorithm:**

Step 0: Start  
Step 1: Read n.  
Step 2: [Check whether the number is +ve or -ve].  
    if( $n < 0$ )  
        Display “The number is negative”  
    else  
        Display “The number is positive”.  
Step 3: Stop.

**8. Algorithm to find the area of the triangle.**

**Algorithm:**

Step 0: Start  
Step 1: [Input 3 sides of the triangle] i.e. Read a,b,c.  
Step 2: [Compute the value of s].  
     $s = (a+b+c)/2$ .  
Step 3: [Compute the area of the triangle].  
    area= $\sqrt{s*(s-a)*(s-b)*(s-c)}$   
Step 4: Display area.  
Step 5: Stop.

**9. Algorithm to find the sum of first N natural numbers.**

**Algorithm:**

Step 0: Start  
Step 1: Read n.  
Step 2: Initialize sum = 0  
Step 3: Find the sum of all terms  
    for i = 1 to n in steps of 1 do  
        sum = sum + i  
    end for.  
Step 4: Display sum.  
Step 5: Stop.

**10. Algorithm to find the biggest of 3 numbers.****Algorithm:**

Step 0: Start  
Step 1: Read a,b,c.  
Step 2: [Comparison]  
    if((a>b) and (a>c) ) then  
        display a is greatest.  
    else if( (b>a) and (b>c))then  
        display b is greatest.  
    else display c is greatest.  
Step 3: Stop.

**11. Algorithm to swap the two integers without using third variable****Algorithm:**

Step 0: Start  
Step 1: Read a, b;  
Step 2: [Exchange a & b]  
    a = a + b;  
    b = a - b;  
    a = a - b;  
Step 3: [Output the result]  
    Display a b.  
Step 4: Stop.

**12. Algorithm to find the area and circumference of the circle.****Algorithm :**

Step 0 : Read the radius of circle r

- Step 1 : Initialize the Pi Value  
 Step 2 : Compute Area =  $\text{Pi} \times r \times r$  ;  
 Step 3: Compute Circumference =  $2 \times \text{Pi} \times r$ ;  
 Step 4 : Display Area and Circumference

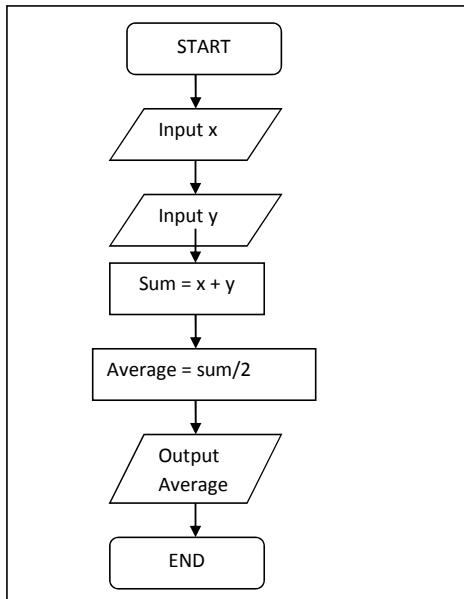
**1.2.2 Flowchart:** Flowchart is a graphical representation of an algorithm. Flowcharts use special shapes to represent different types of actions or steps in a process. Lines and arrows show the sequence of the steps, and the relationships among them. In flowchart symbols or shapes are used to represent the data flow, operations, process and recourses used to complete a given task. Table1.1 below shows the commonly used flowchart symbols and their purpose .

**Table 1.1 : Flowchart Symbols**

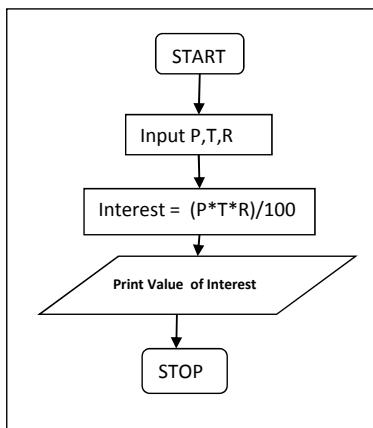
Flowchart Symbol	Purpose
	<b>Start/End:</b> The terminator (Ellipse like) symbol is used to represent the starting or ending point. It usually contains the word "Start" or "End."
	<b>Action or Process :</b> A rectangular box is used to represent a process defined in an algorithm
	<b>Decision:</b> A diamond symbol is used to represent the decision or branching point.
	<b>Input/output:</b> Parallelogram symbol is used to represent an input or output operation.
	<b>Connector:</b> Circle symbol is used to connect one point of flow to another point.
	<b>Flow Line:</b> Used to show the direction of a control flow and to connect the various flowchart symbols.
	<b>Subroutine:</b> The symbol is used to represent the sub functions in a given program.
	<b>Looping:</b> The symbol is used to represent the looping or repetition of a given set of statements.

**Examples:** Write a flowchart for the following

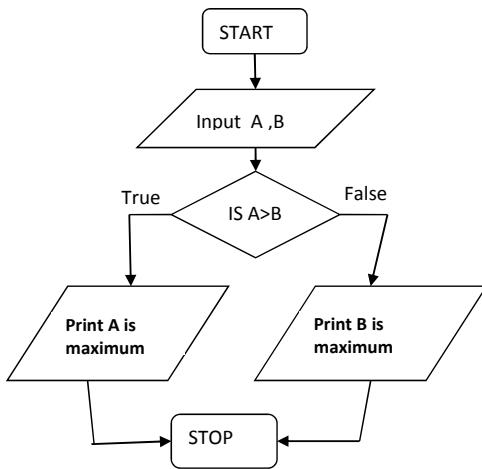
**1. To find the average of two numbers**



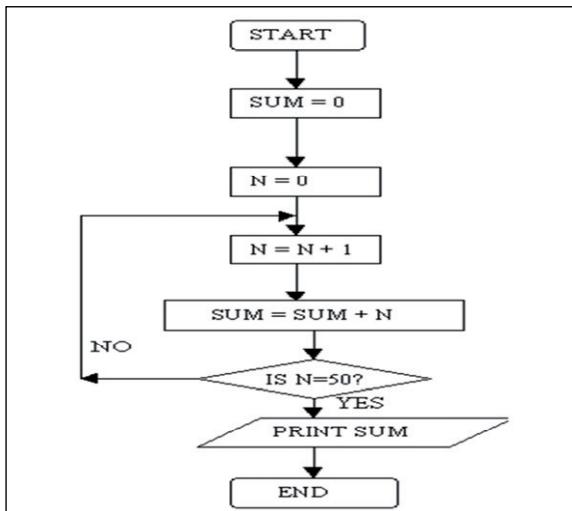
**2. To find the simple interest or interest given the value of P,T and R.**



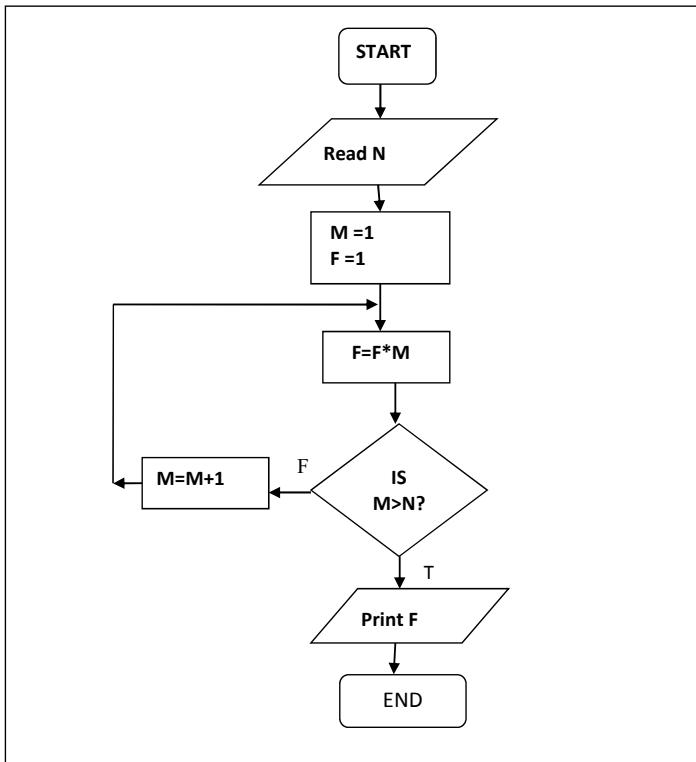
**3. To find the maximum of two numbers**



**4. To find the sum of first 50 natural numbers.**



##### 5. To find the factorial of a given number N.



**1.3: Pseudocode Solution to Problem:** Pseudocode is a problem solving tool which consists of statements written in English and C language. It is a series of steps which describes what must be done to solve a given problem. It is used for developing a program and it is constructed before writing a program. It is an informal high level description of program mixed with natural descriptions and mathematical notation. It does not have any standard syntax. Pseudocode is used to solve a problem and develop a C program as illustrated below:

**Consider a problem** of “*Displaying a list of numbers and its squares from 4 to 9*”.

**Method:** The method to be used to solve the above problem is: *Start with the first number 4 and compute its square i.e. 16. Print the number and its square. Then do the same for remaining numbers (5, 6, 7, 8 and 9).* The above method to solve the problem can be put in the form of pseudo code as given below:

**PseudoCode:**

1. **Start** with the number 4
2. Compute its square (i.e.  $4 \times 4 = 16$ )
3. Print the number and its square (i.e. 4 and 16)
4. Do the same for each of the other number from 5 to 9.
5. **End**

**Program:** The above pseudo code can be utilized to develop the following C program

```
#include<stdio.h>
#include<conio.h>
main()
{
    /*n takes the values from 4 to 9 and s for storing squares of those values */
    int n,s;
    clrscr();
    n = 4;
    s = n*n; /* s is the square of the number */
    printf(" %d %d\n",n,s);
    n = 5; s=n*n;
    printf(" %d %d \n",n,s);
    n = 6; s=n*n;
    printf(" %d %d \n",n,s);
    n = 7; s = n*n;
    printf(" %d %d \n",n,s);
    n=8; s =n*n;
    printf(" %d %d \n",n,s);
    n=9; s =n*n;
    printf(" %d %d \n",n,s);
    getch();
}
```

## **Characteristics of Pseudocode**

- Composed of a sequence of statements or steps
- Statements are written in simple English and C statements.
- Each statement is written on a separate line
- Each statement in pseudocode expresses just one action for the computer.
- There is no fixed syntax.

**Examples: Write a Pseudocode for the following:**

1. **To read the percentage marks of a student and determine whether the student has passed or not.**

**Pseudocode :**

```
Begin
Read Marks
if marks>=40
print result: PASS
else
print result : FAIL
End
```

2. **To read the number and determine whether the number is even or odd number.**

**Pseudocode:**

```
Begin
Read the number say n
if (n%2==0)
print : "The number is Even".
else
print : "The number is ODD".
End
```

3. **To calculate the volume and surface area of the sphere.**

**Pseudocode :**

```
Begin
    Read the Radius of the sphere say R
    Define PI = 3.142
    Compute Volume = (4/3)*PI*(R*R*R)
    Compute Surface_Area = 4*PI*(R*R)
    PRINT Volume and Surface_Area
End
```

**4. To find the sum of two numbers**

**Pseudocode :**

```
Begin
    Read two numbers a,b
    Compute Sum = a + b
    Print Sum
End
```

**5. To swap two numbers using temp variable.**

**Pseudocode**

```
Begin
    Read a,b
    Print a and b before swapping
    temp = a;
    a = b;
    b = temp;
    Print a and b
End
```

**Program :** Program is a set of instructions arranged in sequence used to guide the computer to solve a given problem. C program is a set of programming instructions written in C programming language (ANSI C Version) .

**1.4 Basic Concepts of a C program:** The lists of basic concepts of a C program are as follows:

1. The Basic Structure of a C program
2. Comments and Programming Style
3. The Program Header
4. The body or Action portion of the program

**1.4.1 The Structure of a C program:** The general basic structure of a C program is shown below:

```
[Comments /Documentation Section]
[Pre-processor Directives/Link Section]
[Global Declaration Section]
[Function Declaration Section]

main()
{
    Declaration Section
    Executable part
}

Subprogram section
[Function1
Function2

.
.

Functionn]
```

**Fig 1.1:** Generic Structure of a C program

**The Documentation /Comments Section** consists of a set of comment lines giving the short description /purpose of the program or any statement of the program and other details.

**The Link/Pre-processor Section** provides instructions to the compiler to link functions from the system library.

**The Definition Section** defines all symbolic constants.

**The Global Declaration Section:** In this section variables are declared outside the main function and these variables can be accessed by all functions.

**The Function Declaration Section:** Here the prototype declarations of sub functions are declared.

**The main( ) function:** Every C program must have one main function section. This section contains two parts, declaration and executable part.

**Declaration Part** declares all the variables used in the executable part. There should be at least one statement in the **executable part** which contains instructions to perform certain task. The declaration and executable part must appear between the opening and closing braces. All statements in the declaration part should end with the semicolon.

The **Subprogram Section** contains all the user defined functions that are called in the main function. The example given below illustrates the structure of C program.

**Example :**

```
1 /*Documentation Section: program to find the area of circle*/
2 #include <stdio.h> /*link section*/
3 #include <conio.h> /*link section*/
4 #define PI 3.14 /*definition section*/
5 float area; /*global declaration section*/
6 void main()
7 {
8     float r; /*declaration part*/
9     clrscr();
10    printf("Enter the radius of the circle\n"); /*executable part starts here*/
11    scanf("%f",&r);
12    area=PI*r*r;
13    printf("Area of the circle=%f",area);
14    getch();
15 }
```

**1.4.2 Comments and Programming Style:** Comments are programming constructs which are used to describe briefly the purpose of each statement, set of statements or entire program.

**Uses of Comments:** Comments (documentation/ short descriptions) are written along with program for the following reasons:

- a. To read and understand the logic of the program
- b. To understand the semantics of the program
- c. To maintain and modify the program

Comments or documentation begins with /\* and ends with \*/. The symbols /\* and \*/ are called *comment delimiters* because they delimit or mark the beginning and end of the comment.

### **Examples:**

```
/* Program Written by Palguni , USN : 2SD12345 on 1-12-2015 */  
/*The function avg(t1,t2) determines the average two tests */  
// To Read the Value of n  
// Variable strength represents the strength of class  
/*To Compute the perimeter of the rectangle*/
```

**Programming Style:** The comments can be used *anywhere inside* the program file. It can be used inside the main function or outside the main function. In order to describe the entire program the comments are written at the top of the program before header files. In order to describe the statement or set of statements the comments are included either at the top or at the right side of the statements. Following example illustrates the usage of comments in a C program:

```
/*Program to accept the number and display the number */  
#include<stdio.h> /*Header Files*/  
#include<conio.h>  
main() /* Beginning of the Program */  
{    int n; /* Declaration */  
    clrscr();  
    printf("\n Enter the number\n");  
    scanf("%d",&n); /* Reading the number from the keyboard */  
    /*Displaying the number entered */  
    printf("\n The Entered number is %d\n",n);  
    getch();  
} /* End of the Program */
```

**1.4.3 The program header:** The program header consists of the following sections

- a) Preprocessor directives
- b) Global declaration and definitions
- c) Main program header

**a) Preprocessor directives:** The preprocessor directives are the statements which start with symbol #. These statements instruct the compiler to include some of the files in the beginning of the program. For example,

#include<stdio.h>, #include<math.h> and #include<conio.h> are some of the files that the compiler includes in the beginning of the program. The #include<stdio.h> statement tells the compiler to include the standard input/output library or header file. This file has the function definition for built in functions like printf (), scanf() ,etc.

Using the preprocessor directives the user can define the constants also. For example

```
#define MAX 1024  
#define PI 3.1417
```

Here MAX and PI are called symbolic constants.

**b) Global Declaration and function definitions:** The variables that are declared before all the functions are called global variables. The global variables can be accessed by all the functions including main function. As we declare the global variables, the functions are also declared here as illustrated below:

**Example:**

```
#include<stdio.h>  
#include<conio.h>  
  
/* Demonstrating Global variables and function */  
  
add_numbers ( ); /* Function declaration */  
int value1,value2, value3; /* Declaration of global variables */  
  
add_numbers()  
{  
    int result2; /* Declaration of local variable*/  
    value1=5;  
    value2=3;  
    value3=2;  
    result2 = value1 + value2 + value3;  
    printf (" Result 2 = %d \n",result2);  
}  
main()  
{  
    int result1;
```

```

clrscr();
value1 = 10;
value2 = 20;
value3 = 30;
result1 = value1 + value2 + value3;
printf("Result 1 = %d\n",result1);
add_numbers();
getch();
}

```

**Output :**

```

Result 1 = 60
Result 2 = 10

```

**c) main program header:** In C program the execution always starts from the `main()` function and it can be written as follows :

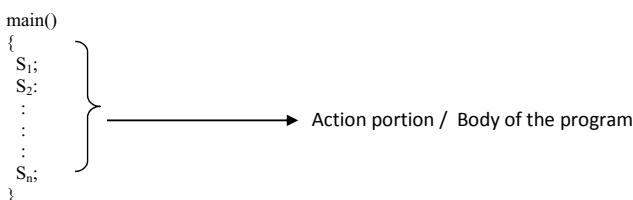
```

#include<stdio.h>
#include<conio.h>
main() /* the main statement*/
{
}

```

The main statement instructs the compiler that execution always starts from function `main` and it is called *main program header*. The pair of brackets denoted by `( )` must follow after the `main`. Every C program must have only one main function.

**1.4.4 The Body or Action Portion of the Program:** Inside every C main program after the header or top line is a set of braces `{` and `}` containing a series of C statements which comprise the body. This is called the action portion of the program.



The body of the program contains two essential parts:

- Declaration Section
- Execution Section

**Declaration Section:** The variables that are used inside the main function or sub function should be declared in the declaration section. The variables can also be initialized during declaration. For example consider the declarations shown below:

```
main ()  
{  
    int sum = 0;  
    int a;  
    float b;  
    -----  
}
```



Declaration of the variables

Here the variable **sum** is declared as an integer variable and it is also initialized to zero. The variable **a** is declared as an integer variable whereas the variable **b** is declared as a floating point variable.

**Executable Section:** This section includes the instructions given to the computer to perform a specific task. An instruction may represent an expression to be evaluated, input/output statement etc. Each executable statement ends with “;”. The instructions can be *input or output statements, simple statements such as if statement, for statement etc.*

**Example:**

```
main()  
{  
    -----  
  
    printf("nEnter the values of a and b\n");  
    scanf ("%d %d",&a,&b);  
    sum =a+b;  
    printf(" \n The sum=%d \n", sum);  
}
```



Executable Statements

## Compiling and Running a C program

### **1. Compiling and Running a C Program on Ubuntu Linux :**

**Step 1:** Open or create file using the command :

***gedit filename.c***

**Step 2:** Write a program and save the file.

**Step 3:** Compile the simple programs using the command :

***cc filename.c***

**(Note : one can also use *gcc filename.c*)**

If the program contains math functions use the following command

***cc filename.c -lm***

**(Note : once can also use *gcc filename.c -lm*)**

**Step 4:** Execute the program using the command : ***./a.out***

### **2. Compiling and Running a C program on Windows Turbo C:**

**Step 1:** Locate the TC.exe file and open it. You will find it at location C:\TC\BIN\.

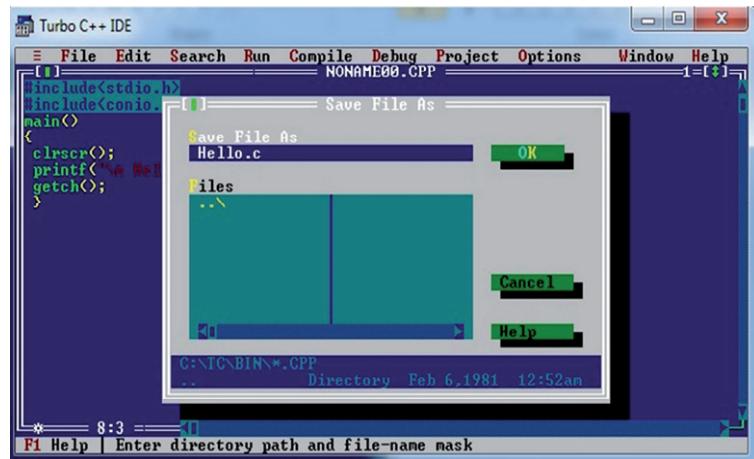


**Step 2:** File-> New (as shown in above picture) and then write your C program as given below

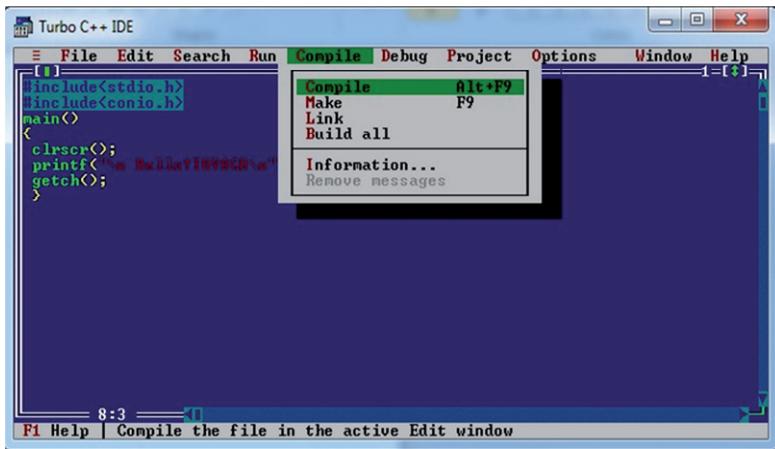


The screenshot shows the Turbo C++ IDE interface. The menu bar includes File, Edit, Search, Run, Compile, Debug, Project, Options, Window, and Help. The title bar says "NONAME00.CPP". The main workspace is empty. The status bar at the bottom shows "8:3" and various keyboard shortcuts like F1 Help, F2 Save, F3 Open, Alt-F9 Compile, F9 Make, and F10 Menu.

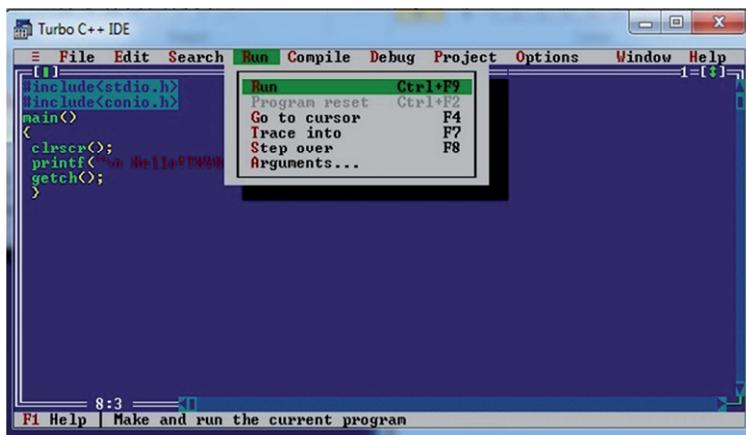
**Step 3:** Save the program using F2 (OR File-> Save as), remember the extension should be ".c". In the below screenshot I have given the name as Hello.c.



**Step 4:** Compile the program using Alt + F9 OR Compile-> Compile (as shown in the below screen shot).



**Step 5:** Press Ctrl + F9 to Run (or select Run-> Run in menu bar ) the C program.



**Step 6:** Alt + F5 to view the output of the program at the output screen.



## 1.5 Alphabet, Tokens, Data Types, Variables and Constants

The basic ( fundamental) concepts of C Language are : Alphabet (Character Set), Tokens, Data Types, Variable and Constants. The steps to learn C language are exactly similar to the steps that we follow to learn natural languages such as Kannada,Tulu, English, etc. The sequence of steps to learn C Language is as follows:

*Alphabet->Tokens->Statements->Program*

**1.5.1 Alphabet:** A symbol that is used while writing a program is called a character or an alphabet. A character may be letter or digit or any special symbol. There are 96 symbols available in C (*that represent letter in English and akshara in Kannada*) as listed in the following table.

**Table 1.2 : The Alphabet of C**

Type	Symbols or Character Set
Letters	a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Digits	0 1 2 3 4 5 6 7 8 9
Symbols /Special Characters	! " # % & ' ( ) * + , - . / : ; < = > ? [ \ ] ^ _ {   } ~
White Spaces	form feed, newline(\n), space, horizontal tab (\t) and vertical tab (\v)
Delimiters	: ; ( ) [ ] { } # ,

**1.5.2 Tokens:** A token is a smallest or basic unit of a C program. They are the basic buildings blocks (*that resemble a word in English, a pada in kannada*) of C language which are used to write a C program. There are six categories of tokens in C as given below:

- a. Keywords (eg: int, while),
  - b. Identifiers (eg: main, total),
  - c. Constants (eg: 10, 20),
  - d. Strings (eg: "total", "hello"),
  - e. Special symbols (eg: (), {}),
  - f. Operators (eg: +, /, -, \*)

Consider a simple C program as given below:

```
int main()
{
    int x,y,total;
    x =10, y = 20;
    total = x + y;
    printf("Total=%d\n",total);
}
```

Following are the different tokens used in above program,

- main – identifier
- {}, () – delimiter or special symbols.
- int – keyword
- x, y, total – identifier
- main, {}, (), int, x, y, total – tokens

Following section discusses in detail the different category of tokens

**a. Keywords:** It is a reserved word with predefined meaning in C. There are 32 keywords in C. Each keyword has fixed meaning that cannot be changed by user. For example in the statement: **int money;** **int** is a keyword, that indicates '**money**' is of type integer variable. As, C programming is case sensitive, all keywords must be written in lowercase. The table below gives the *list of all keywords* predefined by ANSI C.

**Table 1.3 : Keywords used in C Language**

Keywords in C Language			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

## Description of Keywords

1. **auto** : Defines a local variable as having a local lifetime.
2. **break** : Passes control out of the compound statement.
3. **case** : Used in switch statement to represent the branch point containing the compound statement .
4. **char** : Represents the data type that holds characters
5. **const** : Makes variable value or pointer parameter unmodifiable
6. **continue** : Passes control to the beginning of the loop
7. **default** : Specifies the default block of code in a switch statement.
8. **do** : Keyword **do** is usually used together with **while** to make another form of repeating statement. It starts a **do while** loop.
9. **double** : Represents a double precision floating point data type.
10. **else** : Indicates an alternative branch in the if else statement .
11. **enum** : Defines a set of constants of type int
12. **extern** : Indicates that an identifier is defined elsewhere
13. **float** : The keyword float usually represents a single precision floating point data type.
14. **for** : Used in for loop to provide iteration facility repeatedly.
15. **goto** : Used to jump the control unconditionally from one part to another.
16. **if** : Used to execute the statements conditionally.
17. **int** :Refers to a fundamental data type that holds integer type values.
18. **long** : Modifier used to hold long type values along with basic data types.
19. **register** : Tells the compiler to store the variable being declared in a CPU register
20. **return** : Exits immediately from the currently executing function to the calling routine, optionally returning a value.
21. **short** : Modifier used to represent the short type values of basic data types.
22. **signed** : Modifier that holds the signed type values of a data type .
23. **sizeof** : Returns the size of a specified parameter.
24. **static** : Preserves variable value to survive after its scope ends
25. **struct** : Groups variables into a single record
26. **switch** :Multiple branching statement which causes control to branch to one of a list of possible statements in the block of statements .
27. **typedef** : Assigns new name to data type definition.
28. **union** : Groups the variables sharing the same storage space.
29. **unsigned**: modifier used to represent the unsigned type values of a data type.
30. **void** : represents the empty data type.
31. **volatile** : Indicates that a variable can be changed by a background routine.
32. **while** : Repeats execution of statements while the condition is true.

**b. Identifiers:** In C programming language, identifiers *are names* given to C entities, such as *variables, functions, structures* etc. Identifier is created to give unique name to C entities to identify it during the execution of program. For example:

```
float avg, height;  
int no_stud, tot_marks;
```

Here, *avg* and *height* are identifiers which denotes a variables of type float. Similarly, *no\_stud* and *tot\_marks* are identifiers, which denotes the variables of type integer.

### Rules for writing identifier

1. An identifier can be composed of letters (both uppercase and lowercase letters), digits and underscore '\_' only.
2. The first letter of identifier should be either a letter or an underscore.
3. The identifier must not include keywords or reserved words.
4. The length of identifier may be 63 for local entities and 31 for global entities.

**Tips for Good Programming Practice:** *Programmer can choose the name of identifier whatever they want. However, if the programmer choose meaningful name for an identifier, it will be easy to understand and work on, particularly in case of large program.*

### c. Constants:

Constants are the terms that can't be changed during the execution of a program. For example: 1, 2.5, "Programming is easy." etc. In C, constants can be classified as: *character constant , integer constant , floating point constant and string constant.* (The details are discussed in section 1.5.4.2)

**d. Operators:** Operators are the symbol which operates on value or a variable. For example: + is an operator to perform addition. Ex:+, -, \* , / , & , ^ , && , || , ----etc.

**e. Special Symbols:** The following special symbols are used in C having special meaning and thus cannot be used for other purpose: { } , [ ] , ( ) , etc.

**Braces{ }:** These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.

**Parentheses( ):** These special symbols are used to indicate function calls and function parameters.

**Brackets[ ]:** Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.

**1.5.3 Data Types:** It is the type of data value that is stored in particular memory location. Data types are used to classify the values as integer, real, character, boolean etc. In the C programming language, **data types** are used for **declaring variables of different types**. The different categories of types used in C programming languages are as follows:

- a. Primitive data types:
- b. Derived data types:
- c. Enumeration data types :
- d. Void data type
- e. User defined data types

**a. Primitive Data types:** Primitive data types are most fundamental data types which cannot be decomposed **further into simpler ones and can be manipulated directly by machine instructions**. They are also known as basic and built-in data types. These data types are the most basic building blocks of C programming language and numerous composite data types are constructed using them. C language supports **four basic**

**data types specifier like : int** (Ex: 1,20 and 352), **char** (Ex: ‘c’, ‘A’ and ‘X’), **float** (Ex: 25.256789) and **double** (Ex: 12.789123450000000). The C language also supports **optional specifier** like signed ,unsigned ,short and long. Table1.4 gives the different primitive data types and their variation used in C language along with their storage size and value range.

- b. Derived Data Types:** These are the composite data types which are derived from multiple primitive datatypes.

Example : Array, structure and union..

**Table 1.4 : Primitive Data Types used in C along with their Value Range**

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long int	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long int	4 bytes	0 to 4,294,967,295
float (precision 6)	4 bytes	3.4 e-38 to 3.4 e+38
double (precision 15)	8 bytes	1.7e-308 to 1.7e+308

- c. Enumeration Data types:** Enumeration data type consists of named integer constants as a list. It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.

**Syntax :** enum identifier [optional{ enumerator-list }];

**Examples:** enum month {Jan, Feb, Mar};

/\* In the above case Jan, Feb and Mar variables will be assigned to 0, 1 and 2 respectively by default \*/

```
enum month { Jan = 1, Feb, Mar };
```

/\* In the above case Feb and Mar variables will be assigned to 2 and 3 respectively by default \*/

```
enum month { Jan = 20, Feb, Mar };
```

/\* Jan is assigned to 20. Feb and Mar variables will be assigned to 21 and 22 respectively by default \*/

**d. Void Data Type :** Void is an empty data type that has no value. This can be used in functions and pointers.

**e. User defined Data Types :** C supports the features “*typedef*” that allows users to define the identifier which would represent an existing data type. This defined data type can then be used to declare variables:

**Syntax:**

```
typedef int numbers;  
numbers num1,num2;
```

In this example, num1 and num2 are declared as integer variables. The main advantage of user defined data type is that it increases the *program's readability*.

**1.5.4 Variables and constants:** In C program we come across two types of data objects called variables and constant.

**1.5.4.1 Variables:** A variable is a name given to physical location within the computer memory that can hold one value at a time. As the name implies a variable can change its value as the program is running. The purpose of any

variable in a program is to hold the value of location at a particular point of time. The variables may be integer, character, float and double.

**Example:** int x1, x2, si, prod; (here x1,x2,si, prod are integer variables).

The variables may be *automatic*, *external* and *static variables*. The *non-automatic variables* are initialized only once before the program execution starts. An *explicitly initialized automatic* variables, is initialized each time the function is invoked, and they do not retain their values from one call to the next. *External and static variables* are initialized to zero by default. *Automatic variables* for which there is no explicit **initializer** have undefined (garbage) values. The detailed discussion about different storage classes is discussed in Module 3.

**Name of the variable:** The name of a variable can be composed of *letters*, *digits*, and *the underscore character*. The name of the variable is also known as **identifier**. It must begin with either a *letter* or *an underscore*. Upper and lowercase letters are distinct because *C is case-sensitive*.

#### Rules for naming the variables .

1. The first character in the variable name should be a *letter* or *an underscore*.
2. The first character can be followed by any number of *letters or digits* (0 to 9) or *underscores*.
3. No extra symbols are allowed other than *letter, digits and underscore*.
4. C Keywords must *not be used*.
5. The maximum size of the characters supported for naming the *internal variable* is 31. (*depends suitably*)

**1.5.4.2 Constants:** Constants are the data entities or objects that can't be changed during the execution of a program. For example: 1, 2.5, "Programming is easy." etc. In C, constants can be classified as:

**a) Integer constants:** Integer constants are the numeric constants (constant associated with number) without any fractional part or exponential part. There are three types of integer constants in C language: *decimal constant (base 10)*, *octal constant (base 8)* and *hexadecimal constant (base 16)*.

**Decimal constants:** Made up of decimal digits (0 1 2 3 4 5 6 7 8 9)

Examples: 0, -9, 22, etc.

**Octal Constants:** Made up of octal digits (0 1 2 3 4 5 6 7 )

Examples: 021, 077, 033, etc.

**Hexadecimal constants:** Made up of hexadecimal digits (0 1 2 3 4 5 6 7 8 9

A B C D E F.)

Examples: 0x7f, 0x2a, 0x521, etc.

**b) Floating-point constants:** Floating point constants are the numeric constant that has either fractional form or exponent form.

For example: -2.0, 0.000234 and -0.22E-5

**Note:** Here, E-5 represents  $10^{-5}$ . Thus, -0.22E-5 = -0.0000022.

**c) Character constants:** Character constants are the constant which contains only one character enclosed within a pair of single quote marks. For example: 'a', 'l', 'm', 'F' ,etc.

**d) String constants:** String constants (stream of characters) are the constants containing a sequence of zero or more characters which are enclosed in a pair of double-quote marks like, "RAMA", "1SD14CS001", "Bangalore-98".

**Examples:**

" " //null string constant

" " //string constant of six white space

"x" //string constant having single character.

"Earth is round\n" //prints string with newline

**e) Escape Sequences:** An escape sequence character begins with a backslash (\ ) and is followed by single character . When a backslash is used with some characters like a,b,n,t,f,etc., it will give rise to special print effect by escaping

(changing) the meaning of characters. Since the escape characters starts with backslash they are also called as backslash constants. For example: \n is used for newline. The backslash( \ ) causes "escape" from the normal way the characters are interpreted by the compiler. The list of escape sequences commonly used are shown in Table1.5.

**f) Enumeration constants:** Keyword **enum** is used to declare enumeration types. For example: **enum color {yellow, green, black, white};** Here, the variable name is *color* and *yellow, green, black and white* are the enumeration constants having value 0, 1, 2 and 3 respectively by default.

**Note: Constant Expression:** A constant expression is an expression that involves only constants. Such expressions may be evaluated during compilation rather than runtime and accordingly may be used in any place that a constant can occur as in

```
#define MAXLINE 1000  
char line[MAXLINE+1];
```

**Table 1.5 : Escape Sequences**

<b>Escape Sequences</b>	<b>Character</b>
\a	Alert (bell) character
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage Return
\t	Horizontal tab
\v	Vertical tab
\\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character
\ooo	Octal number
\xhhh	Hexadecimal number

## 1.6 Declaration, Assignment and Print statement

**1.6.1 Declaration:** Declaration in C is the instruction that tells the computer to allocate storage locations for the variables of specific data type of required size that is to be used in the program. A declaration specifies a type and contains a list of one or more variables of that type.

**Examples:**

```
int lower, upper, step;  
char c, line [1000];
```

**Variable Declaration :** Each variable must be declared before it is used in a program. A simple variable declaration statement consists of a data type followed by one or more variable names separated by commas and terminated with a semicolon, as shown below:

```
data_type variable1, variable2, -----variablen;
```

**Example:**

```
float radius, volume,surf_area;  
int n,i,sum;
```

Variables can be distributed among declarations in any fashion ; the above declarations can also be written as follows:

```
float radius ; /* Radius of sphere*/  
float volume ; /* Volume of sphere */  
float surf_area; /*Surface area of sphere */  
int n; /* the number */  
int i; /* the counter */  
int sum ; /* sum of first n natural numbers */
```

The above type of declarations takes more space, but it is convenient for adding a comment to each declaration or for subsequent modifications.

**Variable Initialization:** The process of assigning a value to the variable at the time of declaration initially is called initialization. The C language allows us to initialize one or more variables in a declaration statement. If the name is followed by an *equal sign and an expression*, the expression serves as an initializer. The syntax for the variable initialization statement takes the following form:

```
data_type variable1 = expr1,variable2 = expr2,-----;
```

**Example :**

```
int    a = 10, b = 20;  
float  x = 1.25;  
char   ch1= 'A',ch2;
```

**Note :** We cannot use the initialization statement during declaration as below:

```
data_type v1= v2= v3-----; /* Error*/
```

**1.6.2 Assignment Statement:** The statements which are used for assigning the values are called assignment statements. Assignment statement can be used to assign the value for a given variable at any point of time in the given program.

Assignment statement computes a value and places it in a given storage location. The general form of assignment statement, giving the value of the expression *expr* to the variable named *varname* ,is given below:

```
varname = expr;
```

The expression *expr* on the right hand side of the assignment statement is evaluated and this value is stored in the variable name on the left hand side .

**Examples :**

```
number = 45;  
price = cost + 35;  
si = (P*T*R)/100;
```

```
s = n*n;  
bonus_pay = (basic_pay*12)/100+2000;  
test_avg = (t1+t2+t3)/3;  
d = (b*b - 4*a*c)/(2*a);  
y = num[8];  
int_rate = 5.75 ;  
big = largest(a,b,c);
```

When an assignment statement is executed, the expression on the right hand side of assignment operator = is evaluated. The resulting value say integer number or float constant will be assigned *and/or* stored in a named memory location represented by a left hand side variable name.

An variable that may be assigned a value is termed as an ***lvalue*** . On the RHS of = it can be variable/ expression/constant , but the ***lvalue*** must and should be a single variable. Hence the following statements are ***invalid assignment*** statements :

```
x + y = x-y;  
basic_pay + hra = gross_pay;  
(a*a -b)/2.6 = int_rate*principal;  
125 = reg_no;
```

#### ***Multiple Assignment Statements:***

C permits you to assign a *single value to more than one variable name* at a time and in a single statement. This can be achieved using multiple assignment.

#### **Example :**

```
x = y = z = 0.0;  
a = b = (x+y+z)/2;  
l = m = n*=p;
```

**Note :**Multiple assignments cannot be done during declaration of variable names .

**Example :**

int p = int q = int r = 0; is not a valid statement and

int x = y = z = 144; signals compiler error as y and z are undefined.

**1.6.3 Print Statement:** The statement which instructs the compiler to display or print the string, expression value or variable value to the output device is called print statement. The printed or displayed results of running a program are called **output**. In C programming language the output is printed using formatted output statement **printf()**. There are **two** simple forms of **printf()** statement:

1. That has a *literal string*, a sequence of characters within quotation marks.
2. That has *literal string, conversion specifier* and *any or all of the following: variables ,constant and expressions* values to be printed.

**Type 1:** The general form of a printf statement with just a literal string is

```
printf("Literal String ");
```

Here literal string is a stream of characters consisting some message.

**Example :**

```
printf("Welcome to computer programming\n ");
printf(" Hello C Program\n ");
```

**Type 2:** Type 2 printf statement is formatted printf() which uses the format specifier or conversion specifier within the literal string . The General form of printf, with variables or expressions or constants and literal string to be printed is

```
printf("Literal Strings and /Conversion specifier ", list of variables);
```

**Example :**

```
printf ("%d %d ", number,sqnumber);
printf(" The sum = %d ", sum);
printf (" The factorial of a given number n is %d\n",fact(n));
```

**Question:** What is the output of the following printf statements?

1. `printf("%d\n",number+1); /* Assume number =2*/`
2. `printf ("%d\n",number*number);`

**Output :**

3

4

Different conversion specifiers are used for printing different types of variable values. Table below summarizes the different conversion specifiers used in the C programming language:

**Table 1.6 : Conversion Specifiers Used in printf**

Conversion Specifier	Description
<code>%c</code>	To print the character type
<code>%d</code>	To print the integer type
<code>%f</code>	To print the floating point type
<code>%e or % E</code>	To display the floating point with exponent
<code>%g</code>	To display the floating point data with or without exponent ( Trailing zeroes will not be displayed)
<code>%s</code>	To display a string
<code>%lf</code>	To display long float or double
<code>%ld</code>	To display long integer
<code>%o</code>	To display octal integer
<code>%x or %X</code>	To display hexadecimal integer
<code>%i</code>	To display decimal or octal or hexadecimal integer
<code>%h</code>	To display a short integer
<code>%u</code>	To display unsigned integer

**Guidelines to use printf statement**

1. The control string or format string must be included in the quotation marks (“ ”).

**Example :** `printf("abcdef");`

2. To print the value of the variables or constants the printf statement must include the conversion specifiers like `%d`, `%f`, `%c`, etc.

**Example :** `printf("The Value of simple interest is %f",si) ;`

3. When the printf is executed each conversion specifier is replaced by the value of the corresponding expression and then printed according to the rules in the specification.
4. The *escape sequences* like \n,\t,\a, etc., tells the machine to do specific escape function. For example \n tells the machine to skip to new line. This part of the control string affects the appearance of the output but is not displayed as part of it.
5. All other things (*other than escape sequence and conversion specifier*) within the control string will print exactly as it appears.
6. If there are many variables or expressions to be printed, commas are used to separate them from the control string and each other.  
Example : `printf("%d %d %f",x1,x2,x3);` Once comma is used to separate, it is not necessary to add blank spaces. For example the two statements below are equivalent:

`printf("%d %d\n",number,sqnumber) and`  
`printf("%d%d\n",number,sqnumber);`

### Some of the Illustrations to use the printf statement

a. Printing a single string : `printf("Hello , Thyagu ");`

b. Printing strings using multiple printf statements

```
printf ("TGS");
printf("TGS");
printf("TGS");
```

These printf statements produce the output :TGSTGSTGS

```
printf ("TGS ");
printf("TGS ");
printf("TGS ");
```

The above statements will produce the output :TGS TGS TGS

The same output can also obtained using single printf statement as follows : `printf("TGS TGS TGS");`

### c. Using escape sequences in format string

#### Examples:

```
printf("Hello Raju \n");
printf("Thyagu is following You \n");
```

The output of the above statements are as follows :

Hello Raju  
Thyagu is following You

### d. Using tabs to separate output :Consider the printf statement given below : printf ("Raju\t\t Palu \t Rashi \t Kiran\t\t Sheeba \t Ravi");

The output of the above statement is :

Raju              Palu    Rashi    Kiran    Sheeba    Ravi

### e. Printing the values of different data types :

Consider the **printf** statement given below :

```
printf("%d %f %c %c %s \n",100,1.33,65, 'B', "Thyagu");
```

The output of the above statement is as follows :

100              1.330000    A    B    Thyagu

### f. Printing expressions :The **printf** function can also be used to print values of variables and expressions. Consider the example given below:

```
int a=10,b =20,c =2;
printf("%d %d %d \n",a+2,a*c,b/c,b%c*a);
```

The output printed by these printf statement is given below:

12    20    10    0

### g. Using minimum field width and precision: In the printf statement one can specify the minimum field width , precision ,justification (*left or right*), etc., using the format given below:

%	Flag (-)	Minimum Field Width	•	Precision	Conversion OR format specifying character(d,f,c,s, etc.)
---	----------	---------------------	---	-----------	--

- Flag (-) is used to print the values to be left justified
- Precision specifies the minimum number of digits to be printed after the decimal point. *If a number being printed has fewer digits after the decimal point, it is padded to zeros. If it has more digits after the decimal point its value is rounded to the specified digit.*

**Example :** `printf("%10d\n",19);`

`printf("%10.5f %10.3f\n",9.3756789, -6.5);`

`printf("%10s\n","FIVE");`

`printf("%10c\n",'R');`

The output of these statements is given below:

19	9.37567	-6.500
	FIVE	
	R	

#### 1.6.4 Reading Statement (*Reading Data using scanf statement*)

Data read by a program can come from **two** places: *the keyboard or a data file*. In C programming language the data is read from the keyboard using formatted input statement `scanf()`. The general syntax of `scanf` function is given below :

```
scanf("format String ", address list);
scanf("format String",&v1,&v2,----&vn);
```

**Example :**

```
scanf("%d %f %c", &x,&y,&z); /* To read multiple data type */
scanf("%d",&n); /* To read the integer */
scanf("%c",&h); /* To read the character */
scanf("%s",y); /* To read the string one must not use & */
scanf("%d",&a[i]); /* To read array element */
```

**Note :** All the Conversion specifiers used in `printf` statement can also be used in `scanf` statement.

**Table 1.7 : Conversion specifiers used in scanf**

Conversion Specifier	Description
%c	To read the character type
%d	To read the integer type
%f	To read the floating point type
%s	To read a string
%e or % E	To read the floating point with exponent
%g	To read the floating point data with or without exponent
%lf	To read long float
%ld	To read long integer
%o	To read octal integer
%x or %X	To read hexadecimal integer
%i	To read decimal or octal or hexadecimal integer
%h	To read a short integer
%u	To read unsigned integer

### Guidelines to use scanf statement

1. Conversion specifiers has to be enclosed within double quotes :

**Example :** `scanf("%d",&x);`

2. For each input variable , there must be a respective conversion specifier. **Example :** `scanf("%d %d %d",&x,&y,&z);`

3. Multiple conversion specifiers are allowed :

**Example :** `scanf("%d %f %c",&p,&q,&r);`

4. Use ampersand symbol followed by the input variable in the address list . **Example :** `scanf("%d",&x);`

5. Use comma operators to separate input variables .

6. Use comma operator to separate the conversion specifiers enclosed within the double quotes and address list.

## **Input and Output functions:**

The input and output functions supported by C language can be categorized into the following:

**1. Formatted Input Output Functions:** In C language scanf() and printf() are the two formatted input and output functions used respectively. The detailed discussions of printf() and scanf() is provided in the section 1.6.3 and 1.6.4.

**2. Unformatted Input Output Functions:** The list of *unformatted input* functions supported in C language are as follows :

**1. getchar()** : It is used to get or read the input (i.e a single character) at run time from standard input device.

**Syntax:** char ch;  
ch = getchar();

**2. getch()**:It is used to get the character from the console but does not echoes.

**Syntax:** char ch;  
ch = getch(); or getch();

**3. getche()**: It is used to get the character from the console and echoes on the screen .

**Syntax:** char ch;  
ch = getche();

**4. getc()** : It is used to get the next character from the File or console .

**Syntax :** char ch ;  
ch = getc(stdin);

**5. gets()**; Reads a string of characters from the keyboard till the user presses “EnterKey” .

**Syntax :** char s[10];  
gets(s);

The ***list of unformatted output functions*** supported in C language is as follows:

**1.putch()** : It is used to display the character on the output device.

**Syntax :** putch(character\_variable);

**2.putchar()**:It is used to write the character to console. It is same as calling putc(ch,stdout) .

**Syntax :** putchar(character\_variable);

**3.puts()** : It is used display a single or multiple characters of string including spaces to the standard Output device.

**Syntax :** puts(variable\_name);

**Example Declaration:**      char ch[10] = "Example";  
                                  puts(ch);

**4.putc()**: It takes a character and outputs it to the specified File or console.

**Syntax :** putc(variable\_name,FilePointer);

**Example:** Write a C program to read a character and display a character on the screen using ***getchar()*** and ***putchar()*** function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch; /*ch variable takes the typed character */
    clrscr(); /*clears the screen, so that previous displays are cleared */
    printf("\n Enter the Character : \n");
    ch = getchar();
    printf("\n The Entered Character is: \n");
    putchar(ch);
    getch();
}
```

**Output**

```
Enter the character :
R
The Entered Character is :
R
```

**Example:** C program to read a character and to display a character on the screen using **getch()** and **putch()** functions.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("\n Enter the Character: \n");
    ch = getch();
    printf("\n The Entered Character is: \n");
    putch(ch);
    getch();
}
```

**Output**  
Enter the character :  
R  
The Entered Character is :  
R

**Example :** Program to illustrate the usage of **getc( )** and **putc( )** .

```
#include<stdio.h>
#include<conio.h>
int main()
{
    char c;
    clrscr();
    printf("Enter character: ");
    c = getc(stdin); /* Reading the character from standard input device*/
    printf("Character entered: ");
    putc(c, stdout);
    getch();
    return(0);
}
```

**Output :**  
Enter character : a  
Character entered : a

**Example:** C program to read and display a string of characters using **gets** and **puts** functions.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char str[15];
    clrscr();
    printf("\n Enter a string \n");
    gets(str);
    printf("\n The Entered string is \n");
    puts(str);
    getch();
}
```

**Output**  
Enter a string :  
PALU  
The Entered string is :  
PALU

## 1.7 Operators and Expressions:

**Operator:** An operator is a symbol (or a token) that specifies the operation to be performed on various types of operands. Operators connect operands i.e. constants and variables to form expressions.

**Example :** + (addition) , - (subtraction) , \* (multiplication) ,etc.

**Operand:** A constant or a variable or a function which returns a value is an operand. An operator may have one or two or three operands.

**Expression:** A sequence of operands and operators that reduces to a single value is known as expression.

**Example :**

Expression	Operands	Operators
$x + y$	x, y	+
$x * (b - c)$	x, b, c	* , -
$x \% y$	x, y	%

**Operators Classification:** The operators in C language are classified based on

- a. The number of operands an operator has and
- b. The type of operation being performed

a. **Classification based on the number of operands** : The operators are classified into three major categories based on the number of operands as shown below:

1. Unary Operators Example : ++, --, etc.
2. Binary Operators Example : +, - , \*, / etc.
3. Ternary Operators Example: ? and :

**Unary Operators:** An operator that operates on one operand to produce a result is called unary operator. Here operator precedes the operand.

**Example:**  $-10$ ,  $-1$ ,  $*b$ ,  $++x$ ,  $y--$  etc. are all the expressions with unary operators.

**Binary Operator:** An operator that operates on *two operands* in an expression to produce a result is called a binary operator. In an expression involving a binary operator, the operator is in between two operands.

**Example :**  $a+b$ ,  $a*b$  ,  $10/5$  ,etc. are all expressions consisting of binary operators.

**Ternary operators:** An operator that operates on *three operands* to produce a result is called as ternary operator. Ternary operator ? : is also called the conditional operator.

**Example:**  $a ? b : c$

**b. Classification based on the type of the operation :** The operators are classified into following eight types based on the operation they perform on operands :

1. Arithmetic operators
2. Assignment operators
3. Relational operators
4. Logical operators
5. Bit wise operators
6. Conditional operators (ternary operators)
7. Increment/decrement operators
8. Special operators

**1. Arithmetic Operators in C:** The operators that are used to perform arithmetic operation such as addition, subtraction, multiplication, division and modulus operation are called *arithmetic operators*. These operators perform operations on two operands and hence they are all *binary operators*. The meaning of all the operators along with examples is shown in table below:

**Table 1.8 : Arithmetic Operators**

S.no	Arithmetic Operators	Operation	Example
1	+	Addition	$A+B, 4+2, 6+5, -2+2, \text{etc}$
2	-	Subtraction	$A-B, 4-2, 2-4, 5-3, \text{etc}$
3	*	Multiplication	$A*B, 4*2, 2*4, 3*5, \text{etc}$
4	/	Division	$A/B, 1/5, 3/5, 34/-5, \text{etc.}$
5	%	Modulus	$A\%B, 1\%5, 5\%2, \text{etc}$

**Division:** Division operator is denoted by symbol ‘/’ and divides the first operand by second operand and returns the *quotient*. Quotient is the result obtained after division. When one of the operands is negative the truncation depends upon the implementation.

**Example:**  $1/5 = 0$  ,  $2/5 = 0$ ,  $-34/5 = -6$  ,  $4/5 = 0$  ,  $34/-5 = -6$  ,  $-34/-5 = 6$

$3/5 = 0$  and  $-3/-5 = 0$

But  $-3/5$  can be result into 0 or -1 (It is machine dependent)

$1.0/2 = 0.5$

**Modulus:** Modulus operation denoted by % divides the first operand by second operand and returns the remainder. Remainder is the result obtained after modulus operation. To perform modulus operations **both operands must be integers.**

**Examples:**  $1 \% 5 = 1$ ,  $2 \% 5 = 2$  ,  $-34 \% 5 = -4$  ,  $4 \% 5 = 4$  ,  $34 \% -5 = 4$  ,  $-34 \% -5 = -4$  .

**Example program for C arithmetic operators:** In this example program, two values “4” and “2” are used to perform arithmetic operations such as addition, subtraction, multiplication, division, modulus and output is displayed for each operation.

```
#include <stdio.h>
#include<conio.h>
int main()
{
    int a=4,b=2, add,sub,mul,div,mod;

/* The variables a , b represents the two operands for various operations*/
/* The variables add, sub, mul, div and mod are used to store the results of those
operations */

clrscr();

add = a+b;
sub = a-b;
mul = a*b;
```

```

div = a/b;
mod = a%b;
printf("Addition of a, b is : %d\n", add);
printf("Subtraction of a, b is : %d\n", sub);
printf("Multiplication of a, b is : %d\n", mul);
printf("Division of a, b is : %d\n", div);
printf("Modulus of a, b is : %d\n", mod);
getch();
}

}

```

**Output:**

Addition of a, b is : 6  
 Subtraction of a, b is : 2  
 Multiplication of a, b is : 8  
 Division of a, b is : 2  
 Modulus of a, b is : 0

**Integer Arithmetic:** When an arithmetic operation is performed on *two whole numbers or integers* than such an operation is called as *integer arithmetic*. It always gives an integer as the result. In integer division the *fractional part* is **truncated**.

**Examples:**  $2 + 5 = 7$ ,  $2 - 5 = -3$ ,  $2 * 5 = 10$ ,  $2 / 5 = 0$ ,  $5 / 2 = 2$ , etc.

**Floating point arithmetic:** When an arithmetic operation is performed on two real numbers or fraction numbers such an operation is called *floating point arithmetic*. The floating point results can be truncated according to the properties requirement. The *remainder operator is not applicable for floating point arithmetic operands*.

**Examples:**  $2.0 + 3.5 = 5.5$ ,  $5.0 / 2.0 = 2.5$ ,  $5.0 \% 2.0$ (invalid) ,etc.

**Mixed mode arithmetic:** When one of the operand is real and other is an integer and if the arithmetic operation is carried out on these 2 operands then it is called as *mixed mode arithmetic*. If anyone operand is of real type then the result will always be real thus  $15/10.0 = 1.5$ .

**Examples:**  $2/1.5 = 1.3333$ ,  $2.5 + 2 = 4.5$

**Assignment operators in C:** The = operator is also called as the assignment operator. It is used to assign a constant to the variable. The general syntax is as given below:

variable = constant ; or  
variable = expression or  
variable = another\_variable ;

**Examples :**  $a = 100$  ,  $a=b$ ,  $a=l*b;$

$x + y = l*b$  /\* Illegal \*/

There are three types of assignment statements:

**1. Simple assignment statements :** The syntax of simple assignment statement is: *variable = expression ;*

**Examples:**

$a = 10;$   
 $a = b;$   
 $a = b + c;$  etc.

**2. Short hand assignment statement ( Compound statement) :** The syntax of short hand assignment statement is :

*variable op= expression;*

Here the **op** may be operators such as  $+, -, *, %, <<, >>, !, ^$  etc. Some of the short hand assignment operators are  $+=, -=, *=, /=$  and  $\%=$ .

**Example:**  $x += 1$  is same as  $x = x + 1$

Table 1.9 illustrates the usage of simple and compound assignment operators with examples.

**3. Multiple Assignment statement:** A statement using which a value or a set of values are assigned to different variables is called multiple assignment statement. The assignment operator ‘=’ can be used to assign a single value to more than one variable.

**Example :** `i = j= k=10;`

Here the value **10** is copied into variable **k, j** and **i** from right to left.

**Table 1.9 : Examples for Assignment Statements**

Operators		Example	Explanation
Simple assignment operator	=	<code>x = 10</code>	10 is assigned to variable x
Compound assignment operators	<code>+=</code>	<code>x += 10</code>	This is same as <code>x = x + 10</code>
	<code>-=</code>	<code>x -= 10</code>	This is same as <code>x = x - 10</code>
	<code>*=</code>	<code>x *= 10</code>	This is same as <code>x = x * 10</code>
	<code>/=</code>	<code>x /= 10</code>	This is same as <code>x= x / 10</code>
	<code>%=</code>	<code>x%= 10</code>	This is same as <code>x = x % 10</code>
	<code>&amp;=</code>	<code>x&amp;=10</code>	This is same as <code>x = x&amp; 10</code>
	<code>^=</code>	<code>x^= 10</code>	This is same as <code>x = x ^ 10</code>

**Example program for C assignment operators:** In this program, values from 0 – 10 are summed up and total “55” is displayed as output. Assignment operators such as “=” and “+=” are used in this program to assign the values and to sum up the values.

```

# include <stdio.h>
#include<conio.h>
int main(){
    int tot =0,i;
    clrscr();
    for(i=0;i<=10;i++){
        tot +=i;
    }
    printf("Total = %d", tot);
    getch();
}

```

**Output:** Total = 55

- 3. Increment and Decrement Operators:** C provides two unusual and powerful operators for incrementing and decrementing variables known as *increment* and *decrement* operators. The operator `++` adds one to its operand and the operator `--` subtracts one to its operand. The *increment* and *decrement* operators may *post and pre increment or decrement operators*. The syntax and description of the operators is illustrated in the table below:

**Table 1.10 : Syntax and Description of Increment and Decrement Operators**

Operator	Syntax	Equivalent to	Description
Pre Increment Operator	<code>x = ++i</code>	<code>i=i+1</code> <code>x= i;</code>	Value of i is incremented before assigning it to variable x.
Post increment Operator	<code>x=i++</code>	<code>x = i;</code> <code>i= i+1;</code>	Value of i is incremented after assigning it to variable x.
Pre decrement operator	<code>x = --i</code>	<code>i = i -1 ;</code> <code>x = i;</code>	Value of i is decremented before assigning it to variable x.

**Example 1:**  $m = 5;$

$y = ++m; \text{(prefix)}$

In this case the value **m** will be incremented to **6** first and then it will be assigned to **y**. Therefore at the end of the execution, the value of **y** and **m** will be **6**.

**Example 2:**  $m = 5;$

$y = m++; \text{(postfix)}$

In this case the value of **m** will gets assigned first to **y** and then it will gets incremented. Therefore at the end of the execution , the value of **y** will be **5** and **m** will be **6**.

*Note: A prefix operator first adds 1 to the operand and then the result is assigned to the variable on the left. On the other hand, a postfix operator first assigns the value to the variable on the left and then increments the operand.*

**4. Relational Operators:** To take a certain decision it is often required to compare two quantities. *For example to find the good students from two students we can compare their marks in an examination.* For this purpose relational operators can be used. The operators that are used to find the relationship between two operands are called relational operators. The two operands may be constants, variables or expressions. The relationship between the two operand values results in true or false.

A simple relational expression contains only one relational operator and takes the following form.

$\text{exp1 relational\_operator exp2}$

Where **exp1** and **exp2** are expressions, which may be simple constants, variables or combination of them. Given below is a list of examples of relational expressions and evaluated values.

7.5 <= 27    TRUE  
-55 > 0    FALSE  
10 < 6 + 7    TRUE

There are **four** relational operators and two equality operators which are closely associated to relational operators as given in Table 1.11 below:

**Table 1.11 : Relational operators**

S.no	Operators	Example	Description
1	>	$x > y$	x is greater than y
2	<	$x < y$	x is less than y
3	$\geq$	$x \geq y$	x is greater than or equal to y
4	$\leq$	$x \leq y$	x is less than or equal to y
5	$\equiv$	$x \equiv y$	x is equal to y
6	$\neq$	$x \neq y$	x is not equal to y

**Example :** If a,b and c are integer variables having values 4,6 and 8 respectively then the values of several relational expression of a,b and c are:

**Table 1.12 : Relational Expressions**

Expression	Interpretation	Value of the Expression
$a < c$	True	1
$(a+c) \leq b$	False	0
$(a+b) > +c$	True	1
$(a+c) \geq 6+7$	False	0
$a == 3$	False	0
$b == 6$	True	1

**5. Logical Operators:** The operators which are used to combine two or more relational expressions and evaluate logical expressions are known as logical operators. The different types of logical operators supported in C are illustrated in table below:

**Table 1.13 : Logical Operators**

Operator	Meaning	Example
!	Logical NOT	$!(a < 0)$
&&	Logical AND	$(a \geq 0) \&\& (a \leq 100)$
	Logical OR	$(a < 0)    (a > 100)$

The usage of different logical operator and their examples is as follows:

**Logical AND (&&):**This operator is used to evaluate 2 conditions or expressions with relational operators simultaneously. If both the expressions to the left and to the right of the logical operator is true then the whole compound expression is true.

**Example:** Consider the expression  $(a > b) \&\& (x == 10)$ . Here the expression to the left is  $a > b$  and that on the right is  $x == 10$  the whole expression is true only if both expressions are true i.e., *if a is greater than b and x is equal to 10.*

**Logical OR (| |) :** The logical OR is used to combine 2 expressions or the condition evaluates to true if any one of the 2 expressions is **true**.

**Example:** Consider the expression  $(a < m) || (a < n)$ . Here the expression evaluates to true if any one of them is true or if both of them are true. It evaluates to **true** if *a is less than either m or n and when a is less than both m and n.*

**Logical NOT (!):** The logical not operator takes single expression and evaluates to **true** if the expression is **false** and evaluates to **false** if the expression is **true**. In other words it just **reverses** the value of the expression.

**For example:** In the expression ! (x >= y) the NOT expression evaluates to **true** only if *the value of x is neither greater than nor equal to y.*

**6. Bit wise operators in C:** The operators that are used to manipulate the bits of given data are called bitwise operators. When we declare any variable say **a** as follows:

```
int a;
```

```
a=14 ;
```

The variable **a** stores its value as follows (in the 16 bit pattern)

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

There are **6** bitwise operations in C language as illustrated in table 1.14 :

**Table 1.14 : Bit wise Operators**

<b>Bitwise Operators</b>	<b>Symbols Used</b>	<b>Usage</b>
Bitwise AND	&	a &b
BITWISE OR		a b
BITWISE XOR	^	a^b
Left Shift Operator	<<	a<<2
Right Shift Operator	>>	a>>2
One's Complement	~	~a

The Bitwise operations of AND, OR and XOR is given below

<b>AND</b>	<b>OR</b>	<b>XOR(^)</b>
0 & 0 = 0	0 0 =0	0 ^ 0 = 0
0 &1 = 0	0 1=1	0 ^ 1 = 1
1 & 0 = 0	1 0=1	1 ^ 0 =1
1 & 1=1	1 1=1	1^ 1 = 0

**Examples for Bitwise Operations:** Consider **a = 9** and **b = 7**, the bitwise representation of **a** and **b** is as given below:

**a = 9**

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**b = 7**

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**a & b =**

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1

**a|b = 15**

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**a^b = 14**

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**Left Shift Operator (<<):** It is used to shift the bits by the specified number of positions to the left. The general form is: **a<<num;** The left shift operation of (**a<<1**) for **a = 9** is as illustrated below:

**a = 9**

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**a<<1**

0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**Right Shift Operation: (>>)** It is very similar to the left shift operator except that the bits are shifted to the right by the specified number of positions. Its general form is: **a>>num;** The Right shift operation **b>>1** for **b= 10** is as illustrated below:

**b = 10**

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**b>>1**

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**Ones complement (~)** is another bitwise operator which changes the bit with the value 1 to 0 and vice versa . The ones complement of **a=10** is as illustrated below:

**a = 10**

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**~a**

1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**7. Conditional Operators:** Conditional operator which is also known as ternary operator, operates on three operands. *Conditional operators return one value if condition is true and returns another value if condition is false.*

**Syntax :** (exp1)? (exp2):(exp3);

**Example :** (a> 100) ? 0 : 1;

In above example, if **a** is greater than 100, 0 is returned else 1 is returned. This is equivalent to *if else* conditional statements.

The ternary operator works as follows: *exp1 is evaluated first. If the expression is true then exp2 is evaluated & its value becomes the value of the expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression. Note that only one of the expression is evaluated.*

**Example:**

```
a = 10 ,b = 15;  
x = (a > b) ? a : b ;
```

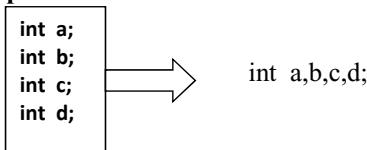
Here the value of b will be assigned to x. Since a>b is false the value of b will be assigned to x and x gets the value 15.

**8. Special Operators in C:** C supports some special operators of interest such as *comma operator, size of operator, pointer operators (& and \*) and member selection operators (. and ->).* The **sizeof** and the **comma** operators are discussed here. The remaining operators are discussed in forth coming chapters.

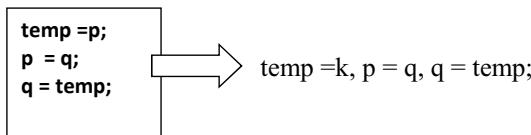
**The Comma Operator:** The comma operator can be used to link related expressions together. It is used to combine many different statements into a single statement. A comma-linked list of expressions is evaluated left to right and value of right most expression is the value of the combined expression.

The multiple declarative statements can be written in single statement as illustrated in the examples below:

**Example a:**



**Example b :**



**Evaluation of the expression with comma operator:** Consider the expression `Z = (x=6,y=2,x+y);` the evaluation of the expression takes place as follows :

```
x = 6;  
y = 2;  
Z = x + y = 8;
```

**The `sizeof()` Operator :** The `sizeof()` operator is an operator which gives the number of bytes allocated for a variable or a data type. Its general form is

`sizeof(datatype/variable);`

**Example1:** `sizeof(char)` will give 1.

**Example2:** `int b; sizeof(b)` will give 2.

The table below describes the special operators & and \*

**Table 1.15 : Special Operator**

Sl.no	Operators	Description
1	&	This is used to get the address of the variable. <b>Example :</b> &a will give address of a.
2	*	This is used as pointer to a variable. <b>Example :</b> *a where, * is pointer to the variable a.

**Associativity and Precedence of Operator:** The *operator precedence* and *associativity rules* specify the order in which operators in an expression are bound to the operands.

**Precedence of Operators:** The order in which different operators are used to evaluate an expression is called *precedence of operators or hierarchy of operators*. The operators are evaluated from highest precedence to least precedence. The precedence of arithmetic operators is as illustrated below. i.e. \* / and % have highest priority and +, - have least priority.

Operators	Precedence/Priority
*	1
/	1
%	1
+	2
-	2

**Example:** Consider an expression  $10+20*5$ . The evaluation of the expression follows the steps as illustrated below:

$$\begin{aligned}
 X &= 10 + 20*5 \quad /* \text{ Since the precedence of } * \text{ is highest } */ \\
 &= 10 + 100 \quad /* \text{ Since the precedence of } + \text{ operator is least } */ \\
 &= 110
 \end{aligned}$$

**Associativity of Operators :** Whenever two or more operators having the same *precedence or priority* occurs in the expression , then the direction order chosen (*left to right or right to left* ) to evaluate an expression is called associativity of operators. They are classified into two categories based on the direction of evaluation chosen as below:

**1. Left to Right Associativity:** Here the direction order chosen for evaluation is left to right. **Ex:** Arithmetic operators, Logical Operators and Relational operators follows left to right associativity.

**2. Right to Left Associativity:** Here the direction order chosen for evaluation is right to left. **Ex:** Assignment operator, unary operators and conditional expression, etc., have right to left associativity.

**Example:** Consider an expression  $x=8/4*6$  .Here since both / and \* has the same priority, they are evaluated based on *Left to Right* associativity.

$$\begin{aligned} X &= 8/4*6 \quad / \text{ will be evaluated based on left to right associativity} \\ &= 2*6 \\ &= 12 \end{aligned}$$

#### **Note : Precedence and Associativity Rules**

1. Precedence rules decides the order in which different operators are applied.
2. Associativity rule decides the order in which multiple occurrences of the same level operator are applied.

The list of operators supported in C language along with their precedence and priority is illustrated in Table **1.16**.

**Table 1.16 : C Operators Precedence Table**

Operator	Description	Precedence or Priority	Associativity
( ) []	Parentheses (function call) Brackets (array subscript)	1	left-to-right
+	Unary Plus		
-	Unary Minus		
++	Increment		
--	Decrement		
!	Logical Negation		
&	Address		
sizeof	Size of an object		
(type)	Type Cast (conversion)		
* / %	Multiplication/division/modulus	3	left-to-right
+ -	Addition/subtraction	4	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	5	left-to-right
< <=	Relational less than/less than or equal to		
> >=	Relational greater than/greater than or equal to	6	left-to-right
== !=	Relational is equal to/is not equal to	7	left-to-right
&	Bitwise AND	8	left-to-right
^	Bitwise exclusive OR	9	left-to-right
	Bitwise inclusive OR	10	left-to-right
&&	Logical AND	11	left-to-right
	Logical OR	12	left-to-right
? :	Ternary Conditional	13	right-to-left
= += -= *= /= %=&=&^=& = <<=>>=	Assignment Operators	14	right-to-left
,	Comma (Separates expressions)	15	left-to-right

**C Expressions:** A sequence of operand and operators that reduces to a single value after evaluation is an expression. The different types of C expressions are:

1. **Arithmetic Expression:** Arithmetic expression is a combination of variables, constants and arithmetic operators arranged as per the syntax of the language. They simplify to a single value, when evaluated. In the expression  $x+y$ ,  $x$  and  $y$  are called operands and  $+$  is the operator. The operands in the expression can hold integer, floating point or double values. Based on the

type of operands in the expression, there are two modes of arithmetic expressions:

#### A. Single Mode Arithmetic Expressions:

A *single mode* arithmetic expression is an expression all of whose operands are of the same type (*i.e.* **INTEGER** or **FLOAT** or **DOUBLE**). In single mode arithmetic expressions, the data type of the result of an operation is identical to that of the operands.

- a. **Integer Mode Arithmetic Operations:** Here all the operands are of integer data type and the result obtained will also be of type integer.

##### Examples:

- $1 + 3$  is **4**
- $3 * 8$  is **24**
- $12/4$  is **3**
- $13/4$  is **3** rather than **3.25**. Since  $13/4$  is a single mode arithmetic expression and since all of its operands are of **INTEGER** type, the result must also be of **INTEGER** type. The computer will *truncate* the mathematical result (3.25) making it an integer. Therefore, the result is **3**.
- $3/5$  is **0** rather than **0.6**

- b. **Floating Mode Arithmetic Operations:** Here all the operands are of float data type and the result obtained will also be of float data type.

##### Examples:

- $1.23 - 0.45 = 0.780000$
- $6.5/1.25 = 5.200000$
- $8.4/4.2 = 2.000000$

**Rules for Evaluating Expressions:** The following are rules for evaluating a more complicated single mode arithmetic expression:

- Expressions are always evaluated from *left to right*
- If an operator is encountered in the process of evaluation, its priority is compared with that of the next one:
  - If the next one is lower, evaluate the current operator with its operands
  - $4 * 5 - 3$

In the above expression, in the left to right scan, operator \* is encountered first. Since the operator - is lower,  $4 * 5$  is evaluated first transforming the given expression to  $20 - 3$ . Hence, the result is 17.

- o If the two operators have the same precedence, the associativity rules are used to determine which one should be evaluated. For example, if both the current and the next operators are \*, then  $2 * 8 * 7$  will be evaluated as  $(2 * 8) * 7$ .
- o If the next one is higher than the current, the scan should continue with the next operator. For example, consider the following expression:
- o  $4 + 5 * 7$ . Here since \* has the highest precedence, it will be evaluated first resulting  $4 + 35$ . Thus the result of the expressions  $4 + 5*7 = 39$ .

**Example 1:** In the following examples, brackets are used to indicate the order of evaluation. The result is 15 rather than 15.333333 since the operands are all integers.

```
2 * 4 * 5 / 3 + 2
--> [2 * 4] * 5 / 3 + 2
--> 8 * 5 / 3 + 2
--> [8 * 5] / 3 + 2
--> 40 / 3 + 2
--> 40 / 3 + 2
--> 13 + 2
--> 15
```

**Example 2:** As in mathematics, sub expressions in parenthesis must be evaluated first.

```
100 + (1 + 250 / 100) * 3
--> 100 + (1 + [250 / 100]) * 9
--> 100 + (1 + 2) * 9
--> 100 + ([1 + 2]) * 9
--> 100 + 3 * 9
--> 100 + [3 * 9]
--> 100 + 27
--> 127
```

**Example 3 :** Floating Point Arithmetic Expression

```

1.000000 + 2.000000 * 3. 000000 / (6. 000000*6. 000000 + 5. 000000*4. 000000)
--> 1. 000000 + 2. 000000 * 3. 000000 / (6. 000000*6. 000000 + 5. 000000*4.
000000) + 0.250000
--> 1. 000000 + 2. 000000 * 3. 000000 / (36. 000000+ 5. 000000*4. 000000) +
0.250000
--> 1. 000000 + 2. 000000 * 3. 000000 / (36. 000000 + 20. 000000]) + 0.250000
--> 1. 000000 + 2. 000000 *3. 000000/ 50. 000000 + 0.250000
--> 1. 000000 + 6. 000000 / 50. 000000 + 0.250000
--> 1. 000000 + 0.120000 + 0.250000
--> 1.120000 + 0.370000
--> 1.490000

```

## B. Mixed mode Arithmetic Expressions :

The expressions which contains both INTEGER and REAL (FLOAT or DOUBLE) constants or variables, are called mixed mode arithmetic expression. In mixed mode arithmetic expressions, INTEGER operands are always converted to REAL before carrying out any computations. As a result, the result of a mixed mode expression is of REAL type. The following is a table showing this fact.

Operand	INTEGER	REAL
INTEGER	INTEGER	REAL
REAL	REAL	REAL

The rules for evaluating mixed mode arithmetic expressions are simple:

- Use the rules for evaluating *single mode arithmetic expressions* for scanning.
- After locating an operator for evaluation, do the following:
  - If the operands of this operator are of the same type, compute the result of this operator.
  - If one of the operand is an integer while the other is a real number convert the integer to a real and compute the result.  
(or *Apply the rule of implicit type conversion discussed later in this chapter*)

### Simple Examples:

- 1 + 2.5 is 3.5
- 1/2.0 is 0.5
- 2.0/8 is 0.25

**More Complicated Examples:** In the following, brackets will be used to indicated the order of evaluation and braces will be used to indicated an integer-to-real conversion.

```
5 * (11.0 - 5)* 2 / 4 + 9
--> 5 * (11.0 - {5}) * 2 / 4 + 9
--> 5 * (11.0 - 5.0) * 2 / 4 + 9
--> 5 * ([11.0 - 5.0]) ** 2 / 4 + 9
--> 5 * 6.0 * 2 / 4 + 9
--> [5 * 6.0] * 2 / 4 + 9
--> [30.0 * 6.0] / 4 + 9
--> 180.0 / {4} + 9
--> 180.0 / 4.0 + 9
--> 45.0 + {9}
--> 45.0 + 9.0
--> 54.0    // 54.000000 (with precision 6)
```

### Programming Example :

```
#include<stdio.h>
#include<conio.h>
main( )
{ float n1,n2,n3,r1,r2,r3;
clrscr();
n1 = 7;
n2 = 10;
n3 = 5;
r1 = n1-n2/3 +n3*2 - 1;
r2 = n1- n2/(3+n3)*(2-1);
r3= n1- (n2/(3+ n3)*2) -1 ;
printf("\n r1 = %f\n",r1);
printf("\n r2 = %f\n",r2);
printf("\n r3 = %f\n",r3);
getch();
}
```

### Output :

```
r1 = 12.666667
r2= 5.750000
r3=3.5000000
```

**2. Relational Expression:** The expression that contains relational operators, variables and constants is called relational expression. The resultant value of any relational expression is either true or false.

**Example:**  $(a+b) > c$  and  $(b*b - 4*a*c) == 0$

**3. Logical Expression:** It is a combination of logical operators and relational expression. The resultant value of any relational expression is also TRUE or FALSE.

**Example:**  $(a==b) \&\& (b==c)$

**4. Assignment Expression:** The expression that contains assignment operator, variables and constants is called assignment expression.

**Example:**  $x=a + b;$

**5. Short hand assignment expression:** The expression that contains shorthand assignment operator, variables and constants is called short hand assignment expression.

**Example :**  $a+=10;$

**Conversion of Algebraic Expressions into C Expressional** the mathematical expressions cannot be written as it is in C. The mathematical expressions have to be converted into appropriate C Expressions. The table 1.17 below shows some of the mathematical expressions and their equivalent C expression:

**Table 1.17 : Conversion of Mathematical Expression**

<b>Mathematical Expressions</b>	<b>C Equivalent Expressions</b>
Sum = $x+y$	Sum=x+y
Product = $x*y$	Product = x*y
Difference = $x-y$	Difference =x-y
Quotient = $\frac{x}{y}$	Quotient = x/y
$S = \frac{(side1+side2+side3)}{2}$	$S = (side1+side2+side3)/2$
$Area = \sqrt{s(s - a)(s - b)(s - c)}$	$Area = \text{sqrt}(s*(s-a)*(s-b)*(s-c))$
$X = e^{-i\omega t}$	$X = \exp(-i*\omega*t)$
$A = \pi r^2$	$A = \pi*r*r$
$a^2 + b^2 = c^2$	$c*c= a*a + b*b$
$\frac{\delta y}{\delta x}$	deltay/deltax
$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$	$(-b+\text{sqrt}(b*b-4*a*c))/(2*a)$
$\sqrt{a^2 + b^2}$	$\text{sqrt}(a*a+b*b)$
$f(x) = a_0 + \left( a_n \cos \frac{n\pi x}{L} + b_n \sin \frac{n\pi x}{L} \right)$	$fx=a0+(an*\cos(n*PI*x)/L + bn*\sin(n*PI*x)/L)$
$\tan \theta = \frac{\sin \theta}{\cos \theta}$	$\tan(theta) =\sin(theta) / \cos(theta)$
$\log_{10}(x + y)$	$\log10(x+y)$
$sum = 1 + \frac{nx}{1!} + \frac{n(n - 1)x^2}{2!} + \dots$	$sum = 1+n*x/\text{fact}(1) +n*(n-1)*\text{pow}(x,2)/\text{fact}(2) + -----$
$\cos^{-1} \frac{x}{y}$	$\text{acos}(x/y)$
$\sinh^{-1} x$	$\text{asinh}(x)$
$\tanh(xy)$	$\tanh(x*y)$
$\sqrt[3]{x}$	$\text{pow}(x,1/3)$
$y = x^{35} + x^{45}$	$y=\text{pow}(x,35) + \text{pow}(x,45)$

**Mathematical Functions:** Mathematical functions supported by C programming language is listed in the table below :

**Table 1.18 : C Mathematical Functions**

<b>C- Function</b>	<b>Meaning in Mathematics</b>
acos(x)	$\cos^{-1} x$
asin(x)	$\sin^{-1} x$
atan(x)	$\tan^{-1} x$
atan2(x,y)	$\tan^{-1} x / y$
cos(x)	$\cos x$
sin(x)	$\sin x$
tan(x)	$\tan x$
cosh(x)	$\cosh x$
sinh(x)	$\sinh x$
tanh(x)	$\tanh x$
ceil(x)	x rounded up to the nearest integer
exp(x)	$e^x$
fabs(x)	absolute value of x $ x $
floor(x)	x rounded down to the nearest integer
fmod(x,y)	remainder of x/y
log(x)	natural log of x and x>0
log10(x)	base 10 log of x, x>0
pow(x,y)	x to the power y
sqrt(x)	$\sqrt{x}$ where $x \geq 0$

**Evaluation of Expression:** In order to evaluate any given C Expression one must use the rules described below:

#### **Rules for Evaluating Expressions**

- a. **Parentheses Rule:** All expressions in parentheses must be evaluated separately. Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.
- b. **Operator precedence Rule :** Operators in the same expression are evaluated in the following order :
  1. Unary Operator
  - 2.Arithmetic Operators
  - 3.Left Shift and Right Shift Operators
  - 4.Relational Operators
  - 5.Equality and In Equality Operators
  - 6.Bitwise Operators
  - 7.Logical Operators
  - 8.Ternary Operators
  - 9.Assignment Operators and
  - 10.Comma Operators.

- c. **Associativity Rule** : Whenever two or more operators having the same precedence appear in the same expression then they must be evaluated on the basis of *Left to Right Associativity and Right to Left associativity* . Unary operators, Ternary operators and Assignment operators are evaluated on the basis of Right to Left Associativity. Whereas the remaining all category of operators are evaluated on the basis of Right to Left Associativity

**Example 1:**The expression  $a+2>b||!c\&\& a==d||a-2 <=e$  (where  $a=11$ ,  $b=6$   $c=0,d=7$  and  $e=5$ ) will be evaluated as illustrated below by applying the associativity and precedence rules.

Expression	Operator Evaluated
$11+2>6  !0\&\&11==7  11-2<=5$	[Operator !]
$11+2>6  1\&\&11==7  11-2<=5$	[Operator + ]
$13>6  1\&\&11==7  11-2<=5$	[Operator - ]
$13>6  1\&\&11==7  9<=5$	[Operator >]
$1  1\&\&11==7  9<=5$	[Operator <=]
$1  1\&\&11==7  0$	[Operator == ]
$1  1\&\&0  0$	[Operator \&\&]
$1  0  0$	[Operator    ]
1	

**Example 2:**The expression  $10!=10||5<4\&\&8$  will be evaluated as follows

$10!=10  5<4\&\&8$	[Operator !=]
$10!=10  0\&\&8$	[Operator \&\&]
$0  0\&\&8$	[Operator \&\&]
$0  0$	[Operator    ]
0	

**Example 3:** If  $a=5$ ,  $b=7$  then the value of  $c$  in the following expression is

i)  $c = ++a + b = 6 + 7 = 13$

ii)  $c = a++ + b = 5 + 7 = 12$

**Example 4:** If  $a = 100$ ,  $b = 200$ ,  $c = 300$  then the value of  $d$  in the following expression is

i.  $d = ++a + ++b + ++c = 101 + 201 + 301 = 603$  // all are pre incremented

ii.  $d = a++ + b++ + c++ = 100 + 200 + 300 = 600$  // all are post incremented

**Example 5:** If  $i=3, j=4, k=2$  then the value of  $m$  in the following expression is

i)  $m = i++ - j-- = 3 - 4 = -1$  // post incrementation and pre decrementation

ii)  $m = ++k \% --j = 3 \% 3 = 0$  // pre incrementation and pre decrementation

iii)  $m = j++ / i-- = 4 / 3 = 1$  // post incrementation and pre decrementation

**Type Conversion:** The process of converting the data from one data type to another data type is called type conversion. In C programming language there are two types of type conversion:

i. Implicit Conversion and

ii. Explicit Conversion

i. **Implicit Type Conversion:** The process of conversion of one data type to another data type automatically by C compiler is called implicit type conversion. During the type conversion if the operands are of different types the lower type is automatically converted to the higher type before operation proceeds.

The type conversion rules or hierarchy that is used for evaluating C expression involving all data types is as follows:

*short/char -> int -> unsigned int -> long int -> unsigned long int -> float -> double -> long double .*

**Explanation:** Consider the expression containing two operands.

- If one of the operand is *char* and the other operand is *int*. Then the *char* operand will be converted automatically into *int*.
- Suppose if one of the operand is *int* and the other is *double* then the *int* operand is converted into *double*.

**Example 1:**  $x = 4.0/3$  (Implicit Conversion of 3 into 3.0)

$$\begin{aligned} &= 4.0/3.0 \\ &= 1.333333 \end{aligned}$$

**Example 2:**    int p = 7;  
              float q=2.0;  
              float r;

$$\begin{aligned} r &= p/q \\ &= 7/2.0 \quad (\text{Implicit Conversion of 7 into 7.0}) \\ &= 7.0/2.0 \\ &= 3.50000 \end{aligned}$$

- ii. **Explicit type Conversion:** The process of forcible conversion of data type from one data type to another data type is called explicit type conversion or explicit type casting a value. The general form or syntax of explicit type conversion is as given below:

*(type\_name) expression ;*

**Example 1:**

X = (int) 8.5 = 8      (Explicit conversion of 8.5 into 8 )

Y = (int)22.3 / (int) 4.5 = 22/4 = 5

(Explicit conversion of 22.3 into 22 and 4.5 into 4)

A= (double) sum/N    (Explicit conversion of sum into double)

P = (int) (x+y)      (Explicit conversion of x + y into int)

Q = (int)a +b        (Explicit conversion of a into int)

R = sin(double(x))    (Explicit conversion of x into double)

**Example 2:**

```
int    p = 7;  
int    q = 2 ;  
float  r;
```

r= (float)p/q;

= (float)7/2 ( Explicit conversion of 7 int 7.0)

= (7.0)/2.0 (Implicit conversion f 2 into 2.0)

=3.500000

## Additional Concepts

**Type Qualifiers :** There are two types of qualifiers available in C language. They are:

1. const
2. volatile

**1. const keyword:** Constants are also like normal variables. But, only difference is, their values can't be modified by the program once they are defined. They refer to fixed values. They are also called as literals. They may be belonging to any of the data type.

- **Syntax:** `const data_type variable_name;` (or)

```
const data_type *variable_name;
```

**Example : Find the output of the following program**

```
#include<stdio.h>
int main()
{
    const int i= 100;
    i =300;
    printf("\n Value of i is %d ",i)
    return 0;
}
```

### **Output :**

Compile Time Error. As variable is declared as const integer type variable and in the next statement we are changing its value . As constant variable cannot be changed , therefore compile time errors occurs.

**2. volatile keyword :** When a variable is defined as volatile, the program *may not change* the value of the variable explicitly. But, these variable values *might keep* on changing without any explicit assignment by the program. These types of qualifiers are called volatile.

For example, if global variable's address is passed to clock routine of the operating system to store the system time, the value in this address keep on changing without any assignment by the program. These variables are named as volatile variable.

### Syntax:

```
volatile data_type variable_name; (or)  
volatile data_type *variable_name
```

**Example :** Find the output of the following program

```
#include<stdio.h>  
int main()  
{  
    volatile int i= 100;  
    printf("\n Value of i is = %d ",i)  
    i = 300;  
    printf("\n Modified value of i is =%d",i);  
    return 0;  
}
```

### Output :

```
Value of i is = 100  
Modified value of i is = 300
```

### Interesting Facts about Bitwise Operators in C :

- 1) **The left shift and right shift operators should not be used for negative numbers.** They must be used for only unsigned numbers or positive numbers where the MSB is not treated as sign bit. Also, if the number is shifted more than the size of integer, the behavior is undefined.
- 2) **The bitwise XOR operator is the most useful operator from technical interview perspective.** It is used in many problems. A simple example could be “Given a set of numbers where all elements occur even number of times except one number, find the odd occurring number” This problem can be efficiently solved by just doing XOR of all numbers.

```
// Function to return the only odd occurring element  
int findOdd(int arr[], int n) {  
    int res = 0, i;  
    for (i = 0; i < n; i++)
```

```

        res ^= arr[i];
        return res;
    }
int main(void) {
    int arr[] = {12, 12, 14, 90, 14, 14, 14};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf ("The odd occurring element is %d ", findOdd(arr, n));
    return 0;
}

```

**Output: The odd occurring element is 90**

**Note :** The following are many other interesting problems which can be used using XOR operator: *Find the Missing Number, swap two numbers without using a temporary variable, A Memory Efficient Doubly Linked List, and Find the two non-repeating elements.*

### 3) The bitwise operators should not be used in-place of logical operators.

The result of logical operators (`&&`, `||` and `!`) is either 0 or 1, but bitwise operators return an integer value. Also, the logical operators will consider any non-zero operand as 1. For example consider the following program, the results of `&` and `&&` are different for same operands.

```

int main()
{
    int x = 2, y = 5;
    (x & y)? printf("True ") : printf("False ");
    (x && y)? printf("True ") : printf("False ");
    return 0;
}

```

**Output: False True**

- 4) The left-shift and right-shift operators are equivalent to multiplication and division by 2 respectively.**

As mentioned in point 1, it works only if numbers are positive.

```
int main()
{
    int x = 19;
    printf("x << 1 = %d\n", x << 1);
    printf("x >> 1 = %d\n", x >> 1);
    return 0;
}
```

**Output: 38**

**9**

- 5) The & operator can be used to quickly check if a number is odd or even :** The value of expression ( $x \& 1$ ) would be non-zero only if  $x$  is odd, otherwise the value would be zero.

```
int main()
{
    int x = 19;
    (x & 1)? printf("Odd"): printf("Even");
    return 0;
}
```

**Output: Odd**

- 6) The ~ operator should be used carefully :** The result of  $\sim$  operator on a small number can be a big number if result is stored in a unsigned variable. And result may be negative number if result is stored in signed variable (assuming that the negative numbers are stored in 2's complement form where leftmost bit is the sign bit)

```
// Note that the output of following program is compiler
// dependent
int main()
{
    unsigned int x = 1;
    printf("Signed Result %d \n", ~x);
    printf("Unsigned Result %ud \n", ~x);
```

```
    return 0;  
}
```

### Output:

Signed Result	-2
Unsigned Result	65534d

7) The & bitwise operator is suitable for turning off a particular bit in a number

8) The || bitwise operator is suitable for turning on a particular bit in a number .

9) The & bitwise operator is suitable for checking whether a particular bit is on or off.

10) Left shifting is multiplying by  $2^K$  . If you shift left on an unsigned int by K bits, this is equivalent to multiplying by  $2^K$  . Shifting left on signed values also works, but overflow occurs when the most significant bit changes values (from 0 to 1, or 1 to 0).

11) One can swap two numbers using XOR operators :

```
x = x ^ y ;  
y = x ^ y ;  
x = x ^ y ;
```

## Uses of Bitwise Operations

**1. Compression:** Bitwise operators can be used for compressing the data. For example , occasionally, one may want to use large number of Boolean variables in a program, without using a lot of space. In such case a 32-bit int can be used to store 32 Boolean variables. Using appropriate Boolean operators the data can be manipulated accordingly. Normally, the minimum size for one Boolean variable is one byte. All types in C must have sizes that are multiples of bytes. However, only one bit can be used to represent a Boolean value.

**2. Set operations:** The Bitwise operators can be used for representing the elements of set and performing set operations like intersection and union. If a bit is 1, then element say i is in the set, otherwise it's not. One can use bitwise AND to implement set intersection, bitwise OR to implement set union.

**3. Encryption:** The Bitwise operators can be used for encrypting the data . For example, swapping the bits of a string according to a predefined shared key will create an encrypted string.

## **Errors**

Errors in C program can be classified into 1. Compiler/Syntax Errors 2. Linker Errors and 3. Logical Errors

**1. Syntax Errors :** If the program does not follow the syntax of the language , the compiler will raise errors and halts the compilation process. Some of the typical scenarios when compiler raises syntax errors are :

- Missing ‘;’ at the end of statements
- Missing curly braces or parentheses
- Typographical errors , spelling mistakes , invalid keywords
- Improper case used for keywords
- Using a variable without declaring it
- Trying to assign a different type of data for a particular data type.

**2. Linker Errors :** These type of errors typically arise when it is not able to find a particular piece of code that is required for your program to be created. **Some of the typical scenarios when linker raises errors are :**

- Variable or function or header files referenced , but not present or defined anywhere in code.
- Mismatch between the definition of a function and its actual implementation in code.

**3. Logical errors :** An error in a program that makes it do something other than what the programmer is intended to do. Some of the typical scenarios when logical errors arises are :

- a. Writing wrong logic or wrong formulas
- b. Improper testing condition in the loop
- c. Usage of null statement unnecessarily

## ASCII Table (source:ascii.com)

ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. Below is the list of the ASCII character tables.

1. **ASCII control characters (Character code 0-31)** : The first 32 characters in the ASCII-table are unprintable control codes and are used to control peripherals such as printers.

DEC	OCT	HEX	BIN	Symbol	Description
0	000	00	00000000	<b>NUL</b>	Null char
1	001	01	00000001	<b>SOH</b>	Start of Heading
2	002	02	00000010	<b>STX</b>	Start of Text
3	003	03	00000011	<b>ETX</b>	End of Text
4	004	04	00000100	<b>EOT</b>	End of Transmission
5	005	05	00000101	<b>ENQ</b>	Enquiry
6	006	06	00000110	<b>ACK</b>	Acknowledgment
7	007	07	00000111	<b>BEL</b>	Bell
8	010	08	00001000	<b>BS</b>	Back Space
9	011	09	00001001	<b>HT</b>	Horizontal Tab
10	012	0A	00001010	<b>LF</b>	Line Feed
11	013	0B	00001011	<b>VT</b>	Vertical Tab
12	014	0C	00001100	<b>FF</b>	Form Feed
13	015	0D	00001101	<b>CR</b>	Carriage Return
14	016	0E	00001110	<b>SO</b>	Shift Out / X-On
15	017	0F	00001111	<b>SI</b>	Shift In / X-Off
16	020	10	00010000	<b>DLE</b>	Data Line Escape
17	021	11	00010001	<b>DC1</b>	Device Control 1 (oft. XON)
18	022	12	00010010	<b>DC2</b>	Device Control 2
19	023	13	00010011	<b>DC3</b>	Device Control 3 (oft. XOFF)

20	024	14	00010100	<b>DC4</b>	Device Control 4
21	025	15	00010101	<b>NAK</b>	Negative Acknowledgement
22	026	16	00010110	<b>SYN</b>	Synchronous Idle
23	027	17	00010111	<b>ETB</b>	End of Transmit Block
24	030	18	00011000	<b>CAN</b>	Cancel
25	031	19	00011001	<b>EM</b>	End of Medium
26	032	1A	00011010	<b>SUB</b>	Substitute
27	033	1B	00011011	<b>ESC</b>	Escape
28	034	1C	00011100	<b>FS</b>	File Separator
29	035	1D	00011101	<b>GS</b>	Group Separator
30	036	1E	00011110	<b>RS</b>	Record Separator
31	037	1F	00011111	<b>US</b>	Unit Separator

2. **ASCII printable characters (character code 32-127) :** Codes 32-127 are common for all the different variations of the ASCII table, they are called printable characters, represent letters, digits, punctuation marks, and a few miscellaneous symbols. Almost all characters will be present on the keyboard. Character 127 represents the command DEL.

<b>DEC</b>	<b>OCT</b>	<b>HEX</b>	<b>BIN</b>	<b>Symbol</b>	<b>Description</b>
32	040	20	00100000		Space
33	041	21	00100001	!	Exclamation mark
34	042	22	00100010	"	Double quotes (or speech marks)
35	043	23	00100011	#	Number
36	044	24	00100100	\$	Dollar
37	045	25	00100101	%	Procenttecken
38	046	26	00100110	&	Ampersand
39	047	27	00100111	'	Single quote
40	050	28	00101000	(	Open parenthesis (or open bracket)
41	051	29	00101001	)	Close parenthesis

					(or close bracket)
42	052	2A	00101010	*	Asterisk
43	053	2B	00101011	+	Plus
44	054	2C	00101100	,	Comma
45	055	2D	00101101	-	Hyphen
46	056	2E	00101110	.	Period, dot or full stop
47	057	2F	00101111	/	Slash or divide
48	060	30	00110000	0	Zero
49	061	31	00110001	1	One
50	062	32	00110010	2	Two
51	063	33	00110011	3	Three
52	064	34	00110100	4	Four
53	065	35	00110101	5	Five
54	066	36	00110110	6	Six
55	067	37	00110111	7	Seven
56	070	38	00111000	8	Eight
57	071	39	00111001	9	Nine
58	072	3A	00111010	:	Colon
59	073	3B	00111011	;	Semicolon
60	074	3C	00111100	<	Less than (or open angled bracket)
61	075	3D	00111101	=	Equals
62	076	3E	00111110	>	Greater than (or close angled bracket)
63	077	3F	00111111	?	Question mark
64	100	40	01000000	@	At symbol
65	101	41	01000001	A	Uppercase A
66	102	42	01000010	B	Uppercase B
67	103	43	01000011	C	Uppercase C
68	104	44	01000100	D	Uppercase D
69	105	45	01000101	E	Uppercase E
70	106	46	01000110	F	Uppercase F
71	107	47	01000111	G	Uppercase G
72	110	48	01001000	H	Uppercase H
73	111	49	01001001	I	Uppercase I

74	112	4A	01001010	<b>J</b>	Uppercase J
75	113	4B	01001011	<b>K</b>	Uppercase K
76	114	4C	01001100	<b>L</b>	Uppercase L
77	115	4D	01001101	<b>M</b>	Uppercase M
78	116	4E	01001110	<b>N</b>	Uppercase N
79	117	4F	01001111	<b>O</b>	Uppercase O
80	120	50	01010000	<b>P</b>	Uppercase P
81	121	51	01010001	<b>Q</b>	Uppercase Q
82	122	52	01010010	<b>R</b>	Uppercase R
83	123	53	01010011	<b>S</b>	Uppercase S
84	124	54	01010100	<b>T</b>	Uppercase T
85	125	55	01010101	<b>U</b>	Uppercase U
86	126	56	01010110	<b>V</b>	Uppercase V
87	127	57	01010111	<b>W</b>	Uppercase W
88	130	58	01011000	<b>X</b>	Uppercase X
89	131	59	01011001	<b>Y</b>	Uppercase Y
90	132	5A	01011010	<b>Z</b>	Uppercase Z
91	133	5B	01011011	<b>I</b>	Opening bracket
92	134	5C	01011100	\	Backslash
93	135	5D	01011101	<b>J</b>	Closing bracket
94	136	5E	01011110	<sup>^</sup>	Caret - circumflex
95	137	5F	01011111	<u>_</u>	Underscore
96	140	60	01100000	<sup>`</sup>	Grave accent
97	141	61	01100001	<b>a</b>	Lowercase a
98	142	62	01100010	<b>b</b>	Lowercase b
99	143	63	01100011	<b>c</b>	Lowercase c
100	144	64	01100100	<b>d</b>	Lowercase d
101	145	65	01100101	<b>e</b>	Lowercase e
102	146	66	01100110	<b>f</b>	Lowercase f
103	147	67	01100111	<b>g</b>	Lowercase g
104	150	68	01101000	<b>h</b>	Lowercase h
105	151	69	01101001	<b>i</b>	Lowercase i
106	152	6A	01101010	<b>j</b>	Lowercase j
107	153	6B	01101011	<b>k</b>	Lowercase k
108	154	6C	01101100	<b>l</b>	Lowercase l

109	155	6D	01101101	<b>m</b>	Lowercase m
110	156	6E	01101110	<b>n</b>	Lowercase n
111	157	6F	01101111	<b>o</b>	Lowercase o
112	160	70	01110000	<b>p</b>	Lowercase p
113	161	71	01110001	<b>q</b>	Lowercase q
114	162	72	01110010	<b>r</b>	Lowercase r
115	163	73	01110011	<b>s</b>	Lowercase s
116	164	74	01110100	<b>t</b>	Lowercase t
117	165	75	01110101	<b>u</b>	Lowercase u
118	166	76	01110110	<b>v</b>	Lowercase v
119	167	77	01110111	<b>w</b>	Lowercase w
120	170	78	01111000	<b>x</b>	Lowercase x
121	171	79	01111001	<b>y</b>	Lowercase y
122	172	7A	01111010	<b>z</b>	Lowercase z
123	173	7B	01111011	{	Opening brace
124	174	7C	01111100		Vertical bar
125	175	7D	01111101	}	Closing brace
126	176	7E	01111110	~	Equivalency sign - tilde
127	177	7F	01111111		Delete

3. **The extended ASCII codes (Character code 128-255) :** There are several different variations of the 8-bit ASCII table. One of the variation is ISO 8859-1, also called ISO Latin-1. Codes 128-159 contain the Microsoft Windows Latin-1 extended characters.

#### Example 1 : C Program to generate ASCII value of all characters

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i;
    clrscr();
    for(i=0;i<256;i++)
        printf(" ASCII value of %c is = %d \n",i,i);
    getch();
    return 0;
}
```

```
}
```

### Example 2: C Program to print the ASCII value of the given character

```
#include<stdio.h>
#include<conio.h>
int main
{
    char ch;
    clrscr();
    printf("\n Enter a character:\n");
    scanf("%c",&ch); /* %c is used to read the typed character*/
    printf("\n The ASCII Value of the given character %c is :%d",ch,ch);
    /* %d is used to print the integer (ASCII ) value of the typed character
   */
    getch();
    return 0;
}
```

#### Output:

```
Enter a character
A
The ASCII Value of the given character A is 65
```

## 1.8 Programming Examples

### 1. C program to print the hello world message.

//C hello world example

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

main()
{
    clrscr();
    printf("Hello World\n");
    getch();
}
```

#### Output :

Hello World

### 2. C program to read and print an integer

```
#include <stdio.h>
#include<conio.h>
main()
{
    int a;
    clrscr();
    printf("Enter an integer\n");
    scanf("%d", &a);
    printf("Integer that you have entered is %d\n", a);
    getch();
}
```

#### Output :

Enter an integer

**1975**

Integer that you have entered is 1975

### 3. C program to find the sum of two numbers

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

```
#include<conio.h>
main()
{
    int a, b, c;
    clrscr();
    printf("Enter two numbers to add\n");
    scanf("%d%d",&a,&b);
    c = a + b;
    printf("Sum of entered numbers = %d\n",c);
    getch();
}
```

**Output :**

Enter two numbers to add  
**10 15**  
 Sum of entered numbers = 25

**4. Write a C program to find whether the given number is even or odd**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int n;
    clrscr();
    printf("Enter an integer\n");
    scanf("%d",&n);
    if ( n%2 == 0 ) /* If n is divisible by 2 then remainder is zero */
        printf("Even\n");
    else
        printf("Odd\n");
    getch();
}
```

**Output :**

Enter an integer  
**15**  
 Odd

## 5. C program to find the addition, subtraction and multiplication of two integers

```
#include <stdio.h>
#include<conio.h>
main()
{
    /* first and second are the two operands */
    int first, second, add, subtract, multiply;
    float divide;
    clrscr();
    printf("Enter two integers\n");
    scanf("%d%d", &first, &second);
    add = first + second;
    subtract = first - second;
    multiply = first * second;
    /*(float) will make variable second promoted to take float value */
    divide = first / (float)second; //typecasting
    /* so that first/(float)second becomes a floating point expression */
    printf("Sum = %d\n",add);
    printf("Difference = %d\n",subtract);
    printf("Multiplication = %d\n",multiply);
    printf("Division = %f\n",divide);

    getch();
}
```

### Output :

```
Enter two integers
3 2
Sum = 5
Difference = 1
Multiplication = 6
Division = 1.500000
```

## 6. C program to determine whether the given year is leap year or not.

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

main()
{
```

```

int year;
clrscr();
printf("Enter a year to check if it is a leap year\n");
scanf("%d", &year);

if ( year%400 == 0)
printf("%d is a leap year.\n", year);
else if ( year%100 == 0)
printf("%d is not a leap year.\n", year);
else if ( year%4 == 0 )
printf("%d is a leap year.\n", year);
else
printf("%d is not a leap year.\n", year);
getch();
}

```

### Output :

Enter a year to check if it is a leap year  
2014  
2014 is not a leap year.

**Note 1 : A year is a leap year if**

1. *year is divisible by 4 but not by 100. (year%4==0 && year%100!=0) . Example : 1996 ,1992 and 1998.*
2. *year is divisible by 4 and by 100 and also divisible by 400. ((year%4==0 && year %100==0)&& (year %400==0). Example : 400,800,1200,1600 and 2000*

**Note 2:** Any number which is divisible by 400 is also divisible by 4 and 100.

**7. C program to swap two numbers using third variable.**

```

#include <stdio.h>
#include<conio.h>
main()
{
    int x, y, temp; //temp is the third variable
                    // x and y are the variables to be swapped
    clrscr();
    printf("Enter the value of x and y\n");

```

```

scanf("%d%d", &x, &y);
printf("Before Swapping\n x = %d\n y = %d\n",x,y);
temp = x;
x = y;
y = temp;
printf("After Swapping\nx = %d\ny = %d\n",x,y);
getch();
}

```

**Output :**

```

Enter the value of x and y
2 3
Before Swapping
x = 2
y= 3
After Swapping
x = 3
y = 2

```

## 8. C program to swap two numbers without using third variable.

```

#include<stdio.h>
#include<conio.h>
main()
{
    int a, b;
    clrscr();
    printf("Enter two integers to swap\n");
    scanf("%d%d", &a, &b);
    printf("Before Swapping\n a = %d\n b = %d\n",a,b);
    a = a + b;
    b = a - b;
    a = a - b;
    printf("After Swapping\n a = %d\n b = %d\n",a,b);
    getch();
}

```

**Output :**

```

Enter two integers to swap
2 3
Before Swapping
a = 2
b= 3
After Swapping
a = 3
b = 2

```

## **9. C program to find the area and circumference of a circle.**

```
#include<stdio.h>
#include<conio.h>
main()
{
    float rad,PI = 3.14, area, ci;
    clrscr();
    printf("\nEnter radius of circle: ");
    scanf("%f", &rad);
    area = PI * rad * rad;
    printf("\nArea of circle : %f ", area);
    ci = 2 * PI * rad;
    printf("\nCircumference of a circle: %f ", ci);
    getch();
}
```

### **Output :**

```
Enter radius of circle: 3
Area of circle :28.260000
Circumference of a circle:18.840000
```

## **10.C Program to calculate sum of 5 subjects and to find percentage.**

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

int main()
{
    int s1, s2, s3, s4, s5, sum, total = 500;
    clrscr();
    float per;
    printf("\nEnter marks of 5 subjects : ");
    scanf("%d %d %d %d %d", &s1, &s2, &s3, &s4, &s5);

    sum = s1 + s2 + s3 + s4 + s5;
    printf("\nSum : %d", sum);

    per = (sum * 100) / (float)total; /* To convert into floating
    point */
    printf("\nPercentage : %f", per);
```

```
        getch();
    }
```

### Output :

```
Enter marks of 5 subjects : 75 85 65 92 89
Sum : 406
Percentage : 81.200000
```

**Note :** As sum , 100 and total are integer values, the expression is integer expression which rounds the result to the nearest integer. therefore, we convert total to float ie., 500 to 500.0 in turn we are making the expression as floating point expression, to get accurate value in float

## 11. C Program to convert temperature from degree centigrade to Fahrenheit.

```
#include<stdio.h>
#include<conio.h>
main()
{
    float celsius, fahrenheit;
    clrscr();
    printf("\nEnter temp in Celsius : ");
    scanf("%f", &celsius);

    fahrenheit = (1.8 * celsius) + 32;
    printf("\nTemperature in Fahrenheit : %f ", fahrenheit);

    getch ();
}
```

### Output :

```
Enter temp in Celsius : 100
Temperature in Fahrenheit : 212.000000
```

**12.Identify the syntax errors in the following program**

```
Include<math.h>
Main{}
Float x;
X =25
Y = EXP(x)
/**Printing section */
Print(x,y);
}
```

**Solution :**

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
main()
{ float x,y;
clrscr();
x=25;
y = exp(x);
printf("%f %f ",x,y);
}
```

**13.Evaluate each of the following expressions independent of each other . The declaration and initialization statement is**

int i = 3, j = 4, k = 2 ;

- i)  $i++ - j-- = 3 - 4 = -1$  (since both are post operator ie., use the operand value first and then increment and decrement )
- ii)  $++k \% -j = 3 \% 3 = 0$  (since both are pre operator ie., first increment and decrement and then use the operand value )
- iii)  $j + i / 1 - i = 4 + 3 / 1 - 3 = 4 + 3 - 3 = 4$
- iv)  $j++ / i-- = 4 / 3 = 1$  (since both are post operator ie., use the operand value first and then increment and decrement )

**14.What would be the value of a after the execution of the following expressions ? Assuming the initial value of a =5 .**

- 1)  $a++ = (a++) + (++a)$  // let x = a++ and y = ++a  
 $a++ = (x) + (y)$  // x = 5 and a = 6 (post incrementation)  
 $a++ = 5 + (y)$  // y = ++a = ++6 = 7 , and a = 7 (pre incrementation)  
 $a++ = 5 + 7$   
 $a = a + 12$  // a = 7  
 $a = 19$
- 2)  $a- = (--a) - (a--)$  // let x = --a and y = a--  
 $a- = (x) - (y)$  // x = 4 and a = 4 (because of -a)  
 $a- = 4 - (y)$  // y = a-- = 4-- = 4 and a = 3 (because of a--)  
 $a- = 4 - 4 = 0$   
 $a = a - 0 = 3 - 0 = 3$

**15.Find the final values of the variables in the following program segment**

```
1. int a,b,c;  
    float x,y;  
    a=10;  
    b = 15;  
    c = b/a;  
    x = b/a;  
    y = (float) b/a;
```

**Answer :** c = 15/10 = 1 ( Integer division )

x = 15/10 = 1.000000 ( Implicit type cast , as x is float)  
y = 15.000000/10 (Explicit type cast)  
= 15.000000/10.000000 (Implicit type cast)  
= 1.500000

**Explanation :**

c= 15/10= 1( integer expression)

x=15/10= 1.0( integer expression giving integer result as 1 but as it is equated to float it takes float value as 1.000000

y= (float) b/a= 15.0/10= 1.500000 ( floating point expression. because b is explicitly made as float.)

```
2. int a,b ;  
    float x;  
    a =25/10 + 6.5 = 2 +6.5 = 8; // 8.5 is rounded to 8  
    b = 25/10 +6.6 = 2+6.6 = 8 ; //8.6 is rounded to 8  
    x = 25/10 + 6.6 ;  
        = 2 +6.6  
        = 8.600000 // as x is float
```

**16.Find the values of variable x and m after the execution of the following statements.**

a. x = 15 ;  
m = x++;

**Answer : x =16, and m= 15**

b. x =15;  
m = ++x;

**Answer : x =16 and m =16**

**17.Given the length of a side , write a C – program to compute surface area and volume of a cube.**

```
#include<stdio.h>  
#include<conio.h>  
void main( )  
{  
    float len,s_area,vol;  
    clrscr();  
    printf ("Enter side length of a cube \n");  
    scanf("%f",&len);  
    s_area = 6*len*len;  
    vol = len*len*len;  
    printf("Surface Area is %f\n", s_area);  
    printf("Volume is %f\n",vol);  
    getch();  
}
```

**Output :**

```
Enter side length of a cube  
5  
Surface Area is 150.000000  
Volume is 125.000000
```

**18. Write a function program to evaluate the following investment equation  $V = P(1+r)^n$ .**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float v,p,r,n;
    clrscr();
    float fun (float p, float r,float n);
    printf("\nEnter the value of P :");
    scanf("%f",&p);
    printf("\nEnter the value of r :");
    scanf("%f",&r);
    printf("\nEnter the value of n :");
    scanf("%f",&n);
    v = fun(p,r,n);
    printf("\n V = %f",v);
    getch();
}
float fun(float p,float r,float n)
{
    return (p*pow((1+r),n)); // The C library function pow(x, y)
    returns xy.
}
```

**Output :**

```
Enter the value of P : 1000
Enter the value of r :1.25
Enter the value of n: 2
V = 5062.500000
```

## **19.Program to multiply given number by 4 using bitwise operators .**

```
#include<stdio.h>
#include<conio.h>
void main()
{ int n;
clrscr();
printf("\n Enter an integer number n :");
scanf("%d",&n);
printf("\n Multiply n by 4 using bitwise operators is %d\n",n<<2);
getch();
```

```
}
```

### **Output :**

Enter an integer number n : 5

Multiply n by 4 using bitwise operators is : 20

**Note : To multiply any number with  $2^N$  shift the bits N times to the left.**

Consider the number : **0000 0100 = 4**

If the number 4 has to be multiplied by  $8 = 2^3$  then it has to shift the bits 3 times to left. i.e.,  $8 = (2^3 \rightarrow N = 3) = 3$  bit shift : **0010 0000 = 32**

## **20.What is the output of the following program**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x = 0xad, y = 0x64, z = 78, w ;
    clrscr();
    w = x+y+z+2;
    printf("\n %o %x %d",w,w,w);
    getch();
}
```

### **Output :**

461 131 305

## **21.Program to illustrate the use of Bit wise Operators :**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x ,y;
    clrscr();
    x =78;
    y =456;
    x = x>>3;
    printf(" \n The value of x is = %d \n",x);
    y = y<<7;
    printf(" \n The value of y is = %d\n",y);
    getch();
}
```

### **Output :**

```
The value of x is = 9
The value of y is = 58368d
```

## **22.Find the output of the following program :**

```
#include<stdio.h>
#include<conio.h>
int main()
{
    clrscr();
    printf(" \n Hello!");
    printf(" Welcome to World of Programs!");
    getch();
    return 0;
}
```

### **Output :**

```
Hello! Welcome to World of Programs!
```

## **23.C program to demonstrate the use of bitwise operators**

```

#include<stdio.h>
#include<conio.h>
int main()
{
    unsigned char a = 5, b = 9;      // a = 5(00000101), b = 9(00001001)
    clrscr();
    printf("a = %d, b = %d\n", a, b);
    printf("a&b = %d\n", a&b);    // The result is 00000001
    printf("a|b = %d\n", a|b);    // The result is 00001101
    printf("a^b = %d\n", a^b);    // The result is 00001100
    printf("~a = %d\n", a = ~a);   // The result is 11111010
    printf("b<<1 = %d\n", b<<1); // The result is 00010010
    printf("b>>1 = %d\n", b>>1); // The result is 00000100
    getch();
    return 0;
}

```

### Output:

```

a = 5, b = 9
a&b = 1
a|b = 13
a^b = 12
~a = 250
b<<1 = 18
b>>1 = 4

```

### **24. Find the output of the following program**

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    clrscr();
    int sum ;
    a =5;
    b =3;
    sum = a + -b ;
    printf("\n The sum of a and b is = %d",sum);
    getch();
}

```

### Output

The sum of a and b is = 2

**25.Find the output of the following program**

```
#include<stdio.h>
#include<conio.h>
main()
{ int i,j,k;
clrscr();
k = (i=4,j=5);
printf("k=%d",k);
getch();
}
```

**Output**

```
k = 5
```

**26.Find the output of the following program**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x;
float y;
clrscr();
x = 1000;
y = 800.0 ;
printf("\n Size of x = %d\n", sizeof(x));
printf("\n Size of y = %d\n", sizeof(y));
getch();
}
```

**Output**

```
Size of x = 2
```

```
Size of y = 4
```

**27.What is the output of the following program ? or Write a C program that computes the size of int,float , double and char variables.**

```
#include<stdio.h>
#include<conio.h>
void main()
{ clrscr();
printf("\n%d",sizeof(char));
```

```
    printf("\n%d",sizeof(int));
    printf("\n%d",sizeof(float));
    printf("\n%d",sizeof(double));
    printf("%d",sizeof(200));
    printf("%d",sizeof(2.56));
    getch();
}
Output
1
2
4
8
2
8
```

### 28.Find the output of the following program

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i = 1000;
    printf("\n The value of i is %f",i);
    getch();
    return 0;
}
```

**Output :**  
Floating point linking error

### 29.Find the output of the following program

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int p = 3,q=3 ,r=3,s=3;
    clrscr();
    printf("\n p= %d",p++);
    printf("\n q= %d",++q);
    printf("\n r= %d",r--);
    printf("\n s= %d",--s);
    printf("\n p= %d",p);
    printf("\n q= %d",q);
```

```
    printf("\n r= %d",r);
    printf("\n s= %d",s);
    getch();
    return 0;
}
```

**Output :**

```
p = 3
q= 4
r= 3
s= 2
p= 4
q=4
r=2
s=2
```

**30.Find the output of the following program**

```
#include<stdio.h>
int fun(int a ,int b,int c)
{
    int d =0 ;
    while(a&&b&&c)
    {
        a--;b---;c---;d++;
    }
    return d;
}
int main()
{
    int a =10 ,b= 5, c=30;
    clrscr( );
    printf("\n%d",func(a,b,c));
    getch( );
}
```

**Output :**

```
5
```

**31. Here is list of possible names for variables in C language. Which are valid names and invalid names? If name is invalid, explain why?**

SI.NO	Variables name	Valid or Invalid	Reason if invalid
i	1999_space	Invalid	The first letter of variable must begin with alphabet or _
ii	_apple	Valid	
iii	iNtEL	Valid	
iv	one_2	Valid	
v	for	Invalid	The variable name cannot be a keyword like for
vi	#12	Invalid	The first letter of variable must begin with alphabet or _ and the name must not include any special symbol like #
vii	i.b.m	Invalid	The variable name must not include special character other than _. Here the name contains the special character.
viii	help+me	Invalid	The variable name must not include special character or operator other than _. Here the name contains the operator symbol +.

**32. Write a C program to find the area of a triangle when we know the lengths of all three of its sides.**

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

void main()
{
    int s,a,b,c,area;
    clrscr();
    printf("Enter the values of a, b and c \n");
    scanf("%d %d %d", &a, &b, &c);
    s = (a + b + c) / 2; /* compute s */
    area = sqrt(s * (s - a) * (s - b) * (s - c));
    printf("Area of a triangle = %d \n", area);
    getch();
}
```

**33. Write a C program that computes the size of int, float, double and char variables .**

```
#include<stdio.h>
#include<conio.h>
void main()
{ clrscr();
printf("\n The size of integer is %d\n", sizeof(int));
printf("\n The size of float is %d\n", sizeof(float));
printf("\n The size of double is %d\n", sizeof(double));
printf("\n The size of char is %d\n", sizeof(char));
getch();
}
```

**34. Write a C program which takes as input p, t, r compute the simple interest and display the result.**

**Solution :**

```
#include<stdio.h>
#include<conio.h>
main()
{
float p,t,r,si;
clrscr();
printf("\n Enter the value of p, t and r\n");
scanf("%f %f %f",&p,&t,&r);
si = p*t*r/100;
printf("\n The value of Simple Interest =%f\n",si);
getch();
}
```

## 1.9 Exercises

### Part A : Answer the following

1. What is pseudo code? Explain how pseudo code can be used to solve the problem with suitable example.
2. What is the purpose of pseudocode? What does pseudocode consists of?
3. What is an algorithm? Mention the characteristics of Algorithm.
4. What is Flow Chart? What is the difference between flow chart and algorithm?
5. What is Program ? What is C program?
6. What are header files ? Give Examples
7. Explain why stdio.h and conio.h are used.
8. Mention the Basic Concepts of the C Program. Explain each of the basic concepts of C program with suitable example.
9. Give/Write the basic structure of C program and explain each component of C program with suitable example.
- 10.What is the purpose of a comment ? How does a comment begin and end?
- 11.Which line in the program is an instruction to use standard input/output operations?
- 12.Which two lines will be used in all our programs?
- 13.What is the purpose of a variable?
- 14.In addition to a name, what other pieces of information is associated with each variable in the declaration?
- 15.At any one time how many values can be stored in a single variable? Can that value change? What happens to the old value?
- 16.Define and Give example for the following :
  - a. Variables

- b.** Constants
  - c.** Declaration
  - d.** Data types
  - e.** Assignment Statement
  - f.** Print Statement
- 17.** Which of the following are valid variable names or identifiers?  
Give reason.
- |             |            |                 |             |                       |
|-------------|------------|-----------------|-------------|-----------------------|
| Maximum     | Last.Nam e | First_Name      | n2+n2       | &xyz                  |
| FLOATS      | float      | 4thvar          | xy\$        | _xyz                  |
| sum_product | Row Sum    | Factorial-n     | James_Bon d | 5 <sup>th</sup> Class |
| Xy!         | X,y        | simple_interest | _xy_xyz     | For                   |
| While       | break      | Sum of reverse  | __          | Startend              |
| p qrs       | WIPRO      | 546             | a+b         | for123                |
- 18.** What is a constant? Mention different types of constants in C language?
- 19.** What are data types? Mention the different types of data types supported by C language.
- 20.** Identify which of the following are valid type of int, double or char constants in C and which are not . Identify the data type of each valid constant.
- |         |         |          |               |         |         |          |
|---------|---------|----------|---------------|---------|---------|----------|
| 15      | 'xyx'   | '*'      | \$            | 25.123  | 3.14159 | "Raju"   |
| "X"     | 'TRUE'  | '-5'     | 15.0          | -999    | 122     | 0.005    |
| .123    | 'x'     | 12.0e-12 | 0x123         | 0123    | '\n'    | 12e+5    |
| .12345e | 15e-0.3 | 12.5e.3  | 34,500.<br>99 | 12345.0 | 2.345e2 | +15      |
| 698354L | 25,300  | +6.0E3   | 3.5e-5        | 7.1e4   | -4.5e-2 | 1.5E+2.5 |
| \$2556  | &1234   | 0X7B     | 2.354         | 1.234Z  | 0X8A7   | 0XAB     |
- 21.** What are the Guidelines to be followed to use printf Statement  
. Give Examples?
- 22.** Give the syntax and explain how the printf and scanf can be used in the program with examples.
- 23.** What are printf and scanf statements? Give the difference between them.

- 24.** What is the purpose of a printf?
- 25.** Where are two most common places to send the output from a program? Which of these is the default method?
- 26.** What is the last line of every program?
- 27.** Does the last line need a semicolon?
- 28.** Define and give example for the following: 1. Operator 2. Operand  
3. Expression 4. Precedence 5. Associativity
- 29.** What is an Operator? What are the different types of operators available in C ? Explain all operators with example.
- 30.** Explain the precedence and associativity of operators.
- 31.** What is expression? What are different types of expressions? Give Example.
- 32.** What is Type Conversion? What are the different types of type conversion? Explain with example.
- 33.** Give the Syntax and Example for the following :  
**a.** printf and scanf statement  
**b.** Variable Declaration and Variable initialization  
**c.** Assignment statement and Compound (Shorthand Assignment) statement.
- 34.** Define the following terms with example:  
**a.** Control String or Format string ,  
**b.** Conversion Specifiers
- 35.** Convert the following mathematical expressions into C expressions.  
**a.**  $\frac{ax+b}{ab}$   
**b.**  $\frac{5x+3y}{a+b}$   
**c.**  $C = e^{|x+y-10|}$   
**d.**  $D = x^{25} + y^{35}$   
**e.**  $\frac{2x+3y}{x-6}$   
**f.**  $X = \frac{e^{\sqrt{x}} + e^{\sqrt{y}}}{x \sin \sqrt{y}}$   
**g.**  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

- h.**  $x^5 + 10x^4 + 8x^3 + 4x + 2$   
**i.**  $\tau = \sqrt{\frac{\sigma x - \sigma y}{2}} + \tau xy^2$   
**j.**  $\sin\left(\frac{b}{\sqrt{a^2+b^2}}\right)$   
**k.**  $y = \frac{\alpha+\beta}{\sin\theta} + |x|$   
**l.** Area =  $\pi r^2 + 2\pi rh$   
**m.** Torque =  $\frac{2m_1 m_2}{m_1 + m_2}$   
**n.** Side =  $\sqrt{a^2 + b^2 - 2abc\cos(x)}$   
**o.** Energy = mass[ acceleration xheight +  $\frac{Velocity^2}{2}$  ]  
**p.** C =  $ax^3 + by^6$   
**q.** C =  $\sqrt{\sin^2(x \log_e \sqrt{a+b})}$   
**r.** C =  $\frac{a^{x+y} + b^{x-y}}{x-y}$   
**s.** C =  $\sqrt[3]{\frac{a^{x+y} + b^{x-y}}{x-y}}$   
**t.** C =  $\sqrt[3]{\frac{a^3 + b^4}{a^4 - b^3}}$   
**u.**  $\tan^{-1}(x/y) + 2\cot^{-1}(x^2))$   
**v.**  $\frac{e^{x+y} + e^{-(x+y)}}{\sqrt{x^2+y^2}}$   
**w.** c =  $\cosh^{-1}(x) + \tanh^{-1}(y)$   
**x.** C =  $a^2 + \frac{1}{b + \frac{1}{c+d}}$

**36.**Evaluate the following Expression :

22/7    7/22    22%7    7%22    15/7    -3/16    3/23    -3/2  
-3/-2

**37.**Evaluate the expressions given below .Assume that the variables are initialized as shown below :

- ```

int a=10,b=20;
float p=1.5,q=-2.5;
char c= 'A', d='a';

```
- a.** a+b/5  
**b.** c/a+q  
**c.** a+b%13  
**d.** a%6/b%3  
**e.** ++c+b  
**f.** b%a++

- g.**  $(8*a+b)/b\%3$
- h.**  $++a+b--/2.5$
- i.**  $-a+(-p-q)/4$
- j.**  $a/b+(a/(2*b))$

**38.** Assume that the variables are declared and initialized as follows :

```
int i=2,j=5,x=10;  
float a=1.25,b=5.5,y=1.0;
```

Determine the values of variable x ,y and z in each of the following C assignment expression

- a.**  $x=(float)x-j/2$
- b.**  $y=(y+b)/5$
- c.**  $x=-5*i+j\%3$
- d.**  $x^*=(i+j)-3+3*j/4$
- e.**  $y/=(x+j)/(i*j)\%5$
- f.**  $x=(int)7.5$
- g.**  $x=(int)21.3/(int)4.5$
- h.**  $y=(double)sum/n //Assume sum =1000 n=50$
- i.**  $z=(int)a +b$
- j.**  $p= \cos(double(x)); // Assume x = 60 * \pi /180$

**39.** Evaluate the following expression by specifying the hierarchy and precedence of operators :

- a.**  $100/20<=10-5+100\%10-20==5>=1!=20$
- b.**  $a+2>b\&\&!c||a!=d\&\&a-2<=e$  where  $a=11,b=6,c=0,d=7$  and  $e=5$
- c.**  $a+2>b||!c\&\&a==d||a-2<=e$  where  $a=11,b=6,c=0,d=7$  and  $e=5$
- d.**  $10!=10||5<4\&\&8$

**40.** If  $a=5$  ,  $b=7$  what will be the value of c in the following expression :

- a.**  $c =++a+b$
- b.**  $c=a+++b$

c.  $c = a++ * a++ * a++$

d.  $c = ++a * ++a * ++a$

41. If  $a = 100$ ,  $b = 200$ ,  $c = 300$ , what will be the value of  $d$  in the following expression :

a.  $d = ++a + ++b + ++c$

b.  $d = a++ + b++ + c++$

42. If  $i = 3, j = 4, k = 2, m$  what will be the value of  $m$  in the following expression

a.  $m = i++ - j--$

b.  $m = ++k \% --j$

c.  $m = j++ / i--$

43. Evaluate each of the following expression independent of each other. The declaration and initialization statement is int  $i = 3, j = 4, k = 2;$

a.  $i++ - j--$

b.  $++k \% --j$

c.  $j+1/i-1$

d.  $j++/i--$

## Part B: Write a Pseudo Code and C program for the following

1. To read the percentage marks of students and determine whether the student has passed or not.
2. To read the number and determine whether the number is even or odd numbers
3. To calculate the volume and surface area of the square ( Volume =  $4/3 * \pi * (r)^3$ , Surface Area =  $4 * \pi * (r)^2$  )
4. To print the number and their squares from 4 to 9.
5. To find the area of a triangle.

6. To find the area and perimeter of a rectangle.
7. To find the area and circumference of a circle.
8. To find the average of four numbers.
9. To swap two numbers using third variable
10. To swap two numbers without using the third variable.
11. To find the product of two numbers
12. To find the greatest of two numbers
13. To find the simple interest
14. To check if the given number is even or odd.
15. Write down a program which initialize 3 integers, G1,G2,G3, then calculate average as real (float). Print your name as string using a simple printf statement, then grades G1,G2,G3 in different line with nice heading, and finally, print the average

#### **Part C : Identify the Syntax errors of the following**

1. Identify the valid and invalid define statements in the following :
  - a. #define X=2.5
  - b. #define MAX 10
  - c. #define N 25;
  - d. #define N 5,M 10
  - e. #define ARRAY 11
  - f. #define PRICE \$100
2. Using appropriate variable names , give the declarations for the quantities given below :
  - a. Radius, diameter, area and circumference of a circle
  - b. Marks in five subjects and their sum and average

3. Declare suitable variables and initialize them with specified values :
  - a. Three integer numbers with value 0.
  - b.  $a = 0$  ,  $\beta = 125.50$  ,  $ch = 'A'$
  - c. Age = 20 years , height =1.65 meters weight=60kg

4. Identify the errors in the declaration given below :

- a. intabc;
- b. float a,b;
- c. char ch1,ch2
- d. int single, double,triple;
- e. char char1=char2='n'
- f. int -a,-b,-c;
- g. #define MAX\_STOP=100
- h. #define newline \n

5. Identify the errors and rectify the following statements :

```
int X;  
floatletter,DIGIT;  
double = p,q;  
exponent alpha, beta;  
m,n,z : INTEGER  
short char c;  
longintm;count;  
long float temp;
```

6. Show the output displayed by the following program lines when the data entered are 5 and 7 :

```
printf(" Enter two integers ");  
scanf("%d%d",&m,&n);  
m=m+5;  
n=3*n;  
printf("m=%d\n n = %d\n",m,n);
```

7. Show the output displayed by the following line if the value of **exp** is 15:

```
printf ("My name is ");
printf(" Dr.ThyaguGowda ");
printf("\n");
printf("I Live in ");
printf("UJIRE-Dharmasthal \n");
printf (and I have %d years",exp);
printf("of Programming Experience \n");
```

8. Change the following Comments so they are syntactically correct .

```
/*This is a comment *\n/* This one /* Seems like a comment */ doesn't it */
```

9. Correct the syntax errors in the following program and rewrite the program so that it follows our style convention. What does each statement of your corrected program do ? What output does it display?

```
/*
 *Calculate and display the difference of two input values
 *)
#include<stdio.h>
void main(void)
{
    int X, /* First input value*/x ,
           /* Second input value*/
    Sum; /*Sum of inputs */

    scanf("%i %i",X;x);
    X+x = sum;
    printf("%d+%d = %d\n",X;x;sum);

    getch();
}
```

- 10.Assume that the variables are declared as follows :

```
int    a,b,c;  
float   x,y;  
double  z;  
char    ch1,ch2;
```

I) Identify the error in the following printf statement

- a. print Hello world
- b. printf('C programming is fun');
- c. printf(a,b,c);
- d. printf("%d\n%d\n%d,a,b,cb");
- e. printf("%f,%f",x,y,z);
- f. print f("%5d%5d%.2f%5c",a,x,ch);
- g. printf("%f %f",&a,&b);
- h. printf{"X:.3.5f;y;.2.3f;",y,x};
- i. printf("%ox%oy%ox",x,y,z);
- j. c=printf("ch1:%c\n",ch1);
- k. printf("%s;%3d 3%f 3%oc",Values of all  
variables:",a,b,c,x,y,z,ch1,ch2,ch3);
- l. printf("%d %d",sqrt(x),pow(x,5));
- m. printf[%d %d,a,b];

II) Identify the errors in the following scanf statement :

- a. scanf&a
- b. scanf(&a,&b,&c);
- c. scanf("%d",&a,&b,&c);
- d. scanf("%f+i%f",&x,&y);
- e. c=scanf("%c %c",&ch,&ch1);
- f. SCANF("%D %D',A,B);
- g. scan f("%5d %5d",&c,&d);
- h. scanf("Hello World");
- i. scanf("% % %",&x,&y,&z);

III) Identify the errors in the following statements

- a. X = sqr(a\*a +b\*b);
- b. C = power(a,b);
- c. Z= $\log_{10} x + \log_e y$
- d. D=tan(180/ $\pi$ a);
- e. getch(&ch1);
- f. ch2=getche("%c");
- g. puts(Hello,world);

11. Write equivalent mathematical equation for the following expressions

- a.  $X = \sqrt{a*a + b*b + c*c}$
- b.  $X = -b + \sqrt{b*b - 4*a*c}/2*a;$
- c.  $C = \exp(\text{abs}(x+y-10))$
- d.  $B = \sqrt{s*(s-a)*(s-b)*(s-c)}$
- e.  $Y = \text{pow}(a,5) + \text{pow}(b,5)/\text{pow}(a+b,5);$
- f.  $Z = \log_{10}(\text{floor}(x+y)) + \exp(\text{ceil}(x-y))/\exp(x+y)$
- g.  $Z = \text{pow}(\sin(x+y),4) - \text{pow}(\cos(x-y),4))/\log(x*x + y*y)$
- h.  $Z = (\exp(a*x) + \exp(b*y))/(a+b)$
- i.  $X = (\exp(\sqrt{x}) + \exp(\sqrt{y}))/x*\sin(\sqrt{y}))$

12. Evaluate the expression given below for two sets of variable values

**i.**  $a=2, b=3, c=4$  and

**ii.**  $a=3, b=1, c=1$

- a.  $\text{pow}(a,3) + \text{pow}(b,3)/(a+b)$
- b.  $\text{floor}(\sqrt{2*a + b*c})/3 + \text{ceil}(10.0/(a+1)/2)$
- c.  $\text{pow}(\text{abs}(a-c) + \text{fabs}(b-c), 3)$

## Module 2: Branching and Looping

**Syllabus :** Selection or Branching Statements ,Two way selection statements ,if , if else, nested if else, cascaded if else (else if ladder),switch statement ,ternary operator (? : ), goto, break and continue ,loops :for,do while and while , goto , break and continue ,Programming Examples and Exercises.

**2.1 Introduction:** C program is a set of statements. The statements in C are executed sequentially one after the other from the beginning of the program to the end of a program. This kind of program execution where in the control flows sequentially is termed as *sequential execution of program*.

In some context it becomes inevitable to skip certain set of statements in order to execute another set of statements. This type of program execution is termed as *selective execution*. In C language the selective execution is achieved using the statements known as *Branching statements or Conditional control structures or statements*.

Also in certain situations a set of statements has to be executed *repeatedly* for given number of times. This is achieved with the help of *Loop Control constructs or statements*. The mechanism of repeating one or more statements execution either for a fixed number of times or until certain condition is satisfied is termed as *Looping*.

**Statement:** Any C expression which ends with semicolon is called **C Statement**. An expression such as x =0 or i++ or printf(-----) becomes a statement when it is followed by a semicolon as in

```
x = 0 ;  
i++;  
printf(-----);
```

In C the **semicolon** is a statement terminator.

**Block:** The set of more than one statements grouped within braces { } so that they are syntactically equivalent to a single statement is known as *Block or Compound statement*. The statements within the braces of *functions , if, else ,while and for* are some of examples.

**2.2. Selection or Branching Statements:** The C statements that transfer the control from one place to other place in the program with or without any condition are called *branching or selection* statements. The selection / branching statements can be classified into two categories:

1. Conditional Control /Branch Statements :
2. Unconditional Control /Branch Statements

**2.2.1 Conditional Branching Statements (Conditional Selection Statements):** These are the statements which will transfer control flow from one place to another place, so as **to execute** a set of instructions *,if some condition is met* or, **to skip** the execution of the statements *if the condition is not met*. They are also known as Conditional selection statements or decision statements. They specify the order in which computation are performed.

**2.2.1.1 Two way Selection Statement:** The C programming statement which provides *two paths to control flow* to follow one for the true condition and the other for the false condition as shown in figure 2.1 is called two way selection statement.

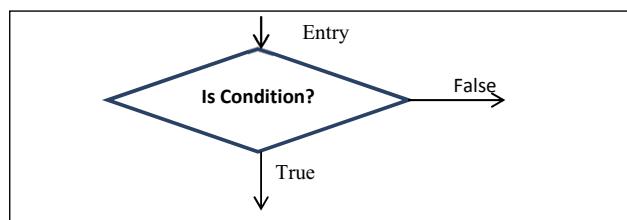


Fig 2.1: Two way Branching

### 2.2.1.2 Types of Condition Control Statements / Two way

**selection Statement:** The different two way selection statements supported by C language are:

1. If statement
2. If else statement
3. Nested if else
4. Cascaded if else (else if ladder)

1. **If statement:** It is basically a two way decision statement and it is used in conjunction with an expression. *It is used to execute a set of statements if the condition is true. If the condition is false it skips executing those set of statements.* The syntax and flow chart of if statement is as illustrated below:

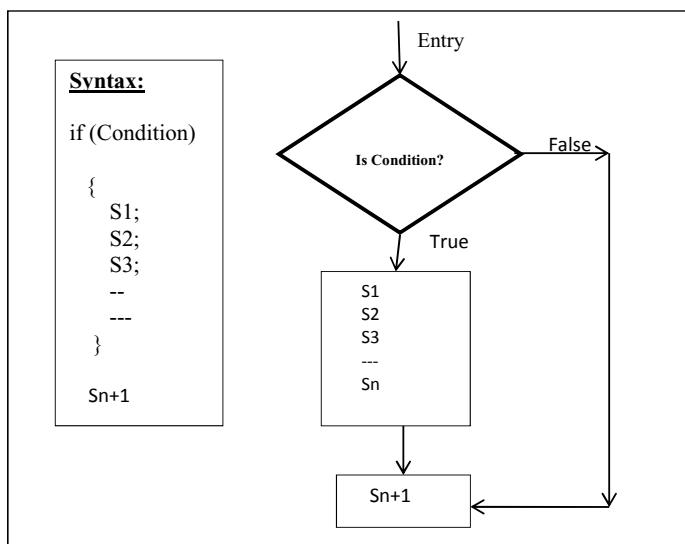


Fig 2.2: Syntax and flow diagram for if statement

**Example 1:C program to find the largest number using if statement.**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int large,x,y;
    clrscr();
    printf("\n Enter the value of x and y\n");
    scanf("%d%d",&x,&y);
    large = x;
    if(y>large)
        large = y;
    printf("Large =%d\n",large);
    getch();
}
```

**Output :**

Enter the value of x and y

**15 -5**

Large = 15

**Example 2: C Program to determine whether a person is eligible to vote using if.**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int age;
    clrscr();
    printf("\n Enter the age :");
    scanf("%d",&age);
    if(age>=18)
        printf("\n You are eligible to vote ");
    getch();
}
```

**Output :**

Enter the age: 20

You are eligible to vote

**2. If else statement:** It is an extension of **if** statement. It is used to execute any one set of two set of statements at a time. If condition is true it executes one set of statements otherwise it executes another set of statements. The syntax and flow diagram of if else is as shown in the figure below. As illustrated in the figure if the condition is true the set of statements  $\{S_{11}, S_{12}, \dots, S_{1n}\}$  gets executed else if the condition is false the set of statements  $\{S_{21}, S_{22}, S_{23}, \dots, S_{2n}\}$  gets executed.

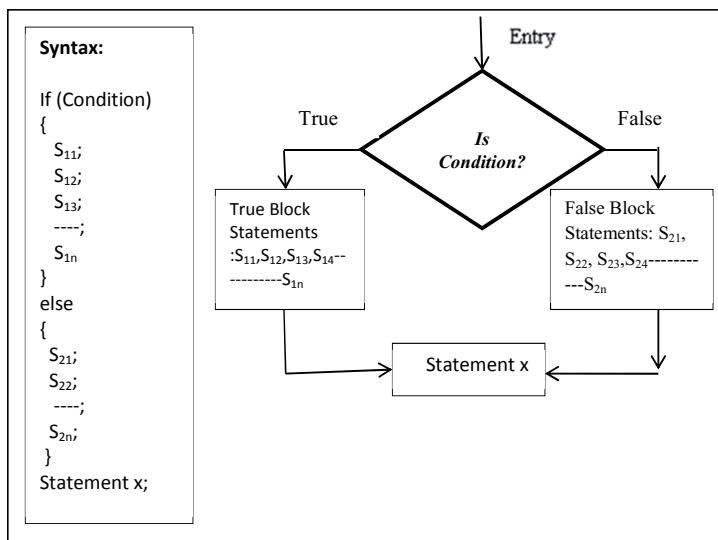


Fig 2.3: Syntax and flow diagram for if else statement

**Example: Program to check whether a given number is even or odd using if else.**

```

#include<stdio.h>
#include<conio.h>
main()
{
    int num;
    clrscr();
    printf("\n Enter the number\n");
    scanf("%d",&num);

```

```

if(num%2==0)
printf("The number is Even\n");
else
printf("\nThe number is odd\n");
getch();
}

```

### Output :

**Run 1:** Enter the number

20

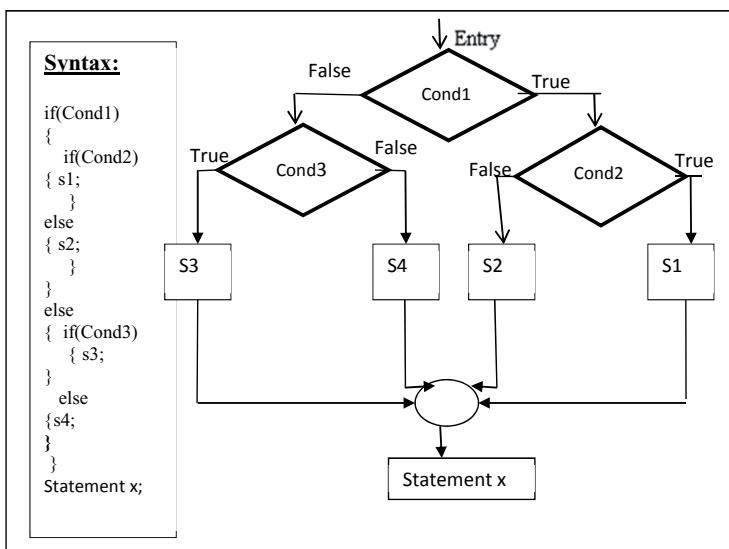
The number is Even

**Run 2:** Enter the number

5

The number is odd

3. **Nested if else:** When a series of decisions are involved we may have to use more than one if else statement in nested form. The nested if else statements are multidecision statements which consist of if else control statement within another if or else section. The syntax and flow diagram for nested if else is as shown below:



**Fig 2.4: Syntax and flow diagram for nested if else statement**

**Example: C program to find the largest of three numbers using *if else* statement**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int a,b,c ;
    clrscr();
    printf("\n Enter the values of a,b and c\n");
    scanf("%d %d %d",&a,&b,&c);

    if(a>b)
    {
        if (a>c)
            printf("\n %d is largest \n", a);
        else
            printf("\n%d is largest \n",c);
    }
    else
    {
        if(b>c)
            printf("\n %d is largest \n",b);
        else
            printf("\n%d is largest \n",c)
    }
    getch();
}
```

**Output :**

**Run 1:**

```
Enter the values of a,b and c
20 30 15
30 is largest
```

**Run 2:**

```
Enter the values of a,b and c
-2 -50 -5
-2 is largest
```

4. **Cascaded if else (else if ladder):** Cascaded *if else* is a multipath decision statements which consist of chain of *if else* where in the nesting take place only in else block. It is a special case of *nested if else* and is also known as *else if ladder*. The syntax and flow diagram for cascaded *if else* is as shown in figure 2.5 .

**Example:** C Program to display the grade obtained by a student based on the marks. The relation between the grade and marks is shown below:

| Marks     | Grades          |
|-----------|-----------------|
| 0 to 39   | F (Fail)        |
| 40 to 49  | E               |
| 50 to 59  | D               |
| 60 to 69  | C               |
| 70 to 79  | B               |
| 80 to 89  | A               |
| 90 to 100 | O (Outstanding) |

```
#include<stdio.h>
#include<conio.h>
main()
{
    int marks;
    clrscr();
    printf("Enter the marks \n");
    scanf("%d",&marks);
    if(marks>=0 && marks<=39)
        printf("Grade F\n");
    else if (marks>=40 && marks<=49)
        printf("Grade E\n");
    else if (marks>=50 && marks<=59)
        printf("Grade D\n");
    else if (marks>=60 && marks<=69)
        printf("\n Grade C\n");
    else if(marks>=70 && marks<=79)
        printf("\nGrade B\n");
    else if (marks>=80 && marks<=89)
```

```

printf("\nGrade A\n");
else if (marks>=90 && marks<=100)
printf("\nOutstanding\n");
else
printf("\n Invalid Entry \n");
getch();
}

```

### Output :

Enter the marks  
55  
Grade D

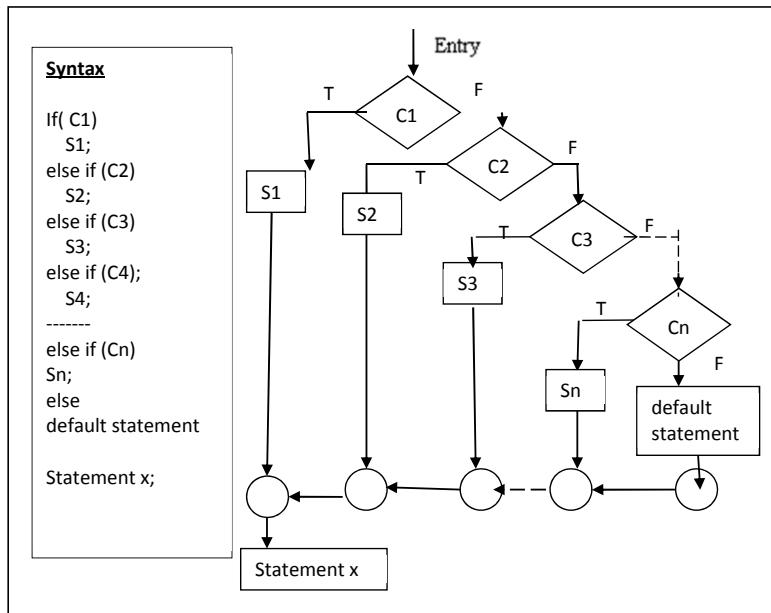


Fig 2.5: Syntax and flow diagram for cascaded if else statement

**2.2.1.3 Switch Statement:** The switch statement is a multiway decision that tests whether an expression matches one of a number of

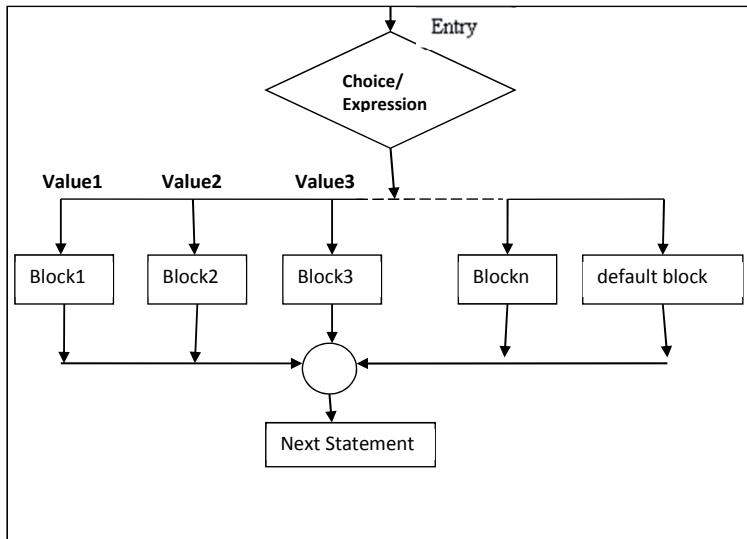
constant integer values and branches accordingly. It is a multiway decision making control statement used to make a selection between many alternatives. It is also known as *switch case break and default statement*. The syntax and flow diagram of switch statement are illustrated below:

### Syntax :

```
switch (choice/expression)
{
    case value1:block1;
    break;
    case value2: block2;
    break;
    case value3: block3;
    break ;
    -----
    case valuen: blockn;
    break;
    default : default_block ;
}

next Statement;
```

The value of choice or expression is always an integer value. If the value of choice or expression is 1 then block1 statements will get executed , if the value is 2 then the block2 statements will get executed and similarly for all the values upto **n**. After executing the respective block the control comes outside the switch statement because of the break statement. If no case values matches with the value of the choice , then the default block will get executed before exiting switch statement.



**Fig2.6: Flow diagram for Switch statement**

**Example : C program to simulate simple calculator .**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int a,b,res;
    int choice ;
    clrscr();
    printf("\nEnter two numbers \n");
    scanf("%d %d",&a,&b);

    printf("\nEnter the choice\n1.Addition
\n2.Substraction\n3.Multiplication\n4.Division\n");
    scanf("%d",&choice);

    switch(choice)
    {
        case 1: res = a+b;
        printf("\nThe result = %d\n",res);
        break ;
        case 2: res = a -b;
        printf("\nThe result = %d\n",res);
        break ;
        case 3: res = a *b;
        printf("\nThe result = %d\n",res);
        break ;
        case 4: res = a /b;
        printf("\nThe result = %d\n",res);
        break ;
        default:
            printf("Wrong choice");
    }
}
```

```

        printf("\n The result = %d\n",res);
        break;
    case 3: res = a*b;
        printf("\n The result = %d\n",res);
        break;
    case 4: if(b==0)
        printf ("\n Divide By Zero Error\n");
        else
        {res= a/b;
            printf("\n The result = %d\n",res);
        }
        break;
    default :printf("\n Invalid Input \n");

    }
    getch();
}

```

### Output :

```

Enter two numbers
2 3
Enter the choice
1.Addition
2.Substraction
3.Multiplication
4.Division
3
The result = 6

```

**2.2.1.4 Ternary Operator:** It is an operator with *three* operands. The ternary operators are **? and ::**. It works like a *if else statement*. It is also known as conditional operator. The syntax for the ternary operators is given below:

**exp1? exp2:exp3 ;**

Here exp1 is evaluated first. If it is true then exp2 is evaluated and becomes the result of the expression, otherwise exp3 is evaluated and becomes the result of the expression.

**Example:**   large = (a>b) ? a : b; **i.e.,** if a>b is true **large = a** else **large = b**

**Nested Ternary Operator:** Nested ternary operator consists of the ternary operator inside another ternary operator . Here the conditional operator is used as an operand of another conditional operator. That is one can have *nested conditional expressions*. Following example illustrates the concept of nested ternary operator which is used to find the smallest of three numbers.

```
int a=5,b=3, c=7,small;  
small = ((a<b)?(a<c?a:c):(b<c?b:c));
```

**Example : Program to find the largest of two numbers using ternary operator**

```
#include<stdio.h>  
#include<conio.h>  
main()  
{  
    int num1,num2,large;  
    clrscr();  
    printf("nEnter the two numbers \n");  
    scanf("%d %d",&num1,&num2);  
    large= (num1>num2)?num1:num2;  
    printf("nThe largest number is :%d",large);  
    getch();  
}
```

#### Output :

```
Enter the two numbers  
10 20  
The largest number is : 20
```

**Example : Program to find the largest of three numbers using nested ternary operator.**

```
#include<stdio.h>  
#include<conio.h>  
main()  
{  
    int num1,num2,num3,large;  
    clrscr();  
    printf("nEnter the three numbers \n");  
    scanf("%d %d %d ",&num1,&num2,&num3);
```

```
large= (num1>num2) ? ((num1>num3) ? num1: num3):  
((num2>num3)?num2:num3);  
printf("\n The largest number is :%d",large);  
getch();  
}
```

### Output :

```
Enter the three numbers  
110 20 15  
The largest number is : 110
```

**2.2.2 Unconditional Branching Statements:** The unconditional branching statements transfer the control from one statement to another statement in the program without any conditions. The different types of unconditional statements supported by C language are as follows :

1. goto
2. break
3. continue
4. return

1. **goto statement :** The goto statement is used to transfer control from one point in the program to another point without any conditions i.e. unconditionally. The goto statement can be used to transfer or jump the control in the forward or backward direction. The general format/syntax of the goto used for forward and backward is as illustrated below :

| Syntax for goto used in<br><i>forward flow or jump</i>                                                                                                                     | Syntax for goto used in<br><i>backward jump</i>                                                                                                                            |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>         Stmt1;         Stmt2; <b>goto Label;</b> ←         Stmt3;         Stmt4;         Stmt5; <b>Label :</b> ←         Stmt6;         ----- <b>(Case 1)</b> </pre> | <pre>         Stmt1;         Stmt2; <b>Label:</b> ←         Stmt3;         Stmt4;         Stmt5; <b>goto Label ;</b> ←         Stmt6;         ----- <b>(Case 2)</b> </pre> |

Where **Label** is any identifier and it is need not be declared. The rules that are applicable for formation of variable name is also applicable for **Label**. The Label should be used along with a statement to which the control is transferred as given below:

### **Label: Statement**

The Label can be anywhere in the program either before or after the **goto Label;** statement. If the label is before the statement **goto Label ;** (**Case 2)** a loop will be formed and some statements will be executed repeatedly. Such jump is known as a ***backward jump*** on the other hand if the **Label;** is placed after the **goto Label ;** some statements will be skipped and such jump is known as a ***forward jump***.

#### **Example 1 :**

```

#include<stdio.h>
#include<conio.h>
main()
{
    float x,y;
    clrscr();
}

```

```

Read:
printf("\n Enter the value of x\n");
scanf("%f",&x,)
if(x<0)
{
    printf ("\n Negative number is entered\n");
    printf("\n Enter only the positive Number\n");
    goto Read ;
}
y =sqrt(x);
printf("\n The Square root of x =%f\n",y);
getch();
}

```

### Output :

```

Enter the value of x
-20
Negative number is entered
Enter only the positive Number
Enter the value of x
16
The Square root of x = 4.000000

```

### Example 2:

```

#include<stdio.h>
#include<conio.h>
main()
{
    int sum=0,i =0;
    clrscr();

top:
if(i>10) goto end ;
sum= sum+i;
i++;
goto top;
end: printf("\n Sum = %d\n",sum);
getch();
}

```

### Output :

## Sum = 55

### Example 3:

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i=1;
    clrscr();

back:
printf("\n XYZ\n");
if(i!=5)
{
    i++;
    goto back ;
}
getch();
}
```

### Output :

```
XYZ
XYZ
XYZ
XYZ
XYZ
```

### Note :Disadvantages of goto :

1. Using goto the code is difficult to read and understand .
2. The usage of goto results in unstructured programming
3. It is not good programming style.

## 2.3 Looping Statements :

The statement which are used to repeat a set of statements repeatedly for a given number of times or until a given condition is satisfied is called as *looping constructs or looping statements*. The set or block of statements used for looping is called loop.

**2.3.1 Types of Looping statements:** Depending on the position of the control statement in the loop the looping statements are classified into the following two types:

1. Entry controlled Loop
2. Exit Controlled Loop

**1. Entry Controlled Loop:** In the entry controlled loop the control conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed. It is also known as *pretest loop or top test loop*. The flow chart for the entry controlled loop is as illustrated in fig 2.7.

**2. Exit Controlled Loop:** In the exit controlled loop the test is performed at the end of the body of the loop and therefore the body is executed unconditionally for the first time. It is also known as *post test loop*. Here the body of the loop will get executed at least once before testing. The flow chart for the exit controlled loop is as illustrated in fig 2.8.

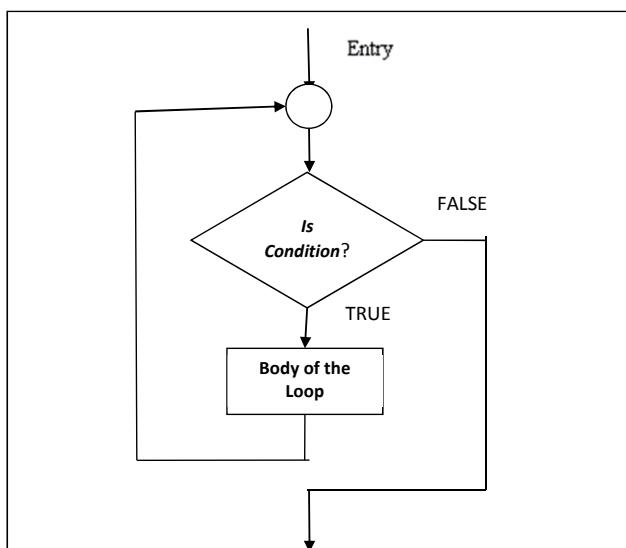
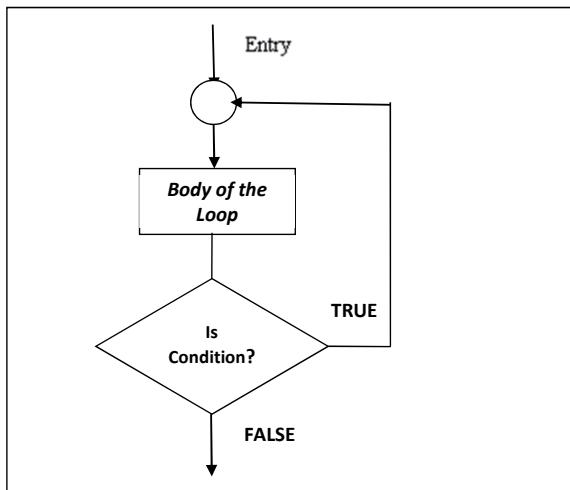


Fig 2.7: Flow diagram for Entry Controlled Loop



**Fig 2.8: Flow diagram for Exit Controlled Loop**

**The looping includes the following four steps:**

1. Initialization of a condition variable
2. Testing for a specified value of the condition variable for execution of the loop.
3. Execution of the statements in the loop
4. Updating (Incrementing or Decrementing) the condition variable

**2.3.2 Types of Loops Supported in C:** The C Language supports the following three looping operations:

1. The while statement
2. The do while statement
3. The for statement

**The while statement:** It is an entry controlled loop statement. In case of while loop the initialization, testing the condition and incrementation or updation is done in separate statements. First initialization of the loop

counter is performed. Next the condition is checked. If the condition evaluates to be true then the control enters the body of the loop and executes the statements in the body.

1. The syntax or basic format of the while statement is as shown in figure 2.9 below :

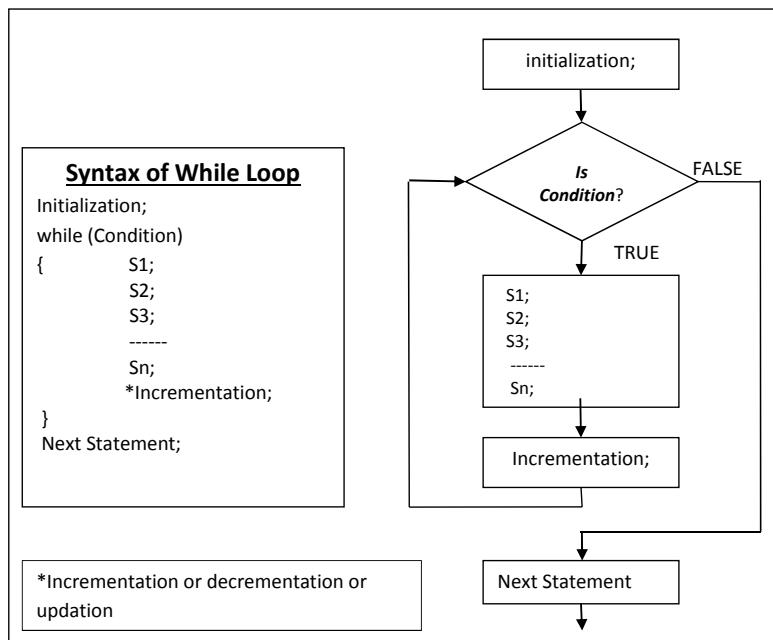


Fig 2.9: Flow diagram for Exit Controlled Loop

**Example: Program to find the sum of n natural numbers using while loop.**

```
#include<stdio.h>  
#include<conio.h>  
main()  
{  
    int i,n,sum=0;  
    clrscr();
```

```

printf("\n Enter the value of n\n");
scanf("%d",&n);
i=1;
while(i<=n)
{
    sum = sum + i;
    i++;
}

printf("\n The sum of natural numbers = %d \n",sum);
getch();
}

```

**Output :**

**Enter the value of n**  
**10**  
**The sum of natural numbers = 55**

- 2. Do While Loop :** It is an exit controlled loop. It is also known as *posttest or bottom test looping statement*. In do while first initialization takes place. After initialization the body of the loop is entered and the statements are executed. Next incrementation or updation is performed and at the end of the body the condition is checked. If condition evaluates to be true then the control reenters the body and executes the statement within the body.

**Example:**

```

i=1;
do
{
    printf("\n Welcome \n");
    i++;
} while (i<=5);
printf("\n Good Bye\n");

```

The syntax or general format of do while loop is as illustrated in Fig 2.10.

**Example : Program to find the sum of n natural numbers using do while loop.**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,n,sum=0;
    clrscr();
    printf("\n Enter the value of n\n");
    scanf("%d",&n);
    i=1;
    do
    {
        sum =sum+i;
        i++;
    }while(i<=n);

    printf("\n The sum of natural numbers = %d \n",sum);
    getch();
}
```

**Output :**

```
Enter the value of n
10
The sum of natural numbers = 55
```

**Example : Program to find the sum of odd numbers upto n natural numbers using do while loop.**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,n,sum=0;
    clrscr();
    printf("\n Enter the value of n\n");
    scanf("%d",&n);
    i=1;
    do
    {
        if(i%2!=0)
        {
```

```

        sum =sum+i;
    }
    i++;
}while(i<=n);

printf("\n The sum of odd numbers = %d \n",sum);
getch();
}

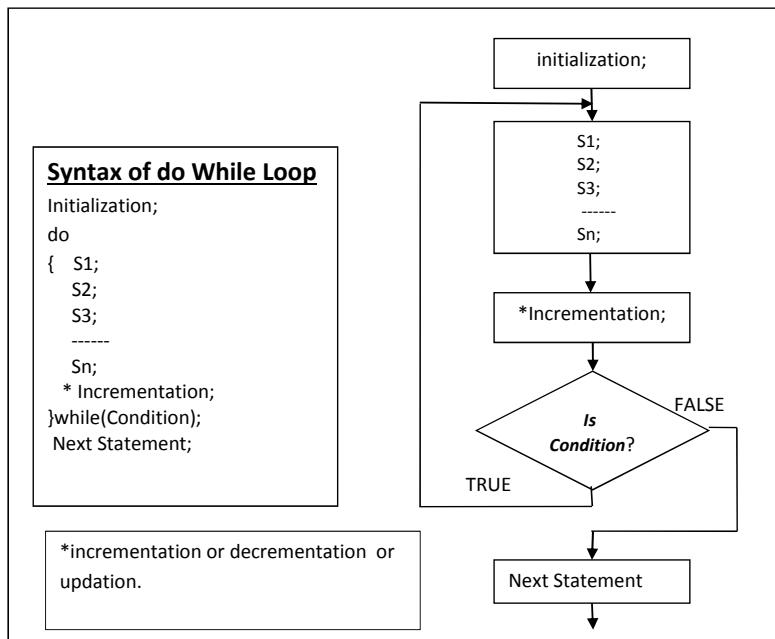
```

### Output :

**Enter the value of n**

**10**

**The sum of odd numbers = 25**

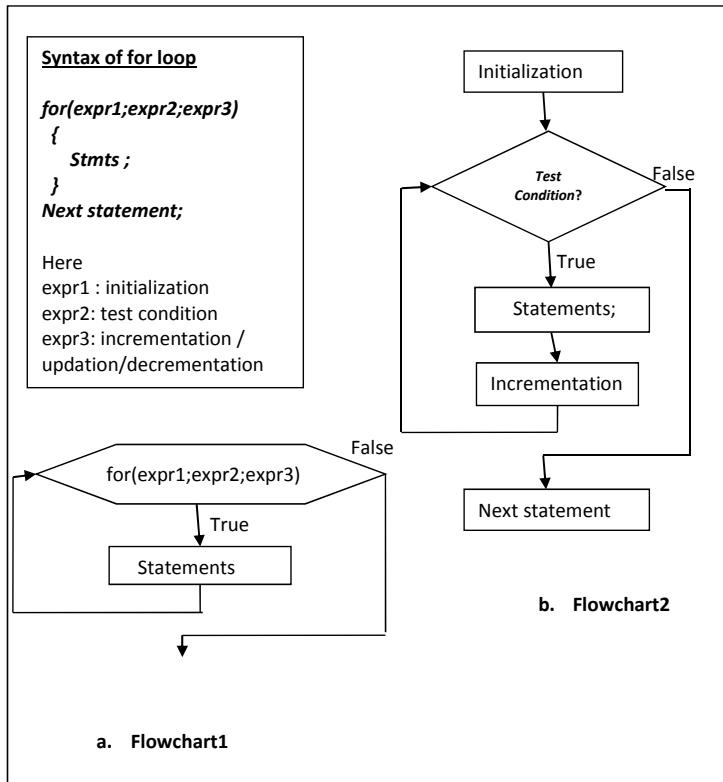


**Fig 2.10: Syntax and Flow diagram for do while Loop**

## Difference between While Loop and Do While Loop

| <b>While Loop</b>                                                                                                                                                           | <b>Do While Loop</b>                                                                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>1.</b> Entry Controlled Loop                                                                                                                                             | <b>1.</b> Exit Controlled Loop                                                                                                                                                      |
| <u><b>2. Syntax :</b></u><br><b>Initialization;</b><br><b>while (Condition)</b><br>{   S1;<br>S2;<br>S3;<br>-----<br>Sn;<br><b>updation;</b><br>}<br><b>Next Statement;</b> | <u><b>2. Syntax:</b></u><br><b>Initialization;</b><br><b>do</b><br>{   S1;<br>S2;<br>S3;<br>-----<br>Sn;<br><b>updation;</b><br><b>}while(Condition);</b><br><b>Next Statement;</b> |
| <b>3.</b> It is also known as pretest or top testing loop                                                                                                                   | <b>3.</b> It is also known as posttest loop or bottom testing loop.                                                                                                                 |
| <b>4.</b> If the condition evaluated to be false then the statements within the body of the loop will not be executed even once (Zero execution of the loop)                | <b>4.</b> The statement inside the loop gets executed at least once since the condition is checked at the end of the loop                                                           |
| <b>5. Example :</b><br><br>i = 0;<br>sum = 0;<br>while(i<=n)<br>{<br>sum = sum+i;<br>i = i+1;<br>}                                                                          | <b>5. Example :</b><br><br>i = 0;<br>sum = 0;<br>do<br>{<br>sum = sum+i;<br>i = i+1;<br>} while(i<=n);                                                                              |

**3. For loop:** The for loop is another entry controlled loop. It is used to execute the set of statements repeatedly for a given condition. In case of for loop *the initialization, condition check and incrementation* can be done in the single statement. The syntax or general format of for loop is as illustrated below:



Just like while loop here also first the loop counter is initialized if the condition evaluates to be true then the control enters the body of the loop. The statement in the body is executed. The ***working of the for statement*** is as described in the following steps:

1. Initialization of the control variables is done first using assignment statements such as  $i = 1$  and  $count=0$ . The variables  $i$  and  $count$  are known as loop control variable.

2. The value of the control variable is tested using the test condition .  
The test condition is a relational expression such as  $i < 10$  that determines when the loop will exit. If the condition is false the body of the loop is terminated and the execution continues with the statement that immediately follows the loop.
3. When the body of the loop is executed the control is transferred back to the for statement after evaluating the last statement in the loop. Now the control variable is incremented using assignment statement such as  $i = i + 1$  and the new value of the control variable is again tested to see whether it satisfies the loop condition. If the condition is satisfied the body of the loop is executed. The process continues till the value of the control variables fails to satisfy the test condition.

**Example: C Program to find the sum of natural numbers upto n.**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,sum=0,n;
    clrscr();
    printf("\nEnter the value of n \n");
    scanf("%d",&n);

    for(i=1;i<=n;i++)
    {
        sum = sum+i;
    }
    printf("\n The sum =%d\n",sum);
    getch();
}
```

**Output :**

Enter the value of n

10

The sum = 55

## **Additional Features of for Loop:**

1. More than one variable can be initialized at a time in the for statement.

**Example:** for(i=1,j=2,k = 0;k<17;++k)

2. The increment section may also have more than one part

**Example:**

```
for(i=1,j=50; i<=j; i++,j--)  
{   k = i/j;  
    printf("%d %d %d\n",i,j,k);  
}
```

3. The test condition may have any compound relation and testing need not be only to the loop control variable.

**Example:**

```
sum= 0;  
for(i=1;i<=30&&sum<100;++i)  
{   sum=sum+i;  
    printf("%d %d\n",i,sum);  
}
```

4. One or more section can be omitted if necessary

**Example:**

```
i=15;  
for (; i!=100 ;)  
{  
    printf("%d\n",i);  
    i= i +5;  
}
```

**Example: C program to find the sum of squares of first n natural numbers.**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int n,i,sum;
    clrscr();

    printf("nEnter the value of n\n");
    scanf("%d",&n);
    sum=0;

    for(i=1;i<=n;i++)
    {
        sum = sum+i*i;
    }
    printf("Sum of squares = %d",sum);
    getch();
}
```

**Output :**

**Run 1:**

```
Enter the value of n
10
Sum of squares = 385
```

**Run 2:**

```
Enter the value of n
5
Sum of squares = 55
```

### Comparison of while Loop and For Loop:

| <b>Sl.NO</b> | <b>While Loop</b>                                                                                                                                                             | <b>For Loop</b>                                                                                                |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <b>1</b>     | It is a pretest loop where the condition is checked before executing the body of the loop                                                                                     | It is also pretest loop where the condition is checked before executing the body of the loop.                  |
| <b>2</b>     | <u><b>Syntax :</b></u><br>Initialize ;<br>while (expression)<br>{<br>S1;<br>S2;<br>S3<br>---<br>Sn<br>Update;<br>}                                                            | <u><b>Syntax :</b></u><br>for(initialize;testconditio<br>n;updation)<br>{ S1;<br>S2;<br>S3;<br>---<br>Sn;<br>} |
| <b>3</b>     | Initialization is done before the while loop, testing is done in the beginning of the while loop and updation is done in the body of the loop normally as the last statement. | Initialization, testing and updating are done in one place in the for statement.                               |
| <b>4</b>     | <u><b>Example :</b></u><br>sum=0;<br>i=1;<br>while(i<=n)<br>{ sum=sum+i;<br>}                                                                                                 | <u><b>Example :</b></u><br>sum=0;<br>for(i=1;i<=n;i++)<br>{ sum=sum+i;<br>}                                    |

**2.3.3 Jumps in Loops:** Sometimes it becomes essential during execution of loop to skip certain statements or iteration and jump outside the loop. C permits a jump from one statement to another within a loop as well as jump out of a loop. This type of jump can be achieved using the statements **1. break and 2. continue**.

**1. Break statement:** The break statement is a jump statement which can be used in switch statement and loops. The break statement works as described below:

**Break in switch:** It causes the control to terminate switch statement and the statement following switch statement will be executed. Consider the switch statement as given below. If the choice is 2 then the program control after executing the statements of case2, will jump outside the switch as it encounters break statement.

```
switch(ch)
{   case 1: stmt1;
    break;
    case 2:stmt2;
    break;
    -----
    -----
    case n: stmtn;
    break;
}
```

next stmt;

**Break in loop:** Whenever the break is encountered in a loop the control comes out of the loop and the statement following the loop will be executed. The break statement is used mainly to terminate the loop when a specific condition is reached. If the break statement is included in nested loop where in the break is present in the innermost loop, *then the break present in innermost loop can make the control come outside the innermost loop only (or from any loop to which it belongs)*. Some of the examples of usage of break in loop is illustrated below:

|                                                                                            |                                                                                               |                                                                                                                                    |
|--------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <pre> while() {     if(TRUE)         break; } </pre>                                       | <pre> for (-----) {     if(TRUE)         break ; } </pre>                                     | <pre> do {     -----     if(TRUE)         break; } while(TRUE); </pre>                                                             |
| <pre> for(-----) {     for(-----)     {         if(TRUE)             break;     } } </pre> | <pre> for(-----) {     for(-----)     {         if(TRUE)             goto END;     } } </pre> | <pre> while (-----) {     if (TRUE)         goto STOP;     -----     if (TRUE)         goto TGS;     -----     TGS: } STOP: </pre> |

**Example:** Consider the programming snippet containing for loop as illustrated below:

```

for(i=1;i<=10;i++)
{
    if(i==5) break ;
    printf("%d",i);
}

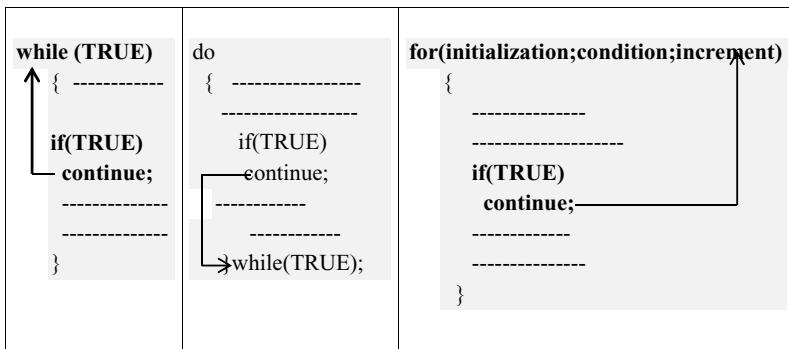
```

The output of the above program snippet will be : 1 2 3 4

**2.The Continue Statement:** The continue statement causes the loop to be continued with the next iteration after skipping any statements in between. The continue statement tells the compiler, “ *SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION* ”. The format of continue statement is

**continue;**

The usage of continue in different looping statements is illustrated below:



### Example 1:

```

for(i=1;i<=5;i++)
{
    if(i==2) continue;
    printf(" %d ",i);
}

```

**Output :** 1 3 4 5

### Example 2:

```

for(i=1;i<=10;i++)
{
    if(i%2==0) continue;
    printf(" %d", i);
}

```

**Output :** 1 3 5 7 9

## Difference between break and continue statement:

| Sl.NO | Break Statement                                                                                                                | Continue Statement                                                                                                                             |
|-------|--------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| 1     | When break is executed the statements following break are skipped and causes the loop to be terminated                         | When continue statement is executed the statements following continue are skipped and causes the loop to be continued with the next iteration. |
| 2     | It can be used in switch statement to transfer the control outside switch                                                      | It cannot be used inside a switch statement                                                                                                    |
| 3     | <pre>for(i=1;i&lt;=5;i++)<br/>{<br/>    if(i==3)<br/>        break ;<br/>    printf("%d\n",i);<br/>}</pre> <p>Output : 1 2</p> | <pre>for(i=1;i&lt;=5;i++)<br/>{<br/>    if(i==3) continue ;<br/>    printf(" %d\n",i);<br/>}</pre> <p>Output : 1 2 4 5</p>                     |

## Additional Concepts

- 1. Dangling else problem :** This problem is created when there is no matching else for if statement . In such case C always pair an else statement to the most recent unpaired if statement in the current block.

**Example :**

```
if(x>10)
if(x>20)
    printf("\n a is greater than 20\n");
else
    printf("\n a is not greater than 10);
```

- 2. Null Statement :** The C statement which includes only ‘;’ is called null statement . Usually it is used to create a delay in the program .

**Example 1:**

```
for(i=0;i<10000;i++);
printf("\n Delay of 10000 Times")
```

Here the loop will executes ; for 10000 times thus creating the delay of 10000 unit of time.

**Example 2:**

```
for(count =1;count<=5;count++);
printf("%d",count);
```

The above loop will print the output as “6”. This is because of the semicolon ; at the end of for loop. When there is a semicolon , it is taken as the end of the for loop. So the loop is executed 5 times without executing any other statement. When it comes out of the loop, the value of count is 6. That is printed by the printf statement. So the printf statement was executed only once.

- 3. Infinite Loop:** An infinite loop is a loop that executes the statement indefinitely. It is a loop whose termination test never evaluates to false.

**Example 1:**

```
int count =1;
while(count<=10)
{
    printf("%d\n", count);
}
```

The above code will print 1 indefinitely . So the loop is an infinite loop. This is because there is no change in the value of the variable count.

### **Example 2:**

```
int count;
while(count<=10)
{
    printf("%d\n",count);
    count++;
}
```

### **4. Terminating the program using exit() Function**

The function exit is used to quit the program . It terminates the program and returns the status code to the operating system. This function is defined in ‘stdlib.h’ header file.

**Syntax :** `exit(int status);`

The status code zero indicates that the program completed successfully. If there is a failure any other code has to be returned.

### **5. Comparison of break , continue and exit**

| <b>break</b>                             | <b>continue</b>                   | <b>exit()</b>                       |
|------------------------------------------|-----------------------------------|-------------------------------------|
| Used to quit an innermost loop or switch | Used to continue the loop         | Used to terminate the program       |
| Can be used only within loops or switch  | Can be used only within the loops | Can be used anywhere in the program |

## 2.4. Programming Examples;

### 2.4.1 Programming Examples based on if, if else, nested if else, cascaded if else (else if ladder) and switch statement.

1. C program to read a year as an input and find it whether the year is leap year or not using if statement. Also consider the end of centuries.

```
/* Program to find whether a year is leap year or not. Also
consider end of centuries */
#include<stdio.h>
#include<conio.h>
main()
{
    int year;
    clrscr();
    printf ("Enter a year : ");
    scanf ("%d", &year);
    if((year%4==0)&&(year%100!=0)||((year%400==0))
    printf("\n The year %d is leap year \n",year);
    getch();
}
```

#### Output:

```
Enter a Year
2016
The year 2016 is leap year
```

2. C program to enter a character and then determine whether it is a vowel or not.

```
#include<stdio.h>
#include<conio.h>
main()
{
    char ch;
    clrscr();
    printf("\n Enter any character \n");
    scanf("%c",&ch);
    if(ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u'||ch=='A'||ch=='E'||ch=='I'||ch=='O'||ch=='U')
```

```

'A'||ch=='E'||ch=='I'||ch=='O'||ch=='U')
    printf("\n %c is a Vowel",ch);
else
    printf("\n %c is not a vowel",ch);
getch();
}

```

**Output:**

Enter any character  
U  
U is a Vowel

3. C program to check leap year or not using else if ladder ( year will be entered by the user).

```

#include <stdio.h>
#include<conio.h>
main()
{
    int year;
    clrscr();
    printf("Enter a year to check if it is a leap year\n");
    scanf("%d", &year);

    if (year%400 == 0)
        printf("%d is a leap year.\n", year);
    else if (year%100 == 0)
        printf("%d is not a leap year.\n", year);
    else if (year%4 == 0)
        printf("%d is a leap year.\n", year);
    else
        printf("%d is not a leap year.\n", year);
    getch();
}

```

**Output:**

Enter a Year  
**2016**  
The year 2016 is a leap year

**4. C program to check whether a number entered is positive, negative or equal to zero.**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int num;
    clrscr();
    printf("\n Enter any number ");
    scanf("%d",&num);
    if(num==0)
        printf("\n The value is equal to zero\n");
    else if(num>0)
        printf("\n The number is positive \n");
    else
        printf("\n The number is negative \n");
    getch();
}
```

**Output:**

```
Enter any number
100
The number is positive
```

**5. C program to display the examination result using else if ladder.**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int marks;
    clrscr();
    printf("\n Enter the marks obtained \n");
    scanf("%d",&marks);
    if(marks>=75&&marks<=100)
        printf("\n DISTINCTION\n");
    else if (marks>=60 && marks<75)
```

```

printf("\n FIRST DIVISION\n");
else if (marks>=50&&marks<=60)
printf("\n SECOND DIVISION\n");
else if(marks>=40 &&marks<50)
printf("\n Third Division\n");
else
printf("\n FAIL\n");
getch();
}

```

**Output:**

Enter the marks obtained

**98**

DISTINCTION

**6. C program to determine whether an entered character is vowel or not using switch statement.**

```

#include<stdio.h>
#include<conio.h>
main()
{
    char ch;
    clrscr();
    printf("\n Enter any character\n");
    scanf("%c",&ch);
    switch(ch)
    {
        case 'A':
        case 'a':
            printf("\n %c is Vowel\n",ch);
            break;
        case 'E':
        case 'e':
            printf("\n %c is Vowel\n",ch);
            break;
        case 'I':
        case 'i':
            printf("\n %c is Vowel\n",ch);
            break;
        case 'O':
        case 'o':

```

```

        printf("\n %c is Vowel\n",ch);
        break;
    case 'U':
    case 'u':
        printf("\n %c is Vowel\n",ch);
        break;
    default : printf("\n %c is not a vowel \n",ch);
}
getch();
}

```

### Output:

Enter any character

**R**

**R** is not a vowel

- 7. C program to enter a number from 1 to 7 and display the corresponding day of the week using switch case statement.**

```

#include<stdio.h>
#include<conio.h>
main()
{
    int day ;
    clrscr();
    printf("\n Enter any number from 1 to 7:\n");
    scanf("%d",&day);

    switch(day)
    {
        case 1: printf("\n SUNDAY\n");
                  break;
        case 2: printf("\n MONDAY\n");
                  break;
        case 3: printf("\n TUESDAY\n");
                  break;
        case 4: printf("\n WEDNESDAY\n");
    }
}

```

```

        break;
    case 5: printf("\n THURSDAY\n");
        break;
    case 6: printf("\n FRIDAY\n");
        break;
    case 7: printf("\n SATURDAY\n");
        break;
    default : printf("\n Invalid Entry\n");
}
getch();
}

```

**Output:**

```

Enter any number from 1 to 7
7
SATURDAY

```

- 8. C program which reads the three sides of a triangle and makes a test to classify the triangle and reports the result.**

```

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

void main( )
{
    float a,b,c ;
    clrscr();

    printf("\n Enter three sides of a triangle \n");
    scanf("%f %f %f", &a,&b,&c);
    if((a>= b+c) ||(b>=c+a)|| (c > = a+b))
        printf (" \n Triangle formation is not possible \n");
    else if( (a==b) ||(b==c)|| (c == a))
        printf(" \n Isoceles Triangle \n");
    else if( (a==b)&&(b==c)&&(c == a))
        printf(" \n Equilateral Triangle\n");
    else
        printf(" \n Scalene Triangle \n");
    getch();
}

```

**Output:**

```

Enter three sides of a triangle
7 7 7
Equilateral Triangle

```

**9. Write a program segment that will read the value of x and evaluate the following function :**

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

**Using nested i) if statement ii) else if statement and iii)conditional operator ?**

i. Using nested if statement

```
if(x>0)
    y=1 ;
else
    if( x ==0)
        y ==0 ;
    else
        y = -1;
```

ii. using else if statement

```
if(x>0) y =1 ;
else if (x ==0) y =0;
else if(x<0) y = -1;
```

iii. using conditional operator

```
y = (x>0) ? 1: (( x==0)? 0:-1);
```

**10.C Program to find the largest of three values using if else statement.**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c;
    clrscr();
    printf("\n Enter three numbers \n");
    scanf("%d %d %d",&a,&b,&c);
    if((a>b)&&(a>c))
        printf("\n Largest of three values is %d",a);
    else if ((b>c)&&(b>a))
        printf("\n Largest of three values is %d",b);
    else if ((c>a)&&( c>b))
        printf("\n Largest of three values is %d \n",c);
    getch();
}
```

**Output:**

Enter three numbers  
**5 7 6**  
Largest of three values is 7

**11. C Program using switch statement to grade the students marks as A =80 and above, B = 70 and above, C = 60 and above , D =40 and above and the rest below 40 as E grade.**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int m,c;
    clrscr();
    printf ("\n Enter marks : ");
    scanf ("%d",&m);
    if(m>=80) c=1;
    else if (m>=70) c =2 ;
    else if (m>=60) c =3;
    else if (m>=40) c = 4;
    switch (c)
    {case 1: printf ("\n Grade A\n");
        break;
    case 2: printf ("\n Grade B\n");
        break ;
    case 3: printf ("\n Grade C\n");
        break;
    case 4:printf ("\n Grade D");
        break ;
    default : printf ("\n Grade E.");
    }
    getch();
}
```

**Output:**

Enter marks :77  
**Grade B**

**12. Given that a perfect square is an integer which is the square of another integer , write a program in C to check whether a given integer is a perfect square or not.**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    int n,m;
    clrscr();
    printf("\n Enter n\n");
    scanf("%d",&n);
    m = (int) sqrt(n);
    if( n ==m*m)
        printf("n is perfect square \n");
    else
        printf(" n is not a perfect square \n");
    getch();
}
```

**Output:**

```
Enter n
16
n is perfect square

Enter n
5
n is not a perfect square
```

#### **2.4.2 Programming Examples based on While Loop**

- 1. C program to find the factorial of a number, where the number is entered by user.**

```
#include <stdio.h>
#include<conio.h>
main()
{
    int    number, factorial;
    clrscr();
    printf("Enter a number.\n");
    scanf("%d",&number);
    factorial=1;
    while (number>0)
    { factorial=factorial*number;
        number =number-1;
    }
    printf("Factorial=%d",factorial);
    getch();
}
```

#### **Output:**

```
Enter a number
5
Factorial = 120
```

- 2. C program to find the sum of the digits of a given number.**

```
#include <stdio.h>
#include<conio.h>
int main()
{
    int    n, t, sum = 0, remainder;
    clrscr();
    printf("Enter an integer\n");
    scanf("%d", &n);
    t = n;
    while (t != 0)
    { remainder = t % 10;
        sum = sum + remainder;
```

```

        t = t / 10;
    }
    printf("\nSum of digits of %d = %d\n", n, sum);
    getch();
}

```

**Output:**

Enter an integer  
**125**  
 Sum of digits of 125 = 8

**3. C program to find hcf and lcm of a given two numbers.**  
 (NOTE. HCF is also known as greatest common divisor(GCD) or greatest common factor(gcf) ).

```

#include <stdio.h>
#include<conio.h>
int main()
{
    int a, b, x, y, t, gcd, lcm;
    clrscr();
    printf("Enter two integers\n");
    scanf("%d%d", &x, &y);
    a = x;
    b = y;
    while (b != 0)
    {
        t = b;
        b = a % b;
        a = t;
    }
    gcd = a;
    lcm = (x*y)/gcd;
    printf("Greatest common divisor of %d and %d = %d\n", x, y,gcd);
    printf("Least common multiple of %d and %d = %d\n", x, y,lcm);
    getch();
}

```

**Output:**

Enter two integers  
**30 40**  
 Greatest common divisor of 30 and 40 = 10  
 Least common multiple of 30 and 40 = 120

#### **4. C Program to reverse a given number**

```
#include <stdio.h>
#include<conio.h>
main()
{
    int n, reverse = 0;
    clrscr();
    printf("Enter a number to reverse\n");
    scanf("%d", &n);
    while (n != 0){ reverse = reverse * 10;
                    reverse = reverse + n%10;
                    n = n/10;
    }
    printf("Reverse of entered number is = %d\n", reverse);
    getch();
}
```

#### **Output:**

```
Enter a number to reverse
1975
Reverse of entered number is = 5791
```

#### **5. To check whether the given number is palindrome or not. (Note : IF the reverse of a given number is same such number is called palindrome)**

```
#include <stdio.h>
#include<conio.h>
main()
{
    int n, reverse = 0, temp;
    clrscr();
    printf("Enter a number to check if it is a palindrome or
not\n");
    scanf("%d",&n);
    temp = n;
    while( temp != 0 ) {
        reverse = reverse * 10;
        reverse = reverse + temp%10;
        temp = temp/10;
    }
}
```

```

if( n == reverse )
    printf("%d is a palindrome number.\n", n);
else
    printf("%d is not a palindrome number.\n", n);
getch();
}

```

}

### Output:

Enter a number to check if it is a palindrome or not  
**1221**  
1221 is a palindrome number.

- 6. To check whether the given number is Armstrong or not.(A number is Armstrong if the sum of cubes of individual digits of a number is equal to the number itself. For example 371 is an Armstrong number as  $3^3 + 7^3 + 1^3 = 371$ . Some other Armstrong numbers are: 0, 1, 153, 370, 407)**

```

#include <stdio.h>
#include<conio.h>
main()
{
    int    number, sum = 0, temp, remainder;
    clrscr();
    printf("Enter an integer\n");
    scanf("%d",&number);
    temp = number;
    while( temp != 0 )
    {
        remainder = temp%10;
        sum = sum + remainder*remainder*remainder;
        temp = temp/10;
    }

    if( number == sum )
        printf("Entered number is an Armstrong number.\n");
    else
        printf("Entered number is not an Armstrong
number.\n");

    getch();
}

```

**Output:**

```
Enter an integer  
153  
Entered number is an armstrong number
```

**7. C program to calculate the power of an integer.**

```
#include <stdio.h>
#include<conio.h>
main()
{
    int base, exp;
    long int value =1;
    clrscr();
    printf("Enter base number and exponent respectively: ");
    scanf("%d%d", &base, &exp);

    while (exp!=0)
    {
        value*=base; /* value = value*base; */
        exp=exp -1;
    }
    printf("Answer = %ld", value);
    getch();
}
```

**Output:**

```
Enter base number and exponent respectively  
2 5  
Answer = 32
```

**8. Write a C program to calculate the value of cosine function using the series  $\cos(x) = 1-x^2/2! + x^4/4! -x^6/6!$  -----to a given accuracy.**

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float x,acc,sum,term;
    int i;
    clrscr();
    printf("\nEnter x");
    scanf("%f",&x);
    x = 3.14*x/180;
    sum =0;
    acc =0.00001;
    i = 0;
    term =1;
    while(fabs(term)>=acc)
    {
        sum = sum+ term ;
        i++;
        term = -term*x*x/((2*i)*(2*i-1));
    }
    printf("\nThe value of cosx using series =%d",sum);
    printf("\n Using library function cos(x) = %f",cos(x));
    getch();
}

```

### Output:

```

Enter x 90
The value of cosx using series = -0.23
Using Library function cos(x) = -0.000204

```

## 9. C program to convert a given decimal number into its binary equivalent.

```

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

```

```
long int n,temp,d,dig,i;
clrscr();
printf("n Enter an integer number n:");
scanf("%ld",&n);
temp = n;
d = 0;
i = 1;
while(n!=0)
{
    dig = n%2;
    d = d + dig*i;
    i = i*10;
    n = n/2;
}
printf("\n Binary equivalent of %ld is %ld",temp,d);
getch();
}
```

**Output:**

```
Enter an integer number n 20
Binary equivalent of 20 is 10100
```

### 2.4.3 Programming Examples based on do While Loop

1. C program to find the sum of all the numbers entered by the user (excluding zero).

```
#include <stdio.h>
#include<conio.h>
main()
{
    int sum=0,num;
    clrscr();
    do
    {
        printf("Enter a number\n");
        scanf("%d",&num);
        sum+=num;
    } while(num!=0);

    printf("\n Sum=%d",sum);
    getch();
}
```

#### Output:

```
Enter a number    20
Enter a number    10
Enter a number    5
Enter a number    35
Enter a number    20
Enter a number    0
Sum =  90
```

2. C Program to find the sum of numbers from 10 to 20

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

int main ()
{
    int a = 10,sum=0;
    clrscr();
    do
    {
        sum = sum +a;
        a = a + 1;
```

```
    }while( a <=20 );
    printf("\n Sum =%d\n",sum);
    getch();
}
```

**Output:**

```
Sum = 165
```

**3. What is the output of the following programs.**

a.

```
#include <stdio.h>
#include<conio.h>
main()
{
    int   i = 10;
    do
    {
        printf("Hello %d\n", i );
        i = i -1;
    }while ( i > 0 );
}
```

**Output:**

```
Hello 10
Hello 9
Hello 8
Hello 7
Hello 6
Hello 5
Hello 4
Hello 3
Hello 2
Hello 1
Hello 0
```

b.

```
#include <stdio.h>
main()
{
    int   i = 10;

    do{printf("Hello %d\n", i );
       i = i -1;
```

```
if( i == 6 )
{break;}
}while ( i > 0 );
}
```

**Output:**

```
Hello 10
Hello 9
Hello 8
Hello 7
```

#### **2.4.4 Programming Examples based on for Loop**

##### **1. C program to find the factorial of a number using for loop.**

```
#include <stdio.h>
#include<conio.h>
int main()
{ int c,n,fact = 1;
clrscr();
printf("Enter a number to calculate it's factorial\n");
scanf("%d", &n);
for (c = 1; c <= n; c++)
fact = fact * c;
printf("Factorial of %d = %d\n", n, fact);
getch();
}
```

##### **Output:**

```
Enter a number to calculate its factorial
5
Factorial of 5 =120
```

##### **2. C program to display character from A to Z using loops .**

```
#include<stdio.h>
#include<conio.h>
int main()
{ char c;
clrscr();
for(c='A'; c<='Z'; ++c)
printf("%c ",c);
getch();
}
```

##### **Output:**

```
ABCDEFGHIJKLMNPQRSTUVWXYZ
```

**3. Write a C program to find the GCD of two numbers using ternary operator and for loop.**

```
#include <stdio.h>
#include<conio.h>
int main()
{
    int num1, num2, min,i;
    clrscr();
    printf("Enter two integers: ");
    scanf("%d %d",&num1,&num2);
    min=(num1>num2)?num2:num1;
    /* minimum is determined using ternary operator */
    for(i=min;i>=1;--i)
    {
        if(num1%i==0 && num2%i==0)
        {
            printf("GCD of %d and %d is %d", num1, num2,i);
            break;
        }
    }
    getch();
    return 0;
}
```

**4. C program to check whether the given number is prime or not .**

```
#include <stdio.h>
#include<conio.h>
int main()
{
    int n, i, flag=0;
    clrscr();
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    for(i=2;i<=n/2;++i)
```

```

    { if(n%i==0)
    { flag=1;
      break;
    }
  }
  if (flag== 0)
    printf("%d is a prime number.",n);
  else printf("%d is not a prime number.",n);
  getch();
}

```

### Output:

Enter a positive integer  
**10**  
**10** is not a prime number

## 5. C program to convert decimal(<32) to binary

```

#include <stdio.h>
#include<conio.h>
int main()
{
  int n, c, k;
  clrscr();
  printf("Enter an integer in decimal number system\n");
  scanf("%d", &n);
  printf("%d in binary number system is:\n", n);
  for (c = 31; c >= 0; c--)
  {
    k = n >> c;
    if (k & 1)
      printf("1");
    else
      printf("0");
  }printf("\n");
  getch();
}

```

## 6. C Program to generate Fibonacci series (Numbers of Fibonacci sequence are known as Fibonacci numbers. First few numbers of series are 0, 1, 1, 2, 3, 5, 8 etc., Except first two terms in sequence every other term is the sum of two previous terms, For example 8 = 3 + 5 (addition of 3, 5)) .

```

#include<stdio.h>
#include<conio.h>
main()
{
    int n, first = 0, second = 1, next, c;
    clrscr();
    printf("Enter the number of terms\n");
    scanf("%d",&n);
    printf("First %d terms of Fibonacci series are :-\n",n);
    for ( c = 0 ; c < n ; c++ )
    { if ( c <= 1 )
        next = c;
        else
        {next = first + second;
         first = second;
         second = next;
        }
        printf("%d ",next);
    }
    getch();
}

```

### Output:

```

Enter the number of terms
5
First 5 terms of Fibonacci series are :-
0 1 1 2 3

```

### 7. Write a C program to compute all the prime numbers between a and b .

```

#include<stdio.h>
#include<conio.h>
void main()
{ int a,b,i,n,flag,count;
    clrscr();
    printf("\nEnter a and b \n");
    scanf("%d %d", &a,&b);
    count = 0 ;
    for(n=a;n<=b;n++)
    {
        flag = 1 ;

```

```

for(i =2;i<=n/2;i++)
{
    if(n%i==0)
        flag =0;
    if(flag ==1)
    {
        printf("\n %d ",n);
        count++;
    }
}
printf("\n Number of prime numbers is %d ",count);
getch();
}

```

**Output:**

Enter a and b  
**2 10**  
2 3 5 7  
Number of prime numbers is 4

**8. C program to evaluate the following series  $f(x) = x - x^3/3! + x^5/5! - x^7/7!$  ----- upto given number of terms.**

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float sum,term,x,n,i ;
    clrscr();
    printf("\n Enter the value of x:");
    scanf("%f",&x);
    printf("\n Enter the value of n:");
    scanf("%f",&n);
    sum = 0;
    term = x;
    for(i =1;i<=n;i++)
    printf("%0.2f\n",x, fun(x));
    getch();
}

float fun(float x)
{
    float sum,term ,i;
    sum =0;
    x= x*3.14/180;
}

```

```

term =1;
for(i=1;i<=n;i++)
{
    sum = sum + term;
    term = -term *x*x/((2*i)*(2*i -1));
}
return sum;
}

```

**9. Write a ‘C’ function to find  $x^y$  without using standard functions**

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int x,y;
    int fun(int x,int y);
    clrscr();
    printf("\nEnter the value of x and y \n");
    scanf("%d %d",&x,&y);
    printf("\nx to the power of y is %d", fun(x,y));
    getch();
}

int fun(int x, int y )
{
    int s,i;
    s = 1;
    for(i=1;i<=y;i++)
        s =s *x;
    return s;
}

```

**Output:**

```

Enter the value of x and y
2 3
x to the power of y is 8

```

## 2.4.5 Programming Examples based on nested loops.

### 1. C program to print the following Pyramid like pattern

```
*  
***  
*****  
*****  
*****  
*****  
*****  
#include <stdio.h>  
#include<conio.h>  
int main()  
{  
    int row, c, n, temp;  
    clrscr();  
    printf("\n Enter the number of rows in pyramid of stars : ");  
    scanf("%d",&n);  
    temp = n;  
    for( row =1; row <= n ; row++)  
    { for( c =1; c < temp ; c++)  
        printf(" ");  
        temp--; // number of initial spaces should go on decreasing , so temp--  
        for(c =1; c <=2*row -1; c++)  
            printf("*"); // no of * should go on increasing in terms of 2*  
        row  
        printf("\n"); // number-1(1,3,5....)  
    }  
    getch();  
}
```

### Output:

Enter the number of rows in pyramid of stars : 5

```
*  
***  
*****  
*****  
*****  
*****
```

## 2. C program to print the following pattern

```
*  
**  
***  
****  
  
#include <stdio.h>  
#include<conio.h>  
int main()  
{  
    int n, c, k;  
    clrscr();  
    printf("Enter number of rows\n");  
    scanf("%d",&n);  
    // printing starts from the very first column only. the number of *  
    // increase linearly 1, 2, 3,4  
    for ( c = 1 ; c <= n ; c++ )  
    { for( k = 1 ; k <= c ; k++ )  
        printf("*");  
        printf("\n");  
    }  
    getch();  
}
```

### Output:

Enter the number of rows

```
4  
*  
**  
***  
****
```

## 3. C program to print the following pattern

```
*  
*A*  
*A*A*  
*A*A*A*  
#include<stdio.h>  
#include<conio.h>  
main()  
{  
    int n, c, k, space, count = 1;
```

```

clrscr();
printf("\nEnter number of rows:");
scanf("%d",&n);
space = n;
for ( c = 1 ; c <= n ; c++)
{
    for( k = 1 ; k < space ; k++)
        printf(" ");
    for ( k = 1 ; k <= c ; k++)
        {printf("*");
        if( c > 1 && count < c)
            { printf("A");
            count++;}
        }
    printf("\n");
    space--;
    count = 1;
}
getch();
}

```

### Output:

Enter the number of rows: 4

```

*
*A*
*A*A*
*A*A*A*

```

- 4. C program to print the Floyd's triangle. (This program prints Floyd's triangle. Number of rows of Floyd's triangle to print is entered by the user. First four rows of Floyd's triangle are as follows:-**

```

1
2 3
4 5 6
7 8 9 10

```

**It's clear that in Floyd's triangle nth row contains n numbers. )**

```

#include <stdio.h>
#include<conio.h>
int main()

```

```

{
    int n, i, c, a = 1;
    clrscr();
    printf("\nEnter the number of rows of Floyd's triangle to print:");
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
    {
        for (c = 1; c <= i; c++) // c represents the line number
        {
            printf("%d ", a); // initial value of a =1
            a++; // Each time prints incremented a value for i
            times
        }
        printf("\n");
    }
    getch();
}

```

### Output:

Enter the number of rows of Floyds triangle to print:**4**

**1**  
**2 3**  
**4 5 6**  
**7 8 9 10**

### 5. C Program to print Pascal Triangle. First four rows of Pascal triangle are shown below :-

```

      1
     1 1
    1 2 1
   1 3 3 1

```

```

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

int factorial(int);
int main()
{

```

```

int i, n, c;
clrscr();
printf("\nEnter the number of rows you wish to see in pascal
triangle:");
scanf("%d",&n);

for (i = 0; i < n; i++) // for line number
{
    for (c = 0; c <= (n - i - 2); c++) // To create space
        printf(" ");
    for (c = 0 ; c <= i; c++) // To generate the numbers
        printf("%d ",factorial(i)/(factorial(c)*factorial(i-c)));
    printf("\n");
}
getch();
}

int factorial(int n)
{
    int c;
    long result = 1;
    for (c = 1; c <= n; c++)
        result = result*c;
    return result;
}

```

### Output:

Enter the number of rows of you wish to see in pascal triangle: 4

|         |
|---------|
| 1       |
| 1 1     |
| 1 2 1   |
| 1 3 3 1 |

### 6. C program to print the following diamond pattern

```

*
***
```

```

*****
 ***
 *
#include <stdio.h>
#include<conio.h>
int main()
{
    int n, c, k, space = 1;
    clrscr();
    printf("Enter number of rows\n");
    scanf("%d", &n);
    space = n - 1;
    for (k = 1; k <= n; k++)
    {
        for (c = 1; c <= space; c++)
            printf(" ");
        space--;
        for (c = 1; c <= 2*k-1; c++)
            printf("*");
        printf("\n");
    }
    space = 1;
    for (k = 1; k <= n - 1; k++)
    {
        for (c = 1; c <= space; c++)
            printf(" ");
        space++;
        for (c = 1 ; c <= 2*(n-k)-1; c++)
            printf("*");
        printf("\n");
    }
    getch();
}

```

## 7. C Program to print the multiplication table upto n using nested for loops.

```

#include<stdio.h>
#include<conio.h>
main()
{
    int i,j,n;

```

```
clrscr( );
printf("nEnter the value of n\n");
scanf("%d",&n);
printf("n The Multiplication Table upto %d\n");
for(i = 1;i<=n;i++)
{
    for(j=1;j<=10;j++)
    {
        printf("%d\t",i*j);
    }
    printf("\n");
}
getch();
}
```

### Output:

Enter the value of n: 4

The Multiplication Table upto 4

|   |   |    |    |    |    |    |    |    |    |
|---|---|----|----|----|----|----|----|----|----|
| 1 | 2 | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
| 2 | 4 | 6  | 8  | 10 | 12 | 14 | 16 | 18 | 20 |
| 3 | 6 | 9  | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |

#### 2.4.6 Programming Examples on Summation of Series

1. Write a C program to find out the sum of series  $1 + 2 + \dots + n$ .

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int n,i;
    int sum=0;
    clrscr();
    printf("Enter the n i.e. max values of series: ");
    scanf("%d",&n);
    sum = (n * (n + 1)) / 2;
    printf("Sum of the series: ");
    for(i = 1;i <= n;i++)
    {
        if(i!=n)
            printf("%d + ",i);
        else
            printf("%d = %d ",i,sum);
    }
    getch();
}
```

#### Output:

Enter the n i.e. maximum values of series

4

Sum of the series :  $1+2+3+4 = 10$

2. Write a C program to find the sum of series  $1^2 + 2^2 + \dots + n^2$ .

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int n,i;
    int sum=0;
    clrscr();
    printf("\nEnter the n i.e. max values of series: ");
    scanf("%d",&n);
```

```

sum = (n * (n + 1) * (2 * n + 1)) / 6;
printf("\nSum of the series : ");
for(i = 1; i <= n; i++)
{
    if (i != n)
        printf("%d^2 + ", i);
    else
        printf("%d^2 = %d ", i, sum);
}
getch();
}

```

### Output

Enter the n i.e. max values of series: 4

Sum of the series :  $1^2 + 2^2 + 3^2 + 4^2 = 30$

### 3. C program to find the sum of series $1 + 1/2 + 1/3 + 1/4 + \dots + 1/N$ .

```

#include <stdio.h>
#include<conio.h>
void main()
{
    double number, sum = 0, i;
    clrscr();
    printf("\nEnter the number ");
    scanf("%lf", &number);
    for (i = 1; i <= number; i++)
    {
        sum = sum + (1 / i);
        if (i == 1)
            printf("\n 1 + ");
        else if (i == number)
            printf(" (1 / %lf)", i);
        else
            printf(" (1 / %lf) + ", i);
    }
    printf("\nThe sum of the given series is %.2lf", sum);
    getch();
}

```

### Output

Enter the number : 4

1 + (1/2.000000) + (1/3.000000) + (1/4.000000)

The sum of the given series is 2.08

#### 4. C program to find the sum of cos(x) series .

```
#include<stdio.h>
#include<math.h>
void main()
{
    int    n,x1,i,j;
    float x,sign,sum,fact;
    clrscr();
    printf("Enter the number of the terms in a series\n");
    scanf("%d",&n);
    printf("Enter the value of x(in degrees)\n");
    scanf("%f",&x);
    x1=x;
    x=x*(3.142/180.0); /* Degrees to radians*/
    sum = 1;
    sign = -1;
    for(i =2; i<=n; i=i+2)
    {
        fact = 1;
        for(j=1;j<=i;j++)
        {
            fact=fact*j;
        }
        sum=sum +(pow(x,i)/fact)*sign;
        sign= sign*(-1);
    }
    printf("Sum of the cosine series= %7.2f\n",sum);
    printf("The value of cos(%d) using library function =
%f\n",x1,cos(x));
    getch();
}
```

**Output**

Enter the number of the terms in a series

3

Enter the value of x(in degrees)

90

Sum of the cosine series = -0.23

The value of cos(90) using library function = -0.000204

## 2.5 Exercises

### Part A: Questions Based on Theoretical Concepts

1. What is two way selection statement ? Explain the two way selection (if, if else, nested if else and cascaded if else ) in C language with syntax and examples.
2. Write a short note on dangling if else.
3. Explain the multi way decision (switch)statement with syntax and example.
4. What is conditional and unconditional branching statements.
5. Explain the unconditional branching statements (goto, break and continue ) with syntax and examples.
6. Explain the different types of loops (for loop ,while and do while loop) in C with syntax and examples.
- 7.What are nested loops. Explain with example.
8. Explain how break and continue is used in a C program with example.
9. Differentiate ( or compare ) the following C constructs:
  - a. if and if else
  - b. if else and nested if else
  - c. nested if else and cascaded if else
  - d. while and do while
  - e. while and for loop
  - f. break and continue
10. What is goto statement . Explain the usage of goto in C programming language with syntax and example. What are disadvantages of goto statement.
11. What is ternary operator ? Explain how the ternary operator can be used to find the smallest of three integers.

## **Part B: Questions based on C Programs**

1. Write a C program that takes three coefficients (a ,b and c) of a quadratic equation :  $ax^2+bx+c$  as input and compute all possible roots and print them with appropriate messages. Also the program must consider only non-zero coefficient a. The program must capable to throw an error message if the user enters 0 for a.
2. Design and develop a C program to read a year as an input and find it whether the year is leap year or not. Also consider the end of centuries.
3. Write a C program to check whether the given number is a palindrome or not.
4. Write a C program to find the square root of a given number without using library functions.
5. Write a C program to find the value of  $\sin(x)$  using summation of series and without using library function.
6. Write a C program to find the maximum of two numbers.
7. Write a C program to find the smallest of three numbers.
8. Write a C program to find the smallest of two number using conditional (Ternary)operator.
9. Write a C program to find the largest of three numbers using conditional (Ternary) operator.
10. Write a program to check whether the given number is odd or even.
11. Write a program to check whether the given number is positive, negative or neutral (zero).
12. Given three sides of a triangle, check whether the triangle can be formed or not? If it can be formed, what is the type of the triangle? i.e. whether the triangle is equilateral, isosceles , scalene ? Is the given triangle is right angled triangle? Write the program to implement the same.

13. Write a program to enter a character and then determine whether it is a vowel or not.
14. A company decides to give bonus to all its employees on Diwali. A 5% bonus on salary is given to the male workers and 10% bonus of salary to the female workers. Write a program to enter the salary and sex of the employee. If the salary of the employee is less than Rupees 10,000 then the employee get an extra 2% bonus on salary. Calculate the bonus that has to be given to the employee and display the salary that the employee will get.
15. Write a program to determine whether an entered character is vowel or not using switch statement.
16. Write a program to enter a number from 1 to 7 and display the corresponding day of the week using switch statement.
17. Write a program that accepts a number from 1 to 10. Print whether the number is even or odd using a switch case construct.
18. Write a program to simulate a simple (rudimentary) calculator using switch statement.
19. Write a C program to enter a number from 1 to 7 and display the corresponding day of the week any switch case statement
20. Write a C program to find the factorial of a number, where the number is entered by user.
21. Write a C program to find the sum of the digits of a given number.
22. Write a C program to find HCF and LCM of a given two numbers.  
(NOTE. HCF is also known as greatest common divisor(GCD) or greatest common factor (GCF)).
23. Write a C Program to reverse a given number .
24. Write a C Program to check whether the given number is palindrome or note. (Note : IF the reverse of a given number is same such number is called palindrome)

25. Write a C Program to check whether the given number is Armstrong or not. (A number is armstrong if the sum of cubes of individual digits of a number is equal to the number itself. For example 371 is an armstrong number as  $3^3 + 7^3 + 1^3 = 371$ . Some other armstrong numbers are: 0, 1, 153, 370, 407 )
26. Write a C program to calculate the power of an integer.
27. Write a C program to find the sum of all the numbers entered by the user (excluding zero).
28. Write a C Program to find the sum of numbers from 10 to 20
29. Write a C program to find the factorial of a number using for loop.
30. Write a C program to display character from A to Z using loops .
31. Write a C program to check whether the given number is prime or not.
32. Write a C program to convert decimal number into binary number.
33. Write a C program to generate Fibonacci series.
34. Write a C program to find out the sum of series  $1 + 2 + \dots + n$
35. Write a C program to find the sum of series  $1^2 + 2^2 + \dots + n^2$ .
36. Write a C program to find the sum of series  $1 + 1/2 + 1/3 + 1/4 + \dots + 1/N$ .
37. Write a C program to find the sum of  $\cos(x)$  series

## Module 3: Arrays, Strings and Functions

**Syllabus :** *Arrays and Strings : Using an array, Using arrays with functions, Multi – Dimensional arrays, Strings : Declaring, Initializing, Printing and reading strings, string manipulation function, string input and output functions, arrays of strings, Programming Examples and Exercises.*

**Functions:** *Functions in C, Argument Passing – call by value, call by reference, void and parameter less Functions, Recursion, Programming examples and exercises.*

### 3.1 Arrays

**3.1.1 Introduction to Arrays:** An array is a data structure which is used to store a multiple values of similar data type using a single variable. It is a derived data type which is used to store, process and print large amount of data using a single variable. When the elements of an array are integers or floating points, the array is called **numerical array** and when the elements of an array are characters the array is called **strings or character array**.

Each element of the array has its own storage location, which can be referenced like any other variable. The elements of an array are stored in consecutive storage locations in memory and identified by the name of the array followed by a number in brackets; these number start at 0.

**3.1.2 Using an Array:** Consider an array named **n** which contains five elements. The five elements of n are represented as  $n[0]$ ,  $n[1]$ ,  $n[2]$ ,  $n[3]$  and  $n[4]$ . The number written in the square brackets 0,1,2,3,4 is called subscript or an index . The elements of array are called subscripted variables. The subscripts of all arrays in C are numbered starting with 0 and increase each time by 1.

**Example :** Consider an array of 5 integers ,  $n[5] = \{35,40,20,57,19\}$

Here  $n[0] = 35$ ,  $n[1] = 40$ ,  $n[2] = 20$ ,  $n[3] = 57$  and  $n[4] = 19$

|      |    |
|------|----|
| n[0] | 35 |
| n[1] | 40 |
| n[2] | 20 |
| n[3] | 57 |
| n[4] | 19 |

**Some of examples where the concept of an array can be used are :**

1. List of employees in a company
2. List of students in a class
3. List of customers

**Following examples illustrate the different ways of using arrays during programming:**

1.  $\text{num}[3] = 5;$
2.  $\text{printf}(\text{"%d"}, \text{num}[4]);$
3.  $i = 4;$   
 $\text{num}[i] = 7;$
4.  $r = 2 * n[0];$

**Rules to be followed while using arrays:**

- The array should be declared with some data type.
- All the arrays elements should be of same data type.
- The size of the array is finite and it should be specified at the time of its declaration
- In an array, elements are stored sequentially that is in contiguous memory locations
- Only one element can be added or removed from the array at a time.
- The subscript of first item is always zero.
- Each data item is accessed using the name of the array but with different subscripts
- The index of the array is always an integer.

**Types of Arrays:** Arrays are classified into two types: 1. Single dimensional Arrays 2. Multidimensional Array

**3.1.3 Single Dimensional Arrays:** Single dimensional array is one in which the array variable uses single subscript and single pair of square brackets to store multiple values of similar data types .

**Example:** x[10], std[10], usn[10] etc. are examples for single dimensional array . The elements are stored contiguously one after the other as illustrated below:

|      |      |      |      |      |
|------|------|------|------|------|
| 10   | 20   | 15   | 30   | 35   |
| x[0] | x[1] | x[2] | x[3] | x[4] |

The elements of single dimensional array can be used in programs just like any other C variable. For example the following are valid statements.

- $a = x[0] + 10;$
- $x[4] = p[0] + r[2] ;$
- $x[2] = n[i]*3$

**Declaration of one Dimensional Arrays:** To use an array we must declare it. Like any other variable arrays must be declared before they are used so that the compiler can allocate space for them in memory. The general form of array declaration is

**type variable\_name[SIZE];**

Here type is data type like **int** or **char** or **float** . Size is the maximum number of elements that can be stored inside the array

**Example :** float height[50];

int group[10];

char name[10];

**Note :** When declaring ***character arrays*** we must allow one extra element space for null terminator.

**Initialization of One Dimensional Arrays:** After an array is declared its element must be initialized. Otherwise they will contain “garbage” values. An array can be initialized at either of the following stages :

1. At compile time
2. At run time

**1. Compile time Initialization:** In this type, the initialization is done during declaration or anywhere inside the program before running a program. The general form of initialization of arrays during compile time is :

```
type variable_name[SIZE] = { list of values };
```

**Examples :**

```
int    number[3] = {0,0,0}
float  total[5] = {0.0,15.75,-10}
int    counter[] = {1,1,1,1}
char   name[] = { 'J','o','h','n','\0'};
char   name[] = "John";
```

**Arrays can also be partially initialized as illustrated below:**

```
int n[5] = {10,20}; Here the remaining elements will be initialized to zero
```

```
char city[5] ={'B'}; Remaining elements will be initialized to null .
```

**Note:** The type of initialization `n[3] = {10,20,30,40}` is illegal in C programming language.

**2. Run Time Initialization:** In this type, the initialization is done during run time or program execution. This approach is usually applied for initializing large arrays. Following example illustrate the run time initialization of arrays.

**Example :**

```
for (i=0; i<100 ;i=i+1)
{
    if (i<50)
        sum[i] = 0.0;
    else
        sum[i] =1.0;
}
```

**3.1.4 Array of Characters:** If the elements of array is made up of characters , the array is said to be character array .

**Declaration :** An array of characters is declared like other types using the syntax : **char name[size];**      *Ex: char message[20];*

**Initialization:** One can assign values to each array element, either in the declaration or through assignment statements. The null character '\0' is treated as a single character in C. The following example illustrates the possible ways of initializing a character array ,say message.

```
char message[20] = {‘w’,’e’,’i’,’r’,’d’};

message[5] = ‘ ‘ ;
message[6] = ‘s’;
message[7] = ‘t’ ;
message[8] = ‘u’ ;
message[9] = ‘f’ ;
message[10] = ‘f’ ;
message[11] = ‘\0’ ;
```

**Printing the Characters in the Array:** The characters in the array can be printed in any one of the following alternatives:

**Alternative 1: Using While Loop**

```
i =0 ;
while(message[i]!='\0')
{
    printf(“%c”,message[i]);
}
```

### **Alternative 2: Using for Loop**

```
for(i=0;message[i]!='0';i++)
printf("%c",message[i]);
```

### **Alternative 3: Using conversion specifier %s**

```
printf("%s",message);
```

**3.1.5 Expressions as Subscripts:** A subscript in an array can be an expression as long as it evaluates to an integer.

**Example 1 :** int i = 4, item[20];

```
item[2*i +3] = 76 ;
// here the expression 2*i+3 evaluates to 2*4+3 = 11
```

**Example 2 :** char letters[8];

```
int i;
for(i = 0; i < 7; i++)
if(letters[i] == letters[i+1])
printf("%c is repeated \n", letters[i]);
```

**3.1.6 Parallel Arrays :** Two arrays which use the same variable as a subscript in a loop are called *parallel arrays*.

**Example :**

```
int i;
int idnum[20];
double average[20];

for(i=0;i<20;i++)
printf("%d and %f\n", idnum[i],average[i]);
```

Here *idnum* and *average* uses the same subscript *i* and hence they are called as *parallel arrays* .

**3.1.7 Using the constant as the Array Size :** The constant can also be used to define the array size . But the constants must have been defined

earlier before the usage of the constants in the array subscript as illustrated below:

```
#define NUM 150  
int test[NUM];
```

**Note :** The following way of declaration is an illegal way of declaration :

```
int num=150;  
int test[num];
```

### 3.1.8 Searching and Sorting

**a. Searching:** Searching is a process of finding a given element in an array of elements. If the element is found in an array then the search is said to be **successful** and if the element is not found then the search is said to be **unsuccessful**. The various searching techniques available are

1. Linear search
2. Binary search
3. Depth First Search
4. Breadth First search

**1. Linear Search:** In this linear search technique the element to be searched (Key) is compared with each item of the array sequentially one after the other. If the key is found the search is said to be successful otherwise unsuccessful. It is also known as *Sequential Search*. This searching technique is inefficient and it is more suitable for the unordered array of smaller size.

**For example consider the following array  $a$  of size 5 which is unsorted:**

$a[0] \quad a[1] \quad a[2] \quad a[3] \quad a[4]$

|    |    |    |    |    |
|----|----|----|----|----|
| 25 | 30 | 20 | 60 | 70 |
|----|----|----|----|----|

**Case 1: The key item to be searched is 20.** The key item 20 will be compared with a[0] first. Since it does not match, the key item will be compared with next element a[1]. Since a[1] also does not match with 20, the key item will be compared with next element a[2]. Since the key item and a[2] matches , search will be ***successful search*** .

**Case 2:** The key item to be searched is 35. Since the key item 35 is not present in the array, the search becomes ***unsuccessful*** .

**Algorithm:** To search a key item , *key* in a given array **a** of size **n**.

Step 0: Start

Step 1: Set pos = -1, flag=0 ;

Step 2:Repeat for i = 0 to n-1

```
    if (key == a[i])
    {
        set pos = i+1;
        flag = 1;
        break;
    }
```

Step 3: if (flag ==1)

```
{ print " search is successful and the location of key item is
      pos";
}
```

```
else print "Unsuccessful Search"
```

Step 4: Stop

[ Note : The C program for linear search is given in the section :3.4.1(Example 6)]

2. **Binary Search:** A binary search is a simple searching technique which can be applied if the items to be compared are either in ascending order or descending order. It is the most efficient searching technique. It searches the given item in the minimum possible comparisons. To do binary search, first we had to sort the array elements.

(The method is exactly the same we follow to search a word in dictionary... first we divide the whole dictionary in two halves and compare the word to be searched with the word in the middle page. if the word is < then it implies the word is in 1st half. we virtually reduce the

*whole dictionary to 1st half ie., ( high(last page) becomes mid-1( middle page)) for further search and continue.. else we virtually reduce the whole dictionary to 2nd half(ie., low(first page) becomes mid+1(middle page) and ---)*

**Algorithm:** To search a key element in a sorted array **a** of size **n** using Binary Search Technique.

```
Step 0: Start  
Step 1: Initialize low =0 , high = n-1 , pos = -1  
Step 2: Repeat Step 3 and Step 4 While low <= high  
Step 3: Set mid = (low +high)/2  
Step 4 : if( a[mid] == key) then  
        { pos = mid ;  
          print “ the key element is present at pos” ;  
          break ;  
        }  
      else if (a[mid]> key) then  
        set high = mid -1;  
      else if (a[mid] < key ) then  
        set low = mid +1 ;  
Step 5 : if ( pos == -1) then print the key element is not present .  
Step 6 : Exit
```

**Procedure:** The program can be designed as follows: If low is compared as the position of the first element and high as the position of the last element, the position of the middle element can be obtained using the statement:

```
mid = (low+high)/2 ;
```

The key to be searched is compared with middle element. This can be done using the statement:

```
if(key==a[mid])  
{ printf(" Successful Search \n");  
exit(0);  
}  
if(key<a[mid])  
high = mid-1;  
else  
low = mid +1;
```

*[The complete C Program for Binary search is given in the section :3.4.1(Example7)]*

**Example:** Consider a sorted array  $\mathbf{a} = \{10, 30, 50, 60, 70\}$ . Where in  $a[0] = 10$ ,  $a[1] = 30$ ,  $a[2] = 50$ ,  $a[3] = 60$ ,  $a[4] = 70$ .

**Case 1:** If the key element to be searched is **60**. The Binary search first computes the value of mid and compares with the key as follows:

Initial values of low = 0 and high = 4

- **First iteration :**

mid = (low + high)/2 = 2 and  
 $a[\text{mid}] = a[2] = 50 < \text{key}$   
=> Key is found in the second half of array  
=> high = 4 and low = mid + 1 = 2+1 = 3

- **Second Iteration:**

mid = (4+3) /2 = 7/2 = 3 and  
 $a[\text{mid}] = a[3] = 60$ ,  
=> $a[\text{mid}] == \text{key}$   
=> Key is found at mid  
=>**Search is Successful.**

**Case 2 :** If the key element to be searched is 10. The Binary search first computes the value of mid and compares with the key as follows:

Initial values of low = 0 and high = 4

- **First Iteration :**

mid = (low + high)/2 = 2 and  
 $a[\text{mid}] = a[2] = 50 > \text{key}$   
=> Key is found in the first half of array  
=> high = mid-1 = 1 and low = 0

- **Second Iteration :**

mid = (0+1)/2 = 0 , and  
 $a[\text{mid}] = a[0] = 10$   
=> Key is found at mid  
=>**Search is Successful.**

**Case 3:** If the key element to be searched is 5. The Binary search first computes the value of mid and compares with the key as follows:

Initial values of low = 0 and high = 4

- **First Iteration :**

$$\text{mid} = (\text{low} + \text{high})/2 = 2 \quad \text{and}$$

$$a[\text{mid}] = a[2] = 50 > \text{key}$$

=> The key element may be found in the first half of the array

$$\Rightarrow \text{high} = \text{mid}-1 = 1 \quad \text{and low} = 0$$

- **Second Iteration**

$$\text{mid} = (0+1)/2 = 0, \quad \text{and}$$

$$a[\text{mid}] = a[0] = 10 > \text{key},$$

=> The key element may be found in the second half of the array.

$$\Rightarrow \text{high} = \text{mid} - 1 = 0 - 1 = -1 \text{ and low} = 0 \text{ which violates the condition } \text{low} \leq \text{high}$$

=> **Search is Unsuccessful**

- b. Sorting:** The process of rearranging the given elements so that they are in ascending order or descending order is called sorting. Let S be a list of n elements S<sub>1</sub>, S<sub>2</sub>, S<sub>3</sub>, ..., S<sub>n</sub> in memory. Sorting S means arranging the contents of S in either increasing or decreasing order i.e.
- S<sub>1</sub> ≤ S<sub>2</sub> ≤ S<sub>3</sub> ≤ ----- ≤ S<sub>n</sub> or

$$S_n \geq S_{n-1} \geq S_{n-2} \geq \dots \geq S_3 \geq S_2 \geq S_1$$

**Example :** Suppose an array contains 10 elements

$$\{ 7, 8, 9, 4, 6, 8, 2, 1, 17, 14 \}$$

After sorting the elements in ascending order, the array will be { 1, 2, 4, 6, 7, 8, 9, 14, 17 }

Similarly after sorting the elements in descending order the array will be { 17, 14, 9, 8, 7, 6, 4, 2, 1 }

The two important sorting techniques used are: **1. Bubble Sort 2. Selection Sort.** Here we will discuss only Bubble Sort.

**Bubble Sort:** The most widely used simple sorting technique is Bubble Sort. This technique sorts the array elements by repeatedly moving the largest element to the highest index position of the array (in case of arranging elements in ascending order). In this technique each element is compared with its adjacent element. If the first element is larger than the

second one then the position of the elements are interchanged, otherwise it is not changed. Then next element is compared with its adjacent element and the same process is repeated for all the elements in the array. During the second pass, the second largest element occupies the second last position. The same process will be repeated until all the elements occupy their proper sorted position. A general algorithm for bubble sort is as follows:

**Algorithm:** To sort elements of array **a** of size **n** in ascending order using bubble sort technique

Step 0: Start

Step 1: Read the size of array **a** , say **n**

Step 2 : Read **n** elements into array **a**.

    for (i = 0 to n )

        Read elements into array **a[i]**.

    End for

Step 3: Perform Sorting

    for(i=0 to n)

        for(j=0 to n-i )

            if (**a[j]>a[j+1]**)

                {   temp = **a[j]** ;

**a[j] = a[j+1]** ;

**a[j+1] = temp;**

            }

Step 4 : Print the sorted array **a**

Step 5: Stop

The sorting technique is called bubble sorting because the smaller elements bubble to the top of the list. In this sorting algorithm the outer loop is for ***the total number of passes which is n-1***. The inner loop will be executed for each and every pass. Also the frequency of the inner loop will decrease after every pass. This is because after every pass one element will be in its proper position. So for each and every pass the inner

loop will be executed **n-i** times, where **n** is the number of the elements in the array and **i** is the count of the pass.

**Example:** As illustrated in the algorithm two important processes swapping and comparison takes place. In this technique the two successive item  $a[i]$  and  $a[i+1]$  are exchanged whenever  $a[i] >= a[i+1]$ . For example consider the array with elements: 40,50,30,20 and 10.

The following comparisons are made during **first pass**:

$a[0]$  with  $a[1]$  i.e. 40 and 50 no interchange

$a[1]$  with  $a[2]$  i.e. 50 and 30 interchanged

$a[2]$  and  $a[3]$  i.e. 50 and 20 interchanged

$a[3]$  and  $a[4]$  i.e. 50 and 10 interchanged

Thus after the first pass, array element are: 40 30 20 10 50. The complete set of passes and iterations is shown below:

| Pass  | <b>a[0]</b> | <b>a[1]</b> | <b>a[2]</b> | <b>a[3]</b> | <b>a[4]</b> |
|-------|-------------|-------------|-------------|-------------|-------------|
| Pass1 | 40          | <b>50</b>   | <b>30</b>   | 20          | 10          |
|       | 40          | 30          | <b>50</b>   | <b>20</b>   | 10          |
|       | 40          | 30          | 20          | <b>50</b>   | <b>10</b>   |
|       | <b>40</b>   | <b>30</b>   | 20          | 10          | 50          |
| Pass2 | 30          | <b>40</b>   | <b>20</b>   | 10          | 50          |
|       | 30          | 20          | <b>40</b>   | <b>10</b>   | 50          |
|       | <b>30</b>   | <b>20</b>   | 10          | 40          | 50          |
| Pass3 | 20          | <b>30</b>   | <b>10</b>   | 40          | 50          |
|       | <b>20</b>   | <b>10</b>   | 30          | 40          | 50          |
| Pass4 | 10          | 20          | 30          | 40          | 50          |

#### **Pass 1:**

Since 50 > 30 ,  $a[1]$  and  $a[2]$  are swapped

Since 50 > 20 ,  $a[2]$  and  $a[3]$  are swapped

Since 50 > 10 ,  $a[3]$  and  $a[4]$  are swapped

#### **Pass 2:**

Since 40 > 30 ,  $a[0]$  and  $a[1]$  are swapped

Since 40 > 20 ,  $a[1]$  and  $a[2]$  are swapped

Since 40 > 10 ,  $a[2]$  and  $a[3]$  are swapped

**Pass 3:**

Since  $30 > 20$ ,  $a[0]$  and  $a[1]$  are swapped  
Since  $30 > 10$ ,  $a[1]$  and  $a[2]$  are swapped

**Pass 4 :**

Since  $20 > 10$ ,  $a[0]$  and  $a[1]$  are swapped  
Now the array gets completely sorted

The array elements in the final stage is 10 20 30 40 and 50.

*[Note: The C program for bubble sort is given in the section :3.4.1 (Example 8) ]*

### 3.1.9 Multidimensional Arrays:

The term dimension represents number of subscripts or indices to access a particular item in an array. Array with two or more dimensions are called multidimensional arrays.

**Example :**  $a[10][10], a[10][5][20], a[5][5]----[n]$ .

**Two dimensional arrays :** Arrays with two pair of square brackets and two indices, so that an element is referred by 2 indices are called two dimensional arrays.

**Example :** `int b[10][10];`

A two dimensional arrays is used when data items has to be arranged in row wise and column wise in a tabular fashion. Here to identify a particular item we have to specify two indices (also called subscripts): the first index identifies the **row** number and second index identify the **column** number of the item.

**Declaration of two dimensional arrays :** The general syntax used for declaring two dimensional arrays is :

**data\_type array\_name[row\_size][column\_size];**

**Example :** int a[3][5];

Here the array **a** has two sets of square brackets `[][]` and hence it is a 2-dimensional arrays with row size 3 and column size 5. This declaration informs the compiler to reserve 15 locations ( $3 \times 5$ ) contiguously one after the other.

**Initialization of two dimensional arrays:** Assigning required values to a variable at the time of declaration before processing is called initialization. Like the one dimensional arrays the two dimensional arrays is also initialized by following their declaration with a list of values enclosed in braces. The general syntax is as described below :

**data\_type array\_name [row][col]**

```
= { { v01,v02,-----v0n },
    { v11,v12,-----v1n },
    { v21,v22,-----v2n },
    -----
    -----
    -----
    { vm1,vm2,-----vmn },
};
```

Here  $v_{01}, \dots, v_{mn}$  are the values of two dimensional array elements.

**Example :**

```
int table[2][3] = { 0,0,0,1,1,1};
int table[2][3] = { {0,0,0} , { 1,1,1} };
int table[2][3] = { {0,0,0},
                    {1,1,1}
};
```

## Points to Remember:

- When the array is completely initialized with all values , explicitly we need not specify the size of the first dimension .

i.e. `int table[ ][3] = { {0,0,0},  
                      {1,1,1}  
                     };`

- If the values are missing in an initialization i.e. if the array is partially initialized they are automatically set to zero :

For instance the statement

```
int table[2][3] = { {1,1},  
                     {2}  
                     };
```

Will initialize the first two elements of the first row to one , the first element of second row to two and all other elements to zero .

- If all the elements are to be initialized zero , the following short cut method may be used:

```
int m[3][5] = { {0},{0},{0}};
```

The first element of each row is explicitly initialized to zero while other elements are automatically initialized to zero .The following statement will also achieve the same result : `int m[3][5] = {0,0};`

- While initializing one must make ensure that the number of elements must be always less than or equal to size of each dimension . Consider the partial initialization shown below :

```
int a[4][3] = { {1,2,3,4},  
                 {5,6},  
                 {7,8},  
                 {9,10}  
               };
```

In the array declared each row should have three columns. But in this example we are trying to initialize 1<sup>st</sup> row with 4 elements . This results in syntax error and the compiler flashes an error.

**Example:** The memory map for the following 2 dimensional arrays is as given below:

```
int [3][3] = { {1,1,1} ,  
                {2,2,2},  
                {3,3,3},  
            }
```

| a[0][0]   | a[0][1]   | a[0][2]   | a[1][0]   | a[1][1]   | a[1][2]   | a[2][0]   | a[2][1]   | a[2][2]   |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 1<br>2000 | 1<br>2002 | 1<br>2004 | 2<br>2006 | 2<br>2010 | 2<br>2012 | 3<br>2014 | 3<br>2016 | 3<br>2018 |

### Reading and Writing Two dimensional Array :

The elements of a matrix can be accessed in two different ways : **1.** Row major order **2.** Column major order

- Row major order :** Here all the elements are accessed one by one row wise i.e. all the elements in each row are accessed starting from 0<sup>th</sup> row to last row .

**Reading two dimensional arrays in row major order :** In order to read the two dimensional array say a[m][n] one can use the following programming segment :

```
for (row=0;row<m;row++)  
{ for(column=0;column<n;column++)  
{ scanf("%d",&a[row][column]);  
 }  
 }
```

### Writing /Printing two dimensional arrays in row major order :

In order to print/write the two dimensional array say a[m][n ] one can use the following programming segment , wherein the elements are accessed row wise (before column )

```

for (row=0;row<m;row++)
{
    for(column=0;column<n;column++)
    {
        printf(“ %d ”,a[row][column]);
    }
    printf(“\n”);
}

```

2. **Column major order** : Here all the elements are accessed one by one column wise i.e. all the elements in each column are accessed starting from 0<sup>th</sup> column to last column.

**Reading two dimensional arrays in column major order** : In order to read the two dimensional array say a[m][n] one can use the following programming segment , wherein the elements are accessed column wise (before row ).

```

for (column=0;column<n;column++)
{
    for(row=0;row<m;row++)
    {
        scanf(“%d”,&a[row][column]);
    }
}

```

## 3.2 Strings

**3.2.1 Introduction to strings:** A string is an array of characters terminated by null character which is denoted by the escape sequence ‘\0’. Consider a string “VISION” . This string is stored in the form of an array as shown below:

|   |   |   |   |   |   |      |
|---|---|---|---|---|---|------|
| V | I | S | I | O | N | ‘\0’ |
| 0 | 1 | 2 | 3 | 4 | 5 | 6    |

The sequence of characters enclosed within two double quotes is called string constant. Ex: “SWAMI”, “PERSEVERANCE”, “TGS-123”, “HASSAN”.

The literal string or string constant is stored in consecutive bytes in memory and the compiler places the null character at end. Figure below shows the storage of a literal string “ RAJU -1075”:

|   |   |   |   |   |   |   |   |   |      |
|---|---|---|---|---|---|---|---|---|------|
| R | A | J | U | - | 1 | 0 | 7 | 5 | ‘\0’ |
|---|---|---|---|---|---|---|---|---|------|

### 3.2.2 Declaring, Initializing, Printing and Reading Strings

**a. Declaring String Variables:** In C string variable is any valid C variable name and is always declared as an array of characters. The general form of declaration of a string variable is :

```
char string_name[size];
```

The size determines the number of characters in the string name including the null character.

**Examples :** char city[10];

```
char name[30];
```

**Null character:** The null character denoted as ‘\0’ marks end of the string. In some cases you must explicitly insert a null character to

terminate a string. In literal string the null character is inserted automatically.

**b. Initializing String Variables:** Like numeric arrays character arrays may be initialized when they are declared. C permits a character array to be initialized in either of the following two forms

```
char city[9] = "New York";
```

```
char city[9] = { 'N','E','W','','Y','O','R','K','\0'};
```

The declaration `char st1[ ] = {"G","O","O","D","\0"};` defines the array string as a five elements array automatically even though the size is not given.

One can declare the size much larger than the string size in the initializer i.e. the statement `char str[10] = "GOOD" ;` is permitted . In this case the computer creates a character array of size 10, places the value “GOOD” in it terminates with the null character and initializes all other remaining elements to NULL. The storage will look like

|   |   |   |   |    |    |    |    |    |    |
|---|---|---|---|----|----|----|----|----|----|
| G | O | O | D | \0 | \0 | \0 | \0 | \0 | \0 |
| 0 | 1 | 2 | 3 | 4  | 5  | 6  | 7  | 8  | 9  |

The following declaration is illegal: Here

```
char str2[3] = "GOOD" ;
```

This will result in a compile time error. Also note that we cannot separate the initialization from declaration. i.e.

```
char str3[5];  
str3 = "GOOD";
```

Similarly ,    `char s1[4] = "abc";`  
               `char s2[4] ;`  
               `s2 = s1;`

is not allowed. An array name cannot be used as the left operand of an assignment operator.

**Difference between Array of characters and string :** Unlike array of characters the string is created as a **unit** and is terminated by the **NULL** character.

- c. **Printing a string:** String is printed using a printf statement. As long as the string is terminated by a null character. The printf function prints all the character upto but not including the null character.

**Example:** char first[11] = “ANDY”;  
printf(“ The name is %s\n”,first);

**Example :** Program to store the string “**India is great**” in the array **country** and display the string.

```
main()
{
    char country[15] = “India is Great”;
    printf(“%s”,country);
}
```

- d. **Reading Strings using scanf:** The conversion specification for reading a string using scanf is **%s** just as in printf. One must **not use ampersand (&)** symbol while reading the string.

**Example :**

```
char name[20];
printf(“\n Enter the name\n”);
scanf(“%s”,name);
printf(“\n The entered name is \n”);
printf(“%s”,name);
```

**Note :** The scanf functions stops reading at the first white space character. If the input for the above program segment is MK Gandhi, the output will be MK.

### 3.2.3 String Manipulation Functions From the Standard Library

C language provides number of built in string handling functions. All the string manipulation functions are defined in the standard library **string.h** . The standard library **string.h** contains many functions for string manipulation. Some of the string handling or manipulation functions used in C are:

1. `strlen()`
2. `strcpy()`
3. `strncpy()`
4. `strcmp()`
5. `strncmp()`
6. `strcat()`
7. `strlwr()`
8. `strupr()`
9. `strrev()`

1. **strlen( )** : This function is used to find the length of the string in bytes .

The general form of a call to **strlen** is the following :

**length = strlen(str);**

The parameter to `strlen` , str is a string . The value it returns is an integer representing the current length of str in bytes excluding the null character.

**Example :**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    char str[20];
    int l;
    clrscr();
    printf("\n Enter a string \n");
    scanf("%s",str);
    l = strlen(str);
    printf("\n Length of the given string is %d",l );
    getch();
}
```

### **Output**

Enter a string  
**Program**  
Length of the given string is 7

2. **strcpy( )** : This function copies the string from one variable to another variable. The strcpy() function will copy the entire string including the terminating null character to the new location .

**General form of a call to strcpy :**

**strcpy(dest,source) ;**

here the characters in source string will get copied to destination string.

**Programming Example :**

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

void main()
{
    char str1[10],str2[10];
    clrscr();
    printf("\n Enter first string \n");
    scanf("%s",str1);
    strcpy(str2,str1);
    printf("\n The value of str1 : %s\n",str1);
    printf("\n The value of str2 is :%s\n",str2);
    getch();
}
```

### **Output**

Enter first string  
**Program**  
The value of str1 : Program  
The value of str2 is : Program

3. **strncpy()** : This function copies the string from one variable to another variable, but only up to the specified length say **n** . The general form of a call to the **strncpy** function is the following :

**strncpy(dest,source,n) ;**

Where,   dest -> Destination string  
          source -> source string  
          n-> is the number of characters

### **Programming Example :**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
    char str1[10],str2[10];
    clrscr();
    printf("\n Enter a string \n");
    scanf("%s",str1);
    strncpy(str2,str1,5);
    printf("\n Value of str1 : %s\n",str1);
    printf("\n Value of str2: %s\n",str2);
    getch();
}
```

### **Output**

```
Enter a string
PROGRAM
The value of str1 : PROGRAM
The value of str2 is : PROGR
```

4. **strcmp( )** : It is used to compare one string with another and it is case sensitive . The general form of the call to **strcmp** is the following where either parameter may be a string literal or variable :

**result = strcmp(first, second);**

The function **strcmp** returns an integer determined by the relationship between the strings pointed to by the two parameters. The result may take any of the following value :

result > 0 if first > second  
result = 0 if first ==second  
result < 0 if first < second

### Programming Example :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main( )
{
    char str1[30],str2[30];
    int n;
    clrscr();
    printf("\n Enter string one :");
    scanf("%s",str1);
    printf("\n Enter string two:");
    scanf("%s",str2);
    n = strcmp(str1,str2);
    if(n==0)
    {
        printf(" Both the strings are equal \n");
    }
    else
    {
        printf("\n Strings are not equal \n");
    }
    getch();
}
```

#### Output

```
Enter string one : Rahul
Enter string two : Modi
Strings are not equal
```

5. **strcmp( )** : It allows us to compare a block of **n** characters from one string with those in another.

**General form of a call to strcmp** : The general form of a call to the strcmp function is as follows;

**result = strcmp(firststring,secondstring,n);**

where first address and second address are pointers to strings and **n** is an integer .

### Programming Example :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
    char    str1[30],str2[30];
    clrscr();
    int    n;
    printf("\n Enter string one :");
    scanf("%s",str1);
    printf("\n Enter string two :");
    scanf("%s",str2);
    n = strncmp (str1,str2,7);
    if(n==0)
        printf("\n Both the strings are equal up to 7 characters\n");
    else
        printf("\n String are not equal ");
    getch();
}
```

#### Output

```
Enter string one : Program
Enter string two: Programming
Both the strings are equal upto 7 characters
```

6. **strcat( ):** This function is used to join (concatenate) two strings . The resulting string has only one null character at the end. The general form of the call to the strcat function is the following, where both first and second are pointers to strings :

**strcat(first, second);**

#### Example:

```
char   first[7] = "Sun";
char   second[7] = "day";
```

| first |   |   |    |   |   |   |
|-------|---|---|----|---|---|---|
| S     | u | n | \0 |   |   |   |
| 0     | 1 | 2 | 3  | 4 | 5 | 6 |

| second |   |   |    |   |   |   |
|--------|---|---|----|---|---|---|
| d      | a | y | \0 |   |   |   |
| 0      | 1 | 2 | 3  | 4 | 5 | 6 |

After calling **strcat(first, second)** , the string **first** takes the value as follows :

| first |   |   |   |   |   |    |
|-------|---|---|---|---|---|----|
| S     | u | n | d | a | y | \0 |
| 0     | 1 | 2 | 3 | 4 | 5 | 6  |

### Example Program :

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

main()
{
    char first[30],second[30];
    clrscr();

    printf("\n Enter string one :");
    scanf("%s",first);
    printf("\n Enter string two :");
    scanf("%s",second);
    strcat(first, second);
    printf("\n The concatenated string is %s\n",first);
    getch();
}
```

### Output

```
Enter string one : Rahul
Enter string two : Modi
The concatenated string is RahulModi
```

7. **strlwr( )** : This function converts uppercase characters to lowercase .

The general syntax is **strlwr(str)** :

**Example Program :**

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

main()
{
    char upr[30];
    clrscr();
    printf("\n Enter a string of upper case characters \n");
    scanf("%s",upr);
    strlwr(upr);
    printf("\n The Entered string in lower case characters is %s\n",upr);
    getch();
}
```

**Output**

```
Enter a string of upper case characters
KARAVALI
The Entered string in lower case characters is
karavali
```

8. **strupr( )** : This function converts lowercase characters to uppercase .

The general syntax is **strupr(str)** .

**Example Program :**

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

main()
{
    char lwr[30];
    clrscr();

    printf("\n Enter a string of lower case characters :");
    scanf("%s",lwr);
   strupr(lwr);
```

```
printf("\n The Entered string in upper case characters is :%s\n",lwr);
      getch();
}
```

**Output**

```
Enter a string of lower case characters: stdio
The Entered string in upper case characters is : STDIO
```

9. **strrev()** : This function reverses a given string . The general syntax is **strrev(str)** .

**Example Program :**

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

main()
{
    char a[30];
    clrscr( );
    printf("\n Enter a string \n");
    scanf("%s",a);
    strrev(a);
    printf("\n The Entered string in Reversed format is %s\n",a);
    getch();
}
```

**Output**

```
Enter a string
RAMA
The Entered string in Reversed format is AMAR
```

### 3.2.4 String Input/Output Functions

The **scanf** and **printf** functions are used to process the individual units like characters, integers, strings separated by white space characters. The individual units are called **tokens**. The **scanf** and **printf** are called **token oriented** Input and output functions. In C language the line oriented **Input/ output** is done using **gets** and **puts** functions.

**Reading and Writing using gets( ) and puts( ) :** The functions **gets()** and **puts()** are called line oriented I/O functions and can be used to process entire line. The **gets( )** function can be used to read entire line including white spaces. The **puts()** function can be used to print the line.

**Printing a string using puts() :** The general form of **puts** function is as given below :

**puts (str) ;**

The **puts** function takes a string or a pointer to string as a parameter and prints the string on the screen. In addition to printing a string it also prints the newline character giving the effect of printing an entire line.

**Reading a string:** The general form of **gets** function is as given below:

**s = gets (str); or gets(str)**

**Example :**Program to illustrate the usage of **gets** and **puts**

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

main()
{
    char str[30];
    clrscr();
    printf("\n Enter a string:");
```

```
    gets(str);
    printf("\n The Entered string is \n");
    puts(str);
    getch();
}
```

**Output**  
**Enter a string: KARAVALI**  
**The Entered string is**  
**KARAVALI**

**Example:** The following program shows how to read and display a string using gets and puts functions.

```
main()
{
    char name[30];
    clrscr();
    printf("\n Enter your name \n");
    gets(name);
    printf("\n Your name is \n");
    puts(name);
}
```

**Output**  
**Enter your name**  
**KUVEMPU**  
**Your name is**  
**KUVEMPU**

### Problems with gets:

- It does not check whether there is enough space for the space for the string which is being read in .
- When reading in data using gets , make sure that the values entered are shorter than the variable into which they are being read .

**3.2.5 Array of Strings :** A string is an array of characters . An array of strings is an array of arrays of characters. To create an array of strings we should use two dimensional array as shown below :

```
char a[row][col] ;
```

**a-** Name of the array

**row-** Number of strings

**col –** Maximum length of each string

Suppose it is required to store the names of 5 students. In such case we can have the following declarations.

```
char a[5][11] = {  
    "DURYODHANA",  
    "RAVANA",  
    "KUMBAKARANA",  
    "KEECHAKA",  
    "BAKASURA"  
};
```

The above strings are stored in a two dimensional arrays as follows :

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6  | 7 | 8  | 9 | 10 |
|---|---|---|---|---|---|---|----|---|----|---|----|
| 0 | D | U | R | Y | O | D | H  | A | N  | A | \0 |
| 1 | R | A | V | A | N | A | \0 |   |    |   |    |
| 2 | K | U | M | B | A | K | A  | R | N  | A | \0 |
| 3 | K | E | E | C | H | A | K  | A | \0 |   |    |
| 4 | B | A | K | A | S | U | R  | A | \0 |   |    |

## 3.3 Functions

**3.3.1 Introduction:** A function is a group or block of statements that perform a particular task. Functions are the main building blocks of C program. The function can be classified into two categories:

1. Library defined functions
2. User defined functions

**1. Library defined functions:** C Library that comes with C compiler has a collection of various functions which perform standard and predefined tasks. The library functions are also called *predefined or built in or in built or standard library functions.*

**Example :** pow(x,y) , sqrt(x,y) , printf() , scanf()

**Example Program :** To find the square root of a given number using library functions.

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
void main()
{
    float s,n;
    clrscr();
    printf("\n Enter the value of n ");
    scanf("%f",&n);
    s = sqrt(n);
    printf("\n Square Root (%f) = %f \n",n,s);
    getch();
}
```

**2. User defined Functions :** The function written by the programmer to do the specific tasks are called user defined functions (UDFs). The main() function is one best example for user defined function.

In the following example the program computes the sum of two numbers using user defined function add().

**Example :**

```
#include<stdio.h>
#include<conio.h>
void add( );

main()
{
    clrscr( )
    add( );
    getch( );
}

void add()
{
    int    sum,a,b;
    clrscr( );
    printf("nEnter the values of a and b\n");
    scanf("%d %d", &a,&b);
    sum = a+b;
    printf("%d",sum);
    return;
}
```

**Note : ( Called Function and Calling Function )**

A function that is invoked by writing the name of the function and its arguments is called “*called function*“ and the function which calls or invokes function is called “*calling function*”. In the above example **main** is calling function and **add()** is called function.

### **Examples of user defined functions :**

1. double cube(double n) {  
    return n\*n\*n;  
}
2. int sum(int a,int b) {  
    return (a+b);  
}
3. double square(double n) {  
    return n\*n;  
}
4. double product(double a, double b) {  
    return a + b;  
}
5. void function (int a) {  
    for(i= 0;i<a; i++)  
        printf("\*");  
    printf("\n");  
}
6. int fact (int n) {  
    int f;  
    if(n>0)  
        f = n\*f(n-1);  
    else  
        f = 1;  
    return f;  
}

**3.3.2 Elements of user defined functions:** The three elements of user defined functions are

1. Function definition
2. Function Call
3. Function declaration

1. **Function definition :** A function definition also known as function implementation shall include the following elements :

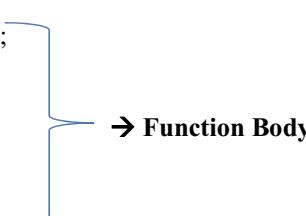
- a. Function name
- b. Function type
- c. List of parameters
- d. Local variable declarations
- e. Function statement
- f. Return statement

All the six elements are grouped into two parts namely :

- Function header (First three elements )
- Function body (Second three elements)

The **general format of a function definition** to implement these two parts is given below :

```
function_type  function_name (parameter list) → Function Header
{
    local_variable declaration ;
    executable statement1;
    executable statement2;
    -----
    -----
    return statement ;
}
```



A diagram illustrating the structure of a function definition. It shows a code snippet with curly braces {} enclosing several lines of code. A blue bracket is placed to the right of the opening brace {, spanning all the code inside, and points to the text "→ Function Body". The code itself includes a function header (function\_type function\_name (parameter list)), local variable declarations, executable statements, and a return statement.

The first line of the function definition i.e., **function type function name (parameter list )** is known as **the function header** and the statements within the opening and closing braces constitute **the function body** , which is a compound statement.

**Function Header** : The function header consists of three parts : ***the function type*** (also known as return type), ***the function name*** and the ***formal parameters list***. **Note that** a semicolon is not used at the end of the function header.

**Name and Type:** The function type specifies ***the type of value*** (like float or double) that the function is expected ***to return*** to the program calling the function. The default return type is **integer**. If the function is not returning anything then we need to specify the return type as **void**. The function name is any valid C ***identifier*** and therefore must follow the same rules of formation.

**Formal Parameter List:** The parameter list declares the variables that will receive the data sent by the calling program. Since these parameters formally represent the actual values and are not the actual values, they are called **formal parameters**. The parameters are also known as **arguments**. The parameter list contains declaration of variables separated by commas and surrounding by parameters.

### **Examples :**

```
float quadratic (int a,int b,int c) { ----}  
double power (double x,int n) { ----}  
float mu( float x,float y) { -----}  
int sum(int a,int b) {-----}
```

To indicate that parameter list is empty we use the keyword void between the parentheses as in

```
void printline(void)  
{  
-----  
}
```

**Function Body :** The function body contains the declarations and statements necessary for performing the required task . The body enclosed in braces contains three parts in the order given below:

1. **Local declaration** that specify the variables needed by the function
2. **Function statements** that perform the task of the function
3. **A return statement** that returns the value evaluated by the function

**Examples :**

```
a. float mul(float x,float y)
{
    float result ;
    result = x*y;
    return(result);
}

b. void sum (int a,int b)
{
    printf ("sum=%d",a+b);
    return;
}
c. void display (void )
{
    printf( "No type No parameters \n");
}
```

**Note :**

1. When a function reaches its return statement the control is transferred back to the calling program. In the absence of a return statement the closing brace acts as a **void return**.
2. A local variable is a variable that is defined inside a function and used without having any role in the communication between functions.

**Return statement:** A function may or may not send back any value to the calling function using return statement. One can pass any number

of values to the called function. But the called function ***can return only one value per*** call at the most.

The return statement can take one of the following forms :

**return;** or  
**return(expression);**

when a return is encountered the control is immediately passed back to the calling function .

**Example :**

```
int mul(int x, int y)
{
    int p;
    p = x*y;
    return (p);
}
```

**2. Function Call :** A function can be called by simply using the function name followed by a list of actual parameters (or arguments) if any enclosed in parentheses.

Example :

```
main( )
{
    int y;
    y = mul(10,5);
    printf("%d\n",y);
}
```

**3. Function Declaration:** Like variables all functions in a C program must be declared before they are invoked. A function declaration (also known as ***function prototype*** ) consists of four parts :

- Function type (return data type)
- Function name
- Parameter List
- Terminating Semicolon

The format for function prototype declaration is:

```
function_type function_name(parameter list);
```

**Example :**

```
int mul(int m, int n);
```

Or

```
int mul(int, int);
```

A function prototype declaration may be placed in two places in a program.

1. Above all the functions (including main) as a global prototype.
2. Inside a function definition as a local prototype.

The place of declaration of a function defines a region in a program in which the function may be used by other functions. This region is known as the *scope of* the function.

**Note:** The parameters also known as arguments are used in three places :

- 1.In declaration (prototype)
2. In function Call
3. In function definition

The parameters used in prototypes and function definitions are called *formal parameters* and those used in function calls are called *actual parameters*, because they are the actual values used for computation.

**3.3.3 Functions and Program Structure:** C is made up of functions. The structure of C program in terms of functions can be rewritten as follows:

```
[Comments /Documentation Section]
[Pre-processor Directives/Link Section]
[Global Declaration Section]
[Function Prototype Declaration Section
type Function1(type a1,type a2,-----type an);
type Function2(type a1,type a2,-----type an);
-----
type Functionn(type a1,type a2,-----type an);
]
main()
{
    Declaration Section
    Executable part
        Function1(a1,a2,--an);
        Function2(a1,a2,--an);
        -----
        Functionn(a1,a2,--an);
}
Subprogram section
[ type Function1( type a1,---type an)
{
    Local Declaration section ;
    Execution Section ;
    Return statement;
}
type Function2( type a1,---type an)
{
    Local Declaration section ;
    Execution Section ;
    Return statement;
}
-----
type Functionn( type a1,---type an)
{
    Local Declaration section ;
    Execution Section ;
    Return statement;
}
]
```

**The Documentation /Comments Section** consists of a set of comment lines giving the short description /purpose of the program or any statement of the program and other details.

**The Link/Pre-processor Section** provides instructions to the compiler to link functions from the system library.

**The Definition Section** defines all symbolic constants.

**The Global Declaration Section:** In this section variables are declared outside the main function and these variables can be accessed by all functions.

**The Function Declaration Section:** Here the prototype declarations of sub functions are declared.

**The main () function:** Every C program must have one main function section. This section contains two parts, declaration and executable part. The execution part contains the execution statements and the function call.

**Declaration Part** declares all the variables used in the executable part. There should be at least one statement in the **executable part** which contains instructions to perform certain task. The declaration and executable part must appear between the opening and closing braces. All statements in the declaration part should end with the semicolon.

The **Subprogram Section** contains the definitions of all the user defined functions that are called in the main function.

### **Example:**

```
/*Program to find the sum of two integers*/
#include <stdio.h> /*link section*/
#include <conio.h> /*link section*/
int addition( int , int ); /*Prototype declaration */

void main()
{
    int n1,n2,sum ;           /*Declaration part*/
    clrscr();                  /*executable part starts here*/
    printf("Enter the values of n1 and n2\n");
    scanf("%d %d",&n1,&n2);
    sum = addition(n1,n2); /* Function Call */
    /* n1 and n2 are called actual paramters*/
    printf("Sum =%d \n ",sum);
    getch();
}
int addition( int x,int y) /* Function definition*/
{
    /* x and y are called formal paramters*/
    int s ; /* Declaration of Local variable */
    s = x+y;
    return s; /* Return statement to return the sum s*/
}
```

### **3.3.4 Location of Functions:**

Location of functions suggest the placement of function subprograms with respect to main () program. There are three alternative to specify the location for functions:

**Alternative 1:** In this type the function definition are placed *before the main() program body*. In this case defining or declaring function prototype can be omitted.

### **Example :**

```
/*Program to find the sum of two integers*/
#include <stdio.h> /*link section*/
#include <conio.h> /*link section*/

int addition( int  x, int  y) /* Function definition*/
{
    int  s ;
    s = x+y;
    return s; /* Return statement to return the sum s*/
}

void main()
{
    int  n1,n2,sum ;           /*declaration part*/
    clrscr();
    printf("Enter the values of n1 and n2\n"); /*executable
part starts here*/
    scanf("%d %d",&n1,&n2);
    sum = addition(n1,n2); /* Function Call */
    printf("Sum=%d \n ",sum);
    getch();
}
```

**Alternative 2:** In this type the body of function are placed after the main( ) program. Here one has to place the function prototype statement well before main( ) program appearance in order to assists C compiler for type checking and function call location with respect to function definition.

### **Example:**

```
/*Program to find the sum of two integers*/
#include <stdio.h> /*link section*/
#include <conio.h> /*link section*/
int addition( int , int ); /*Function Prototype declaration section*/

void  main()
{
    int  n1,n2,sum ;   /*declaration part*/
    clrscr();
```

```

printf("Enter the values of n1 and n2\n"); /*executable part starts
here*/
scanf("%d %d",&n1,&n2);
sum = addition(n1,n2);/* Function Call */
printf("Sum=%d \n ",sum);
getch();
}
int addition( int x,int y) /* Function definition*/
{
    int s ;
    s = x + y;
    return s; /* Return statement to return the sum s*/
}

```

**Alternative 3:** Here the function body is placed in one file and the main() program with function prototypes in another file.

**Example:** In the file say ***add.h*** the following function will be defined

```

int addition( int x,int y) /* Function definition*/
{
    int s ;
    s = x+y;
    return s; /* Return statement to return the sum s*/
}

```

In the file say ***addition.c*** the following code has to be included

```

#include <stdio.h> /*link section*/
#include <conio.h> /*link section*/
#include<add.h> /* the header file which includes the function
definition */
void main()
{
    int n1,n2,sum ; /*declaration part*/
    clrscr();
    printf("Enter the values of n1 and n2\n"); /*executable part
starts here*/
    scanf("%d %d",&n1,&n2);
    sum = addition(n1,n2); /* Function Call */
    printf("Sum=%d \n ",sum);
    getch();
}

```

### 3.3.5 Categories of user defined functions

A user defined function may or may not contain arguments. Similarly it may or may not return a value to the calling function. Based on the parameters (arguments) and return value , the user defined functions are categorized into the following types :

1. Function without parameters and no return values ( Void and parameter less Functions).
2. Function with no parameter and return values.
3. Functions with parameters and no return values
4. Functions with parameters and return values.

1. **Void and Parameter less Functions:** This type of functions will not have any parameter and they will not return any value to the calling functions. i.e. there is no data transfer between the calling function and called function. So calling function cannot send values and hence called function cannot receive the data.

#### Programming Example :

```
#include<stdio.h>
#include<conio.h>
void mul();

void main()
{
    clrscr();
    mul();
    getch();
}

void mul() → function with no parameters
{
    int a,b,c;
    printf("\n Enter the values of a and b: ");
    scanf("%d %d",&a,&b);
    c = a*b ;
    printf("\n Product = %d \n",c);
    return ; → returning nothing
}
```

### Output

```
Enter the values of a and b: 2 3  
Product = 6
```

**2. Functions with parameters and no return values:** Here there is a data transfer from the calling function to called function. But there is no data transfer from called function to the calling function. The called function will have parameters but no return type.

#### Programming Example :

```
#include<stdio.h>  
void mul (int a, int b);  
void main()  
{  
    int m,n;  
    clrscr();  
    printf("\n Enter m and n:");  
    scanf("%d %d",&m,&n);  
    mul(m,n);  
    getch();  
}  
  
void mul(int a, int b) → function with parameters a,b  
{  
    int c ;  
    c= a*b;  
    printf("\n Product = %d ",c);  
    return; → returning nothing  
}
```

### Output

```
Enter m and n : 2 3  
Product = 6
```

**3. Functions with no parameters and return value:** In this category there is no data transfer from the calling function to the called function. But there is data transfer from called function to the calling function. When the function returns a value the calling function receives one value from the called function.

### Programming Example :

```
#include<stdio.h>
#include<conio.h>
int mul();

void main()
{
    int c;
    clrscr();
    c = mul();
    printf("\n Product = %d\n",c);
    return;
}

int mul()      →function with no parameters
{
    int a,b,c ;
    printf("\nEnter the values of a and b\n");
    scanf("%d %d", &a,&b);
    c = a*b;
    return c;   → returning the value of c
}
```

### Output

Enter the values of a and b

2 3

Product = 6

**4. Functions with parameters and return values:** In this category there is data transfer between the calling function and

called function. When parameters are passed, the called function can receive values from the calling function. When the function returns a value the calling function can receive a value from the called function.

**Example :**

```
#include<stdio.h>
int mul(int a, int b);
void main()
{
    int m,n,c;
    clrscr();
    printf("\n Enter the values of m and n\n");
    scanf("%d %d",&m,&n);

    c = mul(m,n);
    printf("\n Product = %d ", c);
    getch();
}

int mul( int a, int b) → function with parameters a,b
{
    int c;

    c = a*b;
    return c; → function returning the value of c
}
```

**Output**

Enter the values of m and n: 2 3

Product = 6

**3.3.6 Argument Passing :** There are two ways of passing parameters to the function

- a. Pass by Value ( Call by value )
- b. Pass by Reference ( Call by Reference )

- a. **Pass by Value (Call by value):** Here the values of actual parameters are copied into formal parameters i.e. *formal parameters contain only the copy of actual parameters*. So even if the value of the formal parameters changes in the called function the values of the actual parameters are not changed.

### Programming Example:

```
#include<stdio.h>
#include<conio.h>
void exchange(int m,int n);
void main()
{
    int a,b;
    clrscr();
    a=10, b=20;
    printf("\n The values of a and b before calling exchange()\n");
    printf("\n a=%d b= %d \n ",a,b);
    exchange(a,b); → Actual Parameters a,b
    printf("\n The values of a and b after calling
exchange()\n");
    printf("\n a =%d and b = %d \n ",a,b);
    getch();
}

void exchange (int m , int n) → Formal Parameters m,n
{
    int temp;
    temp = m;
    m = n;
    n = temp;
}
```

### Output

```
The values of a and b before calling exchange()
a=10  b = 20
The values of a and b after calling exchange()
a= 10 b= 20
```

- b. **Pass by Reference (Call by Reference)** : In pass by reference / address when a function is called the address of actual parameters are sent. In the called function the *formal parameters* should be declared as pointers with the same type as the *actual parameters*. Using these addresses the values of the actual parameters can be changed. This way of changing the actual parameters is called pass by addresses.

### Programming Example :

```
#include<stdio.h>
#include<conio.h>
void exchange (int *m, int *n);
void main()
{
    int a,b;
    clrscr();
    a=10, b=20;
    printf("The values of a and b before calling exchange()\n");
    printf(" a=%d and b = %d\n",a,b);
    exchange(&a,&b); → Actual Parameters a,b
    printf("The values of a and b after calling exchange()\n");
    printf(" a= %d and b= %d\n",a,b);
    getch();
}
void exchange( int *m, int *n) → Formal Parameters m,n
{
    int temp;
    temp = *m;
    *m = *n ;
    *n = temp;
}
```

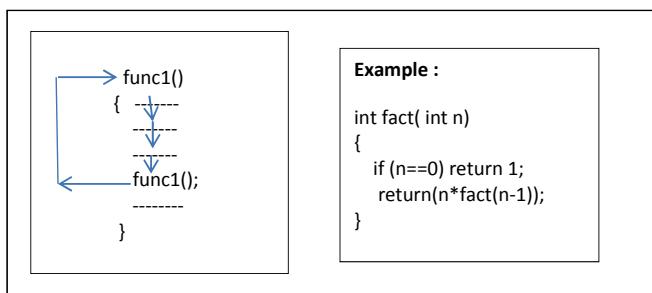
### Output

```
The values of a and b before calling exchange()
a = 10 b = 20
The values of a and b after calling exchange()
a = 20 b = 10
```

**3.3.7 Recursion:** The process of a function calling itself is called recursive. The recursion is a method of solving the problem where the solution to a problem depends on solutions to smaller instances of the same problem. Thus a ***recursive function is a function that calls itself during execution.*** The recursion refers to the process where a function calls itself either directly or indirectly. There are two types of recursion as given below:

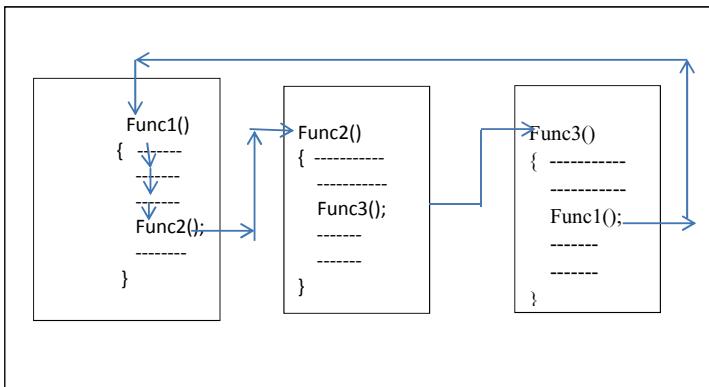
1. Direct Recursion
2. Indirect Recursion

**Direct Recursion :** It refers to a process where a function calls itself directly as illustrated in the figure below :



As illustrated in the figure above `func1()` calls `func1()` . This type of calling itself is called direct recursion and the recursion will take place until the given condition is satisfied. In the example given above the `fact(n)` calls itself and the called function returns `n*fact(n-1)` until the value of `n` becomes equal to `0`.

**Indirect Recursion:** It refers to a process where a function calls another function and that function calls back the original function. The process of indirect recursion takes place as illustrated in the figure below:



Here the Func1() calls Func2() and Func2() calls Func3() and Func3() calls Func1().

### Example : C Program to find the factorial of a given number using recursion .

```

#include <stdio.h>
#include<conio.h>
int factorial(int);
int main()
{ int num;
  int result;
  clrscr();
  printf("Enter a number to find it's Factorial: ");
  scanf("%d", &num);
  if (num < 0)
  { printf("Factorial of negative number is not possible\n");
  }
  else
  {
    result = factorial(num);
    printf("The Factorial of %d is %d.\n", num, result);
  }
  getch();
  return 0;
}

int factorial(int num)
{
  if (num == 0 || num == 1)

```

```
{  
    return 1;  
}  
else  
{  
    return(num * factorial(num - 1));  
}  
}
```

### Output:

Enter a number to find it's Factorial: 6

The Factorial of 6 is 720.

### 3.3.8 Passing Arrays to Functions (Using Arrays With Functions)

1. **Passing One Dimensional Arrays :** Like the values of simple variables it is also possible to pass the values of an array to a function .To pass a one dimensional array to a function it is sufficient to *list the name of the array* , without any subscripts , and *the size of the array* as arguments .

For example, the function call **largest (a,n)** will pass the whole array **a** to the called function. The called function expecting this call must be appropriately defined. The function header of the function **largest()** must look like as follows:

```
int largest(int array[ ], int n);
```

Consider the problem of finding the largest value in an array of elements. The program is as follows:

```
#include<stdio.h>  
#include<conio.h>  
int largest( int a[ ] , int n);  
main()  
{  
    int i,n,a[10],lg ;
```

```

clrscr();
printf("\n Enter the value of n \n");
scanf("%d", &n);
printf("\n Enter %d elements \n", n);
for(i=0;i<n;i++)
scanf("%d",&a[i]);
lg = largest(a,n);
printf("\n The largest element is %d\n",large);
getch();
}

int largest(int a[ ],int n)
{
    int i;
    int max ;
    max = a[0] ;
    for(i =1;i<n ;i++)
    {
        if(max<a[i])
            max = a[i];
    }
    return max;
}

```

**Output:**

```

Enter the value of n
5
Enter 5 elements
10
5
20
60
70
The largest element is 70

```

**Three Rules to pass an array to a function:**

1. The function must be called by passing only the name of the array.
2. In the function definition the formal parameter must be an array type, the size of the array does not need to be specified.
3. The function prototype must show that the argument in an array .

**2. Passing Two Dimensional Arrays:** Like simple array we can also pass two dimensional arrays to functions. The approach is similar to the one we did with a one dimensional arrays. Following rules has to be followed while writing a program.

**Rules :**

1. The function must be called by passing only the array name.
2. In the function definition we must indicate that the array has two dimensions by including two sets of brackets.
3. The size of the second dimension must be specified
4. The prototype declaration should be similar to the function header.

**Example : Program to find the average of a given matrix :**

```
#include<stdio.h>
#include<conio.h>
float average(int a[][5],int m,int n);

main()
{
    int a[5][5],m,n,i,j;
    float avg;

    clrscr();

    printf("\nEnter the number of rows (m) and columns (n) \n");
    scanf("%d %d ", &m,&n);
    printf("\nEnter the matrix \n");
    for(i=0;i<m;i++)
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);

    avg = average(a,m,n);

    printf("\nThe average of the given matrix is %f\n", avg);
    getch();
}
```

```
float average( int  a[][5],int  m,int  n)
{
    int  i,j ;
    float sum =0.0;

    for(i=0;i<m;i++)
        for(j=0;j<n;j++)
            sum += a[i][j];
    return (sum/(m*n));
}
```

**Output :**

Enter the number of rows(m) and columns(n)

**3 3**

Enter the matrix

|          |          |          |
|----------|----------|----------|
| <b>1</b> | <b>2</b> | <b>3</b> |
| <b>2</b> | <b>4</b> | <b>5</b> |
| <b>3</b> | <b>2</b> | <b>2</b> |

The average of the given matrix is    **2.666667**

## **Additional Concepts :**

**Global and Local Variables :** Global variables are the variables which are defined before all functions in global area of the program. Local variables are the variables which are defined within a function. These variables are also called automatic variables.

### **Difference between Global and Local Variables**

| <b>Global Variables</b>                                                                                                            | <b>Local Variables</b>                                                                                                                           |
|------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| These variables are declared before all the functions in global variable declaration section                                       | These variables are declared inside the function.                                                                                                |
| The memory is allocated only once at beginning of program execution and are deallocated once the execution of the program is over. | The memory is allocated when the control enters into the function and memory will be deallocated whenever the control goes out of the function . |
| The Global variables are initialized to zero.                                                                                      | The initial value of Local variables is indeterminate and takes garbage value.                                                                   |
| The variables can be accessed by any function and are alive and active throughout the program.                                     | These variables cannot be accessed by any function and are alive and active within the function.                                                 |
| The scope of the variable is entire program.                                                                                       | The scope of these variables is limited only to the function in which they are declared and cannot be accessed outside the function.             |

**Example : Consider a program given below :**

```
#include<stdio.h>
#include<conio.h>
int x =10; /* Declared Globally */
main()
{ int a = 20; /* a is local to main*/
    clrscr();
    printf("\n a = %d \n",a); /* prints a = 20 */
    printf("\n x = %d\n",x); /* prints x = 10 */
    fun1();
    fun2();
    getch();
}
void fun1()
{
    int b = 30; /* b is local to fun1() */
    printf("\n x = %d b= %d\n",x,c); /* prints x=10 b= 30 */
}
void fun2()
{
    int c = 40; /* c is local to fun2() */
    printf("\n x =%d c =%d \n",x,c); /*prints x =10 c=40 */
}
```

Here x is global variable and a,b and c are called local variables. x is visible to all the functions say , fun1(),fun2() and main function. a,b and c variables are visible only to main ,fun1() and fun2() respectively.

**Actual and Formal Parameters :** Argument or parameter is the expression (containing variable or constant) used to establish a data communication between called function and calling function to perform a particular task. We come across two types of parameters in functions.

**a. Actual Parameters :** *Actual parameters are the variables or constants that are passed in the function call .* These parameters are defined and passed inside the calling function to call a function. The actual parameters possess the actual input passed.

**b. Formal Parameters :** *Formal parameters are the variables that are declared in the function header and function prototype of the called function.* These are a copy of the actual parameters and its scope is local to the function in which they are used.

**The formal parameters names and actual parameters names can be the same or different.**

**Note :** The terms parameter and argument are sometimes used interchangeably and the context is used to distinguish the meaning. The term parameter (sometimes called formal parameter) is often used to refer to the variable as found in the function definition , while argument (sometimes called actual parameter) refers to the actual input passed.

**Example :**

```
#include<stdio.h>
#include<conio.h>
int prod(int a,int b); /* Function Prototype*/
int main()
{
    int n1,n2,p;
    clrscr();
    printf("\n Enter the value of n1 and n2");
    scanf("%d %d",&n1,&n2);
    p = prod(n1,n2); /* Function Call */
    printf("\n The product of two numbers = %d",p);
    getch();
}

/* Function Definition */
int prod(int a, int b) /* Function header */
{
    int r;
    r= a*b;
    return(r);
}
```

In the above example the parameters **n1** and **n2** which are used in the main (calling function) during *function call* are called **actual parameters**. The parameters **a** and **b** which are used in function prototype , function header and function definition of the *called function*, **prod( )** are called **formal parameters**.

**Storage Class Specifiers:** *Storage class specifiers in C language instructs the compiler where to store a variable, how to store the variable, what is the initial value of the variable and life time of the variable.*

**Syntax:** `storageSpecifier data_type variable_name ;`

**Types of Storage Class Specifiers:** There are 4 storage class specifiers available in C language. They are : **1. auto 2. extern 3. static and register**

| Storage class Specifier | Storage place   | Default value | Scope  | Life                                                                |
|-------------------------|-----------------|---------------|--------|---------------------------------------------------------------------|
| auto                    | CPU Memory      | Garbage value | Local  | Within the function only.                                           |
| extern                  | CPU Memory      | Zero          | Global | Till the end of the main program.                                   |
| static                  | CPU Memory      | Zero          | Local  | Retains the value of the variable between different function calls. |
| register                | Register Memory | Garbage value | Local  | Within the function                                                 |

**Note:** *For faster access of a variable, it is better to go for register specifiers rather than auto specifies.*

- Because, register variables are stored in register memory whereas auto variables are stored in main CPU memory.
- Only few variables can be stored in register memory. So, we can use variables as register that are used very often in a C program.

1. **Example program for auto variable in C:** *The scope of the auto variable is within the function only. It is equivalent to local variable. All local variables are auto variables by default.*

```

#include<stdio.h>
void fun(void);

int main()
{
    fun();
    fun();
    fun();
    fun();
    return 0;
}
void fun(void)
{
    auto int i = 0 ;
    printf( "%d ", i );
    i++;
}

```

**Output:** 0 0 0 0

2. Example program for static variable in C:*Static variables retain the value of the variable between different function calls.*

```

#include<stdio.h>
void fun(void);
int main()
{
    fun();
    fun();
    fun();
    fun();
    return 0;
}
void fun(void)
{
    static int i = 0 ;
    printf( "%d ", i );
    i++;
}

```

**Output:**

0 1 2 3

- 3. Example program for extern variable in C:** *The scope of this extern variable is throughout the main program. It is equivalent to global variable.*

Definition for extern variable might be anywhere in the C program.

```
#include<stdio.h>
int x = 5 ;
int main()
{
extern int y;
printf("The value of x is %d \n",x);
printf("The value of y is %d",y);
return 0;
}
int y= 10;
```

**Output:**

```
The value of x is 5
The value of y is 10
```

- 4. Example program for register variable in C:** *Register variables are also local variables, but stored in register memory. Whereas, auto variables are stored in main CPU memory.*

Register variables will be accessed very faster than the normal variables since they are stored in register memory rather than main memory.

But, only limited variables can be used as register since register size is very low. (16 bits, 32 bits or 64 bits)

```
#include <stdio.h>
int main()
{
register int i;
int a[5];
a[0] = 1;
a[1] = 2;
a[2] = 3;
a[3] = 4;
```

```
a[4] = 5;  
for (i=0;i<5;i++)  
{  
    printf("value of arr[%d] is %d \n", i, arr[i]);  
}  
return 0;  
}
```

**Output:**

```
value of a[0] is 1  
value of a[1] is 2  
value of a[2] is 3  
value of a[3] is 4  
value of a[4] is 5
```

## C Standard Library Functions – Header File

C Standard library functions are inbuilt functions in C programming language. Function prototype and data definitions of these functions are written in their respective header files. For example: If you want to use printf() function, the header file <stdio.h> should be included. In the C Programming Language, the Standard Library Functions are divided into several header files. Below is the list of header files supported in C language:

| Header Files | Function                         |
|--------------|----------------------------------|
| <assert.h>   | Diagnostics Functions            |
| <ctype.h>    | Character Handling Functions     |
| <locale.h>   | Localization Functions           |
| <math.h>     | Mathematics Functions            |
| <setjmp.h>   | Nonlocal Jump Functions          |
| <signal.h>   | Signal Handling Functions        |
| <stdarg.h>   | Variable Argument List Functions |
| <stdio.h>    | Input/Output Functions           |
| <stdlib.h>   | General Utility Functions        |
| <string.h>   | String Functions                 |
| <time.h>     | Date and Time Functions          |

**1. assert.h :** The following is a list of functions found within the <assert.h> header file:

|         |                                                         |
|---------|---------------------------------------------------------|
| isalnum | Test for Alphanumeric                                   |
| isalpha | Test for Alphabetic                                     |
| iscntrl | Test for Control Character                              |
| isdigit | Test for Digit                                          |
| isgraph | Test for Graphical Character (does not include a space) |
| islower | Test for Lowercase Letter                               |

|          |                                                    |
|----------|----------------------------------------------------|
| isprint  | Test for Printing Character (does include a space) |
| ispunct  | Test for Punctuation Character                     |
| isspace  | Test for White-Space Character                     |
| isupper  | Test for Uppercase Letter                          |
| isxdigit | Test for Hexadecimal Digit                         |

### **assert () - Assert Truth of Expression (macro)**

**2. ctype.h :** The following is a list of functions found within the <ctype.h> header file:

#### **Character Case-Mapping functions**

|         |                      |
|---------|----------------------|
| tolower | Convert to Lowercase |
| toupper | Convert to Uppercase |

**3. locale.h :** The following is a list of functions found within the <locale.h> header file:

|            |                        |
|------------|------------------------|
| localeconv | Get Locale Conventions |
| setlocale  | Set Locale             |

**4.math.h :** The following is a list of functions found within the <math.h> header file:

#### **Absolute Value functions**

|      |                                         |
|------|-----------------------------------------|
| fabs | Absolute Value of Floating-Point Number |
|------|-----------------------------------------|

#### **Nearest Integer, Absolute Value, and Remainder functions**

|       |                  |
|-------|------------------|
| ceil  | Ceiling          |
| floor | Floor            |
| fmod  | Floating Modulus |

### **Exponential and Logarithmic functions**

|       |                                         |
|-------|-----------------------------------------|
| exp   | Exponential                             |
| frexp | Split into Fraction and Exponent        |
| ldexp | Combine Fraction and Exponent           |
| log   | Natural Logarithm                       |
| log10 | Common Logarithm                        |
| modf  | Split into Integer and Fractional Parts |

### **Power functions**

|      |             |
|------|-------------|
| pow  | Power       |
| sqrt | Square Root |

### **Trigonometric functions**

|       |                         |
|-------|-------------------------|
| acos  | Arc Cosine              |
| asin  | Arc Sine                |
| atan  | Arc Tangent             |
| atan2 | Arc Tangent of Quotient |
| cos   | Cosine                  |
| sin   | Sine                    |

### **Hyperbolic functions**

|      |                    |
|------|--------------------|
| cosh | Hyperbolic Cosine  |
| sinh | Hyperbolic Sine    |
| tanh | Hyperbolic Tangent |

5. **setjmp.h** : The following is a list of functions found within the <setjmp.h> header file:

### **Nonlocal Jump functions**

|         |                           |
|---------|---------------------------|
| longjmp | Nonlocal Jump             |
| setjmp  | Prepare for Nonlocal Jump |

**6. `signal.h`** : The following is a list of functions found within the `<signal.h>` header file:

#### **Signal Handling functions**

|        |                        |
|--------|------------------------|
| raise  | Raise Signal           |
| signal | Install Signal Handler |

**7. `stdarg.h`** : The following is a list of functions found within the `<stdarg.h>` header file:

#### **Variable Argument List functions**

|          |                                            |
|----------|--------------------------------------------|
| va_arg   | Fetch Argument from Variable Argument List |
| va_end   | End Processing of Variable Argument List   |
| va_start | Start Processing of Variable Argument List |

**8. `stdio.h`** : The following is a list of functions found within the `<stdio.h>` header file:

#### **Formatted Input/Output functions**

|          |                                                     |
|----------|-----------------------------------------------------|
| fprintf  | Formatted File Write                                |
| fscanf   | Formatted File Read                                 |
| printf   | Formatted Write                                     |
| scanf    | Formatted Read                                      |
| sprintf  | Formatted String Write                              |
| sscanf   | Formatted String Read                               |
| vfprintf | Formatted File Write Using Variable Argument List   |
| vprintf  | Formatted Write Using Variable Argument List        |
| vsprintf | Formatted String Write Using Variable Argument List |

#### **File Operation functions**

|        |                   |
|--------|-------------------|
| fclose | Close File        |
| fflush | Flush File Buffer |
| fopen  | Open File         |

|         |                              |
|---------|------------------------------|
| freopen | Reopen File                  |
| remove  | Remove File                  |
| rename  | Rename File                  |
| setbuf  | Set Buffer (obsolete)        |
| setvbuf | Set Buffer                   |
| tmpfile | Create Temporary File        |
| tmpnam  | Generate Temporary File Name |

### Character Input/Output functions

|         |                           |
|---------|---------------------------|
| fgetc   | Read Character from File  |
| fgets   | Read String from File     |
| fputc   | Write Character to File   |
| fputs   | Write String to File      |
| getc    | Read Characters from File |
| getchar | Read Character            |
| gets    | Read String               |
| putc    | Write Character to File   |
| putchar | Write Character           |
| puts    | Write String              |
| ungetc  | Unread Character          |

### Block Input/Output functions

|        |                      |
|--------|----------------------|
| fread  | Read Block from File |
| fwrite | Write Block to File  |

### File Positioning functions

|         |                   |
|---------|-------------------|
| fgetpos | Get File Position |
| fseek   | File Seek         |
| fsetpos | Set File Position |

|        |                         |
|--------|-------------------------|
| ftell  | Determine File Position |
| rewind | Rewind File             |

### Error Handling functions

|          |                      |
|----------|----------------------|
| clearerr | Clear Stream Error   |
| feof     | Test for End-of-File |
| ferror   | Test for File Error  |
| perror   | Print Error Message  |

9. **stdlib.h** : The following is a list of functions found within the `<stdlib.h>` header file:

### Communication with the Environment functions

|        |                                                |
|--------|------------------------------------------------|
| abort  | Abort Program                                  |
| atexit | Register Function to be Called at Program Exit |
| exit   | Exit from Program                              |
| getenv | Get Environment String                         |
| system | Perform Operating System Command               |

### Integer Arithmetic functions

|      |                                |
|------|--------------------------------|
| abs  | Absolute Value of Integer      |
| div  | Integer Division               |
| labs | Absolute Value of Long Integer |
| ldiv | Long Integer Division          |

### Pseudo-Random Sequence Generation functions

|       |                                     |
|-------|-------------------------------------|
| rand  | Generate Pseudo-Random Number       |
| srand | Seed Pseudo-Random Number Generator |

### **String Conversion functions**

|         |                                         |
|---------|-----------------------------------------|
| atof    | Convert String to Floating-Point        |
| atoi    | Convert String to Integer               |
| atol    | Convert String to Long Integer          |
| strtod  | Convert String to Double                |
| strtol  | Convert String to Long Integer          |
| strtoll | Convert String to Long Long             |
| strtoul | Convert String to Unsigned Long Integer |

### **Searching and Sorting functions**

|         |               |
|---------|---------------|
| bsearch | Binary Search |
| qsort   | Sort Array    |

### **Dynamically Allocated Array functions**

|         |                                 |
|---------|---------------------------------|
| calloc  | Allocate and Clear Memory Block |
| malloc  | Allocate Memory Block           |
| realloc | Resize Memory Block             |

### **Deallocating Storage functions**

|      |                   |
|------|-------------------|
| free | Free Memory Block |
|------|-------------------|

### **Multibyte Character functions**

|        |                                               |
|--------|-----------------------------------------------|
| mblen  | Compute Length of Multibyte Character         |
| mbtowc | Convert Multibyte Character to Wide Character |
| wctomb | Convert Wide Character to Multibyte Character |

### **Multibyte String functions**

|          |                                                   |
|----------|---------------------------------------------------|
| mbstowcs | Convert Multibyte String to Wide Character String |
| wcstombs | Convert Wide Character String to Multibyte String |

10. **string.h** : The following is a list of functions found within the **<string.h>** header file:

#### Comparison functions

|         |                                                         |
|---------|---------------------------------------------------------|
| memcmp  | Compare Memory Blocks                                   |
| strcmp  | String Compare                                          |
| strcoll | String Compare Using Locale-Specific Collating Sequence |
| strncmp | Bounded String Compare                                  |
| strxfrm | Transform Locale-Specific String                        |

#### Concatenation functions

|         |                              |
|---------|------------------------------|
| strcat  | String Concatenation         |
| strncat | Bounded String Concatenation |

#### Copying functions

|         |                     |
|---------|---------------------|
| memcpy  | Copy Memory Block   |
| memmove | Copy Memory Block   |
| strcpy  | String Copy         |
| strncpy | Bounded String Copy |

#### Search functions

|         |                                                        |
|---------|--------------------------------------------------------|
| memchr  | Search Memory Block for Character                      |
| strchr  | Search String for Character                            |
| strcspn | Search String for Intial Span of Characters Not in Set |
| strupr  | Search String for One of a Set of Characters           |
| strrchr | Search String in Reverse for Character                 |
| strspn  | Search String for Initial Span of Characters in Set    |
| strstr  | Search String for Substring                            |
| strtok  | Search String for Token                                |

### **Miscellaneous functions**

|          |                                |
|----------|--------------------------------|
| memset   | Initialize Memory Block        |
| strerror | Convert Error Number to String |
| strlen   | String Length                  |

**11. time.h :** The following is a list of functions found within the <time.h> header file:

### **Time Conversion functions**

|           |                                         |
|-----------|-----------------------------------------|
| asctime   | Convert Date and Time to ASCII          |
| ctime     | Convert Date and Time to String         |
| gmtime    | Convert to Greenwich Mean Time          |
| localtime | Convert to Local Time                   |
| mktime    | Convert to Calendar Time                |
| strftime  | Write Formatted Date and Time to String |

### **Time Manipulation functions**

|          |                 |
|----------|-----------------|
| clock    | Processor Clock |
| difftime | Time Difference |
| time     | Current Time    |

### **3.4: Programming Examples**

#### **3.4.1 Programming Examples based on Single Dimensional Arrays:**

- 1. Write a C program to read and write a single dimensional array of integer of size where n<=20.**

```
#include<stdio.h>
#include<conio.h>
main()
{
    int a[20],i,n;
    clrscr();

    printf("\n Enter the value of n\n");
    scanf ("%d",&n);

    printf("\n Enter the %d elements one after the other \n",n);
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);

    printf("\n The entered %d elements are :\n",n);
    for(i=0;i<n;i++)
        printf(" %d ",a[i]);

    getch();
}
```

#### **Output**

```
Enter the value of n
5
Enter the 5 elements one after the other
10 20 12 33 45
The entered 5 elements are :
10 20 12 33 45
```

- 2. Write a C program to find the sum of N integers using single dimensional array.**

```

#include<stdio.h>
#include<conio.h>
main()
{
    int n,a[10],i,sum;
    clrscr();
    printf("\n Enter the number of integer \n");
    scanf("%d",&n);
    printf("\n Enter the %d integers \n",n);
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    sum=0;
    for(i=0;i<=n-1;i++)
        sum=sum+ a[i];
    printf(" Sum =%d",sum);
    getch();
}

```

**Output**

Enter the number of integer  
5  
Enter the 5 integers  
1 2 3 4 5  
Sum = 15

3. Write a C program to evaluate the polynomial  $f(x) = a_4x^4+a_3x^3+a_2x^2+a_1x+a_0$  for a given value of x and its coefficients using Horner method.

**Logic:**

$$4x^4+3x^3+2x^2+4x+5=f(x)$$

$$x(4x^3+3x^2+2x+4)+5=f(x)$$

$$x(x(4x^2+3x+2)+4)+5=f(x)$$

$$x(x(x(4x+3)+2)+4)+5=f(x)$$

$$x(x(x(x(4)+3)+2)+4)+5=f(x)$$

```

#include<stdio.h>
#include<conio.h>
main()
{
    int    n,i;
    float a[10],x,sum;
    clrscr( );
    printf("n Enter the value of n\n");
    scanf("%d",&n);
    printf("n Enter n+1 values\n");
    for(i=0;i<=n;i++)
    {
        scanf("%f",&a[i]);
    }
    printf("n Enter the value of x\n");
    scanf("%f",&x);
    sum = a[n]*x;
    for(i=n-1;i>=1;i--)
        sum = (sum+a[i])*x;
    sum = sum +a[0];
    printf("n The value of polynomial = %f\n",sum);
    getch();
}

```

### Output

```

Enter the value of n
4
Enter the n+1 values
1 2 3 4 5
Enter the value of x
1
The value of the polynomial = 15.000000

```

#### 4. Program to find the average of the marks of the student using arrays.

```

#include <stdio.h>
#include<conio.h>
#define SIZE 50
main()
{
    int count,n,sum;

```

```

int mark[SIZE];
float avgmark;
clrscr();

printf("\nEnter the number of subjects\n");
scanf("%d",&n);
if((n<=0) ||(n > size))
{
    printf("\n Invalid number of marks \n");
    exit(0);
}
for(count =0; count <n ;count++)
{
    printf("\nEnter marks of subject no %d\n",count+1);
    scanf("%d",&mark[count]);
}
sum = 0;
for(count =0; count<n;count++)
    sum+=mark[count];

avgmark = (float)sum/n;
printf("\nThe average is %f\n",avgmark);

getch();
}

```

**Output :**

```

Enter the number of subjects
5
Enter marks of subject no 1
98
Enter the marks of subject no 2
75
Enter the marks of subject no 3
65
Enter the marks of subject no 4
77
Enter the marks of subject no 5
88

The average is 80.60000

```

- 5. Write a C program to input n real numbers and to find mean, variance and standard deviation using the formulae :**

- Mean =  $\frac{\sum_{i=0}^{n-1} xi}{n}$
- Variance =  $\frac{\sum_{i=0}^{n-1} (xi - \text{mean}) * (xi - \text{mean})}{n}$
- Standard Deviation =  $\sqrt(\text{variance})$

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
    int    n,i;
    float   x[10],sum1,sum2,mean;
    float   variance ,deviation;
    clrscr();
    printf("\n Enter the value of n\n");
    scanf("%d",&n);
    printf("\n Enter %d real numbers \n",n);
    for(i=0;i<=n-1;i++)
        scanf("%f",&x[i]);

    sum1= 0;

    for(i=0;i<= n-1;i++)
        sum1 = sum1+x[i];

    mean = sum1/n;
    sum2 = 0;
    for(i=0;i<=n-1;i++)
    {
        sum2= sum2 + (x[i] - mean)*(x[i]- mean);
    }
    variance = sum2/n ;
    deviation = sqrt(variance);
    printf(" Mean =%f \n", mean);
    printf("Variance = %f \n", variance);
    printf("\n Standard Deviation = %f\n",deviation);
    getch();
}
```

### Output:

```
Enter the value of n
5
Enter 5 real numbers
34
88
32
12
10
Mean = 35.200000
Variance = 794.560000
Standard Deviation = 28.190000
```

## 6. C Program to implement Linear Search

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,n,a[10],key;
    clrscr();
    printf("\n Enter the number of integers \n");
    scanf("%d",&n);
    printf("\n Enter the %d items \n",n);
    for(i=0;i<=n-1;i++)
        scanf("%d",&a[i]);
    printf("\n Enter the key to be searched \n");
    scanf("%d",&key);

    for(i=0;i<=n-1;i++)
    {
        if(key==a[i])
        {
            printf("\n Search Successful\n");
            exit(0);
        }
    }

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

### Output :

```
Enter the number of integers
5
```

```
Enter the 5 items
10 20 30 40 50
Enter the Key to be searched
20
Search successful
```

## 7. C Program to implement Binary Search

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,n,item,low,high,mid,a[20];
    clrscr();

    printf("\n Enter the value of n\n");
    scanf("%d",&n);

    printf("\n Enter the elements of array in ascending order\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf("Enter the item to be searched \n");
    scanf("%d",&item);
    low = 0;
    high = n-1;

    while(low<=high)
    {
        mid = (low+high)/2;
        if(item==a[mid])
        {
            printf(" Success ");
            exit(0);
        }
        if(item<a[mid])
            high = mid-1;
        else
            low = mid +1 ;
    }

    printf("\n Un Successful Search \n");
    getch();
}
```

### Output :

```
Enter the value of n  
5  
Enter the elements of array in ascending order  
10 20 30 40 50  
Enter the item to be searched  
20  
Search successful
```

### 8. C program that reads N integer numbers and arrange them in ascending order using Bubble sort.

```
#include<stdio.h>  
#include<conio.h>  
main()  
{  
    int n,i,j,temp,a[20];  
    clrscr();  
  
    printf("\n Enter the number of items \n");  
    scanf("%d",&n);  
    printf("\n Enter the %d items to sort \n",n);  
    for(i=0;i<n;i++)  
        scanf("%d",&a[i]);  
  
    for(j=1;j<n;j++)  
    { for(i=0;i<n-j;i++)  
        {  
            if(a[i]>=a[i+1])  
            {  
                temp = a[i] ;  
                a[i] = a[i+1];  
                a[i+1]= temp ;  
            }  
        }  
    }  
  
    printf("\n The sorted items are \n");  
    for(i=0;i<n;i++)  
        printf("%d\t",a[i]);  
    getch();  
}
```

### Output :

```
Enter the number of items
5
Enter the 5 items to sort
50 10 40 30 20
The sorted items area
10 20 30 40 50
```

**9. Write a Program to read N integers into an array A and to**

- i. Find the sum of odd numbers**
- ii. Find the sum of even numbers**
- iii. Find the average of all numbers**

Output the results computed with appropriate headings.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int i,a[100],osum,esum,N;
    float average;
    clrscr();

    printf("\n Enter the value of N:");
    scanf("%d",&N);

    printf("\n Enter %d integers \n",N);
    for(i=0;i<N;i++)
        scanf("%d",&a[i]);
    osum =0;
    esum =0;
    for(i=0;i<N;i++)
        if(a[i]%2==0) // even or odd number check
            esum = esum +a[i]; // even numbers gets added
        else
            osum = osum +a[i]; // odd numbers gets added

    average = (float)(esum+osum)/N;
    printf("\n Sum of even numbers = %d\n",esum);
    printf("\n Sum of odd numbers = %d\n", osum);
    printf("\n The average of all numbers = %f\n",average);
    getch();
}
```

### Output :

```
Enter the value of N: 5
Enter 5 integers
10      12      5      3      8
Sum of even numbers = 30
Sum of odd numbers = 8
The average of all numbers =7.600000
```

- 10. Given two dimensional arrays a and b which are sorted in ascending order, write a program to merge them into a single sorted array c, that contains items from a and b in ascending order.**

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

void main()
{
    int a[25],b[25],c[100],i,j,k,m,n;
    clrscr();

    printf("\n Enter the size of the array a:");
    scanf("%d",&n);
    printf("\n Enter the array a in ascending order:");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf("\n Enter the size of the array b:");
    scanf("%d",&m);
    printf("\n Enter the array b :");
    for(i=0;i<m;i++)
        scanf("%d",&b[i]);
    i = 0;
    j = 0;
    k = 0;
    while(i<n && j<m)
    {
        if(a[i]<b[j])
        {
            c[k] = a[i];
            k++;
            i++;
        }
    }
}
```

*/\*if a[i] < b[j] then a[i] should be inserted in array c and increment index of both array c and a so that it points to next element \*/*

```

        {
            c[k] = b[j];
            k++;
            j++;
        }
    }

while(i<n)
{
    c[k] = a[i];
    k++;
    i++;
}
printf("\n Merged Array \n");
for(i = 0;i<n + m; i++)
printf("%5d",c[i]);
getch();
}

```

### Output :

Enter the size of the array a: **5**  
Enter the array a in ascending order : **2      3      12     15    17**  
Enter the size of the array b : **5**  
Enter the array b : **4      5      6      7      8**  
Merged Array  
**2      3      4      5      6      7      8      12     15    17**

11. Write a C program to generate first n Fibonacci series using arrays.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int i,n,fib[20]={0,1,1};
    clrscr();
    printf("\n Enter the array size \n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    fib[i+3] = fib[i+2] + fib[i+1];
    printf("\n Fibonacci Series :\n");
    for(i=0;i<n;i++)
    printf("\n %d ", fib[i]);
    getch();
}

```

**Output :**

Enter the array size

**10**

Fibonacci Series:

**0    1    1    2    3    5    8    13    21    34**

**12. Write a C program to find the largest two elements in a given array .**

```
#include <stdio.h>
#include<conio.h>
#define MAX 4

void main()
{
    int array[MAX], i, largest1, largest2, temp;
    clrscr();
    printf("Enter %d integer numbers \n", MAX);
    for (i = 0; i < MAX; i++)
    {
        scanf("%d", &array[i]);
    }

    printf("Input integer are \n");
    for (i = 0; i < MAX; i++)
    {
        printf("%5d", array[i]);
    }
    printf("\n");

    /* assume first element of array is the first largest */
    largest1 = array[0];
    /* assume first element of array is the second largest */
    largest2 = array[1];
    if (largest1 < largest2)
    {
        temp = largest1;
        largest1 = largest2;
        largest2 = temp;
    }

    for (i = 2; i < 4;i++)
    {
```

```

if (array[i] >= largest1)
{
    largest2 = largest1;
    largest1 = array[i];
}
else if (array[i] > largest2)
{
    largest2 = array[i];
}
printf("n%d is the first largest \n", largest1);
printf("%d is the second largest \n", largest2);
printf("nAverage of %d and %d = %d \n", largest1, largest2,
(largest1 + largest2) / 2);
getch();
}

```

**Output:**

Enter 4 integer numbers

80

23

79

58

Input integer are

80 23 79 58

80 is the first largest

79 is the second largest

Average of 80 and 79 = 79

- 13. Write a C program to find the second largest and smallest in an array.**

```

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

void main ()
{
    int number[30];
    int i, j, a, n, counter, average;

```

```

clrscr();

printf("Enter the value of N\n");
scanf("%d", &n);
printf("Enter the numbers \n");
for (i = 0; i < n; ++i)
scanf("%d", &number[i]);
for (i = 0; i < n; ++i) // arranging the elements in descending order
{
    for (j = i + 1; j < n; ++j)
    {
        if (number[i] < number[j])
        {
            a = number[i];
            number[i] = number[j];
            number[j] = a;
        }
    }
}
printf("The numbers arranged in descending order are given below \n");
for (i = 0; i < n; ++i)
{
    printf("%d\n", number[i]);
}
printf("The 2nd largest number is = %d\n", number[1]);
printf("The 2nd smallest number is = %d\n", number[n - 2]);

getch();
}

```

### Output

```

Enter the value of N
5
Enter the numbers
25
34
12
70
92
The numbers arranged in descending order are given below
92
70
34

```

25

12

The 2nd largest number is = 70  
The 2nd smallest number is = 25

#### 14. C program to find the largest element in an array.

```
/* C Program to Find the Largest Number in an Array */
#include <stdio.h>
#include<conio.h>

int main()
{
    int array[50], size, i, largest;
    clrscr();
    printf("\n Enter the size of the array: ");
    scanf("%d", &size);
    printf("\n Enter %d elements of the array: ", size);
    for (i = 0; i < size; i++)
        scanf("%d", &array[i]);
    largest = array[0];
    for (i = 1; i < size; i++)
    {
        if (largest < array[i])
            largest = array[i];
    }
    printf("\n largest element present in the given array is : %d",
    largest);
    getch();
    return 0;
}
```

#### Output

Enter the size of the array: 5

Enter 5 elements of the array:

12

56

34

78

largest element present in the given array is : 100

### 15. C program to sort the N names in alphabetical order.

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

void main()
{
    char name[10][8], tname[10][8], temp[8];
    int i, j, n;
    clrscr();
    printf("Enter the value of n \n");
    scanf("%d", &n);
    printf("Enter %d names n", \n);
    for (i = 0; i < n; i++)
    {
        scanf("%s", name[i]);
        strcpy(tname[i], name[i]);
    }
    for (i = 0; i < n - 1 ; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (strcmp(name[i], name[j]) > 0)
            {
                strcpy(temp, name[i]);
                strcpy(name[i], name[j]);
                strcpy(name[j], temp);
            }
        }
    }
    printf("\n-----\n");
    printf("Input Names \t\t Sorted names\n");
    printf("-----\n");
    for (i = 0; i < n; i++)
    {
        printf("%s\t\t%s\n", tname[i], name[i]);
    }
    printf("-----\n");
```

```
getch();
}
```

### Output

```
Enter the value of n
```

```
5
```

```
Enter 5 names
```

```
Sushmitha
```

```
Rashmi
```

```
Kiran
```

```
Palguni
```

```
Thyagu
```

---

| Input Names | Sorted names |
|-------------|--------------|
| Sushmitha   | Kiran        |
| Rashmi      | Palguni      |
| Kiran       | Rashmi       |
| Palguni     | Sushmitha    |
| Thyagu      | Thyagu       |

### 16. C Program to insert an element into a specified position in a given array.

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

void main()
{
    int array[10];
    int i, j, n, m, temp, key, pos;
    clrscr();

    printf("Enter how many elements \n");
    scanf("%d", &n);
    printf("Enter the elements \n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &array[i]);
    }
    printf("Input array elements are \n");
    for (i = 0; i < n; i++)
    {
        printf("%d\n", array[i]);
    }
```

```

for (i = 0; i < n; i++)
{
    for (j = i + 1; j < n; j++)
    {
        if (array[i] > array[j])
        {
            temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }
}
printf("Sorted list is \n");
for (i = 0; i < n; i++)
{
    printf("%d\n", array[i]);
}
printf("Enter the element to be inserted \n");
scanf("%d", &key);
for (i = 0; i < n; i++)
{
    if (key < array[i])
    {
        pos = i;
        break;
    }
}
m = n - pos + 1 ;
for (i = 0; i <= m; i++)
{
    array[n - i + 2] = array[n - i + 1] ;
}
array[pos] = key;
printf("Final list is \n");
for (i = 0; i < n + 1; i++)
{
    printf("%d\n", array[i]);
}
getch()
}

```

/\*pos is the position where element is to be inserted in the array ie., array(pos) = key; all the following elements are to be pushed forward by 1 position.  
for (i=0 to upto m times.) \*/

### Output:

Enter how many elements

5

Enter the elements

```
76
90
56
78
12
Input array elements are
76
90
56
78
12
Sorted list is
12
56
76
78
90
Enter the element to be inserted
61
Final list is
12
56
61
76
78
90
```

### 17. C program to delete the specified integer from a given array.

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

void main()
{
    int vectorx[10];
    int i, n, pos, element, found = 0;
    clrscr();

    printf("Enter how many elements\n");
    scanf("%d", &n);
    printf("Enter the elements\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &vectorx[i]);
```

```

}
printf("Input array elements are\n");
for (i = 0; i < n; i++)
{
    printf("%d\n", vectorx[i]);
}
printf("Enter the element to be deleted\n");
scanf("%d", &element);
for (i = 0; i < n; i++)
{
    if (vectorx[i] == element)
    {
        found = 1;
        pos = i;
        break;
    }
}
if (found == 1)
{
    for (i = pos; i < n - 1; i++)
    {
        vectorx[i] = vectorx[i + 1];
    }
    printf("The resultant vector is \n");
    for (i = 0; i < n - 1; i++)
    {
        printf("%d\n", vectorx[i]);
    }
}
else
    printf("Element %d is not found in the vector\n", element);
getch();
}

```

*/\*pos is the position where element is to be deleted from the array and all the following elements are to be pushed backward by 1 position.*  
*for (i=pos;..... \*/*

### Output

Enter how many elements

4

Enter the elements

45

34

78

87

Input array elements are

45

34

```
78  
87  
Enter the element to be deleted  
34  
The resultant vector is  
45  
78  
87
```

### 3.4.2 Programming Examples based on two dimensional arrays

#### 1. C program to read and write a matrix of size m X n using row major order.

```
#include<stdio.>  
#include<conio.h>  
  
main()  
{ int a[10][10] ,i,j,m,n ;  
clrscr();  
  
printf("\n Enter the row size and column size \n");  
scanf("%d %d",&m,&n);  
  
printf("\n Enter the elements of matrix Row Wise \n");  
  
for(i=0;i<m;i++) // 0 to m-1 rows  
{ for(j=0;j<n;j++) // 0 to n- 1 columns  
    scanf("%d" ,&a[i][j]); // read a[i][j]  
}  
printf("\n The entered matrix is \n");  
  
for(i=0;i<m;i++) // 0 to m-1 rows  
{ for(j=0;j<n;j++) // 0 to n- columns  
    { printf("%d " , a[i][j]); // print a[i][j] of each row  
    }  
    printf("\n"); // To print the next row  
}  
getch();  
}
```

**Output :**

Enter the row size and column size

3 3

Enter the elements of matrix Row Wise

1 1 1

2 2 2

3 3 3

The entered matrix is

1 1 1

2 2 2

3 3 3

**2. C Program to read and write a matrix column wise .**

```
#include<stdio.h>
#include<conio.h>
main()
{ int a[10][10],i,j,m,n;
clrscr();
printf("\n Enter the row size and column size \n");
scanf("%d %d",&m,&n);

printf("\n Enter the elements of matrix column wise \n");
for(j=0;j<n;j++)           // 0 to n- 1 columns
{ for(i=0;i<m;i++)         // 0 to m-1 rows
    { scanf("%d",&a[i][j]); // reading the elements
columnwise
    }
}
printf("\n The entered matrix column wise is \n");
for(j=0;j<n;j++)           // 0 to n- 1 columns
{ for(i=0;i<m;i++)         // 0 to m-1 rows
    { printf("%d\t",a[i][j]); // printing the elements
columnwise
    }
}
printf("\n");
}
getch();
}
```

**Output :**

Enter the row size and column size

```

3      3
Enter the elements of matrix column wise
1      2      3
1      2      3
1      2      3

The entered matrix x column wise is
1      1      1
2      2      2
3      3      3

```

### 3. Write a C program to find the greatest number from two dimensional array.

```

#include<stdio.h>
#include<conio.h>
main()
{ int m, n, c, d, matrix[10][10], maximum;
  clrscr();
  printf("Enter the number of rows and columns of 2d array
(matrix)\n");
  scanf("%d%d",&m,&n);
  printf("Enter the elements of 2d array (matrix)\n");

  for( c = 0 ; c < m ; c++ )
  { for( d = 0 ; d < n ; d++ )
    { scanf("%d",&matrix[c][d]);
    }
  }

  maximum = matrix[0][0];

  for( c = 0 ; c < m ; c++ )
  { for( d = 0 ; d < n ; d++ )
    { if( matrix[c][d] > maximum )
        maximum = matrix[c][d];
    }
  }

  printf("The greatest element in the 2D array is %d\n",maximum);
  getch();
  return 0;
}

```

**4. C program to read two matrices A (mXn) and B(mXn) and perform i. Addition and ii. Subtraction of A and B . Output the given matrices and Resultant matrices.**

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

main()
{
int a[5][5],b[5][5],sum[5][5],sub[5][5],i,j,m,n;
clrscr();

printf("\n Enter the row size and column size of
matrices \n");
scanf("%d %d",&m,&n);

printf("\n Enter the elements of matrix A");
for(i=0;i<m;i++)
for(j=0;j<n;j++)
scanf("%d", &a[i][j]);

printf("\n Enter the elements of matrix B");
for(i=0;i<m;i++)
for(j=0;j<n;j++)
scanf("%d", &b[i][j]);

for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        sum[i][j] = a[i][j] + b[i][j];
        sub[i][j] = a[i][j] - b[i][j];
    }
}

printf("\n The sum of two matrices is \n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        printf("%d\t",sum[i][j]);
    }
}
```

```

        printf("\n");
    }

printf("\n The difference of two matrices is \n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    {
        printf("%d\t",sub[i][j]);
    }
    printf("\n");
}
getch();
}

```

**Output :**

**Enter the row size and column size of matrices**

2     3

**Enter the elements of matrix A**

1     1   3  
2     2   4

**Enter the elements of matrix B**

2     2   2  
3     3   3

**The sum of two matrices is**

3     3   5  
5     5   7

**The difference of two matrices is**

-1    -1   1  
-1    -1   1

5. C program to read a matrix A(mXn) and to find the transpose of the given matrix and output the input and the transposed matrix.

```

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

main()
{
    int   a[5][5] ,tr[5][5],m,n,i,j ;
    clrscr();

```

```

printf("\n Enter the order of the matrix A\n");
scanf("%d %d",&m,&n);

printf("\n Enter the elements of matrixA \n");
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
        {scanf("%d", &a[i][j]);
        }
}
for(i=0;i<m;i++)
{ for(j=0;j<n;j++)
    { tr[j][i] = a[i][j];
    }
}
/* from 0 to m-1 row and
from 0 to n-1 column in each
of that row, transposed
matrix row index= given
matrix column index and
transpose matrix column
index = given matrix row
index. */

```

printf("\n The transposed matrix is\n");

```

for(i=0;i<n;i++)
{ for(j=0;j<m;j++)
    { printf("%d\t", tr[i][j]);
    }
    printf("\n");
}

```

```

getch();
}

```

### Output :

**Enter the order of matrix A**

2      3

**Enter the elements of matrix A**

2      3    4  
4      5    6

**The transposed matrix is**

2    4  
3    5  
4    6

- 6. C program to read a square matrix A (mXn) and to find the trace of the matrix. ( The trace of the matrix is the sum of all elements along its principal diagonal )**

```
#include<stdio.h>
main()
{ int a[5][5],i,j,m,n,trace ;
clrscr();

printf("\nEnter the order of the matrix A\n");
scanf("%d %d",&m,&n);
printf("\nEnter the elements of matrix A\n");
for(i=0 ;i<m;i++)
{
    for(j=0;j<n;j++)
    { scanf("%d",&a[i][j]);
    }
}

trace = 0; /* Diagnol elements of a matrix
means row index = cloumn index
ie., a[1,1], a[2,2], a[3,3], a[4,4]..
*/
for(i=0;i<m;i++)
{
    for(j=0;j<n;j++)
    { if(i==j)
        trace = trace +a[i][j];
    }
}

printf("\n Trace of the matrix = %d", trace);
getch();
}
```

**Output :**

**Enter the order of the matrix A**

**3      3**

**Enter the elements of the matrix A**

**2      2      2  
5      5      5  
6      6      6**

**Trace of the matrix = 13**

- 7. Write a C program that reads two matrices A(m X n ) and B( pXq) and compute product of matrices A and B . Read matrix A and matrix B in row major order and in column major order respectively. Print both the input matrices and resultant matrix with suitable headings and output should be in matrix format only. Program must check the compatibility of orders of the matrices for multiplication. Report appropriate message in case of incompatibility.**

```
#include <stdio.h>
#include<conio.h>
main()
{
    int a[5][5] , b[5][5] ,p[5][5];
    int i,j,k,m,n,p,q ;
    clrscr();
    printf("\n Enter the order of the first matrix \n");
    scanf("%d %d",&m,&n) ;
    printf("\n Enter the order of second matrix \n");
    scanf("%d %d",&p,&q);
    if(n!=p)
    {
        printf("\nThe matrix multiplication cannot be performed \n");
        getch();
        exit(0);
    }
    printf("\n Enter the matrix A row wise \n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }

    printf("\n Enter the matrix B column wise \n");
    for(j=0;j<q;j++)
    {
        for(i=0;i<p;i++)
        {
            scanf("%d",&b[i][j]);
        }
    }
    /* Matrix Multiplication */
    for(i=0;i<m;i++)
        /* Each element in the product
           of a matrix is the sum of
           products of corresponding
           elements of xth row and yth
           column */

```

```

{ for(j=0;j<q;j++)
  { p[i][j] =0 ;
    for(k=0;k<n;k++)
      { p[i][j] =p[i][j] + a[i][k]*b[k][j] ;
      }
    }
}
printf("\n The resultant matrix is \n");

for(i=0;i<m;i++)
{
  for(j=0;j<q;j++)
  { printf ("%d\t",p[i][j]);
  }
  printf("\n");
}
getch();
}

```

### Output:

```

Enter the order of first matrix : 2  2
Enter the order of second matrix : 2  2
Enter Matrix A row wise:
12  56
45  78
Enter Matrix B column wise:
2  5
6  8
The resultant matrix is:
304 520
480 894

```

**8. Develop implement and execute a C program to search a name in a list of names using binary searching technique.**

```

#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
    char name[10][20],key[20];
    int n,i,low,high,mid;
    clrscr();

```

```

printf("\n Enter the number of names to read \n");
scanf("%d",&n);
printf("\n Enter the names in ascending order \n");
for(i=0;i<n;i++)
scanf("%s",name[i]);
printf("\n Enter the name to be search \n");
scanf("%s",key)
    low=0;
    high = n-1;
    while(low<=high)
    {
        mid = (low+high)/2 ;
        {
            if(strcmp(name[mid],key)==0)
            {
                printf("\n Key found at position %d \n",mid+1);
                getch();
                exit(0);
            }
            else if (strcmp(name[mid],key) <0)
            { high = high ;
                low = mid +1;
            }
            else
            {
                low= low;
                high = mid -1;
            }
        }
    }
    getch();
}

```

**Output :**

```

Enter the number of names to read
5
Enter the names in ascending order
Aruna
Kiran
Palguni
Rashmi
Tisen
Enter the name to be searched
Rashmi
Key found at the position 4

```

**9. C program to compare whether the two matrices are equal to each other.**

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

void main()
{
    int a[10][10], b[10][10];
    int i, j, row1, column1, row2, column2, flag = 1;
    clrscr();
    printf("Enter the order of the matrix A \n");
    scanf("%d %d", &row1, &column1);
    printf("Enter the order of the matrix B \n");
    scanf("%d %d", &row2, &column2);
    printf("Enter the elements of matrix A \n");
    for (i = 0; i < row1; i++)
    {
        for (j = 0; j < column1; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
    printf("Enter the elements of matrix B \n");
    for (i = 0; i < row2; i++)
    {
        for (j = 0; j < column2; j++)
        {
            scanf("%d", &b[i][j]);
        }
    }
    printf("MATRIX A is \n");
    for (i = 0; i < row1; i++)
    {
        for (j = 0; j < column1; j++)
        {
            printf("%3d", a[i][j]);
        }
        printf("\n");
    }
    printf("MATRIX B is \n");
    for (i = 0; i < row2; i++)
```

```

{
    for (j = 0; j < column2; j++)
    {
        printf("%3d", b[i][j]);
    }
    printf("\n");
}
/* Comparing two matrices for equality */
if (row1 == row2 && column1 == column2)
{
    printf("Matrices can be compared \n");
    for (i = 0; i < row1; i++)
    {
        for (j = 0; j < column2; j++)
        {
            if (a[i][j] != b[i][j])
            {
                flag = 0;
                break;
            }
        }
    }
    else
    {
        printf(" Cannot be compared\n");
        exit(1);
    }
    if (flag == 1)
        printf("Two matrices are equal \n");
    else
        printf("But, two matrices are not equal \n");
    getch();
}

```

### Output

**Run 1:**

Enter the order of the matrix A

2

Enter the order of the matrix B

2

Enter the elements of matrix A

23 56

45 80

Enter the elements of matrix B

50 26

39 78

MATRIX A is

23 56

45 80

MATRIX B is

50 26

39 78

Matrices can be compared

But,two matrices are not equal

### **Run 2:**

Enter the order of the matrix A

2 2

Enter the order of the matrix B

2 2

Enter the elements of matrix A

10 50

15 30

Enter the elements of matrix B

10 50

15 30

MATRIX A is

10 50

15 30

MATRIX B is

10 50

15 30

Matrices can be compared

Two matrices are equal

## **10.C program to check if a given matrix is identity matrix or not.**

**/\* Identity matrix means each of the elements is either 0 or 1 \*/**

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

```
void main()
```

```

{
int a[10][10];
int i, j, row, column, flag = 1;
clrscr();

printf("Enter the order of the matrix A \n");
scanf("%d %d", &row, &column);
printf("Enter the elements of matrix A \n");
for (i = 0; i < row; i++)
{
    for (j = 0; j < column; j++)
    {
        scanf("%d", &a[i][j]);
    }
}
printf("MATRIX A is \n");
for (i = 0; i < row; i++)
{
    for (j = 0; j < column; j++)
    {
        printf("%3d", a[i][j]);
    }
    printf("\n");
}
/* Check for unit (or identity) matrix */
for (i = 0; i < row; i++)
{
    for (j = 0; j < column; j++)
    {
        if (a[i][j] != 1 && a[j][i] != 0) //checking for 0 and 1
        {
            flag = 0;
            break;
        }
    }
}
if (flag == 1 )
    printf("It is identity matrix \n");
else
    printf("It is not a identity matrix \n");
}

```

## Output

**Run 1:**

Enter the order of the matrix A

3 3

Enter the elements of matrix A

1 2 3

5 1 8

6 4 1

MATRIX A is

1 2 3

5 1 8

6 4 1

It is not a identity matrix

**Run 2:**

Enter the order of the matrix A

3 3

Enter the elements of matrix A

1 0 0

0 1 0

0 0 1

MATRIX A is

1 0 0

0 1 0

0 0 1

It is identity matrix

**11.C program to calculate the sum of elements of each row and column.**

```
#include <stdio.h>
#include<conio.h>
int Addrow(int array1[10][10], int k, int c);
int Addcol(int array1[10][10], int k, int r);

void main()
{
    int arr[10][10];
    int i, j, row, col, rowsum, colsum, sumall=0;
    clrscr();
```

```

printf("Enter the order of the matrix \n");
scanf("%d %d", &row, &col);
printf("Enter the elements of the matrix \n");
for (i = 0; i < row; i++)
{
    for (j = 0; j < col; j++)
    {
        scanf("%d", &arr[i][j]);
    }
}
printf("Input matrix is \n");
for (i = 0; i < row; i++)
{
    for (j = 0; j < col; j++)
    {
        printf("%3d", arr[i][j]);
    }
    printf("\n");
}
/* computing row sum */
for (i = 0; i < row; i++)
{
    rowsum = Addrow(arr, i, col);
    printf("Sum of row %d = %d\n", i + 1, rowsum);
}
/* computing col sum */
for (j = 0; j < col; j++)
{
    colsum = Addcol(arr, j, row);
    printf("Sum of column %d = %d\n", j + 1, colsum);
}
/* computation of all elements */
for (j = 0; j < row; j++)
{
    sumall = sumall + Addrow(arr, j, col);
}
printf("Sum of all elements of matrix = %d\n", sumall);
}
/* Function to add each row */
int Addrow(int array1[10][10], int k, int c)
{
    int rsum = 0, i;
    for (i = 0; i < c; i++)
    {

```

```

        rsum = rsum + array1[k][i];
    }
    return(rsum);
}
/* Function to add each column */
int Addcol(int array1[10][10], int k, int r)
{
    int csum = 0, j;
    for (j = 0; j < r; j++)
    {
        csum = csum + array1[j][k];
    }
    return(csum);
}

```

### Output

Enter the order of the matrix

3 3

Enter the elements of the matrix

2 3 4

7 1 5

3 8 9

Input matrix is

2 3 4

7 1 5

3 8 9

Sum of row 1 = 9

Sum of row 2 = 13

Sum of row 3 = 20

Sum of column 1 = 12

Sum of column 2 = 12

Sum of column 3 = 18

Sum of all elements of matrix = 42

**12.C program to find the trace and normal of a matrix.** (Trace is defined as the sum of main diagonal elements and Normal is defined as square root of the sum of all the elements )

```

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

void main ()
{
    static int array[10][10];

```

```

int i, j, m, n, sum = 0, sum1 = 0, a = 0, normal;
clrscr();

printf("Enter the order of the matrix\n");
scanf("%d %d", &m, &n);
printf("Enter the n coefficients of the matrix \n");
for (i = 0; i < m; ++i)
{
    for (j = 0; j < n; ++j)
        {
            /* a contains the square of each matrix element */
            scanf("%d", &array[i][j]);
            a = array[i][j] * array[i][j];
            sum1 = sum1 + a; /*sum1 will have the sum of all such squares*/
        }
}
normal = sqrt(sum1);
printf("The normal of the given matrix is = %d\n", normal);
for (i = 0; i < m; ++i)
{
    sum = sum + array[i][i];
    /* a[i,i] implies diagonal element, a[1,1], a[2,2], a[3,3]... */
}
printf("Trace of the matrix is = %d\n", sum);
getch();
}

```

### Output

```

Enter the order of the matrix
3 3
Enter the n coefficients of the matrix
3 7 9
2 6 10
8 5 9
The normal of the given matrix is = 21
Trace of the matrix is = 18

```

### 3.4.3 Programming Examples based on Strings

#### 1. Program to read and write multiple strings.

```
#include<stdio.h>
#include<conio.h>
main()
{
    char   a[10][20];
    int    n,i;
    clrscr();
    printf("\n Enter the number of names \n");
    scanf("%d",&n);
    printf("\n Enter %d names one by one \n",n);
    for(i=0;i<n;i++)
        scanf("%s",a[i]);
    printf ("\n The entered %d names are :\n",n);
    for(i=0;i<n;i++)
        printf("%s",a[i]);
    getch();
}
```

#### Output

```
Enter the number of names
3
Enter 3 names one by one
Akash
Ravi
Bob
The entered 3 names are :
Akash
Ravi
Bob
```

#### 2. Program to reverse a string without using library functions

```
#include<stdio.h>
#include<conio.h>
main()
{
    int    i,j ;
    char  str[100],temp[100];
```

```

clrscr();
printf("n Enter a string to reverse\n");
gets(str);
i = strlen(str) - 1 ;
j = 0 ;
while(i>=0)
{
    temp[j] = str[i];
    j++;
    i--;
}
temp[j] = '\0';
printf(" Reversed String :is \n");
puts(temp);
getch();
}

```

### Output

Enter a string to reverse  
RAJU  
Reversed String is UJAR

### 3. Program to check whether a string is palindrome or not .

```

#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{ char s1[20],s2[20];
  int result ;
  clrscr();
  printf("n Enter any string \n");
  gets(s1);
  strcpy(s2,s1);
  strrev(s2);
  result = strcmp(s1,s2);

  if(result ==0)
    printf("n PALINDROME \n");
  else
    printf("n NOT A PALINDROME\n");
  getch();
}

```

**Note :** /\* strcpy () copies s1 to s1;  
\*strrev() reverses s2  
\*strcmp(s1,s2) compares both and  
returns a value that's equated to  
result. If it =0 then s1 and s2 are  
equal else unequal. \*/

}

### Output

**Run 1:**

Enter any string  
GADAG  
PALINDROME

**Run 2:**

Enter any string  
THYAGI  
NOT A PALINDROME

#### 4. Program to copy one string to another without using strcpy .

```
#include<stdio.h>
#include<conio.h>
main()
{
    char str1[20],str2[20];
    int i;
    clrscr();

    printf("\n Enter the value of string1\n");
    gets(str1);

    for(i=0;str1[i]!='\0'; i++)
        str2[i]= str1[i];
    str2[i]='\0';
    printf("\n The value of string2 is \n");
    puts(str2);
    getch();
}
```

**Note :** The characters from str1[0] till it reaches end of string ie, '\0' is copied to str2, finally str2[i] = '\0' to indicate end of string of str2.

### Output

Enter the value of string1  
Aabbccdd  
The value of the string2 is  
Aabbccdd

**5. C program to find the length and to determine whether the given string is palindrome or not without using library functions.**

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

void main()
{
    char string[25], reverse_string[25] = {'\0'};
    int i, length = 0, flag = 0;
    clrscr();

    printf("Enter a string \n");
    gets(string);
    /* Determine the length*/
    for (i = 0; string[i] != '\0'; i++)
    {
        length++;
    }
    printf("The length of the string '%s' = %d\n", string, length);
```

```
for (i = length - 1; i >= 0 ; i--)
{
    reverse_string[length - i - 1] = string[i];
}
```

*/\* Note : from the last character to first char each character is copied to reverse\_string ie., i= length -1 (last char in the beginning) reverse\_string (length-(length-1) -1) implies reverse\_string[0] is copied from string[length-1] and it continues till reverse string[length-1] is copied from str[0] \*/*

```
/* Check if the string is a Palindrome */
/* Note : flag , an indicator is set to 1 . Each character of reverse-string is compared with string . If any one char is not equal the flag = 0 , after checking all the characters if flag remains as 1 it means all the characters are equal hence palindrome . if flag is changed to 0 then it implies its not a palindrome.*/
    for (flag = 1, i = 0; i < length ; i++)
    {
        if (reverse_string[i] != string[i])
            flag = 0;
    }
    if (flag == 1)
```

```
    printf ("%s is a palindrome \n", string);
else
    printf("%s is not a palindrome \n", string);
getch();
}
```

### Output

**Run 1:**

```
Enter a string
India is great
The length of the string 'india is great' = 14
India is great is not a palindrome
```

**Run 2:**

```
Enter a string
madam
The length of the string 'madam' = 5
madam is a palindrome
```

**Run 3:**

```
Enter a string
mam
The length of the string 'mam' = 3
mam is a palindrome
```

## 6. C program to concatenate two strings without using library functions:

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

void main()
{
    char string1[20], string2[20];
    int i, j, pos;
    clrscr();

    /* Initialize the string to NULL values */
    memset(string1, 0, 20);
    memset(string2, 0, 20);
```

```

printf("Enter the first string : ");
scanf("%s", string1);
printf("Enter the second string: ");
scanf("%s", string2);
printf("First string = %s\n", string1);
printf("Second string = %s\n", string2);

/* Concate the second string to the end of the first string */
for (i = 0; string1[i] != '\0'; i++)
{
    /* null statement: simply traversing the string1 */
    ;
}
pos = i;
for (j = 0; string2[j] != '\0'; i++)
{
    string1[i] = string2[j++];
}
/* set the last character of string1 to NULL */
string1[i] = '\0';
printf("Concatenated string = %s\n", string1);
getch();
}

```

### Output

```

Enter the first string : Twenty
Enter the second string: Five
First string = Twenty
Second string = Five
Concatenated string = TwentyFive

```

## 7. C program to change the case (upper to lower and lower to upper) of characters .

```

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

void main()
{

```

```

char sentence[100];
int count, ch, i;
clrscr();

printf("Enter a sentence \n");
for (i = 0; (sentence[i] = getchar()) != '\n'; i++)
{
    ;
}
sentence[i] = '\0';
/* shows the number of chars accepted in a sentence */
count = i;
printf("The given sentence is : %s", sentence);
printf("\n Case changed sentence is: ");
for (i = 0; i < count; i++)
{
    ch = islower(sentence[i])? toupper(sentence[i]) :
    tolower(sentence[i]);
    putchar(ch);
}
getch()
}

```

### Output

Enter a sentence  
wELCOME tO ujire  
The given sentence is : wELCOME tO ujire  
Case changed sentence is: Welcome To UJIRE

#### **Note :**

// islower() library function checks whether the character is lower case  
//toupper() library function converts the given character to uppercase.  
//tolower() library function converts the given character to lowercase.

### **8. C program to count the number of words in a given sentence .**

```

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

void main()
{
    char s[200];

```

```
int count = 0, i;
clrscr();
printf("enter the string\n");
scanf("%[^n]s", s);
for (i = 0;s[i] != '\0';i++)
{
    if (s[i] == ' ')
        count++;
}
printf("number of words in given string are: %d\n", count + 1);
getch();
}
```

**Output:**

```
enter the string
Welcome to Thyagu's C Programming Class
number of words in given string are: 6
```

### 3.4.4 Programming Examples based on Functions

1. Write a program to compute the factorial of a given number n using recursion :

$$\text{fact}(n) = \begin{cases} 1 & \text{if } n == 0 \\ n * \text{fact}(n - 1) & \text{if } n > 0 \end{cases}$$

```
#include<stdio.h>
#include<conio.h>
int fact(int);
void main()
{
    int n,ans;
    clrscr();
    printf("\nEnter the value of n \n");
    scanf("%d",&n);
    ans = fact(n);
    printf("fact(%d) = %d ",n,ans);
    getch();
}
int fact(int n)
{
    if(n==0)
    {
        return 1;
    }
    else
    {
        return n*fact(n-1);
    }
}

/* ex., 5*(4*(3*(2*(1)))) therefore factorial of n, fact(n)= fact(n-1)*n */
```

#### Output

Enter the value of n

5

fact(5) = 120

2. Write a C program to find the cube of a number using function.

```

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

int cube (int num);

main( )
{
    int n,c ;
    clrscr( );
    printf("n Enter the number \n");
    scanf("%d",&n);
    c = cube(n);
    printf("n The cube of a number = %d\n",c);
    getch();
}

int cube( int num )
{
    int x;
    x = pow(num,3);
    return x;
}

```

**Output :**

```

Enter the number
3
The cube of a number = 27

```

### 3. Write a program to find the $n^{\text{th}}$ Fibonacci number using recursion

Note :  $\text{fib}(n) = \{ 0, 1, 1, 2, 3, 5, 8, 13, \dots \}$

$$\text{fib}(n) = \begin{cases} 0 & \text{if } n == 1 \\ 1 & \text{if } n == 2 \\ \text{fib}(n - 1) + \text{fib}(n - 2) & \text{if } n > 2 \end{cases}$$

```

#include<stdio.h>
#include<conio.h>
int fib(int);

```

```

void main()
{
    int n,ans ;
    clrscr();
    printf("n Enter the value of n\n");
    scanf("%d",&n);
    ans = fib(n) ;
    printf("n The %dth Fibonacci number is %d ",n, ans);
    getch();
}

int fib(int n)
{
    if(n==1)
        return 0;
    if( n==2)
        return 1;
    if(n>2)
        return fib(n-1) + fib(n-2);
}

```

### Output

Enter the value of **n**  
**7**  
The **7th** Fibonacci number is **8**

4. Write a recursive function for computing the value of  $x^n$  given **x** and **n** as input arguments .

```

#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float x,sum;
    int n;
    float fun(float x, int n);
    clrscr();
    printf("n Enter x and n\n");
    scanf("%f%d",&x,&n);
    printf("The value of x^n = %f\n",func(x,n));
    getch();
}

```

```

float fun(float x, int n)
{
    float s = 0;
    if(n == 1)
        return (x);
    else
        s = x * fun(x, n - 1);
    return (s);
}

```

### **Output**

Enter x and n  
**2 5**  
The value of  $x^n = 32.000000$

**Note :**

// Example : x to the power of 5,  $x^5=x*x^4$ ,  $x^4=x*x^3$ .....  
// therefore  $x^n = \text{pow}(x,n) = \text{pow}(x,n-1)*x$  or  $\text{fun}(x,n) = x * \text{fun}(x, n-1)$

**5. Write a C program to calculate the Fibonacci sequence , using recursive function.**

```

#include<stdio.h>
#include<conio.h>
int fib(int n);
void main()
{
    int i, n ;
    clrscr();
    printf("\n Enter the value of n");
    scanf("%d",&n);
    printf("\n Fibonacci series upto %d terms is :\n");
    for (i=1;i<=n;i++)
        printf("\n %d",fib(i));
    getch();
}
int fib(int n)
{
    if(n==1)
        return 0;
    else if (n == 2)
        return 1;
    else

```

```
        return fib(n-1) + fib(n-2);
    }
```

### Output

```
Enter the value of n
10
Fibonacci series upto 10 terms is :
0   1   1   2   3   5   8   13  21  34
```

## 6. C Program to find the prime numbers between the given range

```
#include<stdio.h>
#include<conio.h>
int check_prime(int num);
int main(){
    int n1,n2,i,flag;
    printf("Enter two numbers(intervals): ");
    scanf("%d %d",&n1, &n2);
    printf("Prime numbers between %d and %d are: ", n1, n2);
    for(i=n1+1;i<n2;++i)
    {
        flag=check_prime(i);
        if(flag==0)
            printf("%d ",i);
    }
    return 0;
}
int check_prime(int num) /* Function to check prime number*/
{
    int j,flag=0;
    for(j=2;j<=num/2;++j){
        if(num%j==0){
            flag=1;
            break;
        }
    }
    return flag;
}
```

### Output

```
Enter two numbers(intervals): 10    30
Prime numbers between 10 and 30 are: 11 13 17 19 23 29
```

**Note :**

// if a number is not divisible by any number from 2 to its half then it is prime number  
// therefore the num is checked from 2 to num/2, if the remainder is 0 ie, divisible the flag is set to 1 which was initialized to 0  
// after checking with all the numbers if flag = 0 it means its a prime number hence print it

## 7. C program to find the hcf of a given two numbers using recursion.

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

int hcf(int, int);

int main()
{
    int a, b, result;
    clrscr();
    printf("Enter the two numbers to find their HCF: ");
    scanf("%d%d", &a, &b);
    result = hcf(a, b);
    printf("The HCF of %d and %d is %d.\n", a, b, result);
    getch();
}

int hcf(int a, int b)
{
    while (a != b)
    {
        if (a > b)
        {
            return hcf(a - b, b);
        }
        else
        {
            return hcf(a, b - a);
        }
    }
}
```

```
    return a;  
}
```

**Output:**

```
Enter the two numbers to find their HCF: 24 36  
The HCF of 24 and 36 is 12.
```

## 8.C program to find the product of two numbers using recursion.

```
#include <stdio.h>  
#include<conio.h>  
int product(int, int);  
  
int main()  
{  
    int a, b, result;  
    clrscr();  
    printf("Enter two numbers to find their product: ");  
    scanf("%d%d", &a, &b);  
    result = product(a, b);  
    printf("Product of %d and %d is %d\n", a, b, result);  
    getch();  
    return 0;  
}  
  
int product(int a, int b)  
{  
    if (a < b)  
    {  
        return product(b, a);  
    }  
    else if (b != 0)  
    {  
        return (a + product(a, b - 1));  
    }  
    else  
    {  
        return 0;  
    }  
}
```

**Output :**

```
Enter two numbers to find their product: 20 30
Product of 20 and 30 is 600
```

### 9.C program to find the sum of n natural numbers using recursion .

```
#include<stdio.h>
int add(int n);
int main()
{
    int n;
    printf("Enter an positive integer: ");
    scanf("%d",&n);
    printf("Sum = %d",add(n));
    return 0;
}
int add(int n)
{
    if(n!=0)
        return n+add(n-1); /* recursive call */
}
```

#### Output:

```
Enter a positive integer: 10
Sum = 55
```

**Note : // ex., 5+(4+(3+(2+(1))))**  
**//therefore sum of n, add(n)= n+add(n-1)**

### 10.Write a C program to check a number is prime number or not using recursion .

```
#include <stdio.h>
#include<conio.h>
int primeno(int n, int i);

int main()
{
    int num, flag;
    printf("Enter a number: ");
    scanf("%d",&n);
```

```
flag = primeno(n,n/2);
if (flag == 1)
{ printf("%d is a prime number\n", n);
}
else
{ printf("%d is not a prime number\n", n);
}
return 0;
}

int primeno(int n, int i)
{
    if (i == 1)
    { return 1;
    }
    else
    { if (n % i == 0)
        { return 0;
        }
        else
        { return primeno(n, i - 1);
        }
    }
}
```

**Output:**

**Enter a number: 10**  
**10 is not a prime number**

**3.5 Exercises**

## **1. Arrays**

### **Part A : Theory**

1. What is an array? What are the types of arrays .Give example.
2. What is Single dimensional array? Explain the declaration and initialization of Single dimensional array with example.
3. What is sorting? Give Example. Mention any two types of sorting techniques used in C.
4. What is searching? Give Example .Mention any two types of searching techniques used in C.
5. What is Bubble Sort? Explain with an example.
6. What is Linear Search? Explain with an example.
7. What is Binary Search? Explain with an example.
8. What is two dimensional array? Explain the declaration and initialization of two dimensional arrays with example.
9. Explain how to read and write two dimensional arrays using *i. Row major order* and *ii. Column major order* with suitable programming examples.
10. Mention the advantages and disadvantages of arrays. What are parallel arrays? Give Example
11. Given the declaration : int item[25] ,amount[25], i; Answer the following questions
  - a. If i=8, what array element is referred to by the item[i-3] ?
  - b. If i is 24 , what element is amount[i+1] ?

### **Part B: Programs**

1. Write a C program to find the average of marks scored by a student in n number of subjects using single dimensional arrays.

2. Write a C program to input n real numbers and to find mean ,variance and standard deviation
3. Write a C program to evaluate the polynomial  $f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$  for given value of x and its coefficients using Horner's method.
4. Write a C program that reads N integer numbers and arrange them in ascending order using Bubble Sort technique.
5. Write a C program to implement Binary Search technique for given sorted integers numbers in ascending order by accepting key element from the user.
6. Write a C program to read and Write two dimensional arrays in Row major order or column major order.
7. Write a C program that reads two matrices A (mXn) and B(mXn) and compute sum (SUM) and differences (SUB) of two matrices A and B . Read matrix A and matrix B in row major order. At the end the program must display the matrix A, B ,SUM and SUB.
8. Write a C program that reads two matrices A (mXn) and B(pXq) and compute product of two matrices A and B . Read matrix A and matrix B in row major order and in column major order respectively. Print both the input matrices and resultant matrix with suitable headings and output should be in matrix format only.

## 2. Strings

### Part A: Theory

1. What is string? Give example. What is the difference between string and array of characters? Explain with example.
2. Explain the declaration and initialization of strings with example.
3. Explain Reading and Writing (Printing) a strings using scanf and printf respectively with example.

4. Explain reading and writing a string with gets and puts respectively with example.
5. Explain the concept of arrays of strings with examples.
6. Explain string manipulation library functions with examples.
7. Why is it wrong to use the & symbol when reading in a string with scanf ?
8. Show three different ways to give the value “animal” to the string variable str declared as follows : char str[10];
9. What is stored in str if each of the following values separately is read in using a single call to scanf ?
  - a. Cityscape
  - b. New York
  - c. Thin Gamajig
  - d. (One or more blanks)

Which of the input values in part (a) causes unpredictable results ?

### **Part B: Programs**

1. Write a C program to search a name in a list of names using Binary Search technique.
2. Write and execute a C program that implement string copy operation STRCOPY (str1, str2) that copies a string str1 to another string str2 without using library function.
3. Write a C program that read a sentence and print frequency of vowels and total count of consonants.
4. Write a C program to reverse a string without using library functions.
5. Write a C program to check whether a string is palindrome or not.
6. Write a C program to copy one string to another without using strcpy library functions.

### **3. Functions**

#### **Part A : Theory**

1. What are C functions ? Explain the difference between user defined and library functions ?
2. What is function write a function to find the sum of two numbers.
3. Write a structure of C program in terms of user defined functions.
4. Explain the three elements of user defined functions with examples.
5. What are the different possible location of functions . Explain with illustrations.
6. Explain all categories of user defined functions with example.
7. Explain the concept of void and parameters less functions.
8. What is function parameter? Explain different types of parameter in C Functions.
9. Explain two categories of argument passing techniques (pass by value and pass by reference) with examples.
10. Explain the following :
  - a. Local and Global Variables
  - b. Register Variables
  - c. Extern Variables
  - d. Actual and formal arguments
  - e. Pass by value and Pass by reference
11. Explain the concept of recursion with suitable programming example.
12. Differentiate between recursion and iteration with examples.

#### **Part B: Programs**

1. Write a C function *isprime(num)* that accepts an integer argument and returns 1 if the argument is a prime or a 0 otherwise. Write a

program that invokes this function to generate prime numbers between the ranges.

2. Write a C function  $\text{RightShift}(x,n)$  that takes two integers x and n as input and returns value of the integer x rotated to the right by n positions . Assume the integers are unsigned. Write a C program that invokes this function with different values for x and n and tabulate the results with suitable headings.
3. Write a C program for the following using recursive functions :
  - a. To find the factorial of a number
  - b. To compute the binomial coefficient
  - c. To generate Fibonacci series

## Module 4: Structures and File Management

**Syllabus:** *Basics of Structures, Nested Structures, Arrays of Structures ,Structures and Function, Structure data types and type Definition, Files: Defining, Opening and Closing of Files, Input and Output Operations ,Programming Examples and Exercises*

### 4.1 : Structures

**4.1.1 Basics of Structures:** The various primitive data types that are available in C language are int, float , char , double and void . Using these primitive data types we can derive some other data types . Such data types that are derived from the basic data types are called **derived data types**. The various derived data types available in C language are: *Arrays, Pointers , Enumerations , Structures ,Unions,etc.*

**a. Definition of Structures:** A structure is a collection of one or more variables of similar or dissimilar data types, grouped together under a single name.i.e. A structure is a derived data type containing multiple variable of **homogeneous or heterogeneous data types**. The structure is defined using the keyword **struct** followed by an identifier. This identifier is called **struct\_tag**. The syntax and example is as follows:

#### Syntax for Defining Structure:

```
struct struct_tag  
{  
    type var_1;  
    type var_2;  
    -----  
    -----  
    type var_n;  
};
```

Here

- The word struct is a keyword in C

- The item identified as struct\_tag is called a tag and identifies the structure later in the program
- Each type1, -----typen can be any data type in C including structure
- The item var\_1,var\_2,----var\_n are the members of the structure.

**Example :**

```
struct student
{
    char name[20];
    int roll_number;
    float average_marks;
};
```

**b. Declaring Structure Variables:** The syntax of declaring structure variables is shown below :

```
struct struct_tag v1,v2,v3,-----vn;
```

**Example:** struct student s1,s2,s3;

The complete structure definition along with structure declaration is as shown below :

```
struct student
{
    char name[10];
    int roll_number;
    float average_marks;
};
```

```
struct student cse,ise;
```

**c. Structure Initialization:** Assigning the values to the structure member fields is known as structure initialization. The syntax of initializing structure variables is similar to that of arrays i.e. all the elements will be enclosed within braces i.e. '{' and '}' and are separated by commas . The syntax is as shown below:

```
struct struct_tag variable = { v1,v2,v3----- vn};
```

**Example :** struct student cse = { “Raju”,18, 87.5};

The complete structure definition, declaration and initialization is as shown below:

#### Structure Definition :

```
struct employee  
{ char name[20];  
    int salary;  
    int id;  
};
```

#### Structure Declaration

```
struct employee emp1,emp2;
```

#### Structure Initialization :

```
struct employee emp1 = { “ Ramu”,20000,1000};  
struct employee emp2 = { “ThyagaRaju”,10050,200};
```

#### d. Accessing Members of Structures ( Using The Dot (.) Operator ) :

The member of structure is identified and accessed using the dot operator (.). The dot operator connects the member name to the name of its containing structure. The general syntax is :

```
struct_variable.membername;
```

**Example :** Consider the structure definition and initialization as shown below:

```
struct employee  
{    char name[10];  
    float salary ;  
    int id ;  
};
```

```
struct employee E1 = { "RAMA",10950.0,2001};
```

**The members of structure variable E1 can be accessed using dot operator as illustrated below :**

**E1.name**      can access the string “RAMA”

**E1.salary**      can access the value 10950

**E1.id**      can access the value of 2001

**Displaying the various members of Structures:** The values of members of structure variable can be displayed using the **printf** and **dot operator** as illustrated below:

**printf(“%s\n”,E1.name);**      prints the name of the employee

**printf(“%f\n”,E1.salary);**      prints the salary

**printf(“%d\n”,E1.id);**      prints the employee id

**Reading the values for various members of structures :** The values of members of structure variables can be read using **scanf or gets or getc and dot operator** as illustrated below :

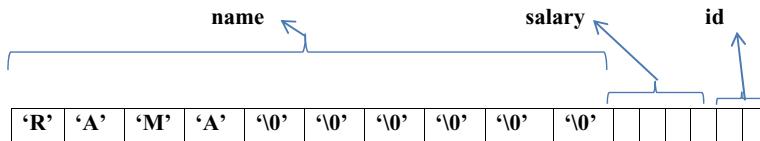
We can read the name of the employee , salary and id as shown below :

```
gets(E1.name);           reads the employee name  
scanf("%f",&E1.salary);  reads the salary of employee  
scanf("%d",&E1.id);      reads the employee id
```

#### e. Memory Representation of Structure:

When structure is declared the sum of memory required for each members will get allocated contiguously. For the structure employee the total amount of memory of 16 Bytes will get allocated. (10 Bytes for name , 4 Bytes for Salary and 2 Bytes for id.)

The memory representation for the structure employee E1 is shown below :



**Programming Example:** To Read and Write a structure called **Car** with members like name, model and Price.

```
#include<stdio.h>  
#include<conio.h>  
struct Car  
{  
    char name[10];  
    int Model;  
    float price;  
};  
  
void main()  
{  
    struct Car c1;
```

```

clrscr();
printf("\n Enter Car name :");
gets(c1.name);
printf("\n Enter Model No:");
scanf("%d",&c1.Model)
printf("\n Enter price of the Car in lakhs:");
scanf("%f",&c1.price);
printf("\n Car name is %s:",c1.name);
printf("\n Car Model is %d:",c1.Model);
printf("\n Car price is :%f lakhs",c1.price);
getch();
}

```

### Output :

```

Enter Car name : CRETA
Enter Model No: 2015
Enter price of the Car in lakhs: 9.58
Car name is :CRETA
Car Model is : 2015
Car price is :9.580000 lakhs

```

### 4.1.2 Nested Structures :

A structure can be a member of another structure. This is called a nested structure .i.e. A structure which includes another structure is called nested structure.

**Example:** Consider the structure **date** which includes the details like *date, month and year*

**struct date**

```

{
    int dd;
    int mm;
    int yyyy;
};
```

The above **structure date** can be used in the structure say **employee** as follows:

```
struct Employee
{
    char empname[20];
    int empid;
    float salary;
    struct date date_of_joining;
}
```

In the above example the *date structure is nested within employee structure*. In order to access the member of date structure we need to first create a variable of employee structure.

```
struct Employee E1;
```

**Now we access member of date structure as follows:**

```
E1.date_of_joining.dd ;
E1.date_of_joining.mm;
E1.date_of_joining.yyyy;
```

**Programming Example for Nested Structure :**

```
#include<stdio.h>
#include<conio.h>
struct date
{
    int d;
    int m;
    int y;
};

struct employee
{
    char empname[20];
    int empid;
    float salary;
    struct date doj;
};
```

```

void main()
{
    struct employee e1;
    clrscr();
    e1.doj.d= 10;
    e2.doj.m=1;
    e1.doj.y=2015;
    printf("\n Date of Joining of employee is %d\n",e1.doj.d);
    printf("\n Month of Joining of employee is %d\n",e1.doj.m);
    printf("\n Year of joining of employee is %d\n", e1.doj.y);
    getch();
}

```

#### Output:

Date of Joining of employee is 10  
Month of Joining of employee is 1  
Year of Joining of employee is 2015

### 4.1.3 Array of Structures:

In C language as we can have an array of integers we can also have an array of structures. For example suppose we want to store the information of 10 students consisting of name and usn. The structure definition can be written as shown below :

```

struct student
{
    char name[20];
    int usn;
};

```

To store the information of **10 students** the structure variable has to be declared as follows:

```
struct student s[10];
```

The general syntax for structure array declaration is :

```
struct struct_tag variable[size];
```

**Programming Example:** Program to store and display the information of **10 students** using arrays of structures is as illustrated below :

```
#include<stdio.h>
#include<conio.h>
struct student
{
    char name[50];
    int usn;
};
void main()
{
    struct student s[10];
    int i;
    clrscr();
    for(i=0;i<10;i++)
    {
        printf("\nEnter student %d name :", i+1);
        scanf("%s",s[i].name);
        printf("\nEnter student %d USN :", i+1);
        scanf("%d",&s[i].usn);
    }
    for(i =0;i<10;i++)
    {
        printf("%d student name is %s\n",i+1,s[i].name);
        printf("%d student USN is : %d\n",i+1,s[i].usn);
    }
    getch();
}
```

**4.1.4. Structure and Functions:** Just like the way normal variables can be passed to the functions it is possible to pass even structure variables to functions. Structure variable can be passed to functions using two techniques:

- 1. Pass by value**
- 2. Pass by Reference**

1. **Passing structure by value:** A structure variable can be passed to the function as an argument just like normal variables are passed to functions. Consider the following example :

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

void disp(struct student stu);
struct student
{
    char name[50];
    int usn;
};

void main()
{
    struct student s1;
    clrscr();

    printf("\nEnter student name \n");
    scanf("%s",s1.name);
    printf("\nEnter student usn\n");
    scanf("%d",&s1.usn);
    disp(s1);
    getch();
}

void disp(struct student stu)
{
    printf("\nName is %s\n",stu.name);
    printf("\n USN is %d\n",stu.usn);
}
```

As illustrated in the above example the structure variable **s1** is passed as an argument of **disp(s1)** function. The value of the members variables are displayed in **disp(struct student stu)** function.

2. **Passing Structures by Reference:** It is also possible to pass structures by references just like normal variables are passed by references. In case of pass by reference the address of the structure is passed . When the address is passed , pointer should collect the address .

To access the member of a structure , *arrow operator* ( $\rightarrow$ ) should be used. If structure is passed by reference , change made in structure variable in function definition reflects in original structure variable in the calling function as illustrated in the example below :

```
#include<stdio.h>
#include<conio.h>
void display (struct student *stu);
struct student
{
    char name[50];
    int usn;
};
void main()
{
    struct student s1;
    clrscr();
    printf("\nEnter the student name:\n");
    scanf("%s",s1.name);
    printf("\nEnter USN:");
    scanf("%d",&s1.usn);
    disp(&s1);
    printf("\n In main Function\n");
    printf("\n Name is %s \n",s1.name);
    printf("\n USN is : %d\n",s1.usn);
    getch();
}
void disp(struct student *stu)
{
    stu->usn=30;
    stu->name[ ] = "Dhoni";
    printf("\n In Display Function\n");
    printf("Name is : %s\n",stu->name);
    printf("USN is : %d\n",stu->usn);
}
```

The output of the above program will be :

Enter student name :Thyagu

Enter USN : 40

In Display Function

Name is :Dhoni

USN is : 30

In main Function

Name is :Dhoni

USN is :30

From the above output , it can be noticed that , when the name and usn are changed in the disp( ) function , the changes are reflected even in the main( ) function.

**4.1.5 Type definition:** The *typedef* is a keyword using which the programmer can create a new data type name for an already existing data type name. So the purpose of *typedef* is to redefine or rename the name of an existing data type.

**Syntax :** `typedef data_type newname1,newname2,-----newnamen;`

**Example 1:** `typedef int PERCENTAGE ,USN;`

**Example 2:** `typedef float SALARY;`

The new names that are given by the user for the already existing data types are called *user defined data types*. In the above example **PERCENTAGE, USN and SALARY** are *user defined data types*. The user defined data types can be used to declare variables as illustrated below:

**Example 1:**

```
typedef int PERCENTAGE , USN;  
PERCENTAGE p1,p2,p3;  
USN u1,u2,u3;
```

**Example 2:**

```
typdef int ARRAY[100];  
ARRAY s;  
Here s is an integer array of size 100.
```

**Example 3 :**

```
typedef char STRING[100];  
STRING s;  
Here s is an array of characters of size 100.
```

**Programming Example:** Program to compute simple interest that uses various *typedef* definition

```
#include <stdio.h>
#include<conio.h>
typedef float PRINCIPAL_AMOUNT ;
typedef float TIME_PERIOD;
typedef float RATE_OF_INTEREST;

void main()
{
    PRINCIPAL_AMOUNT p,si;
    TIME_PERIOD t;
    RATE_OF_INTEREST r;
    clrscr();
    printf("n Enter p , t and r \n");
    scanf("%f %f %f",&p,&t,&r);
    si = (p*t*r)/100;
    printf("\n SI = %f \n",si);
}
```

**Output:**

```
Enter p,t and r
1000      2.5      1.5
SI = 37.500000
```

#### **4.1.6 Defining Structure Data types using `typedef` (*Type Defined Structure*)**

The structure defined using the keyword `typedef` is called type defined structure. The syntax of the *typedefined structure* along with example is as given below:

##### **Syntax:**

```
typedef struct
{
    data_type1 member1;
    data_type2 member2;
-----
} struct_ID ;
```

##### **Example :**

```
typedef struct
{
    char name[10];
    int USN;
    float Percentage_Marks;
} Student ;
```

Here **Student** is the new datatype using which variables can be declared like : **Student S1,S2,S3;**

**Note :** The `typedef` structure can also be defined as illustrated below :

```
struct employee
{
    char name[10];
    int Emp_id;
    float Salary;
}
```

```
typedef struct employee EMPLOYEE;
EMPLOYEE e1,e2,e3;
```

**Programming Example:** Program to simulate the addition of two complex numbers using the `typedef` structures

```

#include<stdio.h>
#include<conio.h>
typdef struct
{
    int rp;
    int ip;
} Complex;

void main ()
{
    Complex c1,c2, sc;
    clrscr();

    printf("\n Enter real and imaginary part of first complex
number:");
    scanf("%d %d",&c1.rp,&c1.ip);

    printf("\n Enter real and imaginary part of second complex
number:");
    scanf("%d %d",&c2.rp,&c2.ip);

    sc.rp = c1.rp + c2.ip;
    sc.ip = c1.ip +c2.ip

    printf("The real and imaginary part of resultant complex
number
is \n");
    printf("%d %d \n", sc.rp,sc.ip);
    getch();
}

```

### Output

```

Enter real and imaginary part of first complex number:      2   3
Enter real and imaginary part of second complex number : 4   5
The real and imaginary part of resultant complex number is
6 8

```

## 4.2 File Management

**4.2.1 Introduction:** A file is defined as an organized collection of data stored on the secondary storage device such as magnetic tape, hard disk, floppy disk etc. An *input file* (also called *data file*) contains the same items you might have typed in from the keyboard during interactive data entry. An *output file* contains the same information that might have been sent to the screen as the output from your program.

**Definition:** A file is defined as an organized collection of data stored on the secondary storage device such as magnetic tape, hard disk, floppy disk etc.

### Advantages:

1. Convenient to read input from the file than the keyboard.
2. Consumes less amount of input time.
3. One can store large amount of data in a file.
4. The data remains secured and can be used for any number of time without loss.

C handles data input-output to a file in much the same as input output from keyboard and to VDU (I/O device).

**Types of Files:** A file is a sequence of character. Depending on the contents of a file there are two types of files: 1.**Text file and 2.Binary file.** A text file stores textual information. On the other hand a binary files stores the internal (binary) representation of data.

#### 4.2.2: Creating, Opening and Closing a File

##### a. Creating a Data File :

An input **data (text)** file is created in the same way a C program file is created by typing data in the keyboard with text editor (like the Turbo editor or **vi** ). This file can then be given a name that associates it within the program. For example if the program is called **example.c**, the data file might be called **sample.txt** as given below :

**example.c**

```
#include<stdio.h>
#include<conio.h>
main()
{ int i,n=10;
 clrscr();
 for(i=0;i<=n;i++)
 printf("%d",n);
}
```

**sample.txt**

|   |                |
|---|----------------|
| 1 | ArunSrinivasan |
| 2 | Thyagaraju G   |
| 3 | Kumbadrona     |
| 4 | EkaRaja        |
| 5 | Sampath        |
| 6 | Palguni Gowda  |
| 7 | Sundaraju      |

**b. Declaring a File :** Whenever we want to make use of files in a program they must be declared. Every file must be associated with a pointer called file pointer.

A **file pointer** variable **fp** should be declared as a pointer to a structure type **FILE** as shown below :

```
FILE *fp ;
```

**FILE** is the derived data type which is defined already in header file **stdio.h** as a structure. The file pointer **fp** can be used either as a local or global variables. The file pointer **fp** can be used to **open a file, update a file and to close a file in sequence.**

**Modes of a File :** The three fundamental modes in which a file can be opened or created successfully are as given below :

**r (read mode)—open a text file for reading purpose only**  
**w(write mode)--- create text file for writing purpose only**  
**a append mode)—append to the existing text file**

The different modes of opening a File along with their descriptions are listed below:

**Table 4.2.1 : Different modes of File opening**

| <b>Sl.No</b> | <b>Mode</b> | <b>Description</b>                                                                                                                                                               |
|--------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1            | w           | Creates a new text file for writing purpose . If the file already exists then its data is deleted .                                                                              |
| 2            | r           | Opens a text file in read only mode. The file must exist .                                                                                                                       |
| 3            | a           | Opens a text file to append the new data. If the file does not exist then the file is created                                                                                    |
| 4            | w+          | Creates a text file to perform read and write operation . If the file already exits then its data is deleted.                                                                    |
| 5            | r+          | Opens a text file to perform read and write operations. The file must exist.                                                                                                     |
| 6            | a+          | Opens a text file to perform read and append operations. If the file does not exist then it is created and if it exists then the new data is appended at the end of the file.    |
| 7            | wb          | Creates a binary file. If the file already exists then its data is deleted.                                                                                                      |
| 8            | rb          | Opens a binary file in read only mode. The file must exist.                                                                                                                      |
| 9            | ab          | Opens a binary file to append the new data. If the file does not exist then the file is created.                                                                                 |
| 10           | wb+         | Creates a binary file to perform read and write operations . If the file already exists then its data is deleted.                                                                |
| 11           | rb+         | Opens a binary file to perform read and write operations. The file must exist.                                                                                                   |
| 12           | ab+         | Opens a binary file to perform read and append operations. If the file does not exists then it is created and if it exists then the new data is appended at the end of the file. |

**c. Opening a File:** Opening a file is done by a call to the function **fopen()** which tells the operating system the name of the file and whether the file is to be opened for reading or for writing.

**General form of call to fopen :**

```
filepointer = fopen ("Filename", "mode");
```

The function **fopen** takes two parameters both of which are strings. The parameter filename is the name given to the file to be opened. If the file is not in the default directory a **full path name** must be provided. Parameter mode defines the way the file is to be opened ; the most common modes are shown below . The mode is a string of 1 or 2 characters as explained below and enclosed in double quotes. The different modes supported in C language are listed in **table 4.2.1.**

**Example:** Creating and opening three different files in different modes

```
FILE *fp1,*fp2,*fp3;
```

```
fp1 = fopen("student.txt","r");
fp2 =fopen("One.txt","w");
fp3 = fopen("newfile.txt","a");
```

**Note :**

1. One must include the complete path of the file if the file is not stored in the default directory as follows:  

```
fp1= fopen("C:\\foldername\\xyz.txt","r");
```
2. The data file may be named with extension **.dat or .txt**

**d. Closing a File :** After file has been used it must be closed. This is done by a call to the function **fclose()**. The **fclose()** function breaks the

connection between the stream and the file. General form of a call to the *fclose()* is

```
fclose(fp);
```

The function **fclose** takes one parameter which is pointer to a file. If there is an error , such as trying to close a file that is not opened, the function return EOF; otherwise it returns 0. The return value is usually not checked

#### **Example :**

```
File *fp1,*fp2 ;  
fclose(fp1) ;  
fclose(fp2);
```

### **4.2.3 Input/Output operations on Files**

The input/output operations can be performed on files using following three types of I/O functions to read and write to the file:

- 1.fscanf() and fprintf()
- 2.fgets() and fputs()
- 3.fgetc() and fputc()

#### **1. Input/Output using fscanf() and fprintf()**

- a. **fscanf()** : This function can be used to read data from the file . The syntax of fscanf () is given below :

```
fscanf(fp, "formatstring",list);
```

**where**

- **fp is a file pointer which can point** to a source file or standard input **stdin**. If **fp** is pointing to **stdin** , data is read from the keyboard. If fp points to any given source file , the data is read from the given source file.

- format string is the list of conversion specifiers which we have seen in scanf() and printf() statements..
- list is the list of variables with address operator.
- The function returns EOF when it attempts to read after the end of the file is reached; otherwise it returns the number of items read in.

**Example 1:** `fscanf(fp, "%d %s %f", &emp_id, emp_name, &salary);`  
`fscanf(fp, "%s", str);`

**Example 2:** Consider the following statements

`fscanf(stdin, "%d %s %f", &emp_id, emp_name, &salary);`

The above statement is same as the following scanf function :

`scanf("%d %s %f", &emp_id, emp_name, &salary);`

- b. fprintf()** : This function can be used to *write data into* the file i.e to print the data in a given output file. The syntax of fprintf() is given below :

`fprintf(fp, "formatstring", list_of_variables);`

**Where**

- **fp is a file pointer which can point** to a source file
- *format* string is the list of conversion specifiers

**Example1 : Consider the following statement:**

`fprintf(fp, "%d %s %f", emp_id, emp_name, salary);`

After executing *fprintf* the value of the variables **emp\_id, emp\_name and salary** are written into the file referred by file pointer **fp**.

**Example 2 : Consider the following statement :**

`fprintf(stdout, "%d%s%f", emp_id, emp_name, salary);`

The above statement is same as the following printf statement

```
printf("%d %s %f",emp_id,emp_name,salary);
```

**Example : C Program to write , read and display the contents of file using printf and fscanf functions.**

```
#include<stdio.h>
#include<conio.h>
struct emp
{
    char name[10];
    int age;
};
void main()
{
    struct emp e;
    FILE *fp;
    clrscr();
    fp = fopen("employee.txt", "w");
    printf("\nEnter Name and Age\n");
    scanf("%s %d", e.name, &e.age);
    fprintf(fp,"%s %d", e.name, e.age);
    fclose(p);
    fp = fopen("employee.txt", "r");
    do
    { fscanf(fp,"%s %d ", e.name, e.age);
        printf("%s %d", e.name, e.age);
    }while( !feof(fp) );
    getch();
}
```

### Output :

*The program creates a new empty file called **employee.txt** with write mode.*

Enter Name and Age

Raju 40

*The program writes a data into the file **employee.txt***

The output in the file employee.txt will be :

Raju 40

The output on the display screen will be :

Raju 40

## 2. Input output Operations using fgets and fputs

a. **fgets()** : This function reads string of characters from a file referred by the file pointer fp . The syntax of fgets() is

**fgets(str,n,fp);**

**where**

**fp** is file pointer

**str** is an array of characters

**n** represents the number of characters to be read from the file

**Example :** The following program uses fgets() to display the content of a file.

```
#include<stdlib.h>
#include <stdio.h>
int main()
{
    FILE *fp;
    char text[100];
    clrscr();
    if((fptr = fopen("test.txt","r"))==NULL)
    {
        printf("Cannot open file\n");
        exit(1);
    }
    fgets(text,100,fp);
    puts(text);
    if(fclose(fp))
        printf("File close error\n");
    getch();
    return 0;
}
```

**fputs()** : This function is used to print a string of characters into a file referred by the file pointer fp . The fputs() function returns non negative on success and EOF on failure. The syntax of fputs() is:

**fputs(str,fp) :**

### **Where**

- fp is a file pointer .
- str is a string variable

The fputs() function writes the content of the string pointed to by str to the specified stream.

**Example:** The following program uses fputs() to write a string into a file.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    FILE *fp;
    char text[100];
    int i = 0;
    clrscr();
    printf("Enter a text:\n");
    gets(text);
    if((fp = fopen("test.txt","w"))==NULL)
    {
        printf("Cannot open file\n");
        exit(1);
    }
    fputs(text,fp);
    if(fclose(fp))
        printf("File close error\n");
    getch();
    return 0;
}
```

### **3. Input/Output operations using fgetc() and fputc()**

The functions **fgetc( )** and **fputc( )** are similar to the functions **getc( )** and **putc( )** respectively but they are used for processing data stored in files.

The **fgetc( )** function is used to read a single character at a time from a given file referred by file pointer fp.The **fputc( )** function is used to print

/write a single character at a time to the file referred by fp. The syntax of **fgetc()** and **fputc()** functions are shown below :

```
ch = fgetc(fp);  
fputc(ch,fp);
```

If there is any error or end of file is encountered the function returns EOF

**Example :** The following example shows the usage of fputc() function.

```
#include <stdio.h>  
#include<conio.h>  
int main ()  
{  
    FILE *fp;  
    int ch;  
    clrscr();  
    fp = fopen("ascii.txt", "w+");  
    for( ch = 65 ; ch <= 90; ch++ )  
    {  
        fputc(ch, fp);  
    }  
    fclose(fp);  
    getch();  
    return(0);  
}
```

After executing the above program it will create a file **ascii.txt** in the current directory, which will have following content:

```
ABCDEFGHIJKLMNPQRSTUVWXYZ
```

**Example :** The following example shows the usage of fgetc() function. The program makes use of the file **ascii.txt** which was created in the previous example.

```
#include <stdio.h>  
#include<conio.h>  
int main ()  
{  
    FILE *fp;  
    int c;  
    clrscr();
```

```
fp = fopen("file.txt","r");
while(1)
{
    c = fgetc(fp);
    if( feof(fp) )
    {
        break ;
    }
    printf("%c", c);
}
fclose(fp);
getch();
return(0);
}
```

**Output :**

```
ABCDEFGHIJKLMNPQRSTUVWXYZ
```

## Difference between Arrays and Structures

| SI.NO | ARRAYS                                                                                       | STRUCTURES                                                                                                                |
|-------|----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| 1     | It is a collection of homogeneous elements i.e All elements of same data type                | It is a collection of homogeneous or heterogeneous elements i.e Members of a structure can belong to different data types |
| 2     | Array allocates static memory and uses index / subscript for accessing elements of the array | Structures allocate dynamic memory and uses (.) operator for accessing the member of a structure.                         |
| 3     | Array is a pointer to the first element of it                                                | Structure is not a pointer                                                                                                |
| 4     | Arrays is always passed/ called by reference                                                 | Structure variables are passed / called by value by default to a function                                                 |
| 5     | <b>Example :</b><br><pre>int a[10]; char x[10];</pre>                                        | <b>Example :</b><br><pre>struct complex {     float rp;     int ip; };</pre>                                              |
| 6     | Array element access takes less time                                                         | Takes more time in comparison to structure                                                                                |
| 7     | An array is a derived data type                                                              | A structure is a derived and programmer-defined data type                                                                 |

## Additional Concepts

**Unions :** A **union** is a special data type that allows to store different data types in the same memory location. One can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

**Defining a Union :** Union is defined similarly as structures using a keyword union followed by tag name. The syntax used for defining union is given below :

```
union union_tag {  
    type member1;  
    type member2;  
-----  
    type member n;  
}variables list ;
```

**Example :**

```
union VALUE {  
    int i;  
    float f;  
    char str[10];  
} v1,v2;
```

Now, a variables of **VALUE** type can store an integer, a floating-point number, or a string of characters. It means a single variable, i.e., same memory location, can be used to store multiple types of data.

The memory occupied by a union will be large enough to hold the largest member of the union. For example, in the above example, **VALUE** type will occupy 10 bytes of memory space because this is the maximum space which can be occupied by a character string as compared to integer (2 Bytes) and float (4 Bytes).

**Example :** The following example displays the total memory size occupied by the above union.

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

union VALUE {
    int i;
    float f;
    char str[10];
};

int main( ) {
    union VALUE v1;
    printf( "Memory size occupied by v1 : %d\n", sizeof(v1));
    return 0;
}
```

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

```
Memory size occupied by data : 10
```

**Accessing Union Members** : To access any member of a union, the **member access operator (.) is used**. The following programming example shows how to use unions in a program –

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

union VALUE {
    int i;
    float f;
    char str[10];
};

int main( ) {
    union VALUE v1;
    v1.i = 100;
```

```

v1.f = 2.5;
strcpy( v1.str, "Hello C");

printf( "data.i : %d\n", data.i);
printf( "data.f : %f\n", data.f);
printf( "data.str : %s\n", data.str);

return 0;
}

```

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

|                                               |                    |
|-----------------------------------------------|--------------------|
| v1.i : 1917853763                             | (corrupted value ) |
| v1.f : 4122360580327794860452759994368.000000 | (corrupted value)  |
| v.str : Hello C                               | (correct value)    |

Here, we can see that the values of **i** and **f** members of union got corrupted because the final value assigned to the variable has occupied the memory location and this is the reason that the value of **str** member is getting printed accurately.

Now let's look into the same example once again where we will use one variable at a time which is the main purpose of having unions –

```

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

union Data {
    int i;
    float f;
    char str[10];
};

int main( ) {

    union Data data;

    data.i = 100;
    printf( "data.i : %d\n", data.i);

    data.f = 2.5;
    printf( "data.f : %f\n", data.f);
}

```

```
strcpy( data.str, "Hello C");
printf( "data.str : %s\n", data.str);

return 0;
}
```

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

```
data.i : 100
data.f : 2.5
data.str : Hello C
```

Here, all the members are getting printed very well because one member is being used at a time.

## Difference between structure and union

| <b>Sl.<br/>NO</b> | <b>Structure</b>                                                                                                                                        | <b>Union</b>                                                                                                                                      |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>1</b>          | It allocates memory equal to sum of memory allocated to its each individual member.                                                                     | It allocates piece of memory that is Large enough to hold the Largest variable of type in union.                                                  |
| <b>2</b>          | Each member have their own memory space                                                                                                                 | One block is used by all the members of union.                                                                                                    |
| <b>3</b>          | Structure can not be implemented in shared memory                                                                                                       | Union is the Best environment where memory is shared.                                                                                             |
| <b>4</b>          | It has less Ambiguity                                                                                                                                   | As memory is shared, Ambiguity is more in union.                                                                                                  |
| <b>5</b>          | Self referential structure can be implemented in data structure.                                                                                        | Self referential union can not be implemented.                                                                                                    |
| <b>6</b>          | All members of structure can be accessed at a time                                                                                                      | Only one member is accessed at a time.                                                                                                            |
| <b>7</b>          | Example structure declaration:<br><br><pre>struct sVALUE {     int ival;     float fval;     char *ptr; }s;</pre> <p>Here size of struct is 8 Bytes</p> | Example union declaration:<br><br><pre>union uVALUE {     int ival;     float fval;     char *ptr; }u;</pre> <p>Here size of union is 4 Bytes</p> |
| <b>8</b>          | Size of structure is equal to the sum of the size of member variables.                                                                                  | Size of union is equal to the size of the large sized member variable.                                                                            |

## 4.3 Programming Examples:

### 4.3.1 Programming examples based on Structures

#### 1. Write a C program to store Name, USN, subject name and IA marks of student using structure.

```
#include<stdio.h>
#include<conio.h>
struct student
{
    char name[20];
    char usn[20];
    char subname[20];
    int iamarks;
};

void main()
{
    struct student s[50];
    int i,n;
    clrscr();
    printf("\n Enter the value of n \n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\n Enter student %d name :", i+1);
        scanf("%s",s[i].name);
        printf("\n Enter student %d USN :", i+1);
        scanf("%s",s[i].usn);
        printf("\n Enter student %d subject name :",i+1);
        scanf("%s",s[i].subname);
        printf("\n Enter student %d IA Marks :",i+1);
        scanf("%d",&s[i].iamarks);

    }
    printf("\n The details of %d students stored \n",n);
    for(i =0;i<n;i++)
    {
        printf("%d student name is %s\n",i+1,s[i].name);
        printf("%d student USN is : %s\n",i+1,s[i].usn);
        printf("%d student Subject Name is :
%s\n",i+1,s[i].subname);
```

```
printf("%d student IA Marks is : %d\n",i+1,s[i].iamarks);
```

```
}
```

```
getch();
```

```
}
```

- 2. To read and display a structure called student with the member name, usn, address, branch and age.**

```
#include<stdio.h>
#include<conio.h>
struct student
{
    char name[50];
    int usn;
    char address[100];
    char branch[20];
    int age;
};

void main()
{
    struct student s;
    clrscr();
    printf("\nEnter student name :");
    scanf("%s",s.name);
    printf("\nEnter student USN :");
    scanf("%d",&s.usn);
    printf("\nEnter student address :");
    scanf("%s",s.address);
    printf("\nEnter student branch :");
    scanf("%s",s.branch);
    printf("\nEnter student age:");
    scanf("%d",&s.age);
    printf("\nThe entered student information is \n");
    printf("\n Name: %s\n", s.name);
    printf("\n USN:%s\n",s.usn);
    printf("\n Address :%s\n",s.address);
    printf("\n Branch :%s\n", s.branch);
    printf("\n Age :%d \n", s.age);
    getch();
}
```

**Output :**

```
Enter student name : Rajiv
Enter student USN:12CS23
Enter student address: #23,Gandhinagar,Ujire.
Enter student branch:CSE
Enter student age :23
```

```
The entered student information is
Name : Rajiv
USN:12CS23
Address: #23,Gandhinagar,Ujire.
Branch:CSE
Age:3
```

### 3. Program to display the data up to 50 books in a library using structure.

```
#include<stdio.h>
#include<conio.h>
    struct lib
    {
        char bookname[50];
        char author_name[30];
        int book_no;
        char issue_date[10];
        char due_date[10];
    };

void main()
{
    struct lib book[50];
    int n,i;
    clrscr();
    printf("\n Enter the number of books \n");
    scanf("%d",&n);
    printf("\n Enter the data of \n");
    for(i=0;i<n ;i++)
    {
        printf(" Book Serial Number %d\n",i+1);
        printf("\nBookName :");
        scanf("%s",book[i].bookname);
        printf("\nAuthorName :");
        scanf("%s",book[i].author_name);
        printf("\nBookNumber :");
```

```

        scanf("%d",&book[i].book_no);
        printf("\nIssue Date :");
        scanf("%s",book[i].issue_date);
        printf("\n Due Date :");
        scanf("%s",due_date);
    }
    printf("\n The entered records are \n");

    for(i=0;i<=n-1;i++)
    {
        printf("\n Book Serial No: %d\n",i+1);
        printf("\n Book Name :%s",book[i].bookname);
        printf("\n Author Name :%s",book[i].author_name);
        printf("\n Book Number :%s",book[i].book_no);
        printf("\n Issue Date : %s ", book[i].issue_date);
        printf("\n Due Date :%s\n", book[i].due_date);
    }
    getch();
}

```

**4. Program to add two complex numbers using function and structure variables.**

```

#include <stdio.h>
#include<conio.h>
struct complex
{
    float real;
    float imag;
} complex;
complex add(complex n1,complex n2);

main()
{
    complex n1,n2,temp;
    clrscr();
    printf("Enter real and imaginary part of first complex
no:\n");
    scanf("%f%f",&n1.real,&n1.imag);
    printf("Enter real and imaginary part of second complex
no:\n");
    scanf("%f%f",&n2.real,&n2.imag);
    temp = add(n1,n2);
    printf("Sum=%f+i*%f",temp.real,temp.imag);
    getch();
}

```

```

}
complex add(complex n1,complex n2)
{
    complex temp;
    temp.real = n1.real + n2.real;
    temp.imag = n1.imag + n2.imag;
    return(temp);
}

```

#### Output

Enter real and imaginary part of first complex no:

1.5

3.5

Enter real and imaginary part of second complex no:

1.4

7.0

sum = 2.900000 + i\*10.500000

5. Write a C program to create a structure *student*, containing *name* and *usn*. Ask user the name and usn of a student in main function. Pass this structure to a function and display the information in that function.

```

#include<stdio.h>
struct student
{
    char name[50];
    int usn;
};

void display(struct student stu);
/* function prototype should be below to the structure declaration otherwise compiler shows
error */

int main()
{
    struct student s1;
    printf("Enter student's name: ");
    scanf("%s",&s1.name);
    printf("Enter usn:");
    scanf("%d",&s1.usn);
    display(s1); // passing structure variable s1 as argument
    return0;
}

```

```

void display(struct student stu)
{
    printf("Output\nName: %s",stu.name);
    printf("\nRoll: %d",stu.usn);
}

```

**Output:**

Enter student name: Thyagu Gowda  
 Enter usn: 777

**Output**

Name: Thyagu Gowda  
 usn: 777

6. Write a C program to add two distances (*Feet-inch system*) entered by user making use of a structure variable (containing feet and inch ) and add( ) function . (Note: 12 inches = 1 foot)

```

#include<stdio.h>
#include<conio.h>
struct distance
{
    int   feet;
    float inch;
};

void add(struct distance d1,struct distance d2,struct distance *d3);

main()
{
    struct distance dist1, dist2, dist3;
    clrscr();
    printf("Enter the value of dist1 in terms of feet and inches \n");
    scanf("%f %d",&dist1.feet, &dist1.inch);
    printf("Enter the value of dist2 in terms of feet and inches \n");
    scanf("%d %f",&dist2.feet ,&dist2.inch);
    add(dist1, dist2,&dist3);

    /*passing structure variables dist1 and dist2 by value whereas passing structure variable dist3 by
    reference */

    printf("\nSum of distances = %.1f",dist3.feet,
    dist3.inch);
    getch();
}

```

```
void add(struct distance d1,struct distance d2,struct distance *d3)
{
    /* Adding distances d1 and d2 and storing it in d3 */
    d3->feet=d1.feet+d2.feet;
    d3->inch=d1.inch+d2.inch;
    if(d3->inch>=12)
        /* if inch is greater or equal to 12, converting it to feet.*/
        d3->inch-=12;
        ++d3->feet;
}
}
```

#### Output:

```
First distance
Enter feet: 12
Enter inch: 6.8
Second distance
Enter feet: 5
Enter inch: 7.5

Sum of distances = 18'-2.3"
```

#### 4.3.2 Programming Examples based on Files

1. Write a C program to read and display a text from the file.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
    FILE *fp1,*fp2;
    char ch;
    clrscr();
    fp1= fopen("Input.txt","r");
    if(fp1==NULL)
    {
        printf("\n Input File is not found\n");
    }
}
```

```

        exit(0);
    }
    fp2= fopen("Output.txt.", "w");
    while((ch=fgetc(fp1))!=EOF)
    {
        fputc(ch,fp2);
        fprintf(stdout,"%c",ch);
    }
    fclose(fp1);
    fclose(fp2);
}

```

**2. Program to read n numbers from keyboard and write into a file *Input.txt* .**

```

#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    int num;
    clrscr();
    fp = fopen("Input.txt", "w");
    if(fp==NULL)
    {
        printf("Error in opening the file\n");
        exit(0);
    }
    printf("\n Enter the list of numbers and press Cntrl + Z\n");
    while(scanf("%d",&num)!=EOF)
    {
        fprintf(fp, "%d ",num);
    }
    fclose(fp);
    getch();
}

```

**Output :**

Enter the list of numbers and press Cntrl + Z

1 2 3 4

The contents of the file Input.txt includes the following :

1 2 3 4

- 3. Program to read n numbers from the input file “*Input.txt*” and display it on the screen.**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    int num;
    clrscr();
    fp = fopen("Input.txt","r");
    if(fp==NULL)
    {
        printf("Error in opening the file \n");
        exit(0);
    }
    printf("\n The contents of the file Input.txt is \n");
    while(fscanf(fp,"%d",&num)!=EOF)
    {
        printf("%d\n",num);
    }
    fclose(fp);
    getch();
}
```

**Output :**

The contents of the file Input.txt is

1  
2  
3  
4

- 4. Program to open a file named *std.txt* and store in the following data :**

| Name    | Roll NO |
|---------|---------|
| Anand   | 1       |
| Beerbal | 2       |
| Chandan | 3 .     |

**Extend the program to read the data from the file *std.txt* and display the data on the screen.**

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

```

#include<conio.h>
main()
{
    FILE *fp;
    char name[20];
    int rollno;
    clrscr();
    printf("\n Input File name\n");
    scanf("%s",filename);
    fp = fopen("filename","w");
    printf("\n Input Name and Roll number \n");
    for( i=1;i<=3;i++)
    {
        scanf("%s %d",name,&rollno);
        fprintf(fp,"%s %d\n",name,rollno);
    }
    fclose(fp);
    printf("\n");
    fp = fopen("filename","r");
    for( i=1;i<=3;i++)
    {
        fscanf(fp,"%s %d\n",name,&rollno);
        fprintf(stdout,"%s %d\n",name,rollno);
    }
    fclose(fp);
    getch()
}

```

#### Output :

Input File name  
std.txt

Input Name and Roll Number

**Anand** 1  
**Beerbal** 2  
**Chandan** 3

The contents o f the file **std.txt** includes the following :

**Anand** 1  
**Beerbal** 2  
**Chandan** 3

5. C program to read a line (of maximum length 30 ) from the keyboard using the function *fgets()*.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    char str[50];
    char *lp;
    clrscr();
    printf("\n Enter the string \n");
    lp = fgets(str,30,stdin);

    if(lp!=NULL)
    {
        printf("\n The entered string is :");
        puts(str);
        return;
    }
    else printf("Reading is unsuccessful\n");
}

```

**Output:**

Enter the string  
My name is Shaneshwara  
The entered string is : My name is Shaneshwara

**6. C program to read a line (of maximum length 30 ) from the file using the function *fgets()* .**

```

#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    char str[50];
    char *ps;
    clrscr();
    fp = fopen("Input.txt","r");
    if(fp==NULL)
    {
        printf(" Open unsuccessful \n");
        exit(0);
    }
    ps = fgets(str,30,fp);

```

```

        if(ps!=NULL)
        {
            printf("The String is :");
            puts(str);
            exit(0);

        }
        else printf("Reading is unsuccessful\n");
        getch();
    }
}

```

**Output:**

Before running the program one has to create a file called **Input.txt** and enter the following line :

**Welcome to the world of Files**

Run the program :- The output will be

The String is

**Welcome to the world of Files**

7. C Program to read the contents of entire file (of maximum 100 characters size) using **fgets()**.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    char str[100];
    clrscr();
    fp = fopen("sample.txt","r");
    fgets(str,100,fp);
    printf("%s\n",str);
    fclose(fp);
    getch();
}

```

**Output:**

Before running the program one has to create a file called **sample.txt** and enter the characters of maximum size 100.

Then run the program , the entire contents of the file **sample.txt** will be displayed on the screen.

**8. C program to read and write using fgets and fputs respectively and using stdin (keyboard input) and stdout(display screen).**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char str[50];
    char *ps;
    clrscr();
    printf("\nEnter characters till EOF\n");
    ps = fgets(str,40,stdin);
    while(ps!=NULL)
    {
        fputs(str,stdout);
        ps = fgets ( str,40,stdin);
    }
    getch();
}
```

**Output:**

**Enter characters till EOF**  
**abcdefg (Press Cntrl +Z)**  
**abcdefg**

**9. C program to read the content of file using fgetc().**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp = fopen("ex.txt","r");
    char ch = fgetc(fp);
    clrscr();
    while(ch!=EOF)
    {
        printf("%c",ch);
        ch= fgetc(fp);
    }
    fclose(fp);
    getch();
}
```

Before running the program one has to create a file called **ex.txt** and enter the following (any) characters :  
**ABCDEFGHIJKLMNPQRSTUVWXYZ**

**Run the Program :**

**Output:**

ABCDEFIGHIJKLMNOPQRSTUVWXYZ

**10.C program to write the contents (characters from A to Z) into the file using fputc().**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp = fopen("sample.txt","w");
    clrscr()
    for(ch = 65;ch<=90;ch++)
    {
        fputc(ch,fp);
    }
    fclose(fp);
    getch();
}
```

**Output:**

The contents of the file sample.txt will be

ABCDEFIGHIJKLMNOPQRSTUVWXYZ

**11.C program to copy the contents of one file into another using fgetc() and fputc().**

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
    FILE *fp1,*fp2;
    char ch;
    clrscr();
    fp1= fopen("file1.txt","r");
    fp2= fopen("file2.txt","w");
    if((fp1==NULL)&&(fp2==NULL))
    {
        printf("\n Cannot be Copied \n");
        exit(0);
    }
    while((ch=fgetc(fp1))!=EOF)
        fputc(ch,fp2);
```

```
    fclose(fp1);
    fclose(fp2);
    getch();
}
```

**Output:**

The contents of the file **file1.txt** will be copied into the file **file2.txt**

## 12. C Program to concatenate two input files.

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

void main()
{
    FILE *fp1;
    FILE *fp2;
    FILE *fp3;

    clrscr( );
    char ch;

    fp1= fopen("file1.txt","r");
    fp2= fopen("file2.txt","r");
    fp3 = fopen("file3.txt","a");

    if((fp1==NULL)
    {
        printf("\n Error in opening the first file\n");
        exit(0);
    }

    if((fp2==NULL)
    {
        printf("Error in opening the second file \n");
        exit(0);
    }

    if((fp3==NULL)
    {  printf("Error in opening the third file \n");
        fclose(fp1);
        fclose(fp2);
```

```

        exit(0);
    }

while(ch=fgetc(fp1)!=EOF) fputc(ch,fp3);
while((ch=fgetc(fp2))!=EOF)fputc(ch,fp3)

fclose(fp1);
fclose(fp2);
fclose(fp3);
getch();
}

```

**Output:**

The contents of the file1 will be **copied** into file3 and  
The contents of the file2 will be **appended** into file3.

- 13. C program to read the contents from a file called abc.txt , count the number of characters, blanks ,tabs ,lines and words and output the same.**

```

#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    char ch;
    int cc =0;
    int bc=0;
    int tc =0;
    int lc=0;
    int wc;
    clrscr();
    fp = fopen("abc.txt","r");

    if((fp==NULL)
    {
        printf("Error in opening the file \n");
        exit(0);
    }
    while((ch=fgetc(fp))!=EOF)
    {
        cc++;
        if (ch ==' ') bc++;

```

```

        if(ch=='\n') lc++;
        if(ch=='\t') tc++;
    }
fclose(fp);
wc = bc + lc;
printf("\n Number of characters = %d\n",cc);
printf("\n Number of tabs = %d\n",tc);
printf("\n Number of lines = %d\n", lc);
printf("\nNumber of blanks = %d\n",bc);
printf("\n Number of words count = %d\n",wc);
getch();
}

```

### **Output:**

**If the input file abc.txt contains the following contents :**

*One Two  
Three Four.*

The output will be :

Number of characters = 19  
 Number of tabs = 0  
 Number of lines =2  
 Number of blanks =3  
 Number of words count =4

## **14. C program to find the size of the file using file handling functions.**

```

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

void main( )
{
    FILE *fp;
    char ch;
    int size = 0;
    clrscr();

    fp = fopen("Sample.txt", "r");
    if (fp == NULL)
        printf("\nFile unable to open ");

```

```
else
    printf("\nFile opened ");
    fseek(fp, 0, 2); /* file pointer at the end of file */
    size = ftell(fp); /* To determine the size */
    printf("The size of given file is : %d\n", size);
    fclose(fp);
    getch();
}
```

## **4.4 EXERCISES:**

### **4.4.1 Theory**

1. What is structure? Explain with syntax and examples, the concept of structure declaration and initialization.
2. Describe how the structure members can be accessed and printed using dot operator with illustration.
3. Explain how the structure variable can be passed as a parameter to a function with example.
4. Explain how the structure variable can be passed as a parameter using pass by mechanism and pass by reference.
5. Explain the concept of array of structures with an example program.
6. Explain the concept of type defining a structure with example.
7. Explain the concept of structure within a structure (nested structure) with example.
8. Explain array of structures and structure within a structure with examples.
9. What is a file? Explain how the file open and file close functions handled in C.
10. What is a mode of a File? Explain the different modes of a file.
11. Explain how the input output operations can be performed on files using *fscanf* and *fprintf*.
12. Explain how the input output operations can be performed on files using *fgetc* and *fputc*.
13. Explain how the input output operations can be performed on files using *fgets* and *fputs*.

#### **4.4.2 Programs**

1. Write a C Program to read and write the address of a person {*House\_NO,Street\_Name,City*} using structures.
2. Write a C program to read and write the details of the students {*Name,Roll\_No, Sem , Branch , Marks*} using structure.
3. Write a C program to illustrate the creation and usage of a structure.
4. Write a C program to maintain a record of “n” students details using an array of structures with four fields ( *Roll number, Name , Marks and Grade* ) . Each field is on an appropriate data type . Print the marks of the student given student name as input.
5. Write a C program to read and display a text from the file to the display screen.
6. Given two text documentary files “Ramayana.in” and “Mahabharatha.in”. Write a C program to create a new file “Karnataka.in” that appends the content of the file “Ramayana.in” to the file “Mahabharatha.in”. Also calculate the number of words and new lines in the output file.

## Module 5: Pointers, Preprocessors and Introduction to Data Structures

**Syllabus:** Pointers and address, pointers and functions arguments (call by reference), pointers and arrays, Initialization of pointer arrays, pointers (address) arithmetic, character pointer, pointers to pointer, Dynamic memory allocations methods, Introduction to preprocessors, compiler control directives , programming examples and exercises.

**Introduction to Data Structures:** Primitive and non-primitive data types. Abstract data types, Definition and applications of Stacks, Queues, Linked lists and Trees.

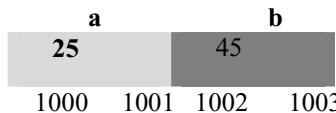
### 5.1 Pointers

**5.1.1 Pointers and address:** Consider the following declaration of two variables **a** and **b**:

```
int a = 25;
```

```
int b = 45;
```

The compiler allocates two bytes of memory starting from the address **1000** and stores the value **25** at that location and gives **a** as the name for that location. Similarly for second variable **b** as illustrated in the figure below :



The value of **a** and **b** can be accessed and printed as shown below:

```
printf("\n Value of a=%d\n",a);
printf("\n Value of b=%d\n",b);
```

In order to access the address of the variables one can use the address operator **&** as below:

&a -- -→ to get the address of variable **a**  
&b ----→ to get the address of variable **b**

The addresses of **a** and **b** can be printed as shown below:

```
printf(" Address of a = %d \n",&a);
printf("Address of b = %d\n",&b);
```

In order to access the values using the address one must use **dereferencing operator** or pointer denoted by the symbol \*. By prefixing the \*operator to the address of variable, we can access the value stored in that address as below:

```
printf("\n Value of a = %d\n",*&a);
printf("\n Value of b = %d\n",*&b);
```

**Note:** The operator \* is known as *indirection operator or dereference operator* and the operator & is known as *address operator or reference operator*.

**Programming Example:** C program to print the values using variables and their addresses.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=100;
    int b=450;

    printf("\n Value of a = %d \n",a);
    printf("\n Value of b = %d \n",b);

    /* Accessing the address of variables */

    printf("\n Address of a = %d\n",&a);
    printf("\n Address of b = %d\n",&b);

    /* Accessing the data using the dereferencing operator */

    printf("\n The value of a =%d \n", *&a);
    printf("\n Value of b = %d\n", *&b);

}
```

**Definition of Pointer:** A pointer is a variable that can hold the address of another variable or address of memory location.

The steps to be followed to use pointers:

- a. Declare a data variable                      Ex: int x;
- b. Declare a pointer variable                    Ex: int \*p;
- c. Initialize a pointer variable                Ex: p = &x;
- d. Access data using pointer variable.      Ex: y= \*p

**Pointer Declaration:** Pointer variables should be declared before they are used. The syntax to declare a pointer variable is

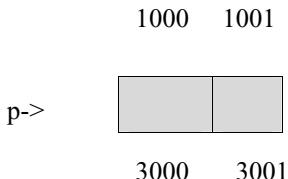
**data\_type \*identifier;**

**Example :**     int     \*pi; /\* pi is a pointer to int \*/  
                float   \*pf; /\* pf is a pointer to float \*/  
                char   \*pc; /\* pc is a pointer to character \*/  
                double   \*pd; /\* pd is a pointer to double \*/  
                FILE   \*fp; /\* fp is a pointer to FILE \*/

**Initialization of Pointer variables:** It is the process of assigning an address to the pointer variable . Consider the following declaration statements :

```
int a;  
int *p;
```

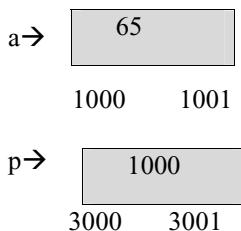
After declaration the variable **a** gets allocated in some memory location (address) say 1000 and the pointer **p** gets the address 3000 as illustrated below.



The pointer **p** can be initialized as follows:

```
a = 65;  
p = &a;
```

After initialization the variable **a** holds the value **65** and the variable **p** holds the value of address of **a** as illustrated below :



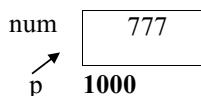
**Note :** A pointer can also be declared and initialized in the same statement as given below:

```
int x;  
int *p = &x;
```

**Accessing Variables Through Pointers :** The contents of variable can be accessed by de referencing a pointer. If a pointer **p** is pointing to a variable **num**, then its value can be accessed by just saying **\*p**. Here **\*** operator acts as the dereference operator.

**[Indirection :** The process of accessing the value of the variables indirectly by pointer pointing to the address of variable is called Indirection.]

For example consider a variable **num** whose value is **777** and the pointer pointing to its address is **p** as given in figure below:



The value 365 can be accessed in 2 methods :

- i) printf("%d",num); output : 777 using variable
- ii) printf("%d",\*p); output : 777 using pointer

**Programming Example :** Accessing the value of the variables using pointers

```
#include<stdio.h>
#include<conio.h>
main() { int a,b,v;
          int *p,*q;
          clrscr( );
          a=30;
          b=10;
          p = &a;
          q = &b;
          v = *p + *q;
          printf("\n The result is %d\n",v);
          getch();
      }
```

**Output :**

The result is 40

**Multiple Pointers:** A single variable can be accessed using multiple pointers. In this case multiple pointers has to declared and all the pointers has to be initialized to single variable. The programming example given below: illustrates how three pointers (\*p1,\*p2 and \*p3) are used to access the value of single variable **num**.

**Programming Example:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int num;
    int *p1;
    int *p2;
    int *p3;
    clrscr( );
```

```

num = 1004;
p1 = &num;
p2 = &num;
p3 = &num;

printf("\n The value of num = %d\n",num);
printf("\n The value of num = %d\n",*p1);
printf("\n The value of num = %d\n",*p2);
printf("\n The value of num = %d\n",*p3);
getch();
}

```

### Output :

The value of num = 1004  
The value of num = 1004  
The value of num = 1004  
The value of num = 1004

### Difference between a pointer variable and a normal variable :

| Sl.<br>NO | Pointer Variable                                                                                       | Normal Variable                                                                             |
|-----------|--------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| 1         | A pointer variable holds address                                                                       | A normal variable holds data                                                                |
| 2         | The general format of declaring a pointer :<br><b><i>data_type *pointer_name;</i></b>                  | The general format of declaring a variable is :<br><b><i>data_type variable_name;</i></b>   |
| 3         | The general format for initialization of pointer variable:<br><b><i>pointer_name=&amp;variable</i></b> | The general format for initialization of variable :<br><b><i>variable_name = value;</i></b> |
| 4         | variable is essential to access data.                                                                  | Data can be accessed without dereferencing a normal variable .                              |

**NULL Pointer:** A pointer pointing to nowhere in memory is called NULL Pointer. It is not the address of any object or function. Null pointers are initialized to NULL. **Example:** int \*p=NULL;

```
#include <stdio.h>
#include<conio.h>
int main()
{ int *ptr = NULL;
    clrscr();
    printf("The value of ptr is %u",ptr);
    getch();
    return 0;
}
```

#### **Output :**

The value of ptr is 0

**Dangling Pointer :** The pointer which does not contain valid address or pointing to no existing memory location is called dangling pointer. Such pointer contains garbage values. Dangling pointer arises when an object is deallocated , without modifying the value of the pointer , so that the pointer still points to the memory location of the de-allocated memory. The problem of dangling pointer can be avoided by initializing to NULL as illustrated below :

```
#include<stdlib.h>
#include<conio.h>
{
    char *ptr = malloc(Constant_Value);
    .....
    .....
    free (ptr); /* ptr now becomes a dangling pointer */
    ptr = NULL /* ptr is no more dangling pointer */
}
```

After de-allocating memory, initialize pointer to NULL so that pointer will be no longer dangling. Assigning NULL value means pointer is not pointing to any memory location

**Programming Example:** Write a C program to add two numbers using pointers

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int a=10,b=20,sum;
    int *p1,*p2;
    clrscr();
    p1 = &a;
    p2 = &b;
    sum = *p1+*p2;
    printf("\n Sum = %d\n",sum);
    getch();
}
```

**Output :**

Sum = 30

### 5.1.2 Pointers and Function Arguments (Call by Reference)

The mechanism in which pointers variable are used as function parameters is known as **call by reference**. Let us see how pointers can be passed as arguments to the function. For example consider the prototype of the swap function used to exchange the value of two variables a,b;

```
void swap(int *p1,int *p2);
```

when the function is called, **the addresses** of the variables to be modified are passed as arguments to the pointer parameters.

Thus to exchange the values of variables **a** and **b** this function is called as follows:

```
swap (&a,&b);
```

The complete program using pointer variables as an illustration for call by reference is as illustrated below :

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

```

void swap(int *p1,int *p2);
void main()
{
    int a,b;
    clrscr();
    printf("n Enter the value of a and b\n");
    scanf("%d %d",&a,&b);
    printf("n Value of a and b before exchange is %d and %d\n",a,b);
    swap(&a,&b);
    printf("n Value of a and b after exchange is %d and %d\n",a,b);
    getch();
}
void swap (int *p1,int *p2)
{
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

```

#### Output :

Enter the value of a and b  
10 20  
Value of a and b before exchange is 10 and 20  
Value of a and b after exchange is 20 and 10

**5.1.3 Pointer and arrays:** In C language the pointers can be used to create and handle arrays. Some of the operations that can be performed on arrays and its elements are:

- Traversing array using a pointer
- Accessing array element
- Reading an array
- Printing an array

**Arrays and Pointers:** The *name of the array* is actually *a pointer* representing the address of memory location of first element of array. For example consider the following declaration

**int a[5] = {10,20,30,40,50} ;**

The memory layout of the array **a** and its elements are as illustrated below:

|          |   |      |      |      |      |      |      |
|----------|---|------|------|------|------|------|------|
| <b>a</b> | → | 1000 | 1002 | 1004 | 1006 | 1008 | 1010 |
| 10       |   | 20   | 30   | 40   | 50   |      |      |
| a[0]     |   | a[1] | a[2] | a[3] | a[4] |      |      |

Here the name of the array **a** is a pointer to integer and holds the address of first element **a[0]**.

i.e **a = &a[0] = 1000**. The relationship of array name **a** (pointer) and different elements is given below :

```

a = &a[0];
a+1 = &a[1]
a+2 = &a[2]
a+3 = &a[3]
a+4 = &a[4]

```

In general **&a[i]** is same as **a+i** ; The data in those address can be obtained using the indirection operator \* as shown below :

\*&a[i]      or      \*(a+i)      or      \*(i+a)      or      \*&i[a];

**Programming Example: Write a C program to find the largest of n numbers using arrays and pointers.**

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a[10],n,i,big,pos;
    clrscr();
    printf("\n Enter number of elements \n");
    scanf("%d",&n);
    printf("\n Enter the elements \n");
    for(i=0;i<=n-1;i++)
        scanf("%d",a+i);
    big = *(a+0);
    pos = 0;

    for(i=1;i<=n-1;i++)
    {
        if(*(a+i)>big)
            { big = *(a+i);
              pos= i;
            }
    }
}

```

```

        }
    }
    printf("\n Largest = %d\n",big);
    printf("\n Position = %d\n",pos+1);
    getch();
}

```

**Output :**

```

Enter number of elements
5
Enter the elements
10 30    75    -20   100
Largest = 100
Position = 5

```

**Programming Example :To find the sum of n natural numbers using pointers and arrays.**

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a[20 ],i,n,sum=0;
    int *p;
    clrscr();
    printf("\nEnter the value of n\n");
    scanf("%d",&n);
    printf("\nEnter the n values \n");
    for(i=0;i<n;i++)
    scanf("%d",&a[i]);
    p = a;
    sum = 0;
    for(i=0;i<=4;i++)
    {
        sum = sum + *p;
        p++;
    }
    printf("\n Sum of all the numbers = %d\n",sum);
    getch();
}

```

**Output :**

```

Enter the value of n
5
Enter the n values
10 30    75    -20   100
Sum of all the numbers = 195

```

**Array of Pointers :** Array of pointers is a collection of multiple pointers of particular data type. Each element of such array will be capable to point and hold the address of set variables whose data type is of same type as that of the array of pointers.

Consider a declaration **int \*pi[3];**

Here pi is array of pointers which can hold three pointers of integer type i.e.,\*pi[0],\*pi[1] and \*pi[2]. Following programming example illustrate how the values of a, b and c is accessed and printed using array of pointers. In this example pi[0],pi[1] and pi[2] holds the address of variables a,b and c respectively. And \*pi[0],\*pi[1] and \*pi[2] holds the values of a,b and c respectively.

```
#include<stdio.h>
#include<conio.h>
main()
{   int  *pi[3],a,b,c;
    clrscr();
    printf("\n Enter the values of a ,b and c \n");
    scanf("%d %d %d",&a,&b,&c);
    pi[0] = &a;
    pi[1] = &b;
    pi[2] = &c;
    printf("\n The Entered value of a,b and c\n");
    printf("%d %d%d\n",*pi[0],*pi[1],*pi[2]);
    getch();
}
```

#### **Output :**

```
Enter the values of a,b and c
10 20 30
The Entered value of a,b and c
10 20 30
```

### **5.1.4 Pointer Arithmetic ( Operations on Pointer) :**

Similar to the way arithmetic operations are possible on normal variables, it is possible to perform arithmetic operations on pointers as well. Various arithmetic operations carried out on pointers are **incrementation, decrementation, addition , subtraction and comparison**.

- a. **Incrementation :** Pointers can be created to *int ,float ,double* and *char* types of data. The pointer created to any of these data types can be incremented. The increment operator (++) increases the value of a pointer by the size of the data the pointer pointing to.

For example if pointer is pointing to character type of data , then ++ would increase the value of pointer by 1. Similarly if the pointer is pointing to integer data type then ++ would increase the value of pointer by 2. In general :

*Final value of the pointer = current value of the pointer + 1\*size of the data type;*

#### **Programming Example :**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char    cvar,*pc;
    int     ivar,*pi;
    float   fvar,*pf;

    pc = &cvar; /* if &cvar = 1000 */
    pi= &ivar;  /* if &ivar = 2000 */
    pf= &fvar; /* if &fvar = 3000*/

    printf("\n The value of pointer before incrementation \n");
    printf("%u %u %u\n",pc,pi,pf);
    /* The output will be 1000 2000 and 3000 */
```

```

pc++; /* 1000 + 1*1 */
pi++; /* 2000 + 1*2 */
pf++; /* 3000+1*4 */

printf("\n The value of pointers after incrementation\n");
printf("%u %u %u \n",pc,pi,pf);
/* The output will be 1001 , 2002 and 3004 */
getch();
}

```

- b. **Decrementation:** Similarly the decrement (--) operators decreases the value of pointer by the size of the data the pointer pointing to.

i.e *Final value of the pointer= current value of the pointer -1\*sizeof( data type);*

- c. **Pointer Addition:** In C language it is possible to add any integer number to a pointer variable. The final value of the pointer variable can be computed using the below given rule:

*Final value of a pointer = current value of the pointer + Integer number \*sizeof ( data type);*

**Example 1:**

```

char a;
char *p = &a; // if &a = 1000
p = p + 5; // p = 1000 +5*1 = 1005

```

**Example 2:** int a;

```

int *p = &a; // if &a = 1000
p = p + 5; // p = 1000 + 5*2 = 1010

```

- d. **Pointer Subtraction:** In C language it is possible to subtract any integer number to a pointer variable. The final value of the pointer variable can be computed using the below given rule :

*Final value of a pointer = current value of the pointer - Integer number \*size of the data type ;*

- e. **Pointers Comparison:** Two pointers can be compared with each other only if both the pointers are pointing to similar type of data. i.e two pointers should point to char or both the pointers should point to *int* etc.

Example : *if(ptr2==ptr1)*  
⇒ Pointers are equal to each other.

Example : *if(ptr2>ptr1)*  
⇒ Pointer ptr2 is far from ptr1.

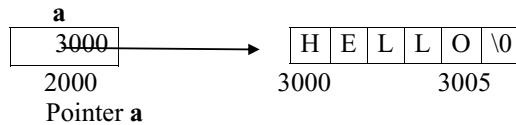
**5.1.5. Character Pointer:** The pointer to “*character and array of characters (string)*” is called as character pointer. In C language, character pointer can be used to access strings (arrays of characters) . A character pointer is created as shown below:

```
char *a;
```

Character pointers can be initialized as shown below :

```
char * a = "Hello";
```

In this case memory is created as shown below



The array present in the above diagram is called as *anonymous array*.

Using pointer **a** , Hello can be printed as shown below :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char *a = "Hello";
    clrscr();
    printf("%s\n",a);
```

```
        getch( );
    }
```

### Output :

Hello

#### 5.1.6 Pointer to Pointer :

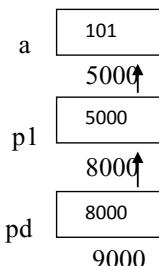
Consider the declaration `:int **p;`  
Here *p* is a pointer to pointer to integer. These types of pointers are called as *double pointer or pointer to pointer*. Single pointers are used to store the address of normal variables. Double pointer is to store the *address of pointer variables*.

Consider the example:

```
int a = 101;
int *p1;
int **pd;
```

```
p1 = &a;
pd = &p1;
```

The address of normal variable **a** is stored in **p1** similarly address of pointer variable **p1** is stored in **pd**. The memory layout for the above declarations is shown below :



### **Programming Example :**

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a = 101;
    int *p1;
    int **pd;
    clrscr();
    p1 = &a;
    pd = &p1;
    printf("%d\n", a);
    printf("%d\n", *p1);
    printf("%d\n", **pd);
    getch();
}
```

### **Output :**

```
101
101
101
```

**5.1.7 Dynamic Memory Allocation:** Dynamic Memory Allocation is process of allocation of memory to variables during runtime or execution. When a C program is compiled, the compiler allocates memory to store different data elements such as constants, variables, arrays and structures. This is referred to as *compile time or static memory allocation*. *This static memory allocation may lead to unnecessary allocation of memory locations which will be a waste if data is lesser than the allocated or may not be sufficient if data is more than the allocated locations.* hence we go for dynamic memory allocation. In case of dynamic memory allocation the memory is allocated at run time i.e during the execution of a program . Dynamic memory management involves the use of *pointers and four standard library functions namely malloc ,calloc , realloc and*

**free**. The first three functions are used to allocate memory whereas the last functions is used to free memory to the system . The pointers are used to point to the blocks of memory allocated dynamically . These function are declared in **stdlib.h** .

1. **malloc()**: it is used to allocate a required single block of memory of specified size dynamically and return a pointer of type void. The general form for memory allocation using **malloc** is

*datatype \*ptr =(data type \*)malloc (requiredAmount of memory \* sizeof(datatype));*

**Example 1 :**      int \*ptr;  
                      ptr = (int \*) malloc(100\*sizeof(int));

**Example 2 :**      char \*ptr ;  
                      ptr = (char \*) malloc(100\*sizeof(char));

2. **calloc()** :It is used to allocate memory in terms of contiguous multiple blocks of specified size. The *malloc()* allocates single block of memory whereas *calloc()* allocates multiple block of memory each of same size.The general format for memory allocation using **calloc** is

*datatype \*ptr= (datatype \*) calloc (RequiredAmountOf memory for Elements, sizeof(datatype)),*

**Example :**    *int \*ptr = (int \*)calloc(5,sizeof(int));*

3. **realloc( )** :If the previous allocated memory is insufficient the **realloc()** function can be used for increasing the size . It can be used to decrease or increase the size.

**The general format for memory allocation using calloc is**

```
datatype *ptr = (datatype *) realloc (ptr, newsize*sizeof(datatype));
```

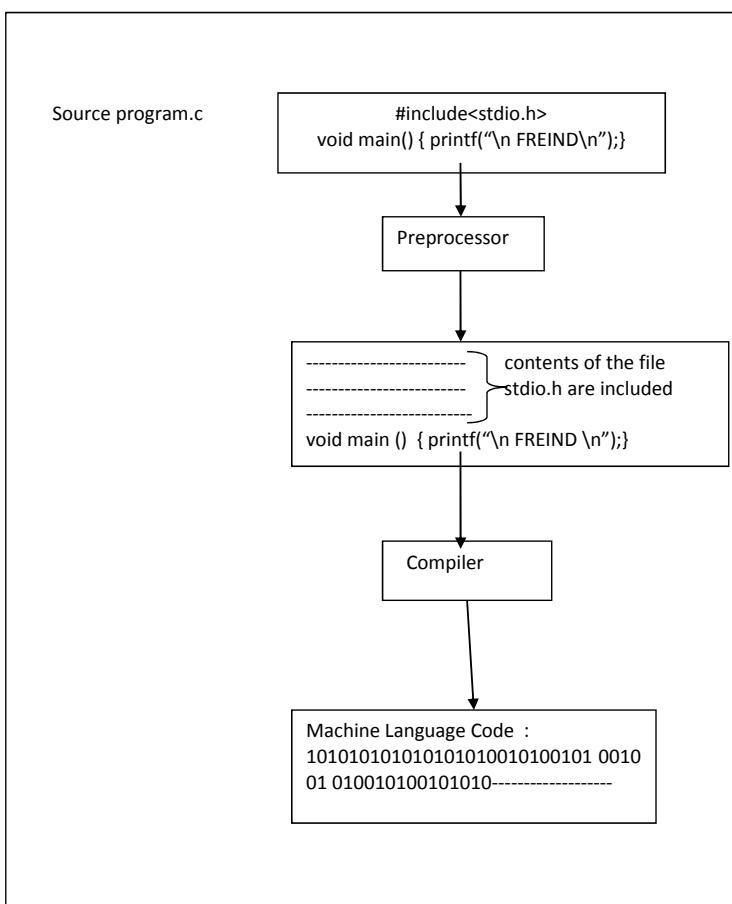
**Example :** p = (int \*) realloc(p,200\*sizeof(int));

4. **free( )** :This function is used to free the dynamically allocated memory from calloc() or malloc()

The general format is : *free(ptr);*

## 5.2 Preprocessors

**5.2.1 Introduction to Preprocessor:** Preprocessor is a program that processes preprocessing statements of the source code before it passes through the compiler. It is a program that accepts a source program with preprocessing statements as the input and produces another source program which will not contain any preprocessing statements. The process of C compilation is as shown in the figure 5.1.



**Figure 5.1 :** Compilation process of C compiler

**5.2.2 Preprocessor Directives (Statements):** Preprocessor directives or statements are the statements of C programming language which begins with # symbol.

**Examples :** #define , #include, #if, #endif , #ifdef and #else

Some of the important preprocessor directive are listed in table below :

| Directive | Meaning                                                             |
|-----------|---------------------------------------------------------------------|
| #define   | Define a macro                                                      |
| #include  | Specifies the files whose contents to be included                   |
| #if       | Test a Compile time condition                                       |
| #endif    | Specifies end of #if ,#ifdef and #ifndef                            |
| #elif     | Combines #else and #if                                              |
| #else     | Specifies alternatives if test fails with #if                       |
| #undef    | Undefines a macro                                                   |
| #ifdef    | Test if a macro is defined                                          |
| #ifndef   | Test if a macro is undefined                                        |
| #error    | Prints error message on stderr                                      |
| #pragma   | Sends special commands to the compiler using a standardized method. |

The Preprocessor directives can be divided into following three categories

1. Macro Substitution directives
2. File Inclusion Directives
3. Compiler control directives

The first two categories are also known as *unconditional preprocessor directives* and the third one is also known as *conditional control directives*.

1. **Macro Substitution Directives:** Macro substitution is a process where an identifier in a program is replaced by a predefined string composed of one or more tokens. The preprocessor accomplishes this task under the directions of **#define** statement. This statement usually known as a

macro substitution directives (or simply a macro) takes the following general form :

```
#define identifier token_string
```

**Example :**

```
#define COUNT 100
#define FALSE 0
#define PI 3.1415926
```

**Programming Example :**

```
#include <stdio.h>
#include<conio.h>
#define height 10
#define number 3.145
#define letter 'R'
#define letter_sequence "XYZ"
#define backslash_char '?'
void main()
{
    printf("value of height : %d \n", height );
    printf("value of number : %f \n", number );
    printf("value of letter : %c \n", letter );
    printf("value of letter_sequence : %s \n", letter_sequence);
    printf("value of backslash_char : %c \n", backslash_char);

}
```

**Output:**

```
value of height : 10
value of number : 3.145000
value of letter : R
value of letter_sequence : XYZ
value of backslash_char : ?
```

**Programming Example : C Program to find the area of a circle**

```
#include <stdio.h>
#include<conio.h>
#define PI 3.1415 /* MACRO SUBSTITUTION DIRECTIVE*/
int main() {
    int radius;
    float area;
    clrscr();
```

```
    printf("Enter the radius: ");
    scanf("%d",&radius);
    area=PI*radius*radius;
    printf("Area=%.2f",area);
    getch();
    return 0;
}
```

## Macros with argument:

Preprocessing directive `#define` can be used to write macro definitions with parameters as well in the form below:

```
#define identifier(identifier 1,...,identifier n) token_string
```

### Examples :

```
#define MAX(x,y) ((x)>(y) ? (x) : (y))
#define square (x) ((x) * (x))
```

**Example :** C Program to find area of a circle, passing arguments to macros. [Area of circle= $\pi r^2$ ]

```
#include <stdio.h>
#include<conio.h>
#define PI 3.1415
#define area(r) (PI*(r)*(r))
int main() {
    int radius;
    float area;
    clrscr( );
    printf("Enter the radius: ");
    scanf("%d",&radius);
    area=area(radius);
    printf("Area=%.2f",area);
    getch( );
    return 0;
}
```

2. **File Inclusion Directives:** An external file containing functions or macro definitions can be included as a part of a program so that we

need not rewrite those functions or macro definitions. This is achieved by the preprocessor directives: **#include <filename>**

**Example :** #include<stdio.h> . Here, "stdio.h" is a header file and the preprocessor replace the above line with the contents of header file.

**Other Examples :** #include<conio.h> , #include “TEST.C” ,  
#include “SYNTAX.C” , etc.

**Example :** Program to illustrate the use of #include directive

```
#include<stdio.h>
#include<conio.h>
void main()
{ clrscr();
    printf("\n This program illustrates the use of #include directive\n");
    getch();
}
```

**3. Compiler Control Directives:** C preprocessor offers a feature known as conditional compilation which can be used to SWITCH ON or OFF a particular line or group of lines in a program. These preprocessing directives can be used for conditional compilation which controls the compiling of the source code. The general form of compiler control directive is :

```
#if condition_1
statement_block1;
#elif condition_2
statement_block2
-----
#elif condition_n
statement_blockn
#else
default_statement_block
#endif
```

- The condition may be expression which evaluates to a constant.

- The statement\_block code contains multiple C statements , including preprocessor directives.
- The #elif and #else directives are optional , whereas the #if and #endif directives are required.
- Multiple #elif directives are allowed.
- Only one #else directive is allowed.

**The #if ,#elif ,#else and #endif Directives:** The control directives like #if ,#elif,#else ,#ifdef and #ifndef are used to determine whether the preprocessor will remove lines of code before giving to the compiler. When the compiler comes across the #if directive , it tests the condition specified with the #if directive. If it evaluates to false , the compiler then tests the conditions specified in each #elif directive. The statements related with the first true #elif directive are compiled. However, if not a single condition evaluate to true, then the statements following the #else directive are compiled.

The #if directive enables you to use relational operators ( such as >,<,>=) in a test condition and also allows to connect multiple test conditions together using logical operators. If the specified conditions are true then the subsequent code statements are executed.

#### **Example: Illustration of #if ,#else ,#ifndef and #endif**

```
#include<stdio.h>
int main() {
#ifndef X
    printf(" X Not Defined");
#else
#if X < 10
    printf(" X is less than 12 \n");
#endif
#endif
}
```

**Output :**

X Not Defined

**The #elif Directives:** The **#elif** directive is similar to the **#else** directive with a difference that it allows to specify a condition to be checked before executing the **#else** construct. It is used after the **#if**, **#ifdef** and **#ifndef** directives.

### Example : Usage of #elif Directive

```
#include<stdio.h>
#ifndef USD
    #define Currency_Value 60
#elif ( #ifndef EURO )
    #define Currency_Value 80
#else
    #define Currency_Value 1
#endif
void main()
{
    int rupee;
    clrscr();
    rupee = 10*Currency_Value;
    printf("%d\n",rupee);
}
```

#### Output :

If USD is defined the output is 600

If USD is not defined but EURO is defined the output is 800

If both USD and EURO is undefined the output is 10.

**The #undef Directive :** It is used to undefine the value assigned to a variable by using the **#define** directive. The following code snippet illustrates the usage of **#undef** directive.

```
#include<stdio.h>
#ifndef DEBUG
    #undef DEBUG
    #define DEBUG 1
#endif
```

**The #ifdef Directive :** It verifies whether a macro is defined . If the macro is defined , then compiler translates the lines of code that are followed by the #ifdef condition.

**Example :**

```
#ifdef DEBUG 1  
printf("HELLO RASHI");  
#end if
```

In this case the output will be HELLO RASHI ,since DEBUG is defined and its value is 1.

**The #ifndef Directive :** It is opposite to the #ifdef Directive . It verifies whether a macro is not defined.

**Example:**

```
#include<define.h>  
#ifndef TEST  
#define TEST 1  
#end if
```

Here the compiler will check whether TEST is defined or not. If it is not defined already it will define as 1.

**The #error Directive :** It will print custom error messages at compile time and terminate the process of compilation. The syntax for #error directive is : #error “ text\_string”

**Example3:**

```
#ifndef TEST  
#error "Error Occurred"  
#endif
```

## 5.3 Introduction to Data Structures

**5.3.1 Primitive and Non Primitive Data Structures:** Data is recorded facts and figures that can be retrieved and manipulated in order to produce useful information / results. A **data structure** is a way of organizing or storing data in a computer so that it can be used efficiently. Data structures can be broadly classified into

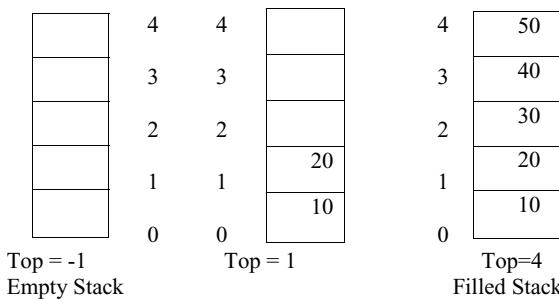
1. **Primitive Data Structure**
2. **Non-Primitive Data Structures**

1. **Primitive Data Structures :**The primitive data structures are data structures that can be manipulated directly by machine instructions . They are basic data types or fundamental data types. The C language provides the following primitive data types :*character, integer, float ,double and void.*
2. **Non Primitive Data structures :**are those that are not defined by the programming language but are instead created by the programmer. These are once again classified into **a.** Linear and **b.** Non Linear Data Structures.
  - a. **Linear Data Structures:** Here the data elements are arranged in linear fashion. These include *stack , queue and linked list.*
  - b. **Non Linear Data Structures:** Here the data elements are arranged in nonlinear fashion. These include *Tree, Graph and Map.*

**5.3.2 Stack :** A **stack** is a non-primitive linear data structure. It is an ordered list in which addition of new data and deletion of already existing data item is done from only one end, known as top of stack (**TOP**). Here the last added element will be the first to be removed from the stack. That is the reason why stack is also called **Last in First Out (LIFO)** type of list. The initial value of **TOP = -1** and final value of **TOP = MAX-1**;

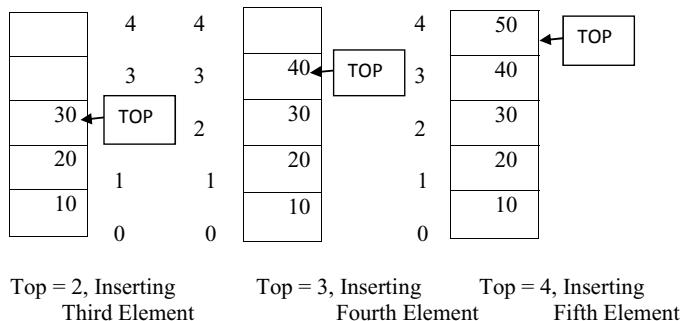
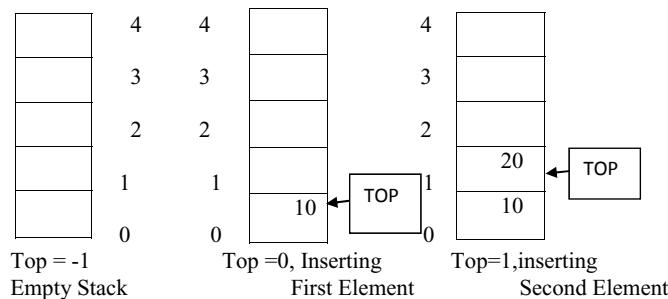
**Example1:** A common realtime example of a stack is arrangement of plates in a marriage party. New plates are “*pushed*” onto to the top and “*popped*” off the top.

**Example 2:** Another example is biscuits (or Poppins) pack in which only one side of the cover is torn and biscuits are taken (pooped) off one by one. Also biscuits can be pushed into the pack through the same torn for preserving. Figure below illustrates the stack of size 5:

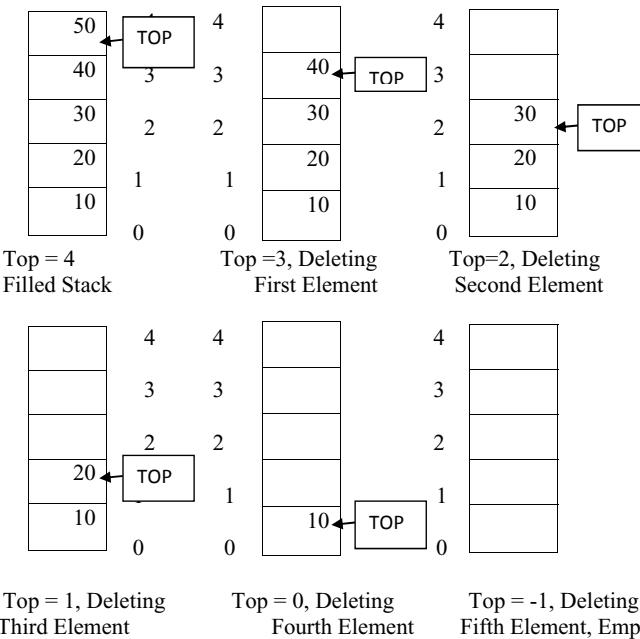


**Operations on Stack:** A stack has three basic operations:*push, pop and peep*. The push operation adds an element to the stack and pop operations removes an element from the top of the stack. The peep operation returns the value of the top most elements of stack operations.

**Push Operations:** Here the elements are added to the top of the stack. For empty stack the value of Top = -1. At each time of adding (pushing) the element the value of the stack increases by 1. Following figure illustrates the push operations.



**Pop Operations:** Here the elements are deleted from the top. At each time of the pop operation the value of top decrements by 1. Following figure illustrates the pop operations.



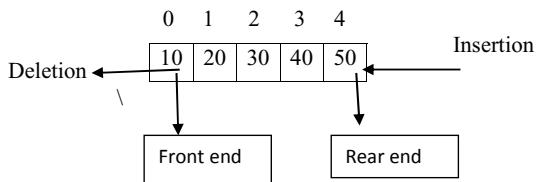
**Peep operation:** This operation is used to display all the elements in the stack from bottom to top.

**Overflow and Underflow:** Pushing an element to completely filled stack is called *Overflow* and the process of deleting an element from the empty stack is called *Underflow*.

**Applications of Stack :** Stack data structures can be used in numerous applications that require LIFO operations. Some of the common applications include

- To convert an infix expression into postfix expression
- To evaluate Postfix expression
- To implement Recursion
- To reverse string
- To evaluate arithmetic expressions
- To check whether a string is palindrome or not.
- To store history data (like Browsing history)

**5.3.3 Queue :** is a non-primitive linear data structure where elements are inserted from *rear end* and elements are deleted from the *front end*. Since the first item inserted is the first item to be deleted from queue , queue is also called First In First Out (**FIFO**) data structure . The pictorial representation of the queue is as illustrated below :



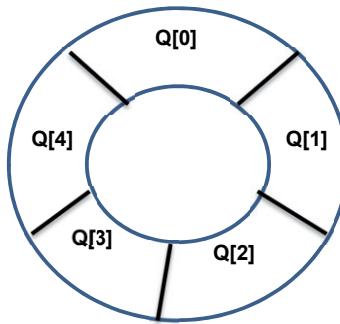
**The operations possible on queue are :**

1. Insert operation (Adding Element /Enqueue)
2. Delete operations ( Removing Elements /Dequeue)
3. Display Operations ( Display elements in Queue)

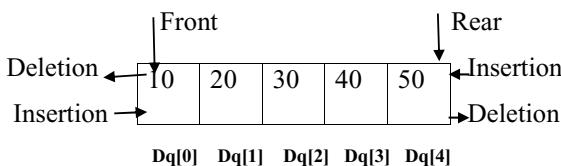
**Types of Queues:** Linear Queue, Circular Queue and Double Ended Queue.

**1. Circular Queue :** A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location of the queue is full. In other words if we have a queue Q of say n elements , then after inserting an element in last (i.e in the n-1 th) location of the array the next element will be inserted at the very first location of the array. It is possible to insert new elements if and only if those locations are empty.

Figure below shows a empty circular queue Q[5] which can accommodate five elements.



**2. Double Ended Queue (Deque)** : It is also a homogeneous list of elements in which insertion and deletion operations are performed from both the ends. That is , we can insert elements from the rear end or from the front ends. Hence it is called double ended queue. It is commonly referred as deque. Figure below shows a deque of 5 elements:



**3. Priority Queues** : Here a priority is assigned to each element of the queue.These priorities are considered at the time of processing the elements . The element having higher priority is processed before that of lower priority. In memory , the priority queues can be represented as a one way list .

#### Applications of Queue :

1. Used in Job Scheduling Algorithm
2. Used in printers to store the requests made for printing when the printer is busy ( network printer).
3. It can be used to store data in the first in first out format .
4. For CPU Scheduling

**5.3.4 Linked List :** A linked list is a linear data structure which is a collection of zero or more connected nodes with each node consisting of two fields data and link. The data field holds the actual information that has to be processed and the link field includes the address of another node. The pictorial representation of the node is shown below:

Node = info + link

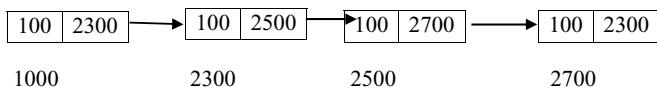


A node in a list consists of two fields namely info and link :

**Info ->** This field is used to store the actual data or information to be manipulated.

**Link ->** A link basically contains address of the next node

**Example :**



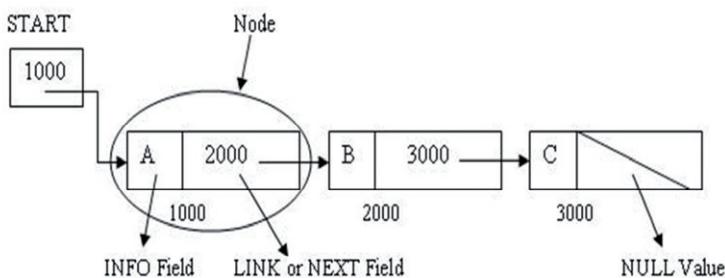
**Types of Linked Lists :** There are basically Three types of linked list as given below :

1. Singly linked List
2. Doubly Linked List
3. Circular Linked List

#### 1. Linear Linked List or One Way List or Singly Linked List:-

It is linear collection of data elements which are called ‘Nodes’. The elements may or may not be stored in consecutive memory locations. So pointers are used to maintain linear order. Each node is divided into two parts. The first part contains the information of the element and is

called ‘INFO Field’. The second part contains the address of the next node and is called ‘LINK Field’ or ‘NEXT Pointer Field’. The START contains the starting address of the linked list i.e. it contains the address of the first node of the linked list. The LINK Field of last node contains NULL Value which indicates that it is the end of linked list. It is shown below



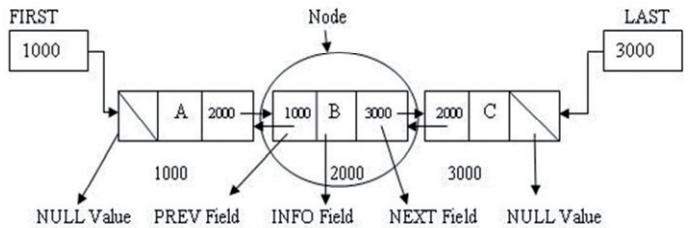
## 2. Doubly Linked List or Two-Way Linked List or Two-Way Chain:-

It is a type of linked list in which each node is divided into three parts:

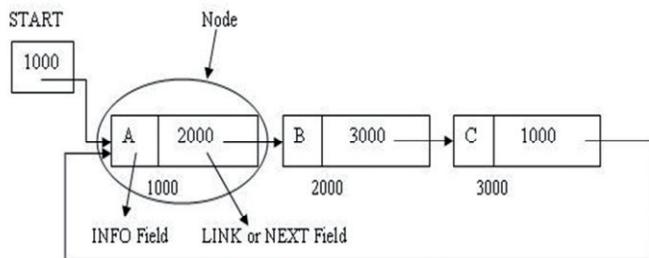
1. The first part is PREV part. It is previous pointer field. It contains the address of the node which is before the current node (i.e address of left node /left link).
2. The second part is the INFO part. It contains the information of the element.
3. The third part is NEXT part. It is next pointer field. It contains the address of the node which is after the current node. (i.e address of right node /right link)

There are two pointers FIRST and LAST. FIRST points to the first node in the list. LAST points to the last node in the list. The PREV field of first node and the NEXT field of last node contain NULL value. This shows the end of list on both sides. This list can be traversed in both directions that is forward and backward.

It is shown below:



3. **Circular Linked List:**-It is a list of nodes in which the last node contains the address of first node as illustrated in figure below:



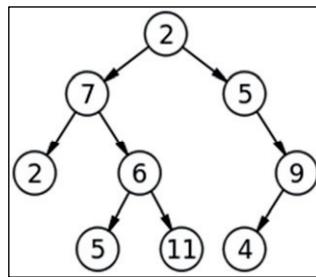
**Circular doubly linked lists :**It is a variation of doubly linked list in which right link of the last node contains the address of the first node and left link of the first node contains the address of the last node.

**Applications of linked lists :**Linked Lists are used in the following applications

1. Polynomial manipulation
2. Linked dictionary or symbol table of a compiler
3. Multiple precision arithmetic operations

**5.3.5 Trees :**A tree is a non-linear data structure made up of nodes or vertices and edges without having any cycle that represents parent – child relationship between the data items stored in it. The tree with no nodes is called the **null** or **empty** tree. A tree that is not empty consists of a root node

and potentially many levels of additional nodes that form a hierarchy . The pictorial representation of trees is represented in figure below :



Some of the terminologies used to describe Tree data structure are as follows:

**Root** – The top node in a tree. Example : 2

**Parent** – Node which have children's. Example : 2,7,5,6 and 9

**Siblings** – Nodes with the same parent. Example : 7,5 and 2,6 and 5,11

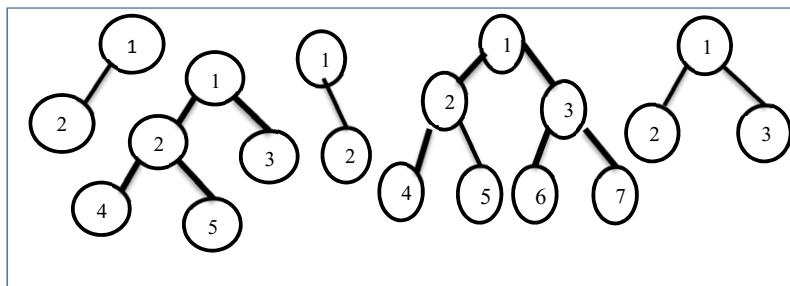
**Leaf** – a node with no children. Example : 2,5,11 and 4

**Degree** – number of sub trees of a node.

**Edge** – connection between one node to another.

**Types of Trees:** There are various type of tree data structure that can be created. They are 1.*Binary Tree* 2.*Binary Search Tree* 3. *Expression Trees* 4.*AVL Tree* 5.*B Tree* and 6.*B+ Tree* etc.

**Binary Tree :**A binary Tree is a tree in which each node can have at most two children . One of them is designated as the left child and the other one has right child. Figure below gives the examples for Binary Tree.



### **Applications of Trees:**

1. For decision making assignment problems like decision trees
2. To implement game playing applications like chess, checkers ,etc.
3. To implement searching problem .
4. To implement Sorting problem
5. To represent infix ,prefix and postfix expressions.

**5.3.6 Abstract Data type (ADT):** ADT is a collection of related data items together with basic relations between them and operations to be performed on them. **Examples :** Stack ADT,Queues ADT and Linked List ADT.

#### **1. Stack Abstract Data Type :**

**Definition :** A list of data items that can only be accessed at one end called the top of the stack .

**Operations :** Push( ) , Pop( ) and PeeP( ).

#### **2. Queue Abstract Data Type :**

**Definition :** A list of data items in which an item can be deleted from the front end of the queue and an item can be inserted at the rear end of the queue.

**Operations :** Enqueue() , Dequeue() and Display().

#### **3. Linked List Abstract Data Type :**

**Definition :** A list of connected nodes , where in each node is made up of two parts 1.Information part and 2. Address part.

**Operations :** Insert() , Delete () , Retrieve () , Traverse() and Display()

## Difference between Static Memory Allocation and Dynamic Memory Allocation

| <b>Sl.NO</b> | <b>Static Memory Allocation Technique</b>                                                                                 | <b>Dynamic Memory Allocation Technique</b>                                                                                         |
|--------------|---------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>1</b>     | Memory is allocated during compilation time                                                                               | Memory is allocated during execution or runtime                                                                                    |
| <b>2</b>     | The size of the memory to be allocated is fixed during compilation time and cannot be altered during execution time       | When required memory can be allocated and when not required memory can be deallocated                                              |
| <b>3</b>     | Used only when the data size is fixed and known in advance before processing                                              | Used only for unpredictable memory requirement                                                                                     |
| <b>4</b>     | Execution is faster , since memory is already allocated and data manipulation is done on these allocated memory locations | Execution is slower since memory has to be allocated during run time . Data manipulation is done only after allocating the memory. |
| <b>5</b>     | Memory is allocated either in stack area (for local variables) or data area ( for global and static variables)            | Memory is allocated only in heap area.                                                                                             |
| <b>6</b>     | Example : arrays                                                                                                          | Example : Dynamic arrays , linked Lists, Trees                                                                                     |

## Difference between Arrays and Pointers

| Sl. NO | Pointers                                                                           | Arrays                                                                                                               |
|--------|------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| 1      | Pointer is a variable                                                              | Array is a name to set of memory locations                                                                           |
| 2      | Allocation and deallocation of memory is required                                  | Not required                                                                                                         |
| 3      | A pointer is a place in memory that keeps address of another place inside          | An array is a single, pre allocated chunk of contiguous elements (all of the same type), fixed in size and location. |
| 4      | Pointer is dynamic in nature. The memory allocation can be resized or freed later. | They are static in nature. Once memory is allocated , it cannot be resized or freed dynamically.                     |
| 5      | The assembly code of Pointer is different than Array                               | The assembly code of Array is different than Pointer.                                                                |

## Similarity between array and pointer:

- Array name holds the address of first element
- *Arrays are defined as const pointer.*
- Array and pointer usage notation can be interchange.
- Pointer can be used in array notation:

```
char * ptr = new char[3];
ptr[0] = 'a';
ptr[1] = 'b';
```

- **Array can be used in pointer notation:**

```
char arr[ ] ="abcde";
*(arr+4) = 'd';
```

## **Advantages of pointers in C:**

- Pointers provide direct access to memory.
- Pointers are used to access the value of the variable through its address.
- Pointers provide a way to return more than one value to the functions.
- Reduces the storage space and complexity of the program.
- Reduces the execution time of the program, i.e execution is faster.
- Provides an alternate way to access array elements .
- Pointers can be used to pass information back and forth between the calling function and called function.
- Pointers allows us to perform dynamic memory allocation and deallocation.
- Pointers helps us to build complex data structures like linked list, stack, queues, trees, graphs etc.
- Pointers allows us to resize the dynamically allocated memory block.
- Addresses of objects can be extracted using pointers.

## **Disadvantages of pointers in C:**

- Pointers are not safe and insecure because one can access the restricted memory area and the values of other variables without permission.
- Uninitialized pointers might cause segmentation fault.
- Dynamically allocated block needs to be freed explicitly. Otherwise, it would lead to memory leak.
- Pointers are slower than normal variables.
- If pointers are updated with incorrect values, it might lead to memory corruption.

## 5.4: Programming Examples

### 1. Example program for #define, #include preprocessors in C:

```
#include <stdio.h>
#define MAX    100
#define PI     3.14
#define LETTER 'X'
#define STR   "XYZ"
#define backslash_char  '?'

main()
{
    clrscr( );
    printf("value of MAX : %d \n", MAX );
    printf("value of PI : %f \n", PI );
    printf("value of LETTER : %c \n", LETTER );
    printf("value of STR : %s \n", STR );
    printf("value of backslash_char : %c \n", backslash_char );
    getch( );
}
```

#### Output:

```
value of MAX : 100
value of PI : 3.140000
value of LETTER : X
value of STR : XYZ
value of backslash_char : ?
```

### 2. Example program for #ifdef, #else and #endif in C:

```
#include <stdio.h>
#define THYAGARAJU 1000

int main()
{
    #ifdef THYAGARAJU
    printf("THYAGARAJU is defined.\n");
    #else
    printf("THYAGARAJU is not defined\n");
    #endif
    return 0;
}
```

**Output:THYAGARAJU is defined.**

**3) Example program for #ifndef and #endif in C:**

```
#include <stdio.h>
#define INDIA 1000
int main()
{ #ifndef BENGALURU
{ printf("BENGALURU is not defined");
#define BENGALURU 500
}
#else
printf("BENGALURU is already defined in the program");
#endif
return 0;
}
```

**Output: BENGALURU is not defined**

**4) Example program for #if, #else and #endif in C:**

```
#include <stdio.h>
#define a 100
int main()
{
#if (a==100)
printf("This line will be Activated");
#else
printf("This line will be Deactivated");
#endif
return 0;
}
```

**Output:** This line will be Activated

**5) Example program for illustrating the directive which undefines existing macro in the program.**

```
#include <stdio.h>
#define N 100
void main()
{
printf("First defined value for N : %d\n",N);
```

```
#undef N
#define N 600
printf("Value of N after undef :%d",N);
}
```

**Output:**

```
First defined value for N: 100
Value of N after undef : 600
```

- 6) Example program to illustrate the working of pragma in C.**(*Note : Pragma is a macro which is used to call a function before and after main function in a C program.*)

```
#include <stdio.h>
#include<conio.h>
void fun1();
void fun2();

#pragma startup fun1
#pragma exit fun2

int main()
{
    clrscr();
    printf( "\n Control is in the main function" );
    getch();
}

void fun1()
{
    printf("\nFunction1 is called before main function ");
}

void fun2()
{
    printf( "\n Function2 is called just before end of main function )
;"}
```

**Output:**

```
Function1 is called before main function call
Control is in main function
Function2 is called just before end of main function
```

### **7) C program to access the elements of arrays using pointers**

```
#include <stdio.h>
#include<conio.>
int main()
{
    int a[5], i;
    clrscr();
    printf("Enter elements: ");
    for(i=0;i<5;++i)
        scanf("%d",a+i);
    printf("You entered: ");
    for(i=0;i<5;++i)
        printf("%d\n",*(a+i));
    getch()
    return 0;
}
```

### **Output**

Enter elements:

1  
2  
3  
5  
4

You entered:

1  
2  
3  
5  
4

### **8) C program to find the largest number in a list using pointers and dynamic memory allocation.**

```
#include <stdio.h>
#include <stdlib.h>
#include<conio.h>
int main(){
    int i,n;
    float *p;
    clrscr();
    printf("Enter total number of elements");
```

```

scanf("%d",&n);
p=(float *)calloc(n,sizeof(float));
if(p==NULL)
{
    printf("Error!!! memory not allocated.");
    exit(0);
}
printf("\n");
for(i=0;i<n;++i) /* Stores number entered by user.*/
{
    printf("Enter Number %d: ",i+1);
    scanf("%f",p+i);
}
for(i=1;i<n;++i) /* Loop to store largest number at address p */
{
    if(*p<*(p+i))
        *p=*(p+i);
}
printf("Largest element = %.2f",*p);
return 0;
}

```

## Output

Enter total number of elements: 10

Enter Number 1: 12.345  
 Enter Number 2: 13.436  
 Enter Number 3: 16.578  
 Enter Number 4: 12.456  
 Enter Number 5: 77.649  
 Enter Number 6: 99.052  
 Enter Number 7: -33.456  
 Enter Number 8: -19.999  
 Enter Number 9: 25.678  
 Enter Number 10: 249.956  
 Largest element: 249.95

### 9) C program to find the sum of elements of array using pointers.

```

#include <stdio.h>
#include <malloc.h>
#include<conio.h>
void main()

```

```
{  
    int i, n, sum = 0;  
    int *a;  
    clrscr();  
  
    printf("Enter the size of array A \n");  
    scanf("%d", &n);  
    a = (int *) malloc(n * sizeof(int));  
    printf("Enter Elements of First List \n");  
    for (i = 0; i < n; i++)  
    {  
        scanf("%d", a + i);  
    }  
    /* Compute the sum of all elements in the given array */  
    for (i = 0; i < n; i++)  
    {  
        sum = sum + *(a + i);  
    }  
    printf("Sum of all elements in array = %d\n", sum);  
}
```

**Output :**

```
Enter the size of array A  
5  
Enter Elements of First List  
4  
9  
10  
56  
100  
Sum of all elements in array = 179
```

## 5.5 Exercises

### Part A: Theory Questions

1. Define Pointer. Explain the declaration and initialization of pointer.
2. What do you mean by indirection?
3. Explain briefly the concept of pointers with schematic diagram.
4. What is the difference between a variable and pointer variable?
5. Distinguish between address operator and indirection operator.
6. Explain the different arithmetic operations that can be performed on pointers.
7. Describe the following with examples
  - a. Pointer Operations or Pointer Arithmetic
  - b. Pointer to pointers
  - c. Character Pointers
  - d. Call by Reference or Passing pointers as a function arguments
8. Define the following :
  - a. Pointer Constant
  - b. Null pointers
  - c. Dangling Pointers
  - d. Void pointer
  - e. L-Value and R- Value
9. Explain the arrays of pointers with examples .
- 10.Explain how the pointers can be passed as an argument of functions.
- 11.Differentiate between Call by Reference and Call by Value
- 12.Differentiate between Static memory allocation and dynamic memory allocation.
- 13.What is dynamic memory allocation? Write and explain the different dynamic memory allocation functions in C.

- 14.Explain malloc() , calloc() ,realloc() and free() functions with syntax and examples.
- 15.Differentiate between malloc() and calloc().
- 16.Differentiate between calloc() and realloc().
- 17.What is preprocessor? What are preprocessor directives? Explain the types of preprocessor directives with examples.
- 18.Explain Macro Substitution , File inclusion and Compiler Control directives with examples.
- 19.List out the preprocessive directives and give their meaning.
- 20.What are preprocessor directives? Explain #define and #include preprocessor directives.
- 21.What are primitive and non-primitive data types? Give Example.
- 22.What is Data Structures? What are the different types of data structures? Give examples.
- 23.What is Stack ? Explain it with its applications.
- 24.Define Queue. Explain it along with its applications.
- 25.What is Linked List? Explain it along with its applications.
- 26.What are Trees ? Explain it along with its applications.
- 27.What is Binary Tree ? Give Example.
- 28.What is Abstract Data Type. Give examples.
- 29.Explain : i) Abstract Data Types ii) Stack iii) Linked List .

## **Part B: Programs**

1. Write a C program to find the sum of  $n$  natural numbers using pointers.
2. Write a C program to read  $n$  unsorted numbers to an array of size  $n$  and pass the address of this array to a function to sort the numbers in ascending order using bubble sort technique.
3. Write a C program to read and write array elements.
4. Write a program in C to find the sum and mean of all elements in an array . Use pointer technology.
5. Write a C program to swap two numbers using call by pointers method.
6. Write a C program to pass array elements to the function using call by reference method.
7. To copy the contents of one string to another and display the contents of both strings using pointers.
8. Write a C program to sort  $n$  numbers using bubble sort technique and pointers.
9. Write a C program to read  $n$  unsorted numbers to an array of size  $n$  and pass the address of this array to a function to sort the numbers in ascending order using bubble sort technique

## **6. Old Question Papers**

First /Second Semester B.E Degree Examination ,  
June /July 2015

Programming in C and Data Structures  
(14PCD13/23)

Time :3Hrs

Max.Marks : 100

Note : Answer Any FIVE questions , Selecting ONE full question from each part

**PART-1**

- 1 a. What are data types? Mention the different data types supported by C language, giving an example to each. 05 Marks

**Solution :** Refer Section 1.5.3 of Module1

- b. Write a C program which takes as input p, t, r compute the simple interest and display the result. 05 Marks

**Solution :** Refer Section 1.8 , Programming Examples of Module1 (Example 34)

- c. What is an operator? List and explain various types of operators. 10 Marks

**Solution :** Refer Section 1.7 of Module1

- 2 a. What is a token? What are different types of tokens available in C language? Explain. 08 Marks

**Solution :** Refer Section 1.5.2 of Module1

- Write C expressions corresponding to the following ( Assume all b. quantities are of same type) 06 Marks

a.  $\frac{5x+3y}{a+b}$

b. Area =  $\sqrt{(s(s-a)(s-b)(s-c))}$

c. C =  $e^{|x+y-10|}$

d. D =  $x^{25} + y^{35}$

e. X =  $\frac{e^{\sqrt{x}} + e^{\sqrt{y}}}{x \sin \sqrt{y}}$

f.  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

**Solution :**

a.  $(5*x + 3*y)/(a+b)$

b. Area =  $\text{sqrt}(s*(s-a)*(s-b)*(s-c))$

c. C =  $\text{exp}(\text{abs}(x+y-10))$

d. D =  $\text{pow}(x,25) + \text{pow}(y,35)$

e. X =  $(\text{exp}(\text{sqrt}(x)) + \text{exp}(\text{sqrt}(y)))/(x * \text{sin}(\text{sqrt}(y)))$

f. X =  $(-b + \sqrt{b^2 - 4*a*c})/2*a$  and  
 $X = (-b - \sqrt{b^2 - 4*a*c})/2*a$

- c. What is the value of 'x' in following code segments? Justify your answers : 06 Marks

i)      int a,b ;                  ii)    int a,b;

```

float x;           float x;
a = 4;            a = 4;
b = 5 ;          b = 5;
x = b/a;          x = (float) b/a;

```

**Solution :**

i)  $x = b/a = 5/4 = 1.000000$ .

**Justification :** Since a and b are integer , b/a will results in integer division and the resultant value will be converted into float as x is floating pointing number (implicit type cast).

ii)  $x = (\text{float}) b/a = 5.000000/4 = 1.250000$

**Justification:** This is the case of explicit type casting where in the integer b will gets converted into floating point number resulting in mixed mode arithmetic operation.

## Part -2

- 3 a. What are different types of conditional decision making statements ? Explain each with examples. 10 Marks

**Solution :** Refer Section 2.2.1

- b. Write a C program to simulate simple calculator that performs arithmetic operations using switch statement . Error message should be displayed , if any attempt is made to divide by zero. 10 Marks

**Solution :** Refer Section 2.2.1.3 (Switch Statement Example)

- 4 a. Explain with examples formatted input output statements in C. 10 Marks

**Solution :** Under the Section Input and Output Functions of Module1 Refer 1. Formatted Input Output Functions and also Refer Sections 1.6.3 and 1.6.4

- b. List four differences between while loop and do while loop along with syntax and example. 06 Marks

**Solution :** Refer difference between while loop and do while loop under the section 2.3.2

- c. Design and develop a C program to reverse a given four digit integer number and check whether it is a palindrome or not. 08 Marks

**Solution :** Refer Computer Programming Lab Manual (Part B : Program number 2 )

## PART -3

- 5a What is an array ? Explain different methods of initialization of single dimensional arrays. 06 Marks

**Solution :** Refer section 3.1.1 and 3.1.3 of Module 3

- b. Write a C program to read N integers into an array A and to 06 Marks

i) find the sum of odd numbers

ii) find the sum of even numbers

iii) find the average of all numbers

output the results computed with appropriate headings .

- Solution :** Refer section 3.4.1 ( Example 9) of Module 3
- c How string is declared and initialized ? Explain any FOUR string manipulation functions with examples. 08 Marks
- Solution :** Refer section 3.2.2 and 3.2.3 of Module 3
- 6a Explain function call, function definition and function prototype with examples to each. 06 Marks
- Solution :** Refer section 3.3.2 of Module 3
- b What are actual parameters and formal parameters ? Illustrate with example. 06 Marks
- Solution :** Refer Additional Concepts Section of Module 3.
- c What is recursion ? Write a C program to compute the factorial of a given number ‘n’ using recursion. 08 Marks
- Solution :** Refer Section 3.3.7 of Module 3

## Part4

- 7a How structure is different from an array? Explain the declaration of a structure with an example. 06 Marks

**Solution :** Refer Section 4.1.1 and Difference between Arrays and Structures of Module 4

- b Explain with an example how to create a structure using ‘typedef’ 04 Marks

**Solution :** Refer Section 4.1.6 of Module 4

- c Write a C program to input the following details of ‘N’ students using structure: Roll No: integer , Name : string , Marks : float , Grade : char . Print the names of the students with marks  $\geq 70.0\%$  10 Marks

**Solution :**

```
#include<stdio.h>
#include<conio.h>
struct student
{
    int rno; /* Roll No*/
    char name[20]; /* Name*/
    float marks; /*Marks*/
    char grade; /*Grade*/
};

void main()
{
    int i,n,found=0;
    struct student s[20];
    clrscr();
    printf("\nHow many student details ?");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\n Type in %d student detail \n",i+1);
        printf("\n Roll no :");
        scanf("%d",&s[i].rno);
        printf("\n Name:");
        scanf("%s",s[i].name);
    }
}
```

```

        printf("\n Marks:");
        scanf("%f",&s[i].marks);
        printf("\n Grade .");
        scanf(" %c",&s[i].grade);
    }

printf("\n The Names of the students with marks >=70.0%\n");
for(i=0;i<n;i++)
{
    if(s[i].marks>=70.0)
    {
        printf("\n %s\n",s[i].name);
        found =1;
    }
}
if(found ==0)
printf("\n There is no student with marks >=70.0%\n");
getch();
}

```

- 8 a Explain following file operations along with syntax and examples : 10 Marks  
 i) fopen() ii)fclose() iii) fscanf() iv)  
 fprintf() v) fgets()
- Solution :** Refer section 4.2.2 and 4.2.3 of Module 4
- b. Write a C program to read the contents from the file called **abc.txt** , 10 Marks  
 count the number of characters , number of lines and number of  
 white spaces and output the same.
- Solution :** Refer section 4.3.2 Example 13.

## Part 5

- 9a Define pointer variable. Explain with an example the declaration and initialization of pointer variable. 06 Marks  
**Solution :** Refer Section 5.1.1 of Module 5
- b Explain following C functions along with syntax and example to each : i)malloc() ii) calloc() iii) realloc() iv) free() 08 Marks  
**Solution :** Refer section 5.1.7 of Module 5
- c Develop a C program to read two numbers and function to swap these numbers using pointers. 06 Marks  
**Solution :** Refer section 5.1.2 of Module 5
- 10 Write short notes on the following : 20 Marks  
 a. Preprocessor directives  
 b. Primitive and Non Primitive Data types  
 c. Stack Operations  
 d. Types of queues
- Solution :** Refer section 5.2 , 5.3.2 and 5.3.3 of Module5  
**Solution :** Refer section 1.5.3 of Module1 and section 5.3.1 of Module 5 (for b)

**First /Second Semester B.E Degree Examination ,  
Dec.2014 /Jan 2015**  
**Programming in C and Data Structures**  
**(14PCD13/23)**

Time :3Hrs

Max.Marks : 100

*Note : Answer Any FIVE questions , Selecting ONE full question from each part*

**PART-1**

- 1 a. What is pseudocode? Explain with an example. 04 Marks

**Solution :** Refer section 1.3 of Module1

- b. Explain the structure of C program with an example 06 Marks

**Solution :** Refer section 1.4.1 of Module1

- c. Explain any five operators used in C language ? 10 Marks

**Solution :** Refer section 1.7 of Module1

- 2 a. What is a type conversion? Explain two types of type conversion 06 Marks  
with examples

**Solution :** Refer Type Conversion topic at the end of section 1.7 of  
Module1

- b. Write a program in C to find the area and perimeter of a rectangle. 06 Marks

```
#include<stdio.h>
#include<conio.h>
main()
```

```
{ float length, breadth, area, perimeter;
clrscr();
printf("\n Enter the length and breadth of rectangle\n");
scanf("%f %f",&length,&breadth);
area =length *breadth;
perimeter = 2*(length + breadth);
printf("\n The area of rectangle = %f\n", area);
printf("\n The perimeter of rectangle = %f\n",perimeter);
getch();
}
```

- c. Define : i) variable ii) constant iii) associativity iv) precedence 08 Marks

**Solution :**

Refer sections 1.5.4 , and Assocaitivity of operators section of  
Module 1

**Part -2**

- 3 a. What is two way selection statement? Explain if , if else , nested if else and cascaded if –else with examples and syntax. 10 Marks  
**Solution :** Refer section 2.2.1.2 of Module 2
- b. Write a C program that takes three coefficients a, b and c of a quadratic equation :  $(ax^2 +bx+c)$  as input and compute all possible roots and print them with appropriate messages. 10 Marks  
**Solution :**  
Refer Computing Programming Lab Manual ( Part B : Program1)
- 4 a. Explain switch statement with an example. 06 Marks  
**Solution :** Refer section 2.2.1.3 of Module 2
- b. What is a loop? Explain the different loops in C language. 06 Marks  
**Solution :** Refer section 2.3 of Module2
- c. Show how break and continue statements are used in a C program, with example. 04 Marks  
**Solution :** Refer section 2.3.3 of Module2

### PART -3

- 5a. What is an array ? How is a single dimensional array is declared and initialized? 06 Marks  
**Solution :** Refer section 3.1.1 and 3.1.3 of Module3
- b. Write a C program to evaluate the polynomial  $f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$  , for a given value of x and its coefficients using Horner's method . 06 Marks  
**Solution :** Refer Computer Programing Lab Manual Part B: Program Number 4
- c. Explain string manipulation functions , with examples . 08 Marks  
**Solution :** Refer section 3.2.3 of Module3
- 6a. What is function ? Write a function to find the sum of two numbers. 06 Marks  
**Solution:** Refer section 3.3.1 of Module3
- b. Explain the two categories of argument passing techniques, with examples. 06 Marks  
**Solution:** Refer section 3.3.6 of Module3
- c. Write a C function isprime (num) that accepts an integer argument and returns 1 if the argument is a prime or a 0 otherwise. Write a program that invokes this function to generate prime numbers between the given ranges. 08 Marks  
**Solution :** Refer Computer Programming Lab Manual Part B : Program No 9

### Part4

- 7a. What is structure data type? Explain 04 Marks  
**Solution :** Refer Section 4.1 of Module 4
- b. Show how a structure variable is passed as a parameter to a function , 06 Marks with an example.

- Solution :** Refer section 4.1.4 of Module 4
- c Explain the concept of array of structures , with a suitable program. 10 Marks
- Solution :** Refer section 4.1.3 of Module 4
- 8 a What is a file ? Explain fopen () and fclose() functions. 04 Marks
- Solution :** Refer sections 4.2.1 and 4.2.2 of Module4
- b Explain how the input is accepted form a file and displayed. 06 Marks
- Solution :** Refer section 4.2.3
- c Given two text documentary files “Ramayana.in” and “Mahabharatha.in”. Write a C program to create a new file “Karnataka.in” that appends the content of the file “Ramayana.in” to the file “Mahabharatha.in”. Also calculate the number of words and new lines in the output file. 10 Marks
- Solution :**
- ```
#include<conio.h>
#include<stdlib.h>
void main()
{
    FILE *fp1,*fp2,*fp3;
    int nblanks = 0,nlines = 0,nwords;
    char ch;
    clrscr();
    fp1= fopen("Ramayana.in","r");
    if(fp1==NULL)
    {
        printf("\n Ramayana.in is not found\n");
        exit(0);
    }
    fp2 = fopen("Mahabharatha.in","r");
    if(fp2==NULL)
    {
        printf("\n Mahabharatha.in is not found\n");
        exit(0);
    }
    fp3= fopen("Karnataka.in","a");
    while((ch=fgetc(fp1))!=EOF)
    {
        fputc(ch,fp3);
    }
    fclose(fp1);
    while((ch=fgetc(fp2))!=EOF)
    {
        fputc(ch,fp3);
    }
    fclose(fp2);
    fclose(fp3);
    fp3 = fopen("Karnataka.in","r");
    while(ch=fgetc(fp3))!=EOF
    {
        if(ch==' ')nblanks++;
        if(ch=='\n')nlines++;
    }
    fclose(fp3);}
```

```

nwords=nblanks + nlines;
printf("\n Number of Words=%d\n",nwords);
printf("\n Number of new lines =%d\n",nlines);
getch();
}

```

## Part 5

- 9a What is a pointer? Write a program in C to find the sum and mean of all elements in an array. Use pointer technology. 08 Marks  
**Solution :** Refer section 5.1 of Module 5 and Computer Programming Lab Manual : Part B ( Program No : 14)
- b What is preprocessor directive ? Explain #define and #include preprocessor directives. 08 Marks  
**Solution :** Refer Section 5.2.2 of Module 5
- c Explain : 06 Marks
  - i) Dynamic Memory Allocation
  - ii) Malloc function**Solution :** Refer Section 5.1.7 of Module 5
- 10a What are primitive and non-primitive datatypes ? Explain 06 Marks  
**Solution :** Refer section 5.3 of Module5
- b Define queue . Explain it along with its application 06 Marks  
**Solution :** Refer section 5.3.3 of Module 5
- c Explain : 06 Marks
  - i) Abstract data type
  - ii) Stack
  - iii) Linked List

**Solution :** Refer sections 5.3.6, 5.3.2 and 5.3 .4 of Module 5

# Model Question Paper-1

## Programming in C and Data Structures

### (14PCD13/23)

Time :3Hrs

Max.Marks : 100

Note : Answer Any FIVE questions , Selecting ONE full question from each part

### **PART-1**

- 1 a. What is pseudo code? How it is used as a problem solving tool. 06Marks  
**Solution :** Refer section 1.3 of Module1
- b. What is an operator? Explain the arithmetic, relational, logical and assignment operators in C language. 10 Marks  
**Solution :** Refer section 1.7
- c. Write a program in C to print the numbers from 4 to 9 and their squares 04 Marks  
**Solution :** Refer section 1.3 of Module1
- 2 a. Write and explain the basic concepts of a C program 08 Marks  
**Solution :** Refer section 1.4 of Module1
- b. Write the guidelines to use printf() function in C language. 08 Marks  
**Solution :** Refer section 1.6.3 of Module 1
- c. Write a program in C to find the area and perimeter of a circle. 04 Marks  
**Solution :**

```
#include<stdio.h>
#include<conio.h>
main()
{
    float rad,PI = 3.14, area, pm;
    clrscr();
    printf("\nEnter radius of circle: ");
    scanf("%f", &rad);
    area = PI * rad * rad;
    printf("\nArea of circle : %f ", area);
    pm = 2 * PI * rad;
    printf("\n Perimeter of circle : %f ", ci);
    getch();
}
```

### **Part -2**

- 3 a. Explain the two way selection if, if –else , nested if else , cascaded if else in C language with syntax. 08 Marks  
**Solution :** Refer section 2.2.1.2
- b. Explain the switch statement with syntax and example. 08 Marks  
**Solution :** Refer section 2.2.1.3

- c. Design and develop a C program to read a year as an input and find whether it is leap year or not. Also considered end of the centuries  
**Solution :** Refer Computer Programming Lab Manual ,Program Number 3b
- 4 a. Explain the different types of loops in C with syntax and example  
**Solution :** Refer section 2.3.2
- b. Explain the use of break and continue statement in loops with example  
**Solution :** Refer section 2.3.3
- c. Design and develop a C program to reverse an integer number NUM and check whether it is PALINDROME or NOT.  
**Solution :** Refer Computer Programming Lab Manual , Program Number 2.

### PART -3

- 5a What is an array? Explain the declaration and initialization of one and two dimensional arrays with example.  
**Solution :** Refer section 3.1.1 , 3.1.3 and 3.1.9
- b Explain void and parameter less functions in C with examples.  
**Solution :** Refer section 3.3.5
- c Write a C program that:  
i. Implements string copy operations *strcpy(str1,str2)* that copies a string str1 to another str2 without using library function.  
ii. Reads a sentence and prints frequency of each of the vowels and total count of consonants.  
**Solution :** Refer Computer Programming Lab Manual Part B :  
Program No: 9
- 6a Explain any five string manipulation library functions with examples.  
**Solution :** Refer section 3.2.3
- b What is function parameter? Explain different types of parameters in C functions.  
**Solution :** Refer “ Additional Concepts section “ of Module 3
- c Write a C function *isprime(num)* that accepts an integer argument and returns **1** if the argument is prime , a **0** otherwise. Write a C program that invokes this function to generate prime numbers between the given ranges.  
**Solution :** Refer Computer Programming Lab Manual Part B:  
Program No: 10b

### Part4

- 7a What is a structure? Explain the syntax of structure declaration with example.  
**Solution :** Refer section 4.1 of Module4
- b What is a file? Explain how the file open and file close functions

- handled in C?
- Solution :** Refer section 4.2 of Module4
- c Write a C program to maintain a record of “n” student details using an array of structures with four filed (Roll number, Name, Marks and Grade).Each field is of an appropriate datatypes. Print the marks of the student given student name as input.
- Solution :** Refer Computer Programming Lab Manual, Program No: 13
- 8 a Explain array of structures and structure within a structure with examples. 8 Marks
- b. Explain how the structure variable passed as a parameter to a function with example. 6 Marks
- Solution :** Refer section 4.1.4 of Module 4
- c Write a C program to read and display a text from the file. 6 Marks
- ```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
    FILE *fp1,*fp2;
    char ch;
    clrscr();
    fp1= fopen("Input.txt","r");
    if(fp1==NULL)
    {
        printf("\n Input File is not
found\n");
        exit(0);
    }
    fp2= fopen("Output.txt.","w");
    while((ch=fgetc(fp1))!=EOF)
    {
        fputc(ch,fp2);
        fprintf(stdout,"%c",ch);
    }
    fclose(fp1);
    fclose(fp2);
}
```

## Part 5

- 9a What is a pointer? Explain how the pointer variable declared and initialized. 04 Marks
- Solution :** Refer section 5.1.1 of Module 5
- b What is dynamic memory allocation? Write and explain the different dynamic memory allocation functions in C 06 Marks
- Solution :** Refer section 5.1.7 of Module 5
- c What are primitive and non-primitive data types? 04 Marks

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |          |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
|     | <b>Solution :</b> Refer section 1.5.3 of Module1 and section 5.3.1 of Module 5.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |          |
| d   | Write a C program to swap two numbers using call by pointers method<br><b>Solution :</b> Refer section 5.1.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | 06 Marks |
| 10a | Explain the array of pointers with example.<br><b>Solution :</b> Refer section 5.1.3 (Array of Pointers)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 4 Marks  |
| b   | Write and explain any two preprocessor directives in C<br><b>Solution :</b> Refer section 5.2.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | 4 Marks  |
| c   | What is a stack? Explain it with its applications.<br><b>Solution :</b> Refer section 5.3.2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | 4 Marks  |
| d   | Write a C program to read n unsorted numbers to an array of size n and pass the address of this array to a function to sort the numbers in ascending order using bubble sort technique.<br><b>Solution :</b><br><pre>#include&lt;stdio.h&gt; #include&lt;conio.h&gt; void bubblesort(a,n); main() { int n,i,j,temp,a[20]; clrscr(); printf("\n Enter the number of items \n"); scanf("%d",&amp;n); printf("\n Enter the %d items to sort \n",n); for(i=0;i&lt;n;i++) scanf("%d",&amp;a[i]); bubblesort(a,n); printf("\n The sorted items are \n"); for(i=0;i&lt;n;i++) printf("%d\t",a[i]); getch(); } void bubblesort(int a[ ],int n) { int i,j,temp; for(j=1;j&lt;n;j++) { for(i=0;i&lt;n-j;i++) { if(a[i]&gt;=a[i+1]) { temp = a[i] ; a[i] = a[i+1]; a[i+1]= temp ; } } } }</pre> | 8 Marks  |

# Model Question Paper-2

Programming in C and Data Structures

Sub with Code: 15PCD/13/23      Time : 3 Hours      Max.Marks : 80

**Note :** Answer any FIVE full questions , choosing one full question from each module

## Module 1

**1a.** Here is list of possible names for variables in C language. Which are valid names and invalid names? If name is invalid, explain why?

- i) 1999\_space
- ii) \_apple
- iii) iNtEL
- v) one\_2
- v) for
- vi) #12
- vii) i.b.m
- viii) help+me

**Solution :** Refer section 1.8 Ex 31 of Module1

**1b.** What is the purpose of a printf() statement? Explain the formatted printf() along with the respective examples.

**Solution :** Refer section 1.6.3 of Module1

**1c.** Evaluate the given expression by applying associativity and precedence rules  
 $a+2>b||!c&&a==d||a-2<=e$  (where  $a=11, b=6, c=0, d=7, e=5$ )

**Solution :** Refer Topic Evaluation of Expression of section 1.7 , Example1

**2a.** What is an **operator**? Explain the relational, logical, and bitwise operators in C language. Differentiate **between increment and decrement operators** with example.

**Solution :** Refer section 1.7

**2b.** Write a C program to find area of a triangle when we know the lengths of all three of its sides.

**Solution :** Refer section 1.8 , Example 32

**2c.** What is data type? Write a C program that computes the size of int, float, double and char variables.

**Solution :** Refer section 1.5.3 and 1.8 , Example 33

## Module 2

**3a..** Design and develop a C program to reverse a given four digit number and check whether it is a palindrome or not .

**Solution :** Refer Computer Programming Lab Manual , Program 2

**3b.** What is loop? Explain **for () loop** in C along with syntax and flow chart. Give example for the same.

**Solution :** Refer section 2.3.2 of Module2

**3c.** Write a C Program to find GCD of two numbers using ternary operator and for loop.

**Solution :** Refer section 2.4.4 , Ex: 3 of Module2

4a. Explain the Syntax of *nested if...else* statement. Write a C program to find largest of three numbers using *nested if ... else* statement.

**Solution :** Refer section 2.2.1.2 of Module2

4b. Explain the syntax of do-while statement. Write a C program to find the factorial of a number using while loop, where the number n is entered by the user. (Hint: factorial of  $n = 1 * 2 * 3 * \dots * n$ ).

**Solution :** Refer section 2.3.2 and section 2.4.2 , Ex1 of Module2

4c. List four difference between ***while loop*** and ***do while loop*** along with example and syntax

**Solution :** Refer section 2.3.2

### Module 3

5a. Write a C Program to concatenate two strings without using built in function strcat () .

**Solution :**

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
void concat(char[ ], char[ ]);

int main()
{   char s1[50], s2[30];
    clrscr();
    printf("\n Enter String 1 :");
    gets(s1);
    printf("\n Enter String 2 :");
    gets(s2);
    concat(s1, s2);
    printf("\n Concatenated string is :%s", s1);
    return (0);
}

void concat(char s1[], char s2[])
{
    int i, j;
    i = strlen(s1);
    for (j = 0; s2[j] != '\0'; i++, j++)
    {
        s1[i] = s2[j];
    }
    s1[i] = '\0';
}
```

**Output of Program:**

Enter String 1 : Thyagaraju

Enter String 2 : Gowda

Concatenated string is : ThyagarajuGowda

5b. What is an array? Explain different method of initialization of single and two dimensional arrays.

**Solution :** Refer Section 3.1.1 ,3.1.3 and 3.1.9 of Module 3

5c. What is recursion? Write a C program to check a number is a prime number or not using recursion .

**Solution :** Refer 3.3.7 of Module3

6a. What is function? Write a C program to find cube of a Number using function.

**Solution :** Refer sections 3.3.1 and 3.4.4 , Example 2 of Module3

6b. List string manipulation library functions and explain any two of them with example.

**Solution :** Refer section 3.2.3 of Module3

6c. Write a C Program to find greatest number from two dimensional array.

**Solution :** Refer section 3.4.2 , Example3 of Module3

## Module 4

7a. Write a C program to store Name, USN, subject name and IA Marks of students using structure.

**Solution :** Refer section 4.3.1 , Example 1 of Module4

7b. What is a **structure**? Explain the syntax of **structure declaration** with example.

**Solution :** Refer section 4.1.1 of Module4

7c. Explain how the structure variable passed as a parameter to a function with example

**Solution :** Refer section 4.1.4 of Module 4

8a. What is File? Explain following **FILE** operations along with syntax and example.

i)open( ) ii)fclose( ) iii)fgets() iv)fprintf() v)fscanf() vi)fputc () vii)fputs() viii)fgetc()

**Solution :** Refer sections 4.2,4.2.1 and 4.2.2 of Module 4

8b. Explain **array of structures** and **structure within a structure** with Examples.

**Solution :** Refer section 4.1.2 and 4.1.3 of Module4

8c. Write a C program to read and display a text from **the file**.

**Solution :**

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

```

void main()
{
FILE *fp1,*fp2;
char ch;
clrscr();
fp1= fopen("Input.txt","r");
if(fp1==NULL)
{
printf("\n Input File is not found\n");
exit(0);
}
fp2= fopen("Output.txt.","w");
while((ch=fgetc(fp1))!=EOF)
{
putc(ch,fp2);
fprintf(stdout,"%c",ch);
}
fclose(fp1);
fclose(fp2);
}

```

## Module 5

9a. What is pointer? Give the advantages and disadvantages of pointer data type

**Solution :** Refer section 5.1.1 of Module5 and Page no 426.

9b. Explain malloc ( ), calloc ( ) functions with examples.

**Solution :** Refer section 5.1.7 of Module5

9c. What are ***primitive and non-primitive data types?***

**Solution :** Refer section 1.5.3 of Module1 and section 5.3.1 of Module 5.

10a. Explain stack and queue related terms and give their applications

**Solution :** Refer sections 5.3.2 and 5.3.3

10b. Write and explain any five preprocessor directives in C.

**Solution :** Refer section 5.2.2

10c. Differentiate between ***Call by Reference and Call by Value.*** Write a C program to swap two numbers using ***call by pointers*** method

**Solution :** Refer sections 3.3.6 and 5.1.2

**First /Second Semester B.E Degree Examination ,  
Dec.2015 /Jan 2016**  
**Programming in C and Data Structures  
(14PCD13/23)**

Time :3Hrs

Max.Marks : 100

Note : Answer Any FIVE questions , Selecting ONE full question from each part

**PART-1**

- |      |                                                                                                 |          |
|------|-------------------------------------------------------------------------------------------------|----------|
| 1 a. | Explain the structure of “C” program with example.                                              | 08 Marks |
|      | <b>Solution :</b> Refer section 1.4.1 of Module1                                                |          |
| b.   | Explain scanf() and printf() function in ‘C’ language with syntax and examples.                 | 08 Marks |
|      | <b>Solution :</b> Refer sections 1.6.3 and 1.6.4 of Module 1                                    |          |
| c.   | Write a C program to find area of a circle.                                                     | 04Marks  |
|      | <b>Solution :</b> Refer section 1.8 , Example 9 of Module1                                      |          |
| 2 a. | What is an algorithm? Write an algorithm to find largest of 3 natural numbers.                  | 08 Marks |
|      | <b>Solution :</b> Refer section 1.2 , Example 10 of Module1                                     |          |
| b.   | Explain the following operators in ‘C’ language :<br>i) Relational ii) Logical iii) Conditional | 08 Marks |
|      | <b>Solution :</b> Refer section 1.7 of Module1                                                  |          |
| c.   | What is an identifier ? Give any 5 rules that are to be followed , while declaring a variable.  | 04 Marks |
|      | <b>Solution :</b> Refer section 1.5.2 of Module1                                                |          |

**Part -2**

- |      |                                                                                               |          |
|------|-----------------------------------------------------------------------------------------------|----------|
| 3 a. | Explain the ELSE ---IF ladder with syntax and example.                                        | 08 Marks |
|      | <b>Solution :</b> Refer section 2.2.1.2 of Module2                                            |          |
| b.   | List the types of loops. Explain the working of any one type of loop with syntax and example. | 08 Marks |
|      | <b>Solution :</b> Refer section 2.3 ,2.3.1 and 2.3.2 of Module2                               |          |
| c.   | Write a program to read a year as an input and find whether it is a LEAP year or not.         | 04 Marks |
|      | <b>Solution :</b> Refer Lab Manual , Part B, Program 3b                                       |          |
| 4 a. | Explain SWITCH statement , with syntax and example.                                           | 08Marks  |
|      | <b>Solution :</b> Refer section 2.2.1.3 of Module2                                            |          |
| b.   | Differentiate between WHILE and DO –WHILE loops.                                              | 06 Marks |
|      | <b>Solution :</b> Refer section 2.3.2 of Module2                                              |          |
| c.   | Write a program to find reverse of a number and check whether it is a PALINDROME or NOT.      | 06 Marks |
|      | <b>Solution :</b> Refer section Computer Programming Lab Manual ,Part B , Program2            |          |

## PART -3

5a What is an array ? Explain different ways of initializing an array with examples. 07 Marks

**Solution :** Refer section 3.1 of Module 3

b What are the advantages of using user defined functions ? 06 Marks

Advantages of user defined functions

1. User defined functions helps to decompose the large program into small segments which makes programmer easy to understand, maintain and debug.
2. If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
3. Programmer working on large project can divide the workload by making different functions.

c Write a program to read a sentence and print the frequencies of each vowel and total count of CONSONANTS. 07 Marks

**Solution :** Refer Programming Lab Manual , Part B , Program 9b

6a Explain the different types of arrays with syntax and examples. 07 Marks

**Solution :** Refer sections 3.1,3.1.3 and 3.1.9

b Explain any 4 string manipulating functions with examples 08 Marks

**Solution :** Refer section 3.2.3

C Define the following : i) Actual parameter ii) Formal parameter  
iii) Global variable iv) Local Variable 05 Marks

**Solution :** Refer section Additional concepts of Module3

## Part4

7a Define a STRUCTURE.Explain structure with syntax and examples. 05 Marks

**Solution :** Refer section 4.1 of Module 4

b What is a FILE ? Explain any 2 FILE functions with example . 05 Marks

**Solution :** Refer section 4.2 of Module 4

c Write a program to maintain a record of “n” student details using an array of structures with four field ( Roll number , Name , Marks and Grade). Each field is of an appropriate data type. Print the marks of the student given student name as input.

**Solution :** Refer Lab Manual , Part B , Program 13

8 a Differentiate between STRUCTURES and UNIONS. 05 Marks

**Solution :** Refer section Additional Concepts of Module4

b. Explain the various MODES in which a FILE can be created successfully . 05 Marks

**Solution :** Refer section 4.2.2 of Module4

- c Given 2 university information files “studentname.txt” and “usn.txt” that contains students Name and USN respectively . Write a program to create a new file called “output.txt” and copy the contents of files “studentname.txt” and “usn.txt” into output file .

**Solution :** Refer Lab Manual , Part B , Program 12

### Part 5

- 9a Define pointer variable. Explain with an example the declaration and initialization of pointer variable.

**Solution :** Refer section 5.1.1 of Module 1

- b What are primitive and non-primitive data types ? Give Examples

**Solution :** Refer sections 1.5.3 (Module1) and 5.3.1(Module5)

- c Write a program using pointers to compute sum,mean and standard deviation of all elements stored in an array of “n” real numbers.

**Solution :** Refer Lab manual , Part B, Program 14

- 10a Explain any two pre- processor directives in “C” language

**Solution :** Refer section 5.2.2

- b What I STACK ? Explain its applications.

**Solution :** Refer section 5.3.2

- c What is QUEUE ? Explain with example.

**Solution :** Refer section 5.3.3

- d Write a program to swap 2 numbers using call by reference method.

**Solution :** Refer section 5.1.2

**First /Second Semester B.E Degree Examination ,  
Dec.2015 /Jan 2016**  
**Programming in C and Data Structures  
(15PCD13/23)**

Time :3Hrs

Max.Marks : 80

*Note : Answer Any FIVE questions , Choosing ONE full question from each Module*

**Module-1**

- 1 a. What is variable ? Explain the rules for constructing variables in C 06 Marks language. Give examples for valid and invalid variables.
- b. Evaluate the following expressions : 04 Marks

i)  $100\% 20 <=20-5+100\%10 -20 == 5>=1 !=20$

**Solution :**  $100\% 20 <=20-5+100\%10 -20 == 5>=1 !=20$  (operator %)

$0<=20-5+100\%10-20==5>=1!=20$  ( operator %)

$0<=20-5 + 0-20==5>=1!=20$  ( operator - )

$0<=15+0-20==5>=1!=20$  (operator +)

$0<=15-20==5>=1!=20$  (operator -)

$0<=-5==5>=1!=20$  ( Operator <= )

$0==5>=1!=20$  (Operator >= )

$0==!=1!=20$  (Operator ==)

$0!=20$  (Operator !=)

1

ii)  $a+=b*=c=5$  where  $a =3$   $b =5$  and  $c =8$

**Solution :** (Right to left associativity )

$a+=b*= (8-5) \quad // c-=5 \Rightarrow c = c-5$

$a+=b*=3$

$a+=(5*3) \quad // b*=3 \Rightarrow b= b*3)$

$a+=15$

$a= a+15$

$a= 3+15$

$a =18$

- c. Write a C program to find the area and perimeter of a rectangle . 06 Marks

**Solution :**

```
#include<conio.h>
#include<stdio.h>
main()
{
    float area,perimeter, length, breadth ;
```

```

clrscr();
printf("n Enter the length and breadth \n");
scanf("%f %f",&length , breadth);
area = length * breadth ;
perimeter = 2*(length +breadth);
printf ("n The area of reactangle = %f ",area);
printf("n The perimeter of rectangle = %f ",perimeter);
getch();
}

```

- 2 a. Write a C program which takes as input p,t,r . Compute the simple interest and display the result. 06 Marks

**Solution :** Refer section 1.8 (Example 34)

- b. Convert the following mathematical expression into C expressions 04 Marks

:

$$i) \frac{x}{b+c} + \frac{y}{b-c}$$

**Solution :**  $x/(b+c) + y/(b-c)$

$$ii) a + \frac{b(ad+e)}{b-c} - \frac{c}{d}$$

**Solution :**  $a + (b*(a*d + e))/(b-c) - c/d$

- c. What is the value of 'x' in following code segments? Justify your answers : 06 Marks

|              |                  |
|--------------|------------------|
| i) int a,b ; | ii) int a,b;     |
| float x;     | float x;         |
| a = 4;       | a = 4;           |
| b = 5 ;      | b = 5;           |
| x = b/a;     | x = (float) b/a; |

**Solution :**

$$i) x = b/a = 5/4 = 1.000000.$$

**Justification :** Since a and b are integer , b/a will results in integer division and the resultant value will be converted into float as x is floating pointing number (implicit type cast).

$$ii) x = (\text{float}) b/a = 5.000000/4 = 1.250000$$

**Justification:** This is the case of explicit type casting where in the integer b will gets converted into floating point number resulting in mixed mode arithmetic operation.

## Module-2

- 3 a. Explain the syntax of do while statement . Write a C program to find the factorial of a number using do while , where the number n is entered by user. 08 Marks

**Solution :**

**Refere section 2.3.2 and the following program**

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

```

#include<conio.h>
main()
{ int n, f,i ;
clrscr();
printf("\n Enter the number \n");
scanf("%d",&n);
f=1;
i=1;
do
{ f= f*i;
i++;
} while(i<=n);
printf("\n The factorial of a given number = %d \n",f);
getch();
}

```

- b. What is two way selection statement ? Explain if,if else and cascaded if else with examples. 08 Marks

**Solution :**

Refer Section 2.2.1.2

- 4 a. Write a C program that takes from use an arithmetic operator ('+', '-','\*' or '/') and two operands . Perform the corresponding arithmetic operation on the operands using switch statement. 08 Marks

**Solution :**

```

#include<stdio.h>
#include<conio.h>
main()
{ int n1,n2,result;
char op;
clrscr();
printf("\n Enter two operands (numbers )\n");
scanf("%d %d",&n1,&n2);
printf"\n Enter the arithmetic operator ('+', '-' '*' or '/' )\n";
scanf(" %c", &op);
switch(op)
{
    case '+': result = n1+n2;
        printf("\n Result : %d +%d = %d\n",n1,n2,result);
        break;
    case '-': result = n1- n2;
        printf("\n Result : %d -%d = %d\n",n1,n2,result);
        break;
    case '*': result = n1*n2;
        printf("\n Result : %d *%d = %d\n",n1,n2,result);
        break;
    case '/': if (n2==0)
        { printf("\n Divide by Zero Error\n");
        break;
        }
    result = n1/ n2;
}

```

```

        printf("\n Result : %d / %d = %d\n",n1,n2,result);
        break;
    default : printf("\n Invalid Entry \n");
}
getch();
}

```

- b. What is an array ?How to declare and initialize the two dimensional array? 08 Marks

**Solution :** Refer section 3.1,3.1.3 and 3.1.9 of Module3

### Module-3

- 5a What is a function ? Write a C program tofind the cube of a number 05 Marks using function .

- Solution :** Refer section 3.3.1 and 3.4.4 (Example 2) of Module 3  
b Write a C program to check a number is a prime or not using 05 Marks recursion.

**Solution :** Refer section 3.4.4 (Example 9) of Module 3

- c Write a program to replace each constant in a string with the next one except letter ‘z’ ‘Z’ and ‘a’, ‘A’. Thus the string “programming in C is fun” should be modified as “Qsphsannjohjo D jt gvo”. 06 Marks

```

Solution :
#include<stdio.h>
#include<conio.h>
main()
{
    char str[100],ch;
    int i;
    clrscr();
    printf("\n Enter the string \n");
    gets(str);
    printf("\n The given string is %s\n",str);
    for(i=0;i< strlen(str);i++)
    {
        ch =str[i];
        if(ch =='a'||ch=='A'||ch=='Z'||ch=='z'||ch= ' ')
        {
            str[i] =str[i];
        }
        else
        {
            str[i] = str[i] +1;
        }
    }
    str[i] = '\0';
    printf ("\n The resultant string is %s\n",str);
    getch();
}

```

- 6a Write a C program to sort the element by passing array as function 08 Marks argument.

**Solution :**

```
#include<stdio.h>
#include<conio.h>
void sort(a,n);
main()
{ int n,i,j,temp,a[20];
 clrscr();
 printf("\n Enter the number of items \n");
 scanf("%d",&n);
 printf("\n Enter the %d items to sort \n",n);
 for(i=0;i<n;i++)
 scanf("%d",&a[i]);
 sort(a,n);
 printf("\n The sorted items are \n");
 for(i=0;i<n;i++)
 printf("%d\t",a[i]);
 getch();
}
void sort(int a[ ] ,int n)
{ int i,j,temp;
 for(j=1;j<n;j++)
 { for(i=0;i<n-j;i++)
 {
 if(a[i]>=a[i+1])
 {
 temp = a[i];
 a[i] = a[i+1];
 a[i+1]= temp ;
 }
 }
}
```

- b Write a C program to concatenate two strings without using built in function strcat() . 08 Marks

**Solution :**

Refer section 3.4.3 (Example 6) of Module3

## Module-4

- 7a What is structure ? Explain the C syntax of structure declaration with example. 05 Marks

**Solution :** Refer section 4.1.1 of Module4

- b Write a C program to pass structure variable as function arguments . 07 Marks

**Solution :** Refer section 4.1.4 of Module4

- c Explain fopen() and fclose() functions. 04 Marks

**Solution :** Refer section 4.2 of Module4

- 8 a Write a C program to store and print name,USN,subject and IA marks of students using structure. 08 Marks  
**Solution :** Refer section 4.3.1 (Example1) of Module4
- b. Explain fputc() , fputs() , fgetc() and fgets() functions with syntax. 08 Marks  
**Solution :** Refer section 4.2 of Module4

## **Module-5**

- 9a What is pointer ? Write a C program to find the sum and mean of all elements in an array using pointer. 08 Marks  
**Solution :** Refer section 5.1.1 of Module5 and Lab Manual , Program 14
- 9b What is stack? Explain its operations with examples. 08 Marks  
**Solution :** Refer section 5.3.2 of Module5
- 10 a Write a C program to swap two numbers using call by address . 06 Marks  
**Solution :** Refer section 5.1.2 of Module5
- b Explain any five preprocessor directives in C. 05 Marks  
**Solution :** Refer section 5.2.2 of Module5
- c What are primitive and non primitive data types ? Explain with examples. 05 Marks  
**Solution :** Refer section 1.5.3 of Module1 and section 5.3.1 of Module 5.

## **Question Paper pattern for 2015 – 2016 Batch Students of VTU**

- The question Paper will be of Maximum 80 Marks.
- The question paper will have ten questions.
- Each full Question consisting of 16 marks
- There will be 2 full questions (with a maximum of four sub questions) from each module.
- Each full question will have sub questions covering all the topics under a module.
- The students will have to answer 5 full questions , selecting one full questions from each module.

## 7. Basic Mathematical Formulae and Definitions

1. **The sum of first n natural numbers** =  $\sum n = n(n+1)/2$
2. **The sum of squares of first n natural numbers** =  $\sum n^2 = n(n+1)(2n+1)/6$
3. **The sum of cubes of first n natural numbers**  $\sum n^3 = n^2(n+1)^2/4$
4. i) **The sum of first n even positive integers**  $\sum 2n = n(n + 1)$ .  
ii) **The sum of first n odd positive integers** =  $\sum(2n - 1) = n^2$
5. **Average** = [Sum of observations / Number of observations]
6. **Suppose a man covers a certain distance** at x kmph and an equal distance at y kmph. Then, the average speed during the whole journey is  $[2xy / (x + y)]$  kmph.
7. **Natural Numbers** : Counting numbers 1, 2, 3, 4, 5, .. are called natural numbers.
8. **Whole Numbers** : All counting numbers together with zero form the set of whole numbers. Thus,
  - I. 0 is the only whole number which is not a natural number.
  - II. Every natural number is a whole number.
9. **Triangular Numbers** : 1, 3, 6, 10, 15, 21, 28, 36, 45,-----
10. **Fibonacci sequence** : 0,1,1,2,3,5,8,13,21,34,-----

- 11.i. **Geometric sequence** : Ex:2,4,8,16,32,64,128,256,-----  
ii. **Arithmetic sequence** : Ex:1,4,7,10,13,16,19,22,25,-----

## 12. SIMPLE INTEREST

1. **Principal** : The money borrowed or lent out for a certain period is called the principal of the sum.
2. **Interest** : Extra money paid for using other's money is called interest.
3. **Simple Interest (S.I.)** : If the interest on a sum borrowed for a certain period is reckoned uniformly, then it is called simple interest.

Let Principal = P, Rate = R% per annum (p.a.) and Time = T years,  
Then,

$$(i) S.I. = [P * R * T / 100]$$

$$(ii) P = [100 * S.I. / R * T]$$

$$R = [100 * S.I. / P * T] \text{ and } T = [100 * S.I. / P * R]$$

### 13.The Compound Interest Equation

$$P = C (1 + r/n)^{nt}$$

where

P = future value

C = initial deposit

r = interest rate (expressed as a fraction: eg. 0.06)

n = # of times per year interest is compounded

t = number of years invested

#### Simplified Compound Interest Equation

When interest is only compounded once per year (n=1), the equation simplifies to:

$$P = C (1 + r)^t$$

#### Continuous Compound Interest

When interest is compounded continually (i.e. n  $\rightarrow \infty$ ), the compound interest equation takes the form:

$$P = C e^{rt}$$

### 14.TIME AND DISTANCE

1. Speed = [Distance/Time],  
Time=[Distance/Speed],  
Distance = (Speed\*Time)
2. x km/hr = [x\*5/18] m/sec.
3. If the ratio of the speeds of A and B is a:b, then the ratio of the times taken by them to cover the same distance is 1/a : 1/b or b:a.
4. x m/sec = [x\*18/5] km/hr.
5. Suppose a man covers a certain distance at x km/hr and an equal distance at y km/hr. then, the average speed during the whole journey is  $[2xy/x+y]$  km/hr.

### 15.Cost Price , Selling Price, Loss and Profit

**Cost Price :** The price at which an article is purchased, is called its cost price, abbreviated as C.P.

**Selling Price :** The price at which an article is purchased, is called its cost price, abbreviated as C.P.

**Profit or Gain :** The price at which an article is purchased, is called its cost price, abbreviated as C.P.

**Loss :** If S.P is less than C.P., the seller is said to have incurred a loss.

1. Gain = (S.P.) - (C.P.)
2. Loss or gain is always reckoned on C.P.
3. gain% = [Gain\*100/C.P.]
4. Loss = (C.P.) - (S.P.)
5. Loss% = [Loss\*100/C.P.]
6. S.P. =  $(100+Gain\%)/100 * C.P.$
7. S.P. =  $(100-Loss\%)/100 * C.P.$
8. C.P. =  $100/(100+Gain\%) * S.P.$
9. C.P. =  $100/(100-Loss\%) * S.P.$
10. If an article is sold at a gain of say, 35%, then S.P. = 135% of C.P.
11. If an article is sold at a loss of say, 35%, then S.P. = 65% of C.P

## 16. Formulas - Perimeter, Circumference, Area, Surface Area, Volume - of 2D and 3D Shapes

- **2D –geometric shape:** square, rectangle, triangle, circle, trapezoid, parallelogram, etc.



- **Composite figure:** a combination of 2 or more geometric shapes



- **3D –geometric shape:** rectangular prism, triangular prism, cylinder, cone, sphere, pyramid, etc.



- **Perimeter of a Rectangle:** distance around the rectangle (add up the lengths of all 4 sides)

$$\begin{array}{c} L \\ \boxed{\phantom{LW}} \\ W \end{array} \quad P = L + L + W + W \quad P = 2(L + W)$$

- **Area of a Rectangle:** number of square units that it covers (multiply length x width)

$$A = L \times W$$

- **Perimeter** of a **Square**: distance around the square (add up length of all 4 sides)

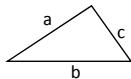


$$P = L + L + L + L \quad \text{or} \quad P = 4L$$

- **Area** of a **Square**: number of square units that it covers (multiply length x length)

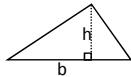
$$A = L \times L \quad \text{or} \quad A = L^2$$

- **Perimeter** of a **Triangle**: distance around the triangle (add up the lengths of all 3 sides)



$$P = a + b + c$$

- **Area** of a **Triangle**: number of square units that it covers ( $\frac{1}{2}$  x base of triangle x height of triangle)



$$A = \frac{\text{base} \times \text{height}}{2}$$

$$A = \frac{b \cdot h}{2}$$

- **Circumference** of a **Circle**: distance around the circle (the “perimeter” of the circle)



$$C = 2\pi r \quad \text{or}$$

(r is radius)

$$C = \pi d$$

(d is diameter)

- **Area** of a **Circle**: number of square units that it covers ( $\pi \times \text{radius} \times \text{radius}$ )

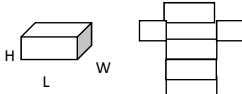
$$A = \pi r^2$$

$$A = \pi \times r \times r$$

- **Surface Area** of a **Rectangular Prism**: add up the areas of all 6 sides of the prism

$$SA = (2 \times L \times W) + (2 \times L \times H) + (2 \times W \times H)$$

$$\text{or } SA = 2(LW + LH + WH)$$



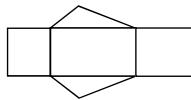
- **Volume** of a **Rectangular Prism**: amount of space it takes up (area of base x height of prism)



$$V = \text{area of the rectangle base} \times \text{height of the prism}$$

$$V = L \times W \times H$$

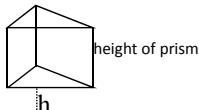
**Surface Area** of a **Triangular Prism**: add up the area of all 5 sides (2 triangles, 3 rectangles) of the prism



$$SA = \frac{\text{base} \times \text{height}}{2} + \frac{\text{base} \times \text{height}}{2} + LW + LW + LW$$

$$SA = \text{area triangle} + \text{area triangle} + \text{area rectangle} + \text{area rectangle} + \text{area rectangle}$$

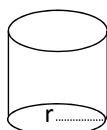
- **Volume** of a **Triangular Prism**: amount of space it takes up (area of base triangle x height of prism)



$$V = \text{area of base triangle} \times \text{height of prism}$$

$$V = \frac{\text{base} \times \text{height}}{2} \times \text{height of prism} \quad V = \frac{bh}{2} \times h \text{ of prism}$$

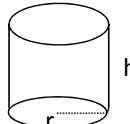
- **Surface Area** of a **Cylinder**: area of the 2 circles and the area of the rectangle



$$SA = 2\pi r^2 + 2\pi r \times \text{height of cylinder}$$

$$SA = (\text{area of 2 circles } 2\pi r^2) + (\text{area of rectangle } 2\pi r \times h)$$

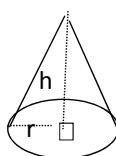
- **Volume** of a **Cylinder**: amount of space it takes up



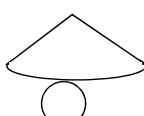
$$V = \pi r^2 \times h$$

$$V = (\text{area of the base circle } \pi r^2) \times (\text{height of the cylinder } h)$$

- **Volume** of a **Cone**: amount of space it takes up



$$V = \frac{1}{3}\pi r^2 \times h \quad \text{OR} \quad V = \frac{\pi r^2 h}{3}$$



$$V = (1/3) \times (\text{area of the base circle}) \times (\text{height of the cone})$$

- **Surface Area** of a Sphere: amount of space it takes up

$$SA = 4\pi r^2$$



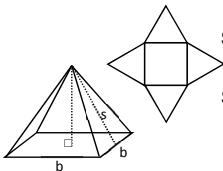
$SA = 4 \times \text{area of the circle cross-section } (\pi r^2) \text{ at the equator of the sphere}$

- **Volume** of a Sphere: amount of space it takes up



$$V = \frac{4}{3}\pi r^3$$

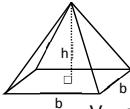
- **Surface Area** of a Square-Based Pyramid: area of all the faces (square base, and 4 triangles)



$$SA = b^2 + 4\left(\frac{bh}{2}\right) \quad \text{where } s = \text{(slant) height of a triangular face}$$

$SA = (\text{area of the square base} + \text{areas of 4 side triangles})$

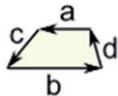
- **Volume** of a Square-Based Pyramid: amount of space it takes up



$$V = \frac{1}{3}b^2h$$

$V = \text{one-third the area of the square base } (b^2) \times \text{height of the pyramid } (h)$

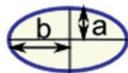
- **Area of a Trapezoid :**



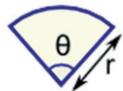
$$\text{Area} = \frac{1}{2} * (a+b) * h$$

$$\text{Perimeter} = a+b+c+d$$

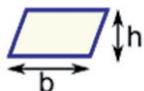
**Area of Ellipse :**  $\text{Area} = \pi ab$



**Area of sector :** Area =  $\frac{1}{2} \times r^2 \times \theta$  , where as r = radius and  $\theta$  = angle in radians



**Area of Parallelogram :** Area =  $b \times h$  , where as b = base and h = vertical height



**17. Regular Solids:** Tetrahedron – 4 faces , Cube – 6 faces ,Octahedron – 8 faces ,Dodecahedron – 12 faces, Icosahedron – 20 faces

## 18. Types of Triangles:

### By Sides:

- Scalene – no congruent sides
- Isosceles – 2 congruent sides
- Equilateral – 3 congruent sides

### By Angles:

- Acute – all acute angles
- Right – one right angle
- Obtuse – one obtuse angle
- Equiangular – 3 congruent angles( $60^\circ$ )
- Equilateral  $\leftrightarrow$  Equiangular

## 19. Heron's formula to find the area of triangle in terms of sides

A =  $\sqrt{s(s-a)(s-b)(s-c)}$ . Where as  $s = (a+b+c)/2$

**20. Unit Conversion Tables for Lengths & Distances :The notation  
1.23E - 4 stands for  $1.23 \times 10^{-4} = 0.000123$ .**

| from \\ to | feet        | inches      | meters   | miles            | yards       |
|------------|-------------|-------------|----------|------------------|-------------|
| foot       |             | 12          | 0.3048   | (1/5280)         | (1/3)       |
| inch       | (1/12)      |             | 0.0254   | (1/63360)        | (1/36)      |
| meter      | 3.280839... | 39.37007... |          | 6.213711...E - 4 | 1.093613... |
| mile       | 5280        | 63360       | 1609.344 |                  | 1760        |
| yard       | 3           | 36          | 0.9144   | (1/1760)         |             |

mile = 1760 yards = 5280 feet

yard = 3 feet = 36 inches

foot = 12 inches

inch = 2.54 centimeters

## 21. Series

### Arithmetic Progression

$$S_n = a + (a + d) + (a + 2d) + \dots + [a + (n-1)d] = (n/2) * [2a + (n-1)*d]$$

### Geometric Progression

$$S_n = a + ar + ar^2 + \dots + ar^{n-1} = a(1-r^n) / (1-r)$$

### Binomial expansion

$$(1+x)^n = 1 + \frac{nx}{1!} + \frac{n(n-1)x^2}{2!} + \frac{n(n-1)(n-2)x^3}{3!} \dots$$

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, \quad -\infty < x < \infty$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} + \dots,$$

$$\cos x = (e^{ix} + e^{-ix})/2 = 1 - x^2/2! + x^4/4! - x^6/6! + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} \dots$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \dots$$

$$\tan x = x + \frac{x^3}{3} + \frac{2x^5}{15} + \dots,$$

$$\tan^{-1} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} \dots$$

$$\sin^{-1} x = x + \frac{1.x^3}{2.3} + \frac{1.3.x^5}{2.4.5} + \dots$$

## 22. Matrices

- A Matrix is an array of numbers:

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$$

A Matrix

(This one has 2 Rows and 3 Columns)

**Adding :** To add two matrices: add the numbers in the matching positions:

$$\begin{bmatrix} 3 & 8 \\ 4 & 6 \end{bmatrix} + \begin{bmatrix} 4 & 0 \\ 1 & -9 \end{bmatrix} = \begin{bmatrix} 7 & 8 \\ 5 & -3 \end{bmatrix}$$

These are the calculations:

$$3+4=7 \quad 8+0=8$$

$$4+1=5 \quad 6-9=-3$$

## Subtracting

To subtract two matrices: subtract the numbers in the matching positions:

$$\begin{bmatrix} 3 & 8 \\ 4 & 6 \end{bmatrix} - \begin{bmatrix} 4 & 0 \\ 1 & -9 \end{bmatrix} = \begin{bmatrix} -1 & 8 \\ 3 & 15 \end{bmatrix}$$

These are the calculations:

$$3-4=-1 \quad 8-0=8$$

$$4-1=3 \quad 6-(-9)=15$$

### Multiplying a Matrix by Another Matrix

Here we compute the dot product of row vector with column vector.  
Consider an example: To work out the answer for the **1st row** and **1st column**:

**"Dot Product"**

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 \end{bmatrix}$$

The "Dot Product" is where we **multiply matching members**, then sum up:

$$(1, 2, 3) \cdot (7, 9, 11) = 1 \times 7 + 2 \times 9 + 3 \times 11 = 58$$

We match the 1st members (1 and 7), multiply them, likewise for the 2nd members (2 and 9) and the 3rd members (3 and 11), and finally sum them up. Want to see another example? Here it is for the 1st row and **2nd column**:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \end{bmatrix}$$

$$(1, 2, 3) \cdot (8, 10, 12) = 1 \times 8 + 2 \times 10 + 3 \times 12 = 64$$

We can do the same thing for the **2nd row** and **1st column**:

$$(4, 5, 6) \cdot (7, 9, 11) = 4 \times 7 + 5 \times 9 + 6 \times 11 = 139$$

And for the **2nd row** and **2nd column**:

$$(4, 5, 6) \cdot (8, 10, 12) = 4 \times 8 + 5 \times 10 + 6 \times 12 = 154$$

And we get:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix} \quad \checkmark$$

When we do multiplication:

- The number of **columns of the 1st matrix** must equal the number of **rows of the 2nd matrix**.
- And the result will have the same number of **rows as the 1st matrix**, and the same number of **columns as the 2nd matrix**.

### Transposing

To "transpose" a matrix, swap the rows and columns. We put a "T" in the top right-hand corner to mean transpose:

**Identity Matrix:** It is a square matrix whose elements other than the principal diagonal elements is 0. All the elements of principal diagonal is 1.

**Upper Triangular Matrix :** A matrix with all-zero entries below the top-left-to-lower-right diagonal ("the diagonal") is called "upper triangular".

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

A matrix with non-zero entries only on the diagonal is called "**diagonal**".

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

A diagonal matrix whose non-zero entries are all 1's is called **an "identity" matrix**.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

**Trace of the Matrix :** Here trace of the matrix is the sum of the elements of the main diagonal i.e the diagonal from the upper left to the lower right of a matrix.

**Normal of the matrix:** is the square root of the sum of all the elements.

## 23.Temperature Conversion Formula

| From             | To Fahrenheit             | To Celsius       | To Kelvin                 |
|------------------|---------------------------|------------------|---------------------------|
| Fahrenheit (F)   | F                         | $(F - 32) * 5/9$ | $(F - 32) * 5/9 + 273.15$ |
| Celsius (C or °) | $(C * 9/5) + 32$          | C                | $C + 273.15$              |
| Kelvin (K)       | $(K - 273.15) * 9/5 + 32$ | $K - 273.15$     | K                         |

## **8.Computer Programming Laboratory Manual**

# **COMPUTER PROGRAMMING LABORATORY**

**[As per Choice Based Credit System (CBCS) scheme]  
(Effective from the academic year 2015 -2016)  
SEMESTER - I/II**

**Sub Code : 15CPL16/15CPL26  
Exam Marks : 80**

**IA Marks : 20  
Credits : 02**

**Course objectives:** To provide basic principles C programming language. To provide design & develop of C programming skills. To provide practical exposures like designing flowcharts, algorithms, how to debug programs etc.

## **PART-A**

**Demonstration of Personal Computer and its Accessories:** Demonstration and Explanation on Disassembly and Assembly of a Personal Computer by the faculty in charge. Students have to prepare a write up on the same and include it in the Lab record and evaluated.

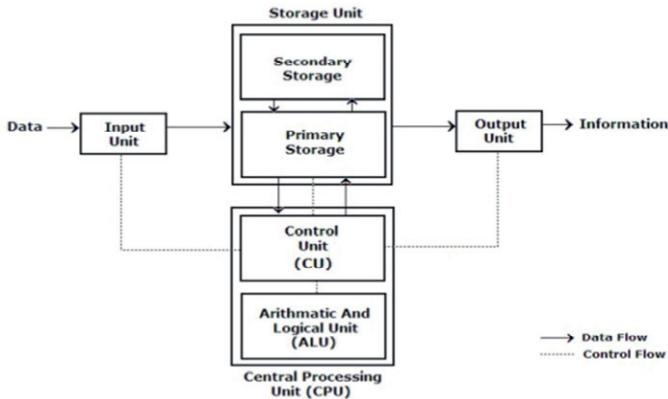
**Laboratory Session-1: Write-up on Functional block diagram of Computer, CPU, Buses, Mother Board, Chip sets, Operating System & types of OS, Basics of Networking & Topology and NIC.**

### **Solution:**

Computer is an electronic device which is capable, of receiving information (data) in a particular form and of performing a sequence of operations in accordance with a predetermined but variable set of procedural instructions (program) to produce a result in the form of information or signals. A computer as shown in Fig. Performs basically five major computer operations or functions irrespective of their size and make. These are

- 1) It accepts data or instructions through different modes of input,
- 2) It stores data,
- 3) It can process data as required by the user,
- 4) It gives results in the form of output, and
- 5) It controls all operations inside a computer.

## Block diagram of computer



The various Computer operations are discussed below :

**1. Input:** Inputting is the process of entering data and programs into the computer system. The computer is an electronic machine like any other machine which takes inputs as raw data and performs some processing giving out processed data. Therefore, the input unit takes data from user to the computer in an organized manner for processing.

**2. Storage:** The process of saving data and instructions permanently is known as storage. Data has to be fed into the system before the actual processing starts. It is because the processing speed of Central Processing Unit (CPU) is so fast that the data has to be provided to CPU with the same speed. Therefore the data is first stored in the storage unit for faster access and processing. The storage unit or the primary storage of the computer

system is designed to do the above functionality. It provides space for storing data and instructions. The storage unit performs the following major functions:

- All data and instructions are stored here before and after processing.
- Intermediate results of processing are also stored here.

**3. Processing:** The task of performing operations like arithmetic and logical operations is called processing. The Central Processing Unit (CPU) takes data and instructions from the storage unit and makes all sorts of calculations based on the instructions given and the type of data provided. It is then sent back to the storage unit.

**4. Output:** This is the process of producing results from the data for getting useful information. Similarly the output produced by the computer after processing must also be kept somewhere inside the computer before being given to user in human readable form. Again the output is also stored inside the computer for further processing.

**5. Control:** The manner how instructions are executed and the above operations are performed. Controlling of all operations like input, processing, and output are performed by control unit. It takes care of step by step processing of all operations inside the computer.

## FUNCTIONAL UNITS

In order to carry out the operations mentioned in the previous section the computer allocates the task between its various functional units. The computer system is divided into three separate units for its operation. They are

- 1) Arithmetic logical unit
- 2) Control unit.
- 3) Central processing unit.

### **Arithmetic Logical Unit (ALU)**

After you enter data through the input device it is stored in the primary storage unit. The actual processing of the data and instruction are performed by Arithmetic Logical Unit. The major operations performed by the ALU are addition, subtraction, multiplication, division, logic and comparison. Data is transferred to ALU from storage unit when required. After processing the output is returned back to storage unit for further processing or getting stored.

### **Control Unit (CU)**

The next component of computer is the Control Unit, which acts like the supervisor seeing that things are done in proper fashion. Control Unit is responsible for co-ordinating various operations using time signal. The control unit determines the sequence in which computer programs and instructions are executed. Things like processing of programs stored in the main memory, interpretation of the instructions and issuing of signals for other units of the computer to execute them. It also acts as a switch board operator when several users access the computer simultaneously. Thereby it coordinates the activities of computer's peripheral equipment as they perform the input and output.

### **Central Processing Unit (CPU)**

The ALU and the CU of a computer system are jointly known as the central processing unit. The CPU is known as the brain of any computer system. It is just like brain that takes all major decisions, makes all sorts of calculations

and directs different parts of the computer functions by activating and controlling the operations.

**Bus:** When referring to a computer, the bus also known as the address bus, data bus, or local bus is a data connection between two or more devices connected to the computer. For example, a bus enables a computer processor to communicate with the memory or a video card to communicate with the memory.

## Types of Bus's

### 1. Data Buses :

Data bus is the most common type of bus. It is used to transfer data between different components of computer. The number of lines in data bus affects the speed of data transfer between different components. The data bus consists of 8, 16, 32, or 64 lines. A 64-line data bus can transfer 64 bits of data at one time.

The data bus lines are bi-directional. It means that:

1. CPU can read data from memory using these lines
2. CPU can write data to memory locations using these lines

### 2. Address Bus:

Many components are connected to one another through buses. Each component is assigned a unique ID. This ID is called the address of that component. If a component wants to communicate with another component, it uses address bus to specify the address of that component. The address bus is a unidirectional bus. It can carry information only in one direction. It carries address of memory location from microprocessor to the main memory.

### **3. Control Bus:**

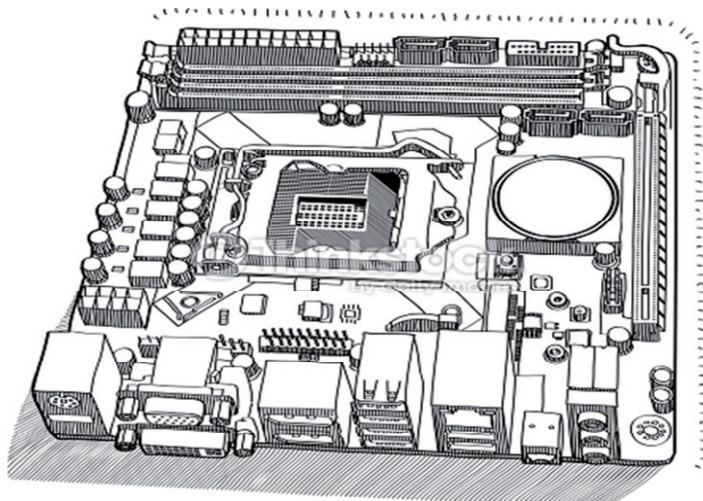
Control bus is used to transmit different commands or control signals from one component to another component. Suppose CPU wants to read data from main memory. It will use control bus to transmit control signals like ASKS (Acknowledgement signals). A control signal contains the following:

1. Timing information: It specifies the time for which a device can use data and address bus.
2. Command Signal: It specifies the type of operation to be performed.

Suppose that CPU gives a command to the main memory to write data. The memory sends acknowledgement signal to CPU after writing the data successfully. CPU receives the signal and then moves to perform some other action.

### **Mother Board:**

The motherboard is a sheet of plastic that holds all the circuitry to connect the various components of a computer system. Learn how the motherboard functions to make all the other components work together. A motherboard is one of the most essential parts of a computer system. It holds together many of the crucial components of a computer, including the central processing unit (CPU), memory and connectors for input and output devices. The base of a motherboard consists of a very firm sheet of non-conductive material, typically some sort of rigid plastic. Thin layers of copper or aluminum foil, referred to as traces, are printed onto this sheet.

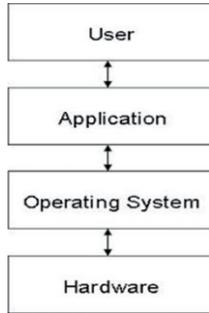


### **Chip sets:**

A chipset is a group of integrated circuits (microchips) that can be used together to serve a single function and are therefore manufactured and sold as a unit. For example, one chipset might combine all the microchips needed to serve as the communications controller between a processor and memory and other devices in a computer. The chips may be controllers for memory, cache, hard drive, key board and peripherals.

### **Operating System:**

It is software that manages the computer hardware, and provides common services for execution of various application software. For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between application programs and the computer hardware.



## Objectives of Operating System:

- **Convenience:** makes computer user friendly.
- **Efficiency:** allows computer to use resources efficiently.
- **Ability to evolve:** constructed in a way to permit effective development, testing and introduction of new functions without interfering with service.

## Functions of Operating System:

- **Resource Management:** The resource management function of an OS allocates computer resources such as CPU time, main memory, secondary storage, and input and output devices for use.
- **Process Management:** The operating system is responsible for the following activities in connection with process management:
  - i. Creating and deleting both user and system processes.
  - ii. Suspending and resuming processes.
  - iii. Providing mechanisms for process synchronization.
  - iv. Providing mechanisms for process communication.
  - v. Providing mechanisms for deadlock handling.
- **Memory Management:** The operating system is responsible for the following activities in connection with memory management:
  - i. Keeping track of which parts of memory are currently being used and by whom.
  - ii. Deciding which processes and data to move into and out of memory.
  - iii. Allocating and deallocating memory space as needed.

- **Storage Management:**
  - i. **File – System Management:** The operating system is responsible for the following activities in connection with the file management:
    - a. Creating and deleting files
    - b. Creating and deleting directories to organize files.
    - c. Supporting primitives for manipulating files and directories.
    - d. Mapping files onto secondary storage.
    - e. Backing up files on stable (non volatile) storage media.
  - ii. **Mass – Storage Management:** The operating system is responsible for the following activities in connection with disk management:
    - a. Free-space Management
    - b. Storage Allocation
    - c. Disk Scheduling.
- **Device Management:** One of the purposes of operating system is to hide the peculiarities of specific hardware devices from the user.
- **Data Management:** The data management functions of an OS govern the input and output of the data and their location, storage, and retrieval.
- **Job Management:** The job management function of an OS prepares, schedules, controls, and monitors jobs submitted for execution to ensure the most efficient processing. A job is a collection of one or more related programs and their data.
- **Standard means of communication between user and computer:** The OS establishes a standard means of communication between users and their computer systems. It does this by providing a user interface and a standard set of commands that control the hardware.

In a multitasking operating system where multiple programs can be running at the same time, the operating system determines which applications should run in what order and how much time should be allowed for each application before giving another application a turn. It manages the sharing of internal memory among multiple applications.

## Types of Operating System:

- **Batch Operating System:** Batch operating system is the operating system which analyses your input and groups them into batches i.e. data in each batch is of similar characteristics. And then it performs operation on each individual batch.
- **Real-time:** A real-time operating system is a multitasking operating system that aims at executing real-time applications. Real-time operating systems often use specialized scheduling algorithms so that they can achieve a deterministic nature of behaviour. The main object of real-time operating systems is their quick and predictable response to events. They either have an event-driven or a time-sharing design. An event-driven system switches between tasks based on their priorities while time-sharing operating systems switch tasks based on clock interrupts.
- **Hard real-time system:** It has the most stringent requirements, guaranteeing that real-time tasks be completed within their deadlines. Safety-critical systems are typically hard real-time systems.
- **Soft real-time system:** It is less restrictive, simply providing that a critical real-time task will receive priority over other tasks and that it will retain that priority until it completes. Many commercial operating systems – as well as Linux – provide soft real-time support.
- **Multi-user vs. Single-user:** A **multi-user operating system** allows multiple users to access a computer system concurrently. Time-sharing system can be classified as multi-user systems as they enable a multiple user access to a computer through the sharing of time. **Single-user operating systems**, as opposed to a multi-user operating system, are usable by a single user at a time. Being able to have multiple accounts on a Windows operating system does not make it a multi-user system. Rather, only the network administrator is the real user. But for a Unix-like operating system, it is possible for two users to login at a time and this capability of the OS makes it a multi-user operating system.
- **Multi-tasking vs. Single-tasking:** When a single program is allowed to run at a time, the system is grouped under a single-tasking system, while in case the operating system allows the execution of multiple

tasks at one time, it is classified as a multi-tasking operating system. Multi-tasking can be of two types namely, pre-emptive or co-operative. In **pre-emptive multitasking**, the operating system slices the CPU time and dedicates one slot to each of the programs. Unix-like operating systems such as Solaris and Linux support pre-emptive multitasking. **Cooperative multitasking** is achieved by relying on each process to give time to the other processes in a defined manner. MS Windows prior to Windows 95 used to support cooperative multitasking.

- **Single-processor Systems:** On a single-processor system, there is one main CPU capable of executing a general-purpose instruction set, including instructions from user processes.
- **Multi-processor Systems:** A multiprocessing operating system allows a program to run on more than one central processing unit (CPU) at a time. This can come in very handy in some work environments, at schools, and even for some home-computing situations.
  - a. **Asymmetric multiprocessing:** In this each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the slave processors.
  - b. **Symmetric multiprocessing (SMP):** In this each processor performs all tasks within the operating system. SMP means that all processors are peers; no master-slave relationship exists between processors.
- **Distributed:** A distributed operating system manages a group of independent computers and makes them appear to be a single computer. The development of networked computers that could be linked and communicate with each other, gave rise to distributed computing. Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they make a distributed system.
- **Embedded:** Embedded operating systems are designed to be used in embedded computer systems. They are designed to operate on small machines like PDAs with less autonomy. They are able to operate with a limited number of resources. They are very compact and extremely

efficient by design. Windows CE and Minix 3 are some examples of embedded operating systems.

## Basics of Networking & Topology and Network Interface Card(NIC):

A network consists of two or more computers that are linked in order to share resources (such as printers and CDs), exchange files, or allow electronic communications. The computers on a network may be linked through cables, telephone lines, radio waves, satellites, or infrared light beams.



Computer networks share common devices, functions, and what features including servers, clients, transmission media, shared data, shared printers and other hardware and software resources, network interface card(NIC), local operating system(LOS), and the network operating system (NOS).

**Servers** - Servers are computers that hold shared files, programs, and the network operating system. Servers provide access to network resources to all the users of the network. There are many different kinds of servers, and one server can provide several functions. For example, there are file servers, print servers, mail servers, communication servers, database servers, fax servers and web servers, to name a few.

**Clients** - Clients are computers that access and use the network and shared network resources. Client computers are basically the customers(users) of the network, as they request and receive services from the servers.

**Transmission Media** - Transmission media are the facilities used to

interconnect computers in a network, such as twisted-pair wire, coaxial cable, and optical fiber cable. Transmission media are sometimes called channels, links or lines.

**Shared data** - Shared data are data that file servers provide to clients such as data files, printer access programs and e-mail.

**Shared printers and other peripherals** - Shared printers and peripherals are hardware resources provided to the users of the network by servers. Resources provided include data files, printers, software, or any other items used by clients on the network.

### **Network Interface Card**

Each computer in a network has a special expansion card called a network interface card (NIC). The NIC prepares(formats) and sends data, receives data, and controls data flow between the computer and the network. On the transmit side, the NIC passes frames of data on to the physical layer, which transmits the data to the physical link. On the receiver's side, the NIC processes bits received from the physical layer and processes the message based on its contents.

**Local Operating System** - A local operating system allows personal computers to access files, print to a local printer, and have and use one or more disk and CD drives that are located on the computer. Examples are MS-DOS, Unix, Linux, Windows 2000, Windows 98, Windows XP etc.

**Network Operating System** - The network operating system is a program that runs on computers and servers, and allows the computers to communicate over the network.

**Hub** - Hub is a device that splits a network connection into multiple computers. It is like a distribution center. When a computer requests information from a network or a specific computer, it sends the request to the hub through a cable. The hub will receive the request and transmit it to the entire network. Each computer in the network should then figure out whether the broadcast data is for them or not.

**Switch** - Switch is a telecommunication device grouped as one of computer network components. Switch is like a Hub but built in with advanced features. It uses physical device addresses in each incoming messages so that it can deliver the message to the right destination or port.

## Basic components of a computer network:

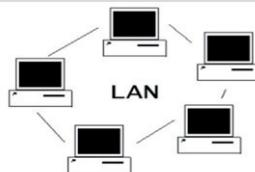
The basic components of a network are as follows.

1. **Protocol:** Set of rules used during the data transmission.
2. **Transmission Medium:** The media used to connect computer to each other like telephone lines, twisted pair wire, co-axial cable, fiber optics, satellite signals and radio signals, etc.
3. **Processors:** Modem, Multiplexers, bridges, routers, gateways, hub etc. are the processors used in the network for the flow of data.
4. **Channels:** Analog/Digital, Synchronous/Asynchronous, Switched/Non switched, Simplex / duplex, etc.
5. **Topology:** Physical network layout used for networking. For example, bus topology, star topology, ring topology, and mesh topology
6. **Software:** User interface software like Internet Explorer, Netscape Navigator, FTP (File Transfer Protocol), Telnet (Telecommunication Network), PPP (Point to Point Protocol), and SMTP (Simple Mail Transfer Protocol) etc.

## Types of Networks:

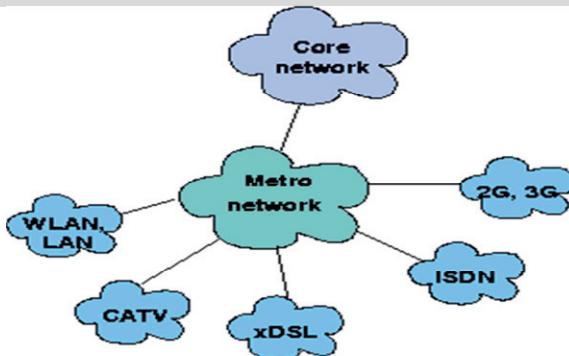
The computer networks are mainly classified into 3 types.

### LAN



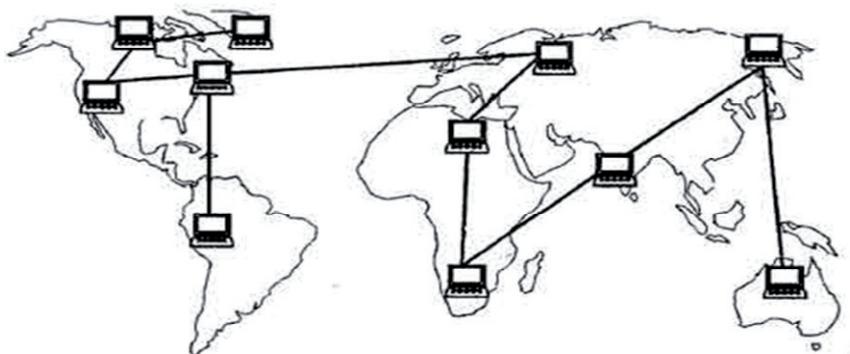
1. LAN (Local Area Network) is a group of computers and other network devices which are connected together.
2. All the devices that are part of LAN are within a building or multiple building.
3. LAN network has very high speed mainly due to proximity of computer and network devices.
4. LAN connection speeds can be 10Mbps or 100Mbps or 1000Mbps also.
5. LAN uses Guided Media.

## MAN



1. MAN ((Metropolitan Area Network) is a larger network of computers and other network devices which are connected together usually spans several buildings or large geographical area.
2. All the devices that are part of MAN are span across buildings or small town.
3. MAN network has lower speed compared to LAN.
4. MAN connection speeds can be 10Mbps or 100Mbps.
5. MAN uses Guided Media or Unguided media.

## WAN



### Wide area network

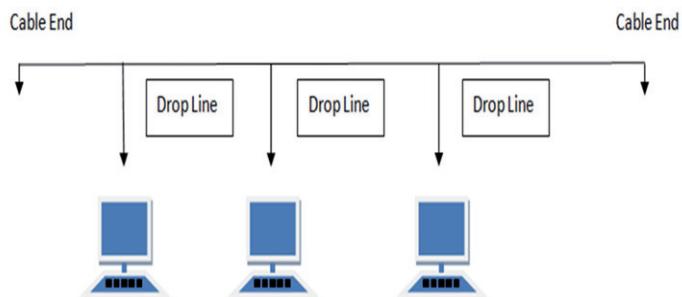
1. WAN (Wide Area Network) is a group of computers and other network devices which are connected together which is not restricted to a geographical location. Internet is WAN
2. All the devices that are part of WAN have no geographical boundaries.

3. WAN speed varies based on geographical location of the servers.  
WAN connects several LANs
4. WAN connection speeds can be 10Mbps or 100Mbps.
5. WAN mainly uses Guided Media or Unguided media. Its long distance communications, which may or may not be provided by public packet network.

## **Network Topologies:**

It is the arrangement of the various elements (links, nodes, etc.) of a computer **network**. Essentially, it is the topological structure of a **network** and may be depicted physically or logically.

**BUS Topology:** It is a network type in where every computer and network device is connected to single cable.



### **Features of Bus topology**

- It transmits data only in one direction.
- Every device is connected to a single cable

### **Advantages of Bus Topology**

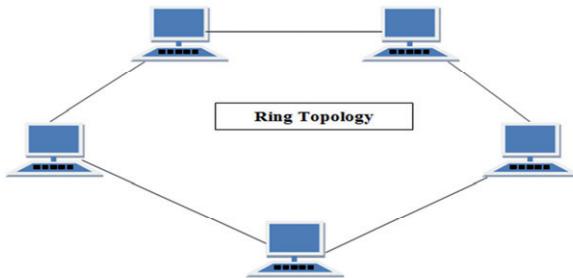
- It is cost effective.
- Cable required is least compared to other network topology.
- Used in small networks.
- It is easy to understand.

- Easy to expand joining two cables together.

### **Disadvantages of Bus topology**

- Cables fails then whole network fails.
- If network traffic is heavy or nodes are more the performance of the network decreases.
- Cable has a limited length.
- It is slower than the ring topology.

**RING Topology:** It is called ring topology because it forms a ring as each computer is connected to another computer, with the last one connected to the first. Exactly two neighbours for each device.



### **Features of Ring topology**

- A number of repeaters are used and the transmission is unidirectional.
- Date is transferred in a sequential manner that is bit by bit.

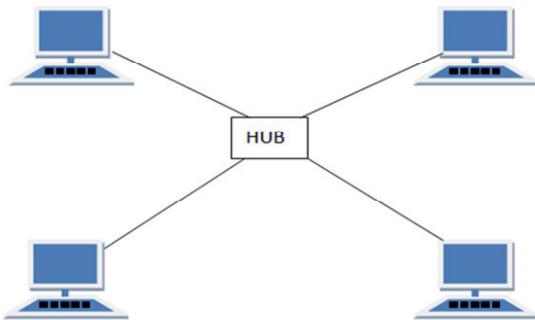
### **Advantages of Ring topology**

- Transmitting network is not affected by high traffic or by adding more nodes, as only the nodes having tokens can transmit data.
- Cheap to install and expand

### **Disadvantages of Ring topology**

- Troubleshooting is difficult in ring topology.
- Adding or deleting the computers disturbs the network activity.
- Failure of one computer disturbs the whole network.

**STAR Topology:** In this type of topology all the computers are connected to a single hub through a cable. This hub is the central node and all others nodes are connected to the central node.



### Features of Star Topology

- Every node has its own dedicated connection to the hub.
- Acts as a repeater for data flow.
- Can be used with twisted pair, Optical Fiber or coaxial cable.

### Advantages of Star Topology

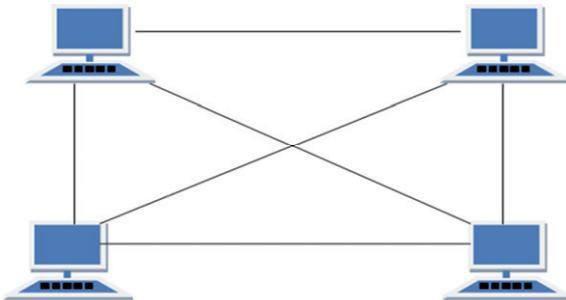
- Fast performance with few nodes and low network traffic.
- Hub can be upgraded easily.
- Easy to troubleshoot.
- Easy to setup and modify.
- Only that node is affected which has failed rest of the nodes can work smoothly.

### Disadvantages of Star Topology

- Cost of installation is high.

- Expensive to use.
- If the hub is affected then the whole network is stopped because all the nodes depend on the hub.
- Performance is based on the hub that is it depends on its capacity

**MESH Topology:** It is a point-to-point connection to other nodes or devices. Traffic is carried only between two devices or nodes to which it is connected. Mesh has  $n(n-2)/2$  physical channels to link  $h_n$  devices.



#### Features of Mesh Topology

- Fully connected.
- Robust.
- Not flexible.

#### Advantages of Mesh Topology:

- Each connection can carry its own data load.
- It is robust.
- Fault is diagnosed easily.
- Provides security and privacy.

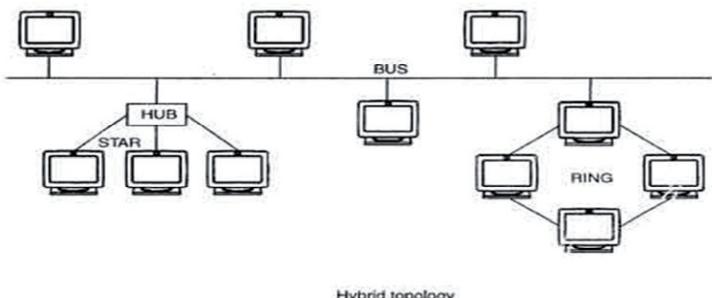
#### Disadvantages of Mesh Topology:

- Installation and configuration is difficult.
- Cabling cost is more.
- Bulk wiring is required.

**Hybrid Topology:** It is two different types of topologies which is a mixture of two or more topologies. For example if in an office in one department ring topology is used and in another star topology is used, connecting these topologies will result in Hybrid Topology (ring topology and star topology).

### Features of Hybrid Topology

1. It is a combination of two or more topologies
2. Inherits the advantages and disadvantages of the topologies included



Hybrid topology

### **Network Interface Cards (NICs):**

This is the important hardware component, which connects the machine to the computer network. This will be fixed into one of the free slot on the mother board. It has one port for the connection of a network cable.

These cards typically use an Ethernet connection and are available in 10, 100, and 1000 Base-T configurations. A 100 Base-T card can transfer data at 100 Mbps. The cards come in ISA and PCI versions and are made by companies like 3Com and Linksys.



## **Laboratory Session-2: Write-up on RAM, SDRAM, FLASH memory, Hard disks, Optical media, CD-ROM/R/RW, DVDs, Flash drives, Keyboard, Mouse, Printers and Plotters. Introduction to flowchart, algorithm and pseudo code.**

### **Random Access Memory (RAM):**

Alternatively referred to as **main memory**, **primary memory**, or **system memory**, **Random Access Memory (RAM)** is a hardware device that allows information to be stored and retrieved on a computer. RAM is usually associated with DRAM, which is a type of memory module. Because information is accessed randomly instead of sequentially like it is on a CD or hard drive, the computer can access the data much faster. However, unlike ROM or the hard drive, RAM is a volatile memory and requires power to keep the data accessible; if power is lost all data contained in memory is lost.



### **Synchronous Dynamic Random Access Memory (SDRAM):**

Synchronous dynamic random access memory (SDRAM) is dynamic random access memory (DRAM) with an interface synchronous with the system bus carrying data between the CPU and the memory controller hub. SDRAM has a rapidly responding synchronous interface, which is in sync with the system bus. SDRAM waits for the clock signal before it responds to control inputs.

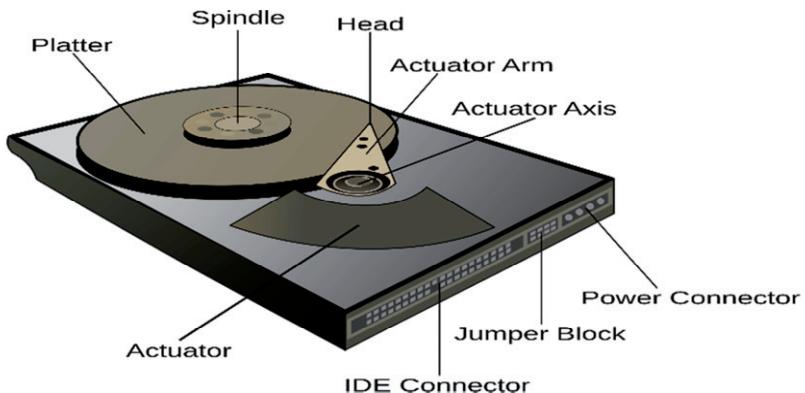
### **Flash Memory:**

Flash memory is a non-volatile memory chip used for storage and for

transferring data between a personal computer (PC) and digital devices. It has the ability to be electronically reprogrammed and erased. It is often found in USB flash drives, MP3 players, digital cameras and solid-state drives.

### **Hard Disks:**

The hard disk drive is the main, and usually largest, data storage hardware device in a computer. The operating system, software titles and most other files are stored in the hard disk drive. The Hard Disk Drive is Also Known as HDD (abbreviation), hard drive, hard disk, fixed drive, fixed disk, fixed disk drive.



### **Optical media:**

An optical storage media is kind of storage, which is coated with thin metal on which bits are stored. The data can be stored in to optical storage media or read form the optical storage media.

- The devices which perform read or write operation on optical storage media are called optical storage media.
- The laser technology is used to read the data or write the data on optical storage devices.

**Examples:** CD-ROM, DVD etc.

### **Compact Disc Read-Only-Memory (CD-ROM):**

CD-ROM (Compact Disc, read-only-memory) is an adaptation of the CD that is designed to store computer data in the form of text and graphics, as well as hi-fi stereo sound. The original data format standard was defined by Philips and Sony in the 1983 Yellow Book.

It is an optical disc which contains audio or software data whose memory is read only. A CD-ROM Driver optical drive is the device used to read them. CD-ROM drives have speeds ranging from 1x all the way up to 72x, meaning it reads the CD roughly 72 times faster than the 1x version. As you would imagine, these drives are capable playing audio CDs and reading data CDs. Below is a picture of the front and back of a standard CD-ROM drive. CD-Drive will be with motor to rotate the disks to perform read and write operations. A CD-drive will consists of the components like Disc drive, disk drive motor, laser pick up assembly tracking drive and tracking motor and so on.

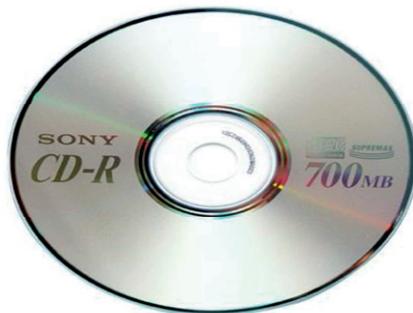


### **Compact Disk Recordable (CD-R):**

Stands for "Compact Disc Recordable." CD-R discs are blank CDs that can record data written by a CD burner. The word "recordable" is used because CD-Rs are often used to record audio, which can be played back by most CD players. However, many other kinds of data can also be

written to a CD-R, so the discs are also referred to as "writable CDs."

The data burned onto a CD-R disc is permanent, meaning it cannot be altered or erased like the data on a hard drive. Typically, once a CD has been burned, it will not be able to record any more data. Some CD burning programs can record data as "sessions," allowing a disc to be written to multiple times until it is full. Each session creates a new partition on the disc, meaning a computer will read a disc with multiple sessions as multiple discs. CD-RWs, on the other hand, can be erased and completely re-recorded. Like CDs, the data on CD-RWs cannot be altered, meaning the disc has to be completely erased each time you want to add new data.



#### **Compact Disc Rewritable (CD-RW):**

Stands for "Compact Disc Re-Writable." A CD-RW is a blank CD that can be written to by a CD burner. Unlike a CD-R (CD-Recordable), a CD-RW can be written to multiple times. The data burned on a CD-RW cannot be changed, but it can be erased. Therefore, you have to completely erase a CD-RW every time you want to change the files or add new data. While it may be somewhat inconvenient, this capability makes CD-RWs a good choice for making frequent backups. However, because CD-RWs can be erased, they don't store data reliably for as long as CD-Rs do. Therefore, you should use regular CD-Rs for long-term backups.



#### **Digital Video Disk or Digital Versatile Disc (DVD-ROM):**

DVD is an optical disc technology with a 4.7 gigabyte storage capacity on a single-sided, one-layered disk, which is enough for a 133-minute movie. DVDs can be single- or double-sided, and can have two layers on each side; a double-sided, two-layered DVD will hold up to 17 gigabytes of video, audio, or other information. This compares to 650 megabytes (.65 gigabyte) of storage for a CD-ROM disk.

DVD uses the MPEG-2 file and compression standard. MPEG-2 images have four times the resolution of MPEG-1 images and can be delivered at 60 interlaced fields per second where two fields constitute one image frame. (MPEG-1 can deliver 30 non-interlaced frames per second.) Audio quality on DVD is comparable to that of current audio compact discs.



**Advantages:** Storage capacity is more compared to CDs

### **Flash Drives (Pen drives):**

USB flash drives are removable, rewritable, and physically much smaller drives weighing even less than 30 g. A small, portable flash memory card that plugs into a computers USB port and functions as a portable hard drive. USB flash drives are touted as being easy-to-use as they are small enough to be carried in a pocket and can plug into any computer with a USB drive. USB flash drives have less storage capacity than an external hard drive, but they are smaller and more durable because they do not contain any internal moving parts.

USB flash drives also are called *thumb drives*, *jump drives*, *pen drives*, *key drives*, *tokens*, or simply *USB drives*.



#### **Advantages:**

- Faster read and write compared to traditional hard disk drives.
- Smaller size.
- Less prone to damage.
- Cheaper than traditional drives in small storage capacities.
- Uses less power than traditional hard disk drives.

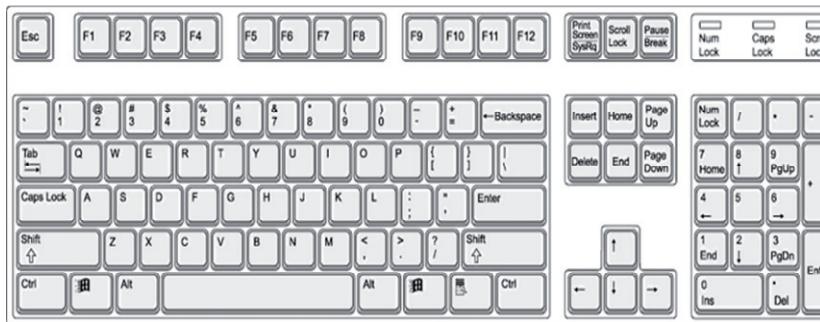
#### **Disadvantages:**

- Flash memory cells have a limited number of write and erase cycles before failing.
- Most flash drives do not have a write-protection mechanism.
- Smaller size devices, such as flash drives make them easier to lose.

- Currently costs a lot more per gigabyte than traditional hard drives for large storage capacities.
- May require a special version of a program to run on a flash-based drive to protect from prematurely wearing out the drive.

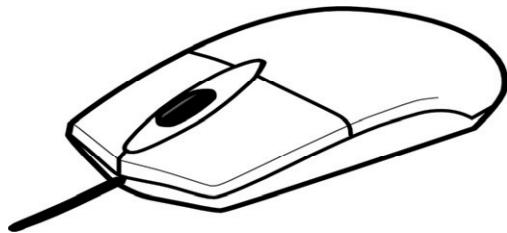
## Keyboard:

A keyboard is the primary input device used in all computers. Keyboard has a group of switches resembling the keys on an ordinary typewriter machine. Normally keyboard has around 101 keys. The keyboard includes key that allows us to type letters, numbers and various special symbols such as \*, /, [ , % etc.



## Mouse:

The mouse is the key input device to be used in a Graphical User Interface (GUI). The users can use mouse to handle the cursor pointer easily on the screen to perform various functions like opening a program or file. With mouse, the users no longer need to memorize commands, which was earlier a necessity when working with text-based command line environment such as MS-DOS.



### Printer:

A **printer** is an external hardware device responsible for taking computer data and generating a hard copy of that data. Printers are one of the most used peripherals on computers and are commonly used to print text, images, and photos. The picture is the Lexmark Z605 Inkjet, an example of a computer printer.



The printer is an output device, which is used to get hard copy of the text displayed on the screen. The printer is an external optional device that is connected to the computer system using cables. The printer driver software is required to make the printer working.

### Types of Printers:

**I) Impact Printers:** Impact printer refers to a class of printers that work by

banging a head or needle against an ink ribbon to make a mark on the paper. This includes dot matrix printer, daisy-wheel printers, and Line printers. In contrast, laser and ink-jet printers are *nonimpact printers*. The distinction is important because impact printers tend to be considerably noisier than nonimpact printers but are useful for multipart forms such as invoices.

**2) Non-Impact Printers:** a printer that creates images without mechanically impacting the page, as an ink-jet or laser printer.**E.g.** Inkjet Printers and Laser Printer.

### Dot matrix, Inkjet and Laser printers

| Sl. No. | Dot Matrix Printer                                                                      | Inkjet Printer                                                                                                                                | Laser Printer                                                                       |
|---------|-----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| 1       | <b>Impact Printer</b>                                                                   | <b>Non-impact Printer</b>                                                                                                                     | <b>Non-impact printer</b>                                                           |
| 2       | It uses metal pins in its head to create text and graphics in the form of dots.         | Its print head does not have metal pins; instead it has several tiny nozzles that spray ink onto the paper. Each nozzle is thinner than hair. | The laser printer uses a beam of laser for printing.                                |
| 3       | The process of printing involves striking a pin against a ribbon to produce its output. | The ink cartridges are attached to the printer head that moves horizontally from left to right.                                               | The printer uses a cylindrical drum, a toner and the laser beam.                    |
| 4       | Printing speed is slower than laser printer,                                            | Printing speed is slower than laser dot matrix.                                                                                               | Printing speed is higher than both.                                                 |
| 5       | Character by character printing                                                         | Line by line printing                                                                                                                         | It is a page printer                                                                |
| 6       | Low quality printing                                                                    | High quality printing                                                                                                                         | High quality printing                                                               |
| 7       | Less expensive                                                                          | High expensive                                                                                                                                | High expensive                                                                      |
| 8       | Generates much noise while printing                                                     | Generates less noise while printing                                                                                                           | No noise                                                                            |
| 9       | Speed is measured in DPI (Dots Per Inch)                                                | Speed is measured in CPI (Characters Per Inch)                                                                                                | Speed is measured in PPM (Pages Per Minute)                                         |
| 10      | Monochrome (Black & White) Printers                                                     | Colour printer                                                                                                                                | Monochrome and colour printer                                                       |
|         |      |                                                            |  |

## **Plotters:**

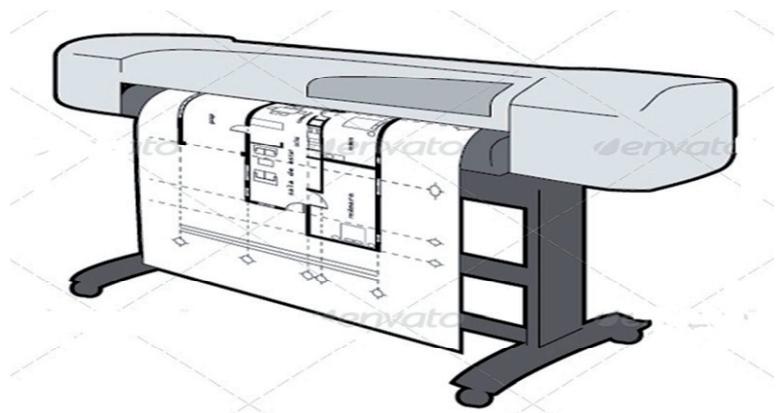
A **plotter** is a computer hardware device much like a printer that is used for printing vector graphics. Instead of toner, plotters use a pen, pencil, marker, or another writing tool to draw multiple, continuous lines onto paper rather than a series of dots like a traditional printer. Though once widely used for computer system design, these devices have more or less been phased out by wide-format printers. Plotters are used to produce a hard copy of schematics and other similar applications.

### **Advantages of plotters**

- Plotters can work on very large sheets of paper while maintaining high resolution.
- They can print on a wide variety of flat materials including plywood, aluminum, sheet steel, cardboard, and plastic.
- Plotters allow the same pattern to be drawn thousands of time without any image degradation.

### **Disadvantages**

- Plotters are quite large when compared to a traditional printer.
- Plotters are also much more expensive than a traditional printer.



## PART B

### C PROGRAMMING LABORATORY EXPERIMENTS

**Note:** *Implement the following programs with WINDOWS / LINUX platform using appropriate C compiler.*

1. Design and develop a flowchart or an algorithm that takes three coefficients ( $a$ ,  $b$ , and  $c$ ) of a Quadratic equation ( $ax^2+bx+c=0$ ) as input and compute all possible roots. Implement a C program for the developed flowchart/algorithm and execute the same to output the possible roots for a given set of coefficients with appropriate messages.
  
2. Design and develop an algorithm to find the *reverse* of an integer number **NUM** and check whether it is PALINDROME or NOT. Implement a C program for the developed algorithm that takes an integer number as input and output the reverse of the same with suitable messages. Ex: Num: **2014**, Reverse: **4102**, Not a Palindrome.
  
- 3a. Design and develop a flowchart to find the square root of a given number  $N$ . Implement a C program for the same and execute for all possible inputs with appropriate messages. Note: **Don't use library function *sqrt(n)*.**
  
- 3b. Design and develop a C program to read a *year* as an input and find whether it is *leap year* or not. Also consider end of the centuries.
  
4. Design and develop an algorithm to evaluate polynomial  $f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ , for a given value of  $x$  and its coefficients using Horner's method. Implement a C program for the same and execute the program with different set of values of coefficients and  $x$

**5.** Draw the flowchart and Write a C Program to compute **Sin(x)** using Taylor series approximation given by  $\text{Sin}(x) = x - (x^3/3!) + (x^5/5!) - (x^7/7!) + \dots$ . Compare your result with the built-in Library function.

Print both the results with appropriate messages.

**6.** Develop an algorithm, implement and execute a C program that reads  $N$  integer numbers and arrange them in ascending order using **Bubble Sort**.

**7.** Develop, implement and execute a C program that reads two matrices  $A$  ( $m \times n$ ) and  $B$  ( $p \times q$ ) and Compute product of matrices  $A$  and  $B$ . Read matrix  $A$  and matrix  $B$  in row major order and in column major order respectively. Print both the input matrices and resultant matrix with suitable headings and output should be in matrix format only. Program must check the compatibility of orders of the matrices for multiplication. Report appropriate message in case of incompatibility.

**8.** Develop, implement and execute a C program to search a Name in a list of names using **Binary searching** Technique.

**9.** Write and execute a C program that

- i)** Implements string copy operation **STRCOPY** (str1,str2) that copies a string  $str1$  to another string  $str2$  without using library function.
- ii)** Read a *sentence* and print frequency of vowels and total count of consonants.

**10.a.** Design and develop a C function **RightShift(x ,n)** that takes two integers  $x$  and  $n$  as input and returns value of the integer  $x$  rotated to the right by  $n$  positions. Assume the integers are unsigned. Write a C program that invokes this function with different values for  $x$  and  $n$  and tabulate the results with suitable headings.

**b.** Design and develop a C function *isprime*(num) that accepts an integer argument and returns 1 if the argument is prime, a 0 otherwise. Write a C program that invokes this function to generate prime numbers between the given range.

**11.** Draw the flowchart and write a **recursive C** function to find the factorial of a number,  $n!$ , defined by  $fact(n)=1$ , if  $n=0$ . Otherwise  $fact(n)=n*fact(n-1)$ . Using this function, write a C program to compute the binomial coefficient  ${}^nC_r$ . Tabulate the results for different values of  $n$  and  $r$  with suitable messages.

**12.** Given two university information files “**studentname.txt**” and “**usn.txt**” that contains students Name and USN respectively. Write a C program to create a new file called“**output.txt**” and copy the content of files “studentname.txt” and “usn.txt” into output file in the sequence shown below. Display the contents of output file “output.txt” on to the screen.

| Student Name | USN   | Heading |
|--------------|-------|---------|
| Name1        | USN1  |         |
| Name2        | USN2  |         |
| Name3        | USN3  |         |
| -----        | ----- |         |
| -----        | ----- |         |

**13.** Write a C program to maintain a record of  $n$  student details using an array of structures with four fields (Roll number, Name, Marks, and Grade). Assume appropriate data type for each field. Print the marks of the student, given the student name as input.

**14.** Write a C program using pointers to compute the sum, mean and standard deviation of all elements stored in an array of **n** real numbers.

### **Course Outcomes:**

- Gaining Knowledge on various parts of a computer
- Able to draw flow charts and write algorithms
- Able to design and development of C problem solving skills
- Able to design and develop modular programming skills
- Able to trace and debug a program

### **Conduction of Practical Examinations:**

1. All laboratory experiments (14 in number ) are to be included for practical examination .
2. Students are allowed to pick one experiment from the lot.
3. Strictly follow the instructions as printed on the cover page of answer script for breakup of marks ( Generally : 15% (12 Marks ) for Write up, 15% (12 Marks ) for Viva and 70% (56 Marks ) for Execution) . However the break up marks will be as per the VTU Rules and Regulations exists at that time.
4. Change of experiment is allowed only once and 15% marks allotted to the procedure part to be made zero.

### **Note:**

*The C programs given in this lab manual are tested in Ubuntu Linux. Inorder to execute the programs in Turbo C, one should add the header files like #include<conio.h> and functions like clrscr() and getch().*

- 1. Design and develop a flowchart or an algorithm that takes three coefficients (a, b and c) of a quadratic equation ( $ax^2+bx+c=0$ ) as input and compute all possible roots. Implement a C program for the developed flowchart/algorithm and execute the same to output the possible roots for a given set of coefficients with appropriate message.**

### **Problem Description:**

The given problem expects to find the roots of given quadratic equation for given three coefficients. A QUADRATIC equation is a polynomial whose highest exponent is 2.  $ax^2 + bx + c=0$ . The coefficient  $a$ , of  $x^2$  is called the leading coefficient. Where  $b$  and  $c$  are the remaining two coefficients. In addition the value of  $a$  must not be equal to 0. If  $a$  is equal to zero the degree of equation becomes 1 and violates the definition of quadratic equation (whose degree must be 2). Each quadratic equation has two roots. The quadratic formula for solving the equation is given by

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

**Discriminant:** The radical portion of this formula  $\sqrt{b^2 - 4ac}$  determines the nature of the root. This quantity under radical sign  $d = b^2 - 4ac$  is called discriminant (d). In order to determine the roots of quadratic equation one has to find the value of discriminator of the given equation.

The nature of the roots will be determined using the value of d. Three things may occur regarding the discriminant:

- i. If  $d = 0$ , the roots are real and equal

**Root 1 = Root 2 =  $-b/2*a$**

Let us consider  $r = \text{sqrt}(\text{abs}(d))$ .

- ii. If  $d > 0$ , the roots are real and distinct

$$\text{Root 1} = (-b + r) / 2*a;$$

$$\text{Root 2} = (-b - r) / 2*a;$$

iii. If  $d < 0$ , the roots are imaginary

$$r_p = -b/2*a;$$

$$i_p = (r)/2*a;$$

$$\text{Root 1} = r_p + i*i_p;$$

$$\text{Root 2} = r_p - i*i_p;$$

Some of the examples of the quadratic equations and their roots are as follows:

1.  $x^2 + 2x - 8 = 0 \Rightarrow (x + 4)(x - 2) \Rightarrow$  (Here  $d > 0$ ) The roots of quadratic equations are -4 and 2

2.  $x^2 - 16 = 0 \Rightarrow (x + 4)(x - 4) = 0 \Rightarrow$  (Here  $d > 0$ ) The roots are real and distinct and are equal to  $\pm 4$

3.  $x^2 - 3x + 9 = 0 \Rightarrow x = 6 \pm \sqrt{0}$  (Here  $d = 0$ ) The roots are real and equal  $= 3$

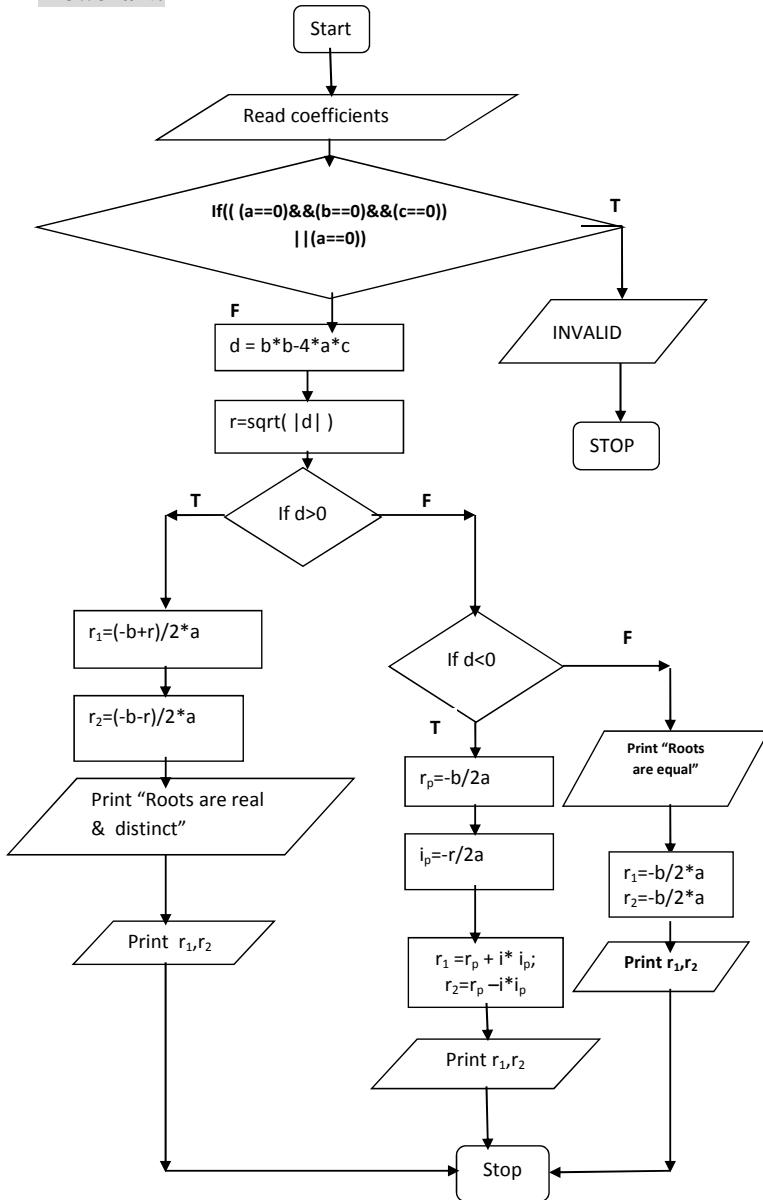
4.  $x^2 + x + 3 = 0 \Rightarrow x = \frac{-1 \pm \sqrt{-11}}{2}$  (Here  $d < 0$ ) The roots are imaginary and are equal to

$$\text{Root 1} = (-1+i\sqrt{11})/2 \text{ and Root 2} = (-1-i\sqrt{11})/2$$

## **Algorithm:**

- 1.Start.
- 2.Input non zero co-efficient of equation a, b, c .
3. **If** any or all the coefficients are zero  
    Go to Step 2.  
**End If**
- Else**  
    Compute  $d = b^2 - 4ac$  and  $r = \sqrt{|d|}$   
**If** ( $d > 0$ )  
        Root 1 =  $(-b + r) / (2a)$   
        Root 2 =  $(-b - r) / (2a)$   
        Output “Roots are REAL and DISTINCT”  
        Output Root1, Root2  
**End if**  
**Else If** ( $d < 0$ )  
         $r_p = -b / (2a)$   
         $i_p = r / (2a)$   
        Output “Roots are COMPLEX”  
        Output  $r_p + i_p$  and  $r_p - i_p$   
**End else if**  
**Else**  
    Root 1 = Root 2 =  $-b / (2a)$   
    Output “Roots are EQUAL”  
    Output Root 1, Root 2  
**End Else**  
**End Else**
- 4.Stop

**Flowchart:**



## **Program:**

```
#include <stdio.h>
#include <math.h>
void main()
{
    int    a,b,c;
    float d,x1,x2,r;
    printf("Enter three co-efficients :\n");
    scanf("%d%d%d",&a,&b,&c);

    if ((a==0)&&(b==0)&&(c == 0))
    {
        printf("\n Invalid Input ");
        exit(0);
    }

    if(a==0)
    { printf("\n Linear Equation \n");
        printf("\n The Root = %f\n",-c/b);
        exit(0);
    }
    else
    {
        d = b * b - 4 * a * c;
        r=sqrt(fabs(d));
        if (d > 0)
        {
            x1 = (-b +r) / (2.0*a);
            x2 = (-b -r) / (2.0*a);
            printf("\n The Roots are Real and distinct\n");
            printf("\n The Roots are :\n ");
            printf("\n Root 1=%f\n" ,x1);
            printf("\n Root 2 = %f \n",x2);
        }
        else if (d == 0)
        {
            x1 = x2 = -b/(2.0*a);
        }
    }
}
```

```

printf("\n The roots are real and equal\n");
printf("\n The roots are: \n");
printf ("\n Root 1= Root 2=%f",x1);
}
else
{
    x1 = -b / (2.0 * a);
    x2 = r / (2.0*a);
    printf("\n The roots are real and imaginary\n");
    printf("\n The roots are:\n");
    printf("\n Root 1: %f + i *%f \n",x1,x2);
    printf("\n Root 2 : %f - i *%f \n",x1,x2);
}
}
}
}

```

### **Sample Input and Output:**

#### **Run 1:**

Enter three co-efficients:

1 4 4

The roots are real and equal

The roots are:

Root 1= Root 2 = 2.0000

#### **Run 2:**

Enter three co-efficients:

1 - 5 6

The roots are real and distinct

The roots are:

Root 1 = 3.0000

Root 2 = 2.0000

**Run 3:**

Enter three co-efficients:

2 3 4

The roots are real and imaginary

The roots are:

1) -0.750000 +i 1.198958

2) -0.750000 - i 1.198958

**Run 4:**

Enter three co-efficients :

0 0 0

Invalid Input

**2. Design and develop an algorithm to find the reverse of an integer number NUM and check whether it is PALINDROME or NOT. Implement a C program for the developed algorithm that takes an integer number as input and output the reverse of the same with suitable messages. Ex: 2014 , Reverse :4012 , Not a Palindrome.**

#### **Algorithm:**

Step 1: Start.  
Step 2: Input n.  
Step 3: Initialize m= n and rev = 0.  
Step 4: Until m not equal to zero  
            rem = m%10;  
            m = m/10;  
            rev = rev\*10+ rem;  
Step 5: If n is equal to rev  
            Output “Palindrome”.  
            else  
                Output “Not a Palindrome”.  
Step 6: Stop

#### **C Program :**

```
#include<stdio.h>
main()
{
    int m,n,rev=0,rem;
    printf("\n Enter the number :");
    scanf("%d",&n);
    m = n;
    while(m!=0)
    {
        rem = m%10; //To determine the last digit of number
        m = m/10; // To reduce the number by one digit less
        rev = rev*10+ rem; // To reverse a number by appending
                            // the number by a digit (rem)
    }
    printf("\n The reverse number is :%d\n", rev);

    if(rev == n)
        printf("\n %d is a palindrome",n);
    else
        printf("\n %d is not a palindrome",n);
}
```

**Note:** A palindromic number is a number that is the same when written forwards or backwards, i.e., of the form  $a_1 \ a_2 \ \dots \ a_2 \ a_1$ . The first few palindromic numbers are therefore are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99, 101, 111, 121, -----

**Example:** 1221, 1111, 2222, 1331, 2552, 121, 1001, etc are palindrome numbers.

**Sample Input /Output :**

**Run 1:**

Enter the number : 1234

The reverse number is: 4321

4321 is not a palindrome

**Run 2:**

Enter the number: 1201

The reverse number is 1021

1201 is not a palindrome

**Run 3:**

Enter the number : 1221

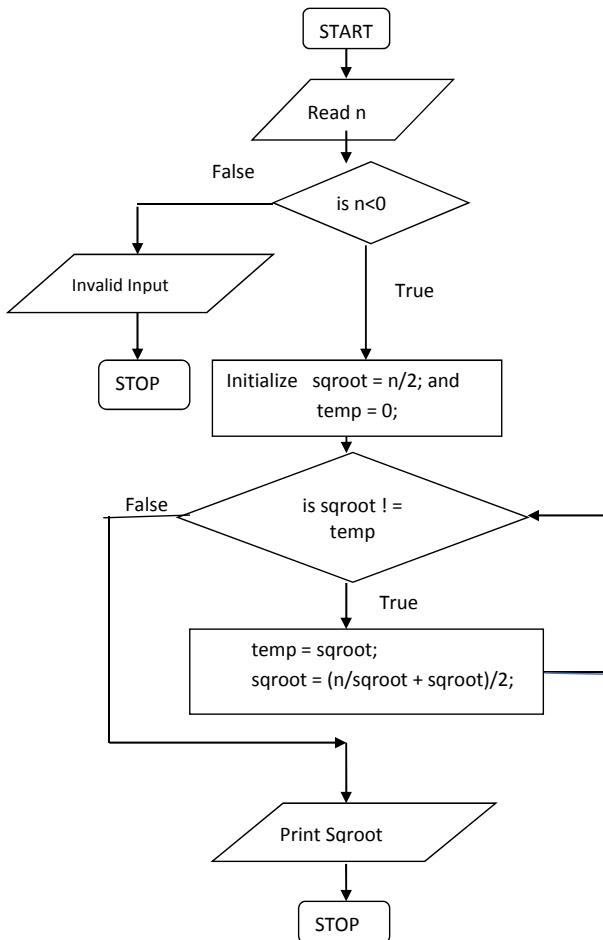
The reverse number is 1221

1221

1221 is a palindrome number

**3a. Design and develop a flowchart to find the square root of a given number N. Implement a C Program for the same and execute for all possible inputs with appropriate messages . Note : Don't use Library function.**

**Flowchart :**



### **Program:**

```
#include<stdio.h>
main()
{
    float n;
    float sqroot,temp;

    printf("\n Enter a number :");
    scanf("%f",&n)

    if(n<0)
    {
        printf("\n The square root of negative number cannot be determined \n");
    }
    else
    {
        sqroot=n/2;
        temp =0;
        while(sqroot!=temp)
        {
            temp = sqroot;
            sqroot = (n/sqroot + sqroot)/2;
        }
        printf("\n Square root of %f is %.5f \n",n,sqroot);
    }
}
```

### **Sample Input/Output :**

#### **Run 1:**

**Enter a number: -4**

**The square root of negative number cannot be determined**

#### **Run 2:**

**Enter a number: 25**

**Square root of 25.00000 is 5.00000**

#### **Run 3:**

**Enter a number: 49**

**Square root of 49.000000 is 7.00000**

**Run 4:**

**Enter a number: 33**

**Square root of 33.000000 is 5.74456**

**Run 5:**

**Enter a number: 72.567893**

**Square root of 72.567893 is 8.51868**

**Note:** The square root of a number, n, written  $\sqrt{n}$  is the number say x that gives n when multiplied by itself (ie.  $n = x * x$ ). For example,  $\sqrt{100} = 10$  because  $10 * 10 = 100$ .

**Examples:** Here are the square roots of all the perfect squares from 1 to 100.

|                   |                       |
|-------------------|-----------------------|
| $\sqrt{1} = 1$    | since $1 = 1 * 1$     |
| $\sqrt{4} = 2$    | since $4 = 2 * 2$     |
| $\sqrt{9} = 3$    | since $9 = 3 * 3$     |
| $\sqrt{16} = 4$   | since $16 = 4 * 4$    |
| $\sqrt{25} = 5$   | since $25 = 5 * 5$    |
| $\sqrt{36} = 6$   | since $36 = 6 * 6$    |
| $\sqrt{49} = 7$   | since $49 = 7 * 7$    |
| $\sqrt{64} = 8$   | since $64 = 8 * 8$    |
| $\sqrt{81} = 9$   | since $81 = 9 * 9$    |
| $\sqrt{100} = 10$ | since $100 = 10 * 10$ |

**3b. Design and develop a C program to read a year as an input and find whether it is leap year or not. Also consider the end of centuries.**

### **Regarding Leap Year:**

**Note1:** A year is a leap year if

3. **year** is divisible by 4 but not by 100. (**year % 4 == 0 && year%100!=0**). Example: 1996, 1992 and 1998.
4. **year** is divisible by 4 and by 100 and also divisible by 400. (**(year%4==0 && year %100==0)&& (year %400==0)**). Example : 400,800,1200,1600 and 2000

**Note2:** Any number which is divisible by 400 is also divisible by 4 and 100.

```
#include<stdio.h>

main()
{
    int year;

    printf("\n Enter a year \n");
    scanf("%d",&year);
    if((year%4==0)&&(year%100!=0)|(year%400==0))
    {
        printf("%d is a leap year \n",year );
    }
    else
    {
        printf("%d is not a leap year \n",year);
    }
}
```

### **Sample Input/ Output :**

#### **Run 1:**

Enter a year  
2016  
2016 is a leap year

#### **Run 2:**

Enter a year  
2017  
2017 is not a leap year

#### **Run 3:**

Enter a year  
200  
200 is not a leap year

#### **Run 4:**

Enter a year  
4  
4 is a leap year

#### **Run 5:**

Enter a year  
400  
400 is a leap year

**4. Design and develop an algorithm for evaluating the polynomial  $f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$  for a given value of  $x$  and its coefficients using Horner's method. Implement a C program for the developed algorithm and execute for different sets of values of coefficients and  $x$ .**

**Horner's Method** : Consider the polynomial  $f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ . The polynomial can be solved using Horner's method as given below:

$$\begin{aligned}
 f(x) &= a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \\
 &= (((a_4x + a_3)x + a_2)x + a_1)x + a_0 \quad // \text{sum} = a_4x \\
 &= (((\text{sum} + a_3)x + a_2)x + a_1)x + a_0 \quad // \text{sum} = (\text{sum} + a_3)x \\
 &= ((\text{sum} + a_2)x + a_1)x + a_0 \quad // \text{sum} = (\text{sum} + a_2)x \\
 &= (\text{sum} + a_1)x + a_0 \quad // \text{sum} = (\text{sum} + a_1)x \\
 &= \text{sum} + a_0
 \end{aligned}$$

**Example: Evaluate the polynomial  $f(x) = x^4 + 3x^3 + 5x^2 + 7x + 9$  at  $x = 2$**

**Here , a0 = 9 , a1 = 7 , a2 = 5 , a3 = 3 , a4 = 1 and x = 2**

$$f(x) = (((a_4x + a_3)x + a_2)x + a_1)x + a_0$$

$$\begin{aligned}
 &= (((x + 3)x + 5)x + 7)x + 9 \\
 &= (((2 + 3)2 + 5)2 + 7)2 + 9 \\
 &= (((5)2 + 5)2 + 7)2 + 9 \\
 &= ((10 + 5)2 + 7)2 + 9 \\
 &= ((15)2 + 7)2 + 9 \\
 &= (30 + 7)2 + 9 \\
 &= (37)2 + 9 \\
 &= 74 + 9 \\
 &= 83
 \end{aligned}$$

### **Algorithm :**

**Step 1:** Read n  
**Step 2:** Read the coefficients  
    for i = 0 to n  
        Read coefficients into a[i]  
    end for  
**Step 3:** Read x  
**Step 4:** Initialize sum = a[n]\*x;  
**Step 5:** for i = n-1 down to 1  
        sum = (sum+a[i])\*x;  
    end for  
**Step 6** : sum = sum + a[0]  
**Step 7** : write sum  
**Step 8** : Stop

### **Program:**

```
#include<stdio.h>
main()
{ int n,i;
float a[10],x,sum;
printf("\n Enter the order of polynomial , n \n");
scanf("%d",&n);
printf("\n Enter %d coefficients \n",n+1);
for(i=0;i<=n;i++)
{ scanf("%f",&a[i]);
}
printf("\n Enter the value of x\n");
scanf("%f",&x);
sum = a[n]*x;
for(i=n-1;i>=1;i--)
{ sum = (sum+a[i])*x;
}
sum = sum +a[0];
printf("\n The value of polynomial f(%f) = %f \n", x,sum);
}
```

### **Sample Input/Output**

#### **Run 1:**

Enter the order of polynomial, n

4

Enter 5 co-efficients 1 2 3 4 5

Enter the value of x

2

The value of polynomial f(2.000000) = 129.00000

#### **Run 2:**

Enter the order of polynomial, n

4

Enter 5 co-efficients 5 4 3 2 1

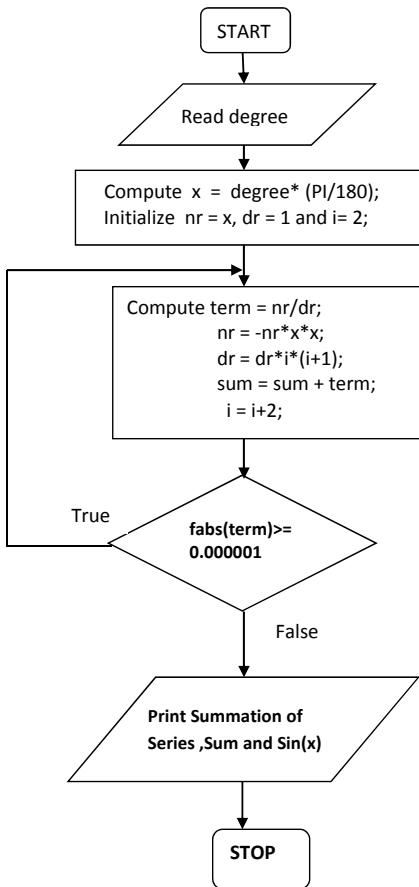
Enter the value of x

5

The value of polynomial f(5.000000) = 975.00000

**5. Draw the flowchart and write a C program to compute  $\sin(x)$  using Taylors series approximation given by  $\sin(x) = x/(1!) - (x^3/3!) + (x^5/5!) - (x^7/7!) + \dots$ . Compare the results with the built in library function and print both results.**

**Flowchart :**



**Note:** To determine the absolute value of integer numbers (int, long int, etc.) **abs()** is used. To determine the absolute value of floating point numbers (float, double, etc.) **fabs()** is used. **abs()** is defined in **stdlib.h** and **fabs()** is defined in **math.h**.

### **Program :**

```
#include<stdio.h>
#include<math.h>
#define PI 3.142

main()
{
    int i,degree;
    float x,sum=0,term,nr,dr;

    printf("\n Enter the value of degree :");
    scanf("%d",&degree);
    x = degree* (PI/180);
    nr = x;
    dr = 1;
    i = 2;
    do
    {
        term = nr/dr;
        nr = -nr*x*x;
        dr = dr*i*(i+1);
        sum = sum + term;
        i = i+2;
    } while(fabs(term)>=0.000001);

    printf("\n The value of summation of sine series of %d
degree is %f\n",degree,sum);

    printf("\n The value of sin(%d) using library function
is %f\n",degree,sin(x));
}
```

### **Sample Input/Output:**

#### **Run 1:**

Enter the value of degree: **60**  
The value of summation of sine series of 60 degree **0.855862**  
The value of sin(60) using library function is **0.866093**

#### **Run 2:**

Enter the value of degree: **0**  
The value of summation of sine series of 0 degree **0.000001**

The value of  $\sin(0)$  using library function is 0.000000

**Run 3:**

Enter the value of degree: 35

The value of summation of sine series of 35 degree 0.57358

The value of  $\sin(35)$  using library function is 0.57358

**Run 4:**

Enter the value of degree: 45

The value of summation of sine series of 45 degree 0.70711

The value of  $\sin(60)$  using library function is 0.70711

**Table of sin (angle)**

| Angle | sin (a) |
|-------|---------|
| 0.0   | 0.0     |
| 1.0   | .0174   |
| 2.0   | .0349   |
| 3.0   | .0523   |
| 4.0   | .0698   |
| 5.0   | .0872   |
| 6.0   | .1045   |
| 7.0   | .1219   |
| 8.0   | .1392   |
| 9.0   | .1564   |
| 10.0  | .1736   |
| 11.0  | .1908   |
| 12.0  | .2079   |
| 13.0  | .2249   |
| 14.0  | .2419   |
| 15.0  | .2588   |
| 16.0  | .2756   |
| 17.0  | .2924   |
| 18.0  | .3090   |
| 19.0  | .3256   |
| 20.0  | .3420   |
| 21.0  | .3584   |
| 22.0  | .3746   |
| 23.0  | .3907   |
| 24.0  | .4067   |

| Angle | sin (a) |
|-------|---------|
| 25.0  | .4226   |
| 26.0  | .4384   |
| 27.0  | .4540   |
| 28.0  | .4695   |
| 29.0  | .4848   |
| 30.0  | .5000   |
| 31.0  | .5150   |
| 32.0  | .5299   |
| 33.0  | .5446   |
| 34.0  | .5592   |
| 35.0  | .5736   |
| 36.0  | .5878   |
| 37.0  | .6018   |
| 38.0  | .6157   |
| 39.0  | .6293   |
| 40.0  | .6428   |
| 41.0  | .6561   |
| 42.0  | .6691   |
| 43.0  | .6820   |
| 44.0  | .6947   |
| 45.0  | .7071   |

| Angle | sin (a) |
|-------|---------|
| 46.0  | .7193   |
| 47.0  | .7314   |
| 48.0  | .7431   |
| 49.0  | .7547   |
| 50.0  | .7660   |
| 51.0  | .7772   |
| 52.0  | .7880   |
| 53.0  | .7986   |
| 54.0  | .8090   |
| 55.0  | .8191   |
| 56.0  | .8290   |
| 57.0  | .8387   |
| 58.0  | .8480   |
| 59.0  | .8571   |
| 60.0  | .8660   |
| 61.0  | .8746   |
| 62.0  | .8829   |
| 63.0  | .8910   |
| 64.0  | .8988   |
| 65.0  | .9063   |
| 66.0  | .9135   |
| 67.0  | .9205   |
| 68.0  | .9272   |
| 69.0  | .9336   |
| 70.0  | .9397   |

| Angle | sin (a) |
|-------|---------|
| 71.0  | .9455   |
| 72.0  | .9511   |
| 73.0  | .9563   |
| 74.0  | .9613   |
| 75.0  | .9659   |
| 76.0  | .9703   |
| 77.0  | .9744   |
| 78.0  | .9781   |
| 79.0  | .9816   |
| 80.0  | .9848   |
| 81.0  | .9877   |
| 82.0  | .9903   |
| 83.0  | .9926   |
| 84.0  | .9945   |
| 85.0  | .9962   |
| 86.0  | .9976   |
| 87.0  | .9986   |
| 88.0  | .9994   |
| 89.0  | .9998   |
| 90.0  | 1.00    |

**6. Develop an algorithm, implement and execute a C program that reads N integer numbers and arrange them in ascending order using Bubble sort.**

### **Algorithm:**

To sort elements of array **a** of size **n** in ascending order using bubble sort technique

Step 0: Start

Step 1: Read the size of array **a** , say **n**

Step 2 : Read **n** elements into array **a**.

    for (i = 0 to n )

        Read elements into array **a[i]**.

    End for

Step 3: Perform Sorting

    for(i=0 to n)

        for(j=0 to n-i )

            if (**a[j]**>**a[j+1]**)

                {     **temp** = **a[j]** ;

**a[j]** = **a[j+1]** ;

**a[j+1]** = **temp**;

            }

Step 4 : Print the sorted array **a**

Step 5: Stop

**Program :**

```
#include<stdio.h>
main()
{
    int n,i,j,temp,a[20];

    printf("\n Enter the number of items : ");
    scanf("%d",&n);
    printf("\n Enter the %d items to sort \n",n);
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);

    for(j=1;j<n;j++)
    {
        for(i=0;i<n-j;i++)
        {
            if(a[i]>=a[i+1])
            {
                temp = a[i] ;
                a[i] = a[i+1];
                a[i+1]= temp ;
            }
        }
    }

    printf("\n The sorted items are \n");
    for(i=0;i<n;i++)
        printf("%d\t",a[i]);
}
```

**Sample Input/Output :****Run 1:**

Enter the number of items : 5  
Enter the 5 items to sort  
50 10 40 30 20  
The sorted items area  
10 20 30 40 50

**Run 2:**

Enter the number of items : 4  
Enter the 4 items to sort  
77 15 0 33

The sorted items area

0 15 33 77

**Run 3:**

Enter the number of items : 5

Enter the 5 items to sort

-50 -10 25 3 -72

The sorted items area

-72 -50 -10 3 25

7. Write a C program that reads two matrices A(m X n ) and B( (pXq) and compute product of matrices A and B . Read matrix A and matrix B in row major order and in column major order respectively. Print both the input matrices and resultant matrix with suitable headings and output should be in matrix format only. Program must check the compatibility of orders of the matrices for multiplication. Report appropriate message in case of incompatibility.

**Program :**

```
#include <stdio.h>
#include<stdlib.h>
main()
{
    int   a[5][5] , b[5][5] ,p[5][5];
    int   i,j,k,m,n,p,q ;
    printf("\n Enter the order of the first matrix \n");
    scanf("%d %d",&m,&n) ;
    printf("\n Enter the order of second matrix \n");
    scanf("%d %d",&p,&q);
    if(n!=p)
    {
        printf("\nThe matrix multiplication cannot be performed \n");
        exit(0);
    }
    printf("\n Enter the matrix A  row wise \n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("\n Enter the matrix B column wise \n");
    for(j=0;j<q;j++)
    {
        for(i=0;i<p;i++)
        {
            scanf("%d",&b[i][j]);
        }
    }

/* Matrix Multiplication */
for(i=0;i<m;i++)
{
    for(j=0;j<q;j++)
    {
        p[i][j] =0 ;
    }
}
```

```

for(k=0;k<n;k++)
{
    p[i][j] =p[i][j] + a[i][k]*b[k][j] ;
}
}

printf("\n The resultant matrix is \n");

for(i=0;i<m;i++)
{
    for(j=0;j<q;j++)
    {
        printf ("%d\t",p[i][j]);
    }
    printf("\n");
}
}

```

**Sample Input/Output:**

**Run 1:**

Enter the order of first matrix : 2 2

Enter the order of second matrix : 2 2

Enter Matrix A row wise:

12 56

45 78

Enter Matrix B column wise:

2 5

6 8

The resultant matrix is:

304 520

480 894

**Run 2:**

Enter the order of first matrix : 2 2

Enter the order of second matrix : 3 3

The matrix multiplication cannot be performed

**8. Develop implement and execute a C program to search a name in a list of names using Binary Search Technique.**

**Program :**

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
main()
{
    char   name[10][20],key[20];
    int    n,i,low,high,mid;

    printf("\n Enter the number of names to read \n");
    scanf("%d",&n);

    printf("\n Enter the names in ascending order \n");
    for(i=0;i<n;i++)
        scanf("%s",name[i]);
    printf("\n Enter the name to be searched \n");
    scanf("%s",key);
    low = 0;
    high = n-1;
    while(low<=high)
    {
        mid = (low+high)/2 ;
        if(strcmp(name[mid],key)==0)
            { printf("\n Name found at position %d\n",mid+1);
              getch();
              exit(0);
            }
        else if (strcmp(name[mid],key)<0)
            { high = high ;
              low= mid +1;
            }
        else
            { low = low;
              high = mid -1;
            }
    }
    printf("\n Name not found \n");
}
```

**Output**

**Run 1:**

**Enter the number of names to read**

5

**Enter the names in ascending order**

Amar

Chethan

Gouri

Krishna

Naveen

**Enter the name to be searched**

Thyagu

**Name not found**

**Run 2:**

**Enter the number of names to read**

5

**Enter the names in ascending order**

Amar

Chethan

Gouri

Krishna

Naveen

**Enter the name to be searched**

Gouri

**Name found at position 3**

**9a. Write and execute a C program that implements string copy operation STRCOPY (str1,str2) that copies a string str1 to another string str2 without using library function .**

### **Program**

```
#include<stdio.h>
void  strcpy(char  s1[50],char s2[50])
{
    int i =0 ;
    while (s1[i]!='\0')
    {
        s2[i] = s1[i] ;
        i++;
    }
    s2[i] ='\0';
}
int main()
{
    char  str1[50] , str2[50] ;
    printf("\n Enter the source string: ");
    gets(str1);
    strcpy(str1,str2);
    printf("\n Destination String is :");
    puts(str2);
    return 0;
}
```

### **Output:**

#### **Run 1:**

Enter the source string : ABCDEF  
Destination String is : ABCDEF

#### **Run 2:**

Enter the source string : UJIRE  
Destination String is: UJIRE

**9b. Write and execute a C Program that read a sentence and print frequency of each of the vowels and total count of consonants.**

**Program :**

```
#include<stdio.h>
#include<string.h>
void main()
{
    char s[100],ch;
    int i,vowel=0,consonant =0;
    printf("\n Enter the sentence \n");
    gets(s);
    for(i=0;i<strlen(s);i++)
    {
        if(isalpha(s[i]))
        {
            ch = tolower(s[i]);
            if(ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u')
                vowel++;
            else
                consonant++;
        }
    }
    printf("\n Number of vowels in the given string is %d\n",vowel);
    printf("\n No of consonants in the given string is %d\n",consonant);
}
```

**Output**

**Run 1:**

Enter the sentence  
Hello Young India  
Number of vowels in the given string is 7  
No of consonants in the given string is 8

**Run 2:**

Enter the sentence  
Welcome to CSE  
Number of vowels in the given string is 5  
No of consonants in the given string is 7

**10a. Design and develop a C function rightshift(x,n) that takes two integers x and n as input and returns value of the integer x rotated to the right by n positions. Assume the integers are unsigned . Write a C program that involves this function with different values for x and n and tabulate the results with suitable headings.**

**BIT ROTATION :** A rotation (or circular shift) is an operation similar to shift except that the bits that fall off at one end are put back to the other end.

In **left rotation**, the bits that fall off at left end are put back at right end.

In **right rotation**, the bits that fall off at right end are put back at left end.

**Example:** Let n is stored using 8 bits.

**Left rotation of n** = 11100101 by 3 makes n = 00101111 (Left shifted by 3 and first 3 bits are put back in last ). If n is stored using 16 bits or 32 bits then left rotation of n (000...11100101) becomes00..001**1100101**000.

**Right rotation of n** = 11100101 by 3 makes n = 10111100 (Right shifted by 3 and last 3 bits are put back in first ) if n is stored using 8 bits. If n is stored using 16 bits or 32 bits then right rotation of n (000...11100101) by 3 becomes **101000..0011100**.

## Program :

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

unsigned int rightshift(unsigned int x,unsigned int n);

void main( )
{
    unsigned int x,y,n,i,value;

    do
    {
        printf("\n Enter the number \n");
        scanf("%u",&x);
        printf("\n Enter the number of bits \n");
        scanf("%u",&n);
        y = x;
        value = rightshift(x,n);

        printf("\n Right Rotated value of %u by %u is %u\n",y,n,value);
        printf("\n To continue press y else press 'n'\n");
        input = getch();
    } while (input!='n');

    }

unsigned int rightshift(unsigned int x,unsigned int n)
{
    int i;

    for(i=0;i<n;i++)
    {
        if(x%2==0)
            x=x>>1;
        else
        {
            x = x >> 1;
            x = x + 32768 ;
        }
    }
    return x;
}
```

### Sample Inout/Output: (For 16 Bit Rotation)

#### Run 1:

Enter the number

4

Enter the number of bits

1

**Right Rotated value of 4 by 1 is 2**

To continue press y else press 'n'

**n**

#### Run 2:

Enter the number

5

Enter the number of bits

2

**Right Rotated value of 5 by 2 is 16385**

To continue press y else press 'n'

**n**

**10b. Design and develop a C function *isprime(num)* that accepts an integer argument and returns 1 if the argument is prime and 0 otherwise. Write a C program that invokes this function to generate prime numbers between the given ranges.**

**Program :**

```
#include<stdio.h>
int isprime(int num);

int main()
{
    int x,y,i,flag =0;
    printf("\n Enter a range \n");
    scanf("%d %d",&x,&y);
    printf("\n The prime numbers are \n");
    for(i=x;i<=y;i++)
    {
        if(isprime(i))
        {
            printf("%d\t",i);
            flag =1;
        }
    }
    if(flag==0)
    {
        printf("\n There are no prime numbers between this range \n");
    }
}
int isprime(int num)
{
    int i;
    if(num==0||num==1)
    {
        return 0;
    }
    for(i=2;i<=sqrt(num);i++)
    {
        if(num%i==0)
        return(0);
    }
    return(1);
}
```

**Output:****Run 1:**

Enter a range

2 10

The prime numbers are

2 3 5 7

**Run 2:**

Enter a range

10 20

The prime numbers are

11 13 17 19

**Run 3:**

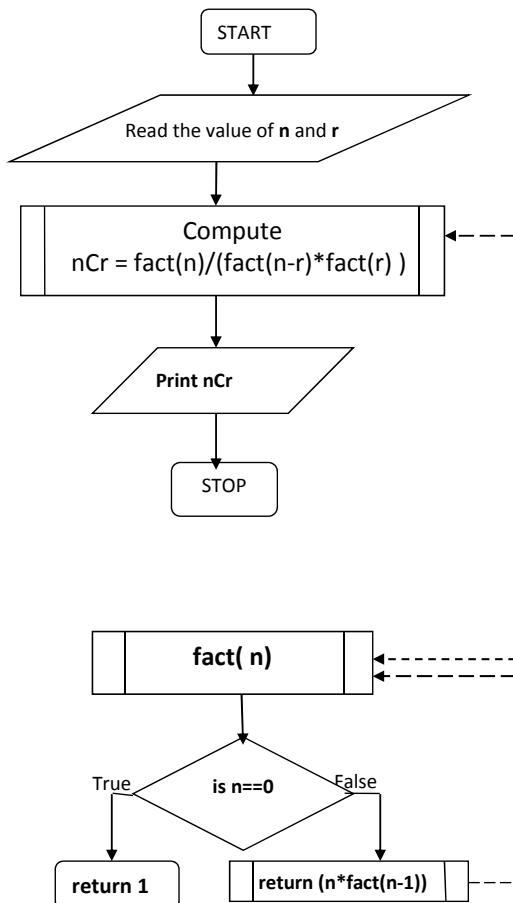
Enter a range

8 10

There are no prime numbers between this range

**11. Draw the flowchart and write a recursive C function to find the factorial of a number  $n!$  defined by  $\text{fact}(n) = 1$ , if  $n=0$  . Otherwise  $\text{fact}(n) = n * \text{fact}(n-1)$  .Using this function write a C program to compute the binomial coefficient  $nCr$  .Tabulate the results for different values of  $n$  and  $r$  with suitable messages.**

### Flow Chart



**Program :**

```
#include<stdio.h>
int fact(int) ;
void main()
{
    int n, nCr, r;
    printf("\nEnter the value of n and r \n");
    scanf("%d %d",&n,&r);
    nCr = fact(n)/(fact(n-r)*fact(r));
    printf("Binomial Coefficient nCr = %d",nCr);

}

int fact(int n)
{
    if(n==0)
    { return 1;
    }
    else
    { return n*fact(n-1);
    }
}
```

**Output:**

| Run# | Enter the value of n and r |   | Binomial Coefficient nCr = |
|------|----------------------------|---|----------------------------|
|      | n                          | r |                            |
| 1    | 4                          | 2 | 6                          |
| 2    | 5                          | 5 | 1                          |
| 3    | 5                          | 4 | 5                          |
| 4    | 5                          | 2 | 10                         |
| 5    | 7                          | 3 | 35                         |

**12.** Given two university information files “studentname.txt” and “usn.txt” that contains students Name and USN respectively . Write a C program to create a new file called “output.txt” and copy the content of files “studentname.txt” and “usn.txt” into output file in the sequence shown below.

| Student_name | USN   | Heading |
|--------------|-------|---------|
| Name1        | USN1  |         |
| Name2        | USN2  |         |
| -----        | ----- |         |
| Namen        | USNn  |         |

#### Program :

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

void main()
{
    FILE *fp1,*fp2,*fp3;
    char name[20],usn[20];
    fp1= fopen("studentname.txt", "r");
    if(fp1==NULL)
    {
        printf("\n studentname.txt is not found\n");
        exit(0);
    }
    fp2 = fopen("usn.txt", "r");
    if(fp2==NULL)
    {
        printf("\n usn.txt is not found\n");
        exit(0);
    }

    fp3= fopen("output.txt", "w");
    while(!feof(fp1)&&!feof(fp2))
    {
        fscanf(fp1, "%s\n",name);
        fscanf(fp2, "%s\n",usn);
        fprintf(fp3, "%s %s\n",name,usn);
    }
    fclose(fp1);
```

```

fclose(fp2);
fclose(fp3);

fp3 = fopen("output.txt", "r");

printf("\n NAME           USN\n");

while(!feof(fp3))
{
    fscanf(fp3, "%s",name);
    fscanf(fp3, "%s\n",usn);
    printf("%s %s\n",name,usn);
}
fclose(fp3);
}

```

**Sample Input/Output:**

Create two files : 1. studentname.txt and 2.usn.txt in the current working directory:

**1.studentname.txt**

Aravind  
Palguni  
Rashi  
Ravi  
John

**2.usn.txt**

2SD04CS123  
2SD04CS124  
2SD04CS125  
2SD04CS126  
2SD04CS127

The output of the program will be

**In the output.txt**

|         |            |
|---------|------------|
| Aravind | 2SD04CS123 |
| Palguni | 2SD04CS124 |
| Rashi   | 2SD04CS125 |
| Ravi    | 2SD04CS126 |
| John    | 2SD04CS127 |

**In the output device (Display Screen)**

|         |            |
|---------|------------|
| Aravind | 2SD04CS123 |
| Palguni | 2SD04CS124 |
| Rashi   | 2SD04CS125 |
| Ravi    | 2SD04CS126 |
| John    | 2SD04CS127 |

**13. Write a C program to maintain a record of n student details using an array of structures with four fields (*Roll number, Name, Marks and Grade*). Assume appropriate data types for each field. Print the marks of the student given the student name as input.**

### **Program:**

```
#include<stdio.h>

struct student
{
    int rno;
    char name[20];
    int marks;
    char grade;
};

void main()
{
    int i,n,found=0;
    char keyname[20];
    struct student s[20];

    printf("\nHow many student details ?");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\n Type in student no %d detail \n",i+1);
        printf("\n Roll no :");
        scanf("%d",&s[i].rno);
        printf("\n Name:");
        scanf("%s",s[i].name);
        printf("\n Marks:");
        scanf("%d",&s[i].marks);
        printf("\n Grade :");
        scanf(" %c",&s[i].grade);
    }
    printf("\n Key in student name to get his /her marks\n");
    scanf("%s",&keyname);

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

```
{ if(strcmp(s[i].name,keyname)==0)
    { printf("\n His/her marks is %d\n",s[i].marks);
        found =1;
    }
}
if(found ==0)
printf("\n This student Name does not exist in the given list\n");
}
```

#### **Sample Input/Output:**

How many student details ?

3

#### **Type in student no 1 detail**

Roll no :12

Name: Rishi

Marks: 89

Grade : A

#### **Type in student no 2 detail**

Roll no :12

Name:Swarna

Marks:75

Grade :B

#### **Type in student no 3 detail**

Roll no :55

Name: Poojary

Marks: 61

Grade : C

#### **Key in student name to get his /her marks**

Rishi

His/her marks is 89

**14. Write a C program using pointers to compute the sum ,mean and standard deviation of all elements stored in an array of n real numbers.**

**Note :**

- Mean =  $\frac{\sum_{i=0}^{n-1} xi}{n}$
- Variance =  $\frac{\sum_{i=0}^{n-1} (xi - mean) * (xi - mean)}{n}$
- Standard Deviation =  $\sqrt(\text{variance})$

**Program :**

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

int main()
{
    float a[10],*ptr , mean ,std,sum =0,var,sumstd=0;
    int n,i;

    printf("\n Enter the number of elements \n");
    scanf("%d",&n);
    printf("\n Enter the %d array elements \n",n);
    for(i=0;i<n;i++)
    { scanf("%f",&a[i]);
    }

    ptr = a;
    for(i=0;i<n;i++)
    {
        sum = sum+ *ptr;
        ptr++;
    }

    mean = sum/n;
    ptr = a;

    for(i=0;i<n;i++)
    {
        sumstd = sumstd + pow((*ptr- mean),2);
        ptr++;
    }
}
```

```
var = sumstd/n;
std = sqrt(var);
printf("\n Sum = %f",sum);
printf("\n Mean = %f",mean);
printf("\n Variance = %f",var);
printf("\n Standard deviation = %f",std);
}
```

**Sample Input/Output :**

Enter the number of elements

5

Enter the 5 array elements

34

88

32

12

10

Sum = 176.000000

Mean = 35.200000

Variance = 794.560000

Standard deviation = 28.190000

## Viva Questions

1. Define Algorithm, Flowchart and Psedocode.
2. Mention the characteristics of Algorithm.
3. What is Program?
4. What is C Program ?
5. Mention the characteristics or features of C programming Language.
6. What is the history of C programming Language?
7. What are header files? Give Example.
8. What is stdio.h?
9. Explain stdio.h, math.h and conio.h .
10. Differentiate between printf and scanf.
11. What are formatted and unformatted input output statements? Give examples.
12. What is conversion or format specifier? List some of them.
13. Define Loop?
14. What are Entry controlled and Exit controlled loops in C?
15. Give example for entry controlled loop and exit controlled loop.
16. What are Looping control statements?
17. Explain while loop.
18. What is the difference between while and for loops?
19. Write the flowchart for 'while' loop.
20. Write the flowchart for 'do while' loop
21. What is typecasting? Explain with examples.
22. Difference between float and double data types.
23. Explain for loop?
24. What is a use of break statement?
25. Difference between continue and break statement?
26. Why **n++ executes faster than n+1?**
27. Explain about % and ! Operator.
28. **What is the operator precedence and operator associativity?**
29. Which of the operators has associativity from Right to Left?
30. List the operators having same precedence?
31. Which of the following operator has the highest precedence in the following?
  - a) ( )
  - b) sizeof
  - c) \*
  - d) +
32. What is an array? Give an example
33. What is a multidimensional array?
34. How to declare and initialize one dimensional array?
35. What are the advantages of an array?
36. Differentiate between i++ and i-- ?

- 37.**What is pre-processor directive?
- 38.**What is difference between const and #define.
- 39.**What is use of fabs().
- 40.**What is variable initialization and why is it important?
- 41.**What is the difference between the = symbol and == symbol?
- 42.**Can the curly brackets { } be used to enclose a single line of code?
- 43.**Why the name bubble sort?
- 44.**Mention the different types of sorting techniques?
- 45.**Explain the logic of bubble sort with an example.
- 46.**What is nested for loop?
- 47.**What is a multi-dimensional array?
- 48.**How to initialize two dimensional arrays?
- 49.**How to pass a two dimensional array as function parameter?
- 50.**How the memory is allocated for two dimensional array
- 51.**Explain how the binary search program works?
- 52.**Which are the different types of searching techniques?
- 53.**What is difference between linear search and binary search?
- 54.**What are the advantages and disadvantages of binary search techniques?
- 55.**Define a Function?
- 56.**What is the difference between User-defined and Standard library/Pre-defined functions?
- 57.**Is main() a user-defined function or pre-defined function?
- 58.**What is the Calling function and Called function?
- 59.**What are the Actual parameters /arguments and Formal parameters/arguments?
- 60.**Explain the concept of pass by value / call by value?
- 61.**Explain the void type in C?
- 62.**What is NULL Character?
- 63.**What is the difference between character and a string?
- 64.**What are string constants?
- 65.**What are character constants?
- 66.**Mention some built-in functions of string.h header file
- 67.**What is isalpha () & tolower() function?
- 68.**What is strlen(), strcpy(), strcat() and strcmp( ) function works?
- 69.**Explain bitwise operators in C?
- 70.**What is difference between integer and unsigned integer?
- 71.**Explain the function without return type and return value with example?

- 72.**What is difference between getch() and getche()?
- 73.**What are library functions?
- 74.**Why is the return type used?
- 75.**What is function prototype?
- 76.**What is the default value returned by a function?
- 77.**Which is the even prime number?
- 78.**Define the term recursion.
- 79.**Explain how recursive function works.
- 80.**What are the advantages and disadvantages of recursive function.
- 81.**Give the general syntax of a recursive function.
- 82.**What is file? File is type of \_\_\_\_\_? **Ans: Struct type**
- 83.**What is file mode ? Mention the different types of file mode ?
- 84.**What is pointer and NULL pointer?
- 85.**Write the syntax of fopen( ) and fclose ( )?
- 86.**How the fgetc( ) and fputc( ) works ?
- 87.**EOF indicate\_\_\_\_\_? **Ans : End – of – file (EOF) is used to report end of file.**
- 88.**What is difference between fgetc( ) and fgets( )?
- 89.**Define structure?
- 90.**Give the difference between an array and a structure?
- 91.**Does the definition of a structure create memory space?
- 92.**How do you access a structure variable?
- 93.**What is a enumerated data type in c?
- 94.**Define union?
- 95.**What are the difference between structure and union?
- 96.**What is the use of fflush( ).
- 97.**Difference between **strcmp()** and **strcmpi()**.
- 98.**Define pointer?
- 99.**How do you declare a pointer variable?
- 100.**What is \* and &in pointer concept.
- 101.**What are the advantages and disadvantages of using pointer?
- 102.**Give the difference between static allocation and dynamic allocation of memory space.
- 103.**Mention the different memory allocation functions.
- 104.**What is the effect of the ++ and -- operators on pointer variable
- 105.**Explain the pointers to arrays concept?
- 106.**How do you construct an increment statement or decrement statement in C?
- 107.**What is the difference between Call by Value and Call by Reference?

- 108.** Some coders debug their programs by placing comment symbols on some codes instead of deleting it. How does this aid in debugging?
- 109.** What is the equivalent code of the following statement in WHILE LOOP format?
- 110.** What is a stack?
- 111.** What is a sequential access file?
- 112.** What is variable initialization and why is it important?
- 113.** What is spaghetti programming?
- 114.** Differentiate Source Codes from Object Codes
- 115.** In C programming, how do you insert quote characters (‘ and “) into the output screen?
- 116.** What is the use of a ‘\0’ character?
- 117.** What is the difference between the = symbol and == symbol?
- 118.** What is the modulus operator?
- 119.** What is a nested loop?
- 120.** Which of the following operators is incorrect and why? ( >=, <=, <>, ==)
- 121.** Compare and contrast compilers from interpreters.
- 122.** How do you declare a variable that will hold string values?
- 123.** Can the curly brackets { } be used to enclose a single line of code?
- 124.** What are header files and what are its uses in C programming?
- 125.** What is syntax error?
- 126.** What are variables and it what way is it different from constants?
- 127.** How do you access the values within an array?
- 128.** Can I use “int” data type to store the value 32768? Why?
- 129.** Can two or more operators such as \n and \t be combined in a single line of program code?
- 130.** Why is it that not all header files are declared in every C program?
- 131.** When is the “void” keyword used in a function?
- 132.** What are compound statements?
- 133.** What is the significance of an algorithm to C programming?
- 134.** What is the advantage of an array over individual variables?
- 135.** Write a loop statement that will show the following output:
- 136.** What is wrong in this statement? scanf(“%d”,whatnumber);
- 137.** How do you generate random numbers in C?
- 138.** What could possibly be the problem if a valid function name such as tolower() is being reported by the C compiler as undefined?
- 139.** What are comments and how do you insert it in a C program?
- 140.** What is debugging?
- 141.** What does the && operator do in a program code?
- 142.** In C programming, what command or code can be used to determine if a number of odd or even?
- 143.** What does the format %10.2 mean when included in a printf statement?

- 144.** What are logical errors and how does it differ from syntax errors?
- 145.** What are the different types of control structures in programming?
- 146.** What is || operator and how does it function in a program?
- 147.** Can the “if” function be used in comparing strings?
- 148.** What are preprocessor directives?
- 149.** What will be the outcome of the following conditional statement if the value of variable s is 10?
- 150.** Describe the order of precedence with regards to operators in C.
- 151.** What is wrong with this statement? myName = “Robin”;
- 152.** How do you determine the length of a string value that was stored in a variable?
- 153.** Is it possible to initialize a variable at the time it was declared?
- 154.** Why is C language being considered a middle level language
- 155.** What are the different file extensions involved when programming in C?
- 156.** What are reserved words ?
- 157.** What are linked list?
- 158.** What is FIFO ?
- 159.** What are binary trees?
- 160.** Not all reserved words are written in lower case . TRUE or FALSE?
- 161.** What is the difference between the expression “++a” and “a—“ ?
- 162.** What would happen to x in this expression : $x+=5;$  (assuming the value of x is 5)
- 163.** In C language the variable NAME,name and Name are all the same. TRUE or FALSE?
- 164.** What is an endless loop ? or infinite Loop.
- 165.** What is a program flowchart and how it help in writing a program ?
- 166.** What is wrong with this program statement ? void =10
- 167.** Is this program statement valid ? INT = 10.50
- 168.** What are actual arguments ?
- 169.** What is a newline escape sequence ?
- 170.** What is output redirection ?
- 171.** What are run time errors ?
- 172.** What is the difference between functions abs() and fabs()?
- 173.** What are formal parameters ?
- 174.** What are control structures ?
- 175.** Write a simple code fragment that will check if a number is positive or negative
- 176.** When is a “switch” statement preferable over an if statement ?
- 177.** What are global variables and how do you declare them ?
- 178.** What are enumerated types?
- 179.** What is “switch” statement preferable over an “if” statement?
- 180.** What are global variables and how do you declare them?
- 181.** What are enumerated types ?

- 182.** What does the function toupper() do?
- 183.** Is it possible to have a function as a parameter to another function ?
- 184.** What are multidimensional arrays?
- 185.** Which function in C can be used to append a string to another string ?
- 186.** What is the difference between function getch() and getche()?
- 187.** Do these two program statements perform the same output? 1. scanf("%c",&letter) ; 2) letter=getchar()
- 188.** What are structure types in C?
- 189.** What does the characters “r” and “w” mean when writing programs that will make use of files ?
- 190.** What is the difference between text files and binary files?
- 191.** Is it possible to create your own header files ?
- 192.** What is dynamic data structure ?
- 193.** What are the different data types in C?
- 194.** What is the general form of a C program?
- 195.** What is the advantage of a random access file ?
- 196.** In a switch statement , what will happen if a break statement is omitted ?
- 197.** Describe how arrays can be passed to a user defined functions ?
- 198.** What are pointers ?
- 199.** Can you pass an entire structure to functions ?
- 200.** What is gets() function ?
- 201.** The % symbol has a special use in printf statement. How would you place this character as part of the output on the screen?
- 202.** How do you search data in a data file using random access method?
- 203.** Are comments included during compilation stage and placed in the EXE file as well?
- 204.** Is there a built in function in C that can be used for sorting data?
- 205.** What are the advantages and disadvantages of a heap?
- 206.** How do you convert strings to numbers in C?
- 207.** Create a simple code fragment that will swap the values of two variables n1 and n2.
- 208.** What is the use of a semicolon (;) at the end of every program statement?
- 209.** What is a keyword. Difference between keyword and identifier?
- 210.** What are the different operators available in C( Study in detail about each operator)
- 211.** Explain about conditional operator
- 212.** What are the different data types available in C
- 213.** List the control statement available in C
- 214.** List out the differences between while and do while?
- 215.** Explain the working of break and continue
- 216.** What are the searching methods available for arrays?

- 217.** What is the difference between bubble sorting and selection sorting
- 218.** Explain about strings. How strings are handled in C?
- 219.** What are the string handling functions available in C
- 220.** What is a function. What are the benefits of a function?
- 221.** What is recursion?
- 222.** Explain about macro substitution
- 223.** What are the differences between structure and an array?
- 224.** List out the differences between structure and union?
- 225.** What is a pointer?
- 226.** Explain the concept of chain of pointers?
- 227.** What is command line argument?
- 228.** What is a linked list. List the different types of linked list
- 229.** What are the functions used in dynamic memory management?
- 230.** Explain about storage classes available in C
- 231.** What are the file management functions available in C
- 232.** How to open and close a file?
- 233.** What do you mean by Hardware and Software?
- 234.** Mention the main components of a computer and their functions
- 235.** What is Operating System(OS) ?
- 236.** Name the four data types in C language?
- 237.** Describe at least five different format specifiers?
- 238.** Define and explain scanf() function ?
- 239.** Define and explain printf() function?
- 240.** What are the maximum and minimum possible ranges of values for long and short type?
- 241.** What is preprocessor?
- 242.** What exactly is ‘variable scope’, ‘local variables’ and ‘global variable’ ?
- 243.** What are signed values?
- 244.** Define reserved words.
- 245.** What is identifier?
- 246.** Explain While Loop?
- 247.** Explain for Loop?
- 248.** Explain do while Loop?
- 249.** List a few unconditional control statement in C.
- 250.** List a few unconditional control statement in C.
- 251.** What is an array?
- 252.** What is multidimensional arrays?
- 253.** Define String?
- 254.** Mention four important string handling functions in C languages.
- 255.** Explain about the constants which help in debugging?
- 256.** Define and explain about ! operator?
- 257.** What is function ?
- 258.** Differentiate between built in function and user defined functions.

- 259.** Distinguish between actual and formal arguments.
- 260.** Explain the concept and use of type void
- 261.** What is recursion ?
- 262.** What are Library functions?
- 263.** Mention the types of network
- 264.** How does the type float differ from double in C language?
- 265.** What is an operator and operand ?
- 266.** What is RAM?
- 267.** What is ROM?
- 268.** Define System Software.
- 269.** Define Application Software
- 270.** What are control Statements ?
- 271.** Explain increment and decrement operators.
- 272.** Mention the types of memory.
- 273.** What are input and output device?
- 274.** What is a Computer?
- 275.** What is CPU?
- 276.** What is ALU?
- 277.** What is CU?
- 278.** What is RAM?
- 279.** What is ROM?
- 280.** What is PROM?
- 281.** What is EPROM?
- 282.** What is EEPROM?
- 283.** Differentiate between Volatile and non-volatile memory.
- 284.** What are the input and output devices?
- 285.** What is the need for cache memory?
- 286.** Differentiate between main memory and second memory.
- 287.** What is hardware?
- 288.** Mention the different types of hardware components?
- 289.** What are the different types of keyboards?
- 290.** Differentiate between serial and parallel keyboard.
- 291.** How do you classify printers?
- 292.** Differentiate between serial and parallel printer.
- 293.** Differentiate between impact and non-impact printers.
- 294.** Differentiate between line and page printers.
- 295.** Which printer do you select when high quality output is to be produced?
- 296.** What is meant by softcopy output?
- 297.** What is meant by hardcopy output?
- 298.** What is printer buffer?
- 299.** What is meant by tracks?
- 300.** What is meant by sectors?
- 301.** What is CD-ROM?

- 302.** What are the different types of mouse?
- 303.** What are the main parts of a floppy disk?
- 304.** Differentiate between magnetic disks and optical disks.
- 305.** What is programming?
- 306.** What is software?
- 307.** What are the different types of software?
- 308.** What are the different types of programming languages?
- 309.** Differentiate between interpreter and compiler.
- 310.** Differentiate between loader and linker.
- 311.** Differentiate between Application software and System software.
- 312.** What is an operating system?
- 313.** What are language processors?
- 314.** Mention some operating systems.
- 315.** Differentiate between compiler and assembler.
- 316.** What is translator?
- 317.** What is meant by interpretation?
- 318.** What is meant by source program?
- 319.** What is meant by object program?
- 320.** Give examples for High Level Languages.
- 321.** Give examples for Assembly Level Languages.
- 322.** Give examples for General purpose HLL.
- 323.** Give examples for Specific purpose HLL.
- 324.** Differentiate between Analog and Digital computers.
- 325.** Differentiate between microcomputer & minicomputer.
- 326.** Differentiate between internal & external DOS commands.
- 327.** How do you classify the computers based on the size & capability?
- 328.** How do you classify the computers based on principle of working?
- 329.** What is computer network?
- 330.** How do you classify the computer networks?
- 331.** What is Batch processing?
- 332.** What is Time Sharing?
- 333.** Differentiate between LAN and WAN.
- 334.** What are the different types of DOS?
- 335.** What is DOS?
- 336.** What is Real Time Systems?
- 337.** What are the advantages of computer network?
- 338.** What is BCPL?
- 339.** Who developed BCPL?
- 340.** Who developed B language?
- 341.** Who developed C language?
- 342.** How do you make comments in C program?
- 343.** How the name C is derived?
- 344.** What is K & R C?
- 345.** What is preprocessor statement?

- 346.** Differentiate between constant and variable.  
**347.** What is data type?  
**348.** Name the basic data types of C.  
**349.** Differentiate between string constant and character constant.  
**350.** What is the range of integer, char, float for a 16-bit computer?  
**351.** What is a statement?  
**352.** What is a keyword?  
**353.** Differentiate between keywords and identifiers.  
**354.** What is the need for an escape sequences?  
**355.** What is a symbolic constant?  
**356.** How do you classify C operators?  
**357.** What is the use of modulus operator?  
**358.** What is meant by mixed mode operation?  
**359.** What are bitwise operators?  
**360.** What is unary operator?  
**361.** What is binary operator?  
**362.** What is typecasting?  
**363.** What is a conditional / ternary operator?  
**364.** What is need for type conversion?  
**365.** Differentiate between && and &.  
**366.** Differentiate between pre-increment/decrement & post-increment/decrement.  
**367.** Differentiate between Unformatted and formatted i/o statements.  
**368.** How do you classify the control statements?  
**369.** Differentiate between while and do-while loop.  
**370.** Differentiate between break and continue.  
**371.** When do you prefer for loop statement?  
**372.** What is looping?  
**373.** What is an array?  
**374.** Give the classification of arrays?  
**375.** Differentiate between an array and an ordinary variable.  
**376.** Array variable is also called as \_\_\_\_\_.  
**377.** What are character arrays?  
**378.** When do you use two-dimensional character array?  
**379.** Name the different string handling functions?  
**380.** What is meant by modularization?  
**381.** Differentiate between standard functions & user-defined functions?  
**382.** Differentiate between arguments and parameters.  
**383.** Differentiate between local and global variables.  
**384.** Name the different methods of parameter passing?  
**385.** How does the function definition differ from function declaration?  
**386.** What is recursive function?  
**387.** What is meant by scope of a variable?  
**388.** What is a structure?

389. Differentiate between array and structure.
390. What are embedded structures?
391. How do you access the member of a structure?
392. What is union?
393. Differentiate between union and structure.
394. What is a pointer?
395. Differentiate between address operator and dereferencing operator.

## Additional Viva & Exercise Questions

### 1. Basic concepts of C program

#### Viva Questions

1. What does **include** stands for?
2. Give some examples of built-in functions stored in math.h , conoi.h ,stdio.h,string.h.
3. Define **preprocessor directives**.
4. What is the syntax for preprocessor directive?
5. What are **header files**?
6. What type of a function is **main()**?
7. What are **variables**?
8. Name the basic data type available in C.
9. What is the size of following data type?
  - a) int b) float c) char d)long int e) double
10. What are **format specifiers**?
11. Name the format specifiers for the following data type.
  - a) int b) float c) char d)long int e) double
12. Name formatted input output functions.
13. Give the syntax for printf() and scanf().
14. In which header file, printf() is defined?
15. What are the arguments taken by the function printf( )?
16. What are the two arguments taken by scanf()?
17. What is the output of the following code?

```
void main()
{
    int a;
    printf ("%d",x);
}
```
18. Name unformatted input output functions.
19. How many characters is read by the function getchar() and gets ()?

20. How do you differentiate getche() and getchar()?
21. What is the significance of using getch() in the program?
22. What are **compound statements**?
23. What are **built-in functions**?
24. Define keywords?
25. Mention some of the keywords available in C?
26. Keywords are also known as \_\_\_\_\_.
27. What are **Identifiers**?
28. What are **Constants**?
29. What are **qualifiers**?
30. What is the range of unsigned integer?
31. What is **const**? Where is it used?
32. Give an example for character constant.
33. What is **ASCII value**?
34. How characters are stored in memory?
35. How many bits is equal to 1 byte?
36. Half byte is known as \_\_\_\_\_.
37. What is a **compiler**?
38. What is an **assembler**?

### Exercise

1. Write a program to print your name, college, usn.
2. Write a program to print the pattern \* \* \* \* \*  
\* \* \* \* \*  
\* \* \* \* \*
3. Write a program to read a number and display the number.
4. Write a program to add, subtract, multiply and divide two integers.
5. Write a program to read a character and display it using unformatted input output functions.
6. Write a program to find the ASCII value of a character.

## 2. Operators

### Viva Questions

1. What is an **operator**?
2. Name the different types of operators in C.
3. What is **associativity**?

4. Arithmetic operators are left to right associative. (*True / false*)
5. How multiple assignments are done? Is it left to right or right to left associative?
6. Name relational operators? It is also known as \_\_\_\_\_.
7. What are **unary operators**?
8. Give some examples for unary operators.
9. What is an **address operator**?
10. Give the syntax for ternary operator.
11. What type of operator is increment operator?
12. Name logical operator. Why is it used?
13. Which operator has highest precedence among logical operators?
14. What type of operators are >> and <<?
15. = is an example of ----- operator.
16. What is **declaration**?
17. Give an example of initialization of variable. How it works?
18. What are **local variables and global variables**?
19. What does this indicates \a and \t?
20. Differentiate implicit and explicit type conversion.
21. **Explicit type conversion** is known as \_\_\_\_\_.
22. Both the variables should be of same data type in implicit type conversion (*True/false*)?
23. Explain explicit conversion with syntax.
24. What is the output of following code?

```
void main ()  
{   int   a=6, c;  
    float b=9.0;  
    c=a/b;  
    printf("%d",c);  
}
```

25. What is the output of the following code?

```
void main ()  
{  
    int a=10,b=5;  
    float c;  
    c = (float) a/b;  
    printf("%f",c);  
}
```

26. Differentiate post and pre increment operator.
27. Write the output for the following code?

```
void main ()  
{  
    int a=8, b, c=10, d;  
    b = ++a;  
    printf("%d\n%d",a,b);  
    d=c++;  
    printf("%d\n%d",c,d);  
}
```

### Exercise

1. Write a program to demonstrate **arithmetic operators**.
2. Write a program to print the right most digit of a number.
3. Write a program to demonstrate **relational operators**.
4. Write a program to demonstrate **logical operators**.
5. Write a program to demonstrate **bitwise operators**.
6. Write a program to demonstrate post& pre increment of a variable (post & pre decrement)?
7. Write a program to find the size of all data type using **sizeof( )** operator.
8. Write a program to find largest to two integers using **conditional operator**.
9. Write a program to swap two integers with and without using temporary variables.
10. Write a program to convert degrees into radians and vice versa.
11. Write a program to convert Celsius into Fahrenheit.
12. Write a program to demonstrate literal constant defined constant and memory constant.

## **3. Branching and Looping**

### Viva Questions

1. Name conditional branching statements.
2. Write the syntax for **if**.
3. Write the syntax for **switch case** and **else if ladder**.
4. Name the different instance where **switch case** and **else if ladder** is used.
5. What is **default**? What is the significance of using it in **switch case**?
6. Can we use **continue** in switch case?
7. Can the case value be float in switch case?
8. Explain unconditional branching statement.

- 9.** What is **break**?
- 10.** Explain the syntax for **goto**? Why it is not recommended in C?
- 11.** How many times conditional branching statements will work?
- 12.** Differentiate event controlled and counter controlled loop.
- 13.** Explain the syntax of **while** and **do-while** loop.
- 14.** The minimum times do-while loop will work-----.
- 15.** While loop is also known as \_\_\_\_\_.(**entry controlled loop**)
- 16.** Why **do-while** loop is also known as exit-controlled loop? Justify.
- 17.** Explain the syntax of **for** loop.
- 18.** Explain **nested for** loop.
- 19.** Write the code to print the following code.

```
*  
* *  
* * *
```

- 20.** What is the output of the above code?
- |                                                                                               |                                                                                           |
|-----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <pre>for(i=0;i&lt;5;i++)<br/>{<br/>    if(i==3) continue;<br/>    printf("%d",i);<br/>}</pre> | <pre>for(i=0;i&lt;5;i++)<br/>{<br/>    if(i==3)break;<br/>    printf("%d",i);<br/>}</pre> |
|-----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
- 21.** Write for loop in terms of while loop.
- 22.** What type of loop is this. for ( ; ; )
- 23.** Give a real time example where **nested for** loop is used.
- 24.** Which is the preferable loop to find gcd of a number?
- 25.** What is the advantage of using Horner's method over simple method?
- 26.** Name the different instances where **break** statement is used.

### Exercise

1. Write a program to check whether a person is eligible to vote using **if** construct.
2. Write a program to check whether a student has secured distinction using **if** construct.
3. Write a program to check whether a number is positive, negative or zero using
4. Write a program to find largest of two numbers using **if else** statement.
5. Write a program to check whether a number is even or odd using **if else** statement.

6. Write a program to check whether the entered digit is vowel or consonant using **if else** statement.
7. Write a program to find largest of three numbers using nested **if else** statement.
8. Write a program to find largest of three numbers using **else if ladder**.
9. Write a program to check whether a number is positive negative or zero using **else if ladder**.
10. Write a program to perform arithmetic operations based on the users choice using **else if ladder**.
11. Write a program to perform arithmetic operations based on users choice using **switch** statement.
12. Write a program to calculate area of circle area of rectangle and area of square based on users choice using **switch** statement.
13. Write a program to check whether an entered character is vowel or consonant using **switch** statement.
14. Write a numbers program to print the numbers 1 to 10 using **while** loop.
15. Write a program to reverse a number using **while** loop.
16. Write a program to find factorial of a number using **while** loop.
17. Write a program to find factorial of a number using **do-while** loop.
18. Write a program to convert binary to decimal and vice versa using **do-while** loop.
19. Write a program to find fibonacci of a given number using **for** loop.
20. Write a program to check whether a number is divisible by 3 using **for** loop.
21. Write a program to calculate pow(x,n) using **for** loop.
22. Write a program to find the sum of digits of a number using **for** loop.
23. Write a program to find the sum of series  $1+1/2+1/3+\dots+1/n$  using **for** loop
24. Write a program to print the following patterns using **nested for** loop
  - a) 1  
    1 2  
    1 2 3
  - b) \* \* \*  
    \* \* \*  
    \* \* \*
  - c) 1 2 3  
    1 2  
    1
25. Write a program to illustrate the usage of unconditional branching statements (break, continue and goto).

## 4. Arrays and Strings

### Viva Questions

1. What is an **array**?
2. Name the different types of arrays.
3. Can we store name usn and marks of a student in the single array?
4. Give the syntax for declaring **One Dimension array**.
5. What are the different ways of 1D array initialization?
6. What does the subscript of the array indicates?
7. How do we access the value stored in array?
8. Can we declare a two Dimension array without size? Justify.
9. Explain the working of **linear search**.
10. Name the different sorting techniques.
11. How binary search is more efficient than linear search?
12. What is a string?
13. In which header file all the string built-in function are stored?
14. Why address operator is not used while reading a string?
15. Generally gets() is used to read a string instead of scanf(). Why?
16. On which basis string comparison is done using the function **strcmp()**?
17. How many arguments does the function **strcat()** takes?
18. What does '\0' represents?
19. What is the size of 0 and \0?
20. What is a null character?
21. Differentiate character constant and string constant?
22. How do you represent string constant?

### Exercise

1. Write a program to find the sum and average of all elements of 1D array
2. Write a program to find the largest of array element.
3. Write a program to perform two one dimension array addition.
4. Write a program to perform binary search on array elements.
5. Write a program to read and display the elements of two dimension array.
6. Write a program to find the sum and average of all elements of two dimension array.
7. Write a program to fill upper triangular and lower triangular matrix with 0 and diagonal elements with 1.

8. Write a program to find norm of a matrix.
9. Write a program to perform linear search.
10. Write a program to read a string and display it.
11. Write a program to find the length of a string using and without using built-in function.
12. Write a program to concatenate the string using and without using built-in function.
13. Write a program to reverse a string using and without using built-in function.
14. Write a program to compare the string using and without using built-in function.
15. Write a program to find the occurrence of a character in a string using and without using built-in function.
16. Write a program to reverse a string and check whether it is palindrome or not.

## 5. Functions

### Viva Questions

1. Define **function**.
2. What is **user defined function** and what is the need of it?
3. Name the three elements of user defined function.
4. Write the syntax for function declaration and function call.
5. What does return type in function declaration indicate?
6. What are actual and formal parameters?
7. How do we declare formal parameters?
8. Is it necessary to mention the name of variable in the function declaration (**True / False**)?
9. What is a function call?
10. Write the syntax for actual parameters.
11. What are three aspects actual parameters and formal parameters should match?
12. How many return statements can be coded in a program?
13. What are the two ways of parameter passing technique?

### Exercise

1. Write a program to perform addition of two numbers using **function**.
2. Write a program to find factorial of a number using **function**.
3. Write a program to swap two integers using **function**.
4. Write a program to check whether any character from string 2 is present in string 1 without using built-in function.

## 6. Pointers and Recursion

### Viva Questions

1. What is a **pointer**?
2. How to declare a pointer?
3. What does a pointer hold?
4. What is a null pointer?
5. \* is also known as \_\_\_\_\_ operator.
6. Can we have pointer to pointer?
7. Differentiate a normal variable and a pointer.
8. What is recursion? Give an example.
9. Differentiate iteration and recursion.

### Exercise

1. Write a program to find factorial of a number using **recursion**.
2. Write a program to demonstrate tower of Hanoi using **recursion**.
3. Write a program to declare and initialize the pointers.
4. Write a program to illustrate pointer to pointer concept.
5. Write a program to illustrate array of pointers.

## 7. Data Structures

### Viva Questions

1. What is **data structure**?
2. What are the two operations implemented on stack?
3. What is queue?

4. List the operations on queue.
5. What is importance of the top in the stack?
6. Give an example of first in first out data structure.
7. What is a **queue**?

### **Exercise**

1. Write a program to create stack using arrays.
2. Write a program to create queue using arrays.

## **8. Structures and unions**

### **Viva Questions**

1. What is a structure?
2. Name the derived data types.
3. What is advantage of using structure over an array?
4. Give the syntax of structure.
5. How the structure elements are accessed?
6. What is union?
7. What is the total memory allocated for union?

### **Exercise**

1. Write a program to create structures.
2. Write a program to implement array of structures.
3. Write a program to implement pointer to array of structures

## 9. References

1. Brian W. Kernighan and Dens M.Ritchie : The C Programming Language ,2<sup>nd</sup> Edition ,PHI, 2012
2. Jacqueline Jones & Keith Harrow : Problem Solving with C ,1<sup>st</sup> Edition , Pearson 2011.
3. Vikas Gupta : Computer concepts and C Programming , Dreamtech Press 2013
4. R S Bichkar, Programming with C , University Press 2012.
5. V. Rajaraman : Computer Programming in C , PHI , 2013
6. J.B.Dixit : Computer Concepts and Programming ,Laxmi Publications , 2010
7. G.S. Baluja: Data Structures Through C , DhanPat Rai and Co, 2012
8. A.M. PadmaReddy C : Programming and Data Structures , Nandi Publications , 2014
9. N.Guruprasad : Computer Concepts and C programming ,Himalaya Publishing House, 2006
10. Rajesh Hongal , Computer Concepts and C programming , Eastern Book Promoters, 2014
11. Reema Thareja , Computer Fundamentals and Programming in C ,Oxford University Press, 2012
12. Herbert L. Cooper, The Spirit of C, An introduction to Modern Programming , Jaico Publishing House.
13. <http://career.guru99.com>
14. <http://www.indiabix.com/>
15. <http://geeksquiz.com/>
16. <http://www.cprogrammingexpert.com>
17. <http://www.studytonight.com>
18. <http://www.w3schools.in>
19. <http://www.thegeekstuff.com>

- 20.<http://www.sanfoundry.com>**
- 21.<http://www.peoi.org>**
- 22.<http://www.mycplus.com>**
- 23.<http://www.stackoverflow.com>**
- 24.<https://en.wikipedia.org>**
- 25.<https://www.quora.com>**
- 26.<http://www.programiz.com>**
- 27.<http://www.tutorialspoint.com>**
- 28.<http://www.fresh2refresh.com>**
- 29.<http://www.cs-fundamentals.com>**
- 30.<http://www.c4learn.com/>**

## 10. Appendix

### 1. Command Line Arguments

The values that are passed from the command line to the C programs are called command line arguments. The command line arguments are handled using main() function arguments argc and argv[ ]. The argc is a integer type and refers to the number of argument passed. The argv[ ] is a pointer array which points to each argument passed to the program .

| Command Line        | argc | argv[ ]                   |
|---------------------|------|---------------------------|
| ./a.out             | 1    | ./a.out                   |
| ./a.out test        | 2    | ./a.out , test            |
| ./a.out test1 test2 | 3    | ./a.out , test1 and test2 |
| C:\tc\bin> add 1 2  | 3    | add,1, and 2              |

**Example 1:** Write a C program on Linux to print the arguments in the command line

```
#include<stdio.h>
int main(int args, char *argv[])
{
    int i = 0;
    printf("\n The list of command line arguments are :\n");
    for (i = 0; i < args; i++)
        printf("\n%fs", argv[i]);
    return 0;
}
```

#### Output :

```
$./a.out one two 3
The list of command line arguments are :
./a.out one two 3
```

**Example 2:** Write a C program to add two numbers using command line arguments

```
#include<stdio.h>
int main(int args, char *argv[])
{
    int i,sum =0;
    if(args!=3){
        printf("\n You have forgotten to type numbers \n");
        exit(1);
    }
}
```

```
}

printf("\n The sum is :");
for(i=1;i<args;i++)
sum = sum +atoi(argv[i]);
printf("%d",sum);
return 0;
}
```

### Output :

```
$./a.out 1 3
The sum is 4
```

## 2. String Conversion Functions

The string conversion functions, *atoi, atol and atof* are available under *stdlib.h* .

1. **atoi()** :This function will convert a string to an integer. Usage of *atoi()*:

```
int atoi ( const char * str );
```

**Parameters:** C string **str** interpreting its content as a integer. White-spaces are discarded until the first non-whitespace character is found.

**Return value:** The function returns the converted integral number as an int value, if successful. If no valid conversion could be performed then an zero value is returned. If the value is out of range then INT\_MAX or INT\_MIN is returned.

### Programming Example of atoi():

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

int main ()
{
    int i;
    char buffer[200];

    printf ("Enter a number: ");
    fgets (buffer, 256, stdin);

    i = atoi (buffer);
    printf ("The value entered is %d.",i);
```

```
    return 0;  
}
```

### Output

```
Enter a number: 12  
The value entered is 12.
```

- 2. `atof()`** : This function will convert a string to a double. Usage of `atof()`:  
**`double atof ( const char * str );`**

**Parameters:** C string `str` interpreting its content as a floating point number. White-spaces are discarded until the first non-whitespace character is found. A valid floating point number for `atof` is formed by: plus or minus sign, sequence of digits, decimal point and optional exponent part (character ‘e’ or ‘E’)

**Return value:** The function returns the converted floating point number as a double value, if successful. If no valid conversion could be performed then an zero value is returned.

### Programming example of `atof ()`:

```
#include<stdio.h>  
#include<stdlib.h>  
int main ()  
{  
    float a,b;  
    char buffer[200];  
    printf( "Input: " );  
    gets(buffer);  
    a = atof(buffer);  
    b = a/2;  
    printf( "a= %f and b= %f\n" , a, b );  
    return 0;  
}
```

### Output of the `atof` example program above:

```
Input: 12  
a= 12.000000 and b= 6.000000
```

**3. atol()** : This function will convert a string to a long integer. Usage of atol():

```
long int atol ( const char * str );
```

**Parameters:** C string **str** interpreting its content as a integer. White-spaces are discarded until the first non-whitespace character is found.

**Return value:** The function returns the converted integral number as an int value, if successful and returns it as a long int. If no valid conversion could be performed then a zero value is returned. If the value is out of range then LONG\_MAX or LONG\_MIN is returned.

### **Programming Example of atol( ):**

```
#include<stdio.h>
#include<stdlib.h>
int main ()
{
    long int long_int;
    char buffer [200];
    printf ("Enter a long number: ");
    gets ( buffer );
    long_int = atol (buffer);
    printf ("The value entered is %ld.\n",long_int);
    return 0;
}
```

### **Output of the atol example program above:**

Enter a long number: 576000

The value entered is 576000.

### 3. Error Handling in C

Although C programming does not provide direct support for error handling (also called exception handling), there are ways to do error handling.

A lot of C function calls return a -1 or NULL in case of an error. In the operating systems like Unix or Linux, if a program successful ends the return value of the program is zero. If the program ends with an error usually a number larger than zero is returned (for example 1). With command ‘echo \$?’ on the command line you can display the return code of a program that has previously run.

**Global Variable *errno*** : The global variable *errno* is used by C functions and this integer is set if there is an error during the function call. To make use of *errno* you need to include *errno.h* and you need to call ‘*extern int errno;*’

Let us take a look at an example:

```
#include <stdio.h>
#include <errno.h>
extern int errno;
int main () {
    FILE * fp;
    fp = fopen ("filedoesnotexist.txt", "rb");
    if (fp == NULL) {
        fprintf(stderr, "Value of errno: %d\n", errno);
    } else {
        fclose (fp);
    }
    return 0;
}
```

**Note:** One should always use *stderr* file stream to output all of the errors

**Output :**

Value of errno is: 2

**strerror( ) and perror( ):** The C programming language has two functions that can be used to display a text message that is associated with errno. The functions are *strerror()* and *perror()*. The function *strerror()* returns a pointer to the textual message of the current errno value. The function *perror()* displays a string you pass to it, followed by a colon and the textual message of the current errno value.

### Programming Example :

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

extern int errno;

int main () {
    FILE * fp;
    fp = fopen ("fileABC.txt", "rb");
    if (fp == NULL) {
        fprintf(stderr, "Value of errno: %d\n", errno);
        fprintf(stderr, "Error opening the file: %s\n", strerror( errno
));
        perror("Error printed by perror");
    }
    else
        {fclose (fp);
    }
    return 0;
}
```

### Output:

```
value of errno: 2
Error opening the file: No such file or directory
Error printed by perror: No such file or directory
```

## 4. Syntax of Pre-processor directives

### 1. The syntax of #define directive is as follows :

```
#define identifier<substitute text>
```

OR

```
#define identifier (argument1, -----argument n) substitute text
```

### 2. The syntax of undef : #undef identifier

### 3. The syntax of #include is : #include <filename>

### 4. The syntax of #ifdef directives is given below:

```
#ifdef <identifier>
{
    statement1;
    statement2;
}
#else
{
    statement3;
    statement4;
}
#endif
```

### 5. The syntax of #ifndef directive is given below :

```
#ifndef <identifier>
{
    statement1;
    statement2;
}
#else
{
    statement3;
    statement4;
}
#endif
```

### 6. The syntax of #error directive is given below :

```
#if !defined (identifier)
#error <ERROR MESSAGE>
#endif
```

## **7. The syntax of #line directive is as follows :**

```
#line <constant> [ <identifier>]
```

Causes the compiler to imagine the line number of the next source line as given by <constant> and <identifier> gives the current input file . If <identifier> is absent , then the current file name remains unchanged.

**Example :** #line 15 pragma.c

**8. The #pragma directive :** These directives deal with formatting source listing and placing components in the object file generated by the compiler . It sets or resets certain warning and errors during compilation of C program.

### **Syntax :**

```
#pragma warn + xxx  
#pragma warn -xxx
```

Where the first statement turns on the warning message and the second statement sets off the warning message.

## **5. Programming Examples :**

1. Write a program to define and create identifier for C statements and variables.

- a. Read X as 10
- b. Replace clrscr() with CLS
- c. Replace getch() with WAIT()
- d. Replace printf with DISPLAY

```
#include<stdio.h>  
#include<conio.h>  
#define X 5  
#define CLS clrscr()  
#define WAIT() getch()  
#define DISPLAY printf  
  
void main ()  
{  
    int i;  
    CLS;
```

```
for(i=1;i<=X;i++)
DISPLAY("%d\t",i);
WAIT();
}
```

**Output :**

```
1 2 3 4 5
```

**2. Write a program to define macros for logical operators .**

```
#include<stdio.h>
#include<conio.h>
#define and  &&
#define equal ==
#define larger  >
void main()
{
    int a,b,c;
    clrscr();
    printf("\n Enter Three Numbers :");
    scanf("%d%d%d",&a,&b,&c);
    if(a larger b and a larger c)
        printf("%d is larger number", a);
    else
        if(b larger a and b larger c)
            printf("%d are larger number",b);
        else
            if(c larger a and c larger b)
                printf("%d are larger number.",c);
            else
                if(a equal b and b equal c)
                    printf("\n Numbers are same \n");
                getch();
}
```

**3. Write a program to undefine a macro**

```
#include<stdio.h>
#include<conio.h>
#define wait getche()
```

```
void main( )
{
    int k;
    #undef wait getche();
    clrscr();
    for(k=1;k<=10;k++)
        printf("%d\t",k);
    wait;
}
```

**Output :**

Error

- 4. Write a program to use conditional compilation statement as to whether the identifier is defined or not.**

```
#include<stdio.h>
#include<conio.h>
#define LINE 1
void main()
{
    clrscr();
    #ifdef LINE;
    printf("This is line number one.");
    #else
    printf("This is line number two");
    #endif
    getch();
}
```

**OUTPUT:**

This is line number one.

- 5. Write a program to check conditional compilation directives #ifndef. If it is observed display one message otherwise display another message.**

```
#include<stdio.h>
#include<conio.h>
#define TGS 1975
void main()
```

```
{  
    clrscr();  
    #ifndef TGS;  
    printf("\n MACRO TGS is not defined.");  
    #else  
    printf("\n MACRO TGS is defined ");  
    #endif  
    getch();  
}
```

#### OUTPUT:

MACRO TGS is defined

**6. Write a program to display user defined error message using #error directive.**

```
#include<stdio.h>  
#include<conio.h>  
#define B 1  
void main( )  
{  
    clrscr( );  
    #if!defined(A)  
    #error MACRO A IS NOT DEFINED  
    #else  
    printf("Macro defined.");  
    #endif  
}
```

## Additional Programming Examples

### 1. C Program to Find Sum of Digits of a Number using Recursion:

```
#include <stdio.h> /* Header File */
#include<conio.h>
int sum (int a); /* Prototype declaration */
int main() /* Begin of the Program */
{
    int num, result; /* Declaration of variables*/
    clrscr();
    printf("Enter the number: ");/* Reading num */
    scanf("%d", &num);
    result = sum(num);
    printf("Sum of digits in %d is %d\n", num, result);
    getch();
    return 0;
} /* End of the Program */

int sum (int num) /* Function Definition */
{
    if (num != 0) /* To check whether the number is zero or not */
    {
        return (num % 10 + sum (num / 10)); /* Recursive Call */
    }
    else
    {
        return 0;
    }
}
```

#### Output:

```
Enter the number: 1234
Sum of digits in 1234 is 10
```

### 2. C Program to find Reverse of a Number using Recursion

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

```

int rev(int, int); /* Function Prototype Declaration */

int main() /* Begin of the program */
{
    int num, result;
    int length = 0, temp;

    clrscr();

    printf("Enter an integer number to reverse: ");
    scanf("%d", &num);
    temp = num;

    while (temp != 0)/*Logic to find the length */
    {
        length++;
        temp = temp / 10;
    }
    result = rev(num, length); /* Function call */
    printf("The reverse of %d is %d.\n", num, result);
    getch();
    return 0;
}/* End of the program */

int rev(int num, int len) /* Function definition */
{
    if (len == 1)
    {
        return num;
    }
    else /* Recursive function call to reverse a number */
    {
        return(((num % 10)*pow(10, len - 1))+ rev(num/10,--len));
    }
}

```

**Output:**

Enter an integer number to reverse: 1234  
 The reverse of 1234 is 4321

### 3. C Program to Check whether a given Number is Perfect Number

**Note :** Perfect number is a number which is equal to sum of its divisor. For eg, divisors of 6 are 1,2 and 3. The sum of these divisors  $1+2+3 = 6$ . So 6 is called as perfect number.

```
#include <stdio.h> /* Header Files*/
#include<conio.h>

int main()/* Begin of the program */
{
    int number, rem, sum = 0, i; /* Variable Declarations*/
    clrscr();
    printf("Enter a Number\n");
    scanf("%d", &number);
    for (i = 1; i <= (number - 1); i++)/* Logic*/
    {
        rem = number % i;
        if (rem == 0)
        {
            sum = sum + i;
        }
    }
    if (sum == number)
        printf("Entered Number is perfect number");
    else
        printf("Entered Number is not a perfect number");
    return 0;
    getch();
}/* End of the Program */
```

#### Output:

```
Enter Number
6
Entered Number is perfect number
```

### 4. C Program to Compute First N Fibonacci Numbers using Command Line Arguments

```
#include <stdio.h> /* Header Files */
#include<conio.h>

int first = 0; /* Global variable Declaration */
```

```

int second = 1;
int third;
void rec_fibonacci(int); /* Function Prototype */

/*Command line Arguments*/
void main(int argc, char *argv[]){
    int number = atoi(argv[1]);
    clrscr();
    printf("\t%d\t%d", first, second);
    rec_fibonacci(number); /*Function Call*/
    printf("\n");
    getch();
}/* End of the program */

/* Recursive Function Definition */
void rec_fibonacci(int num)
{
    if (num == 2)
    {
        return;
    }
    third = first + second;
    printf("\t%d", third);
    first = second;
    second = third;
    num--;
    rec_fibonacci(num);
}

```

### Output:

```
C:\Tuboc\bin>fibonacci 5 //If the name of the program is
fibonacci.c
0 1 1 2 3
```

## 5. C Program to Swap the Contents of two Numbers using Bitwise XOR Operation

```

#include <stdio.h> /* Header Files */
#include<conio.h>
void main()/* Begin of the program */
{
    int i, k; /*Variable Declaration */
    clrscr();

```

```

printf("Enter two integers \n");
scanf("%d %d", &i, &k);
printf("\n Before swapping i= %d and k = %d", i, k);
i = i ^ k; /* Logic to swap using Bitwise Operators */
k = i ^ k;
i = i ^ k;
printf("\n After swapping i= %d and k = %d", i, k);
getch();
/* End of the Program */

```

**Output:**

```

Enter two integers
2   3
Before swapping i = 2 and k = 3
After swapping i = 3 and k = 2

```

## 6. C Program to Illustrate the Concept of Unions

```

#include <stdio.h> /* Header Files */
#include<conio.h>
void main() /* Begin of the Program */
{
    union number
    {
        int n1;
        float n2;
    };
    union number x;
    clrscr();
    printf("Enter the value of n1: ");
    scanf("%d", &x.n1);
    printf("Value of n1 = %d", x.n1);
    printf("\n Enter the value of n2: ");
    scanf("%f", &x.n2);
    printf("Value of n2 = %f\n", x.n2);
    getch();
}/* End of the Program */

```

**Output:**

```

Enter the value of n1: 10
Value of n1 = 10
Enter the value of n2: 2.5
Value of n2 = 2.500000

```

## **7. C Program to Print a Semicolon without using a Semicolon anywhere in the Code**

```
#include <stdio.h> /* Header Files */
#include<conio.h>
int main(void)/* Begin of the Program */
{
    clrscr()
    //59 is the ascii value of semicolon
    if (printf("%c ", 59))
    {
    }
    getch();
    return 0;
}
```

} /End of the Program\*/

## **8. C Program to Print any Print Statement without using Semicolon**

```
#include <stdio.h> /* Header Files */
#include<conio.h>
void main() /* Begin of the Program */
{
    clrscr();
    if (printf("Hello TGS"))
    {
    }
    getch();
    return 0;
}/* End of the Program */
```

## **9. C Program to Display its own source code as its Output**

```
#include <stdio.h> /* Header Files */
#include <conio.h>

int main() /*Begin of the Program */
{
    FILE *fp; /*Declaration of variables */
    char ch;
```

```

clrscr();

fp = fopen("source.c", "r"); /* Opening the source file */
do
{
    ch = getc(fp);
    putchar(ch);
}
while (ch != EOF);
fclose(fp); /* Closing the File */
getch();
return 0;
} /* End of the Program */

```

### 10. C Program to determine the cube of number using Pass by Reference

```

#include <stdio.h> /* Header Files */
#include<conio.h>

void cube( int *x); /* Function Prototype */

int main( ) /* Begin of the Program */
{
    int num = 10; /*Declaration of variables */

    cube(&num); /* Calling by reference */
    printf("\n The cube of the given number is %d", num);
    return 0;
} /* End of the Program */

void cube(int *x) /* Function Definition*/
{
    *x = (*x) * (*x) * (*x);
}

```

#### Output :

The cube of the given number is 1000

## 11. C Program to Calculate the Simple Interest

```
/* C Program to calculate the Simple Interest */
#include <stdio.h> /*Header File */
#include<conio.h>

void main()/* Begin of the Program */
{
    /* Variable Declaration */
    float principal_amt, rate, simple_interest;
    int time;

    clrscr();
    printf("Enter the values of principal_amt, rate and time \n");
    scanf("%f %f %d", &principal_amt, &rate, &time);
    simple_interest = (principal_amt * rate * time) / 100.0;
    printf("Amount = Rs. %5.3f\n", principal_amt);
    printf("Rate = Rs. %5.3f\n", rate);
    printf("Time = %d years\n", time);
    printf("Simple interest = %5.3f\n", simple_interest);
    getch();
} /* End of the Program */
```

## 12. C Program to Find the GCD and LCM of Two Integers

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

void main()
{
    int num1,num2,gcd,lcm,remainder,numerator,denominator;
    clrscr();

    printf("Enter two numbers\n");
    scanf("%d %d", &num1, &num2);
    if(num1 > num2)
    {
        numerator = num1;
        denominator = num2;
    }
    else
    {
        numerator = num2;
        denominator = num1;
```

```

    }
    remainder = num1 % num2;
    while (remainder != 0)
    {
        numerator = denominator;
        denominator = remainder;
        remainder = numerator % denominator;
    }
    gcd = denominator;
    lcm = num1 * num2 / gcd;
    printf("GCD of %d and %d = %d\n", num1, num2, gcd);
    printf("LCM of %d and %d = %d\n", num1, num2, lcm);
    getch();
}

```

### 13. C Program to Find LCM of a Number using Recursion

```

#include <stdio.h>
#include<conio.h>
int lcm(int a, int b);

int main()
{ int a, b, result;
  int prime[100];
  clrscr();
  printf("Enter two numbers: ");
  scanf("%d%d", &a, &b);
  result = lcm(a, b);
  printf("The LCM of %d and %d is %d\n", a, b, result);
  getch();
  return 0;
}

int lcm(int a, int b) /* Function Definition */
{
    static int common = 1; /* variable declaration */

    if (common % a == 0 && common % b == 0)
    {
        return common;
    }
    common++;
    lcm(a, b); /* Calling Recursively */
}

```

```
        return common;
    }
```

#### 14. C Program to Find GCD of given Numbers using Recursion

```
/* Program to find the GCD of two numbers using Recursion */
#include <stdio.h>
#include<conio.h>
int gcd(int, int);/* Function Prototype */

int main() /* Begin of the Program */
{
    int a, b, result; /* Variables Declaration */

    printf("Enter the two numbers to find their GCD: ");
    scanf("%d%d", &a, &b);
    result = gcd(a, b); /* Function Call*/
    printf("The GCD of %d and %d is %d.\n", a,b,result);
    getch();
}

int gcd(int a, int b)/* Function Definition*/
{
    while (a != b)
    {
        if (a > b) /* Calling Recursively */
        {
            return gcd(a - b, b);
        }
        else
        {
            return gcd(a, b - a);
        }
    }
    return a;
}
```

#### 15. C Program to Find the Areas of Different Geometrical Figures

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

```

int fig;
float side, base, length, breadth, height, area, radius;
clrscr();
printf("-----\n");
printf(" 1 --> Circle\n");
printf(" 2 --> Rectangle\n");
printf(" 3 --> Triangle\n");
printf(" 4 --> Square\n");
printf("-----\n");
printf("Enter the Figure No\n");
scanf("%d", &fig);
switch(fig)
{
    case 1:
        printf("Enter the radius\n");
        scanf("%f", &radius);
        area = 3.142 * radius * radius;
        printf("Area of a circle = %f\n", area);
        break;
    case 2:
        printf("Enter the breadth and length\n");
        scanf("%f %f", &breadth, &length);
        area = breadth * length;
        printf("Area of a Reactangle = %f\n", area);
        break;
    case 3:
        printf("Enter the base and height\n");
        scanf("%f %f", &base, &height);
        area = 0.5 * base * height;
        printf("Area of a Triangle = %f\n", area);
        break;
    case 4:
        printf("Enter the side\n");
        scanf("%f", &side);
        area = side * side;
        printf("Area of a Square=%f\n", area);
        break;
    default:
        printf("Error in figure no\n");
        break;
}

```

```
        getch();
    }
```

## 16. C program to Calculate the Value of nPr

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

void main(void)
{
    int n,r,npr;
    clrscr();
    printf("Enter value for n and r\n");
    scanf("%d%d", &n, &r);
    npr = fact(n) / fact(n - r);
    printf("\n Permutation values is = %d", npr);
    getch();
}

int fact(int x)
{
    if (x <= 1)
        return 1;
    return x * fact(x - 1);
}
```

## 17. C Program to Compute the Sum of two One-Dimensional Arrays using Malloc

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

void main()
{
    int i, n;
    int *a1, *a2, *s;
    clrscr();

    printf("How many Elements in each array...\\n");
    scanf("%d", &n);
    a1 = (int *)malloc(n * sizeof(int));
    a2 = (int *)malloc(n * sizeof(int));
```

```

s = (int *)malloc(n * sizeof(int));

printf("Enter Elements of First array \n");
for (i = 0; i < n; i++)
{
    scanf("%d", a1 + i);
}

printf("Enter Elements of Second array \n");
for (i = 0; i < n; i++)
{
    scanf("%d", a2 + i);
}

/* To find the sum of two respective arrays */
for (i = 0; i < n; i++)
{
    *(s + i) = *(a1 + i) + *(a2 + i);
}

printf("Resultant array is\n");
for (i = 0; i < n; i++)
{
    printf("%d\n", *(s + i));
}
getch();
}

```

## 18. C Program to Print the Number of Odd & Even Numbers in an Array

```

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

void main()
{
    int array[100], i, num;
    clrscr();

    printf("Enter the size of an array \n");
    scanf("%d", &num);
    printf("Enter the elements of the array \n");

```

```

for (i = 0; i < num; i++)
{
    scanf("%d", &array[i]);
}
printf("Even numbers in the array are : ");
for (i = 0; i < num; i++)
{
    if (array[i] % 2 == 0)
    {
        printf("%d \t", array[i]);
    }
}
printf("\n Odd numbers in the array are:");
for (i = 0; i < num; i++)
{
    if (array[i] % 2 != 0)
    {
        printf("%d \t", array[i]);
    }
}
getch();
}

```

#### **19. C Program to Perform Matrix Multiplication using Recursion**

```

#include <stdio.h>
#include<conio.h>
void multiply(int, int, int [][]10, int, int, int [][]10, int [][]10);
    void display(int, int, int[]][10]);

int main()
{
    int a[10][10], b[10][10], c[10][10] = {0};
    int m1, n1, m2, n2, i, j, k;
    clrscr();

    printf("Enter rows and columns for Matrix A respectively: ");
    scanf("%d%d", &m1, &n1);
    printf("Enter rows and columns for Matrix B respectively: ");
    scanf("%d%d", &m2, &n2);
    if (n1 != m2)
    {
        printf("Matrix multiplication not possible.\n");
    }
}

```

```

    }
else
{
    printf("Enter elements in Matrix A:\n");
    for (i = 0; i < m1; i++)
        for (j = 0; j < n1; j++)
        {
            scanf("%d", &a[i][j]);
        }
    printf("\nEnter elements in Matrix B:\n");
    for (i = 0; i < m2; i++)
        for (j = 0; j < n2; j++)
        {
            scanf("%d", &b[i][j]);
        }
    multiply(m1, n1, a, m2, n2, b, c);
}
printf("On matrix multiplication of A and B the result is:\n");
display(m1, n2, c);
}

```

```

void multiply (int m1, int n1, int a[10][10], int m2, int n2, int b[10][10],
int c[10][10])
{
    static int i = 0, j = 0, k = 0;

    if (i >= m1)
    {
        return;
    }
    else if (i < m1)
    {
        if (j < n2)
        {
            if (k < n1)
            {
                c[i][j] += a[i][k] * b[k][j];
                k++;
                multiply(m1, n1, a, m2, n2, b, c);
            }
            k = 0;
            j++;
            multiply(m1, n1, a, m2, n2, b, c);
        }
    }
}

```

```

        j = 0;
        i++;
        multiply(m1, n1, a, m2, n2, b, c);
    }
}

void display(int m1, int n2, int c[10][10])
{
    int i, j;

    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < n2; j++)
        {
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
}

```

**Output :**

Enter rows and columns for Matrix A respectively: 2

2

Enter rows and columns for Matrix B respectively: 2

2

Enter elements in Matrix A:

12 56

45 78

Enter elements in Matrix B:

2 6

5 8

On matrix multiplication of A and B the result is:

304 520

480 894

**20. C Program to Determine if a given Matrix is a Sparse Matrix**

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

```
void main ()
```

```

{
    static int array[10][10];
    clrscr();
    int i, j, m, n;
    int counter = 0;

    printf("Enter the order of the matix \n");
    scanf("%d %d", &m, &n);
    printf("Enter the co-efficients of the matix \n");
    for (i = 0; i < m; ++i)
    {
        for (j = 0; j < n; ++j)
        {
            scanf("%d", &array[i][j]);
            if (array[i][j] == 0)
            {
                ++counter;
            }
        }
    }
    if (counter > ((m * n) / 2))
    {
        printf("The given matrix is sparse matrix \n");
    }
    else
    {
        printf("The given matrix is not a sparse matrix \n");
        printf("There are %d number of zeros", counter);
        getch();
    }
}

```

**Run1:**

Enter the order of the matix

3 3

Enter the co-efficients of the matix

10 20 30

5 10 15

3 6 9

The given matrix is not a sparse matrix

There are 0 number of zeros

**Run2:**

Enter the order of the matix

3 3

Enter the co-efficients of the matix

```
5 0 0  
0 0 5  
0 5 0
```

```
The given matrix is sparse matrix  
There are 6 number of zeros
```

## 21. C Program to Find the Frequency of Odd & Even Numbers in the given Matrix

```
#include <stdio.h>  
#include<conio.h>  
  
void main()  
{  
    static int array[10][10];  
    int i, j, m, n, even = 0, odd = 0;  
    clrscr();  
  
    printf("Enter the order of the matrix \n");  
    scanf("%d %d", &m, &n);  
    printf("Enter the coefficients of matrix \n");  
    for (i = 0; i < m; ++i)  
    {  
        for (j = 0; j < n; ++j)  
        {  
            scanf("%d", &array[i][j]);  
            if ((array[i][j] % 2) == 0)  
            {  
                ++even;  
            }  
            else  
                ++odd;  
        }  
    }  
    printf("The given matrix is \n");  
    for (i = 0; i < m; ++i)  
    {  
        for (j = 0; j < n; ++j)  
        {  
            printf(" %d", array[i][j]);  
        }  
        printf("\n");
```

```

    }
printf("\n The frequency of occurrence of odd number = %d \n", odd);
printf("The frequency of occurrence of even number = %d\n", even);
getch();
}
Enter the order of the matrix
3 3
Enter the coefficients of matrix
3   6   9
2   5   8
3   9   4
The given matrix is
3   6   9
2   5   8
3   9   4

```

The frequency of occurrence of odd number = 5  
 The frequency of occurrence of even number = 4

## 22. C Program to Accept a Matrix of Order MxN & Interchange the Diagonals

```

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

void main ()
{
    static int array[10][10];
    int i, j, m, n, a;
    clrscr();

    printf("Enter the order of the matrix \n");
    scanf("%d %d", &m, &n);
    if (m == n)
    {
        printf("Enter the co-efficients of the matrix\n");
        for (i = 0; i < m; ++i)
        {
            for (j = 0; j < n; ++j)
            {
                scanf("%dx%d", &array[i][j]);
            }
        }
    }
}

```

```

printf("The given matrix is \n");
for (i = 0; i < m; ++i)
{
    for (j = 0; j < n; ++j)
    {
        printf(" %d", array[i][j]);
    }
    printf("\n");
}
for (i = 0; i < m; ++i)
{
    a = array[i][i];
    array[i][i] = array[i][m - i - 1];
    array[i][m - i - 1] = a;
}
printf("The matrix after changing the \n");
printf("main diagonal & secondary diagonal\n");
for (i = 0; i < m; ++i)
{
    for (j = 0; j < n; ++j)
    {
        printf(" %d", array[i][j]);
    }
    printf("\n");
}
else
printf("The given order is not square matrix\n");
getch();
}

```

Enter the order of the matix

2 2

Enter the co-efficients of the matrix

1 2

2 1

The given matrix is

1 2

2 1

The matrix after changing the  
main diagonal & secondary diagonal

2 1

1 2

### 23. C Program to check whether the Lower Triangular Matrix.

```
#include <stdio.h>

void main()
{
    int array[3][3], i, j, flag = 0 ;
    printf("\n\t Enter the value of Matrix : ");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            scanf("%d", &array[i][j]);
        }
    }
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            if (array[i] < array[j] && array[i][j] == 0)
            {
                flag = flag + 1;
            }
        }
    }
    if (flag == 3)
        printf("\n\n Matrix is a Lower triangular matrix");
    else
        printf("\n\n Matrix is not a lower triangular matrix");
}
```

#### Output:

```
Enter the value of Matrix :
1 2 0
1 0 0
0 0 0
Matrix is not a lower triangular matrix
```

### 24. C Program to Find the Sum of ASCII values of All Characters in a given String

```

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

void main()
{
    int sum = 0, i, len;
    char string1[100];
    clrscr();

    printf("Enter the string : ");
    scanf("%[^\\n]s", string1);
    len = strlen(string1);
    for (i = 0; i < len; i++)
    {
        sum = sum + string1[i];
    }
    printf("\nSum of ASCII values of all characters:%d ",sum);
}

```

**Output :**

Enter the string : Dr. Thyagaraju Gowda  
 Sum of ASCII values of all characters : 1830

**25. C Program to Create a Linked List & Display the Elements in the List**

```

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include<conio.h>

void main()
{
    struct node
    {
        int num;
        struct node *ptr;
    };
    typedef struct node NODE;

    NODE *head, *first, *temp = 0;
    int count = 0;
    int choice = 1;

```

```

first = 0;
clrscr();
while (choice)
{
    head = (NODE *)malloc(sizeof(NODE));
    printf("Enter the data item\n");
    scanf("%d", &head-> num);
    if (first != 0)
    {
        temp->ptr = head;
        temp = head;
    }
    else
    {
        first = temp = head;
    }
    fflush(stdin);
    printf("Do you want to continue(Type 0 or 1)?\n");
    scanf("%d", &choice);

}
temp->ptr = 0;
/* reset temp to the beginning */
temp = first;
printf("\n status of the linked list is\n");
while (temp != 0)
{
    printf("%d=>", temp->num);
    count++;
    temp = temp -> ptr;
}
printf("NULL\n");
printf("No. of nodes in the list = %d\n", count);
}

```

**Output :**

**Run 1:**

Enter the data item

25

Do you want to continue(Type 0 or 1)?

0

status of the linked list is

**25=>NULL**

No. of nodes in the list = 1

**Output:**

**Run 2:**

Enter the data item

55

Do you want to continue(Type 0 or 1)?

1

Enter the data item

77

Do you want to continue(Type 0 or 1)?

1

Enter the data item

88

Do you want to continue(Type 0 or 1)?

0

status of the linked list is

**55=>77=>88=>NULL**

No. of nodes in the list = 3

## 26. C Program to Implement a Stack

```
#include <stdio.h>
#define MAXSIZE 5
#include<conio.h>
struct stack
{
    int stk[MAXSIZE];
    int top;
};
typedef struct stack STACK;
STACK s;

void push(void);
int pop(void);
void display(void);

void main ()
{
    int choice;
    int option = 1;
```

```

s.top = -1;
clrscr();

printf("STACK OPERATIONS\n");
while (option)
{
    printf("-----\n");
    printf(" 1 --> PUSH      \n");
    printf(" 2 --> POP       \n");
    printf(" 3 --> DISPLAY   \n");
    printf(" 4 --> EXIT      \n");
    printf("-----\n");

    printf("Enter your choice\n");
    scanf ("%d", &choice);
    switch (choice)
    {
        case 1:
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            return;
    } // End of switch
    fflush (stdin);
    printf("Do you want to continue(Type 0 or 1)?\n");
    scanf ("%d", &option);
} // End of while
getch();
} // End of Main

```

```

/* Function to add an element to the stack */
void push ()
{
    int num;
    if (s.top == (MAXSIZE - 1))
    {
        printf("Stack is Full\n");
    }
}

```

```
        return;
    }
else
{
    printf ("Enter the element to be pushed\n");
    scanf ("%d", &num);
    s.top = s.top + 1;
    s.stk[s.top] = num;
}
return;
}
```

```
/* Function to delete an element from the stack */
int pop ()
{
    int num;
    if (s.top == - 1)
    {
        printf ("Stack is Empty\n");
        return (s.top);
    }
    else
    {
        num = s.stk[s.top];
        printf ("poped element is = %dn", s.stk[s.top]);
        s.top = s.top - 1;
    }
    return(num);
}
```

```
/* Function to display the status of the stack */
void display ()
{
    int i;
    if (s.top == -1)
    {
        printf ("Stack is empty\n");
        return;
    }
    else
    {
        printf ("\n The status of the stack is \n");
    }
}
```

```
for (i = s.top; i >= 0; i--)  
{  
    printf ("%d\n", s.stk[i]);  
}  
printf ("\n");  
}
```

### Output :

#### Run1:

##### STACK OPERATION

---

```
1 --> PUSH  
2 --> POP  
3 --> DISPLAY  
4 --> EXIT
```

---

Enter your choice

1

Enter the element to be pushed

22

Do you want to continue(Type 0 or 1)?

0

#### Run2:

##### STACK OPERATION

---

```
1 --> PUSH  
2 --> POP  
3 --> DISPLAY  
4 --> EXIT
```

---

Enter your choice

1

Enter the element to be pushed

33

Do you want to continue(Type 0 or 1)?

1

---

```
1 --> PUSH  
2 --> POP  
3 --> DISPLAY  
4 --> EXIT
```

---

Enter your choice  
2  
poped element is = 33  
Do you want to continue(Type 0 or 1)?  
1

---

1 --> PUSH  
2 --> POP  
3 --> DISPLAY  
4 --> EXIT

---

Enter your choice  
3  
Stack is empty  
Do you want to continue(Type 0 or 1)?  
1

---

1 --> PUSH  
2 --> POP  
3 --> DISPLAY  
4 --> EXIT

---

Enter your choice  
1  
Enter the element to be pushed  
22  
Do you want to continue(Type 0 or 1)?  
1

---

1 --> PUSH  
2 --> POP  
3 --> DISPLAY  
4 --> EXIT

---

Enter your choice  
1  
Enter the element to be pushed  
33  
Do you want to continue (Type 0 or 1)?  
1

---

1 --> PUSH  
2 --> POP  
3 --> DISPLAY

4 --> EXIT

---

Enter your choice

3

The status of the stack is

33

22

Do you want to continue(Type 0 or 1)?

1

---

1 --> PUSH

2 --> POP

3 --> DISPLAY

4 --> EXIT

---

Enter your choice

4

## 27. C Program to Implement a Queue using an Array

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

#define MAX 50
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
    int choice;
    while (1)
    {
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
```

```

insert();
break;
case 2:
delete();
break;
case 3:
display();
break;
case 4:
exit(1);
default:
printf("Wrong choice \n");
} /*End of switch*/
} /*End of while*/
} /*End of main()*/



insert()
{
int add_item;
if (rear == MAX - 1)
printf("Queue Overflow \n");
else
{
if (front == - 1)
/*If queue is initially empty */
front = 0;
printf("Insert the element in queue : ");
scanf("%d", &add_item);
rear = rear + 1;
queue_array[rear] = add_item;
}
} /*End of insert()*/



delete()
{
if (front == - 1 || front > rear)
{
printf("Queue Underflow \n");
return ;
}
else
{
printf("Element deleted from queue is : %d\n",
queue_array[front]);
}
}

```

```
        front = front + 1;
    }
} /*End of delete() */
display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
} /*End of display() */
```

### Output:

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 1  
Insert the element in queue : 10

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 1  
Insert the element in queue : 15

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 1  
Inset the element in queue : 20

- 1.Insert element to queue

- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 1  
Insert the element in queue : 30

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 2  
Element deleted from queue is : 10

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 3

Queue is :  
15 20 30

- 1.Insert element to queue
- 2.Delete element from queue
- 3.Display all elements of queue
- 4.Quit

Enter your choice : 4

# **Model Questions for Placement**

**(Source : [13] to [30] specified in References Section)**

## **1.Descriptive Questions**

### **Model Questions Set 1:**

1. What does static variable mean?
2. What is a pointer?
3. What is a structure?
4. What are the differences between structures and arrays?
5. In header files whether functions are declared or defined?
6. What are the differences between malloc() and calloc()?
7. What are macros? what are its advantages and disadvantages?
8. Difference between pass by reference and pass by value?
9. What is static identifier?
10. Where are the auto variables stored?
11. Where does global, static, local, register variables, free memory and C Program instructions get stored?
12. Difference between arrays and linked list?
13. What are enumerations?
14. Describe about storage allocation and scope of global, extern, static, local and register variables?
15. What are register variables? What are the advantage of using register variables?
16. What is the use of typedef?
17. Can we specify variable field width in a scanf() format string? If possible how?
18. Out of fgets() and gets() which function is safe to use and why?
19. Difference between strdup and strcpy?
20. What is recursion?
21. Differentiate between a for loop and a while loop? What are its uses?
22. What are the different storage classes in C?
23. Write down the equivalent pointer \_expression for referring the same element a[i][j][k][l].
24. What is difference between Structure and Unions?
25. What are the advantages of using Unions?
26. What are the advantages of using pointers in a program?
27. What is the difference between Strings and Arrays?
28. In a header file whether functions are declared or defined?
29. What is a far pointer? where we use it?
30. How will you declare an array of three function pointers where each function receives two ints and returns a float?

31. what is a NULL Pointer? Whether it is same as an uninitialized pointer?
32. What is a NULL Macro? What is the difference between a NULL Pointer and a NULL Macro?
33. What does the error 'Null Pointer Assignment' mean and what causes this error?
34. What is near, far and huge pointers? How many bytes are occupied by them?
35. How would you obtain segment and offset addresses from a far address of a memory location?
36. Are the expressions arr and &arr same for an array of integers?
37. Does mentioning the array name gives the base address in all the contexts?
38. Explain one method to process an entire string as one unit?
39. What is the similarity between a Structure, Union and enumeration?
40. Can a Structure contain a Pointer to itself?
41. How can we check whether the contents of two structure variables are same or not?
42. How are Structure passing and returning implemented by the compiler?
43. How can we read/write Structures from/to data files?
44. What is the difference between an enumeration and a set of pre-processor # defines?
45. what do the 'c' and 'v' in argc and argv stand for?
46. Are the variables argc and argv are local to main?
47. What is the maximum combined length of command line arguments including the space between adjacent arguments?
48. If we want that any wildcard characters in the command line arguments should be appropriately expanded, are we required to make any special provision? If yes, which?
49. Does there exist any way to make the command line arguments available to other functions without passing them as arguments to the function?
50. What are bit fields? What is the use of bit fields in a Structure declaration?
51. To which numbering system can the binary number 1101100100111100 be easily converted to?
52. Which bit wise operator is suitable for checking whether a particular bit is on or off?
53. Which bit wise operator is suitable for turning off a particular bit in a number?
54. Which bit wise operator is suitable for putting on a particular bit in a number?

55. Which bit wise operator is suitable for checking whether a particular bit is on or off?
56. which one is equivalent to multiplying by 2:Left shifting a number by 1 or Left shifting an unsigned int or char by 1?
57. Write a program to compare two strings without using the strcmp() function.
58. Write a program to concatenate two strings.
59. Write a program to interchange 2 variables without using the third one.
60. Write programs for String Reversal & Palindrome check
61. Write a program to find the Factorial of a number
62. Write a program to generate the Fibinocci Series
63. Write a program which employs Recursion
64. Write a program which uses Command Line Arguments
65. Write a program which uses functions like strcmp(), strcpy() etc
66. What are the advantages of using typedef in a program?
67. How would you dynamically allocate a one-dimensional and two-dimensional array of integers?
68. How can you increase the size of a dynamically allocated array?
69. How can you increase the size of a statically allocated array?
70. When reallocating memory if any other pointers point into the same piece of memory do you have to readjust these other pointers or do they get readjusted automatically?
71. Which function should be used to free the memory allocated by calloc()?
72. How much maximum can you allocate in a single call to malloc()?
73. Can you dynamically allocate arrays in expanded memory?
74. What is object file? How can you access object file?
75. Which header file should you include if you are to develop a function which can accept variable number of arguments?
76. Can you write a function similar to printf()?
77. How can a called function determine the number of arguments that have been passed to it?
78. Can there be at least some solution to determine the number of arguments passed to a variable argument list function?
79. How do you declare the following:
- \* An array of three pointers to chars
  - \* An array of three char pointers
  - \* A pointer to array of three chars
  - \* A pointer to function which receives an int pointer and returns a float pointer
  - \* A pointer to a function which receives nothing and returns nothing
80. What do the functions atoi(), itoa() and gcvt() do?

81. Does there exist any other function which can be used to convert an integer or a float to a string?
82. How would you use qsort() function to sort an array of structures?
83. How would you use qsort() function to sort the name stored in an array of pointers to string?
84. How would you use bsearch() function to search a name stored in array of pointers to string?
85. How would you use the functions sin(), pow(), sqrt()?
86. How would you use the functions memcpy(), memset(), memmove()?
87. How would you use the functions fseek(), freed(), fwrite() and ftell()?
88. How would you obtain the current time and difference between two times?
89. How would you use the functions randomize() and random()?
90. How would you implement a substr() function that extracts a sub string from a given string?
91. What is the difference between the functions rand(), random(), srand() and randomize()?
92. What is the difference between the functions memmove() and memcpy()?
93. How do you print a string on the printer?
94. Can you use the function fprintf() to display the output on the screen?
95. What is C language?
96. What is hashing?
97. Can static variables be declared in a header files?
98. Can a variable be both constant and volatile.
99. Can include files be nested.
100. What is null pointer?
101. What is null statement?
102. What is infinite Loop?

### **Model Questions Set 2 :**

#### **Basics Of C language**

1. What is a local block?
2. Should variables be stored in local blocks?
3. When is a **switch** statement better than multiple **if** statements?
4. Is a default case necessary in a **switch** statement?
5. Can the last case of a **switch** statement skip including the break?
6. Other than in a **for** statement, when is the comma operator used?
7. How can you tell whether a loop ended prematurely?
8. What is the difference between **goto** and long **jmp( )** and **setjmp()**?
9. What is an **lvalue**?

10. Can an array be an ***lvalue***?
11. What is an ***rvalue***?
12. Is left-to-right or right-to-left order guaranteed for operator precedence?
13. What is the difference between ***++var*** and ***var++***?
14. What does the modulus operator do?

## Variables and Data Storage

1. Where in memory are my variables stored?
2. Do variables need to be initialized?
3. What is page thrashing?
4. What is a ***const*** pointer?
5. When should the ***register*** modifier be used? Does it really help?
6. When should the ***volatile*** modifier be used?
7. Can a variable be both ***const*** and ***volatile***?
8. When should the ***const*** modifier be used?
9. How reliable are floating-point comparisons?
10. How can you determine the maximum value that a numeric variable can hold?
11. Are there any problems with performing mathematical operations on different variable types?
12. What is operator promotion?
13. When should a type cast be used?
14. When should a type cast not be used?
15. Is it acceptable to declare/define a variable in a C header?
16. What is the difference between declaring a variable and defining a variable?
17. Can static variables be declared in a header file?
18. What is the benefit of using ***const*** for declaring constants?

## Bits and Bytes

1. What is the most efficient way to store flag values?
2. What is meant by "bit masking"?
3. Are bit fields portable?
4. Is it better to bitshift a value than to multiply by 2?
5. What is meant by high-order and low-order bytes?
6. How are 16- and 32-bit numbers stored?

## Functions

1. When should I declare a function?

2. Why should I prototype a function?
3. How many parameters should a function have?
4. What is a `static` function?
5. Should a function contain a `return` statement if it does not return a value?
6. How can you pass an array to a function by value?
7. Is it possible to execute code even after the program exits the `main()` function?
8. What does a function declared as PASCAL do differently?
9. Is using `exit()` the same as using `return`?

## Preprocessors

1. What is a macro, and how do you use it?
2. What will the preprocessor do for a program?
3. How can you avoid including a header more than once?
4. Can a file other than a `.h` file be included with `#include`?
5. What is the benefit of using `#define` to declare a constant?
6. What is the benefit of using `enum` to declare a constant?
7. What is the benefit of using an `enum` rather than a `#define` constant?
8. How are portions of a program disabled in demo versions?
9. Is it better to use a macro or a function?
10. What is the best way to comment out a section of code that contains comments?
11. What is the difference between `#include` and `#include "file"`?
12. Can you define which header file to include at compile time?
13. Can include files be nested?
14. How many levels deep can include files be nested?
15. What is the concatenation operator?
16. How can type-insensitive macros be created?
17. What are the standard predefined macros?
18. How can a program be made to print the line number where an error occurs?
19. How can a program be made to print the name of a source file where an error occurs?
20. How can you tell whether a program was compiled using C versus C++?
21. What is a pragma?
22. What is `#line` used for?
23. What are the `__DATE__` and `__TIME__` preprocessor commands?
24. How can you be sure that a program follows the ANSI C standard?
25. How do you override a defined macro?
26. How can you check to see whether a symbol is defined?

## Strings

1. What is the difference between a string copy (`strcpy`) and a memory copy (`memcpy`)? When should each be used?
2. How can I remove the trailing spaces from a string?
3. How can I remove the leading spaces from a string?
4. How can I right-justify a string?
5. How can I pad a string to a known length?
6. How can I copy just a portion of a string?
7. How can I convert a number to a string?
8. How can I convert a string to a number?
9. How can you tell whether two strings are the same?
10. How do you print only part of a string?

## Arrays

1. Do array subscripts always start with zero?
2. Is it valid to address one element beyond the end of an array?
3. Can the `sizeof` operator be used to tell the size of an array passed to a function?
4. Is it better to use a pointer to navigate an array of values, or is it better to use a subscripted array name?
5. Can you assign a different address to an array tag?
6. What is the difference between `array_name` and `&array_name`?
7. Why can't constant values be used to define an array's initial size?
8. What is the difference between a string and an array?

## Pointers

1. What is indirection?
2. How many levels of pointers can you have?
3. What is a null pointer?
4. When is a null pointer used?
5. What is a `void` pointer
6. When is a `void` pointer used?
7. Can you subtract pointers from each other? Why would you?
8. Is `NULL` always defined as `0(zero)`?
9. Is `NULL` always equal to `0(zero)`?
10. What does it mean when a pointer is used in an `if` statement?
11. Can you add pointers together? Why would you?
12. How do you use a pointer to a function?
13. When would you use a pointer to a function?
14. Can the size of an array be declared at runtime?
15. Is it better to use `malloc()` or `calloc()`?
16. How do you declare an array that will hold more than 64KB of data?

17. What is the difference between **far** and **near** ?
18. When should a **far** pointer be used?
19. What is the stack?
20. What is the heap?
21. What happens if you free a pointer twice?
22. What is the difference between **NULL** and **NUL**?
23. What is a "null pointer assignment" error? What are bus errors, memory faults, and core dumps?
24. . How can you determine the size of an allocated portion of memory?
25. How does **free()** know how much memory to release?
26. Can math operations be performed on a **void** pointer?
27. How do you print an address?

## Data Files

1. If **errno** contains a nonzero number, is there an error?
2. What is a stream?
3. How do you redirect a standard stream?
4. How can you restore a redirected standard stream?
5. Can **stdout** be forced to print somewhere other than the screen?
6. What is the difference between text and binary modes?
7. How do you determine whether to use a stream function or a low-level function?
8. How do you list files in a directory?
9. How do you list a file's date and time?
10. How do you sort filenames in a directory?
11. How do you determine a file's attributes?
12. How do you view the PATH?
13. How can I open a file so that other programs can update it at the same time?
14. How can I make sure that my program is the only one accessing a file?
15. How can I prevent another program from modifying part of a file that I am modifying?
16. How can I avoid the Abort, Retry, Fail messages?
17. How can I read and write comma-delimited text?

## Standard Library Functions

1. Why should I use standard library functions instead of writing my own?
2. What header files do I need in order to define the standard library functions I use?
3. How can I write functions that take a variable number of arguments?

4. What is the difference between a free-standing and a hosted environment?
5. What standard functions are available to manipulate strings?
6. How do I determine whether a character is numeric, alphabetic, and so on?
7. What is a "locale"?
8. Is there a way to jump out of a function or functions?
9. What's a signal? What do I use signals for?
10. Why shouldn't I start variable names with underscores?
11. What math functions are available for integers? For floating point?
12. What are multibyte characters?
13. How can I manipulate strings of multibyte characters?

### **Model Questions Set 3**

- 1)How do you construct an increment statement or decrement statement in C?
- 2)What is the difference between Call by Value and Call by Reference?
- 3)Some coders debug their programs by placing comment symbols on some codes instead of deleting it. How does this aid in debugging?
- 4) What is a stack?
- 6) What is a sequential access file?
- 7) What is variable initialization and why is it important?
- 8 What is spaghetti programming?
- 9) Differentiate Source Codes from Object Codes
- 10) In C programming, how do you insert quote characters (' and ") into the output screen?
- 11) What is the use of a '\0' character?
- 12) What is the difference between the = symbol and == symbol?
- 13) What is the modulus operator?
- 14) What is a nested loop?
- 15) Which of the following operators is incorrect and why? (>=, <=, <>, ==)
- 16) Compare and contrast compilers from interpreters.
- 17) How do you declare a variable that will hold string values?
- 18) Can the curly brackets { } be used to enclose a single line of code?
- 19) What are header files and what are its uses in C programming?
- 20) What is syntax error?
- 21) What are variables and it what way is it different from constants?
- 22) How do you access the values within an array?
- 23) Can I use “int” data type to store the value 32768? Why?

- 24) Can two or more operators such as \n and \t be combined in a single line of program code?
- 25) Why is it that not all header files are declared in every C program?
- 26) When is the “void” keyword used in a function?
- 27) What are compound statements?
- 28) What is the significance of an algorithm to C programming?
- 29) What is the advantage of an array over individual variables?
- 30) Write a loop statement that will show the following output:

1  
12  
123  
1234  
12345

- 31) What is wrong in this statement? scanf(“%d”,whatnumber);
- 32) How do you generate random numbers in C?
- 33) What could possibly be the problem if a valid function name such as tolower() is being reported by the C compiler as undefined?
- 34) What are comments and how do you insert it in a C program?
- 35) What is debugging?
- 36) What does the && operator do in a program code?
- 37) In C programming, what command or code can be used to determine if a number of odd or even?
- 38) What does the format %10.2 mean when included in a printf statement?
- 39) What are logical errors and how does it differ from syntax errors?
- 40) What are the different types of control structures in programming?
- 41) What is || operator and how does it function in a program?
- 42) Can the “if” function be used in comparing strings?
- 43) What are preprocessor directives?
- 44) What will be the outcome of the following conditional statement if the value of variable s is 10?  
 $s >= 10 \&\& s < 25 \&\& s != 12$
- 45) Describe the order of precedence with regards to operators in C.
- 46) What is wrong with this statement? myName = “Robin”;
- 47) How do you determine the length of a string value that was stored in a variable
- 48) Why is C language considered a middle level language?
- 49) What are the different file extensions involved when programming in C?
- 50) What are reserved words ?
- 51) What are linked lists?
- 52)What is FIFO?
- 53) What are binary tress?
- 54)Not all reserved words are written in lower case. TRUE or FALSE?

- 55) What is the difference between the expression “`++a`” and “`++`”?
- 56) What would happen to X in this expression : `X+=55` ; (assuming the value of X is 10)
- 57) In C language the variables ROLL, roll and Roll are all the same .TRUE OR FALSE?
- 58)What is an endless loop?
- 59) What is a program flowchart and how does it help in writing a program?
- 60) What is wrong with this program statement? `void = 10;`
- 61) What are actual arguments?
- 62) What is a newline escape sequence?
- 63) What is output redirection ?
- 64) What are run time errors ?
- 65) What is the difference between function `abs()` and `fabs()`?
- 66) What are formal and actual parameters?
- 67) What are control structures?
- 68) Write a simple code fragment that will check if a number is neutral, positive or negative .
- 69) When is a switch statement preferable over an if statement?
- 70) What are global variables and how do you declare them?
- 71) What are enumerated types?
- 72) What does the function `toupper()` do?
- 73) Is it possible to have a function as a parameter in another function?
- 74) What are multidimensional arrays?
- 75)Which function in C can be used to append a string to another string?
- 76)What is the difference between functions `getch()` and `getche()`?
- 77) Do these two program segments perform the same output? 1)  
`scanf("%c",&letter);` 2) `letter = getchar();`
- 78) What are structure types in C?
- 79) What does the characters “r” and “w” mean when writing programs that will make use of files ?
- 80) What is the difference between text files and binary files ?
- 81) Is it possible to create your own header files ?
- 82) What is dynamic data structures?
- 83) What are the different data types in C?
- 84) What is dynamic data structure?
- 85) What are the different data types in C?
- 86) What is the general form of a C program?
- 87) What is the advantage of a random access file ?
- 88) In a switch statement, what will happen if a break statement is omitted?
- 89)Describe how arrays can be passed to a user defined function?
- 90) What are pointers?
- 91) Can you pass an entire structure to functions ?

- 92) What is gets() function?
- 93) The % symbol has a special use in a printf statement. How would you place this character as part of the output on the screen?
- 94) How do you search data in a data file using random access method?
- 95) Are comments included during the compilation stage and placed in the EXE file as well?
- 96) Is there a built in function in C that can be used for sorting data?
- 97) What are the advantages and disadvantages of a heap?
- 98) How do you convert strings to numbers in C?
- 99) Create a simple code fragment that will swap the values of two variables num1 and num2.
- 100) What is the use of a semicolon (;) at the end of every program statement ?

## 2. Multiple Choice Questions

(Source : <http://www.indiabix.com> )

**1. What is the output of the following problem ?**

```
#include<stdio.h>
main() {
    int i,j;
    j = 10;
    i = j++ - j++;
    printf("%d %d", i,j);
}
```

**ans: 0, 12**

**2. What is the output of the following problem ?**

```
main()
{
    int j;
    for(j=0;j<3;j++)
        foo();
}
foo()
{
    static int i = 10;
    i+=10;
    printf("%d\n",i);
}
```

**ans:**

20

30

40

**3. Point out the error in the following program.**

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

```
struct emp
```

```
{
```

```
    char name[20];
```

```
    int age;
```

```
};
```

```
int main()
```

```
{
```

```
    emp int xx;
```

```
int a;  
printf("%d\n", &a);  
return 0;  
}
```

- A. Error: in *printf*                   B. Error: in *emp int xx*;  
C. No error.                           D. None of these.

**4. Point out the error in the following program.**

```
#include<stdio.h>  
int main()  
{  
    int (*p)() = fun;  
    (*p)();  
    return 0;  
}  
int fun()  
{  
    printf("SDMIT.com\n");  
    return 0;  
}
```

- A. Error: in *int(\*p)() = fun;*  
B. Error: *fun()* prototype not defined  
C. No error  
D. None of these

**5.If the binary equivalent of 5.375 in normalized form is 0100 0000 1010 1100 0000 0000 0000, what will be the output of the program (on Intel machine)?**

```
#include<stdio.h>  
#include<math.h>  
int main()  
{  
    float a=5.375;  
    char *p;  
    int i;  
    p = (char*)&a;  
    for(i=0; i<=3; i++)  
        printf("%02x\n", (unsigned char)p[i]);  
    return 0;  
}
```

- A. 40 AC 00 00                   B. 04 CA 00 00  
C. 00 00 AC 40                   D. 00 00 CA 04

**6.Which of the following range is a valid *long double* (Turbo C in 16 bit DOS OS) ?**

- A.  $3.4E^{-4932}$  to  $1.1E^{+4932}$       B.  $3.4E^{-4932}$  to  $3.4E^{+4932}$   
C.  $1.1E^{-4932}$  to  $1.1E^{+4932}$       D.  $1.7E^{-4932}$  to  $1.7E^{+4932}$

**7.Which statement will you add in the following program to work it correctly?**

```
#include<stdio.h>
int main()
{
    printf("%f\n", log(36.0));
    return 0;
}
```

- A. #include<conio.h>      B. #include<math.h>  
C. #include<stdlib.h>      D. #include<dos.h>

**8.The binary equivalent of 5.375 is**

- A. 101.101110111      B. 101.011  
C. 101011      D. None of above

**9.A *float* occupies 4 bytes. If the hexadecimal equivalent of these 4 bytes are A, B, C and D, then when this *float* is stored in memory in which of the following order do these bytes gets stored?**

- A. ABCD  
B. DCBA  
C. 0xABCD  
D. Depends on big endian or little endian architecture

**10.Can you combine the following two statements into one?**

```
char *p;
p = (char*) malloc(100);
```

A. char p = \*malloc(100);  
B. char \*p = (char) malloc(100);  
C. char \*p = (char\*)malloc(100);  
D. char \*p = (char \*)(malloc\*)(100);

**11.What will be the output of the program ?**

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

```
int main()
{
    union a
```

```

{
    int i;
    char ch[2];
};

union a u;
u.ch[0]=3;
u.ch[1]=2;
printf("%d, %d, %d\n", u.ch[0], u.ch[1], u.i);
return 0;
}

```

- A. 3, 2, 515  
C. 3, 2, 5

- B. 515, 2, 3  
D. 515, 515, 4

### 12. Point out the error in the program?

```

struct emp
{
    int ecode;
    struct emp *e;
};

A. Error: in structure declaration
B. Linker Error
C. No Error
D. None of above

```

### 13. Point out the error in the program?

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

```

int main()
{
    struct emp
    {
        char n[20];
        int age;
    };
    struct emp e1 = {"Dravid", 23};
    struct emp e2 = e1;
    if(e1 == e2)
        printf("The structure are equal");
    return 0;
}

```

- A. Prints: The structure are equal  
B. Error: Structure cannot be compared using '=='

- C. No output
- D. None of above

**14.In which numbering system can the binary number  
*101101111000101* be easily converted to?**

- |                          |                              |
|--------------------------|------------------------------|
| <u>A.</u> Decimal system | <u>B.</u> Hexadecimal system |
| <u>C.</u> Octal system   | <u>D.</u> No need to convert |

**15.Assumming, integer is 2 byte, What will be the output of the program?**

```
#include<stdio.h>
int main()
{
    printf("%x\n", -1>>1);
    return 0;
}
```

- |                |                |
|----------------|----------------|
| <u>A.</u> ffff | <u>B.</u> 0fff |
| <u>C.</u> 0000 | <u>D.</u> fff0 |

**16.If an *unsigned int* is 2 bytes wide then, What will be the output of the program ?**

```
#include<stdio.h>
int main()
{
    unsigned int m = 32;
    printf("%x\n", ~m);
    return 0;
}
```

- |                |                |
|----------------|----------------|
| <u>A.</u> ffff | <u>B.</u> 0000 |
| <u>C.</u> ffdf | <u>D.</u> ddऽ  |

**17.What will be the output of the program?**

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

```
int main()
{
    printf("%d >> %d %d >> %d\n", 4 >> 1, 8 >> 1);
    return 0;
}
```

- |                                                 |
|-------------------------------------------------|
| <u>A.</u> 4 1 8 1                               |
| <u>B.</u> 4 >> 1 8 >> 1                         |
| <u>C.</u> 2 >> 4 Garbage value >> Garbage value |

D. 24

**18.What will be the output of the program?**

```
#include<stdio.h>
int main()
{
    char c=48;
    int i, mask=01;
    for(i=1; i<=5; i++)
    {
        printf("%c", c|mask);
        mask = mask<<1;
    }
    return 0;
}
```

A. 12400  
C. 12500

B. 12480  
D. 12556

**19.What will be the output of the program ?**

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

```
int main()
{
    int i=32, j=0x20, k, l, m;
    k=i|j;
    l=i&j;
    m=k^l;
    printf("%d, %d, %d, %d, %d\n", i, j, k, l, m);
    return 0;
}
```

A. 0, 0, 0, 0, 0  
C. 32, 32, 32, 32, 0

B. 0, 32, 32, 32, 32  
D. 32, 32, 32, 32, 32

**20.How many times "SDMIT" is get printed?**

```
#include<stdio.h>
int main()
{
    int x;
    for(x=-1; x<=10; x++)
    {
        if(x < 5)
```

```

        continue;
    else
        break;
    printf("SDMIT");
}
return 0;
}
A. Infinite times
C. 0 times
B. 11 times
D. 10 times

```

**21.What will be the output of the program?**

```
#include<stdio.h>
int main()
{
    int i=0;
    for( ; i<=5; i++)
        printf("%d", i);
    return 0;
}
A. 0, 1, 2, 3, 4, 5
C. 1, 2, 3, 4
B. 5
D. 6
```

**22.What will be the output of the program?**

```
#include<stdio.h>
int main()
{
    int a=0, b=1, c=3;
    *((a) ? &b : &a) = a ? b : c;
    printf("%d, %d, %d\n", a, b, c);
    return 0;
}
A. 0, 1, 3
C. 3, 1, 3
B. 1, 2, 3
D. 1, 3, 1
```

**23.What is the notation for following functions?**

1. int f(int a, float b)
 

```

{
    /* Some code */
}
```

2. int f(a, b)
 

```

int a; float b;
```

- ```

{
    /* Some code */
}

A. 1. KR Notation
    2. ANSI Notation
C. 1. ANSI Notation
    2. KR Notation
B. 1. Pre ANSI C Notation
    2. KR Notation
D. 1. ANSI Notation
    2. Pre ANSI Notation

```

**24.What will be the output of the program?**

```
#include<stdio.h>
void fun(int*, int*);
int main()
{
    int i=5, j=2;
    fun(&i, &j);
    printf("%d, %d", i, j);
    return 0;
}
void fun(int *i, int *j)
{
    *i = *i**i;
    *j = *j**j;
}
A. 5, 2
C. 2, 5
B. 10, 4
D. 25, 4
```

**25.What will be the output of the program?**

```
#include<stdio.h>
int reverse(int);

int main()
{
    int no=5;
    reverse(no);
    return 0;
}
int reverse(int no)
{
    if(no == 0)
        return 0;
    else
        printf("%d,", no);
```

```
    reverse (no--);  
}  
A. Print 5, 4, 3, 2, 1  
C. Print 5, 4, 3, 2, 1, 0      B. Print 1, 2, 3, 4, 5  
D. Infinite loop
```

## 26.What will be the output of the program?

```
#include<stdio.h>  
void fun(int);  
typedef int (*pf) (int, int);  
int proc(pf, int, int);
```

```
int main()  
{  
    int a=3;  
    fun(a);  
    return 0;  
}  
void fun(int n)  
{  
    if(n > 0)  
    {  
        fun(--n);  
        printf("%d,", n);  
        fun(--n);  
    }  
}
```

A. 0, 2, 1, 0,      B. 1, 1, 2, 0,  
C. 0, 1, 0, 2,      D. 0, 1, 2, 0,

## 27.What will be the output of the program?

```
#include<stdio.h>  
int func1(int);  
  
int main()  
{  
    int k=35;  
    k = func1(k=func1(k=func1(k)));  
    printf("k=%d\n", k);  
    return 0;  
}  
int func1(int k)  
{  
    k++;
```

```
    return k;
}
A.   k=35
C.   k=37
B.   k=36
D.   k=38
```

**28.What will be the output of the program ?**

```
#include<stdio.h>
int main()
{
    int a[5] = {5, 1, 15, 20, 25};
    int i, j, m;
    i = ++a[1];
    j = a[1]++;
    m = a[i++];
    printf("%d, %d, %d", i, j, m);
    return 0;
}
A.   2, 1, 15
C.   3, 2, 15
B.   1, 2, 5
D.   2, 3, 20
```

**29.What will be the output of the program ?**

```
#include<stdio.h>
int main()
{
    static int a[2][2] = {1, 2, 3, 4};
    int i, j;
    static int *p[] = {(int*)a, (int*)a+1, (int*)a+2};
    for(i=0; i<2; i++)
    {
        for(j=0; j<2; j++)
        {
            printf("%d, %d, %d, %d\n", *(*(p+i)+j), *(*(j+p)+i),
                   *(*(i+p)+j), *(*(p+j)+i));
        }
    }
    return 0;
}
A.   1, 1, 1, 1
C.   2, 3, 2, 3
B.   3, 2, 3, 2
D.   4, 4, 4, 4
1, 2, 1, 2
2, 3, 2, 3
3, 4, 3, 4
4, 2, 4, 2
```

<u>C.</u>	1, 1, 1, 1 2, 2, 2, 2 2, 2, 2, 2 3, 3, 3, 3	<u>D.</u>	1, 2, 3, 4 2, 3, 4, 1 3, 4, 1, 2 4, 1, 2, 3
-----------	--	-----------	--

### 30.What will be the output of the program ?

```
#include<stdio.h>
int main()
{
    static int arr[] = {0, 1, 2, 3, 4};
    int *p[] = {arr, arr+1, arr+2, arr+3, arr+4};
    int **ptr=p;
    ptr++;
    printf("%d, %d, %d\n", ptr-p, *ptr-arr, **ptr);
    *ptr++;
    printf("%d, %d, %d\n", ptr-p, *ptr-arr, **ptr);
    *++ptr;
    printf("%d, %d, %d\n", ptr-p, *ptr-arr, **ptr);
    ++*ptr;
    printf("%d, %d, %d\n", ptr-p, *ptr-arr, **ptr);
    return 0;
}
```

<u>A.</u>	0, 0, 0 1, 1, 1 2, 2, 2 3, 3, 3 1, 1, 1 2, 2, 2 3, 3, 3 3, 4, 4	<u>B.</u>	1, 1, 2 2, 2, 3 3, 3, 4 4, 4, 1 0, 1, 2 1, 2, 3 2, 3, 4 3, 4, 5
-----------	--	-----------	--

### 31.What will be the output of the program?

```
#include<stdio.h>
int main()
{
    typedef int LONG;
    LONG a=4;
    LONG b=68;
    float c=0;
    c=b;
    b+=a;
    printf("%d, ", b);
    printf("%f\n", c);
```

```

    return 0;
}
A. 72, 68.000000          B. 72.000000, 68
C. 68.000000, 72.000000  D. 68, 72.000000

```

**32.What will be the output of the program?**

```
#include<stdio.h>
#define MAX(x, y) ((x)>(y)) ? (x):(y);
```

```

int main()
{
    int i=10, j=5, k=0;
    k = MAX(++i, j++);
    printf("%d, %d, %d\n", i, j, k);
    return 0;
}
A. 12, 6, 12          B. 11, 5, 11
C. 11, 5, Garbage    D. 12, 6, Garbage

```

**33.Which of the following function is correct that finds the length of a string?**

<pre> int xstrlen(char *s) {     int length=0;     while(*s!='\0')     {         length++;         s++;     }     return (length); } int xstrlen(char *s) {     int length=0;     while(*s!='\0')         length++;     return (length); } </pre>	<pre> int xstrlen(char s) {     int length=0;     while(*s]!='\0')         length++;     return (length); } int xstrlen(char *s) {     int length=0;     while(*s!='\0')         s++;     return (length); } </pre>
---	---

**34.What will be the output of the program ?**

```
#include<stdio.h>
void swap(char *, char *);
```

```

int main()
{
    char *pstr[2] = {"Hello", "ThyaguTGS"};
    swap(pstr[0], pstr[1]);
}
```

```

printf("%s\n%s", pstr[0], pstr[1]);
return 0;
}
void swap(char *t1, char *t2)
{
    char *t;
    t=t1;
    t1=t2;
    t2=t;
}
A. ThyaguTGS
B. Address of "Hello" and
    "ThyaguTGS "
C. Hello
D. HhyaguTGS

```

**35.What will be the output of the program (sample.c) given below if it is executed from the command line?**

```

cmd> sample Jan Feb Mar
/* sample.c */
#include<stdio.h>
#include<dos.h>

int main(int arc, char *argv[])
{
    int i;
    for(i=1; i<=argc; i++)
        printf("%s ", argv[i]);
    return 0;
}

```

- |                       |                              |
|-----------------------|------------------------------|
| <u>A.</u> No output   | <u>B.</u> sample Jan Feb Mar |
| <u>C.</u> Jan Feb Mar | <u>D.</u> Error              |

## Reviewers

- 1.** Rashmi RR , Member of Technical Staff ,Oracle , Bangalore
- 2.** Sushmitha Joshi , Software Engineer ,Robert Bosh Engg and Business Soln Ltd(RBEI).
- 3.** Dr. Demian D Mello, Professor and HOD, Department of CSE, Canara College of Engineering, Dakshina Kannada.
- 4.** Dr.Harish Kenchannanavar, Professor , Department of CSE, GIT – Belgaum.
- 5.** Dr.S.B.Kulkarni , Professor , Department of CSE , SDMCET , Dharwad -580002
- 6.** Anand Uppar, Professor and HOD , Department of CSE , Sridevi Institute Of Technology , Mangaluru , Dakshina Kannada.
- 7.** Dharmanna L, Professor and HOD , Department of ISE ,SDMIT,Ujire-574240
- 8.** GP Hegde , Professor , Department of CSE, SDMIT –Ujire -574240.
- 9.** Raveendra Dastikop , Director , Web Enabled Learning Centre , SDMCET , Dharwad-580002
- 10.**Dr.Prabhakar C J, Professor , Department of CS, Kuvempu University, Shimoga.
- 11.**Vivek Bhat ,System Engineer, Tata Consultancy Services , Bangalore.
- 12.** Professor MariGowda .C.K, Acharya Institute of Technology , Bangalore.
- 13.** Smt .HemaMalini BH, Assoicate Professor ,Department of CSE, BMS Institute Of Technology and Management , Bangalore.
- 14.** Navile Nageshwara Naveen , Assistant Professor , Department of CSE, Coorg Institute of Technology, Coorg.
- 15.** Deepa.T.P, Assistant Professor , Department of Computer Science Engineering , Acharya Institute of Technology,Bangalore
- 16.** Siju V Soman ,Assistant Professor (senior) , SMVITM, Udupi.
- 17.** Rakshith MD, Assistant Professor, Department of CSE,SDMIT,Ujire -574240
- 18.** Pradeep GS , Assistant Professor ,Department of CSE,SDMIT ,Ujire -574240
- 19.** Saraswathi Koppad , Assistant Professor ,Department of CSE,SDMIT ,Ujire -574240

- 20.** Shobha Sharma, Assistant Professor ,Department of CSE,SDMIT ,Ujire -574240
- 21.** Taranath , Assistant Professor , Department of CSE, AIT, Chikmagalore.
- 22.**Shreenath Waramballi, Assistant Professor ,Department of CSE , SMVITM,Udupi.
- 23.**Venkatesh C, Assistant Professor , KVG College of Engineering and Technology , Sullya .

## Authors and Reviewer Profile



**Dr. Thyagaraju Gowda** with qualification B.Sc (1996), M.Sc (1998) from Kuvempu University , M.Tech (2002) from Mysore University and Ph.D (2014) from VTU is working as a Professor and HOD, in the department of Computer Science Engineering of Shri Dharmasthala Manjunatheshwara Institute of Technology , Ujire. He has more than 16 years of teaching and research experience. He has published more than 35 research publications in national and international journals and conference proceedings. He is one of the reviewer for th Elsevier Journal of Advanced Soft Computing. He is a associative editor for the International Journal of Advanced Pervasive Computing published by IGI publishing house,USA. He has authored books titled "Modeling User Context", "Context Aware Computing for Ubiquitous Computing Applications" and "Programming in C – Made Easy" which are published by Lambert Academic Publishing House, Germany. His area of interest is Smart Computing ,IoT , BiG Data and Design & Developing Innovative Computing Projects.



**Dr Latha C A** graduated from Mysore University in 1991, obtained M.Tech from NITK and Phd from Anna University, Chennai. She has a vast academic experience of 24 years serving in almost all the capacities. Presently she is serving the Academia as Professor in Computer Science and Engg department of Global Academy of Technology, Bangalore. She has in her credit several National and International Journal and Conference Publications including filing for a Patent in USPTO. Along with paper publications , Dr Latha C A is contributing to the research field as a reviewer for many International Journal and Conference Publications, including IEEE, EDAS web portal. Recently Elsevier awarded Dr Latha C A for “Outstanding contribution to reviewing”.





# yes I want morebooks!

Buy your books fast and straightforward online - at one of the world's fastest growing online book stores! Environmentally sound due to Print-on-Demand technologies.

Buy your books online at  
**[www.get-morebooks.com](http://www.get-morebooks.com)**

---

Kaufen Sie Ihre Bücher schnell und unkompliziert online – auf einer der am schnellsten wachsenden Buchhandelsplattformen weltweit!  
Dank Print-On-Demand umwelt- und ressourcenschonend produziert.

Bücher schneller online kaufen  
**[www.morebooks.de](http://www.morebooks.de)**

OmniScriptum Marketing DEU GmbH  
Heinrich-Böcking-Str. 6-8  
D - 66121 Saarbrücken  
Telefax: +49 681 93 81 567-9

[info@omnascriptum.com](mailto:info@omnascriptum.com)  
[www.omnascriptum.com](http://www.omnascriptum.com)







