Sri Eshwar College of Engineering
Coimbatore- 641 202

Unit I

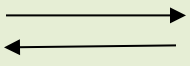
PROBLEM SOLVING FUNDAMENTALS

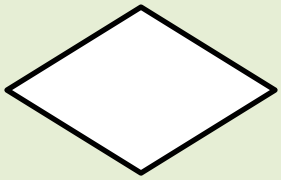
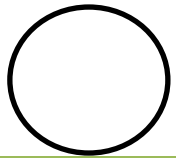
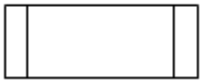
Introduction to problem solving - Flow Chart, Algorithm, Pseudo code - Procedural Programming (Modular and Structural)- Program Compilation, Execution, Debugging, Testing –Pre-processors -Basic features of C, Structure of C program - Data types- Storage Classes-Tokens in C- Input and Output Statements in C, Operators- Bitwise, Unary, Binary and Ternary Operators, Precedence and Associativity -Expression Evaluation

Flow Chart

- A flowchart is a **diagrammatic representation** of an algorithm.
- A flowchart is drawn using boxes of **different shapes** with lines connecting them to show the flow of control.
- The purpose of drawing a flowchart is to make **the logic of the program clearer** in visual form.
- The logic of the program is **communicated** in a much better way using a flowchart. Since flowchart is a diagrammatic representation, it forms a common medium of communication.

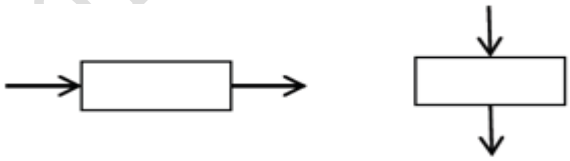
Flow Chart Symbols

Symbol	Symbol Name	Description
	Process	Represents arithmetic and logical instructions
	Terminal Symbol	Represent the start and stop of the program
	Flow Lines	Used to connect symbols

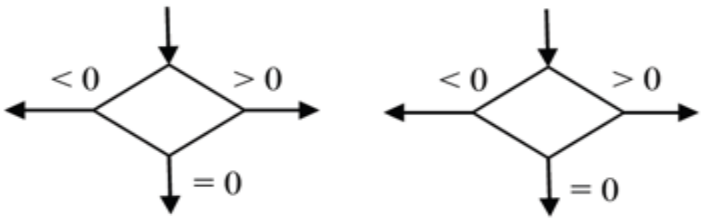
	Input/ Output	Represents input and output operation
	Decision	Represents decision making and branching
	Connector	Used to join different flow lines
	Sun function	Used to call function

Basic Guidelines for Preparing Flow Charts

- The flowchart should be clear, neat and easy to follow
- The usual direction of the flow of a procedure or system is from left to right or top to bottom
- Only one flow line should come out from a process symbol



- Only one flow line should enter a decision symbol. However, two or three flow lines may leave a decision symbol.



- Only one flow line is used in conjunction with terminal symbol.

- Ensure that the flowchart has a logical start and finish.
- It is useful to test the validity of the flowchart by passing through it with a simple test data.

Advantages of Flowcharts

a. Communication
- Flowcharts are better way of communicating the logic of a system to all concerned.

b. Effective Analysis
- With the help of flowchart, problem can be analyzed in more effective way.

c. Effective Coding
- The flowcharts act as a guide or blueprint during the system analysis and program development phase.

d. Proper Testing and Debugging
- The flowchart helps in debugging process.

e. Effective Program Maintenance
- Maintenance of running program becomes easy with the help of flow chart

Disadvantages of Flowcharts

- A complex and long flow chart may run into multiple pages, which becomes difficult to understand and follow
- Updating the flowchart with changing requirements is a challenging job.

Algorithm

- Algorithm is an ordered sequence of finite, well defined, unambiguous instructions for completing a task.
- An algorithm is defined as "**a step by step procedure for solving any problem**".

- It is an English –like representation of logic which is used to solve the problem.
- The algorithm is independent of any programming language.
- To accomplish a particular task, different algorithms can be written. They differ by their time and space.
- The algorithm can be implemented in many different languages by using different methods and programs.

Building Blocks of Algorithms

- Statements
- State
- Control Flow
- Functions

Statements / Instructions

- Algorithm consists of finite number of statements
- The statements must be in an ordered form
- Statements might include some following actions
  - Input data-information given to the program
  - process data-perform operation on a given input
  - Output data-Processed Result

State

- Transition from one process to another process under specified condition with in a time is called state.

Control flow

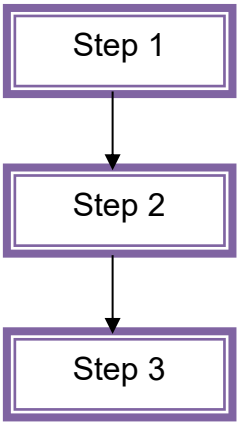
- The process of executing the individual statements in a given order is called control flow.
- The control can be executed in three ways

1. Sequence

2. Selection
3. Iteration

Sequence

- Instructions are executed in the order as they are written
- In short, sequence is a series of steps that are followed one after the other
- When an instruction is executed the execution control moves to the next immediate step



Example 1: Algorithm to find sum of two numbers

Step 1: Start

Step 2: Read two numbers Num1 and Num2

Step 3: Calculate sum, i.e. Total=Num1+Num2

Step 4: Display Total

Step 5: Stop

Example 2: Algorithm to calculate the area of circle

Step 1: Start

Step 2: Read input r as radius of circle

Step 3: Calculate area, i.e. Area=3.14*r*r

Step 4: Display Area

Step 5: Stop

Example 3: Algorithm to convert Fahrenheit to Centigrade

Step 1: Start

Step 2: Read Fahrenheit F

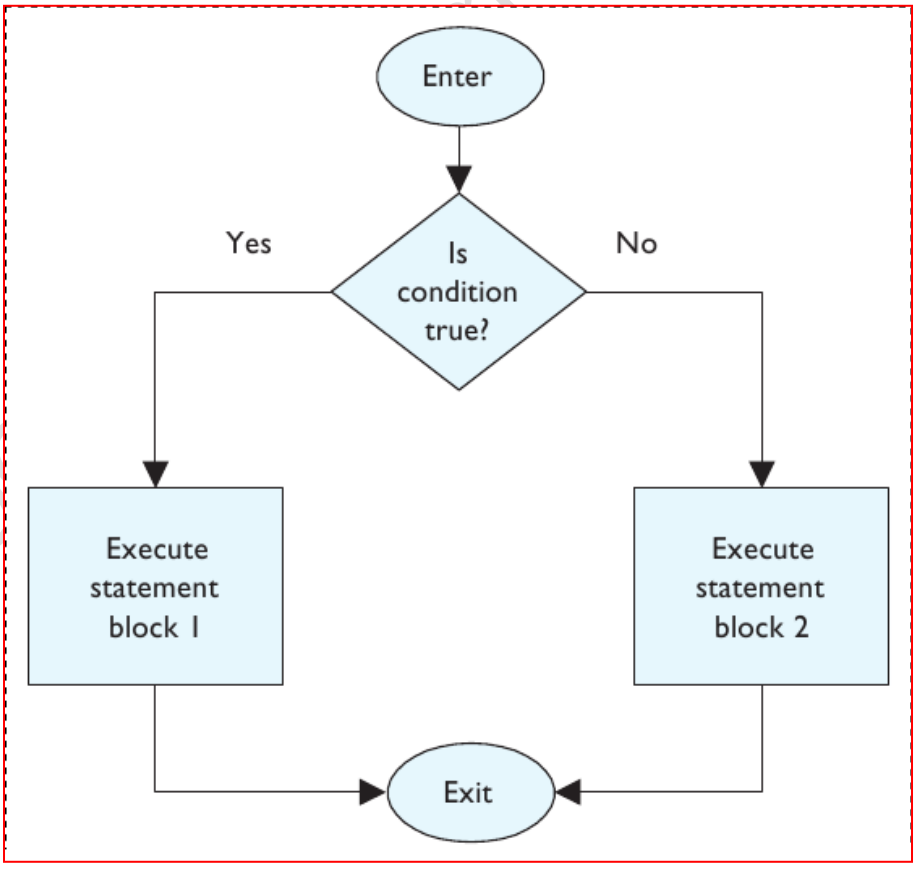
Step 3: Calculate Centigrade C=(F-32)*(5/9)

Step 4: Print C

Step 5: Stop

Selection Statements

- Decision control statements are used to alter the flow of control in a program
- The action is performed only when a particular condition is satisfied in a program
- It helps the programmer to transfer the control from one part to another part of the programs



Example 1 Algorithm to check whether the given number is odd or even

Step1: Start.

Step2: Input Number, i.e Num1.

Step3: If Num1 %2==0. If yes go to Step 4.Else go to Step 6.

Step4: Print number is even.

Step5: Go to Step 7.

Step6: Print number is odd.

Step7: End.

Example 2: Algorithm to find greatest among three numbers

Step 1: Start.

Step 2: Read the three numbers A, B, C.

Step 3: Compare A and B. If A is greater, store A in MAX, else store B in MAX.

Step 4: Compare MAX and C. If MAX is greater, output "MAX is greatest" else output "C is greatest.

Step 5: Stop.

Repetition or Iteration

- Execute the same lines of code for several times.
- In such cases repetition structure is used to repeat one or more statements for number of times.

Example: Algorithm to print all natural numbers upto n

Step 1: Start

Step 2: Get n value.

Step 3: Initialize i=1

Step 4: if (i<=n) go to step 5 else go to step 7

Step 5: Print i value and increment i value by 1

Step 6: Go to step 4

Step 7: Stop

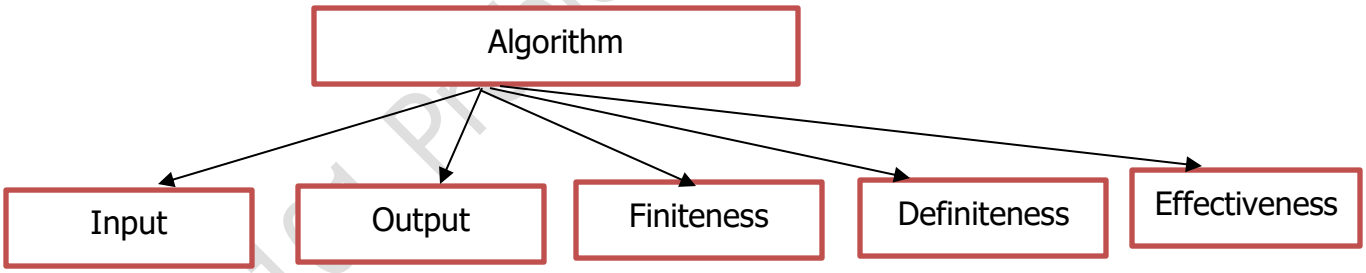
Functions

- Any complex problem will be simpler if the problem is broken smaller and the smaller problems are solved.
- Hence writing functions would increase the readability and efficiency of the algorithm.
- Once a function is written, it can be used over and over and over again

Benefits of Using Functions

- Reduction in line of code
- Code Reuse
- Better readability
- Easy to debug and test
- Improved maintainability

Properties/ Characteristics of an Algorithm



- **Finiteness**
  - The algorithm must terminate after a finite number of steps.
- **Definiteness**
  - Each instruction must be clear, well-defined and precise. There should not be any ambiguity.
- **Effectiveness**
  - Each instruction must be simple and carried out in finite amount of time.

- **Input**
  - It is a set of values as input data.
- **Output**
  - This is result of the program.

Qualities of an Algorithm

- **Accuracy**
  - Algorithm should provide accurate result than others.
- **Memory**
  - It should require minimum computer memory.
- **Time**
  - Lesser the time taken better the quality.
- **Sequence**
  - Procedure of an algorithm must be sequential form.
- **Result**
  - It should lead to correct result each time.

Pseudocode

- Pseudocode is an informal language used by programmers to develop algorithms.
- Pseudo means imitation and code refers to instructions written in programming language.
- Pseudocode cannot be compiled nor executed.
- There is no real formatting or syntax rules for Pseudo code.
- Pseudocode is a set of statements in plain English which may be translated later into a programming language.

- It does not include details like variable declaration, subroutines.
- It gives us the sketch of the program before actual coding.
- It is easy for a programmer and non-programmer to understand the general working of the program, because it is not based on any programming language.

Input	Read, obtain, Get, Prompt
Output	Print, Display, Show
Calculations	Compute, Calculate, Determine
Initiate	Set, Initialize
Add one	Increment

Basic Guidelines for Writing Pseudocode

- Write only one statement per Line.
- Capitalize initial keyword.
- Indent to show hierarchy.
- End multi-line structures.
- Keep statement language independent.

Write only one statement per Line

- Readability improves if just one action for the computer is written in one statement.
- A Pseudo code to add two numbers and display results.

```
READ num1, num2
Result = num1+num2
Write Result
```

Capitalize initial Keyword

- Keywords listed below should be written in capital letters.
  - READ
  - WRITE
  - IF
  - ELSE
  - ENDIF
  - WHILE
  - ENDWHILE
  - REPEAT

- UNTIL

Indent to show hierarchy

- Indentation is a process of showing the boundaries of the structure.
- Example 1

```
If a>b then
        PRINT a
ELSE
        PRINT b
```

- Example 2

```
READ name, class, m1, m2, m3
        Total=m1+m2+m3
        Average=Total/3
IF average is greater than 75
        Rank=Distinction
ENDIF
WRITE name, Total, Average, Rank
```

End multiline structure

- Each structure must be ended properly, which provides more clarity.
- Example: ENDIF for IF statement.

Keep statement language independent

- The programmer must never use the syntax of any programming language.

Control Structures used in Pseudo code

- There are three control structures used in Pseudo code. They are,
  - Sequence control structure.

- Selection control structure.
- Iteration control structure.

Sequence Control Structures

- In sequence control structure, the steps are executed in a linear order one after another.
- They are executed in the order in which they are written, from top to bottom.

EXAMPLE: Find product of any two numbers

READ values of A and B

COMPUTE C by multiplying A with B

PRINT the result C

STOP

Selection Control Structure

- It is a decision in which a choice is made between two alternative courses of action.
- IF-THEN-ELSE
  - In IF-THEN-ELSE selection structure, if the condition is true, the THEN part is executed.
  - Otherwise ELSE part is executed.
- Syntax

```
IF condition THEN
        Sequence 1
ELSE
        Sequence 2
END IF
```

- Example

READ values of A, B, C

IF A is greater than B THEN

ASSIGN A to MAX

ELSE

ASSIGN B to MAX

IF MAX is greater than C THEN

PRINT MAX is greatest

ELSE

PRINT C is greatest

STOP

Iterative Control Structures

- It is a loop (iteration) based on the satisfaction of some condition(s). It comprises of the following constructs:
  - WHILE…END WHILE
  - REPEAT…UNTIL
  - FOR…ENDFOR

Syntax for WHILE…ENDWHILE	Example: To print numbers from 1 to 100
WHILE condition Sequence **ENDWHILE**	n=1 **WHILE** n is less than or equal to 100 DISPLAY n INCREMENT n by 1 **ENDWHILE**
Syntax for REPEAT…UNTIL	Example: To print numbers from 1 to 100
REPEAT Sequence **UNTIL** condition	n=1 **REPEAT** **DISPLAY** n **INCREMENT** n by 1 **UNTIL** n is greater than 100

Syntax for FOR…ENDFOR	Example: To print N natural numbers
FOR (start-value to end-value) **DO** statement …….. **END FOR**	BEGIN GET n INITIALIZE i=1 FOR (i<=n)DO PRINT i i=i+1 END FOR END

Advantages of Pseudocode	Disadvantages of Pseudocode
▪ Easy to modify ▪ Implements structured concepts ▪ Simple because it uses English-like statements ▪ No special symbols are used	▪ It's not visual ▪ There is no accepted standard, it varies from company to company ▪ Cannot be compiled not executed

Structured Programming

- The programs generated using unstructured approach are meant for simple and small problems.
- If the problems gets lengthy, this approach becomes too complex and obscure.
- Using structured programming, a program is broken down into small independent tasks that are small enough to be understood easily.
- These tasks are developed independently, and each task can carry out the specified task on its own, without help of any other task.
- Structured Programming can be performed in two ways:

A. Procedural Programming

- This programming has a single program that is divided into a small pieces called ***procedures (*****also known as functions, routines, subroutines).**
- These procedures are combined into **one single location** with the help of return statements.
- From the main or controlling procedure, a procedure call is used to invoke the required procedure.
- Procedural codes are very **difficult to maintain**, if the code is huge.
- Procedural languages do not have automatic memory management as like in Java. Hence, it makes the programmer to concern more about the memory management of the program.

B. Modular Programming

- The programs coded with procedural paradigms usually fit into **a single code file and are meant for relatively smaller programs.**
- However, if the program gets large, then **modular way of programming** is recommended.
- In case of modular programming, large programs are broken down into a number of smaller programs units known as ***modules.***
- Each module is designed to perform a specific function.
- A program, therefore divided into several smaller parts that interact and build the whole program.

Introduction to C Programming

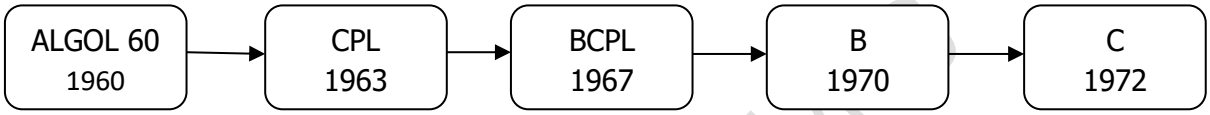
- C is a popular **general purpose programming** language
- C language has been designed and developed by ***Dennis Ritchie*** *at Bell Laboratories in 1972*
- The source code of **Unix Operating System** is coded in C
- C runs under **number of operating system** including MS-DOS
- C programs are efficient, fast and **highly portable**, that is programs written in one computer can be run on another with little or no modification.
- C is a **structured language**
  - Large programs are divided into small programs called functions
  - It is easy for debugging, testing, and maintenance if a language is structured one
  - These languages are easier and most of the developers prefer these languages to non-structured ones like Basic and Cobol

History of C

- The algorithmic language, ALGOL 60 is the root for all modern programming languages. ALGOL was developed by an international committee in the year 1960.

It was the very first language to use the concept of block structure and to introduce the idea of structured programming.

- In 1967 Martin Richards developed BCPL (Basic Combined Programming Language).
- BCPL was basically a type-less (had no concept of data types) language which facilitated direct access of memory. This, made it useful for system programmers.
- In 1970 Ken Thompson developed a language called B.
- C was developed by Dennis Ritchie in 1972 that took concepts from ALGOL, BCPL, and B.
- In addition to the concepts of these languages, C also supports the concept of data types.



Features of C

- Structured Programming Language.
- Portable
- Extensible
- Middle Level Language
- Simple
- Powerful
- Memory management
- Pointer
- Case sensitive

Structure of a C Program

Documentation Section
Link Section
Definition Section
Global declaration section
main () Function section
{

Declaration Part
Executable Part

}
Subprogram section
{

Function 1
Function 2
…….
…….
Function n

}

Documentation section

- The documentation section consists of set of comment lines giving the name of the program, the author details and other details.

Link section

- The Link section provides instructions to the compiler to link functions from the system library such as **using # include directive.**

Definition Section

- The definition section defines all symbolic constants such as # define directive.
- Syntax : **#define constant_name constant_value**

Global Declaration Section

- The global declaration section contains variable declarations which can be accessed anywhere within the program.

Main section

- Every C program must have one main function. This section contains two parts
  - **Declaration part**
    - The Declaration part declares all variables used in executable part.
  - **Executable part**
    - The executable part contains set of statements within open and close braces.
    - Execution of the program beings at the opening braces and ends at the closing braces.

User defined Function section

- The user defined function section (or) the sub program section contains user defined functions which are called by main function.
- Each user defined function contains the function name, the argument and the return value.

Simple example to demonstrate the structure of a C Program

```
/*
        * Name: simple program to explain structure of the C Program
        * Author: C_programmer
        * Created Date: 15/05/2019
        * Last Modified: 22/05/2019
*/
//Linking required Library
#include<stdio.h>
//defining a constant
# define LENGTH 20
// defining a global variable
int max_length= 15;
// declaring an user defined function which is defined later
float addNumbers(float a, float b);
void main()
 {
// declaring local variable for main ()
int localVariable=50;
//Accessing a defined constant
printf("Max array length is %d\n",LENGTH);
//Accessing a global variable
printf("Max input length is %d\n",maxlength);
//Accessing a user defined function
printf("Summation of 5.5 and 8.3 = %f\n",addNumbers(5.5,8.3));
```

```
}
float addNumbers(float a, float b)
{
return (a+b);
}
```

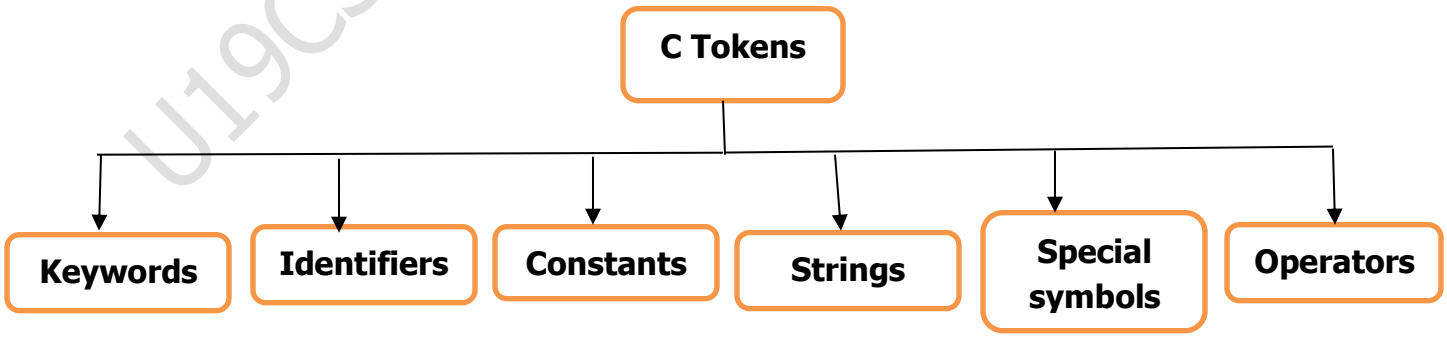
Character set of C

- The characters in C are classified into the following categories
  - Letters
  - Digits
  - Special symbols
  - White space characters

Letters	Uppercase letters A,B,.....Z Lowercase letters a,b,.....z
Digits	0,1,2.....9
Special symbols	& , . ^ ; : ? ` " ! | / \ ~ _ $ % * - + <> ( ) [ ] { } #
White space characters	Blank space, \b Horizontal tab space, \t Vertical return, \v Carriage return, \r Form feed, \f New line, \n

C Tokens

- The smallest individual units of a C Program are known as tokens.



C Keywords

- Keywords are reserved words whose meaning has already explained to the compiler.
- Keywords are also called as reserved words.
- They must be written in lower case.
- There are 32 keywords available in C.

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

Identifiers

- Identifiers refers to **name of any object.**
- Identifiers are basically names given to program elements such as variables, arrays, and functions.
- Identifiers may consist of sequence of letters, numerals, or underscores.

Rules for Forming Identifier Names

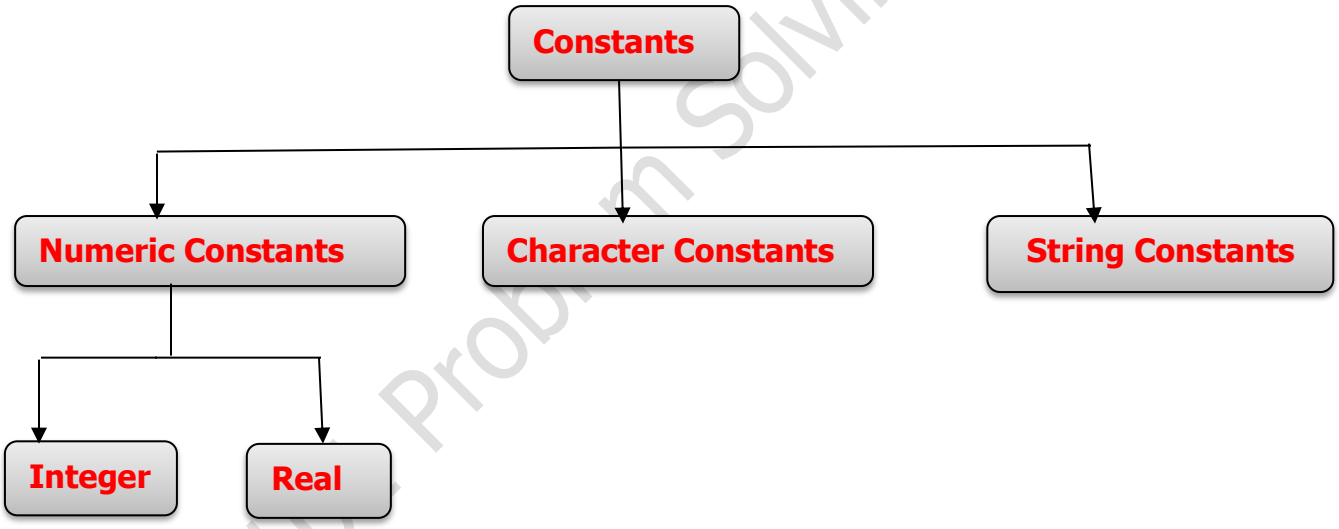
- ➔ Identifiers cannot include any special character or punctuation marks except the underscore '_'.
- ➔ There cannot be two successive underscores.
- ➔ Keywords cannot be used as identifiers.
- ➔ It is case sensitive: upper case letters are treated different from lower case letters. For example, 'DataSize' and 'datasize' are different.
- ➔ Identifiers are formed using the combination of alphabets, digits or underscores.
- ➔ The first character in the identifier must be an alphabet or underscore and it cannot be a digit.

- ➔ Blank spaces are not allowed within identifiers.
- ➔ The maximum length allowed for identifier names are 31 characters.

Valid Identifiers	roll_number, marks, name, emp_number, basic_pay, HRA, DA, dept_code, DeptCode, RollNo, EMP_NO
Invalid Identifiers	23_student, %marks, @name, #emp_number, basic.pay, -HRA, (DA), &dept_code, auto

Constants

- ➔ Constants are fixed values and they remain unchanged during the execution of the program. The constants are as flows:



Integer Constants

- ➔ Integer constant consists of a sequence of digits without any decimal point.
- ➔ An integer literal constant must have at **least one digit.**
- ➔ It can be either **positive or negative.**
- ➔ **No special characters and blank spaces** are allowed within an integer literal constant.
- ➔ If an integer literal constant **starts with 0,** then it is assumed to be in octal number system
  - Example: 023 means 23 is in an octal number system and it is equivalent to 19 in the decimal number system.

- If an integer literal constant starts with **0x or 0X,** then it is assumed to be in a hexadecimal number system
  - Example: 0X23 or 0x23 means 23 is in a hexadecimal number system and is equivalent to 35 in the decimal number system.
- The following are valid decimal constants.
  0, 1, -123, 456, 789, 30001
- The following are invalid decimal constants.

1,000	Comma is not allowed
10.0	Decimal point is not allowed
657 57	Blank space is not allowed
044	The first digit cannot be zero
128-256	Special character – is not allowed

- The Valid octal integer constants are: 0, 07, 016, 0747, 0777
- The following are invalid octal constants.

715	The first digit must be 0
1192	Illegal digit 9
05.77	Decimal point is not allowed

- The valid hexadecimal integer constants are 0X, 0XAAA, 0XABCD, and 0X11234.
- The following are invalid hexadecimal integer constants.

0123	Does not begin with 0X.
0X2A.23	Decimal point not allowed.
0XABH	Illegal character H.

Floating point constants (or) real constants

- Integer numbers are inadequate to express numbers that have a **fractional part.**
- Floating point literal constants are values **with decimal point.**
- A fractional floating point literal constant must have at least one digit.
- It can be either positive or negative.
- No special characters and blank spaces are allowed within a floating point literal constant.
- Floating point literal constants can written in a **fractional form or in an exponential form.**
- A floating point literal constant in an exponential form has two parts: **Mantissa part and the exponent part.** Both are separated by e or E.
- The mantissa can be either positive or negative. The default sign is positive.
- The mantissa part should have at least one digit.

- Example : -2.5E12, -2.5e-12, 2e10

➔ The following are valid floating point constants.

1.0
0.123
501.112
0.0000001
15.1E+10
0.123E-5
0.123E+4
0.00012E+10

➔ The following are invalid floating point constants

1.006.12	Comma is not allowed
50	Either a decimal point or an exponent must be required
1.2E+5.1	The exponent must be an integer quantity
14.2E 10	Blank space in the exponent is not allowed.

Character Constants

➔ A character constant is a single character enclosed in **single quotation marks.**
➔ For example, 'a' and '@' are character constants.
➔ In computers, characters are store using machine's character set using ASCII codes.
➔ All escape sequences mentioned below are also a character constant.

S.No	Escape Sequence	Character Value	Action on Output Device
1.	\'	Single Quotation Mark	Prints '
2.	\"	Double Quotation Mark	Prints "
3.	\?	Question Mark	Prints?
4.	\\	Backslash character	Prints\
5.	\a	Alert	Alerts by generating beep
6.	\b	Backspace	Moves the cursor one position to the left of its current position
7.	\f	Form feed	Moves the cursor to the beginning of the next page
8.	\n	New line	Moves cursor to the beginning of the next line

9.	\f	Carriage return	Moves the cursor to the beginning of the current line
10.	\v	Vertical tab	Vertical tab
11.	\0	Null character	Prints nothing
12.	\t	Horizontal tab	Moves the cursor to the next horizontal tab stop.

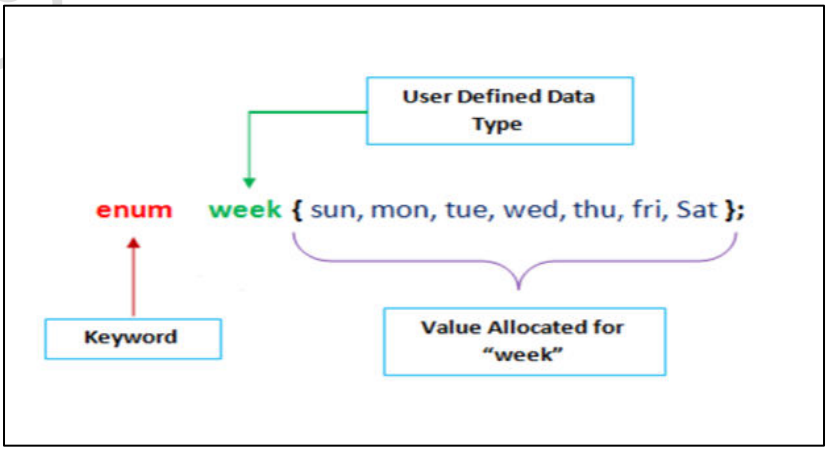
String Constants

- ➔ A string constant is a sequence of characters enclosed in **double quotes.**
- ➔ The following are valid string constants.

1. "Hello" 2. "044-256"
3. "$9.95" 4. "The area=" 5."Hello World"

Enumeration Constants

- ➔ C provides a user defined data type known as "enumerated data type".
- ➔ It attaches names to numbers, thereby increases the readability of the program.
- ➔ To define enumerated data type the keyword enum is used.
- ➔ **Syntax:** enum tagname (val1, val2,.....,valn};
- ➔ In above syntax enum is a keyword. It is a user defined data type.
- ➔ In above tagname is our own variable name. tagname is any variable name.
- ➔ val1, val2, val3, are used to create set of enum values.



Example	Explanation
enum COLORS {RED, BLUE, BLACK, GREEN, YELLOW, PURPLE, WHITE};	RED=0, BLUE=1, BLACK=2, GREEN=3, YELLOW=4, PURPLE= 5, WHITE =6
enum COLORS {RED=2, BLUE, BLACK=5, GREEN=7, YELLOW, PURPLE, WHITE=15};	RED=2, BLUE=3, BLACK=5, GREEN=7, YELLOW=8, PURPLE=9, WHITE=15.

C Program to Illustrate enum data type	Output
#include <stdio.h> int main() { enum COLORS {RED=2, BLUE, BLACK=5, GREEN=7, YELLOW, PURPLE, WHITE=15}; printf("\n RED= %d",RED); printf("\n BLUE= %d",BLUE); printf("\n BLACK= %d",BLACK); printf("\n GREEN= %d",GREEN); printf("\n YELLOW= %d",YELLOW); printf("\n PURPLE= %d",PURPLE); printf("\n WHITE= %d",WHITE); return 0; }	RED= 2 BLUE= 3 BLACK= 5 GREEN= 7 YELLOW= 8 PURPLE= 9 WHITE= 15

Declaring Constants

- To declare a constant, precede the normal variable declaration with *const* keyword.
- Syntax: ***const datatype variable_name = constant__value;***
- Example: const float pi=3.14;
- However another way to designate a constant is to use the pre-processor command *define.*
- Like another preprocessor command define is preceded with a # symbol.
- Example: #define pi 3.14159

Variables

- A variable is an entity whose value can vary during the **execution of a program.**
- The compiler allocates memory for a particular variable based on the type.

- In C, a variable is a data name used for storing a value.
- Its value may be changed during the program execution.
- The value of the variable keeps on changing during the execution of a program.

Declaration of Variables

- The declaration of variables must be done before they are used in the program.
- Syntax: `data_type var1, var2,..... varn;`
- Example: int a,b,c;

Initializing Variables

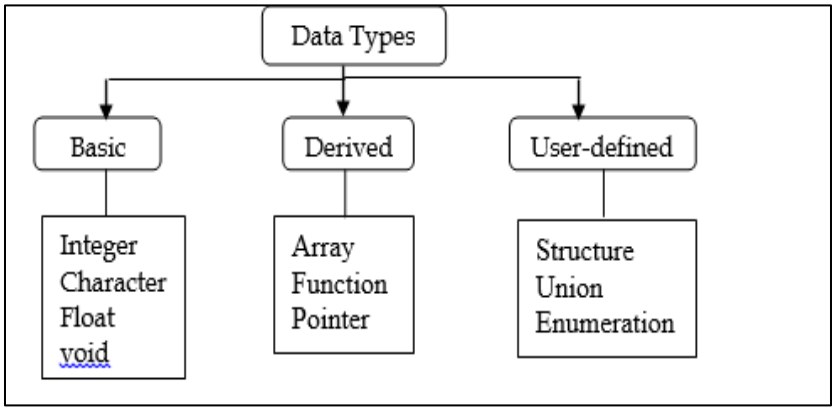
- Variable declared can be assigned or initialized using an assignment operator '='.

```
variable_name = constant;

or

data_type variable_name=constant;
```

- **Example:** int y=2;

Data types

- C language supports different types of data. The data type defines the possible values that an identifier can have and the valid operations that can be applied on it.
- Data types supported by C are broadly classified into following categories:
  - **Basic data types (Primitive data types).**
  - **Derived data types.**
  - **User-Defined data types.**



Basic Data types

- The five fundamental data types available in C are **integer, character, floating point numbers, double precision floating point numbers and void**. The following table depicts their size and range of values.

Data Type	Keyword	Size in Bytes	Range of Values
Integer	int	2	-32768 to +32767
Character	char	1	-128 to +127
Floating point	float	4	3.4 e-38 to 3.4 e+38
Double precision floating point	double	8	1.7 e-308 to 1.7 e+308
Valueless	void	0	

Integer data type

- Integer data type represents **whole number.**
- The range of values of an integer is dependent on the computer.
- C has 3 classes of integer storage namely int, short int and long int.
- All the three classes have both signed and unsigned forms.
- In general, integer needs one word of storage and size of the word may be **2 bytes (in 16-bit machine) or 4bytes (in 32-bit machine) which varies as per the machine.**
- The table gives the description on various integer data types in 16 bit machine.

Data type	Size in Bytes	Format string	Range of values
int (or) signed int	2	%d	-32768 to +32767

unsigned int	2	%u	0 to 65535
short int (or) signed short int	2	%d	-32768 to +32767
unsigned short int	2	%u	0 to 65535
long int (or) signed long int	4	%ld	-2147483648 to +2147483647
unsigned long int	4	%lu	0 to 4294967295

Floating point type

- Floating point data type can hold real numbers such as: 2.34, -9.382, and 5.0 etc.
- A floating point variable is declared by using either ***float or double*** keyword.

S. No	Float	Double
1.	Size of float is four bytes	Size of double is 8 bytes
2.	Precision of float is 6 digits	Precision of double is 15 digits

Data type	Size in Bytes	Format string	Range of values	Precision (decimal places)
float	4	%f	3.4 e-38 to 3.4 e+38	6
double	8	%lf	1.7e-308 to 1.7e+308	15
long double	10	%Lf	3.4 e-4932 to 3.4 e+4932	19

Character type

- Keyword ***char*** is used for declaring character type variables.
- Example char test = 'h';

Data type	Size in Bytes	Format string	Range of values
char (or) signed char	1	%c	-128 to +127
unsigned char	1	%c	0 to 255

Void data type

- Void actually refers to an object that **does not have a value of any type.**
- Void data type is used in function definition it means that function will not return any value.

Derived data type

S.No	Data types	Description
1.	Array	Arrays are sequences of data items having homogeneous values. They have adjacent memory locations to store values.
2.	Functions	A function is a group of statements that together perform a task.
3.	Pointers	It's powerful C feature which is used to access the memory and deal with their addresses.

User defined data type

S.No	Data types	Description
1.	Structure	It is a collection of variables of different types under single name. This is done to handle data efficiently. struct keyword is used to define a structure. Each element in a C structure is called member.
2.	Union	C Union is also like structures, i.e. collection of different data types which are grouped together. Each element in a union is called member. Union and structure in C are same in concepts, except allocating memory for their members.
3.	Enum	Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain. Enum keyword is used to define the enumerated data type.

Operators in C

- ✓ An Operator is a symbol that specifies the **mathematical, logical, or relational operation to be performed.**
- ✓ C Language supports different types of operators, which can be used with variables and constants to form expressions.

✓ These operators can be categorized in to the following major groups.

- ✓ **Arithmetic Operators**
- ✓ **Relational Operators**
- ✓ **Logical Operators**
- ✓ **Unary Operators**
- ✓ **Conditional Operators**
- ✓ **Comma Operator**
- ✓ **Bitwise Operator**
- ✓ **Sizeof operator**
- ✓ **Address-of operator (&)**

Arithmetic Operators

✓ Arithmetic Operators work on **numeric data type.** They are used to perform operations like addition, subtraction, multiplication, division etc.

Operator	Name	Expression	Example	Output
+	Addition	x+y	15+4	19
-	Subtraction	x-y	15-4	11
*	Multiplication	x*y	15*4	60
/	Division	x/y	15/4	3.75
%	Modulus	x%y	15%4	3

Program Using Arithmetic Operators	Output
#include <stdio.h> int main() { int a,b; printf("Enter the first number:\n"); scanf("%d",&a); printf("Enter the second number:\n"); scanf("%d",&b); printf("\n Addition=%d",a+b); printf("\n Subtraction=%d",a-b); printf("\n Multiplication=%d",a*b); printf("\n Division=%.2f",(float)a/b); printf("\n Modulo=%d",a%b);	Enter the first number: 5 Enter the second number: 2 Addition=7 Subtraction=3 Multiplication=10 Division=2.50 Modulo=1

return 0; }	

Relational Operators

- ✓ Relational Operators provide the relationship between the two expressions.
- ✓ If the relation is true then it returns a **value 1, otherwise 0 for false condition.**

Operator	Name	Expression	Example	Output
>	Greater than	x>y	54>18	True
<	Lesser than	x<y	54<18	False
= =	Equal to	x= = y	54 = = 18	False
!=	Not equal to	x!=y	54!=18	True
>=	Greater than or equal to	x>=y	54>=18	True
<=	Less than or equal to	x<=y	54<=18	False

Program Using Relational Operators	Output
#include <stdio.h> int main() { int x=10,y=20; printf("\n %d < %d = %d",x,y,x<y); printf("\n %d = = %d = %d",x,y,x==y); printf("\n %d != %d = %d",x,y,x!=y); printf("\n %d > %d = %d",x,y,x>y); printf("\n %d <= %d = %d",x,y,x<=y); printf("\n %d >= %d = %d",x,y,x>=y); return 0; }	10 < 20 = 1 10 = = 20 = 0 10 != 20 = 1 10 > 20 = 0 10 <= 20 = 1 10 >= 20 = 0

Logical Operators

- ✓ Operators which are used to combine **two or more relational operations** are called *logical operators.*
- ✓ C supports three logical operators- **logical AND (&&), logical OR (||), and logical NOT (!).**

- ✓ Logical AND (&&) operator provides true when both expressions are true otherwise 0.
- ✓ Logical OR (||) operator provides true when one of the expressions is true otherwise 0.
- ✓ The Logical NOT operator(!=) provides 0 if condition is true, otherwise 1

Program Using Logical Operators	Output
#include <stdio.h> int main() { printf("%d\n", 5>3 && 5<10); printf("%d\n",8>5 || 8<2); printf("%d\n",!(8==8)); }	1 1 0

Unary Operators

- ✓ **Unary Minus**
  - Unary minus (-) operator is strikingly different from the binary arithmetic operator that operates on two operands and subtracts the second operand from the first operand.
  - When an operand is preceded by a minus sign, the unary operator negates its value.
  - For example,
    int a, b=10;
    a= - (b)
  - The result of this expression is a=-10, because variable b has positive value.
  - After applying unary minus operator (-) on the operand b, the value becomes -10, which indicates it as a negative value.
- ✓ **Increment Operator (++) and Decrement Operator (--)**
  - The increment operator ++ causes its operand to be increased by 1 whereas decrement operator – causes its operand to be decreased by 1.

++x	Pre-increment operation
x++	Post- increment operation
--x	Pre-decrement operation
x--	Post- decrement operation

- The ***pre increment operation*** **(++x)** increments x by 1 and then assigns the value to x.
- The ***post increment*** **operation (x++)** assigns the value to x and then increments by 1.
- The ***pre decrement*** **operation (--x)** decrements by 1 and then assigns to x.
- The ***post decrement*** **operation (x--)** assigns the value to x and then decrements by 1.

Program Using Pre increment & Pre decrement Operation	**Output**
#include <stdio.h> int main() { int num=3; // Pre increment operation printf("\nThe value of num=%d",num); printf("\nThe value of ++num=%d",++num); printf("\nThe value of num=%d",num); //Pre-decrement operation printf("\nThe value of num=%d",num); printf("\nThe value of --num=%d",--num); printf("\nThe value of num=%d",num); }	The value of num=3 The value of ++num=4 The value of num=4 The value of num=4 The value of --num=3 The value of num=3
Program Using Post increment & Post decrement Operation	**Output**
#include <stdio.h> int main() { int num=3; // Post increment operation printf("\nThe value of num=%d",num); printf("\nThe value of num++=%d",num++); printf("\nThe value of num=%d",num); //Post-decrement operation printf("\nThe value of num=%d",num); printf("\nThe value of num--=%d",num--); printf("\nThe value of num=%d",num); }	The value of num=3 The value of num++=3 The value of num=4 The value of num=4 The value of num--=4 The value of num=3

Conditional Operator

- ✓ It is also called as *ternary operator*, which operates on three operands.

Syntax: expression 1? expression 2: expression 3

- ✓ Here expression 1 is evaluated first; if it is true, then the value of expression 2 is e result; otherwise expression 3 is the result.

Program using conditional operator	Output
#include <stdio.h> int main() { int a=20,b=10,z; z=(a>b) ? a : b; printf("\n %d is the biggest value",z); return 0; }	20 is the biggest value

Comma Operator

- ✓ The *comma operator* is used to **separate two or more expressions.**
- ✓ The comma operator has lowest priority among all the operators.
- ✓ It is not essential to enclose the expression with comma operators within parenthesis.
- ✓ For example,
  - a=2, b=4, c= a + b;
  - (a = 2, b=4, c=a+b);

Program using Comma operator	Output
#include <stdio.h> int main() { int a=20,b=10; printf("\n Addition = %d \n Subtraction = %d",a+b,a-b); printf("\n Multiplication = %d \n Division = %d",a*b,a/b); return 0; }	Addition = 30 Subtraction = 10 Multiplication = 200 Division = 2

Bitwise Operators

- ✓ C language is advantageous over other languages since it allows the user to access memory bits directly using bitwise operators.

- ✓ C provides six operators for bit manipulation.
- ✓ Bitwise operator operates on the individual bits of the operands.

Operators	Operation
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Truth tables for bitwise AND, bitwise OR and bitwise EXOR operations

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Program using Bitwise operator	Output
#include <stdio.h> int main() { int a=4,b=3; printf("\n a & b = %d",a&b); printf("\n a || b = %d",a||b); printf("\n a ^ b = %d",a^b); printf("\n a << 2 = %d",a<<2); printf("\n a >> 2 = %d",a>>2); printf("\n ~a = %d",~a); return 0; }	a & b = 0 a || b = 7 a ^ b = 7 a << 2 = 16 a >> 2 = 1 ~a = -5

Assignment Operators

- ✓ A variable can be assigned a value by using an assignment operator. The assignment operators in C Language are given below.

Operator	Name	Example	Equivalent to
=	Equal to	X=10	X=10
+=	Plus equal to	X+=10	X=X+10
-=	Minus equal to	X-=10	X=X-10
*=	Multiplication equal to	X*=10	X=X*10
/=	Division equal to	X/=10	X=X/10
%=	Modulus equal to	X%=10	X=X%10
&=	AND Equal to	X&=10	X=X&10
|=	OR Equal to	X|=10	X=X|10
^=	XOR Equal to	X^=10	X=X^10
>>=	Right shift equal to	X>>=2	X=X>>2
<<=	Left shift equal to	X<<=2	X=X<<2

Program using Assignment operator	Output
#include <stdio.h> int main() { int a=10,b=20,c=50,x=5; printf("\n Initial value of a=10,b=20,c=50,x=5\n"); printf("%d\n",a+=x); printf("%d\n", b-=x); printf("%d",c*=x); return 0; }	Initial value of a=10,b=20,c=50,x=5 15 15 250

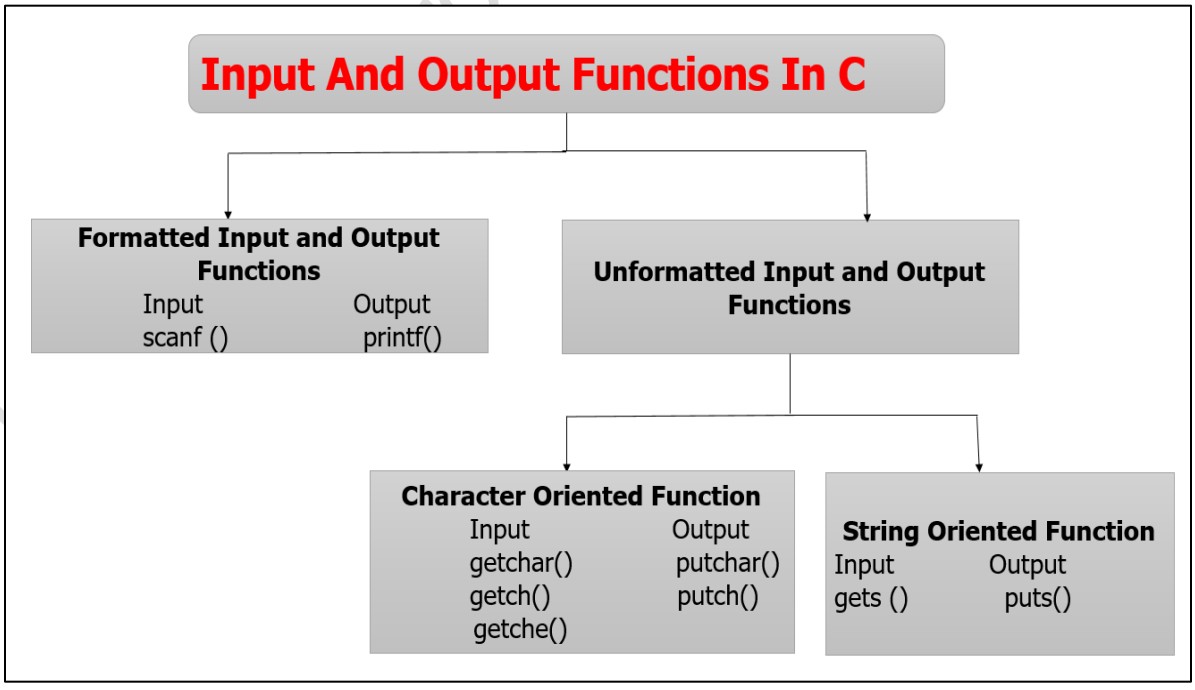
sizeof() and '&' Operator

- ✓ The sizeof() operator gives the **bytes occupied by a variable.**
- ✓ The number of bytes occupied varies from variable to variable depending upon its data types.
- ✓ ***The '&' operator prints address of the variable in the memory.***

Program using sizeof and address operator	Output
#include <stdio.h> int main() { int x=2; float y=2; printf("\n The size of (x)= %d bytes",sizeof(x)); printf("\n The size of (y)= %d bytes",sizeof(y)); printf("\n The Address of x=%u and y=%u",&x,&y); }	The size of (x)= 2 bytes The size of (y)= 4 bytes The Address of x=4066 and y=25096

Input and Output Statements in C

- ✓ C provides many functions for input/output operations. The input functions are used in reading data from **input devices like keyboard, mouse etc.** The output functions are used to display data on the monitor, printer or file storage.



Character Oriented I/O Functions

✓ Reading a Character

- Reading a single character from the keyboard can be done by using the function getchar ()
- Syntax: **variable_name= getchar();**
- C supports many character testing functions which receive a character typed from the keyboard and tests whether it is letter or a digit or a special character and prints out a message accordingly.

Function	Test
isdigit(c)	Is c a digit?
isalpha(c)	Is c is an alphabetic character?
islower(c)	Is c a lower case letter?
isupper(c)	Is c a upper case letter?
ispunct(c)	Is c a punctuation mark?
isspace(c)	Is C a white space character?
isprint(c)	Is C a printable character?

- The character functions are contained in the file ctype.h and therefore the statement ***# include<ctype.h>*** must be included in the program.

✓ Writing a Character

- The function putchar () is used for writing characters one at a time to the terminal.
- The general form of the putchar () function is **putchar(variable_name);**

Program to Read a line of text and display it	Output
#include <stdio.h> int main() { char c; do{ c=getchar(); putchar(c); } while(c!='\n'); return 0; }	**Input** Good Morning **Output** Good Morning

Program to test if a character is an alphabet or a digit and if it is a character then convert lower to uppercase or upper to lower case	Output
#include <stdio.h> #include<ctype.h> int main() { char c; printf("Enter a character:"); c=getchar(); if(isalpha(c)) { printf("The Character is a letter "); if(islower(c)) putchar(toupper(c)); else putchar(tolower(c)); } else if(isdigit(c)) { printf("The Character is a digit"); } else printf("The character is not alphanumeric"); return 0; }	1. Enter the character a The character is a letter A 2. Enter the character 9 The character is a digit 9 3. Enter the character & The character is not alphanumeric

- ✓ **getch() and getche()**
  - These functions read any alphanumeric characters from standard input device.
  - The character entered is not displayed by getch () function.

Program to show the effect of getche() and getch()	Output
#include <stdio.h> void main() { clrscr(); printf("Enter any two alphabets"); getche(); getch(); }	Enter any two alphabets A In above program, even though the two characters are entered, the user can see only one character on the screen. The second character is accepted but not displayed on the console. The **getche()** accepts and displays the character whereas

	getch() accepts but does not display the character.

- ✓ putch( )
  - This function prints any **alphanumeric character** taken by standard input device.

Program using getch() and putch() function	Output
#include <stdio.h> #include <conio.h> void main() { char ch; printf("Press any key to continue…"); ch=getch(); printf("\n You Pressed : "); putch(ch); }	Press any key to continue You Pressed : 9

String Oriented I/O Functions

- ✓ **Gets () function** is sued to read the string from the standard input device (keyboard)
- ✓ **Syntax: gets (character type of array variable);**
- ✓ The puts ()function is used to display / write the string to the standard output device (monitor)
- ✓ **Syntax: puts (character type of array variable);**

Program using gets() and puts()function	Output
#include <stdio.h> #include <conio.h> void main() { char a[10]; puts("\n Enter the string: "); gets(a); puts("\n the given string is"); puts(a); }	Enter the string: balamurugan the given string is balamurugan

Formatted Output Function

- ✓ C provides the printf function to display the data on the monitor.
- ✓ This function is used to display any combination of numerical values, single characters and strings.

Syntax: printf("control string",arg1,arg2,….,argn);

- ✓ The 'control string' indicates how many arguments follow and what their types are.
- ✓ The arguments arg1, arg2, ….argn are variables whose values are formatted and printed according the specifications of the control string.
- ✓ The arguments should match in number, order and type with the format specifications.
- ✓ The control string or format specification must begin with a percent sign (%) followed by a conversation character.

Formatting Integer Input

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a=1234;
printf("%d\n",a);
printf("%6d\n",a); // Right Justified
printf("%-6d\n",a); // Left Justified
printf("%06d\n",a); // Padding
}
```

Output

```
1234
  1234
1234
001234
```

Formatting Float Input

```
#include <stdio.h>
#include <conio.h>
 void main()
{
float y=3456.12065;
printf("%f\n",y); // default precision
printf("%7.2f\n",y); // 2 decimal places
printf("%9.2f\n",y);// right justified
printf("%-9.2f\n",y);// left justified
}
```

```
3456.120650
3456.12
  3456.12
3456.12
```

Formatted Input Function

- ✓ Input data can **be entered into the computer from a standard input device using** the 'C' Library function scanf().
- ✓ This function can be used to enter any combination of numerical vales, single characters and strings that has been arranged in a particular format.

Syntax: scanf("control string",arg1,arg2,....,argn);

- ✓ The control string specifies the field format in which the data is to be entered and the arguments, arg1,arg2….argn specify the individual input data items.
- ✓ Actually the arguments represent the address of location where the data is stored.
- ✓ Control strings and arguments are separated by commas.

Format specifier	Type of data item read
%c	Single character
%d	Signed decimal integer
%e	Floating point number
%f	Floating point number
%g	Floating point number
%h	Short integer
%o	Octal integer
%s	Character string
%u	Unsigned decimal integer
%x	Hexadecimal integer
%[ ]	Set of characters
% %	Percent sign

Program using gets() and puts()function	Output
#include <stdio.h> int main( ) { int num1, num2, sum; printf("Enter two integers: "); scanf("%d %d",&num1,&num2); sum=num1+num2; printf("Sum: %d",sum); return 0; }	Enter two integers: 12 11 Sum: 23

Reading Integer Input

Scanf statement	User Input	Resulted Assignment
scanf("%d",&a);	1234	1 2 3 4
scanf("%3d",&a);	1234	1 2 3
scanf("%5d",&a);	-1234	- 1 2 3 4
scanf("%3d",&a);	-1234	- 1 2

Reading String Input	Output
#include <stdio.h> #include <conio.h> void main() { char name[30]; scanf("%[^\n]",name); printf("\n%s",name); scanf("%[^t]",name); printf("\n%s",name); }	User Input : Neha Atul Godse Output: Neha Atul Godse User Input: Neha Atul Godse Output: Neha A

Storage Classes in C

- ✓ Storage class defines the scope (visibility) and lifetime of variables and/or functions declared within a C Program.
- ✓ The following storage classes are most of used in C Programming,
  - ❖ Automatic variables
  - ❖ External variables
  - ❖ Static variables
  - ❖ Register variables
- ✓ **Automatic variables**
  - ❖ ***By default, any variable declared inside a function without any storage class specification, is called as automatic variable.***
  - ❖ They are created when a function is **called and are destroyed automatically** when the function's execution is completed.
  - ❖ Automatic variables can also be called local variable because they are local to a function
    - **Scope**: Variable defined with auto storage class are local to the function block.

- **Default value**: Garbage value.
- **Lifetime**: Only within the function /method block.

Example for Automatic Storage Class	Output
#include<stdio.h> #include<conio.h> void increment(); void main() { increment(); increment(); increment(); getch(); } void increment() { auto int a=0; a=a+1; printf("a=%d",a); }	a=1 a=1 a=1

✓ **External Variables or Global variables**

- A variable that declared outside the function is a **Global variable**
- Global variables remain available throughout the program execution.
- By default, initial value of global variable is 0 (zero).
  - **Scope**: Global: everywhere in the program
  - **Default initial value:** 0 (zero)
  - **Lifetime**: accessed till the program completes.

Example for External Storage Class	Output
#include<stdio.h> int main( ) { int x = 10 ; extern int y; printf("The value of x is %d \n",x); printf("The value of y is %d",y); return 0; } int y=50;	The value of x is 10 The value of y is 50

✓ **Static variables**

- Static variable is initialized only once **and remains into existence till the end of the program.**

- A static variable can either be internal or external depending upon the place of declaration.
  - **Scope**: Local to the block in which the variable is defined.
  - **Default initial value**: 0 (zero).
  - **Lifetime:** Till the whole program completed.

Example for Static Storage Class	Output
#include<stdio.h> #include<conio.h> void increment(); void main() { increment(); increment(); increment(); getch(); } void increment() { static int a=0; a=a+1; printf("a=%d",a); }	1 2 3

✓ **Register Variable**

- 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 the **than normal variables** since they are stored in register memory rather than main memory.
  - **Scope**: Local to the function which it is declared.
  - **Default initial value**: Garbage value.
  - **Lifetime**: Till the end of the function/ method block.
- Syntax: register int number;

Pre-processor Directives

✓ "Preprocessor" is a process that reads **the source code and performs some operation** before it is passed on to the computer.
✓ Preprocessor directives are placed in the source program before **main function.**
✓ All the preprocessor directives begin with the symbol **#.**

- ✓ Rules for writing preprocessor directive
  - It should begin with # symbol.
  - Semicolon is not needed.
  - Only one directive can appear on a line.
- ✓ Types of preprocessor directive
  - **Macro substitution directive**
  - **File inclusion directive**
  - **Compiler control directive**
  - **Other directives**

Macro substitution directive

- ✓ Macro substitution without argument
  - **General format:**
    - #define macro_name replacement_string
  - **Example:**
    - #define PI 3.14
    - #define TRUE 1
- ✓ Macro substitution with argument
  - General format:
    - #define macro_name(parameter list) replacement_string

Example for Macro substitution with argument	Output
#include<stdio.h> #define SQU(x)(x*x) int main() { int x; float y; x = SQU(3); y = SQU(3.1); printf("\nSquare of Integer : %d",x); printf("\nSquare of Float : %f",y); return(0); }	Square of Integer : 9 Square of Float : 9.610000

File inclusion directive

- ✓ This directive is used to include the content of a file into source code of the program.
- ✓ The general form is
  - # include "file name" (or) #include <file name>
  - Where, #include- preprocessor directive

- File name- name of the file to be included into source code

✓ Example:

- #include<stdio.h>
- #include ”example.c”
- #include<math.h>

Compiler Control directive

✓ This directives are also called “conditional compilation”.

✓ A portion of source code may be compiled conditionally using the conditional compilation directive.

✓ The directives used in the conditional compilation are #ifdef, #endif, #if, #else, #ifndef.

Example program for #ifdef, #else and #endif in c:	Output
#include <stdio.h> #define RAJU 100 int main() { #ifdef RAJU printf("RAJU is defined. So, this line will be added in this C file\n"); #else printf("RAJU is not defined\n"); #endif return 0; }	RAJU is defined. So, this line will be added in this C file.

Other directives

✓ **Syntax:** #undef, #pragma, #error

✓ #undef is used to undefine a defined macro variable.

✓ #pragma is used to call a function before and after main function in a C Program.

C Compilation Process

- C is a compiled language.
- There are four steps in creating a C Program
  - Editing

- Compiling
- Linking
- Executing

Editing

- ✓ The programming process starts with creating a source file that consists of the statements of the program written in C Language.
- ✓ Editor is specially designed for writing the source code of C programs.
- ✓ The source code is saved on the disk with an extension .c

Compiling

- ✓ The process of converting of the **source code into a machine code** is called compiling.
- ✓ The program that is used to convert source code into the machine code is called compiler.
- ✓ It translates the source code into the **object code.**
- ✓ Object code is the intermediate form of the program.
- ✓ The file has an extension **.obj**
- ✓ Before creating object code compiler scans the source code for error.
- ✓ All the **errors must be removed** from the source code before creating the object code of the program.

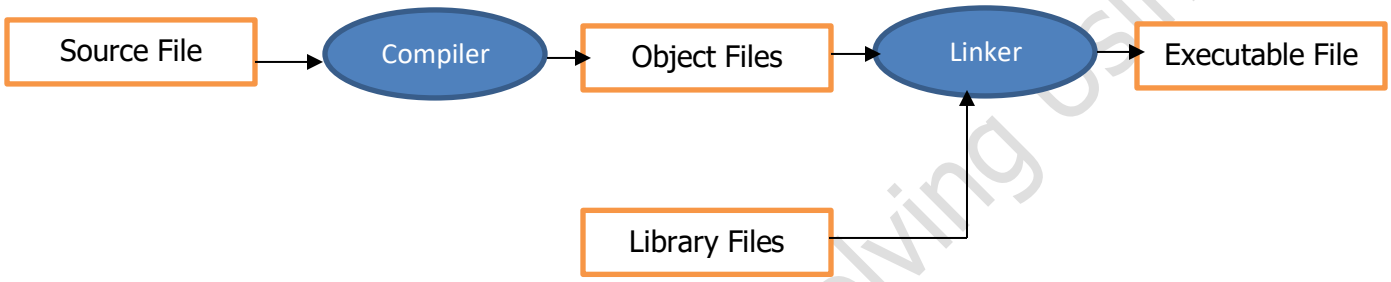
Linking

- ✓ A C Program may contain **predefined routines and functions.**
- ✓ These functions are **contained in separate files.**
- ✓ These files are part of the C Compiler and are called **library files or runtime routines.**
- ✓ In this step necessary libraries are linked to the object code. After linking libraries, the executable file of the program is created.
- ✓ If the source code uses a **library function that does not exist, the linker generates an error.**
- ✓ **If there are errors, the linker does not create the executable file.**
- ✓ The executable file is created with the **.exe extension.**

- ✓ This is directly run on the computer system.

Executing

- ✓ In this step program actually **run on the computer system**.
- ✓ For example, in Windows OS, when the name of an executable file is double clicked, the system loader the file into the computer memory and executes it.



C Compilation Process

Evaluating Expressions in C

- ✓ An expression is a combination of constants, variables and operators.
- ✓ To evaluate a complex expression that has several operators, certain rules have to be followed.
- ✓ These are called hierarchy rules or precedence rules.

Operator precedence

- ✓ Consider the expression, a + b * c
- ✓ Among the two operators, * and +, * has higher precedence.
- ✓ Hence (b * c) will be evaluated first and then added with a.
- ✓ If sum of a and b is required to be evaluated first, then we can change the order of precedence by putting parenthesis as (a + b) * c, since parentheses have highest precedence.

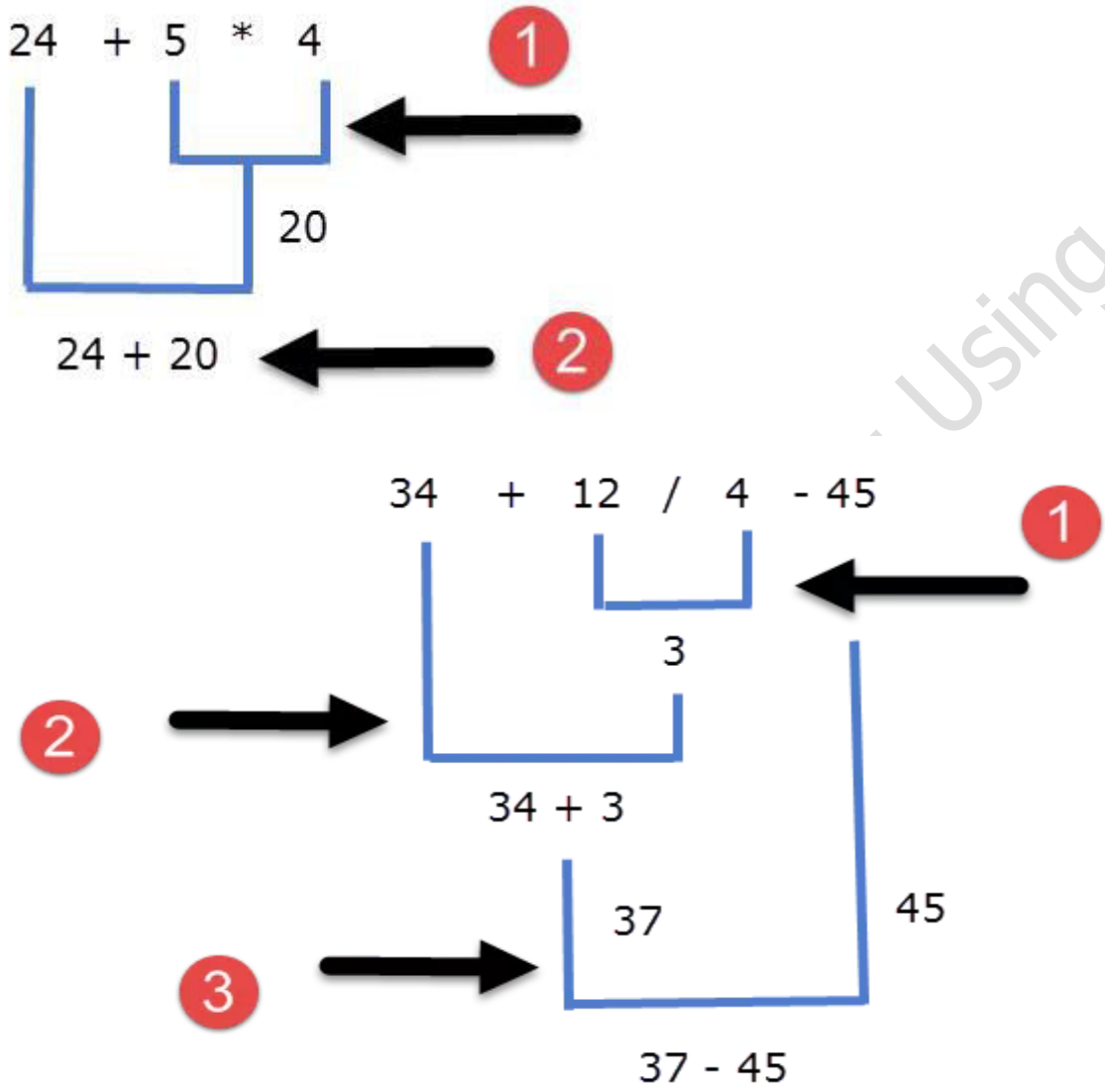
Associativity

- ✓ Associativity defines the direction in which the operator having the same precedence acts on the operands. It can be either left-to-right or right-to-left.

Sri Eshwar College of Engineering
Coimbatore- 641 202

Precedence level	Operators	Associativity
1	( ) (Function) [ ] (Array) . (Dot) ➔ (Arrow)	Left to Right
2	Unary arithmetic operators (+, -, ++, --) Logical NOT (!) One's complement (~) Address of (&) Indirection(*) Typecast operator sizeof	Right to Left
3	Binary arithmetic operators (*, /, %)	Left to Right
4	Binary arithmetic operators (+, -)	Left to Right
5	Shift operators (>>, <<)	Left to Right
6	Relational operators (>, < , >=, <=)	Left to Right
7	Relational operators (==, !=)	Left to Right
8	Bitwise AND (&)	Left to Right
9	Bitwise exclusive OR (^)	Left to Right
10	Bitwise OR (|)	Left to Right
11	Logical AND (&&)	Left to Right
12	Logical OR (||)	Left to Right
13	Conditional Operator (?:)	Right to Left
14	Assignment operators (Simple and compound) (=, +=, *=, /=, %=, >>=, <<=, &= ,^=, |=	Right to Left
15	Comma (,)	Left to Right

Example

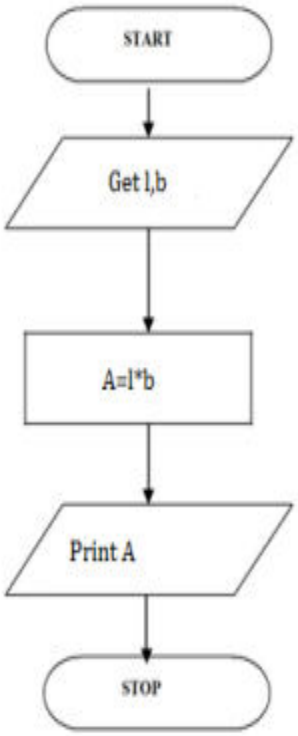


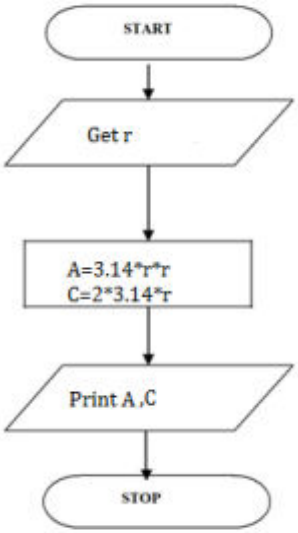
Examples

1. **x=3*4+5*6**
2. **x=3*(4+5)*6**
3. **x=3*4%5/2**
4. **x=3*(4%5)/2**
5. **x=3*((4%5)/2)**

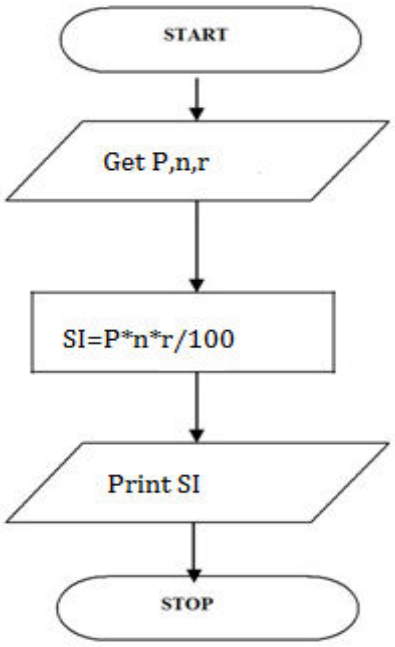
Sri Eshwar College of Engineering
Coimbatore- 641 202

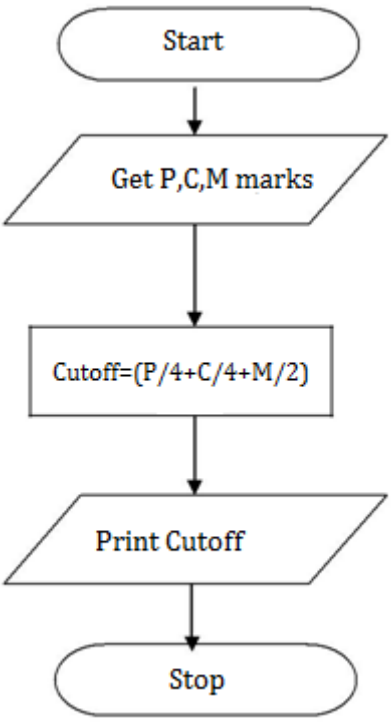
Example Algorithm, Pseudo code and Flow charts

Write an algorithm to find area of rectangle		
Algorithm	**Flow Chart**	**Pseudo code**
Step 1: Start Step 2: Get l,b values Step 3: Calculate A=l*b Step 4: Display A Step 5: Stop		BEGIN READ l,b CALCULATE A=l*b DISPLAY A END

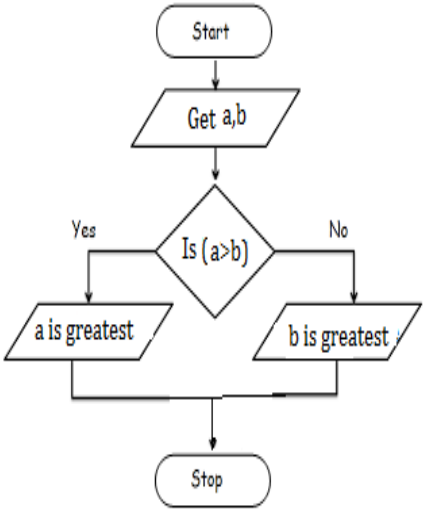
Write an algorithm for Calculating area and circumference of circle		
Algorithm	**Flow Chart**	**Pseudo code**
Step 1: Start Step 2: get r value Step 3: Calculate A=3.14*r*r Step 4: Calculate C=2.3.14*r Step 5: Display A,C Step 6: Stop		BEGIN READ r CALCULATE A and C A=3.14*r*r C=2*3.14*r DISPLAY A END

Sri Eshwar College of Engineering
Coimbatore- 641 202

Write an algorithm for Calculating simple interest		
Algorithm	**Flow Chart**	**Pseudo code**
Step 1: Start Step 2: get P, n, r value Step3:Calculate SI=(p*n*r)/100 Step 4: Display S Step 5: Stop		BEGIN READ P, n, r CALCULATE S SI=(p*n*r)/100 DISPLAY SI END

Write an algorithm for Calculating engineering cutoff		
Algorithm	**Flow chart**	**Pseudo code**
Step 1: Start Step2: get P,C,M value Step3:calculate Cutoff= (P/4+C/4+M/2) Step 4: Display Cutoff Step 5: Stop		BEGIN READ P,C,M CALCULATE Cutoff= (P/4+C/4+M/2) DISPLAY Cutoff END

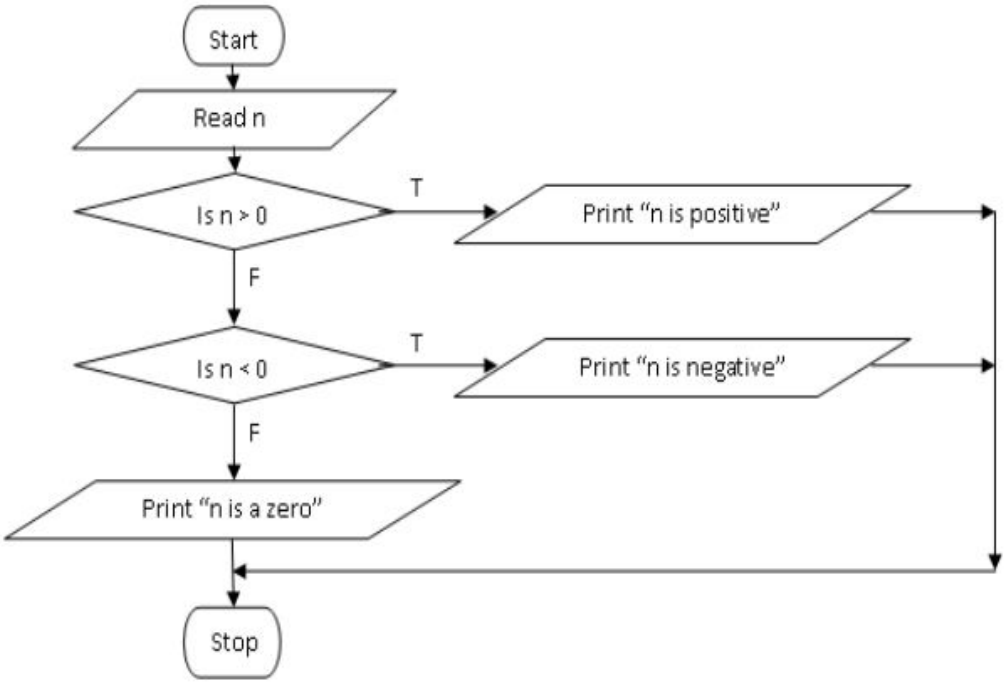
Sri Eshwar College of Engineering
Coimbatore- 641 202

To check greatest of two numbers		
Algorithm	**Flow chart**	**Flow Chart**
Step 1: Start Step 2: get a,b value Step 3: check if(a>b) print a is greater Step 4: else b is greater Step 5: Stop		BEGIN READ a,b IF (a>b) THEN DISPLAY a is greater ELSE DISPLAY b is greater END IF END

Write an algorithm to check whether given number is +ve, -ve or zero.	
Algorithm	**Pseudo code**
Step 1: Start Step 2: Get n value. Step 3: if (n ==0) print “Given number is Zero” Else goto step4 Step 4: if (n > 0) then Print “Given number is +ve” Step 5: else Print “Given number is -ve” Step 6: Stop	BEGIN GET n IF(n==0) THEN DISPLAY “ n is zero” ELSE IF(n>0) THEN DISPLAY “n is positive” ELSE DISPLAY “n is negative” END IF END IF END

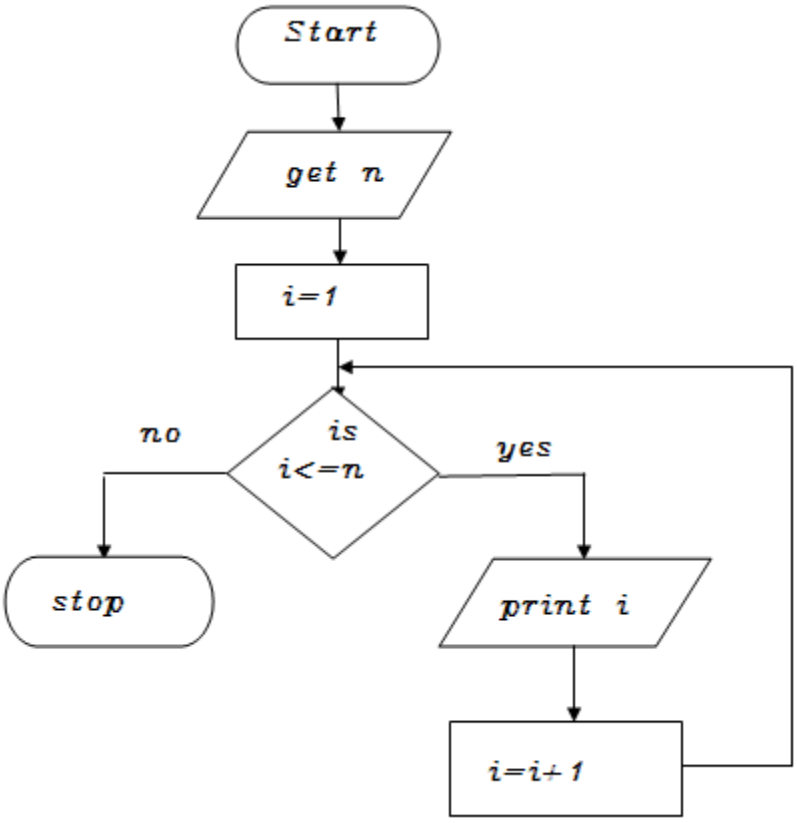
Flow chart

Write an algorithm to check whether given number is +ve, -ve or zero.



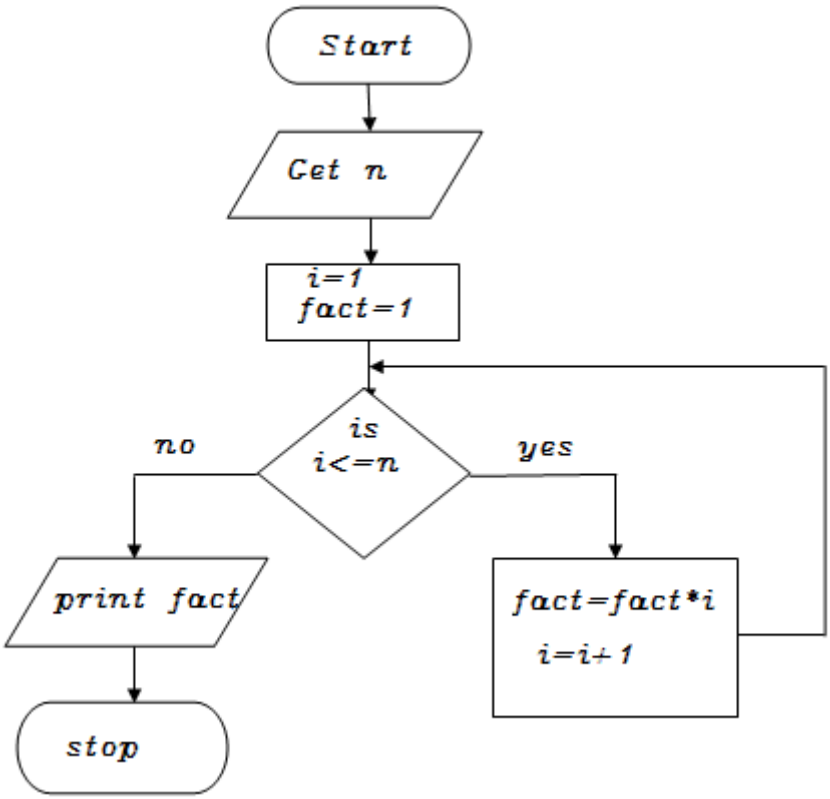
Write an algorithm to print all natural numbers up to n.	
Algorithm	**Pseudo code**
Step 1: Start Step 2: Get n value. Step 3: Initialize i=1 Step 4: if (i<=n) go to step 5 else go to step 8 Step 5: Print i value step 6 : Increment i value by 1 Step 7: go to step 4 Step 8: Stop	BEGIN GET n INITIALIZE i=1 WHILE(i<=n) DO PRINT i i=i+1 ENDWHILE END

Sri Eshwar College of Engineering
Coimbatore- 641 202



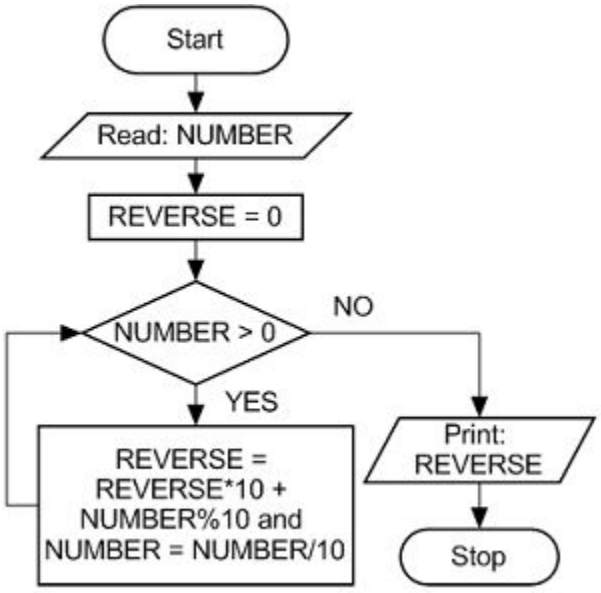
Write an algorithm to factorial of a given number	
Algorithm	**Pseudo code**
Step 1: start step 2: get n value step 3: set initial value i=1, fact=1 Step 4: Check i value if(i<=n) goto step 5 else goto step8 step 5: calculate fact=fact*i step 6: increment i value by 1 step 7: goto step 4 step 8: Print fact value step 9: Stop	BEGIN GET n INITIALIZE i=1,fact=1 WHILE(i<=n) DO fact=fact*i i=i+1 ENDWHILE PRINT fact END

Sri Eshwar College of Engineering
Coimbatore- 641 202



Write an algorithm to print reverse of given number	
Algorithm	**Pseudo code**
Step 1: Start step 2: Initialize r=0, sum=0 step 3: Read the value of n Step 4: If n>0 do the following else goto step 6 Step 4.1 : r= n mod 10 Step 4.2 : sum=sum*10+r Step 4.3 : n=n/10 Step 5: goto step 4 Step 6: Print sum Step 7: Stop	BEGIN INITIALIZE sum=0, r=0 READ n WHILE(n>0) r=n%10 sum=sum*10+r n=n/10 ENDWHILE WRITE sum STOP

Sri Eshwar College of Engineering
Coimbatore- 641 202



Write a program to swap two numbers without using a temporary variable	**Output**
#include <stdio.h> int main() { int num1,num2; printf("\n Enter the first number: "); scanf("%d",&num1); printf("\n Enter the second number: "); scanf("%d",&num2); num1= num1+num2; num2=num1-num2; num1=num1-num2; printf("\n The first number is %d",num1); printf("\n The first number is %d",num2); return 0; }	Enter the first number: 3 Enter the second number: 5 The first number is 5 The second number is 3
Write a program to swap two numbers using a temporary variable	**Output**
#include <stdio.h> int main() { int num1,num2; printf("\n Enter the first number: ");	Enter the first number: 3 Enter the second number: 5 The first number is 5

scanf("%d",&num1); printf("\n Enter the second number: "); scanf("%d",&num2); int temp=num1; num1=num2; num2=temp; printf("\n The first number is %d",num1); printf("\n The second number is %d",num2); return 0; }	The second number is 3
Write a program to print the ASCII value of a character	**Output**
#include <stdio.h> int main() { char ch; printf("Enter any character:"); scanf("%c",&ch); printf("\n The ASCII value of %c is:%d",ch,ch); return 0; }	Enter any character: A The ASCII value of A is:65
Write a program to convert degrees Fahrenheit into degrees Celsius	**Output**
#include <stdio.h> int main() { float fahrenheit,celsius; printf("\n enter the temperature in fahrenheit: "); scanf("%f",&fahrenheit); celsius=(0.56)*(fahrenheit-32); printf("\n temperature in degree celsius=%f",celsius); return 0; }	enter the temperature in fahrenheit: 32 temperature in degree celsius=0
Write a program to read a character in upper case and then print it in lower case	**Output**
#include <stdio.h> int main() { char ch; printf("Enter any character in upper case:"); scanf("%c",&ch);	Enter any character in upper case: A The character in lower case is: a

printf("\n The character in lower case is: %c",ch+32); return 0; }	
Write a program to calculate the area of a circle	**Output**
#include <stdio.h> int main() { float radius; double area, circumference; printf("\n Enter the radiud of the circle: "); scanf("%f",&radius); area=3.14*radius*radius; circumference=2*3.14*radius; printf("Area = %.2le",area); printf("\n circumference= %.2e", circumference); return 0; }	Enter the radiud of the circle: 7 Area = 1.54e+02 circumference= 4.40e+01
Example for Enumerated Constants	**Output**
#include<stdio.h> int main() { enum Day{sun,mon,tue,wed,thu,fri,sat}; enum Day d=wed; printf("Day : %d\n",d); return 0; }	Day : 3

Sri Eshwar College of Engineering
Coimbatore- 641 202

Unit II

CONDITIONAL STATEMENTS AND LOOPING CONSTRUCTS

Problem solving using Conditional or Selection or Branching Statements: Structure of if, if-else, else-if ladder, nested-if, switch constructs - Looping constructs: Structure of for, while, do-while constructs, usage of break, return, goto and continue keywords

Conditional Branching Statements

- The conditional branching statements help to **jump from one part of the program to another** depending on whether a particular condition is satisfied or not.
- The decision control statements include
  - Simple if
  - if-else
  - if-else-if-ladder
  - nested-if
  - switch case

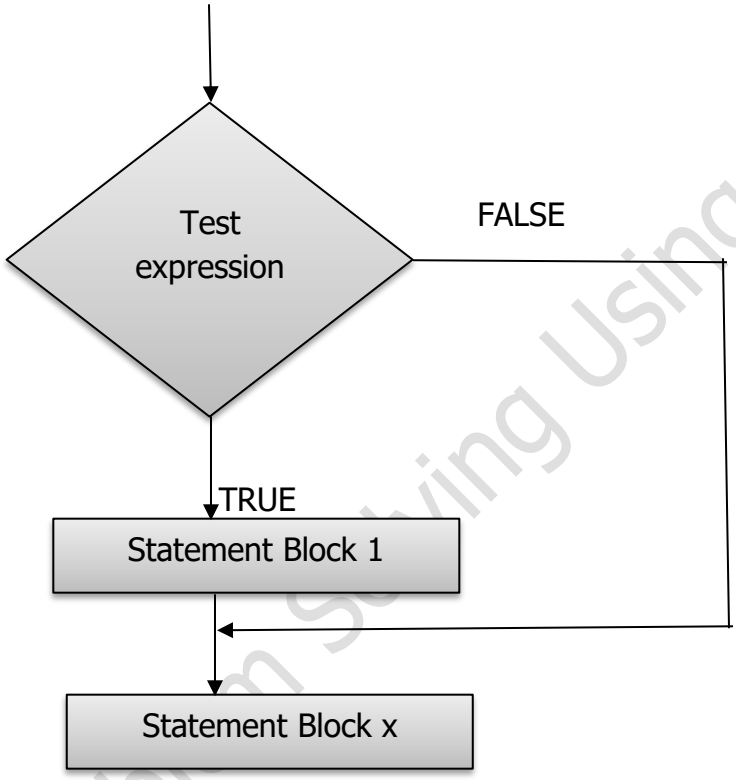
Simple if statement

- The if statement is the **simplest form of selection statement** that frequently used in decision making.
- The syntax for simple if statement is,

```
if (condition)
{
statement 1;
---------------
statement n;
}
statement x;
```

- The if block may include one statement or n statements enclosed within curly braces.
- First the test expression is evaluated.

- If the test expression is true, the statement of if block (statement 1 to n) are executed otherwise these statements will be skipped and the execution continues from the next statement.



Write a program to determine whether a person is eligible to vote.	**Output**
#include <stdio.h> int main() { int age; printf("Enter the age: "); scanf("%d",&age); if(age>=18) printf("\n You are eligible to vote"); return 0; }	Enter the age: 28 You are eligible to vote
Program to find the given number is divisible by 2	**Output**
#include <stdio.h> int main() { int n;	Enter the number:10 The given number 10 is divisible by 2

```
  printf("Enter the number:");
  scanf("%d",&n);
  if(n%2==0)
  {
     printf("\n The given number %d is
divisible by 2",n);
  }
  return 0;
}
```

Write a program to use curly brace in the if block. Enter only three numbers and calculate their sum and multiplication	**Output**

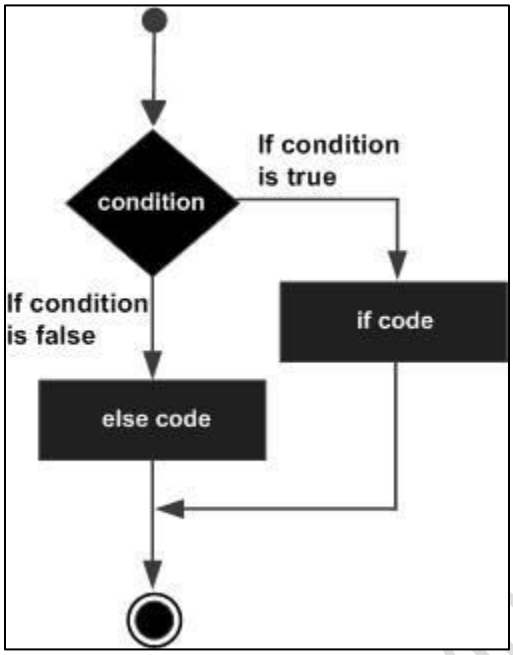
```
#include <stdio.h>
int main()
{
    int a,b,c,x;
    printf("\n Enter Three Numbers:");
    x=scanf("%d %d %d",&a,&b,&c);
    if(x==3)
    {
        printf("\n Addition    :%d",a+b+c);
        printf("\n Multiplication    :%d",a*b*c);

    }
    return 0;
}
```

Test Case 1:
Enter Three Numbers:1 2 4
Addition :7
Multiplication :8
Test Case 2:
Enter Three Numbers:5 v 8
In second test case the **numbers are not correctly** entered. Hence, the if condition is false and operation is performed

THE if-else statement

- We observed the execution of if statement in the previous programs.
- It is observed that if statement executed only when the condition following if is true. It does nothing when the condition is false.
- The if-else statement take care of true as well as false conditions.
- It has two blocks.
- One block is for if and it is executed when the condition is true.
- The other block is for else block and it is executed when the condition is false.
- The else statement cannot be used without if.

Sri Eshwar College of Engineering
Coimbatore- 641 202



Write a program to find whether the given number is odd or even	**Output**
#include<stdio.h> int main() { int n; printf("Enter any number: \n"); scanf("%d",&n); if(n%2==0){ printf("%d is an even number",n); } else{ printf("%d is an odd number",n); } return 0; }	Enter any number: 11 11 is an odd number Enter any number: 10 10 is an even number
Write a program to find whether a given year is a leap year or not	**Output**
#include<stdio.h> int main() { int year; printf("Enter any year: \n"); scanf("%d",&year); if(((year%4==0)&&(year%100!=0))||(year%400==0))	Enter any year: 1996 1996 is a leap year

```
    {
        printf("%d is a leap year",year);
    }
    else{
        printf("%d is not a leap year",year);
    }
    return 0;
}
```

Write a program to enter a character and then determine whether it is vowel or not

Output

```
#include<stdio.h>
int main()
{
   char ch;
   printf("Enter any character: ");
   scanf("%c",&ch);

if(ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u'||ch=='A'||ch=='E'||ch=='O'
||ch=='U')
   printf("\n %c is a VOWEL.",ch);
   else
   printf("\n %c is not a VOWEL.",ch);
      return 0;
}
```

```
Enter any character: A

 A is a VOWEL.
```

Write a program to enter any character. If the entered character is in lower case then convert it into upper case and if it is a lower case character then convert it into uppercase

Output

```
#include<stdio.h>
int main()
{
   char ch;
   printf("Enter any character: ");
   scanf("%c",&ch);
   if(ch>='A'&&ch<='Z')
   printf("\n The entered character was in upper case. In Lower case
it is %c",(ch+32));
   else
    printf("\n The entered character was in lower case. In Upper case
it is %c",(ch-32));
       return 0;
}
```

```
Enter any character: a

 The entered character was in lower case. In Upper case it is A
```

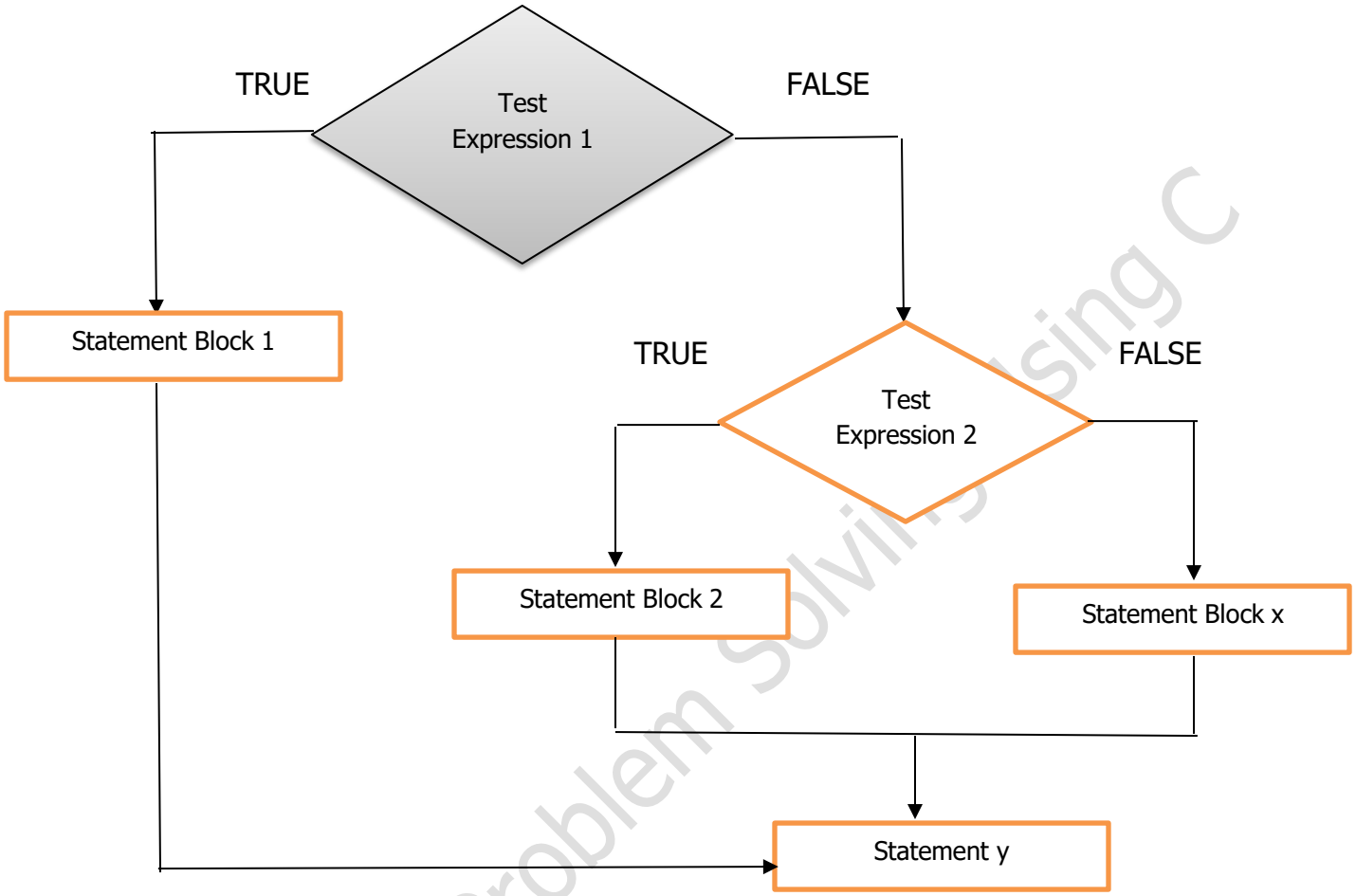
THE if-else-if statement

- Sometimes we wish to make a multi-way decision based on several conditions.
- The most general way of doing this is by using the else if variant on the if statement.
- This works by cascading several comparisons.
- As soon as one of these gives a true result, the following statement or block is executed and no further comparisons are performed.
- If none of the condition is true, the final else is executed.
- The final else statement is optional.

Syntax

```
if (expression 1)
{
   block of statements 1;
}
else if(test expression 2)
{
block of statements 2;
}
.....................
else {
statement block x;
}
Statement y;
```

Sri Eshwar College of Engineering
Coimbatore- 641 202



Write a program to accept roll no, name and total mark obtained by a student and assign grades according to the following conditions, display the roll number, name, total mark and grade:

Total Mark	Grade
>=90	A
>=80 and <90	B
>=70 and <80	C
>=60 and < 70	D
>=50 and <60	E
<50	Fail

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

Output

```
Enter the roll number of student
1
Enter the name of student
Bala
Enter the total mark of student
95
Grade details
1 Bala 95 A
```

```
{
    int roll_no,total_marks;
    char name[100];
    printf("Enter the roll number of student\n");
    scanf("%d",&roll_no);
    printf("Enter the name of student\n");
    scanf("%s",name);
    printf("Enter the total mark of student\n");
    scanf("%d",&total_marks);
    printf("Grade details\n");
    printf("%d %s %d ",roll_no,name,total_marks);
    if(total_marks>=90)
    printf("A");
    else if(total_marks>=80 && total_marks<90)
    printf("B");
    else if(total_marks>=70 && total_marks<80)
    printf("C");
    else if(total_marks>=60 && total_marks<70)
    printf("D");
    else if(total_marks>=50 && total_marks<60)
    printf("E");
    else
    printf("Fail");
    printf("\n");
return 0;
}
```

Write a program to calculate energy bill. Read the starting and ending meter reading. The charges are as follows.

No. of Units Consumed	Rate in (Rs.)
200-500	3.50
100-200	2.50
Less than 100	1.50

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

   int initial,final,consumed;
   float total;
   printf("Initial and final readings:");
   scanf("%d%d",&initial,&final);
   consumed=final-initial;
   if(consumed>=200 && consumed <=500)
   total=consumed*3.50;
   else if(consumed>=100 && consumed <=199)
```

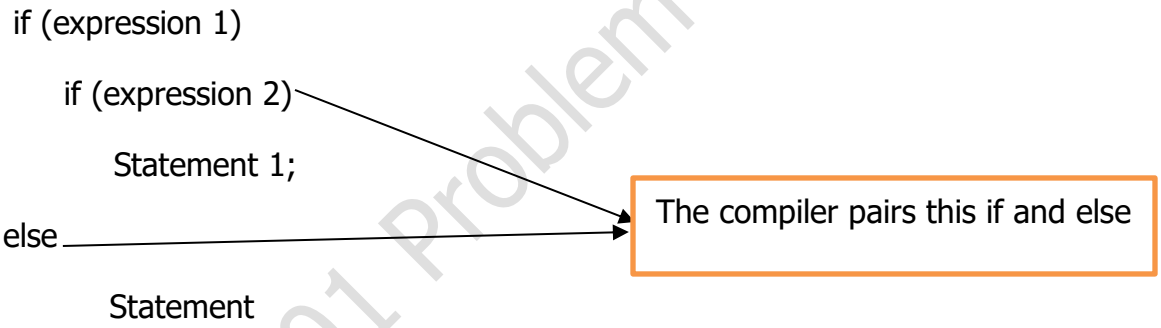
Output

Initial and final readings:800 850
Total Bill for 50 unit is 75.000000

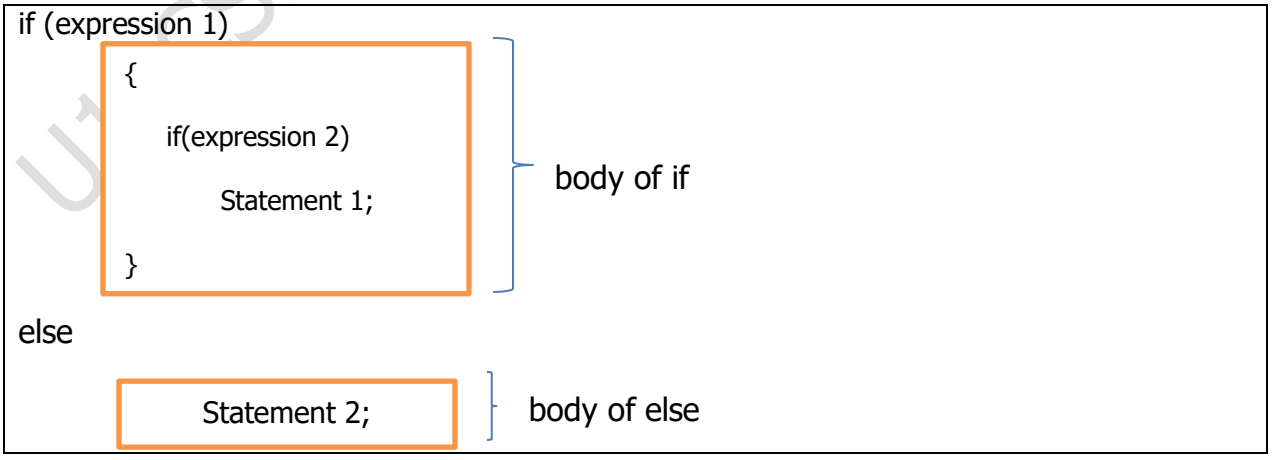
```
   total=consumed*2.50;
    else if(consumed<100)
   total=consumed*1.50;
printf("Total Bill for %d unit is %f",consumed,total);
return 0;
}
```

Dangling Else Problem

- Once we start nesting if-else statements, we may encounter a classic problem known as the dangling else.
- This problem is created when there is no matching else for every if.
- The solution to this problem is to follow the simple rule: *Always pair an else to the most recent unpaired if in the current block*
- Consider the example shown below from the code alignment, we conclude that the programmer intended the else statement to be paired with the first if.
- However, the compiler will pair it with the second if.



- The below table shows the solution to the dangling else.



Nested if statement

- In the if-else statement, if body of either if or else or both include another if-else statement, the statement is known as nested if.
- Nested ifs are very commonly used in programming.
- **Syntax:**

Nested if syntax

```
if(expression 1)
{
        if(expression 2)
                statement;
        else
                statement
}
else
      statement
```

Biggest of three numbers using nested if statement	**Output**
#include <stdio.h> int main() { int a,b,c; printf("Enter 3 numbers..."); scanf("%d%d%d",&a,&b,&c); if(a>b) { if(a>c) { printf(" %d is greatest",a); } else { printf("%d is greatest",c); } }	Enter 3 numbers...90 45 12 90 is greatest

```
else
   {
      if(b>c)
      {
         printf("%d is greatest",b);
      }
      else
      {
         printf("%d is greatest",c);
      }
   }
    return 0;
}
```

Switch case

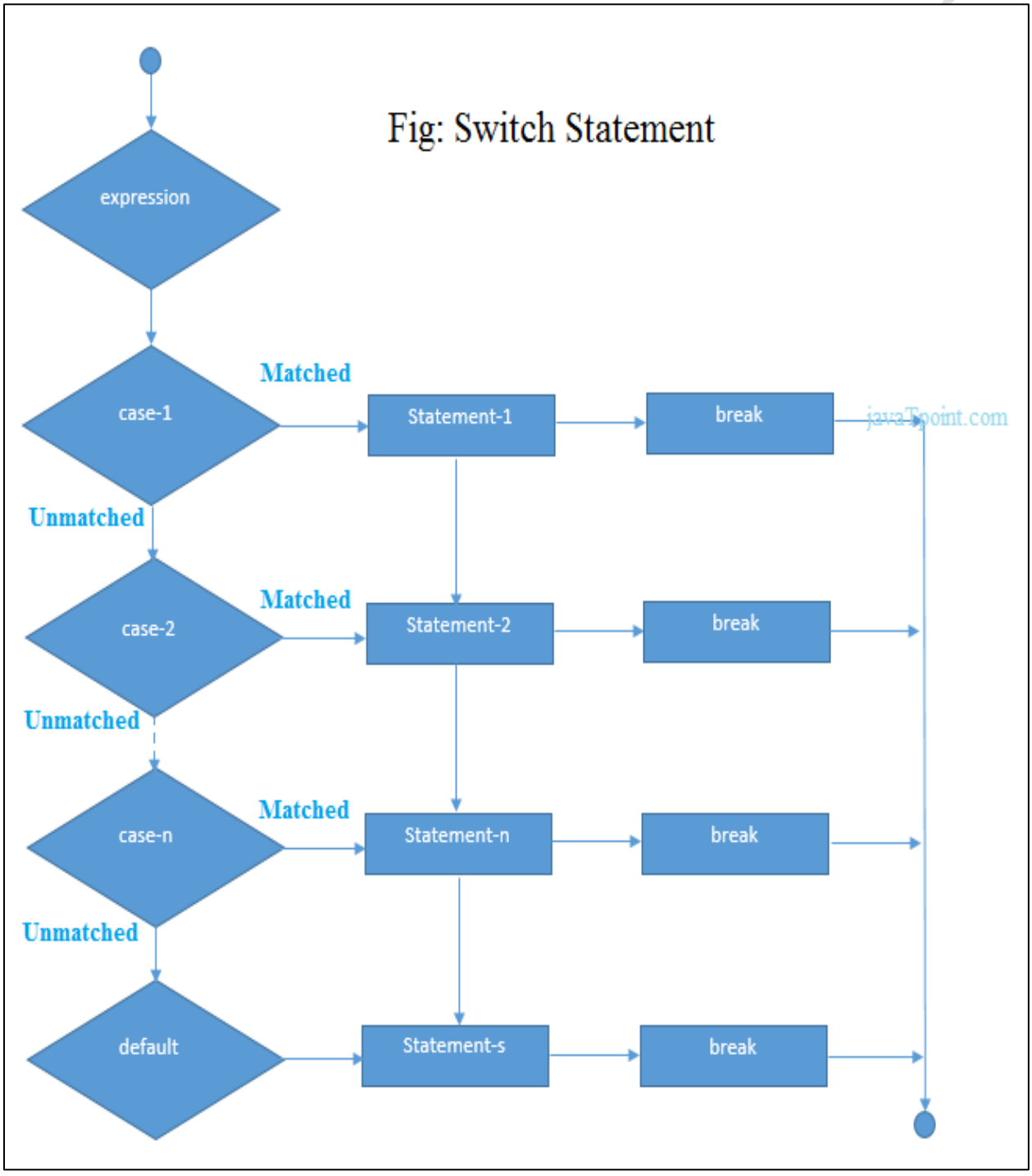
- The switch() case statement is like the if statement that allows us to make a decision from a number of choices.
- The switch statement requires only one argument of any data type, which is checked with a number of case options.
- The switch statement evaluates the expression and then looks for its value among case constants.
- If the value matches with a case constant, this particular case statement is executed.
- If not, the default is executed.

Syntax:

```
switch <expr>
{
 case constant_1:
  {
   statements 1;
   break;
  }
 case constant_2:
  {
  statements 2;
  break;
  }
```

```
  default:
   {
    default statements;
    }
}
```

Flow Chart for Switch statement



Facts Related to Switch

1. Duplicate cases are not allowed

Program	Error
#include <stdio.h> int main() { int x=1; switch(x) { case 1:printf("x is 1"); break; case 1:printf("x is 1"); break; case 2:printf("x is 2"); break; } return 0; }	main.c: In function 'main': main.c:18:9: error: duplicate case value case 1:printf("x is 1"); ^~~~ main.c:16:9: error: previously used here case 1:printf("x is 1"); ^~~~

2. **Only those expressions are allowed in switch which results in an integral constant value**

Allowed	Not Allowed
#include <stdio.h> int main() { int a=1,b=2,c=3; switch(a+b*c) { case 1: printf("choice 1"); break; case 2:printf("choice 2"); break; default:printf("default"); break; } return 0; }	#include <stdio.h> int main() { int a=1.15,b=2.0,c=3.0; switch(a+b*c) { case 1: printf("choice 1"); break; case 2:printf("choice 2"); break; default:printf("default"); break; } return 0; }

Output: default	**Error:** switch quantity not an integer

3. **Float value is not allowed as a constant value in case label. Only integer constants/ constant expressions are allowed in case label**

Allowed	Not Allowed
#include <stdio.h> int main() { int x=23; switch(x) { case 3+3: printf("choice 1"); break; case 3+4*5:printf("choice 2"); break; default:printf("default"); break; } return 0; } **Output:** choice 2	#include <stdio.h> int main() { float x=3.14; switch(x) { case 3.14: printf("choice 1"); break; case 2.1:printf("choice 2"); break; default:printf("default"); break; } return 0; } **Error:** case label does not reduce to an integer constant

4. **Variable expressions are not allowed in case labels. Although macros are allowed**

Allowed	Not Allowed
#include <stdio.h> #define y 2 #define z 23 int main() { int x=2; switch(x) { case y:printf("Number is 2"); break;	#include <stdio.h> int main() { int x=2,y=2,z=23; switch(x) { case y:printf("Number is 2"); break; case z:printf("Number is 23");

```
        case z:printf("Number is 23");
        break;
        default:printf("default case");
        break;
    }

    return 0;
}
```
Output: Number is 2

```
        break;
        default:printf("default case");
        break;
    }

    return 0;
}
```
Error:
case label does not reduce to an integer constant

5. **Default can be placed anywhere inside switch.**

Example program using switch statement

Menu Driven Calculator

```
#include<stdio.h>
int main()
{
int a,b,op;
printf("Enter the first value :\n");
scanf("%d",&a);
printf("Enter the second value :\n");
scanf("%d",&b);
printf("Enter the choice from the menu\n");
printf("1.Addition\n2.Subtraction\n3.Multiplication\n4.Division\n");
scanf("%d",&op);
switch(op)
{
    case 1:
    printf("The value after Addition is %d.",a+b);
    break;
    case 2:
    printf("The value after Subtraction is %d.",a-b);
    break;
    case 3:
    printf("The value after Multiplication is %d.",a*b);
    break;
    case 4:
    printf("The value after Division is %d.",a/b);
    break;
}
```

Output

```
 Enter the first value :
12
Enter the second value :
12
Enter the choice from the menu
1.Addition
2.Subtraction
3.Multiplication
4.Division
4
The value after Division is 1.
```

```
        return 0;
}
```

Program to list the numbers between 0 and 4 using switch case

```
#include <stdio.h>
int main()
{
    int n;
    printf("Enter the numbers between 0 and 4:");
    scanf("%d",&n);
    switch(n)
    {
        case 0:
        {
            printf("\n The entered number =%d",n);
            break;
        }
        case 1:
        {
            printf("\n The entered number =%d",n);
            break;
        }

       case 2:
        {
            printf("\n The entered number =%d",n);
            break;
        }
        case 3:
        {
            printf("\n The entered number =%d",n);
            break;
        }
        case 4:
        {
            printf("\n The entered number =%d",n);
            break;
        }
        default:
        {
            printf("\n Invalid Input");
            break;
        }
```

Output

```
Enter the numbers between 0 and 4:3

 The entered number =3
```

```
    }
}
```

Iterative Statements

- Iterative statements are used to repeat the execution of a list of statements, depending on the value of an integer expression.
- C language supports three types of iterative statements also known as looping statements. They are
  - while loop
  - do-while loop
  - for loop

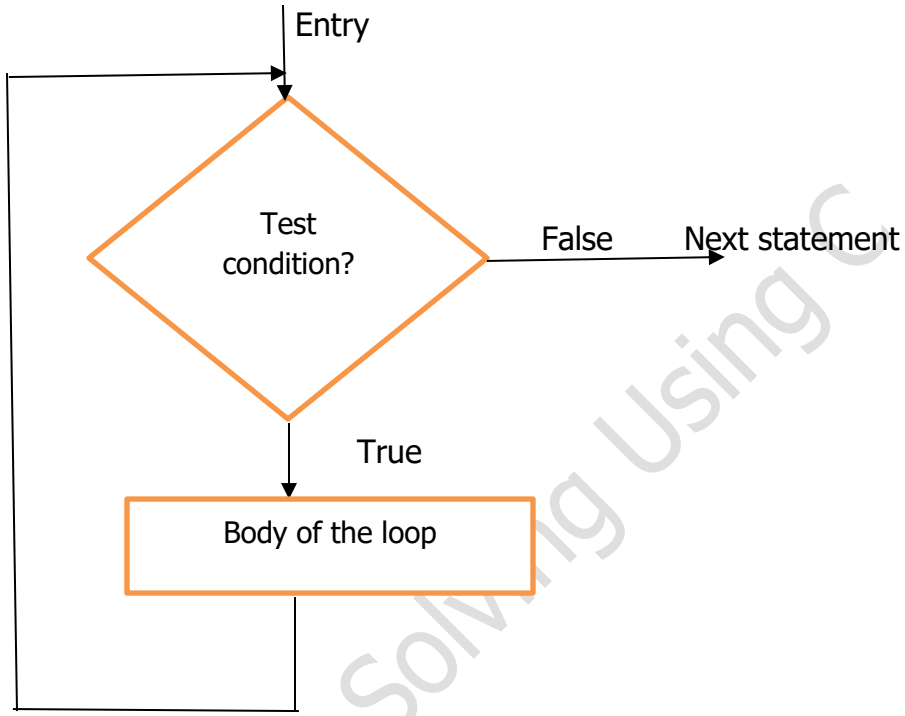
THE WHILE LOOP

- The *while* loop provides a mechanism to repeat one or more statements while a particular condition is true.
- It is an entry controlled loop.
- In the *while* loop, the condition is tested before any of the statements in the statement block is executed.
- If the control condition evaluates to false, then the statements enclosed in the loop are never executed.

Syntax:

```
while(condition)
{
Body of the loop
}
```

- In *while* loop the condition is tested first and it is true, then the body of the loop is executed.
- After the execution of the body, again the condition is tested, if it is true the body of the loop is again executed, otherwise it exit the body of loop.



C Program for Checking Armstrong Number	Output
#include<stdio.h> int main() { int n,r,c,sum=0; printf("Enter the number:\n"); scanf("%d",&n); int temp=n; while(n>0) { r=n%10; c=r*r*r; sum=sum+c; n=n/10; } if(temp==sum) printf("Armstrong Number"); else printf("Not An Armstrong Number"); return 0;	Enter the number: 153 Armstrong Number

}	
C Program to find the sum of individual digits of a number	Output
#include<stdio.h> int main() { int num,rem,sum=0; printf("Enter the value :\n"); scanf("%d",&num); int temp=num; while(num>0) { rem=num%10; sum=sum+rem; num=num/10; } printf("Sum of digits in %d is %d",temp,sum); return 0; }	Enter the value : 123 Sum of digits in 123 is 6
C Program to print multiplication table using while loop	Output
#include<stdio.h> int main() { int m,n,i=1; printf("Enter n\n"); scanf("%d",&n); printf("Enter m\n"); scanf("%d",&m); printf("The multiplication table of %d is\n",n); while(i<=m) { printf("%d*%d=%d\n",i,n,(i*n)); i++; } return 0; }	Enter n 5 Enter m 3 The multiplication table of 5 is 1*5=5 2*5=10 3*5=15
Reversing a Number	Output
#include<stdio.h> int main()	Enter the number: 123

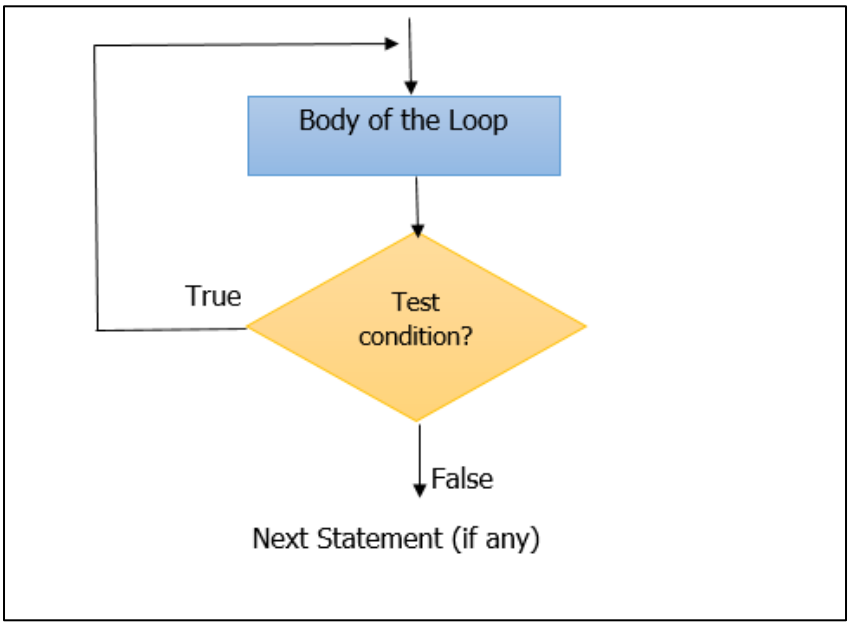
{ int n,r,c,sum=0; printf("Enter the number:\n"); scanf("%d",&n); int temp=n; while(n>0) { r=n%10; sum=sum*10+r; n=n/10; } printf("The reverse of a number %d is=%d\n",temp,sum); return 0; }	The reverse of a number 123 is=321

THE DO-WHILE LOOP

- The do-while loop is an exit controlled loop because the test condition is evaluated at the end of the loop.
- The do-while loop will execute at least one time even if the condition is false initially.
- Note that the test condition is enclosed in parenthesis and followed by a semicolon.

Syntax:

```
do{
statement/s;
}
while(condition);
```



Use the do-while loop and display a message "This is a program of do-while loop" for 5 times	Output
#include<stdio.h> int main() { int i=1; do{ printf("\n This is a program for do-while loop."); i++; } while(i<=5); return 0; }	This is a program for do-while loop. This is a program for do-while loop. This is a program for do-while loop. This is a program for do-while loop. This is a program for do-while loop.
Use the do-while loop and display a message "This is a program of do-while loop". Use the false condition, that is, the value of I should be initially larger than the value in the do-while loop.	Output
#include<stdio.h> int main() { int i=7; do{ printf("\n This is a program for do-while loop."); i++; } while(i<=5); return 0; }	This is a program for do-while loop.

THE FOR LOOP

- For loop is used to execute a set of statements repeatedly until a particular condition is satisfied.
- It is a pre-test loop and it is used when the number of iterations of the loop is known before the loop is entered.
- The head of the loop consists of three parts.
    - Initialization expression
    - Test expression

- Update expression separated by semicolon.

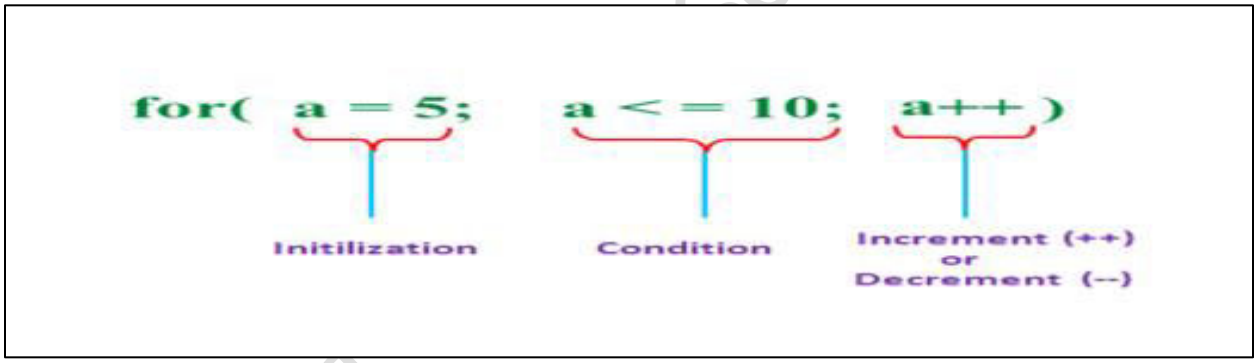
Execution of for loop

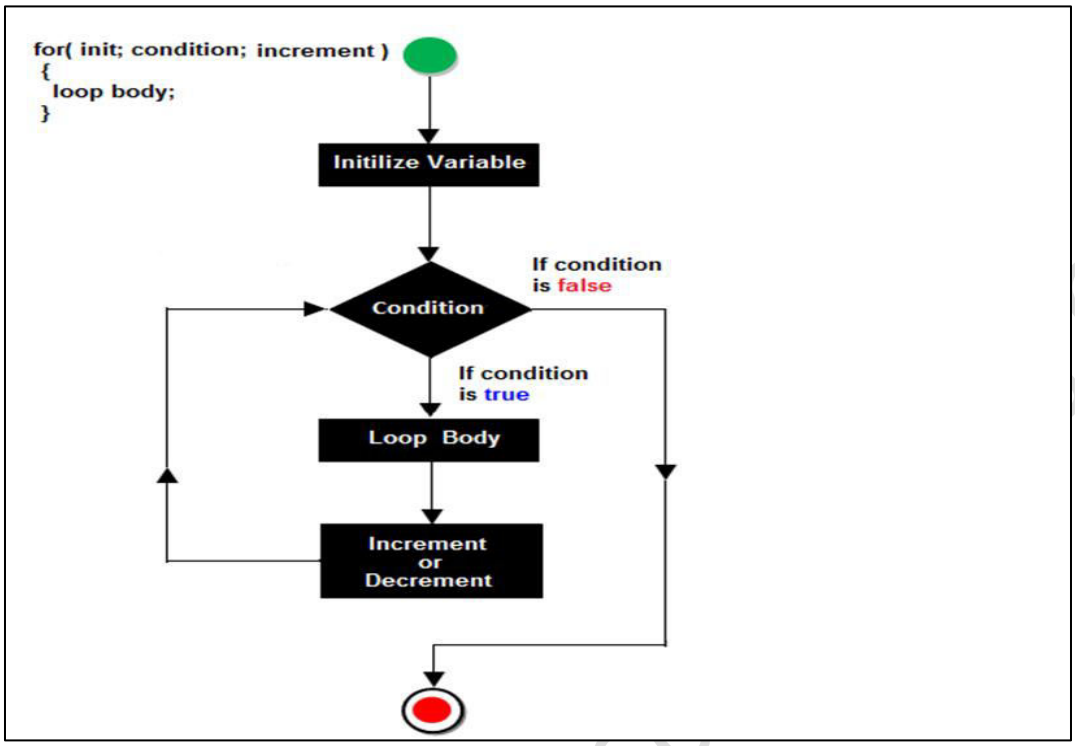
1. It first evaluates the initialization code.
2. Then it checks the condition expression.
3. If it is true, it executes the for-loop body.
4. Then it evaluates the increment/decrement condition and again follows from step 2.
5. When condition expression becomes false, it exits the loop.

Syntax:

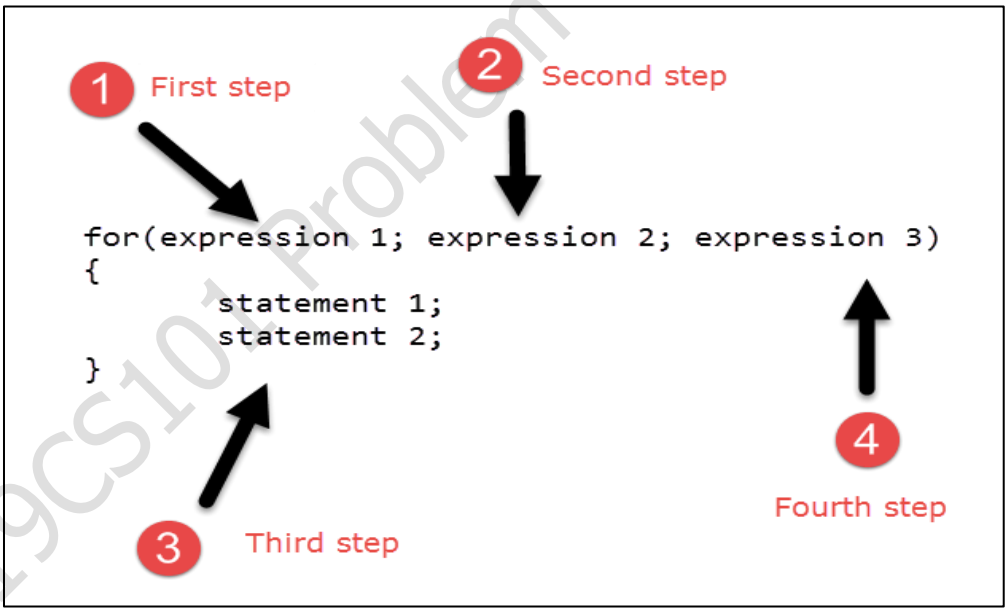
```
for(initialization expression; test expression; update expression)
{
Statement(s);
}
```

Example:





Control Flow of for loop



Different Flavors of for loop	
#include <stdio.h> int main() { int i; for(i=1; ;i++)	Infinite Loop

{ printf("%d\n",i); } return 0; }	
#include <stdio.h> int main() { int i=1; for(;i<=10;i++) { printf("%d\n",i); } }	1 2 3 4 5 6 7 8 9 10
#include <stdio.h> int main() { int i=1; for(;i<=10;) { printf("%d\n",i); i=i+1; } }	1 2 3 4 5 6 7 8 9 10

Example C Programs using for loop

C program to find the factorial of a given number	Output
#include <stdio.h> int main() { int n,fact=1,i; scanf("%d",&n); for(i=1;i<=n;i++) { fact=fact*i; } printf("The factorial of %d is %d",n,fact); return 0; }	5 The factorial of 5 is 120

Multiplication Table	Output
#include <stdio.h> int main() { int i,number; printf("Enter a number:"); scanf("%d",&number); for(i=1;i<=10;i++) { printf("%d\n",(number*i)); } return 0; }	Enter a number:2 2 4 6 8 10 12 14 16 18 20
Program to generate Fibonacci series	**Output**
#include <stdio.h> int main() { int i,n,a=0,b=1; printf("Enter the value of n:\n"); scanf("%d",&n); printf("%d%d",a,b); for(i=3;i<=n;i++) { int c=a+b; a=b; b=c; printf("%d",c); } return 0; }	Enter the value of n: 10 0 1 1 2 3 5 8 13 21 34
Sum of first N Numbers	**Output**
#include<stdio.h> int main() { int i,num,sum=0; scanf("%d",&num); for(i=0;i<=num;i++) { sum=sum+i; } printf("%d",sum); return 0;	10 55

}	
Printing First N odd Numbers	Output
#include<stdio.h> int main() { int i,num; scanf("%d",&num); for(i=1;i<=num;i=i+2) { printf("%d ",i); } return 0; }	5 1 3 5
Printing First N even Numbers	Output
#include<stdio.h> int main() { int i,num; scanf("%d",&num); for(i=2;i<=num;i=i+2) { printf("%d ",i); } return 0; }	10 2 4 6 8 10
Sum of first N Odd numbers	Output
#include<stdio.h> int main() { int i,num,sum=0; scanf("%d",&num); for(i=1;i<=num;i=i+2) { sum=sum+i; } printf("Sum = %d",sum); return 0; }	5 Sum = 9

Checking Prime Number	Output
#include<stdio.h> int main() { int n,i,flag=0; scanf("%d",&n); for(i=1;i<=n;i++) { if(n%i==0) { flag=flag+1; } } if(flag==2) printf("Prime"); else printf("Not prime"); return 0; }	13 Prime 33 Not Prime

Decimal to Binary Conversion	Output
#include<stdio.h> int main() { int n,i=0,b[32],j; printf("Enter the decimal number\n"); scanf("%d",&n); printf("The binary equivalent of decimal number %d is ",n); while(n>0) { b[i]=n%2; n=n/2; i++; } for(j=i-1;j>=0;j--) { printf("%d",b[j]); } return 0; }	Enter the decimal number 10 The binary equivalent of decimal number 10 is 1010

Nested for loops

- C allows its users to have nested loop, i.e., loops that can be placed inside other loops.
- Although this feature will work with any loop like while, do-while and for, it is most commonly used in for loop because this is easier to control.

Example for Nested for loops Pattern Printing Problems	
#include <stdio.h> int main() { for(int i=1;i<=5;i++) { for(int j=1;j<=5;j++) { printf("*"); } printf("\n"); } return 0; }	* *
#include <stdio.h> int main() { for(int i=1;i<=5;i++) { for(int j=1;j<=i;j++) { printf("*"); } printf("\n"); } return 0; }	* ** *** **** *****
#include <stdio.h> int main() { for(int i=1;i<=5;i++)	**1** **12** **123** **1234**

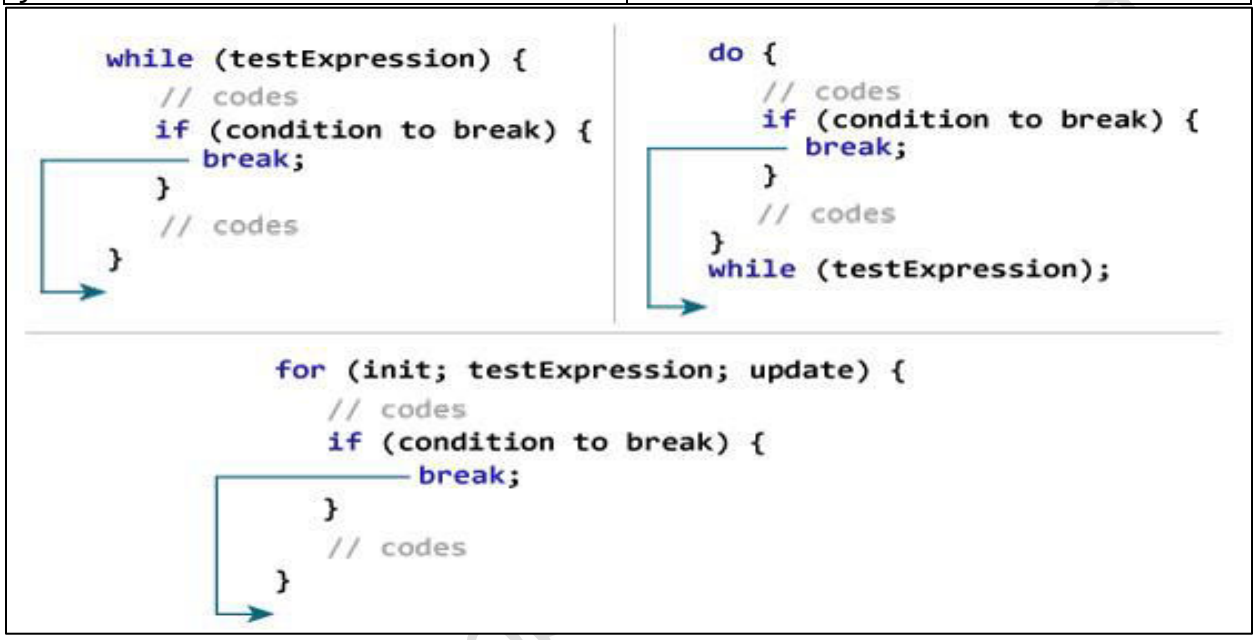
{ for(int j=1;j<=i;j++) { printf("%d",j); } printf("\n"); } return 0; }	**12345**
#include <stdio.h> int main() { for(int i=1;i<=5;i++) { for(int j=1;j<=i;j++) { printf("%d",i); } printf("\n"); } return 0; }	**1** **22** **333** **4444** **55555**

THE break STATEMENT

- The keyword **break allows the programmers to terminate the loop**.
- The **break skips from the loop or block** in which it is defined.
- The control automatically goes to **the first statement after the loop or block.**
- The break statement is widely used with for loop, while loop, and do while loop.
- In switch statement if the break statement is missing then every case from the matched case label till the end of the switch, including the default, is executed.

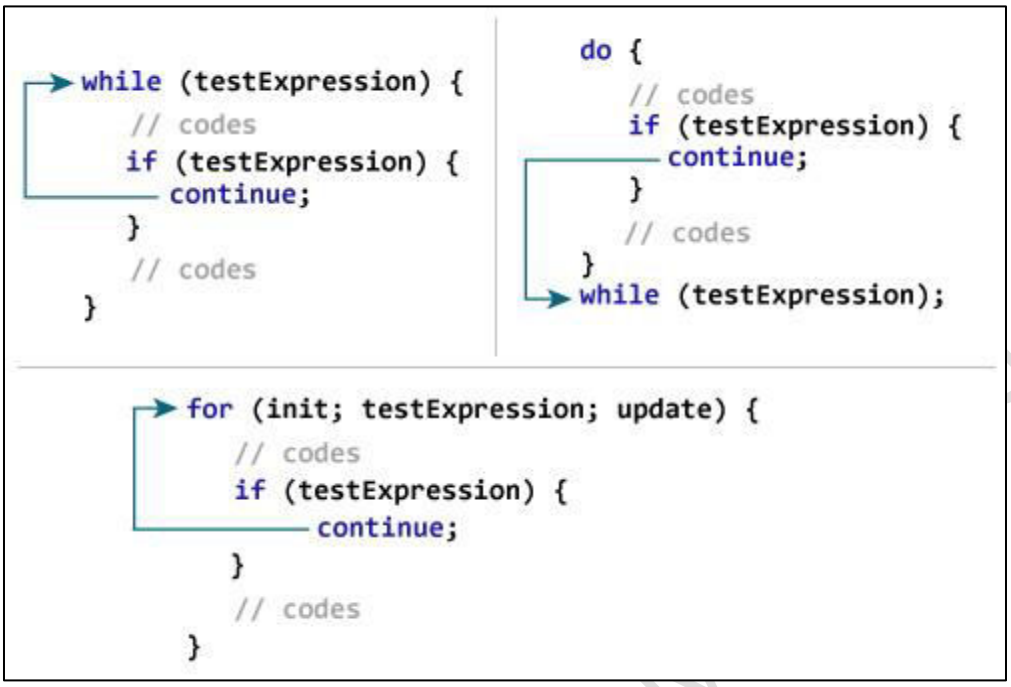
Example program for break statement	Output
#include <stdio.h> int main() { int i; for(i=1;i<=5;i++)	1 2

```
    {
        if(i==3)
        {
            break;
        }
        printf("%d\n",i);
    }
    return 0;
}
```



THE continue STATEMENT

- The *continue* statement is exactly opposite to *break.*
- The *continue* statement is used to continue next iteration of loop statement.
- When it occurs in the loop it does not terminate but it skip the statement after this statement.



Example program for continue statement	Output
#include <stdio.h> int main() { int i; for(i=1;i<=5;i++) { if(i==3) { continue; } printf("%d\n",i); } return 0; }	1 2 4 5

THE goto STATEMENT

- This statement does not require any condition.
- This statement passes control anywhere in the program, that is, control is transferred to another part of the program without testing any condition.
- The user has to define **goto** statement as follows.

goto label;

- Where, the label name must start with any character.
- Label is the position where the condition is to be transferred.

Even or Odd using goto statement	Output
#include <stdio.h> int main() { int x; printf("Enter a number..."); scanf("%d",&x); if(x%2==0) goto even; else goto odd; even: printf("%d is Even Number.",x); return; odd: printf("%d is Odd Number.",x); return 0; }	Enter a number…..4 4 is Even Number. Enter a number…..5 5 is Odd Number. **Explanation** In the above program, a number is entered. The number is checked for even or odd with module operator. When the number is even, the goto statement **transfers the control to the label even**. Similarly when the number is odd, the goto **statement transfers the control to the label odd and respective message will be displayed.**
Leap Year or Not using goto statement	**Output**
#include <stdio.h> int main() { int leap; printf("Enter a Year..."); scanf("%d",&leap); if(leap%4==0) goto leap; else goto noleap; leap: printf("%d is Leap Year.",leap); return; noleap: printf("%d is not a Leap Year.",leap); return 0; }	Enter a Year….2000 2000 is Leap Year.

Note: goto statement, break and continue statements are called as jump

Break	Continue
• Exits from current block or loop	• Loop takes next iteration
• Control passes to the next statement	• Control passes to the beginning of the loop
• Terminates the program	• Never terminates the program

Additional C-Programs

C-Program to find the roots of the quadratic equation	Output
#include<stdio.h> #include<math.h> int main() { int a,b,c,d; double root1,root2; printf("Enter the values of a,b,c :\n"); scanf("%d%d%d",&a,&b,&c); d=b*b-4*a*c; root1=(-b+sqrt(d))/(2*a); root2=(-b-sqrt(d))/(2*a); printf("The roots are:\nroot1 = %.1f\nroot2 = %.1f",root1,root2); return 0; }	Enter the values of a,b,c : 1 4 4 The roots are: root1 = -2.0 root2 = -2.0
Biggest of three numbers using ternary operator	**Output**
#include<stdio.h> int main() { int a,b,c,big; printf("Enter the numbers :\n"); scanf("%d%d%d",&a,&b,&c); big=a>b?(a>c?a:c):(b>c?b:c); printf("%d is the greatest number",big); return 0; }	Enter the numbers : 5 8 2 8 is the greatest number
C Program to check perfect number or not	**Output**
# include <stdio.h> int main() { int i, Number, Sum = 0 ; printf("\n Please Enter any number \n") ; scanf("%d", &Number) ; for(i = 1 ; i < Number ; i++) {	• 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 is 6. • So 6 is called as a perfect number.

```
   if(Number % i == 0)
     Sum = Sum + i ;
  }
 if (Sum == Number)
    printf("\n %d is a Perfect Number", Number) ;
 else
    printf("\n%d is not the Perfect Number", Number) ;
return 0 ;
}
```

```
Please Enter any number
6

 6 is a Perfect Number
```

C Program to check strong number or not

```
#include <stdio.h>
int main()
{
    int i, originalNum, num, lastDigit, sum;
    long fact;
    printf("Enter any number to check Strong number: ");
    scanf("%d", &num);
    originalNum = num;
    sum = 0;
    while(num > 0)
    {
       lastDigit = num % 10;
        fact = 1;
        for(i=1; i<=lastDigit; i++)
        {
            fact = fact * i;
        }
        sum = sum + fact;
        num = num / 10;
    }
 if(sum == originalNum)
    {
        printf("%d is STRONG NUMBER", originalNum);
    }
    else
    {
        printf("%d is NOT STRONG NUMBER", originalNum);
    }
    return 0;
}
```

Output

```
Enter any number to check
Strong number: 145
145 is STRONG NUMBER

Strong number is a special
number whose sum of
factorial of digits is equal to
the original number.
For example: 145 is strong
number. Since, 1! + 4! + 5!
= 145
```

Question Bank

Unit 1 & Unit 2

1. Write down the steps involved in writing a program to solve a problem.

To design a program, a programmer must determine three basic steps:

a. The instruction to be performed.

b. The sequence in which those instructions are to be performed.

c. The data required to perform those instructions.

2. List out the various stages in evolution of C.

The various stages in evolution of C are as follows

Language	Year	Founder
ALGOL	1960	International group
BCPL	1967	Martin Richards
B	1970	Ken Thompson
C	1972	Dennis Ritchie
K & RC	1978	Kernighan and Ritchie
ANSIC	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee

3. What is the use of main ( ) function in C program? (May 2009)

Every C program must have main () function. All functions in C, has to begin with (and end with) parenthesis. It is a starting point of all C programs. The program execution starts from the opening brace {and ends with closing brace} within which executable part of the program exists.

4. List the delimiters in C.

The delimiters in C are

a. : Colon

b. ; Semicolon

c. ( ) Parenthesis

d. [ ] Square Bracket

e. { } Curly Brace

f. # Hash

g. , Comma

5. What is the purpose of main ( ) in C?

main ( ) is a special function used by the C system to tell the computer where the program starts. Every program must have exactly one main function. The empty parenthesis immediately following main indicates that the function main has no arguments.

6. Write any four escape sequences in C. (Jan 2013)

Following are the escape sequences in C

- \n -new line
- \t –tab
- \a –alert
- \0 –null

7. What you mean by C tokens? (June 2012)

C tokens are the basic buildings blocks in C language which are constructed together to write a C program. Each and every smallest individual unit in a C program are known as C tokens.

„C" language contains the individual units called C tokens. C tokens are of six types. They are as follows

a. Keywords (Ex: int, while)

b. Identifiers (Ex: main, total)

c. Constants (Ex: 10, 20)

d. Strings (Ex: "total", "hello")

e. Special symbols (Ex: (), {})

f. Operators (Ex: +, /,-,*)

8. What is a variable? Illustrate with an example. (Nov 2014)

A variable is an entity whose value can vary during the execution of a program. A variable can be assigned different data items that at various places within the program. A variable is a data name used for storing a data value. It can be assigned different values at different times during program execution. Ex: a=3, b=5.

9. What are the different data types available in C? (June 2014)

Basic /Primitive Data Types Integer types (signed int, unsigned int) Floating Type (float, double) Character types (signed char)	Derived Data Types Arrays Pointers Structures, Enumeration
User Defined Data Types Structure, Union	Void

10. What is meant by Enumerated data type?

- Enumerated data type is a user defined data type in C language. Enumerated data type variables can only assume values which have been previously declared.
- Ex. enum month { jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec};

11. What are Keywords? What is the importance of keywords in C? (May 2015)

- Keywords are pre-defined / reserved words in a C compiler.
- Each keyword is meant to can be used only for a specific function in a C program.
- Since keywords are referred names for compiler, they can"t be used as variable name. Ex. auto, break, const, else

12. Define Constants in C. Mention the types.

- The constants refer to fixed values that the program may not alter during its execution.
- These fixed values are also called literals.
- Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string constant. Ex. const int LENGTH = 10

13. Differentiate between local and global variable in C.

Local Variable	Global Variable
• Variables are declared inside a function.	• Variables are declared outside any function
• They are accessed only by the statements, inside a function in which they are declared.	• They are accessed by any statement in the entire program.
• Local variables are stored on the stack, unless specified.	• Stored on a fixed location decided by a compiler

• Created when the function block is entered and destroyed upon exit.	• Remain in existence for the entire time • your program is executing.
• They are unknown to other functions and to the main program.	• They are implemented by associating • memory locations with variable names.
• They are recreated each time a function is executed or called.	• They do not get recreated if the function is recalled

14. List out the rules to be followed in forming in identifier. (June 2010)

a. First letter should be alphabets.
b. Both numeric and alphabets are permitted.
c. Both lower case and upper case are allowed.
d. No special symbols are permitted except underscore (_).
e. Keywords are not permitted, blank space is not allowed.

15. What is identifier? Give any two examples for an identifier. (Jan 2009)

Identifiers are the names given to variable program elements such as variables, functions and arrays. Ex. STDNAME, sub, TOT_MARKS, sub123

16. What are Operators? Mention their types in C. What are various types of C operators? (Jan 2014)

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C language is rich in built-in operators and provides following type of operators

a. Arithmetic Operators
b. Relational Operators
c. Logical Operators
d. Bitwise Operators
e. Assignment Operators
f. Misc Operators

17. Is 'main' a keyword in C language? Justify your answer. (May 2011)

The name main is not a keyword in C. From the compiler's perspective, main () is a function. Just like any other function that you may define in your program, such as factorial (), sort (). The simplest way to prove is that you can create a variable "main" in the program and that will work.

```
#include <stdio.h>
int main()
{
   int main = 10;
   printf("The value of main is: %d \n", main);
```

Output

```
The value of main is: 10
```

}

18. What is the difference between ++a and a++?

++a means do the increment before the operation (pre increment) a++ means do the increment after the operation (post increment).

19. Give two examples for logical and relational expression. (Jan 2011)

Relational Expression	Logical Expression
(a>b)	if((a>b)&&(a>c))
(a==b)	if((a>b)‖(a>c))

20. What are the Bitwise Operators available in C? (Jan 2011)

It is used to manipulate the data at bit level. It operates only integers.

Operator	Meaning
&	Bitwise AND
!	Bitwise OR
^	Bitwise XOR
<<	Shift Left
>>	Shift Right
~	One"s complement

21. What is type casting?

- Type casting is a way to convert a variable from one data type to another data type.
- For example, if you want to store a long value into a simple integer then you can typecast long to int. You can convert values from one type to another explicitly using the cast operator.

int x,y;

c = (float) x/y;

where x and y are defined as integers. Then the result of x/y is converted into float.

24. What do you meant by conditional or ternary operator? Give an example for Ternary operator. (Nov 2014)

- Conditional or ternary operator"s returns one value if condition is true and returns another value is condition is false. Using ?: reduce the number of line codes and improve the performance of application.

Syntax: (Condition? true_value: false_value); Ex: a<b ? printf("a is less") : printf("a is greater");

25. What is the use of sizeof() operator in C?

- sizeof() is used with the data types such as int, float, char… it simply return amount of memory is allocated to that data types.
- sizeof(char); //1
- sizeof(int); //4
- sizeof(float); //4
- sizeof(double); //8

sizeof() is used with the expression, it returns size of the expression.

```
int a = 0;

double d = 10.21;

printf("%d", sizeof(a+d)); //8
```

26. Mention the various decisions making statement available in C.

a. if statement
b. if...else statement
c. nested if statements
d. switch statement
e. nested switch statements

27. What is the difference between if and while statement?

If	While
a. It is a conditional statement. b. If the condition is true, it executes some statements. c. If the condition is false then it stops the execution the statements.	a. It is a loop control statement. b. Executes the statements withinthe while block if the condition is true. c. If the condition is false the control is transferred to the next statement of the loop.

28. What are the types of looping statements available in C?

C programming language provides following types of loop to handle looping requirements.

a. for loop
b. while loop
c. do...while loop

29. Distinguish between while and do..while statement in C. (Jan 2009)

while	do while
Executes the statements within the while block if only the condition is true.	Executes the statements within the while block at least once.
The condition of the loop is checked at the starting point of the loop	The condition is checked at the end of the loop

It is an entry controlled loop	It is an exit controlled loop

30. Write a for loop statement to print the numbers from 10 to 1. (Jan 2014)

```
#include <stdio.h>

{

int count; // Display the numbers 10 to 1

 for(count = 10; count >= 1; count--) Output:

printf("%d ", count);

}
```

```
10 9 8 7 6 5 4 3 2 1

Press any key to continue . . .
```

31. What are the types of I/O statements available in C?

There are two types of I/O statements available in C

a. Formatted I/O Statements

b. Unformatted I/O Statements.

32. List the various input and output statements in C. (May 2015)

The various input and output statements in C are

Input Statements	Output Statements
a. gets() b. getch() c. getchar() d. scanf() e. getche()	a. puts() b. putch() c. putchar() d. printf()

33. Write the limitations of using getchar() and scanf() functions for reading strings. (Jan 2009)

- getchar(): It is written in standard I/O library. It reads a single character only from a standard input device. This function is not use for reading strings.
- Scanf: It is use for reading single string at a time. When there is a blank was typed, the scanf() assumes that it is an end.

35. What are the pre-processor directives? (Jan 2014, May 2014, 2015)

- Preprocessor directives are the commands used in preprocessor and they begin with "#" symbol.
- Before a C program is compiled in a compiler, source code is processed by a program called preprocessor. This process is called preprocessing.

Macro	Syntax: #define This macro defines constant value and can be any of the basic data types.
Header file inclusion	Syntax: #include <file_name> The source code of the file "file_name" is included in the main program at the specified place.
Conditional compilation	Syntax: #ifdef, #endif, #if, #else, #ifndef Set of commands are included or excluded in source program before compilation with respect to the condition.
Unconditional compilation	Syntax: #undef, #pragma #undef is used to undefine a defined macro variable. #Pragma is used to call a function before and after main function in a C program.

36. What are storage classes?

- A storage class is the one that defines the scope (visibility) and life time of variables and/or functions within a C Program.

37. What are the storage classes available in C?

Storage classes are categorized in four types as,

- Automatic Storage Class - auto
- Register Storage Class – register
- Static Storage Class – static
- External Storage Class – extern

38. What is register storage in storage class?

- 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 the than normal variables since they are stored in register memory rather than main memory.
  - Scope: Local to the function which it is declared.
  - Default initial value: Garbage value.
  - Lifetime: Till the end of the function/ method block.
- Syntax: register int number;

39. What is static storage class? (Nov 2014)

- Static is the default storage class for global variables.
- The static storage class object will be stored in the main memory.
- Ex. static int Count=19;

40. Define Auto storage class in C.

- Auto is the default storage class for all local variables.
- The auto storage class data object will be stored in the main memory.
- These objects will have automatic (local) lifetime.
- Ex. auto int Month;

41. What is an algorithm?

- Algorithm is an ordered sequence of finite, well defined, unambiguous instructions for completing a task.
- It is an English-like representation of the logic which is used to solve the problem. It is a step- by-step procedure for solving a task or a problem.
- The steps must be ordered, unambiguous and finite in number.

42. Give the rules for writing Pseudo codes.

- Write one statement per line.
- Capitalize initial keywords.
- Indent to show hierarchy.
- End multiline structure.
- Keep statements to be language independent.

43. Give the difference between flowchart and pseudo code.

- Flowchart and Pseudo code are used to document and represent the algorithm.
- In other words, an algorithm can be represented using a flowchart or a pseudo code.
- Flowchart is a graphical representation of the algorithm.
- Pseudo code is a readable, formally styled English like language representation of the algorithm.

44. Define a flowchart.

- A flowchart is a diagrammatic representation of the logic for solving a task.
- A flowchart is drawn using boxes of different shapes with lines connecting them to show the flow of control.
- The purpose of drawing a flowchart is to make the logic of the program clearer in a visual form.

45. Mention the characteristics of an algorithm.

- Algorithm should be precise and unambiguous.
- Instruction in an algorithm should not be repeated infinitely.
- Ensure that the algorithm will ultimately terminate.
- Algorithm should be written in sequence.
- Algorithm should be written in normal English.
- Desired result should be obtained only after the algorithm terminates.

46. What is the difference between algorithm and pseudo code?

- An algorithm is a systematic logical approach used to solve problems in a computer while pseudo code is the statement in plain English that may be translated later to a programming language.
- Pseudo code is the intermediary between algorithm and program.

47. Mention the advantages and disadvantages of Pseudocode

Advantages of Pseudocode	Disadvantages of Pseudocode
▪ Easy to modify ▪ Implements structured concepts ▪ Simple because it uses English-like statements ▪ No special symbols are used	▪ It's not visual ▪ There is no accepted standard, it varies from company to company ▪ Cannot be compiled not executed

48. State the advantages of using flow charts

a. Communication
   - Flowcharts are better way of communicating the logic of a system to all concerned.

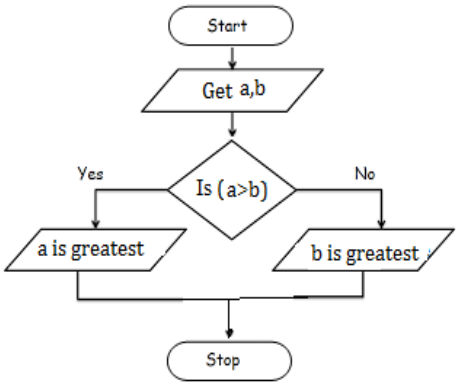
b. Effective Analysis
   - With the help of flowchart, problem can be analyzed in more effective way.

c. Effective Coding
   - The flowcharts act as a guide or blueprint during the system analysis and program development phase.

d. Proper Testing and Debugging
   - The flowchart helps in debugging process.

e. Effective Program Maintenance
   - Maintenance of running program becomes easy with the help of flow chart

49. Draw the flow chart to find the greater of two numbers



Part-B

1. Explain different data types in C with examples.
2. What are the different operators available in C? Explain with examples.
3. Differentiate Signed and Unsigned integer.
4. Describe the statements for decision making, branching and looping.
5. Explain in detail about the formatted and unformatted I/O functions in C.
6. Discuss about bitwise operators and logical operators in C.
7. Explain any four format string with examples.
8. Write the syntax of „for" construct in C. Give an example.
9. Write the algorithm and C program to print the prime numbers less than 500.
10. Explain the following

    conditional statements.

    Nested if-else

    statement Switch-case

    statement
11. Write about the need and types of looping statements in C language and discuss with examples**. (Jan, Nov 2014, May 2015)**
12. Write about the need and types of branching statements in C language and discuss with examples. **(Jan 2014)**
13. Write a program to check whether a given number is prime or not. **(May 2014)**
14. Write a C program to find sum of digits of an integer**. (May 2014)**
15. Write a C program to find roots of a quadratic equation. **(May 2014)**
16. Differentiate entry and exit checked conditional constructs with an example**. (May 2014)**
17. What are the various operators available in C? Discuss each one of them with suitable illustrations. **(Nov 2014, May 2015)**
18. What are constants? Explain the various types of constants in C**. (May 2015)**
19. Write a C program to solve any quadratic equations. **(May 2015)**
20. Draw a flow chart to accept three distinct numbers find the greatest and print the result. **(AU Jan 2018)**
21. Draw the flow chart to find the sum of the series 1+2+3+4+5+…..+100. **(AU Jan 2018)**

22. Explain about algorithm, Pseudocode and flowchart with an example of finding sum of n numbers. (**AU Nov 2016)**
23. Draw flowchart to check whether the given number is zero, positive or negative. (**AU Apr/May 2015)**
24. Draw a flowchart to find the factorial of a number. **(June 2014)**
25. Write an algorithm to find the largest of three numbers. **(Jan 2013, Jan 2010)**
26. Draw the flowchart to solve roots of a quadratic equation. (**AU May 2016, Jan 2013)**

Sri Eshwar College of Engineering
Coimbatore- 641 202

Unit III

ARRAYS AND STRINGS

1D Array –Declaration, Initialization, 2DArray - Declaration, Initialization, Multi-dimensional Arrays Strings: Declaration, Initialization, String operations: length, compare, concatenate, copy

INTRODUCTION TO ARRAYS

- Consider the following example.

```
main ()
{
int a=2;
a=4;
printf("%d",a);

OUTPUT: 4
```

- In the above example, the value of **a** printed is 4. 2 is assigned to **a** before assigning 4 to it.
- When we assign 4 to **a** then the value stored in **a** is replaced with the new value.
- **Hence ordinary variables are capable of storing one value at a time.**
- This fact is same for all the data types.
- But in many applications the variables must be assigned to more than one value. This can be obtained with help of arrays.
- **Array variables are able to store more than one value at a time.**

DEFINITION OF ARRAY

An array is a collection of similar data types in which each element is located in separate memory location

TYPES OF ARRAY

- One Dimensional array
- Two dimensional array

- Multi-dimensional array

ONE DIMENSIONAL ARRAY

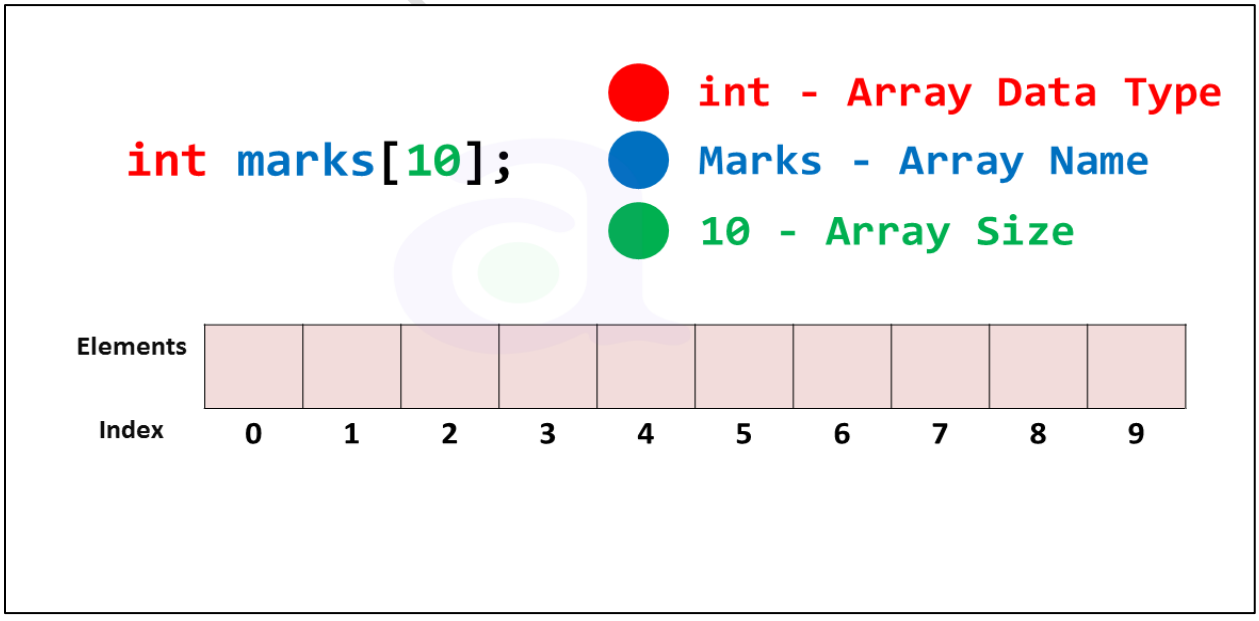
- An array which has only one subscript (or) index is known as one dimensional array.
- Single dimensional array is used to store data in sequential order.
- The declaration of a one-dimensional array takes the following form:

```
data_type array_name [size];
```

- Data type- what kind of values it can store, for example int, char, float, double.
- Array name- to identify the array
- Size- the maximum number of values that the array can hold

Programming Tip:
To declare and define an array, one must specify its name, type, and size

- An 1D array can be declared as follows
  - int marks [10];
  - It tells the compiler that marks is an integer type of array and can store 10 integers.
  - The compiler reserves 2 bytes of memory for each integer array element.



- The elements of an integer array **marks [10]** are stored in continuous memory location.
- It is assumed that the starting memory location is **2000.**
- Each integer requires 2 bytes.
- Hence subsequent elements appears after a gap of 2 locations.

Element	marks[0]	marks[1]	marks[2]	marks[3]	marks[4]	marks[5]	marks[6]	Marks[7]	Marks[8]	Marks[9]
Address	2000	2002	2004	2006	2008	2010	2012	2014	2016	2018

- Arrays are referenced with the help of index vales.
- Assume that the array contains "n" integers, then the first element are indexed with the "0" value and the last element are indexed with "n-1" value.

ONE DIMENSIONAL ARRAY INITIALIZATION

- The process of giving initial values to array elements during declaration is called as initialization of arrays.
- The initialization of 1-D arrays in C is done in two ways as follows:
  - At compile time
    1. Initializing all specified memory locations
    2. Partial array initialization
    3. Initialization without size
    4. String initialization
  - At runtime

Compile Time Initialization

- We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.
- The general form of initialization of array is

data_type array-name[size]= {list of values};

1. Initializing all specific memory locations

- Arrays can be initialized at the time of declaration when their **initial values are known in advance.**
- Ex:- int a[5]={10,15,1,3,20};
- During compilation, 5 contiguous memory locations are reserved by the compiler for the variable **a** and all these locations are initialized as shown in figure.

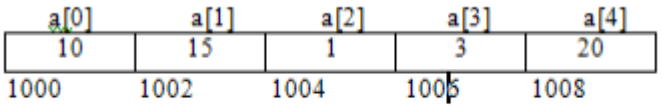


Fig: Initialization of int Arrays

- Example: int a[3]={9,2,4,5,6}; //error: no. of initial vales are more than the size of array.

2. Partial array initialization

- Partial array initialization is possible in c language.
- If the number of **values to be initialized is less than the size of the array, then the elements will be initialized to zero automatically.**
- Ex:- int a[5]={10,15};
- Even though compiler allocates 5 memory locations, using this declaration statement; the compiler initializes first two locations with 10 and 15, the next set of memory locations are automatically initialized to 0's by compiler as shown in figure.

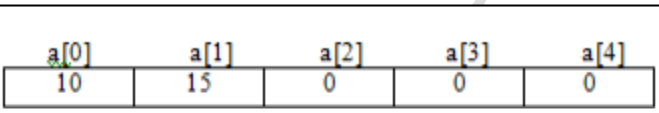
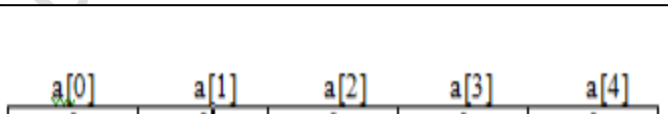


Fig: Partial Array Initialization

- Initialization with all zeros:-
  - Ex:- int a[5]={0};



3. Initialization without size

- In this scheme of compile time initialization, we does not provide size to an array but instead we provide set of values to the array.
- Ex:- int num[] = {2,8,7,6,0}; // array size is automatically set to 5

- **Compiler counts the number of elements written inside pair of braces** and determines the size of an array.

4. String Initialization

- Consider the declaration with string initialization.
- Ex:- char b[]="COMPUTER";
- The array b is initialized as shown in figure.

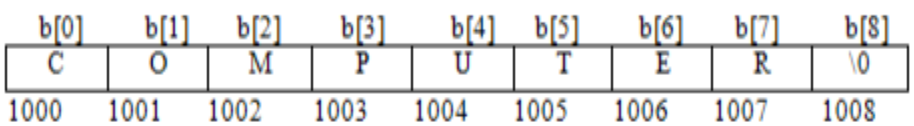


Fig: Array Initialized with a String

- Even though the string "COMPUTER" contains 8 characters, because it is a string, it always ends with null character. So, the array size is 9 bytes (i.e., string length 1 byte for null character).
- Ex:-
  - char b[9]="COMPUTER"; // correct
  - char b[8]="COMPUTER"; // wrong

Run time Initialization

- An array can be explicitly initialized at run time. This approach is usually applied for initializing large arrays.
- Ex:- scanf can be used to initialize an array

```
int x[3];
scanf ("%d%d%d",&x[0],&x[1],&x[2]);
```

- Using for loop one can input the value for each element of the array.

```
int i, marks[10];
for (i=0;i<10;i++)
        scanf ("%d",&marks[i]);
```

CHARACTERISTICS OF ARRAY

- All the elements of an **array share the same name**, and they are distinguished from one another with the help of an element number.
- The **element number** in an array plays major role for calling each element.
- Any particular element of an array can be modified separately without **disturbing other elements.**
- All elements of array are stored in the **continuous memory location.**
- The **size** of an array must be a ***constant integer value.***
- The first element in an array index is **zero**, whereas the last element is at index **[size_of_array-1].**
- The total size in bytes for a single dimensional array is computed as shown below.

Total bytes= sizeof (data type) × size of array

ADVANTAGES OF ARRAY

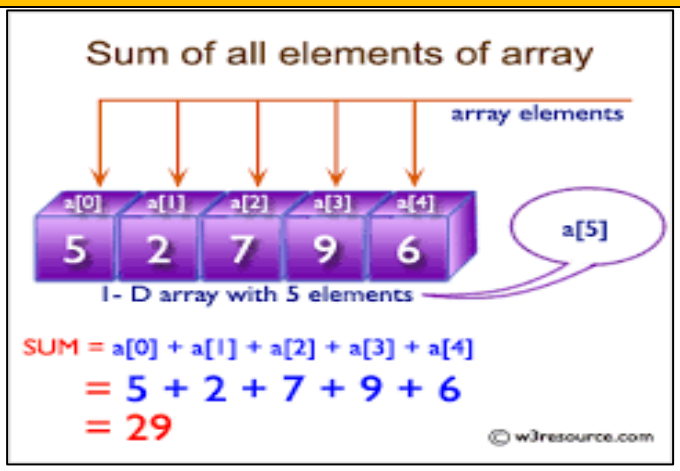
- **Code Optimization**: Less code to the access the data.
- **Ease of traversing**: By using for loop, we can retrieve the elements of an array easily.
- **Ease of sorting**: To sort the elements of the array, we need a few lines of code only.
- **Random Access**: We can access any element randomly using the array.

DISADVANTAGES OF ARRAY

- The elements in the **array must be same data types.**
- The size of an array is **fixed.**
- It is not possible to **extend the limit of an array.**
- The **insertion and deletion an operation is an array require shifting of elements which takes times.**

Sri Eshwar College of Engineering
Coimbatore- 641 202

EXAMPLE PROGRAMS USING ONE DIMENSIONAL ARRAY

Write a program to print bytes reserved for various types of data and space required for storing them in memory using arrays	**Output**
#include<stdio.h> #include<conio.h> void main() { int i[10]; char c[10]; long l[10]; clrscr(); printf("The type int requires %d bytes",sizeof(int)); printf("\nThe type char requires %d bytes",sizeof(char)); printf("\nThe type long requires %d bytes",sizeof(long)); printf("\n %d memory location are reserved for 10 int elements",sizeof(i)); printf("\n %d memory location are reserved for 10 char elements",sizeof(c)); printf("\n %d memory location are reserved for 10 long elements",sizeof(l)); getch(); }	The type int requires 2 bytes The type char requires 1 bytes The type long requires 4 bytes 20 memory location are reserved for 10 int elements 10 memory location are reserved for 10 char elements 40 memory location are reserved for 10 long elements Explanation: The **sizeof ()** function provides the size of data types in bytes. Memory locations required for the arrays is equal to **argument of an array × sizeof [data type]**
Program to calculate sum of array content	**Output**
#include <stdio.h> int main() { int a[100],sum=0,i,n; printf("Enter the size of an array:\n"); scanf("%d",&n); printf("Enter the array elements one by one:\n"); for(i=0;i<n;i++) scanf("%d",&a[i]); for(i=0;i<n;i++) sum=sum+a[i];	

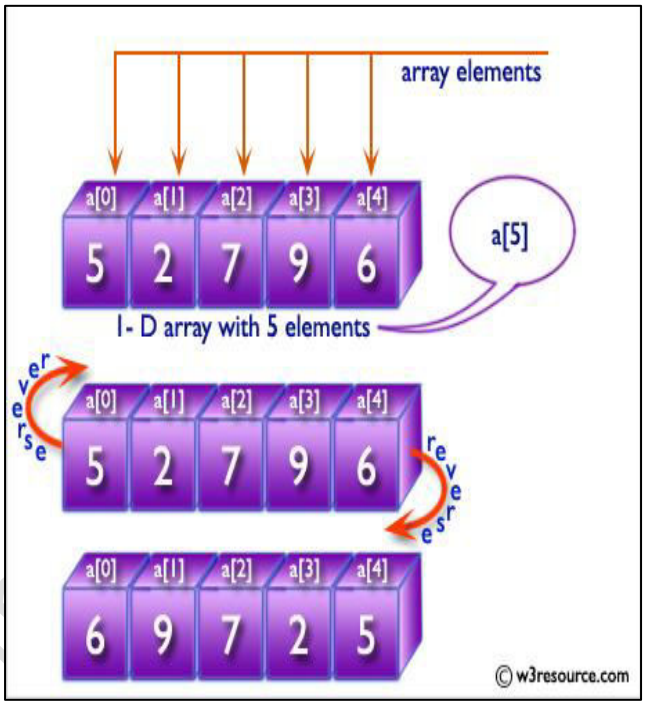
```
    printf("The sum of array values is
%d",sum);
    return 0;
}
```

```
Enter the size of an array: 5
Enter the array elements one by one:
5 2 7 9 6
The sum of array values is 29
```

Program for reversing an array

```
#include <stdio.h>
int main()
{
  int a[10],i;
  int n;
  printf("Enter the array size:\n");
  scanf("%d",&n);
  printf("Enter the array elements one by
one:\n");
  for(i=0;i<n;i++)
  {
      scanf("%d",&a[i]);
  }
  printf("*****Array in Reverse
Order*****\n");
  for(i=n-1;i>=0;i--)
  {
      printf("%d\t",a[i]);
  }
return 0;
}
```

Output




```
Enter the array size:
5
Enter the array elements one by one:
5
11
12
13
14
*****Array in Reverse Order*****
14      13      12      11      5
```

C-Program to find Linear Search

Linear Search

- Linear search is a simple search algorithm for searching an element in an array.
- It works by comparing each element of an array.
- It is most basic and easiest algorithm in computer science to find an element in an array.
- The time complexity of linear search is O(n)

Steps to perform Linear Search

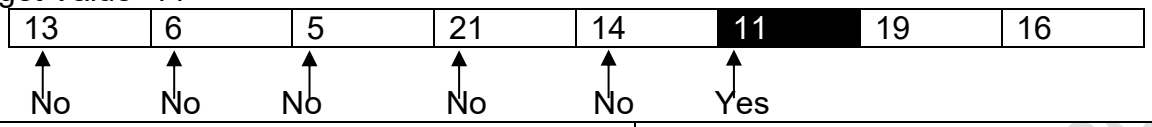
- Read the key value from the user.

- Compare the key value with the first element in the list.
- If both are matching, then display element is found.
- If both are matching, then compare with next element.
- Continue step 2, 3, and 4 till the last element.

Example

Unsorted array : 13,6,5,21,14,11,19,16

Target Value=11




```
#include<stdio.h>
int main()
   {
      int a[15];
      int size,i,key,flag=0;
      printf("Enter the array size:\n");
      scanf("%d",&size);
      printf("Enter the array elements:\n");
      for(i=0;i<size;i++)
      {
         scanf("%d",&a[i]);
      }
      printf("Enter the key value to perform
search operation:\n");
      scanf("%d",&key);
     for(i=0;i<size;i++)
      {
         if(key==a[i])
         {
            flag=1;
            break;
         }
      }
      if(flag==1)
      printf("%d is present in the array",key);
      else
      printf("%d is not present in the
array",key);
      return 0;
   }
```

```
Enter the array size:
5
Enter the array elements:
8
3
2
1
5
Enter the key value to perform search operation:
5
5 is present in the array
```

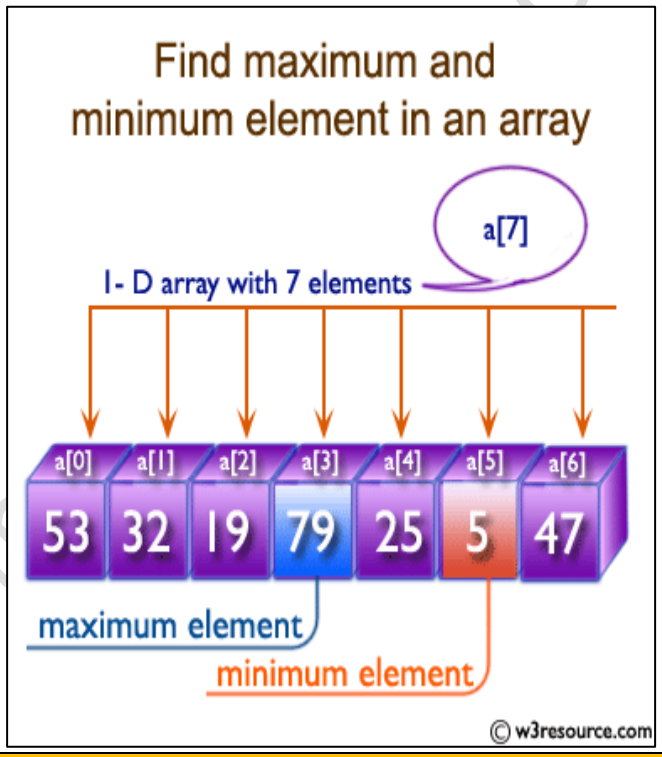
Write a C Program to find the largest and smallest element in an array	**Output**

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

```
Enter the number of elements in an array:
5
```

```
{
int array[5],small,large,i,n;
printf("Enter the number of elements in an
array:\n");
scanf("%d",&n);
printf("Enter the array elements one by
one:\n");
for(i=0;i<n;i++)
{
scanf("%d",&array[i]);
}
small=array[0];
large=array[0];
for(i=1;i<n;i++)
{
if(array[i]<small)
{
        small=array[i];
    }
    if(array[i]>large)
    {
        large=array[i];
    }
  }
  printf("Large =%d,small=%d",large,small);
return 0;
}
```

```
Enter the array elements one by one:
10
5
20
30
2
Large =30,small=2
```



Program to sort the give numbers in ascending order

```
#include<stdio.h>
int main()
{
    int i,j,n,a[100],temp;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(a[i]>a[j])
```

Output

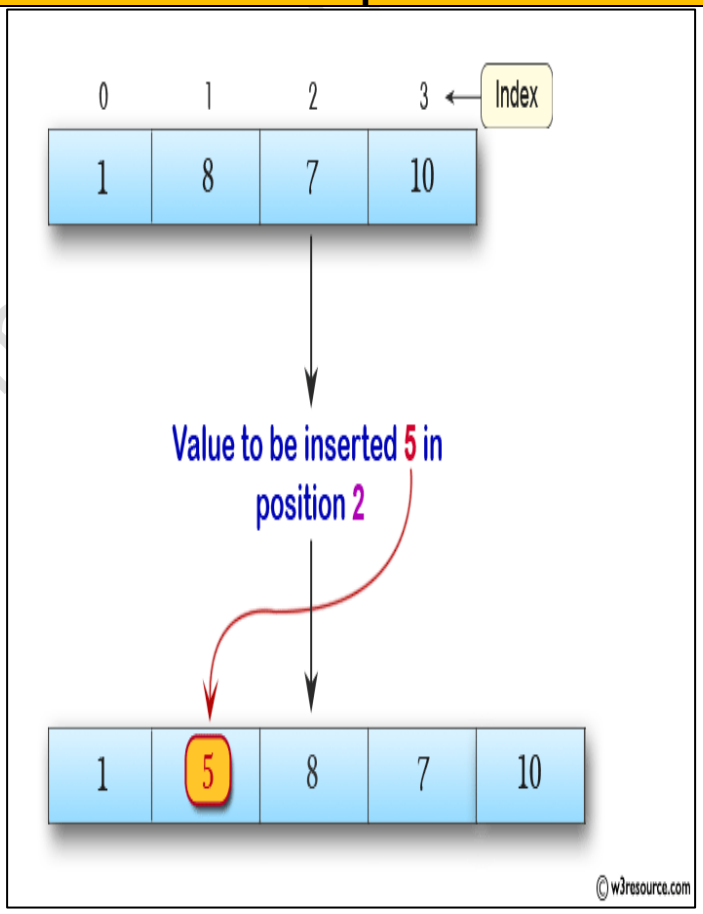
```
5
23
56
10
20
5
5 10 20 23 56
```

```
        {
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
        }
    }
}
for(i=0;i<n;i++){
    printf("%d ",a[i]);
}
return 0;
}
```

Program to insert an element in an array

```
#include <stdio.h>
int main()
{
    int list[100], i, n, position, data;
    printf("Enter the number of elements : ");
    scanf("%d", &n);
    printf("Enter %d elements : ", n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &list[i]);
    }
    printf("Enter the location of element to
insert : ");
    scanf("%d", &position);
    printf("Enter the number to insert : ");
    scanf("%d", &data);
    for (i = n - 1; i >= position-1; i--)
    {
        list[i + 1] = list[i];
    }
    list[position-1] = data;
    printf("Resultant array is\n");
    for (i = 0; i <= n; i++)
    {
        printf("%d ", list[i]);
    }
    return 0;
}
```

Output



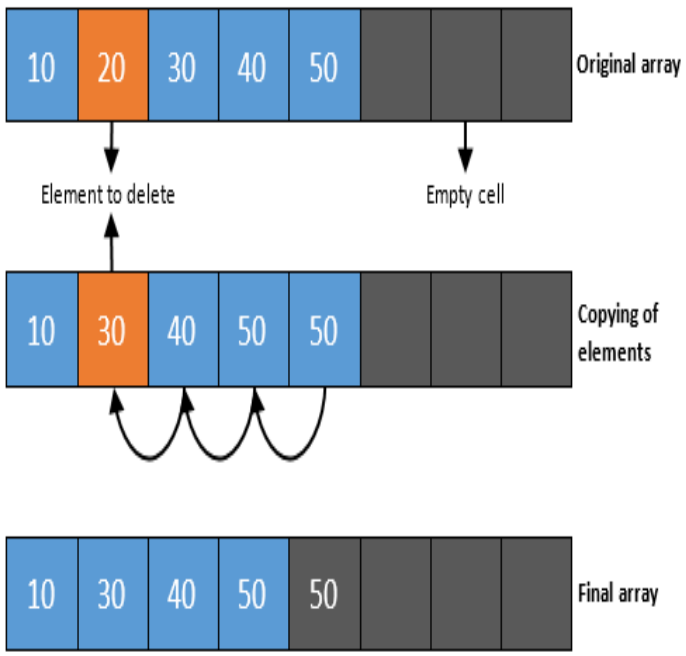

```
Enter the number of elements : 5
Enter 5 elements : 10
20
30
40
50
```

Enter the location of element to insert : 2
Enter the number to insert : 15
Resultant array is
10 15 20 30 40 50

Program to delete an element in an array

```
#include <stdio.h>
#define MAX_SIZE 100
int main()
{
    int arr[MAX_SIZE];
    int i, size, pos;
    printf("Enter size of the array : ");
    scanf("%d", &size);
    printf("Enter elements in array : ");
    for(i=0; i<size; i++)
    {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element position to delete : ");
    scanf("%d", &pos);
    if(pos < 0 || pos > size)
    {
        printf("Invalid position! Please enter position between 1 to %d", size);
    }
    else
    {
      for(i=pos-1; i<size-1; i++)
       {
          arr[i] = arr[i + 1];
       }

       size--;
    }
    printf("\nElements of array after delete are : ");
    for(i=0; i<size; i++)
    {
        printf("%d\t", arr[i]);
    }

    return 0;
```

Output



Step by step descriptive logic to remove element from array.

1. Move to the specified location which you want to remove in given array.
2. Copy the next element to the current element of array. Which is you need to perform **array[i]=array[i+1]**
3. Repeat above steps till last element of array.
4. Finally decrement the size of array by one.

Enter size of the array : 5
Enter elements in array : 10
20
30
40
50

}	Enter the element position to delete : 2 Elements of array after delete are : 10 30 40 50

TWO DIMENSIONAL ARRAY

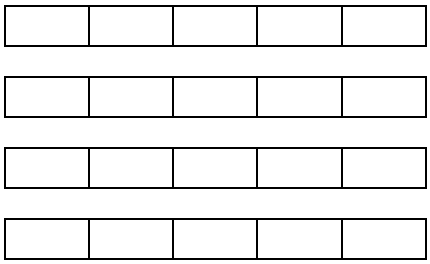
- An array with **two subscript** is known as two dimensional array.
- The elements of a 2D array are arranged in rows and columns.
- The general form of 2D array is as follows

data_type array_name [row size] [column size]

- Where data type refers to the type of array like int, float, char, etc.,
- Array name denotes the name of the two dimensional array.
- Row size refers to the maximum number of rows in the array.
- Column size refers to the maximum number of columns in the array.
- The pictorial representation of 2D array is as follows

	Col1	Col2	Col3
Row1	x[0][0]	x[0][1]	x[0][2]
Row2	x[1][0]	x[1][1]	x[1][2]
Row3	x[2][0]	x[2][1]	x[2][2]

- Hence, we see that a 2D array is treated as a collection of 1D arrays.
- Representation of int arr[4][5] as individual 1D arrays is given below:



- Size of arr[4][5]=4x5=20 elements

TWO DIMENSIONAL ARRAY INITIALIZATION

- Different ways to initialize 2-D arrays are as follows
  - Row wise assignment

- Combine assignment
- Selective assignment

- Initializing all elements row wise
  - Example: int a[3][2] ={
    {1, 4},
    {5, 2},
    {6, 5}
    };

Example for row wise initialization	Output
#include<stdio.h> #include<conio.h> void main() { int i,j; int a[3][2]={{1,4},{5,2},{6,5}}; clrscr(); for(i=0;i<3;i++) { for(j=0;j<2;j++) { printf("%d ",a[i][j]); } printf("\n"); } getch(); }	1 4 5 2 6 5

- Combine and initializing 2D array
  - int a[3][2]={1,4,5,2,6,5};

Example for row wise initialization	Output
#include<stdio.h> #include<conio.h> void main() { int i,j; int a[3][2]={1,4,5,2,6,5}; clrscr(); for(i=0;i<3;i++) { for(j=0;j<2;j++) { printf("%d ",a[i][j]);	1 4 5 2 6 5

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

- Selective Assignment
  - In this type, values are not given for all elements as,
  - Ex: int a[3][2]= {{1},
{5, 2},
{6}
};

Example for row wise initialization	Output
#include<stdio.h> #include<conio.h> void main() { int i,j; int a[3][2]={{1},{5,2},{6}}; clrscr(); for(i=0;i<3;i++) { for(j=0;j<2;j++) { printf("%d ",a[i][j]); } printf("\n"); } getch(); }	1 0 5 2 6 0

How to access 2D Array?

- Using row index and column index
- For example:
  - One can access element stored in 1st row and 2nd column of below array

1	2	3
4	5	6

- Using a[0] [1]

Run time Initialization of 2D Array

- The looping statements are generally used to assign values to array elements. The general form is,

```
for(i=0;i<rowsize;i++)
{
   for(j=0;j<columnsize;j++)
   {

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

C Program to perform matrix multiplication **Note:** • **To multiple two matrices, the number of columns of first matrix should be equal to the number of rows to the second matrix.**	**Output**
#include <stdio.h> # define MAX 50 int main() { int a[MAX][MAX],b[MAX][MAX],product[MAX][MAX]; int arows,acolumns,brows,bcolumns; int i,j,k; int sum=0; printf("Enter the row and columns of Matrix A:\n"); scanf("%d%d",&arows,&acolumns); printf("Enter the elements of Matrix A:\n"); for(i=0;i<arows;i++) { for(j=0;j<acolumns;j++) { scanf("%d",&a[i][j]); } } printf("Enter the row and columns of Matrix B:\n");	Enter the row and columns of Matrix A: 3 3 Enter the elements of Matrix A: 1 2 3 1 2 1 3 1 2 Enter the row and columns of Matrix B: 3 3 Enter the elements of Matrix B: 1 2 3 1 2 1 3 1 2 Resultant Matrix 12 9 11 6 7 7 10 10 14

```
    scanf("%d%d",&brows,&bcolumns);
    if(acolumns!=brows)
    {
       printf("Sorry! We cannot multiply the matrices a
and b\n");
         }
    else
    {
       printf("Enter the elements of Matrix B:\n");
    for(i=0;i<brows;i++)
    {
       for(j=0;j<bcolumns;j++)
       {
          scanf("%d",&b[i][j]);
               }
    }
    }
printf("\n");
    for(i=0;i<arows;i++)
    {
       for(j=0;j<bcolumns;j++)
       {
          for(k=0;k<brows;k++)
          {
             sum+=a[i][k]*b[k][j];
          }
          product[i][j]=sum;
          sum=0;
       }
    }
//Printing array elements
    printf("Resultant Matrix\n");
    for(i=0;i<arows;i++)
    {
       for(j=0;j<bcolumns;j++)
       {
          printf("%d ",product[i][j]);
       }
       printf("\n");
    }
    return 0;
    }
```

C Program to find the transpose of a given matrix	Output
#include <stdio.h> int main() { int A[10][10],T[10][10]; int r,c,i,j; scanf("%d%d",&r,&c); for(i=0;i<r;i++) { for(j=0;j<c;j++) { scanf("%d",&A[i][j]); } } for(i=0;i<r;i++) { for(j=0;j<r;j++) { T[j][i]=A[i][j]; } } for(i=0;i<r;i++) { for(j=0;j<c;j++) { printf("%d ",T[i][j]); } } printf("\n"); }	2 2 1 2 3 6 1 3 2 6
Write a C Program to add two matrices	**Output**
#include<stdio.h> int main() { int n,i,j,a[10][10],b[10][10]; scanf("%d",&n); for(i=0;i<n;i++) { for(j=0;j<n;j++) { scanf("%d",&a[i][j]); }	2 1 1 1 1 1 1 1 1

```
    }
for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&b[i][j]);
        }
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            printf("%d ",a[i][j]+b[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```
2
2
2
2
```

MULTI DIMENSIONAL ARRAYS

- A multidimensional array in simpler terms is an array of arrays.
- Like we have one index in a one-dimensional array, two indices in a two-dimensional array, in the same way **we have n indices in an n-dimensional array or multi-dimensional array.**
- The general form of a multi-dimensional array is

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

- data_type: Type of data to be stored in the array.
- array_name: Name of the array.
- size1, size2,..., sizeN: Sizes of the dimensions.

INITIALIZING 3-D Array

- Initialization in Three-Dimensional array is same as that of Two-dimensional arrays.

- The difference is as the number of dimension increases so the number of nested braces will also increase.
- Method 1:

```
int x[2][3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                  11, 12, 13, 14, 15, 16, 17, 18, 19,
                  20, 21, 22, 23};
```

- Better Method

```
int x[2][3][4] =
 {
   { {0,1,2,3}, {4,5,6,7}, {8,9,10,11} },
   { {12,13,14,15}, {16,17,18,19}, {20,21,22,23} }
 };
```

Example for 3 Dimensional Array	**Output**
#include<stdio.h> #include<conio.h> void main() { int i, j, k; int arr[3][3][3]= { { {11, 12, 13}, {14, 15, 16}, {17, 18, 19} }, { {21, 22, 23}, {24, 25, 26}, {27, 28, 29} }, { {31, 32, 33}, {34, 35, 36}, {37, 38, 39} }, }; printf(":::3D Array Elements:::\n\n"); for(i=0;i<3;i++)	:::3D Array Elements::: 11 12 13 14 15 16 17 18 19 21 22 23 24 25 26 27 28 29 31 32 33 34 35 36 37 38 39

```
{
   for(j=0;j<3;j++)
   {
      for(k=0;k<3;k++)
      {
      printf("%d\t",arr[i][j][k]);
      }
      printf("\n");
   }
   printf("\n");
}
getch();
}
```

STRINGS- INTRODUCTION, DECLARATION & INITIALIZATION

- The string can be defined as the one-dimensional array of characters **terminated by a null (‘\0’).**
- The character array or string is used to manipulate text such as word or sentences.
- In C Language, a string is a ***null-terminated character*** array.
- This means that after the last character, a null character (‘\0’) is stored to signify the end of the character array.
- Below is the basic syntax for declaring a string.

char str_name[size];

- str_name is any name given to the string variable.
- size is used define the length of the string, i.e the number of characters strings will store.
- For example, if we write **char str []=”HELLO”**
- We are declaring a character array that has five usable characters namely H, E, L, L, and O.
- Apart from these characters, a null character (‘\0’) is stored at the end of the string.
- So, the internal representation of the string becomes HELLO’\0’.
- To store a string of length 5, we need 5+1 locations (1 extra for the null character).

- The name of the character array (or the string) is a pointer to the beginning of the string.
- Like we use subscripts (also known as index) to access the elements of an array, similarly subscripts are also used to access the elements of the character array.
- The subscript starts with a zero.
- All the characters of a string array are stored in successive memory locations.

str[0]	H	1000
str[1]	E	1001
str[2]	L	1002
str[3]	L	1003
str[4]	O	1004
str[5]	\0	1005

Memory Representation of a character array

- The other way to initialize is to initialize it as an array of characters, like

```
char str []= {'H', 'E', 'L', 'L', 'O', '\0'};
```

- In the above example the compiler automatically calculates the size based on the number of elements initialized.
- So, in this example 6 memory slots are reserved to store string variable, str.
- We also declare a string with size much larger than the number of elements that are initialized. For example

```
char str [7]= "HELLO";
```

- In such cases, the compiler creates a character array of size 10; stores the value "HELLO" in it and finally terminates the value with a null character.
- Rest of the elements are automatically initialized to NULL.

H	E	L	L	O	\0	\0

- char s[4]="HELLO"
- In the above declaration array is initialized with more elements than it can store.
- So it will generate a compile time error.

- char s[6]="Hello";
- char s[6]={'H', 'e', 'l', 'l', 'o', '\0'};

They both are equal

READING STRINGS

- **scanf()**
  - The scanf function with %s format specification can be used to read a string from the user and store it in a character array.
  - However, the scanf () function takes the first entered word.
  - The function terminates when it encounters a white space (or just space).

To read a string using scanf()	Output
#include <stdio.h> int main() { char name[20]; printf("Enter name: "); scanf("%s",name); printf("Your name is %s.",name); }	Enter name: Mahatma Gandhi Your name is Mahatma.

  - Although, scanf () has the way to **set the limit for the number of characters** to be stored in the character array.
  - By using **%ns**

To read a string using scanf()	Output
#include <stdio.h> int main() { char name[20]; printf("Enter name: "); scanf("%9s",name); printf("Your name is %s",name); }	Enter name: YOUAREMOSTWELCOME Your name is YOUAREMOS

- **gets()**
  - gets () is a simple function that overcomes the drawbacks of the scanf() function.
  - Since gets () is a function it requires a set of parenthesis.

Example using gets()	Output
#include <stdio.h> int main() { char name[20]; printf("Enter the name:\n"); gets(name); printf("The entered string is %s\n",name); return 0; }	Enter the name: chennai city The entered string is chennai city

- **Using getchar ()**
  - String can also read by calling getchar () function repeatedly to read a sequence of single characters (unless a terminating character is entered) and simultaneously storing it in a character array as shown below.

Example using getchar()	Output with Explanation
#include<stdio.h> void main() { int i = 0; char name[20]; printf("\nEnter the Name : "); while((name[i] = getchar())!='\n') i++ ; getch(); }	Enter the Name : Sri Eshwar while((name[i] = getchar())!='\n') i++ ; While loop will accept the one character at a time and check it with newline character. Whenever user enters newline character then control comes out of the loop.

WRITING STRINGS

- **Using printf ()**
  - We can use conversion character's' to output a string.
  - We may also use width and precision specifications along with %s.
  - For example printf("%5.3s",str);
  - The above statement would print only first three characters in a total field if five characters.
  - Also these characters are right justified in the allocated width.
  - To make string left justified, we must use a minus sign.

- For example: printf("%-5.3s",str);

Example using printf()	Output with Explanation
#include <stdio.h> int main() { char str[]="Introduction to C"; printf("\n |%s|",str); printf("\n |%20s|",str); printf("\n |%-20s|",str); printf("\n |%.4s|",str); printf("\n |%20.4s|",str); printf("\n |%-20.4s|",str); return 0; }	|Introduction to C| | Introduction to C| |Introduction to C | |Intr| | Intr| |Intr |

- **Using puts ()**
  - The function puts () is an extension of the printf () function.
  - It is a combination of printf () with a new line character.
  - In printf () we use a new line character '\n' to skip to the next line.
  - Whereas in puts () it will automatically skip to the next line after printing the message on the screen.

Example using puts()	Output with Explanation
#include<stdio.h> int main() { char name[20]; printf("Enter the name:\n"); gets(name); puts("The entered string is"); puts(name); }	Enter the name: bala The entered string is bala

String Operations

- **Finding Length of a String : strlen**
  - The strlen () function is used to find the length of a string.
  - The terminating character is not counted while determining the length of the string.

- **Syntax:**

```
var_name= strlen (string);
```

C Program to calculate the length of the string using strlen ()	**Output**
#include <stdio.h> #include <string.h> int main() { char name[20]; int len; gets(name); len=strlen(name); printf("length=%d",len); }	bala murugan length=12
C Program to calculate the length of the string without using strlen ()	**Output**
#include <stdio.h> #include <string.h> int main() { char s[30]; int i,length=0; printf("Enter a string:\n"); gets(s); for(i=0;s[i]!='\0';i++) { length++; } printf("The length of %s = %d\n", s, length); }	Enter a string: bala murugan The length of bala murugan = 12

- **String Copying : strcpy**
  - This function copies the content of one string to another.

```
strcpy (s2,s1);
```

  - Where, s1 is source string

- s2 is destination string
- s1 is copied to s2

To copy one string to another using strcpy () function	**Output**
#include <stdio.h> #include <string.h> int main() { char str1[20]="Hello"; char str2[20]; strcpy(str2,str1); printf("String 2=%s",str2); }	String 2=Hello
To copy one string to another without using strcpy () function	**Output**
#include <stdio.h> int main() { char s1[100], s2[100], i; printf("Enter string s1: "); scanf("%s",s1); for(i = 0; s1[i] != '\0'; ++i) { s2[i] = s1[i]; } s2[i] = '\0'; printf("String s2: %s", s2); return 0; }	Enter string s1: hello String s2: hello

- **String concatenate strcat ()**
  - The process of joining two strings together is called concatenation.
  - The general form is,

strcat (string 1, string 2);

  - It takes two arguments.
  - The characters of second string are appended at the end of the first string.
  - The null terminator originally ending string 1 is overwritten by the first character of string 2.

Illustration of strcat function	Output
#include <stdio.h> int main() { char str1[50],str2[20]; printf("Enter the first string :\n"); gets(str1); printf("Enter the second string :\n"); gets(str2); strcat(str1,str2); printf("Resultant string is : %s",str1); return 0; }	Enter the first string : Good Enter the second string : Luck Resultant string is : GoodLuck

String concatenation without using strcat()	Output
#include <stdio.h> int main() { char firstString[100],secondString[100]; int i,j; printf("Enter your first string : "); gets(firstString); printf("Enter your second string : "); gets(secondString); i = 0; while(firstString[i] != '\0'){ i++; } j = 0; while(secondString[j] != '\0'){ firstString[i] = secondString[j]; i++; j++; } firstString[i] = '\0'; printf("Final String : %s\n",firstString); return 0; }	Enter your first string : bala Enter your second string : murugan Final String : balamurugan

- **String reverse: strrev ()**
  - The strrev () function is used to reverse a given string

```
strrev(string);
```

C Program to reverse a string using strrev()	Output
#include <stdio.h> #include <string.h> #include<conio.h> void main() { char str1[20]="Hello"; strrev(str1); printf("Reverse =%s",str1); getch(); }	Hello olleH
C Program to reverse a string without using strrev()	**Output**
#include<stdio.h> #include<string.h> void main() { int i,n; char str[20]; printf("Enter the String to get reversed\n"); gets(str); n=strlen(str); printf("\nReversed string is \n"); for(i=n-1;i>=0;i--) { printf("%c",str[i]); } return 0; }	Enter the String to get reversed bala Reversed string is alab

- **String Comparison: strcmp**
  - *The strcmp () function compares two strings.*
  - If both the strings are identical then this function returns 0.
  - Otherwise it returns numerical value which is the difference between the ASCII values of the first mismatching characters.
  - The comparison terminates when the first mismatch occurs.

strcmp (string 1, string 2);

  - Example:
    - str1=Trisea

- str2=publishers
- strcmp (str1,str2);

- The execution of this statement results 4.
- Because the ASCII value of T is 84 and ASCII value of p is 80.
- The difference between the ASCII values is 84-80 = 4, so the string are not identical.

- ***Return Value***

Return Value	***Remarks***
0	if both strings are identical (equal)
negative	If the ASCII value of the first unmatched character is less than second.
positive integer	If the ASCII value of the first unmatched character is greater than second.

C Program to compare two strings using strcmp()	**Output**
#include <stdio.h> #include <string.h> int main() { char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd"; int result; // comparing strings str1 and str2 result = strcmp(str1, str2); printf("strcmp(str1, str2) = %d\n", result); // comparing strings str1 and str3 result = strcmp(str1, str3); printf("strcmp(str1, str3) = %d\n", result); return 0; }	strcmp(str1, str2) = 32 strcmp(str1, str3) = 0 • The first unmatched character between string str1 and str2 is third character. • The ASCII value of 'c' is 99 and the ASCII value of 'C' is 67. • Hence, when strings str1 and str2 are compared, the return value is 32. • When strings str1 and str3 are compared, the result is 0 because both strings are identical.

C Program to compare two strings without using strcmp()	**Output**
#include<stdio.h> int main() { char str1[30], str2[30];	Enter two strings :bala bala str1 = str2

```
   int i;
   printf("\nEnter two strings :");
   gets(str1);
   gets(str2);
   i = 0;
   while (str1[i] == str2[i] && str1[i] != '\0')
      i++;
   if (str1[i] > str2[i])
      printf("str1 > str2");
   else if (str1[i] < str2[i])
      printf("str1 < str2");
   else
      printf("str1 = str2");
   return (0);
}
```

- **String Uppercase**
  - The strupr( ) function is used to converts a given string to uppercase.
  - Syntax: **strupr(string);**

Illustration of strupr () function	Output
#include <stdio.h> #include <string.h> #include<conio.h> void main() { char str1[20]="hello"; strupr(str1); printf("Uppercase =%s",str1); getch(); }	Uppercase = HELLO

- **String Lowercase**
  - The strlwr( ) function is a built-in function in C and is used to convert a given string into lowercase.
  - Syntax: **strlwr(string);**

Illustration of strlwr () function	Output
#include <stdio.h> #include <string.h> #include<conio.h>	Lowercase = hello

```
void main()
{
   char str1[20]="HELLO";
   strupr(str1);
   printf("Lowercase  =%s",str1);
   getch();
}
```

Additional C Programs Using Strings

Write a C Program to check whether the given string is palindrome or not without using string functions	Output

```
#include<stdio.h>
int main()
{
   char a[100],b[100];
   int len=0;
   int i=0,e,c=0;
   printf("Enter a string\n");
   scanf("%s",a);
   while(a[i]!='\0')
   {
      len++;
      i++;
   }
   e=len-1;
   for(i=0;i<len;i++)
   {
      b[i]=a[e];
      e--;
   }
for(i=0;i<len;i++)
   {
      if(a[i]==b[i])
      c++;
   }
   if(c==len)
   printf("%s is a palindrome",a);
   else
   printf("%s is not a palindrome",a);
```

```
Enter a string
madam
madam is a palindrome
```

```
    return 0;
}
```

Write a C Program to check whether the given string is palindrome or not using string functions

Output

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

int main()
{
  char a[100], b[100];
  printf("Enter a string to check if it is a
palindrome\n");
  gets(a);
  strcpy(b, a);
  strrev(b);
  if (strcmp(a, b) == 0)
    printf("The string is a palindrome.\n");
  else
    printf("The string isn't a palindrome.\n");
  return 0;
}
```

```
Enter a string to check if it is a palindrome
madam
The string is a palindrome.
```

Write a C Program to calculate the occurrence of the given character in the string

Output``

```
#include <stdio.h>
int main()
{
   char str[1000], ch;
   int i, frequency = 0;
   printf("Enter a string: ");
   gets(str);
   printf("Enter a character to find the frequency: ");
   scanf("%c",&ch);
   for(i = 0; str[i] != '\0'; ++i)
   {
       if(ch == str[i])
         frequency++;
   }
   printf("Frequency of %c = %d", ch, frequency);
   return 0;
}
```

G	O	O	D		M	O	R	N	I	N	G	total	
	1	2				3						3	

```
Enter a string: Good Morning
Enter a character to find the frequency: o
Frequency of o = 3
```

Write a program in C to count number of Uppercase and Lower case characters in the given string	**Output**
#include <stdio.h> int main() { int upper=0, lower=0,others=0; char ch[80]; int i; printf("Enter the string:\n"); gets(ch); i=0; while(ch[i]!='\0') { if(ch[i]>='A' && ch[i]<='Z') upper++; else if(ch[i]>='a' && ch[i]<='z') lower++; else others++; i++; } printf("\nUppercase Letters: %d",upper); printf("\nLowercase Letters: %d",lower); printf("\nOther Characters: %d",others); return 0; }	Enter the string: Good Morning Uppercase Letters: 2 Lowercase Letters: 9 Other Characters: 1
C Program to count number of characters and spaces in the given string	**Output**
#include <stdio.h> #include<string.h> int main() { char str[81]; int spaces=0,chars=0; puts("Enter a string :"); gets(str); for(int i=0;str[i]!='\0';i++) if(str[i] ==' ') spaces++; else chars++;	Enter a string : God is great Number of spaces 2 Number of characters 10

```
    printf("Number of spaces %d",spaces);
    printf("\nNumber of characters %d",chars);

}
```

C-Program to count the number of vowels, consonants, digits and spaces

```
#include <stdio.h>
int main()
{
    char line[150];
    int i, vowels, consonants, digits, spaces;
    vowels =  consonants = digits = spaces = 0;
    printf("Enter a line of string: ");
    scanf("%[^\n]", line);
    for(i=0; line[i]!='\0'; ++i)
    {
        if(line[i]=='a' || line[i]=='e' || line[i]=='i' ||
           line[i]=='o' || line[i]=='u' || line[i]=='A' ||
           line[i]=='E' || line[i]=='I' || line[i]=='O' ||
           line[i]=='U')
        {
            ++vowels;
        }
        else if((line[i]>='a'&& line[i]<='z') ||
(line[i]>='A'&& line[i]<='Z'))
        {
            ++consonants;
        }
        else if(line[i]>='0' && line[i]<='9')
        {
            ++digits;
        }
        else if (line[i]==' ')
        {
            ++spaces;
        }
    }
    printf("Vowels: %d",vowels);
    printf("\nConsonants: %d",consonants);
    printf("\nDigits: %d",digits);
    printf("\nWhite spaces: %d", spaces);
    return 0;
}
```

Output

```
Enter a line of string: Welcome to C programming
Vowels: 7
Consonants: 14
Digits: 0
White spaces: 3
```

Sri Eshwar College of Engineering
Coimbatore- 641 202

Question Bank

Part-A

1. What is an array? (Jan 2009, 2014)

- An array is a group of similar data types stored under a common name.
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Example:

  int a[10];

  Here a[10] is an array with 10 values.

2. List the characteristics of an array. (Jan 2013)

- All the elements of an **array share the same name**, and they are distinguished from one another with the help of an element number.
- The **element number** in an array plays major role for calling each element.
- Any particular element of an array can be modified separately without **disturbing other elements.**
- All elements of array are stored in the **continuous memory location.**
- The **size** of an array must be a ***constant integer value.***

3. What is the difference between an array and pointer?

Array	Pointer
• Array allocates space automatically.	• Pointer is explicitly assigned to point to an allocated space.
• It cannot be resized.	• It can be resized using realloc ().
• It cannot be reassigned	• Pointers can be reassigned.
• sizeof (array name) gives the number • of bytes occupied by the array.	• sizeof(pointer name) returns the number of bytes used to store the pointer variable.

4. How to initialize an array?

- You can initialize array in C either one by one or using a single statement as follows:
- double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
- The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

5. Why is it necessary to give the size of an array in an array declaration?

- When an array is declared, the compiler allocates a base address and reserves enough space in the memory for all the elements of the array.

- The size is required to allocate the required space. Thus, the size must be mentioned.

6. Define Strings.

Strings:

- The group of characters, digit and symbols enclosed within quotes is called as String (or) character arrays.
- Strings are always terminated with '\0' (NULL) character.
- The compiler automatically adds '\0' at the end of the strings.
- Example:
  - char name[]={'C','O','L','L','E','G','E','E','\0'};

7. Mention the various String Manipulation Functions in C.

S.NO.	FUNCTION & PURPOSE
strcpy(s1, s2);	Copies string s2 into string s1.
strcat(s1, s2);	Concatenates string s2 onto the end of string s1.
strlen(s1);	Returns the length of string s1.
strcmp(s1, s2);	Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
strchr(s1, ch);	Returns a pointer to the first occurrence of character ch in string s1.
strstr(s1, s2);	Returns a pointer to the first occurrence of string s2 in string s1.

8. Define one dimensional Array.

- One dimensional array can be defined as the collection of data item that can be stored under a one variable name using only one subscript. Ex. int a[10];

9. Define Two Dimensional Arrays.

- Two dimensional arrays can be defined as an array with two subscripts. A two dimensional array enables us to store multiple row of elements.
- Ex. int a[10][10]

10. What are the limitation of one-dimensional and two dimensional arrays?

The limitations of one dimensional array are

a. There is no easy method to initialize large number of array elements
b. It is difficult to initialize selected array element

The limitations of two dimensional arrays are

a. Deletion of any element from an array is not possible
b. Wastage of memory when it is specified in large array size

11. Why array elements must be initialized?

- After declaration array elements must be initialized otherwise it holds garbage value. There are two types of initialization
- **Compile time initialization(initialization at the time of declaration)**
  - int a[5]={12,34,23,56,12};
- **Run time initialization (when the array has large number of elements it can be initialized at run time)**
  - for(i=0;i<10;i++)
  - scanf("%d",&a[i]);

12. How strings are declared and initialized in C?

Strings are declared as a array of characters Syntax: char string_name[size]; Example: char text[30];	Initialization of strings char name[6]="RAMKI"; char city[]={„B","O","M","B","A","Y","\0"};

13. Write the syntax and example for two dimensional array in C?

- Synatx: data-type arr_name[num_of_rows][num_of_col];

Array declaration syntax: data_type arr_name [num_of_rows][num_of_column]; **Array initialization syntax:** data_type arr_name[2][2] = {{0,0},{0,1},{1,0},{1,1}}; **Array accessing syntax:** arr_name[index];	**Integer array example:** int arr[2][2]; int arr[2][2] = {1,2, 3, 4}; arr [0] [0] = 1; arr [0]]1] = 2; arr [1][0] = 3; arr [1] [1] = 4;

14. Write example code to declare 2D array? [June 2014]

```
#include<stdio.h>

#include<conio.h>

void main()

{

int i,j;

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

clrscr();

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

```
{
for(j=0;j<2;j++)
{
printf("%d ",a[i][j]);
}
printf("\n");
}
getch();
}
```

15. Design a C Program to compare any two strings

```
#include <stdio.h>
#include <string.h>
int main()
{
char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd";
int result;
result = strcmp(str1, str2);
printf("strcmp(str1, str2) = %d\n", result);
result = strcmp(str1, str3);
printf("strcmp(str1, str3) = %d\n", result);
return 0;
}
```


Part-B

1. **Discuss about any eight built in functions of string. (Jan 2013)**
2. **Briefly explain the various string handling functions in C. (Jan 2010)**
3. **Write C program to sort the given set of numbers in ascending order. (Jan 2013, 2011)**

4. **Write C program to find addition of two matrices. (Jan 2013, 2014)**
5. **Write C program to find multiplication of two matrices.**
6. **Write a C program to reverse a given string. (May 2011)**
7. **Write a C program to convert the given string from lowercase characters to uppercase character and uppercase to lowercase. (May 2011)**
8. **Write a C program to find largest and smallest number in the given array. (May 2010)**
9. **Write a C program to concatenate two strings. (May 2009)**
10. **Write a C program to demonstrate one dimensional, two dimensional and multidimensional arrays.**
11. **Write a C Program to count the number of vowels, Consonants, digits and white space in a string.**

Sri Eshwar College of Engineering
Coimbatore- 641 202

Unit IV

FUNCTIONS AND POINTERS

Functions: Built-in Functions, User defined functions – Function Prototypes –Recursion – Command Line Argument -Arrays and Functions – Strings and Functions. Pointers: Declaration – Pointer operators – Pointer arithmetic -Passing Pointers to a Function – Pointers and one dimensional arrays - Dynamic Memory Allocation

FUNCTIONS

- **Definition**
  - **Self-containing block of one or more statements or a sub-program**which is designed for a particular task is called **function**.
  - Every C program starts with a function main ().
  - The main () function calls another function to share the work.
  - C function contains set of instructions enclosed by "{ }" which performs specific operation in a C program.
- **Advantages**
  - Reduces the size of the program
  - Enabling code reuse
  - Improves re-usability
  - Easy to debug and test
  - Better readability
  - Makes program development easy.

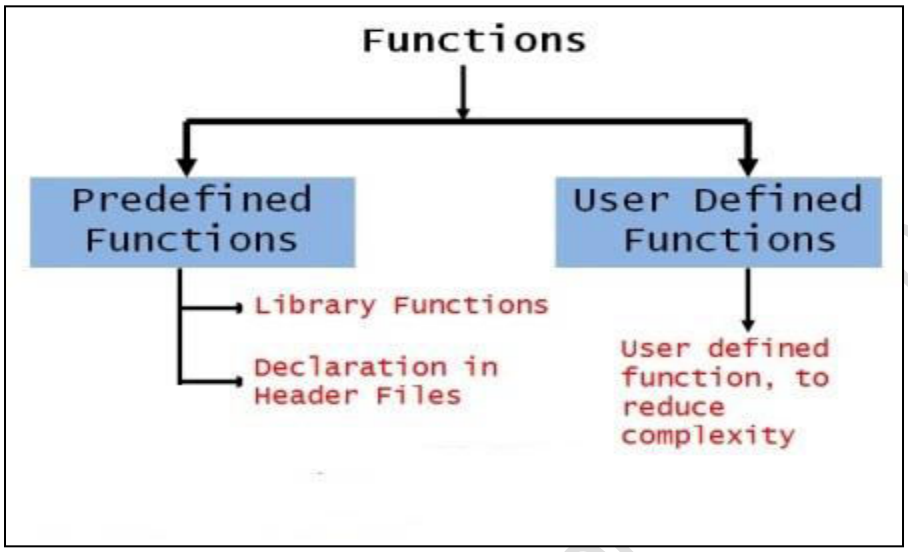
CLASSIFICATION OF FUNCTIONS

- In C Language functions are classified into two types. They are
  - Built-in functions
  - User defined functions

Built-in functions (Library functions)

- The library functions are **pre-defined functions.**
- These functions are already defined in header files.

- Appropriate header files are included in the program to use these functions.
- Example: printf(),scanf(),pow(),etc.,



User defined function

- User-defined functions are **defined by the user at the time of writing a program.**
- User can understand the internal working of the function and can change or modify them.

- **Elements of user defined function**
  - Function declaration (or) Function prototype
  - Function call
  - Function definition
- **Function declaration / Function prototype**
  - All identifiers in C need to be declared before they are used.
  - This is true for functions as well as variables.
  - For functions declaration needs to be before the first call of the function.
  - A function declaration is also known as function prototype.
  - It consists of four parts
    - Return type
    - Function name
    - Parameter list
    - Terminating semicolon

Syntax:

***return_typefunction_name*(*arguments list*);**
Example:
int area(int length, int breath); *where,* *int – return type* *area – function name* *intlength,int breath – parameters that the functions accepts*

Function Definition

Function definition consists of two parts

1. Function header
2. Function body

Function Definition
return_typefunction_name(list of parameters) // **Function header** { declaration executable statements return statement; } // **Function body**

Function header

1. **Return type**
   - Specifies the type of value that the function is expected to return to the calling function.
2. **Function name**
   - The function name is any valid C identifiers.
3. **Parameter list**
   - The parameters are also known as **arguments.**
   - It is a comma separated list of variables of function, enclosed with parentheses.

- The number of arguments and the order of arguments in the function header must be same as that given in the function declaration statement.
- There are two types of parameters.

a) Actual parameters

- The **parameters in the calling program or the parameters in the function call** are known as actual parameters.

b) Formal parameters

- The parameters in the **called program or the parameters in the function header** are known as formal parameters.

Example

```
#include<stdio.h>
int main()
{
inta,b,c;
printf("Enter two numbers");
scanf("%d%d",&a,&b);
c=add(a,b);
printf("The addition of two numbers is %d",c);
return 0;
}
int add(intx,int y)
{
int z;
z=x+y;
return(z);
```

Actual Arguments

Function call

Formal arguments

Function Header

}

Where

a,b are the actual arguments

x,y are the formal arguments

- **Function call or Function Revocation**
  - A function can be called by specifying its name, followed by a list of arguments enclosed in parentheses and separated by commas.
  - Syntax

function_name (variable1, variable2…);

FUNCTION TYPES

1. Function with no arguments and no return values
2. Function with no arguments and with a return value
3. Function with arguments and no return value
4. Function with arguments and with a return value

Function with no arguments and no return values

```
void sum() // User Defined Function
{
int x, int y;
printf("Enter the Value of  x:")
scanf("%d",&x);
printf("Enter the Value of  y:")
scanf("%d",&y);
printf("Sum of  x and y is:%d",x+y);
}
```

```
void main() // Main Function
{
sum(); // Function Call
}
```

Output:

Enter the Value of X: 10 Enter the Value of Y: 10 Sum of X and Y is: 20
Explanation: In the above program , • **User defined function** – sum() • **arguments** – no • **return value -** void • sum() function is written to perform addition operation on x and y inputs from the user. • sum() function is called by the main() function.

Function with no arguments and with a return value

int sum() // User Defined Function { int x, int y; printf(“Enter the Value of x:”) scanf(“%d”,&x); printf(“Enter the Value of y:”) scanf(“%d”,&y); return (x+y); }	void main() // Main Function { int result; result=sum(); // Function Call printf(“Sum of x and y is:%d”,result); }
Output: Enter the Value of X: 10 Enter the Value of Y: 10 Sum of X and Y is: 20	

Explanation: In the above program , • **User defined function** – sum() • **arguments** – no • **return value -** int • sum() function is written to perform addition operation on x and y inputs from the user and return the result to main() function. • sum() function is called by the main() function

Function with arguments and no return value

void sum(int x, int y) // User Defined Function { int result; result=x+y; printf("Sum is:%d",result); }	void main() // Main Function { int a, int b; printf("Enter the First Value:") scanf("%d",&a); printf("Enter the Second Value :") scanf("%d",&b); sum(a,b)//Function Call }
Output: Enter the First Value: 10 Enter the Second Value: 10 Sum is: 20	
Explanation: In the above program ,	

- **User defined function** – sum()
- **arguments** – 2 arguments x and y
- **return value -** void
- sum() function is written to perform addition operation on x and y inputs received from the main() function.
- sum() function is called by the main() function with two arguments.

Function with arguments and with a return value

```
int sum(int x, int y)
// User Defined Function
{
return (x+y);
}
```

```
void main() // Main Function
{
int a, intb,result;
printf("Enter the First Value:")
scanf("%d",&a);
printf("Enter the Second Value :")
scanf("%d",&b);
result=sum(a,b)//Function Call
printf("Sum is:%d",result);
}
```

Output:

Enter the First Value: 10

Enter the Second Value: 10

Sum is: 20

Explanation:

In the above program ,

- **User defined function** – sum()
- **arguments** – 2 arguments x and y

- **return value -** int
- sum() function is written to perform addition operation on x and y inputs received from the main() function and return the result to main() function.
- sum() function is called by the main() function with two arguments.

PARAMETER PASSING METHODS

- In C Programs there are two ways to pass parameters to a function. They are
  I. Pass by Value
  II. Pass by reference
- **Pass by value**
  - The method of passing arguments by value is also known as **call by value.**
  - In this method, the values of actual arguments are **copied** to the formal parameters of the function.
  - If the arguments are passed by value, the changes made in the values of formal parameters inside the called function are **not reflected back** to the calling function

Illustration of call by value	Output
#include <stdio.h> int fun(int,int); int main() { int x=10,y=20; fun(x,y); printf("x=%d, y=%d",x,y); return 0; } int fun(inta,int b) { a=20; b=10; }	x=10, y=20
C Program to interchange the values of two variables using pass by value	Output

#include <stdio.h> void interchange(int,int); int main() { int num1=50,num2=70; interchange(num1,num2); printf("\n Value after interchange :"); printf("\nNumber 1 :%d",num1); printf("\nNumber2 :%d",num2); return 0; } void interchange(int number1,int number2) { int temp; temp=number1; number1=number2; number2=temp; }	Value after interchange : Number 1 :50 Number2 :70

- **Advantages**
  - It protects the value of the variable from alterations within the function.
- **Disadvantages**
  - Copying data consumes additional storage space.
  - In addition, it takes a lot time to copy thereby resulting in performance penalty, especially if the function is called many times.
- **Pass by reference**
  - The method of passing arguments by **address or reference** is also known as call by reference.
  - In this method the **addresses** of the actual arguments are passed to the formal parameters of the function.
  - If the arguments are passed by reference, the changes made in the values of formal parameters in the called function are **reflected back**to the calling function.

Illustration of call by reference	Output
#include <stdio.h> int fun(int*,int*); int main() { int x=10,y=20; fun(&x,&y); printf("x=%d, y=%d",x,y); }	x=20, y=10

int fun(int *ptr1,int *ptr2) { *ptr1=20; *ptr2=10; }	
Program for swapping two numbers using Pass by reference	Output
#include <stdio.h> void interchange(int*,int*); int main() { int num1=50,num2=70; interchange(&num1,&num2); printf("\n Value after interchange :"); printf("\nNumber 1 :%d",num1); printf("\nNumber2 :%d",num2); return 0; } void interchange(int *number1, int *number2) { int temp; temp=*number1; *number1=*number2; *number2=temp; }	Value after interchange : Number 1 :70 Number2 :50

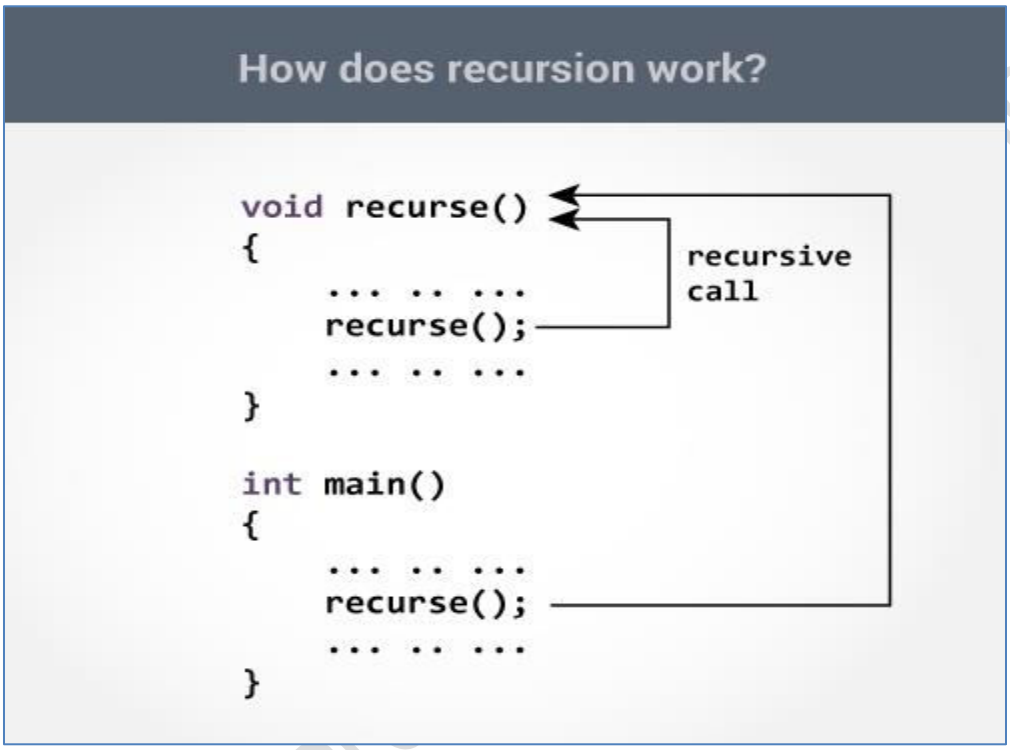
- **Advantages**
  - Since arguments are not copied into new variables, it provides greater time and space efficiency.
  - The called function can change the value of the argument and the change it reflected in the calling function.
  - A return statement can return only one value. In case we need to return multiple values, pass those arguments by reference.

RECURSION

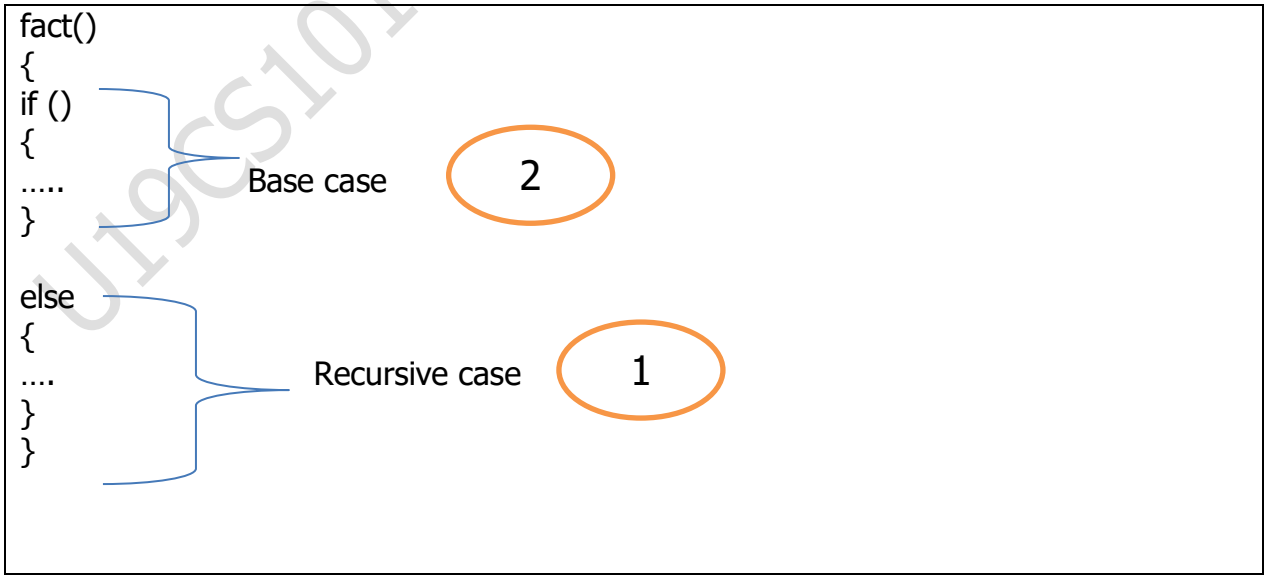
- In C, a function can **call itself.** In this case, the function is said to be recursive.
- **Example**

```
void recursion(){
recursion();/* function calls itself */
```

```
}

int main(){
recursion();
}
```



Basic Structure of Recursion



TYPES OF RECURSION

1. Direct Recursion
2. Indirect recursion
3. Tail recursion
4. Non-tail recursion

Direct recursion

- A function is called direct recursive if it calls the **same function again**.

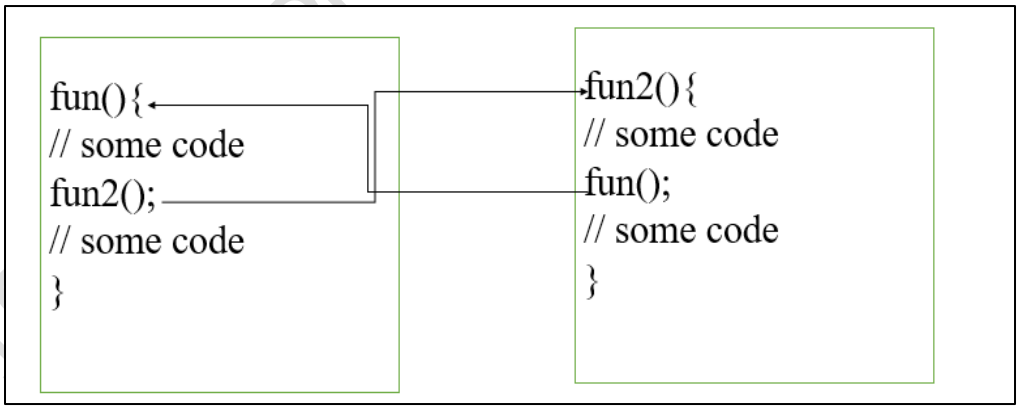
Structure of direct recursion

```
fun(){
// some code
fun();
// some code
}
```

Indirect Recursion

- A function is said to be **indirectly recursive**if it contains a call to another function which ultimately calls it.

Structure of Indirect recursion



Write a program to print from 1 to 10in such a way that when number is odd, add 1 and when number is even subtract 1	Output
#include <stdio.h> void odd();	2 1 4 3 6 5 8 7 10 9

void even(); int n=1; int main() { odd(); } void odd(){ if(n<=10){ printf("%d ",n+1); n++; even(); } return; } void even(){ if(n<=10){ printf("%d ",n-1); n++; odd(); } return; }	

Tail Recursion

Definition

- A Recursive function is said to be **tail recursive** if the **recursive call is the last thing done by the function.** There is no need to keep record of the previous state.

Example for Tail Recursion	**Output**
#include<stdio.h> void fun(int n) { if (n==0) return; else printf("%d",n); return fun(n-1); } int main() { fun(3);	321

return 0; }	

Non-Tail Recursion

Definition

- A Recursive function is said to be **non-tail recursive** if the recursive call is not the last thing done by the function.
- After returning back, there is something left to evaluate.

Example for Non- Tail Recursion

```
#include<stdio.h>
void fun(int n) {
if (n==0)
return;
else
return fun(n-1);
printf("%d",n);
}
int main()
{
fun(3);
return 0;
}
```

EXAMPLE PROGRAMS USING RECURSION

Write a C Program to compute factorial of a number using recursive functions	**Output**
#include<stdio.h> #include<conio.h> int factorial (int); void main() { int n, f; printf("enter a number"); scanf("%d",&n); printf("answer ="); printf("%d",factorial (n)); getch();	enter a number5 answer =120

```
}
int factorial(int n)
{
if ( n == 1)
return 1;
else
return (n*factorial(n-1) );
}
```

Write a C Program to findFibonacci series using recursive functions

- Fibonacci series is a series of numbers where the next term is the sum of previous two terms
- The function calls itself is called recursion
- Formula: F(n)=F(n-1)+F(n-2)

F0	F1	F2	F3	F4	F5
0	1	1	2	3	5

```
#include <stdio.h>
int Fibonacci(int);
int main()
{
intn,i=0,res;
printf("Enter the number of terms\n");
scanf("%d",&n);
printf("Fibonacci series\n");
   for(i=0;i<n;i++)
   {
      res=Fibonacci(i);
printf("%d\t",res);

   }
   return 0;
}
int Fibonacci(int n)
{
   if(n==0)
   return 0;
   else if(n==1)
   return 1;
   else
   return (Fibonacci(n-1)+Fibonacci(n-2));
}
```

```
Enter the number of terms
5
Fibonacci series
0       1       1       2       3
```

C Program to compute GCD of two numbers using recursion *[Euclid's algorithm]*	**Output**
#include <stdio.h> intComputeGCD(int,int); int main() { intz,x,y; printf("Enter the two numbers:\n"); scanf("%d%d",&x,&y); z=ComputeGCD(x,y); printf("The GCD of %d and %d is %d",x,y,z); return 0; } intComputeGCD(inta,int b) { if(b==0) return a; else return ComputeGCD(b,a%b); }	Enter the two numbers: 64 48 The GCD of 64 and 48 is 16
C Program to calculate exponent using recursive functions	**Output**
#include <stdio.h> int Calculate(intb,int p); // Function prototype void main() { intbase,power; int result; printf(" Enter the base:"); scanf("%d",&base); printf(" Enter the power:"); scanf("%d",&power); result=Calculate(base,power); //calls the Calculate function printf(" %d power %d is =%d",base,power,result); } int Calculate(intb,int p) { int results=1; if(p == 0) return results; else	Enter the base:2 Enter the power:4 2 power 4 is =16

results=b*(Calculate (b,p-1)); // Calculate () function calls itself recursively return results; }	

POINTERS

- Every variable in C Language has a name and value associated with it.
- When a variable is declared, a specific block of memory is allocated to hold the value of the variable.
- The size of the allocated block depends on the type of the variable.
- Pointer in C Language is a **variable which contains the address of another variable.**
- A pointer in C is used **to allocate memory dynamically i.e. at run time**.
- The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

Advantages of pointers

- Pointers are more efficient in **handling arrays and data tables.**
- Pointers can be used to return multiple values from a function via function arguments.
- Pointers allow **dynamic memory management**.
- Pointers provide efficient tool for manipulating dynamic data structure such as structures, linked list, queues, stacks and trees.
- Pointers **reduce length and complexity** of programs.
- Pointers increase the **execution speed** and reduce the program execution time.

POINTER DECLARATION

data_type*var_name;

- Here, **data_type**is the pointer's base type; it must be a valid C data type and **var_name**is the name of the pointer variable.
- The asterisk * is used to designate a variable as a pointer.
- Here are some pointer declarations
    - int *ip; /* pointer to an integer*/

- double *dp; /* pointer to a double*/
- float *fp; /* pointer to a float */
- char *ch /* pointer to a character */

- int*ip tells the compiler that ip is a pointer variable and it will be used only for storing the address of the integer valued variables.
- Consider

```
int x=10; -------- (1)
int *ptr; ---------- (2)
ptr=&x;------------ (3)
```

- Here line (2) says that ptr is pointer variable which can hold the address of an integer variable.
- Line (3) says that ptr will hold the address of x (for example, FFFD)
- Hence, the content of ptr is FFFD
- *ptr will give the value of 10 which is the content of address pointed by ptr i.e., the value at address FFFD (the value of x i.e., 10.)

POINTER OPERATORS

Symbol	Name	Description
& (ampersand sign)	Address of operator	Give the address of a variable
* (asterisk sign)	Indirection operator	Gives the contents of an object pointed to by a pointer.

- In order to create pointer variable we use '*' operator and to find the address of variable we use "&" operator.
- 'Value at' operator is also called as '**Indirection Operator'.**

Example 1	Output with explanation
#include <stdio.h> int main() { int *p,q; q=50;	50 p: is a pointer variable that holds the address of a integer &q: gives address of the memory location whose name is 'q'

/* address of q is assigned to p*/ p=&q; /* display q's value using p variable */ printf("%d",*p); return 0; }	p=&q :p holds the address of an integer q *p gives value at address specified by &q *p=*(&q) =*(Address of Variable 'q') =*(1000) =Value at the address 1000 =50
Example 2	Output
#include <stdio.h> int main() { int n=20; printf("\nthe address of n is %u",&n); printf("\nthe value of n is %d",n); printf("\n the value of n is %d",*(&n)); }	The address of n is 3952800492 The value of n is 20 The value of n is 20
C Program to assign vales to the variables using pointer variables and modify their values	Output
#include <stdio.h> int main() { int n,*p; p=&n; *p=10; printf("\n *p=%d",*p); printf("\n n=%d",n); *p=*p+1; printf("\n *p=%d",*p); printf("\n n=%d",n); return 0; }	*p=10 n=10 *p=11 n=11
C program to print the size occupied by a pointer variable	Output
#include <stdio.h> int main() { int *pnum; char *pch; float *pfnum; double *pdnum;	size of integer pointer = 2 size of character pointer = 2 size of float pointer = 2 size of double pointer =2 size of long pointer =2

long *plnum; printf("\n size of integer pointer = %d",sizeof(pnum)); printf("\n size of character pointer = %d",sizeof(pch)); printf("\n size of float pointer = %d",sizeof(pfnum)); printf("\n size of double pointer =%d",sizeof(pdnum)); printf("\n size of long pointer =%d",sizeof(plnum)); }	
C Program to illustrate pointer expressions	**Output**
#include <stdio.h> int main() { int n1=2,n2=3,sum,mul,div; int *p1,*p2; p1=&n1; p2=&n2; sum=*p1+*p2; mul=sum* *p1; div=9+*p1/ *p2-30; printf("\n sum= %d mul=%d div=%d",sum,mul,div); }	sum= 5 mul=10 div=-21

NULL pointers

- A null pointer is a special pointer that does not **point anywhere**.
- It does not hold the address of any object or function.
- It has numeric value 0.

int *ptr=NULL

- When a null pointer is compared with a pointer to any object or a function the result of comparison is always false.

- **Dereferencing a null pointer leads to runtime error**.

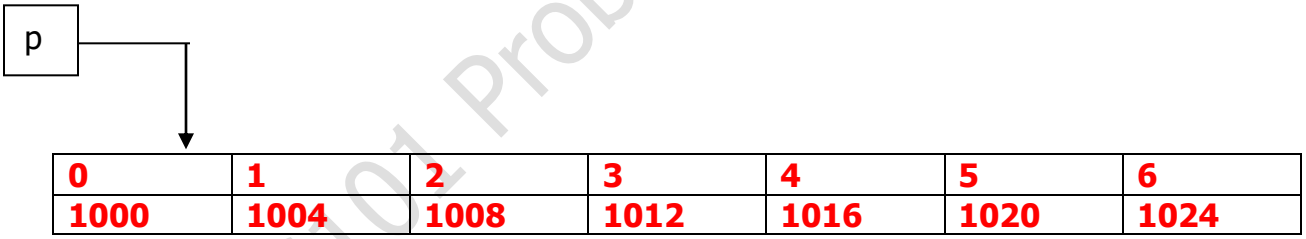
Generic pointers

- A pointer variable that has void as its data type is called generic pointer.
- void *gp;
- It can be **used to point variables of any type** using type cast.
- For example
  - (int*)gp- gp is pointing to integer
  - (char*)gp- gp is pointing to character

Pointer Arithmetic

- **C Pointer Addition**
  - We can add a value to the pointer variable. The formula for adding value to a pointer is given below:

new_address=current_address+(number*sizeof(data type))

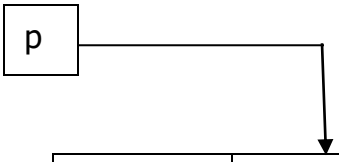


p=p+1

new_address=current_address+(number*sizeof(datatype))

= 1000+(1*4)

=1004



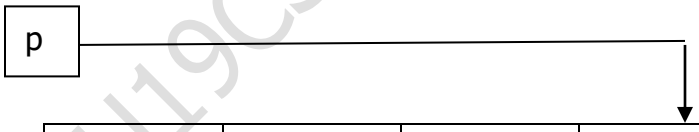
0	1	2	3	4	5	6
1000	1004	1008	1012	1016	1020	1024

- For 32 bit int variable, it will add 2*number
- For 64 bit int variable, it will add 4*number

C Program to illustrate pointer addition	Output
#include<stdio.h> int main(){ int number=50; int *p; p=&number; printf("Address of p variable is %u \n",p); p=p+1; printf("After adding 1: Address of p variable is %u \n",p); return 0; }	Address of p variable is 1383092548 After adding 1: Address of p variable is 1383092552

- **C Pointer Subtraction**
  - Like pointer addition, we can subtract a value from the pointer variable.
  - Subtracting any number from a pointer will give an address.
  - The formula of subtracting value from the pointer variable is given below:

new_address=current_address-(number*sizeof(data type))



0	1	2	3	4	5	6
1000	1004	1008	1012	1016	1020	1024

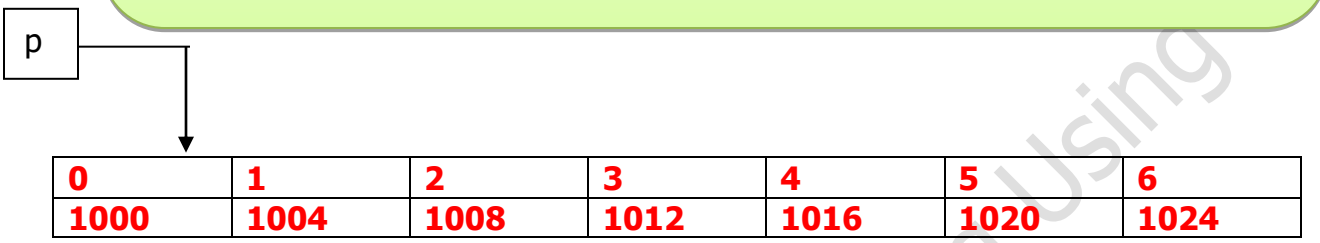
p=p-3

Moving the pointer in three position in REVERSE direction

p=p-3 = &a[3-3]= &a[0]

p=1012-3*4

=1012-12



- **Incrementing Pointer in C**
  - "Incrementing a pointer increases its value by the number of bytes of its data type.
  - Let ptr be an integer pointer which points to the memory location 5000 and size of an integer variable is 4-bytes.
  - Now, when we increment pointer ptr

ptr++;

  - Will point to memory location 5004 because it will jump to the next integer location which is 4 bytes next to the current location.
  - Incrementing a pointer is not same as incrementing an integer value.
  - The Rule to increment the pointer is given below:
    - new_address= current_address + i * size_of(data type)
- **Decrementing Pointer in C**
  - If we decrement a pointer, it will start pointing to the previous location.
  - The formula of decrementing the pointer is given below:
    - new_address= current_address - i * size_of(data type)

C Program to illustrate pointer increment/decrement	**Output**
#include <stdio.h> int main() { int m=5,n=10; int *p1,*p2; p1=&m; p2=&n;	p1=3449840440 p2=3449840444 p1++=3449840444 p2--=3449840440

printf("p1=%u\n",p1); printf("p2=%u\n",p2); p1++; printf("p1++=%u\n",p1); p2--; printf("p2--=%u\n",p2); return 0; }	
Pointer Post Increment	**Output**
#include <stdio.h> int main() { int a[]={5,16,7,89,45,32,23,10}; int *p=&a[0]; printf("%d",*(p++)); printf("%d",*p); return 0; }	5 16
Pointer Pre increment	**Output**
#include <stdio.h> int main() { int a[]={5,16,7,89,45,32,23,10}; int *p=&a[0]; printf("%d",*(++p)); return 0; }	16

Pointers & Arrays

- We know that array elements are located continuously inn memory and array name is really pointer to the first element in the array.
- Consider the following

```
int array[]={1,20,32,12,45,100,56,78};
```

- Here we have an array consisting 8 integers
- We refer to each of these integers by means of subscript to **array,** i.e. using array [0] through array [7].

- But we could alternatively access them via a pointer as follows

```
int *ptr
ptr=&array[0] /* point our pointer at the first integer in our array
```

C Program to print base address of an array using pointers	**Output**
#include <stdio.h> int main() { int a[]={5,16,7,89,45,32,23,10}; int *p; p=a; printf("Base address of 'a' is =%u\n",p); return 0; }	Base address of 'a' is =2387547792
Simple C Program to illustrate the relationship between array and pointers	**Output**
#include <stdio.h> int main() { int array[]={1,20,32,12,45,100,56,78}; int *ptr; inti; ptr=&array[0]; printf("\n"); for(i=0;i<8;i++) { printf("array[%d]=%d",i,array[i]); printf("\tptr+%d=%d\n",i,*(ptr+i)); } }	array[0]=1 ptr+0=1 array[1]=20 ptr+1=20 array[2]=32 ptr+2=32 array[3]=12 ptr+3=12 array[4]=45 ptr+4=45 array[5]=100 ptr+5=100 array[6]=56 ptr+6=56 array[7]=78 ptr+7=78
C Program to sort the names using pointers	Output
#include <stdio.h> int main() { char *fruit[]={"apricot","banana","pineapple","apple","persimmon","pear","blueberry"}; char*tmp; inta,b,x;	apricot banana pineapple apple persimmon pear blueberry

```
for(a=0;a<6;a++)
for(b=a+1;b<7;b++)
if(*(fruit+a)>*(fruit+b))
{
tmp=*(fruit+a);
   *(fruit+a)=*(fruit+b);
   *(fruit+b)=tmp;
}
   for(x=0;x<7;x++)
   puts(fruit[x]);

}
```

Dynamic Memory Allocation in C

- The concept of dynamic memory allocation in c language enables the C programmer to allocate memory at runtime.
- Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

1. malloc()
2. calloc()
3. realloc()
4. free()

Static memory allocation	Dynamic memory allocation
Memory is allocated at compile time.	Memory is allocated at run time.
Memory can't be increased while executing program.	Memory can be increased while executing program.
Used in array.	Used in linked list.

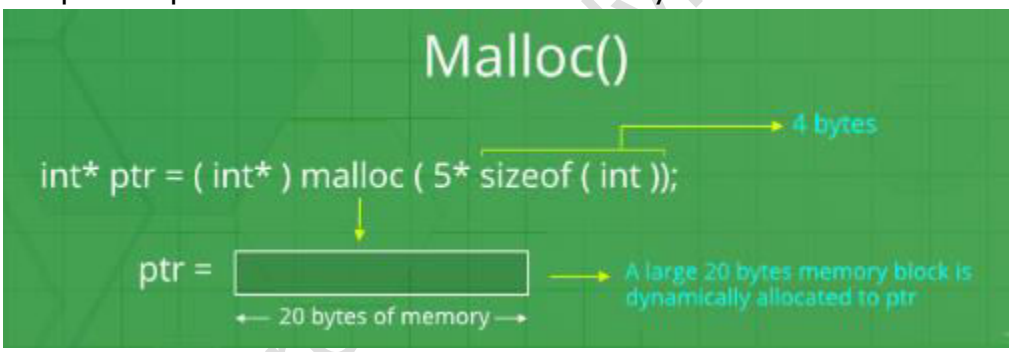
malloc()	allocates requested size of bytes and returns a void pointer pointing to the first byte of the allocated space
calloc()	allocates space for an array of elements, initialize them to zero and then returns a void pointer to the memory
realloc()	Reallocates the memory occupied by malloc() or calloc() functions.
free()	Frees the dynamically allocated memory.

malloc() function in C

- Themallocfunctionallocates a block of memory that contains the number of bytes specified in its parameter.
- It returns a void pointer to the first byte of the allocated memory.
- The prototype for malloc function is given below

```
void*malloc(size_t size);
```

- The type, size_t is defined in <stdlib.h> header file.
- The type is usually an unsigned integer.
- We can use **sizeof**operator to specify number of bytes to be allocated in malloc function.
- Example : ptr=(int*)malloc(100*sizeof(int));
- Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.



C Program to illustrate malloc () function	Output
#include <stdio.h> #include <stdlib.h> int main() { int* ptr; int n, i, sum = 0; n = 5; printf("Enter number of elements: %d\n", n); ptr = (int*)malloc(n * sizeof(int)); if (ptr == NULL) { printf("Memory not allocated.\n"); exit(0); }	Enter number of elements: 5 Memory successfully allocated using malloc. The elements of the array are: 1, 2, 3, 4, 5,

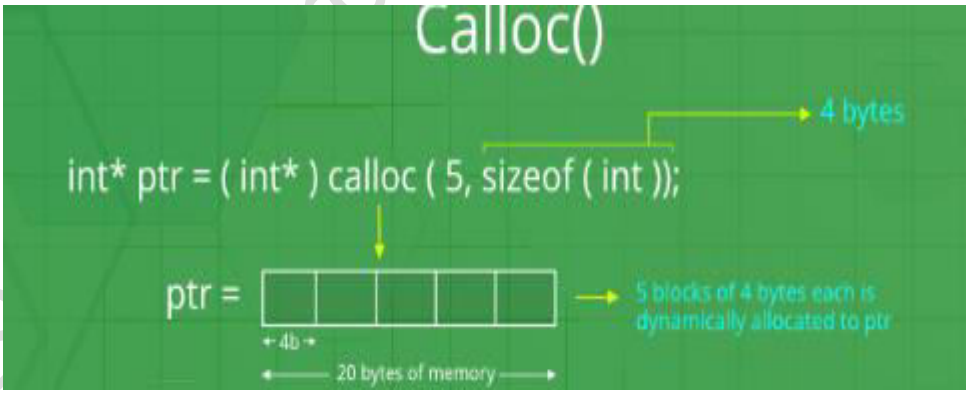
```
else {
printf("Memory successfully allocated using
malloc.\n");
for (i = 0; i< n; ++i) {
ptr[i] = i + 1;
}
printf("The elements of the array are: ");
for (i = 0; i< n; ++i) {
printf("%d, ", ptr[i]);
}
}
return 0;
}
```

calloc() function in C

- calloc () function is also like malloc () function. But calloc () initializes the allocated memory to zero. But, malloc() doesn't.
- The syntax is

```
void*calloc(size_tnitems, size_t size);
```

- Example :ptr = (float*) calloc(25, sizeof(float));
- This statement allocates contiguous space in memory for 25 elements each with the size of the float.



C Program to illustrate calloc () function	Output
#include <stdio.h> #include <string.h> #include <stdlib.h> int main() { char *mem_allocation;	Dynamically allocated memory content : Sri Eshwar

```
mem_allocation = calloc( 20, sizeof(char) );
if( mem_allocation== NULL )
{
printf("Couldn't able to allocate requested
memory\n");
}
else
{
strcpy( mem_allocation,"SriEshwar");
}
printf("Dynamically allocated memory content   :
%s\n", mem_allocation );
free(mem_allocation);
}
```

realloc() function in C

- If memory is not sufficient for malloc() or calloc(), one can reallocate the memory by realloc() function. In short, it changes the memory size.
- Let's see the syntax of realloc() function.

```
void*realloc(void *ptr, size_t size);
```

free() function in C

- The memory occupied by malloc() or calloc() functions must be released by calling free() function. Otherwise, it will consume memory until program exit.
- Syntax : free(ptr);

Command Line Arguments

- The arguments passed from command line are called command line arguments.
- These arguments are handled by main () function.
- To support command line argument, you need to change the structure of main () function as given below.

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

- argc is an integer value that specifies number of command line arguments.
- argv[] – pointer to a character array, this array has the actual values passed in the command line
  argv[0] – name of the program
  argv[1] – pointer to the first string passed in the command line

argv[argc-1] – pointer to the last string passed in the command line

Example Program

```
//commandline.c
#include<stdio.h>
int main(intargc, char *argv[])
{
printf("%d",argc);
printf("\n");
printf("%s",argv[0]);
printf("\n");
printf("%s",argv[1]);
printf("\n");
printf("%s",argv[2]);
return 0;
}
```

Command to run the program : ./commandline hi hello

Output:

```
3
./commandline
hi
hello
```


Part-A

1. **What is a function? (Nov 2014)**

- A function is a group of statements that together perform a task.
- Every C program has at least one function which is main(), and all the most trivial programs can define additional functions.

2. How will define a function in C?

- The general form of a function definition in C programming language is as follows
  return_typefunction_name (parameter list)
  {
  body of the function
  }

3. What does a function header and function body consist of?

A function definition consists of a. Function header b. Function body	The Function header consists of a. Return Type b. Function Name c. Parameters
The Function body consists of a. Declarations and statements necessary for performing the required task.	

4. What are the steps in writing a function in a program?

a. **Function Declaration/Prototype:** Every user-defined functions has to be declared before the main ().

b. **Function Callings:** The user-defined functions can be called inside any functions like main (), user-defined function.

c. **Function Definition:** The function definition block is used to define the user-defined functions with statements.

5. What is the need for functions? (Jan 2014)

a. Functions are self-contained block or sub program of one or more statements that performs a specific task.

b. It increases the modularity, reusability of a program.

6. What is the purpose of the main ()? (May 2009)
The main () invokes other functions within it. It is the first function to be called when the program starts execution.

7. What are the elements of user-defined function?
a. Function Definition.

b. Function Declaration.

c. Function call

8. List the advantages of user-defined function.
The advantages of user defined functions are as follows
a. The program will be easier to understand, maintain and debug.

b. Reusable codes that can be used in other programs

c. A large program can be divided into smaller modules. A large project can be divided among many programmers.

9. Is it better to use a macro or a function?
Macros are more efficient than function, because their corresponding code is inserted directly at the point where the macro is called. There is no overhead involved in using a macro like there is in placing a call to a function. Macros are generally small and cannot handle large, complex coding constructs. In cases where large, complex constructs are to handled, functions are more suited, additionally.

10.Classify the functions based on arguments and return values.
Depending on the arguments and return values, functions are classified into four types.
a. Function without arguments and return values.
b. Function with arguments but without return values.
c. Function without arguments but with return values.
d. Function with arguments and return values.

11. What are address operator and indirection operator? (Nov 2014)
a. The address operator (&) - It represents the address of the variable.
b. Indirection pointer (*) - When a pointer is dereferenced, the value stored at that address by the pointer is retrieved.

12. What is function prototyping? Why it is necessary? (May 2011)
Many built in functions can be used in C programs. The prototype of these functions is given in the respective header files. With the help of a function prototype, the compiler can automatically perform type checking on the definition of the function, which saves the time to delay the program.

13. What are actual parameters and formal parameters? (May 2015)
Actual Parameters: The parameters in the calling program or the parameters in the function call are known as actual parameters.
Formal Parameters: The parameters in the called program or the parameters in the function header are known as formal parameters.

14. State the advantage of using functions.
The advantages of using functions are as follows
a. Reduces the size of the program: The length of the source program can be reduced by using functions at appropriate places.

b. Avoids rewriting the code: As function can be called many times in the program.

c. Improves re-usability: Same function may be used later by many other programs.

d. Makes program development easy: Work can be divide among project members thus implementation can be completed in parallel.

e. Program testing becomes easy: Each function can be tested separately and it is easy to locate and isolate a faculty function for further investigation.

f. Increases program readability.

15.What is a recursive function?
If a function calls itself again and again, then that function is called Recursive function. This technique is known as recursion.

```
void recursion()
{
recursion(); /* function calls itself */
}
int main()
{
recursion();
}
```

16. State the advantage and disadvantage of recursive function.

Advantage of recursive function	**Disadvantage of recursive function**
a. Reduce unnecessary calling of function. b. Through recursion one can solve problem in easy way while its iterative solution is very big and complex.	a. It is very difficult to trace (debug and understand) b. Takes a lot of stack space. c. Uses more processor time.

17. What is a Pointer? How a variable is declared to the pointer? (Jan 2013, May 2009)
Pointer is a variable which holds the address of another variable.
Syntax: data_type *variable_name;
Ex.: int *x, a=5;
x=&a;

18. List the key points to remember about pointers in C.
The key points to remember about pointers in C are
a. The content of the C pointer always be a whole number i.e. address.
b. Always C pointer is initialized to null i.e., int *p=null.
c. The value of null pointer is 0.
d. The size of any pointer in 2 byte.

19. What are the uses of Pointers? (Jan 2014, May 2012, May 2010)
The uses of pointers are as follows
a. Pointers are used to return more than one value to the function

b. Pointers are more efficient in handling the data in arrays ·Pointers reduce the length and complexity of the program ·
c. They increase the execution speed
d. The pointers save data storage space in memory

20. **Distinguish between Call by value Call by reference. (May 2014)**

Call by Value / Pass by value	**Call by reference / Pass by reference**
Copy of original parameter is passed	Address of original parameter is passed
No effect on original parameter after modifying parameter in function	Original parameter gets affected if value of parameter changed inside function
Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

PART - B

1. Explain in detail about Function with example.

2. Explain function with and without arguments with example for each. (Jan 2014)

3. What is recursion? Give an example and explain in detail. (Jan 2014, Nov 2014)

4. Explain (Jan 2014, May 2014, Nov 2014)

(i) Function declaration

(ii) Call by reference; call by value with an example.

5. Write a function using pointers to add two matrices and to return the resultant matrix to the calling function. (May 2012)

6. Write a C program to find the factorial of a given number using function. (May 2014)

7. Write a C program to exchange the values of two variables using pass by reference. (May2014)

8. Write a C program to find the sum of the digits using recursive function. (May 2015)

9. Write a C program to using pointers to read in an array of integers and print its elements in reverse order. (May 2015)
10. Write a program to sort names using pointers
11. Explain the concept of dynamic memory allocation in C with example programs.

Additional C Programs on Functions& Pointers

C Program to calculate the area of circle using functions	Output
#include<stdio.h> const float PI= 3.14; float calcarea(float x); int main()	3

{ float x; scanf("%f",&x); printf("The area of the circle is %.2f",calcarea(x)); return 0; } float calcarea(float x) { return(PI*x*x); }	The area of the circle is 28.26
Maximum of three numbers using functions	Output
#include<stdio.h> intfindmaximum(inta,intb,int c) { int largest=0; if(a>b && a>c) largest=a; else if(b>a && b>c) largest=b; else largest=c; return largest; } int main() { inta,b,c; scanf("%d%d%d",&a,&b,&c); printf("%d is the maximum number",findmaximum(a,b,c)); return 0; }	13 45 23 45 is the maximum number
Menu driven calculator using functions	**Output**
#include<stdio.h> int addition(inta,int b) { return(a+b); } int subtraction(inta,int b) { return(a-b); } int multiplication(inta,int b) { return(a*b); }	**Sample Input 1:** 23 22 1 **Sample Output1:** 45 **Sample Input 2:** 23 2 3

```
float division(inta,int b)
{
   float k;
   k=(float)a/b;
   return(k);
}
intmodulo(inta,int b)
{
   return(a%b);
}
int power(inta,int b)
{
inti,power=1;
   for(i=1;i<=b;i++)
   {
      power=power*a;
   }
   return power;
}
float average(inta,int b)
{
   float k=a+b;
   return (k/2);
}
int main()
{
inta,b,ch;
scanf("%d%d%d",&a,&b,&ch);
   switch(ch)
   {
      case 1:
printf("%d",addition(a,b));
      break;
      case 2:
printf("%d",subtraction(a,b));
      break;
      case 3:
printf("%d",multiplication(a,b));
      break;
      case 4:
printf("%.2f",division(a,b));
      break;
      case 5:
printf("%d",modulo(a,b));
      break;
      case 6:
printf("%.2f",average(a,b));
      break;
```

Sample Output2:

46

```
        case 7:
printf("%d",power(a,b));
        break;
    }
    return 0;
}
```

Sum of digits using recursion in C

Output

```
#include<stdio.h>
intcomputeSum(int);
int main()
{
intnum,result;
printf("Enter the value of n\n");
scanf("%d",&num);
    result=computeSum(num);
printf("The sum of digits in %d is %d",num,result);
    return 0;
}
intcomputeSum(intnum)
{
    if(num!=0)
    {
        return(num%10+computeSum(num/10));
    }
    else
    {
        return 0;
    }
}
```

```
Enter the value of n
432

The sum of digits in 432 is 9
```

Add two numbers using pointers

Output

```
#include<stdio.h>
void addition(int *,int *);
int main()
{
intfirst,second;
printf("Enter the value of a\n");
scanf("%d",&first);
printf("Enter the value of b\n");
scanf("%d",&second);
    addition(&first,&second);
    return 0;
}
void addition(int *a,int *b)
{
printf("Sum of two elements = %d",*a+*b);
}
```

```
Enter the value of a
5
Enter the value of b
-3
Sum of two elements = 2
```

Find odd or even using pointers	Output
#include<stdio.h> void oddoreven(int *); int main() { int n; printf("Enter the number\n"); scanf("%d",&n); oddoreven(&n); return 0; } void oddoreven(int *a) { if(*(a)%2==0) { printf("%d is an even number",*a); } else { printf("%d is an odd number",*a); } }	Enter the number **6** 6 is an even number
Maximum of three numbers using pointers	Output
#include<stdio.h> int maximum(int *a,int *b,int*c); int main() { inta,b,c; printf("Enter the value of a\n"); scanf("%d",&a); printf("Enter the value of b\n"); scanf("%d",&b); printf("Enter the value of c\n"); scanf("%d",&c); inti=maximum(&a,&b,&c); printf("Maximum element is %d",i); return 0; } int maximum(int*x,int*y,int*z) { int max; if(*x>*y) max=*x;	Enter the value of a **5** Enter the value of b **6** Enter the value of c **2** Maximum element is 6

else max=*y; if(*z>max) max=*z; return max; }	

Sri Eshwar College of Engineering
Coimbatore- 641 202

Unit V

Structures, Unions and File Handling

Structure: Create a Structure-Member initialization - Accessing Structure Members - Nested structures– Pointer and Structures – Array of structures -Self Referential Structures – type def-Unions, Files –Opening and Closing a Data File, Reading and writing a data file.

STRUCTURES

- The primitive data types as the name implies, represent single data.
- Arrays represent a collection of homogenous elements, but there is a situation like representation of student records, inventory etc., which work with a collection of heterogeneous elements.
- C provides the type structure to facilitate such representations.
- Structure in C, is a collection of composite data elements to a single entity.
- Each element in a structure is called its member.
- It can be created with the keyword struct.

Creating a Structure

Syntax:

```
struct structure_name
{
data_type member 1;
:
data_type member n;
};
```

Example

```
struct passenger
{
char name[20];
int age;
char source[20];
```

```
char destination[20];
};
```

STRUCTURE VARIABLE DECLARATION

A Structure variable can be declared in two ways,

1. **While creating a structure**

```
struct passenger
{
char name[20];
int age;
char source[20];
char destination[20];
}P; // Structure variable declaration
```

Here in this example P is the structure variable.

2. **Separately like normal variable declaration.**

```
struct passenger
{
char name[20];
int age;
char source[20];
char destination[20];
};

void main()
{
struct passenger P; // Structure variable declaration
}
```

Here in this example P is the structure variable.

Member Initialization

- Members of the structure can be initialized in 2 ways,
  1. Using structure variable operator and dot operator
  2. Group initialization of members using structure variable

1. **Using structure variable and dot operator**

```
struct passenger
{
char name[20];
int age;
char source[20];
char destination[20];
};
void main()
{
struct passenger P; // Structure variable declaration
P.name="Krishna";
P.age=30;
P.source="Chennai";
P.destination="Canada";
}
```

Members Initialization

2. **Group initialization of members using structure variable**

```
struct passenger
{
char name[20];
int age;
char source[20];
char destination[20];
};
void main()
{
struct passenger P={"Krishna",30, "Chennai", "Canada"}; // Members
                                                          Initialization
}
```

Accessing Structure Members

Accessing structure members can be done with the dot operator.

Example Program1

```
struct passenger
{
char name[20];
int age;
char source[20];
char destination[20];
};

void main()
{
struct passenger P={"Krishna",30, "Chennai", "Canada"}; // Members Initialization
printf("Name=%s\n", P.name);
printf("Age=%d"\n, P.age);
printf("Source=%s\n", P.source);
printf("Destination=%s\n",  P.destination);
}
```

//Accessing structure members using dot operator

Output:

```
Krishna
30
Chennai
Canada
```

Explanation:

- Create a structure passenger with 4 members- name, age, source, destination
- Group initialization of members using structure variable P as, name ="Krishna", age=30, source="Chennai", destination="Canada"
- Print the name, age, source and destination by accessing the members of the structure using dot operator

Example Program 2

```
#include<stdio.h>
#include <string.h>
struct employee
{
int id;
char name[50];
float salary;
}e1,e2;  //declaring e1 and e2 variables for structure
int main( )
{
 //store first employee information
e1.id=101;
e1.name= "Krishna";
e1.salary=56000;
 //store second employee information
e2.id=102;
e2.name= "Bala";
e2.salary=126000;
//printing first employee information
printf( "employee 1 id : %d\n", e1.id);
printf( "employee 1 name : %s\n", e1.name);
printf( "employee 1 salary : %f\n", e1.salary);
//printing second employee information
printf( "employee 2 id : %d\n", e2.id);
printf( "employee 2 name : %s\n", e2.name);
printf( "employee 2 salary : %f\n", e2.salary);
return 0;
}
```

Output:

```
employee 1 id : 101
employee 1 name : Krishna
employee 1 salary : 56000.000000
employee 2 id : 102
employee 2 name : Bala
employee 2 salary : 126000.000000
```

Explanation:

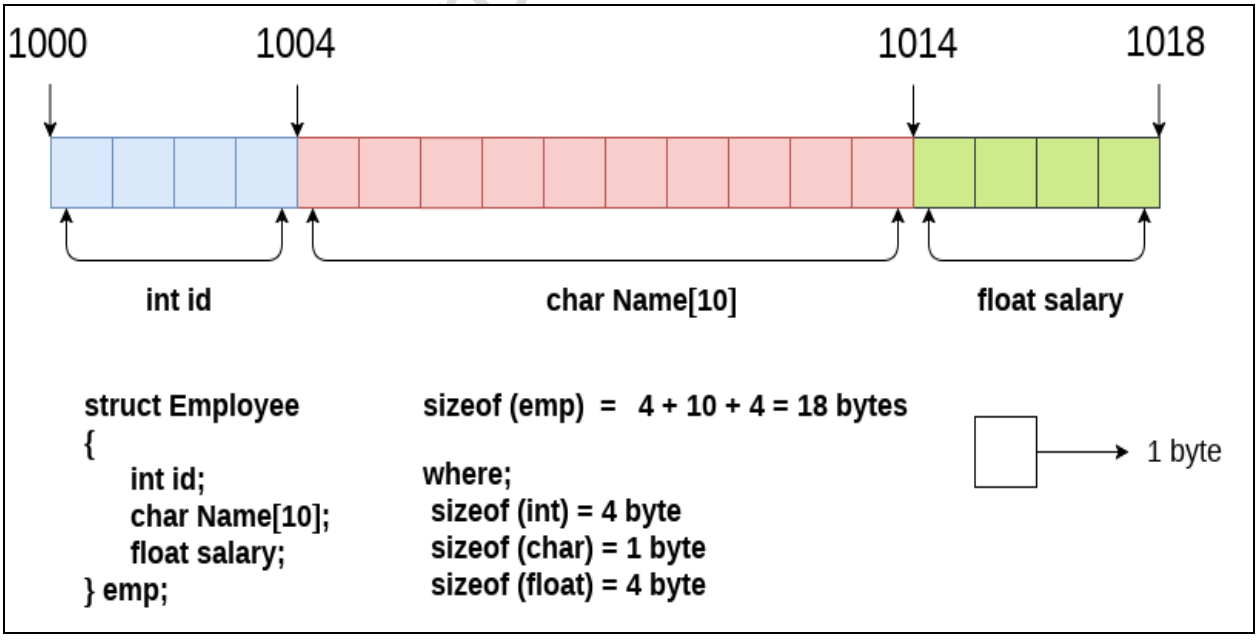
- Create a structure employee with 3 members- id, name, and salary.
- Print the id, name and salary of the employee by accessing the members of the structure using dot operator.

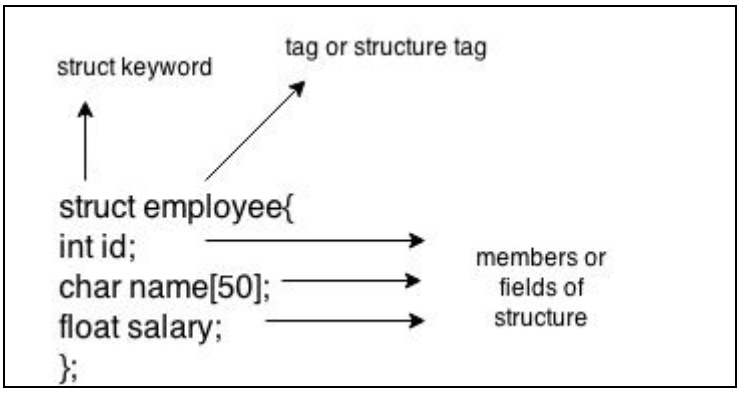
Storage Representation of Structure Variables

Let's see the example to define a structure for an entity employee in c.

```
struct employee
{
int id;
char name[20];
float salary;
};
```

The following picture shows the memory allocation of the structure employee that is defined in the above example.





Nested Structures

- If a structure contains one or more structures as its members, it is known as nested structure.
- It is used to increase the readability of the program by reducing the complexity.

Example Program

```
struct passenger
{
char name[20];
int age;
    struct route
    {
    char source[20];
    char destination[20];
    } R;
}P;
void main()
{
printf("Enter the passenger's name:");
```

// **Structure route** inside the **Structure Passenger**

```
scanf("%s",P.name);
printf("Enter the passenger's age:");
scanf("%d",&P.age);
printf("Enter the passenger's source route:");
scanf("%s",P.R.source);                          // Accessing the
printf("Enter the passenger's destination route:");   inner structure
scanf("%s",P.R.destination);                          members


printf("Name=%s\n", P.name);
printf("Age=%d"\n, P.age);
printf("Source=%s\n", P.R.source);
printf("Destination=%s\n",  P.R.destination);
}
```

Output:

```
Enter the passenger 's name: Krishna
Enter the passenger's age: 30
Enter the passenger's source route: Chennai
Enter the passenger's destination route: Canada

Name=Krishna
Age=30
Source=Chennai
Destination=Canada
```

Explanation:

- Create a structure "passenger" with two members name and age
- Inside the structure passenger create a nested structure "route" with two members source and destination
- Structure variable for passenger is P and Structure variable for route is R.
- Access the members of passenger structure with the structure variable P and access the members of the inner structure as P.R.source and P.R.destination.

Array of Structures

- A group of structures can be organised in an array.
- The structure is used to store data about a particular structure object.
- If we want to store data about a group of structure objects, the Array of structures can be used.

```
struct passenger
{
char name[20];
int age;
char source[20];
char destination[20];
}P[3];  // Creating a Array of structure to store 3 passengers' records
void main()
{
int i;
for(i=0i<3;i++)
{
printf("Enter the passenger's name:");
scanf("%s",P[i].name);
printf("Enter the passenger's age:");
scanf("%d",&P[i].age);
printf("Enter the passenger's source route:");
```

```
scanf("%s",P[i].source);

printf("Enter the passenger's destination route:");
scanf("%s",P[i].destination);

}

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

{

printf("Name=%s\n", P[i].name);

printf("Age=%d"\n, P[i].age);

printf("Source=%s\n", P[i].source);

printf("Destination=%s\n",  P[i].destination);

}
```

Output:

Enter the passenger 's name: Krishna

Enter the passenger's age: 30

Enter the passenger's source route: Chennai

Enter the passenger's destination route: Canada

Enter the passenger 's name: Kumar

Enter the passenger's age: 31

Enter the passenger's source route: Chennai

Enter the passenger's destination route: Singapore

Enter the passenger ‘s name: Deepa

Enter the passenger’s age: 27

Enter the passenger’s source route: Coimbatore

Enter the passenger’s destination route: Chennai

Name=Krishna

Age=30

Source=Chennai

Destination=Canada

Name=Kumar

Age=31

Source=Chennai

Destination= Singapore

Name=Deepa

Age=27

Source= Coimbatore

Destination= Chennai

Explanation:

- Create a structure passenger with four members, name, age, source and destination.
- Create a array of structure variable P to store 3 different passenger records

- Using loop get 3 different passenger inputs for the structure passenger and print the same.

Pointer and Structures

- Structures can be created and accessed using pointers. A pointer variable of a structure can be created as below:

```
struct name
{
member 1;
member 2;
} ;
int main()
{
struct name *ptr;
}
```

Structure members can be accessed using a pointer in two ways.

- Using arrow (->) operator or membership operator.

```
struct passenger
{
char name[20];
int age;
char source[20];
char destination[20];
```

```
};
void main()
{
struct passenger *Ptr; // Structure Pointer
printf("Name=%s\n", Ptr->name);
printf("Age=%d"\n, Ptr->age);
printf("Source=%s\n", Ptr->source);
printf("Destination=%s\n",  Ptr->destination);
}
```

Accessing structure members using structure pointer

Output:

Krishna

30

Chennai

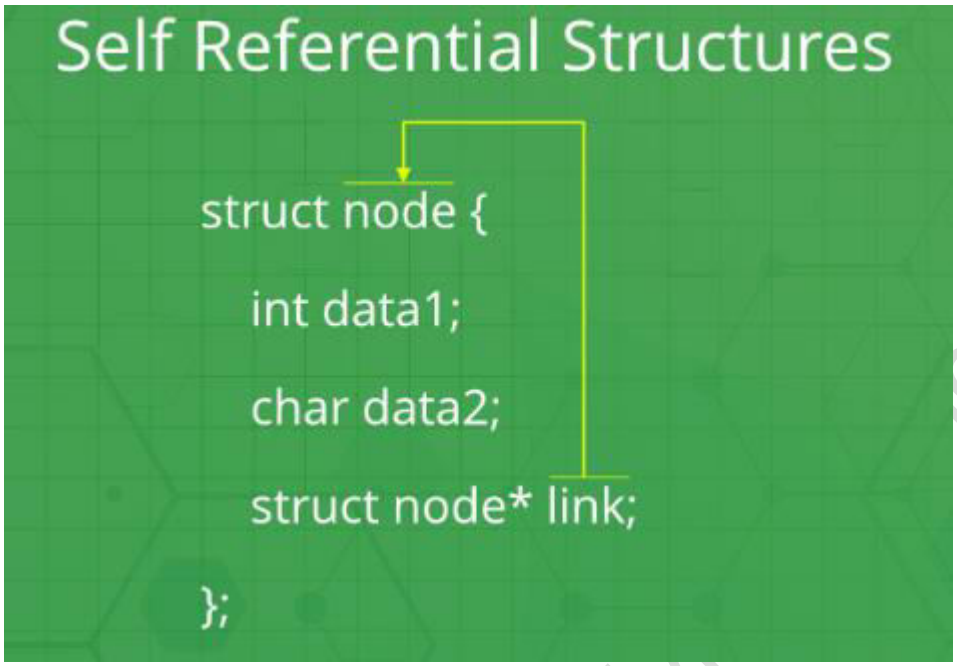
Canada

Explanation:

- Create a structure passenger with four members, name, age, source and destination.
- Create a structure pointer Ptr and access the structure members using the membership operator(->)

Self-Referential Structures

- Self Referential structures are those structures that have one or more pointers which point to the same type of structure, as their member.
- In other words, structures pointing to the same type of structures are self-referential in nature.



Syntax

```
struct structure_name
{
datatype variable;
structure_name  *pointer_variable;
};
```

Example

```
struct cnode
{
struct cnode *header;  // pointer member pointing to the same structure
char cvalue;
};
```

typedef

- typedef is used for renaming data types.
- The purpose of typedef is to redefine the name of an existing variable type.

- Usually typedef is used with user defined data types.

Syntax:

```
typedef struct structure_name
{
   datatype member1;
   datatype member2;
   datatype member3;
} alias_name;
```

Example Program:

```
#include<stdio.h>
#include<string.h>
struct employees
{
   char employee_name[20];
   int age;
};
void main( )
{
   //alias name for "struct employees" is empo  i.e. empo = "struct employees"
   typedef struct employees empo;
   empo eo; // create the structure variable eo using the alias name empo
```

```
    printf("\n Employees record……\n");
    printf("\n Enter the Employee name: ");
    scanf("%s", eo.employee_name);
    printf("\n Enter the Employee age: ");
    scanf("%d", & eo.age);
    printf("\n Name of the student is: %s", eo.employee_name);
    printf("\n Age of the employee is: %d", eo.age);
}
```

Output:

Employees record……

Enter the Employee name: Sudha

Enter the Employee age: 32

Name of the student is: Sudha

Age of the employee is: 32

Explanation:

- In the above example typedef is applied to the user defined data type structure.
- create a structure employees with two members employee_name, age
- alias name for "struct employees" is empo i.e. empo = "struct employees"
- By using the alias name empo, structure variable eo is created.

Unions

- Since the concept of union comes from the structure, the syntax is same for both structure and union.

- They differ only in storage.
- The members of the structure will have their own storage locations.
- The members of a union are of different type and it can handle only one member at a time.
- It allocates storage for the largest member.

Syntax

union union_name

{

data_type member 1;

:

data_type member n;

};

Example:

union Student

{

char cname[20];

int iage;

};

Explanation:

In the above example union contains 2 members, each with different type. However, we can use only one at a time because union uses a shared memory location that is equal to the size of its largest member.

Accessing Union Members

Accessing union members are same as that of the structure members.

```
#include<stdio.h>
void main()
{
        union Student
        {
        char cname[20];
        int iage;
        }S;
//Accessing the members of union with union variable S
S.cname=”Akash”;
S.iage=18;
printf(“Student name:%s”,S.cname);
printf(“Student age:%d”,S.iage);
}
```

Files

- File is a collection of bytes that is stored on secondary storage devices like disk.
- File is created for permanent storage of data.
- Files can be accessed using Library functions or system calls of operating system.
- A stream refers to a source or destination of data that may be associated with the disk or other I/O devices.
- The source stream provides data to a program and it is known as input stream.

- The destination stream receives the output from the program and is known as output stream.
- A file is identified by a name. Filename is a string of characters consisting of 2 parts primary name and an optional period with the extension.
- **Example:** test.dat, program.c, text

Types of Files

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

1. Text files
2. Binary files

1. Text files

- Text files are the **normal .txt files**. We can easily create text files using any simple text editors such as Notepad.
- When we open those files, we'll see all the contents within the file as plain text. We can easily edit or delete the contents.
- They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.

2. Binary files

- Binary files are mostly the **.bin** files in your computer.
- Instead of storing data in plain text, **they store it in the binary form (0's and 1's).**
- They can hold a higher amount of data, are not readable easily, and provides better security than text files.

Operations Performed in a File

1. Opening a file
2. Reading data from a file
3. Writing data to a file
4. Closing a file

Opening a File

- The data structure of the file is defined as FILE in the standard I/O library function definitions.
- Therefore all files should be declared as type FILE before they are used.
- The library function to open a file is fopen(). It has two arguments,
  1. filename
  2. mode

File Accessing Modes

Mode	Purpose
r	Used to open the file for reading purpose.
w	Used to open the file for writing purpose.
a	Used to appending data into the file.
r+	Used to open an existing file for reading and writing
w+	Used to open a new file for both reading and writing
a+	Used to open an existing file for both reading and writing

Syntax:

FILE *file_pointer;

file_pointer=fopen (filename,mode);

where,

- First statement is pointer to the data type FILE.
- Second statement opens the file named filename and assigns identifier to the FILE type pointer, which contains all the information about the file and it is used as a communication link between the system and the program.
- mode in the second statement specifies the purpose of accessing the file

like read, write, etc.,

Example:

filefp=fopen("Input","w");

Explanation:

It opens the file Input in the write mode. If Input file already exists its contents are deleted. If it does not exists, the Input file is created.

Closing a File

A file must be closed after all the operations of the file have been completed.

Syntax:

fclose(file_pointer);

Example:

fclose(filefp);

Explanation:

In the above example the statement fclose(filefp) is used to close the file after all operations on them are completed. All files are closed automatically whenever a program terminates.

Function in C Library to perform various operations on a file

Function	Description
getc()	Used to read a character from a file
putc()	Used to write a character to a file
fscanf()	Used to read a set of data from a file
fprintf()	Used to write a set of data to a file

getw()	Used to read a integer from a file
putw()	Used to write a integer to a file
fseek()	Used to set the position to desire point
ftell()	Used to give current position in the file
rewind()	Used to set the position to the beginning point

Reading and Writing Operation on File using getc() and putc()

getc ()

- This function reads a single character from the opened file and moves the file pointer
- **Syntax:** getc(file *fptr);

putc()

- This function is used to write a single character into a file. If an error occurs it returns **EOF.**
- **Syntax: putc(int c, File*fp);**

Example program to illustrate getc () and putc ()

```
#include<stdio.h>
int main()
{
   FILE *filefp;
   char character;

   //open the file fileone.txt in write mode using fopen()
   filefp = fopen("fileone.txt", "w");

   printf("Enter data in to fileone.txt ....");
   while( (character = getchar()) != EOF)
   {
      putc(character, filefp);
```

```
    }

    fclose(filefp);

    //open the file fileone.txt in read mode using fopen()
    filefp = fopen("fileone.txt", "r");

    while( (character = getc(filefp)! = EOF)
    printf("%c",character);

    // close the file pointer using fclose()
    fclose(filefp);
    return 0;
}
```

Explanation:

- Create a File pointer filefp to point the file "fileone.txt"
- Open the file "fileone.txt" in write mode to perform write operation on the file using fopen()
- Write the characters in to the file "fileone.txt" using putc()
- **putc()** – putc() is a function used to write a character in to a file.
- **Syntax**: putc(character,file_pointer)
- Close the File pointer filefp after writing characters in to the file.
- Open the file "fileone.txt" in read mode to perform read operation on the file using fopen()
- Read the characters from the file "fileone.txt" using getc()
- **getc()** – getc() is a function used to read a character from a file.
- **Syntax**: getc(file_pointer)
- Close the File pointer filefp after reading characters from the file.

Reading and Writing Operation on File using fprintf() and fscanf()

fscanf()

- **fscanf()** reads character, strings, integer, floats, and so on, from the file pointed by file pointer.

- **Syntax: fscanf(file_pointer, "format_specifier",variable_name);**

fprintf()

- fprintf() is used for writing characters, strings, integers, floats, and so on, to the file.
- It contains one more parameter this is file pointer, which points the opened file.
- **Syntax**: **fprintf(file_pointer, "format_specifier",variable_name);**

C Program to read and write contents from a data file using fscanf() and fprintf()

```
#include<stdio.h>
struct employees
{
   char cname[10];
   int cage;
};
void main()
{
   struct employees e1;
   FILE *f1,*f2;
   f1 = fopen("fileone.txt", "a");
   f2 = fopen("fileone.txt", "r");
   printf("Enter the Name and Age of the Employee:");
   scanf("%s %d", e1.cname, &e1.cage);
   fprintf(f1,"%s %d", e1.cname, e1.cage);
   fclose(f1);
   do
   {
      fscanf(f2,"%s %d", e1.cname, e1.cage);
      printf("%s %d", e1.cname, e1.cage);
   }
   while(!feof(f2));
}
```

Explanation:

- Create a structure employees with two members namely name and age
- Create two file pointers f1 and f2. f1 is to open the file "fileone.txt" in append mode, f2 is to open the file "fileone.txt" in read mode.
- Get input for the structure members name and age through console and write in to the file "fileone.txt" using fprintf().
- **fprintf()** is a function used to write formatted content in to a file.
- **Syntax:** fprintf(file_pointer, "format_specifier",variable_name);
- Close the file pointer f1once after writing the content into the file.
- **fscanf()** is a function used to read formatted content from the file
- **Syntax:** fscanf(file_pointer, "format_specifier",variable_name);
- Close the file pointer f2 once after reading the content from the file.

Random File Access

- **fseek()**
  - The fseek() function helps us send a file pointer to a specified location. The syntax of fseek() is as follows:

int fseek(FILE *pointer, long int offset, int whence);

- The function accepts three parameters; the file pointer, an integer offset (which is the number of bytes to be shifted from the position mentioned in the third parameter), and the third parameter which specifies the position from where the offset is added.
- Along with this, there are three macros used in fseek(). These are SEEK_SET (beginning of the file), SEEK_CUR (current position in the file), SEEK_END (end of the file).

ftell()

- The ftell() function is used to find out the exact position of the file pointer with respect to the beginning. This function accepts the file pointer as the parameter.
- Syntax: **ftell(FILE *pointer);**

rewind()

- This function also accepts the file pointer as a parameter and sets the file pointer to the beginning of the file again. The syntax is as follows:

void rewind(FILE *pointer);

C Program to illustrate random file access

```
#include <stdio.h>
int main () {
  FILE *fp;
  int c;
  fp = fopen("file.txt","w+");
  fputs("This is study.com", fp);
  // we are using fseek to shift the file pointer to the 7th position
  fseek( fp, 7, SEEK_SET );

//Now we overwrite C programming in the 7th position
  fputs(" C Programming", fp);

//now we print the current position of the file pointer using ftell
  printf("The current position of the file pointer is: %ld\n", ftell(fp));
//we take the file pointer to the beginning of the file
  rewind(fp);

//now we verify if rewind() worked using ftell
  printf("The current position of the file pointer is: %ld\n", ftell(fp));
  while(1) {
   c = fgetc(fp);
   if( feof(fp) ) {
    break;
   }
   printf("%c", c);
  }
  fclose(fp);
  return(0);
```

```
}
```

Output

The current position of the file pointer is: 21
The current position of the file pointer is: 0
This is C Programming

Part-A

1) Distinguish between arrays and structures

Arrays	Structures
a. An array is a collection of data items of same data type. Arrays can only be declared. b. There is no keyword for arrays c. An array name represents the address of the starting element d. An array cannot have bit fields	a. A structure is a collection of data items of different data types. Structures can be declared and defined. b. The keyword for structures is struct. c. A structure name is known as tag. It is a Shorthand notation of the declaration. d. A structure may contain bit fields

2. Differentiate structures and unions

Structures	Unions
• Every member has its own memory	All members use the same memory
• The keyword used is struct	The keyword used is union.
• All members occupy separate memory location; hence different interpretations of the same memory location are not possible.	Different interpretations for the same memory location are possible.
• Consumes more space compared to union.	Conservation of memory is possible

3. Define Structure in C.

Structure can be defined as a collection of different data types which are grouped together and each element in a C structure is called member. To access structure members in C, structure variable should be declared. The keyword used is struct.

4. How will you define a structure?

A structure can be defined as

```
struct tag
```

```
{
        datatype member 1;
        datatype member 2;
        ……
        ……
        datatype member n;
};
```

where struct is a keyword, tag is a name that identifies structures, member 1, member 2,….. member n are individual member declarations.

5. How will you declare structure variables?

Declaration of structure variables includes the following statements

a. The keyword struct
b. The structure name
c. List of variable names separated by commas
d. A terminating semicolon

```
struct library_books
{
char title[20];
char author[15];
int pages;
float price;
};
struct library_books b1,b2,b3;
```

6. What is meant by Union in C? (May 2014)

A union is a special data type available in C that enables to store different data types in the same memory location. Union can be defined 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 multi-purpose.

7. How to define a union in C.

The format of the union statement is as follows

```
union tag
{
        member definition;
        member definition;
        ...
        member definition;
} one or more union variables;
```

8. How can you access the members of the Union?

To access any member of a union, use the member access operator (.). The member access operator is coded as a period between the union variable name and the union member to access. Union keyword is used to define variables of union type.

9. List the features of structures. (May 2015)

The features of structures are as follows

a. All the elements of a structure are stored at contiguous memory locations
b. A variable of structure type can store multiple data items of different data types under

the one name

10. List the main aspects of working with structure.

a. Defining a structure type (Creating a new type).
b. Declaring variables and constants (objects) of the newly created type.
c. Initializing structure elements

11. What are the two ways of passing a structure to function in C?

a. Passing structure to a function by value
b. Passing structure to a function by address(reference)

12. Write any two advantage of Structure.

a. It is used to store different data types.
b. Each element can be accessed individually.

13. How to initialize a structure variable? Give it's syntax.

Static storage class is used to initialize structure. It should being and end with curly braces.

Syntax: Static structure tag-field structure variable = {value1, value2,...value 3};

14. Define Anonymous structure.

Unnamed anonymous structure can be defined as the structure definition that does not contain a structure name. Thus the variables of unnamed structure should be declared only at the time of structure definition.

15. What are the operations on structures?

The operations on structures are

a. Aggregate operations: An aggregate operation treats an operand as an entity and operates on the entire operand as whole instead of operating on its constituent members.
b. Segregate operations: A segregate operation operates on the individual members of a structure object.

16. How to access members of a structure object?

a. Direct member access operator (dot operator)
b. Indirect member access operator(arrow operator)

17. What is the use of 'period (.)' in C?

To access any member of a structure, we use the member access operator (.). The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. Period is used to initialize a structure. The members of structure can be accessed individually using period operator. Ex: S1.roll.no;

18. How will you access the structures member through pointers?

The structures member can be accessed through pointers by the following ways

a. Referencing pointer to another address to access memory
b. Using dynamic memory allocation

19. Define Nested structure.

Nested structure can be defined as the structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure. The structure variables can be a normal structure variable or a pointer variable to access the data.

20. Define array of structures.

Each elements of an array represent a structure variable. If we want to store more array objects in a structure, then we can declare "array of structure".

21. Consider the declaration and illustrate the application of size of operator to this structure. (Nov 2010)

```
struct student
{
char name;
int num;
} S;
```

Size of this is 3 bytes:1 byte for name and 2 bytes for integer num.

22. What is a file?

A file is a collection of related data stored on a secondary storage device like hard disk. Every file contains data that is organized in hierarchy as fields, records, and databases. Stored as sequence of bytes logically contiguous (may not be physically contiguous on disk).

23. List out the types of files.

The following are the types of files

a. Text file or ASCII text file: collection of information or data which are easily readable by humans. Ex. .txt, .doc, .ppt, .c, .cpp
b. Binary file: It is collection of bytes. Very tough to read by humans. Ex. .gif, .bmp, .jpeg, .exe, .obj

24. **What is file pointer?**

The pointer to a **FILE** data type is called as a stream pointer or a file pointer. A file pointer points to the block of information of the stream that had just been opened.

25. **List the different modes of opening a file.**

The function fopen() has various modes of operation that are listed below

Mode	Description
R	Opens a text file in reading mode
W	Opens or create a text file in writing mode.
A	Opens a text file in appended mode.
r+	Opens a text file in both reading and writing mode.
w+	Opens a text file in both reading and writing mode.

26. **What are file attributes?**

a. File name

b. File Position

c. File Structure

d. File Access Methods

e. Attributes flag

27. **What is the use of functions fseek(), fread(), fwrite() and ftell()?**

a) fseek(f,1,i)Move the pointer for file f of a distance 1 byte from location i.
b) fread(s,i1,i2,f)Enter i2 dataitems, each of size i1 bytes from file f to string s.
c) fwrite(s,i1,i2,f)send i2 data items,each of size i1 bytes from string s to file f.
d) ftell(f)Return the current pointer position within file f.
e) The data type returned for functions fread,fseek and fwrite is int and ftell is long int.

28. How is a file opened and file closed?

a. A file is opened using fopen()function. Ex: fp=fopen(filename,mode);

b. A file closed using fclose()function. Ex: fclose(fp);

29. What are the statements used for reading a file? (Nov 2014)

a. FILE*p;Initialize file pointer.

b. fp=fopen("File_name" "r");Open text file for reading.

c. Getc(file_pointer_name);Reads character from file.

d. fgets(str,length,fp); Reads the string from the file.

30. What is the difference between getc() and getchar()? (May 2015)

int getc(FILE * stream) gets the next character on the given input stream and file pointer to point

to the next character.

int getchar(void) returns the next character on the input stream stdin.

PART - B

1. Define Structures. Explain structures in detail.
2. Explain array of structure in C.
3. Write a C program to create mark sheet for students using structure. (Jan 2014)
4. How can you insert structure with in another structure?
5. Explain the concept of structures and functions, structures and pointers with example.
6. Define Union. Explain Union in detail. (May 2015)

Define a structure called book with book name, author name and price. Write a C program to read the details of book name, author name and price of 200 books in a library and display the total cost of the books and the book details whose price is above Rs. 500. (May 2015)

7. Write a C program using structure to store date, which includes day, month and year. (Jan 2014)

8. Write a C program to store the employee information using structure and search a particular employee using Employee Number. (June 2014)

9. A sample C programming code for real time Bank application program is given below. This program will perform all below operations.

(i) Creating new account: To create a new account

(ii) Cash Deposit: To Deposit some amount in newly created account

(iii)Cash withdrawal: To Withdraw some amount from your account

(iv)Display Account information: It will display all information‟s of the existing accounts

(v) Log out

(vi)Clearing the output screen and display available options

10. Explain the various operations to be performed on a file and write a c program for reading and writing to a file.

Additional C Program using Array of structures (Very Important Programs)

Display Student Grades using array of structures	Output
#include<stdio.h> struct Student { int rm; char name[100]; int tm; }; int main() { int n;	Enter the number of students 3 Enter the roll number of student 1 1 Enter the name of student 1 bala Enter the total mark of student 1 40 Enter the roll number of student

```
int i;
printf("Enter the number of students\n");
scanf("%d",&n);
struct Student s[n];
for(i=0;i<n;i++)
{
   printf("Enter the roll number of student %d\n",i+1);
   scanf("%d",&s[i].rm);
   printf("Enter the name of student %d\n",i+1);
   scanf("%s",s[i].name);
   printf("Enter the total mark of student %d\n",i+1);
   scanf("%d",&s[i].tm);
}
printf("Grade details\n");
for(i=0;i<n;i++)
{
  printf("%d %s %d ",s[i].rm,s[i].name,s[i].tm);
     if(s[i].tm>=90)
     printf("A");
     else if(s[i].tm>=80 && s[i].tm<90)
     printf("B");
     else if(s[i].tm>=70 && s[i].tm<80)
```

```
2
2
Enter the name of student 2
ramya
Enter the total mark of student 2
100
Enter the roll number of student 3
3
Enter the name of student 3
raja
Enter the total mark of student 3
65
Grade details
1 bala 40 Fail
2 ramya 100 A
3 raja 65 D
```

```
        printf("C");
        else if(s[i].tm>=60 && s[i].tm<70)
        printf("D");
        else if(s[i].tm>=50 && s[i].tm<60)
        printf("E");
        else
        printf("Fail");
        printf("\n");
    }
    return 0;
}
```

Display Employee details using array of structures

```
#include<stdio.h>
struct emp
{
    char name[50];
    int emp_id;
    int age;
    char designation[50];
    float salary;
};
int main()
```

Output

```
Enter details for employee 1
Enter employee name
sam
Enter employee id
1
Enter employee age
35
Enter employee designation
Manager
Enter employee salary
13000
Enter details for employee 2
Enter employee name
Ram
Enter employee id
```

```
{
   struct emp e[5];
   int i;
   for(i=0;i<5;i++)
   {
      printf("Enter details for employee %d\n",i+1);
      printf("Enter employee name\n");
      scanf("%s",e[i].name);
      printf("Enter employee id\n");
      scanf("%d",&e[i].emp_id);
      printf("Enter employee age\n");
      scanf("%d",&e[i].age);
      printf("Enter employee designation\n");
      scanf("%s",e[i].designation);
      printf("Enter employee salary\n");
      scanf("%f",&e[i].salary);

   }
   for(i=0;i<5;i++)
   {
      printf("Details of employee %d\n",i+1);
      printf("Employee name:%s\n",e[i].name);
```

```
2
Enter employee age
40
Enter employee designation
Assistant
Enter employee salary
7000
Enter details for employee 3
Enter employee name
santosh
Enter employee id
3
Enter employee age
37
Enter employee designation
Driver
Enter employee salary
6000.50
Enter details for employee 4
Enter employee name
Sunil
Enter employee id
4
Enter employee age
45
Enter employee designation
Clerk
Enter employee salary
6500
Enter details for employee 5
Enter employee name
Mandy
Enter employee id
5
```

```
        printf("Employee id:%d\n",e[i].emp_id);
        printf("Employee age:%d\n",e[i].age);
        printf("Employee designation:%s\n",e[i].designation);
        printf("Employee salary:%.2f",e[i].salary);
    }
    return 0;
}
```

```
Enter employee age
31
Enter employee designation
Admin
Enter employee salary
9000
Details of employee 1
Employee name:sam
Employee id:1
Employee age:35
Employee designation:Manager
Employee salary:13000.00
Details of employee 2
Employee name:Ram
Employee id:2
Employee age:40
Employee designation:Assistant
Employee salary:7000.00
Details of employee 3
Employee name:santosh
Employee id:3
Employee age:37
Employee designation:Driver
Employee salary:6000.50
Details of employee 4
Employee name:Sunil
Employee id:4
Employee age:45
Employee designation:Clerk
Employee salary:6500.00
Details of employee 5
Employee name:Mandy
Employee id:5
Employee age:31
```

	Employee designation:Admin Employee salary:9000.00