SYLLABUS

	Topic
1.	**Basics of Programming :** Approaches to problem solving, Use of high level programming language for systematic development of programs, Concept of algorithm and flowchart, Concept and role of structured programming. **Basics of C :** History of C, Salient features of C, Structure of C Program, Compiling C Program, Link and Run C Program, Character set, Tokens, Keywords, Identifiers, Constants, Variables, Instructions, Data types, Standard Input/Output, Operators and expressions.
2.	**Conditional Program Execution :** if, if-else, and nested if-else statements, Switch statements, Restrictions on switch values, Use of break and default with switch, Comparison of switch and if-else. **Loops and Iteration :** for, while and do-while loops, Multiple loop variables, Nested loops, Assignment operators, break and continue statement. **Functions :** Introduction, Types, Declaration of a Function, Function calls, Defining functions, Function Prototypes, Passing arguments to a function Return values and their types, Writing multifunction program, Calling function by value, Recursive functions.
3.	**Arrays :** Array notation and representation, Declaring one-dimensional array, Initializing arrays, Accessing array elements, Manipulating array elements, Arrays of unknown or varying size, Two-dimensional arrays, Multidimensional arrays. **Pointers :** Introduction, Characteristics, and & operators, Pointer type declaration and assignment, Pointer arithmetic, Call by reference, Passing pointers to functions, array of pointers, Pointers to functions, Pointer to pointer, Array of pointers. **Strings :** Introduction, Initializing strings, Accessing string elements, Array of strings, Passing strings to functions, String functions.
4.	**Structure :** Introduction, Initializing, defining and declaring structure, Accessing members, Operations on individual members, Operations on structures, Structure within structure, Array of structure, Pointers to structure. **Union :** Introduction, Declaring union, Usage of unions, Operations on union. Enumerated data types Storage classes: Introduction, Types- automatic, register, static and external.
5.	**Dynamic Memory Allocation :** Introduction, Library functions – malloc, calloc, realloc and free. **File Handling :** Basics, File types, File operations, File pointer, File opening modes, File handling functions, File handling through command line argument, Record I/O in files. **Graphics :** Introduction, Constant, Data types and global variables used in graphics, Library functions used in drawing, Drawing and filling images, GUI interaction within the program.

BASICS OF PROGRAMMING

1. *What is concept of problem solving? Explain different problem solving techniques.*

Problem Solving

The term problem solving is used in many disciplines, sometimes with different perspectives, and often with different terminologies. For instance, it is a mental process in psychology and a computerized process in computer science. Problems can also be classified into two different types (ill-defined and well-defined) from which appropriate solutions are to be made. Ill-defined problems are those that do not have clear goals or solution paths, while well-defined problems have specific goals and clearly defined solution paths. Computer science and in the part of artificial intelligence that deals with algorithms ("algorithmic"), problem solving encompasses a number of techniques known as algorithms, heuristics, root cause analysis, etc. In these disciplines, problem solving is part of a larger process that encompasses problem determination, de-duplication, analysis, diagnosis, repair, etc.

Problem-solving strategies are the steps that one would use to find the problem that are in the way to getting to one's own goal. Some would refer to this as the 'problem-solving cycle'. In this cycle one will recognize the problem, define the problem, develop a strategy to fix the problem, organize the knowledge of the problem, figure-out the resources at the user's disposal, monitor one's progress, and evaluate the solution for accuracy. Although called a cycle, one does not have to do each step in order to fix the problem; in fact those who don't are usually better at problem solving. The reason it is called a cycle is that once one is completed with a problem another usually will pop up. The first looking at those problems that only have one solution (like math problems, or fact based questions) which are grounded in psychometric intelligence. The other that is socio emotional in nature and are unpredictable with answers that are constantly

changing. The following techniques are usually called problem-solving strategies.

Problem-Solving Techniques

Problem-solving strategies are the steps that one would use to find the problem that are in the way to getting to one's own goal. Some would refer to this as the 'problem-solving cycle'. In this cycle one will recognize the problem, define the problem, develop a strategy to fix the problem, organize the knowledge of the problem, figure-out the resources at the user's disposal, monitor one's progress, and evaluate the solution for accuracy. Although called a cycle, one does not have to do each step in order to fix the problem; in fact those who don't are usually better at problem solving. The reason it is called a cycle is that once one is completed with a problem another usually will pop up. The first looking at those problems that only have one solution (like math problems, or fact based questions) which are grounded in psychometric intelligence. The other that is socioemotional in nature and are unpredictable with answers that are constantly changing. The following techniques are usually called problem-solving strategies.

(1) **Abstraction** : Solving the problem in a model of the system before applying it to the real system.

(2) **Analogy** : Using a solution that solves an analogous problem.

(3) **Brainstorming** : (Especially among groups of people) suggesting a large number of solutions or ideas and combining and developing them until an optimum is found.

(4) **Hypothesis Testing** : Assuming a possible explanation to the problem and trying to prove (or, in some contexts, disprove) the assumption.

(5) **Lateral Thinking** : Approaching solutions indirectly and creatively

(6) **Means-Ends Analysis** : Choosing an action at each step to move closer to the goal.

(7) **Method of Focal Objects** : Synthesizing seeming non-matching characteristics of different objects into something new.

(8) **Morphological Analysis** : Assessing the output and interactions of an entire system.

(9) **Proof** : Try to prove that the problem cannot be solved. The point where the proof fails will be the starting point for solving it.

(10) **Reduction** : Transforming the problem into another problem for which solutions

(11) **Research** : Employing existing ideas or adapting existing solutions to similar problems.

(12) **Root Cause Analysis** : Identifying the cause of a problem.

(13) **Trial-and-Error** : Testing possible solutions until the right one is found.

(14) **Divide and Conquer** : In computer science, divide and conquer (D&C) is an important algorithm design paradigm based on multi-branched recursion. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

This technique is the basis of efficient algorithms for all kinds of problems, such as sorting (e.g., quick sort, merge sort), multiplying large numbers syntactic analysis), and computing the discrete Fourier transform. On the other hand, the ability to understand and design D&C algorithms is a skill that takes time to master. As when proving a theorem by induction, it is often necessary to replace the original problem by a more general or complicated problem in order to get the recursion going, and there is no systematic method for finding the proper generalization.

The name "divide and conquer" is sometimes applied also to algorithms that reduce each problem to only one sub problem, such as the binary search algorithm for finding a record in a sorted list (or its analog in numerical computing, the bisection algorithm for root finding). These algorithms can be implemented more efficiently than general divide-and-conquer algorithms; in particular, if they use tail recursion, they can be converted into simple loops. Under this broad definition, however, every algorithm that uses recursion or loops could be regarded as a "divide and conquer algorithm". Therefore, some authors

consider that the name "divide and conquer" should be used only when each problem may generate two or more sub problems. The name divide and conquer has been proposed instead for the single-sub problem class.

The correctness of a divide and conquer algorithm is usually proved by mathematical induction, and its computational cost is often determined by solving recurrence relations.

2. *What are the steps in problem solving? Explain in brief.*

Steps used to Problem Solving

An organization needs to define some standard of problem solving, so that leadership can effectively direct others in the research and resolution of issues.

In problem solving, there are four basic steps :

(1) **Define the Problem :** Diagnose the situation so that your focus is on the problem, not just its symptoms. Helpful techniques at this stage include using flowcharts to identify the expected steps of a process and cause-and-effect diagrams to define and analyze root causes.

The chart below identifies key steps for defining problems. These steps support the involvement of interested parties, the use of factual information, comparison of expectations to reality and a focus on root causes of a problem. What's needed is to :

(a) Review and document how processes currently work (who does what, with what information, using what tools, communicating with what organizations and individuals, in what time frame, using what format, etc).

(b) Evaluate the possible impact of new tools and revised policies in the development of a model of "what should be".

(2) Generate Alternative Solutions : Postpone the selection of one solution until several alternatives have been proposed. Having a standard with which to compare the characteristics of the final solution is not the same as defining the desired result. A standard allows us to evaluate the different intended results

offered by alternatives. When you try to build toward desired results, it's very difficult to collect good information about the process.

Considering multiple alternatives can significantly enhance the value of your final solution. Once the team or individual has decided the "what should be" model, this target standard becomes the basis for developing a road map for investigating alternatives. Brainstorming and team problem-solving techniques are both useful tools in this stage of problem solving.

Many alternative solutions should be generated before evaluating any of them. A common mistake in problem solving is that alternatives are evaluated as they are proposed, so the first acceptable solution is chosen, even if it's not the best fit. If we focus on trying to get the results we want, we miss the potential for learning something new that will allow for real improvement.

(3) Evaluate and Select an Alternative : Skilled problem solvers use a series of considerations when selecting the best alternative. They consider the extent to which :

(a) A particular alternative will solve the problem without causing other unanticipated problems.

(b) All the individuals involved will accept the alternative.

(c) Implementation of the alternative is likely.

(d) The alternative fits within the organizational constraints.

(4) Implement and Follow Up on the Solution : Leaders may be called upon to order the solution to be implemented by others, "sell" the solution to others or facilitate the implementation by involving the efforts of others. The most effective approach, by far, has been to involve others in the implementation as a way of minimizing resistance to subsequent changes. Feedback channels must be built into the implementation of the solution, to oduce continuous monitoring and testing of actual events against expectations. Problem solving, and the techniques used to derive elucidation, can only be

effective in an organization if the solution remains in place and is updated to respond to future changes.

3. ◆ *What is an algorithm? Give the characteristics of an algorithm* (2020-21)
◆ *What is an algorithm? Explain with the help of examples.*

Algorithm

Algorithm is step-by-step approach to solve a given problem. It is represented in an English like language and has some mathematical symbols like ->, >, <, = etc. To solve a given problem or to write a program you approach towards solution of the problem in a systematic, disciplined, non-ad hoc, step-by-step way is called Algorithmic approach.

Example 1 : Algorithm/Pseudo Code to Add Two Numbers :

Step 1 : Start
Step 2 : Read the two numbers in to a, b
Step 3 : c=a + b
Step 4 : write/print c
Step 5 : Stop.

Example 2 : Write an Algorithm to Find Greatest among Three Numbers :

Step 1 : start
Step 2 : input a, b, c
Step 3 : if a>b go to step 4,otherwise go to step 5
Step 4 : if a>c go to step 6,otherwise go to step 8
Step 5 : if b>c go to step 7,otherwise go to step 8
Step 6 : output "a is the largest ",go to step 9
Start 7 : Output "b is the largest", goto step 9
Start 8 : Output " c is the largest", goto step 9
Start 9 : Stop

Example 3 : Write an Algorithm to Find the Factorial of a Number Entered by the User :

Step 1 : Start
Step 2 : Declare variables n, factorial and i.
Step 3 : Initialize variables
factorial←1
i←1
Step 4 : Read value of n
Step 5 : Repeat the steps until i=n
factorial ← factorial*i
i←i+1
Step 6 : Display factorial
Step 7 : Stop

Characterisitics of an Algorithm

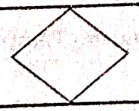
(1) **Unambiguous :** Algorithm should be clear and unambiguous.
(2) **Input :** An algorithm should have 0 or more well-defined inputs.
(3) **Output :** An algorithm should have 1 or more well-defined outputs, and should match the desired output.
(4) **Finiteness :** Algorithms must terminate after a finite number of steps.
(5) **Feasibility :** An algorithm should be feasible with the available resources.
(6) **Independent :** An algorithm should have step-by-step directions, which should be independent of any programming code.

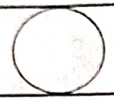
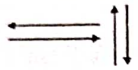
4. ***What are the basic components a flow chart? Explain with the help of examples.***

Flow Chart

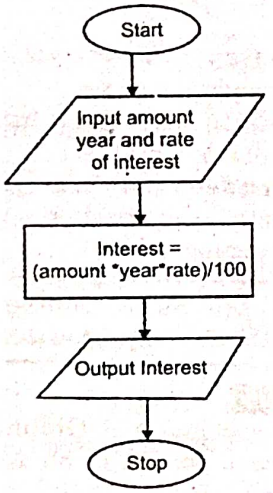
A Flow chart is a Graphical representation of an Algorithm or a portion of an Algorithm. Flow charts are drawn using certain special purpose symbols such as Rectangles, Diamonds, Ovals and small circles. These symbols are connected by arrows called flow lines.

Table : Uses of Different Symbol in the Flow Chart

Type of Element	Symbol	Use	Description
Oval		Terminal	It is used for start/stop or begins/ end.
Parallelogram		Input/ output	Making data available for processing (input) or recording of the process information (output).
Rectangle		Process	Any processing to be done. A process changes or moves data. An assignment operation can also be done.
Diamond		Condition/ Decision	Decision or switching type of operations

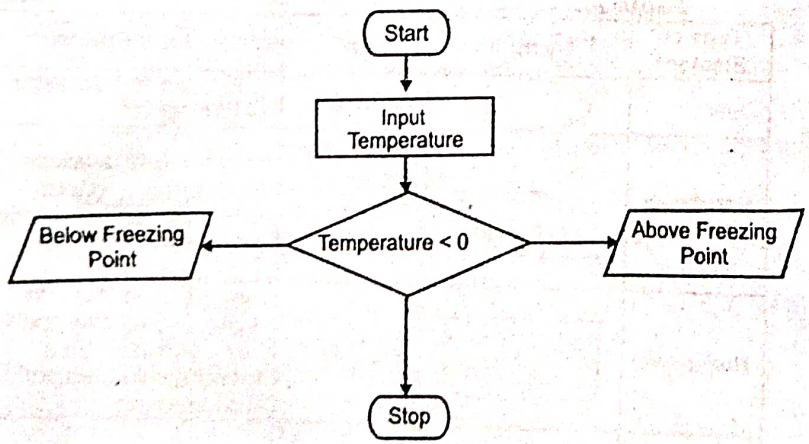
Circle		Connector	Used to connect different parts of flowchart.
Arrow		Flow	Joins two symbols and also represents flow of execution.

Example : Draw a Flow Chart to Calculate the Interest of a Bank Deposit.



(Figure : Flow Chart of Calculation of Interest)

Example 2 : Determine Whether a Temperature is below or above the Freezing Point



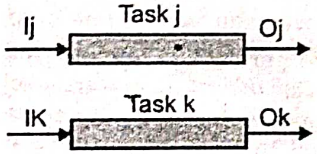
(Figure : Flow Chart of Analyzing Temperature)

5. *Discuss the use of high level programming language for the systematic development of programs.*

Use of High Level Programming Language for the Systematic Development of Programs :

Systemic development is an alternative definition of sustainable development which has its basis in systems thinking.

Customized Digital Computer :



(Figure : Rig Different Circuit for Each Task)

High Level Languages :

(1) Provides notation to describe problem solving strategies rather than organize data and instruction at machine- level.

(2) Improves programmer productivity by supporting features to abstract/reuse code, and to improve reliability/robustness of programs.

(3) Requires a compiler.

Evolution of Programming Languages :

FORTRAN (Formula Translator)

Goals : Scientific Computations
Efficiency of execution
Compile-time storage determination

Features : Symbolic Expressions
Subprograms
Absence of Recursion

COBOL :

Goal : Business Application

Features : Record/Structure; File Handling

ALGOL (Algorithmic Language) :

Goals : Communicating Algorithms

Features : Block Structure (Top-down design)

Recursion (Problem-solving strategy)

BNF-Specification

LISP (LIST Processing) :

Goals : Manipulating symbolic information

Features : List Primitives

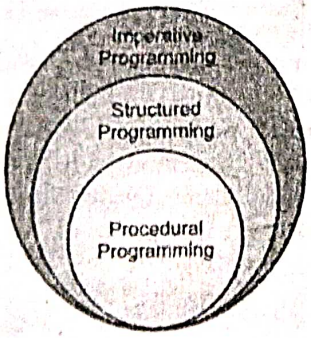
Interpreters/Environment

So, these are the Base of Systematic Development of High level Programming languages.

6. *Discuss Structured Programming Approach with Advantages and Disadvantages.*

Structured Programming Approach, as the word suggests, can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other. It doesn't support the possibility of jumping from one instruction to some other with the help of any statement like GOTO, etc. Therefore, the instructions in this approach will be executed in a serial and structured manner. The languages that support Structured programming approach are :

(1) C
(2) C++
(3) Java
(4) C#
..etc



(Figure)

On the contrary, in the Assembly languages like Microprocessor 8085, etc, the statements do not get executed in a structured manner. It allows jump statements like GOTO. So the program flow might be random.

The structured program mainly consists of three types of elements :

(1) Selection Statements
(2) Sequence Statements
(3) Iteration Statements

The structured program consists of well structured and separated modules. But the entry and exit in a structured program is a single-time event. It means that the program uses single-entry and single-exit elements. Therefore, a structured program is well maintained, neat and clean program. This is the reason why the Structured Programming Approach is well accepted in the programming world.

Advantages of Structured Programming Approach :

(1) Easier to read and understand
(2) User Friendly
(3) Easier to Maintain
(4) Mainly problem based instead of being machine based
(5) Development is easier as it requires less effort and time
(6) Easier to Debug
(7) Machine-Independent, mostly.

Disadvantages of Structured Programming Approach :

(1) Since it is Machine-Independent, So it takes time to convert into machine code.
(2) The converted machine code is not the same as for assembly language.
(3) The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.
(4) Usually the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

7. *Briefly discuss the history of 'C' programming.*

Introduction to 'C' Programming

C language facilitates a very efficient approach to the development and implementation of computer programs.

The History of C started in 1972 at the Bell Laboratories, USA where Dennis M. Ritchie proposed this language. In 1983 the American National Standards Institute (ANSI) established committee whose goal was to produce "an unambiguous and machine independent definition of the language C " while still retaining its spirit.

C is the programming language most frequently associated with UNIX. Since the 1970s, the bulk of the UNIX operating system and its applications have been written in C. Because the C language does not directly rely on any specific hardware architecture, UNIX was one of the first portable operating systems. In other words, the majority of the code that makes up UNIX does not know and does not care which computer it is actually running on. Machine-specific features are isolated in a few modules within the UNIX kernel, which makes it easy for you to modify them when you are porting to different hardware architecture. C was first designed by Dennis Ritchie for use with UNIX on DEC PDP-11 computers. The language evolved from Martin Richard's BCPL, and one of its earlier forms was the B language, which was written by Ken Thompson for the DEC PDP-7. The first book on C was The C Programming Language by Brian Kernighan and Dennis Ritchie, published in 1978.

In 1983, the American National Standards Institute (ANSI) established a committee to standardize the definition of C. The resulting standard is known as ANSI C, and it is the recognized standard for the language, grammar, and a core set of libraries. The syntax is slightly different from the original C language, which is frequently called K&R for Kernighan and Ritchie. There is also an ISO (International Standards Organization) standard that is very similar to the ANSI standard. It appears that there will be yet another ANSI C standard officially dated 1999 or in the early 2000 years.

8. *What is the structure of a 'C' program? Also explain the different section of a 'C' program.*

Structure of a 'C' Program

The program written in C language follows this basic structure. The sequence of sections should be as they are in the basic structure. A C program should have one or more sections but the sequence of sections is to be followed.

(1) Documentation section
(2) Linking section
(3) Definition section
(4) Global declaration section
(5) main function section
(6) Sub program or function section

Documentation Section : It comes first and is used to document the use of logic or reasons in your program. It can be used to write the program's objective, developer and logic details. The documentation is done in C language with /* and */. Whatever is written between these two are called comments.

Linking Section : This section tells the compiler to link the certain occurrences of keywords or functions in your program to the header files specified in this section.

e.g. #include <stdio.h>

Definition Section : It is used to declare some constants and assign them some value.

e.g. #define MAX 25

Here #define is a compiler directive which tells the compiler whenever MAX is found in the program replace it with 25.

Global Declaration Section : Here the variables which are used throughout the program (including main and other functions) are declared so as to make them global(i.e. accessible to all parts of program)

e.g. int i;
(before main())

Main Function Section : It tells the compiler where to start the execution from

```
main()
{
Point from execution starts
}
```

Main Function has two Sections :

(1) Declaration Section : In this the variables and their data types are declared.

(2) Executable Section : This has the part of program which actually performs the task we need.

Sub Program or Function Section : This has all the sub programs or the functions which our program needs.

How to Write First 'C' Program

To write a program for a computer to follow, we must go through a two-phase process: problem solving and implementation :

Problem-Solving Phase :

(1) Analysis and Specification : Understand (define) the problem and what the solution must do.

(2) General Solution (Algorithm) : Specify the required data types and the logical sequences of steps that solve the problem

(3) Verify : Follow the steps exactly to see if the solution really does solve the problem.

Implementation Phase :

(a) Concrete Solution (Program) : Translate the algorithm (the general solution) into a programming language.

(b) Test. Have the Computer follow the Instructions : Then manually check the results. If you find errors, analyze the program and the algorithm to determine the source of the errors, and then make corrections.

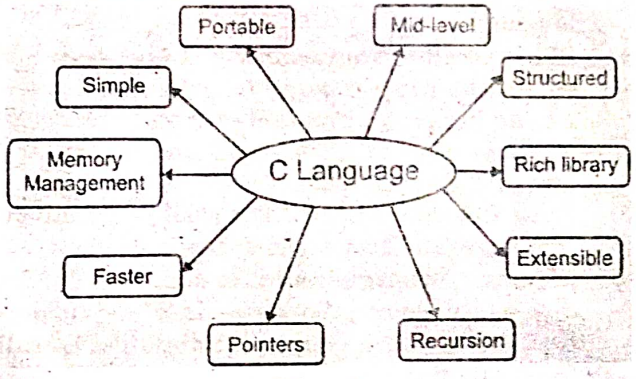
Example :

```
/* Simple Program in C */
#include<stdio.h>
void main()
{
printf("welcome to c programming");
} /* End of main */
```

9. ◆ ***What are salient features of C Language?*** **(2020-21)**

◆ ***What are the features of C Language?***

Features of C Language



(Figure)

C is the widely used language. It provides many features that are given below.

(1) Simple
(2) Machine Independent or Portable
(3) Mid-level programming language
(4) structured programming language
(5) Rich Library
(6) Memory Management
(7) Fast Speed
(8) Pointers
(9) Recursion
(10) Extensible

(1) **Simple :** C is a simple language in the sense that it provides a structured approach (to break the problem into parts), the rich set of library functions, data types, etc.

(2) **Machine Independent or Portable :** Unlike assembly language, c programs can be executed on different machines with some machine specific changes. Therefore, C is a machine independent language.

(3) **Mid-level Programming Language :** Although, C is intended to do low-level programming. It is used to develop system applications such as kernel, driver, etc. It also supports the features of a high-level

language. That is why it is known as mid-level language.

(4) **Structured Programming Language :** C is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify. Functions also provide code reusability.

(5) **Rich Library :** C provides a lot of inbuilt functions that make the development fast.

(6) **Memory Management :** It supports the feature of dynamic memory allocation. In C language, we can free the allocated memory at any time by calling the free() function.

(7) **Speed :** The compilation and execution time of C language is fast since there are lesser inbuilt functions and hence the lesser overhead.

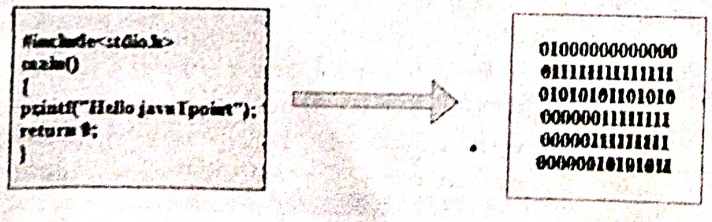
(8) **Pointer :** C provides the feature of pointers. We can directly interact with the memory by using the pointers. We can use pointers for memor structures, functions, array, etc.

(9) **Recursion :** In C, we can call the function within the function. It provides code reusability for every function. Recursion enables us to use the approach of backtracking.

(10) **Extensible :** C language is extensible because it can easily adopt new features.

10. ◆ ***Write a note on Compilation process in C.***
◆ ***What is a compilation?***

The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.

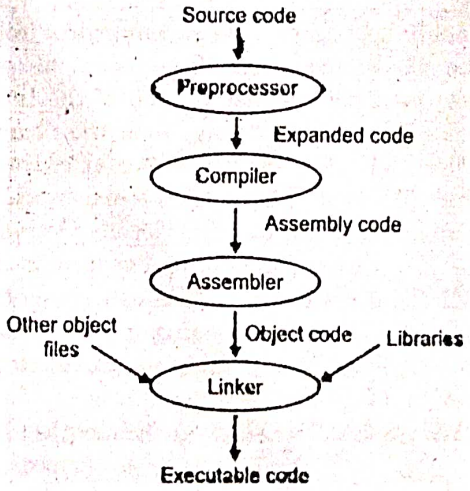


The C compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.

The preprocessor takes the source code as an input, and it removes all the comments from the source code. The preprocessor takes the preprocessor directive and interprets it. For example, if <stdio.h>, the directive is available in the program, then the preprocessor interprets the directive and replace this directive with the content of the 'stdio.h' file.

The following are the phases through which our program passes before being transformed into an executable form :

(1) Preprocessor
(2) Compiler
(3) Assembler
(4) Linker



(Figure)

Preprocessor : The source code is the code which is written in a text editor and the source code file is given an extension ".c". This source code is first passed to the preprocessor, and then the preprocessor expands this code. After expanding the code, the expanded code is passed to the compiler.

Compiler : The code which is expanded by the preprocessor is passed to the compiler. The compiler converts this code into assembly code. Or we can say that the C compiler converts the pre-processed code into assembly code.

Assembler : The assembly code is converted into object code by using an assembler. The name of the object file generated by the assembler is the same as the source file. The extension of the object file in DOS is '.obj,' and in UNIX, the extension is 'o'. If the name of the source file is 'hello.c', then the name of the object file would be 'hello.obj'.

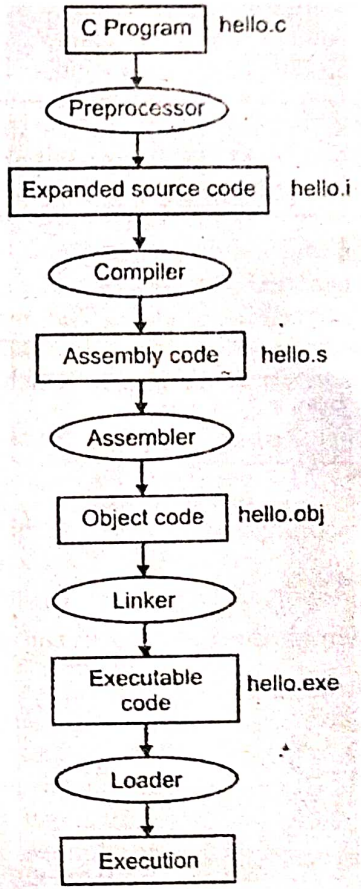
Linker : Mainly, all the programs written in C use library functions. These library functions are pre-compiled, and the object code of these library files is stored with '.lib' (or '.a') extension. The main working of the linker is to combine the object code of library files with the object code of our program. Sometimes the situation arises when our program refers to the functions defined in other files; then linker plays a very important role in this. It links the object code of these files to our program. Therefore, we conclude that the job of the linker is to link the object code of our program with the object code of the library files and other files. The output of the linker is the executable file. The name of the executable file is the same as the source file but differs only in their extensions. In DOS, the extension of the executable file is '.exe', and in UNIX, the executable file can be named as 'a.out'. For example, if we are using printf() function in a program, then the linker adds its associated code in an output file.

Let's Understand Through an Example :

hello.c

```
#include <stdio.h>
int main()
{
   printf("Hello javaTpoint");
   return 0;
}
```

Now, we will create a flow diagram of the above program :



(Figure)

In the above flow diagram, the following steps are taken to execute a program :

(1) Firstly, the input file, i.e., hello.c, is passed to the preprocessor, and the preprocessor converts the source code into expanded source code. The extension of the expanded source code would be hello.i.

(2) The expanded source code is passed to the compiler, and the compiler converts this expanded source code into assembly code. The extension of the assembly code would be hello.s.

(3) This assembly code is then sent to the assembler, which converts the assembly code into object code.

(4) After the creation of an object code, the linker creates the executable file. The loader will then load the executable file for the execution.

11. ◆ ***What is Token? What are the different types of token available in C language?*** **(2020-21)**
◆ ***What are tokens, keywords and identifiers.***

C - Tokens

Tokens are individual words and punctuations marks in English language sentence. The smallest individual units are known as C tokens. C tokens are the basic buildings blocks in C language which are constructed together to write a C program.

C tokens are of six types. They are :

(1) Keywords (eg : int, while),
(2) Identifiers (eg : main, total),
(3) Constants (eg : 10, 20),
(4) Strings (eg : "total", "hello"),
(5) Special symbols (eg : (), {}),
(6) Operators (eg : +, /,-,*)

Example : Identify the C- tokens in the given C program :

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

Tokens are given below :

main – keyword
{,}, (,) – delimiter
int – keyword
x, y, total – identifier

Keywords : Keywords are pre-defined words in a C compiler. Each keyword is meant to perform a specific function in a C program. C language supports 32 keywords which are given below :

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

Identifiers : Names of the variables and other program elements such as functions, array, etc, are known as identifiers.

There are few rules that govern the way variable are named (identifiers).

(1) Identifiers can be named from the combination of A-Z, a-z, 0-9, _ (Underscore).
(2) The first alphabet of the identifier should be either an alphabet or an underscore. Digits are not allowed.
(3) It should not be a keyword.
e.g. : name, ptr, sum

12. ***What are the types of constants in 'C' programming?***

Constants

A quantity that does not vary during the execution of a program is known as a constant supports two types of constants namely Numeric constants and character constants.

Numeric Constants :

(1) Example for an integer constant is 786,-127
(2) Long constant is written with a terminal 'l'or 'L',for example 1234567899L is a Long constant.
(3) Unsigned constants are written with a terminal 'u' or 'U', and the suffix 'ul' and 'UL' indicates unsigned long. for example 123456789u is a Unsigned constant and 1234567891ul is an unsigned long constant.
(4) The advantage of declaring an unsigned constant is to increase the range of storage.
(5) Floating point constants contain a decimal point or an exponent or both. For Eg : 123.4,1e-2,1.4E-4,etc.The suffixes f or F indicate a float constant while the absence of f or F indicate the double, l or L indicate long double.

Character Constants : A character constant is written as one character with in single quotes such as 'a'. The value of a character constant is the numerical value of the character in the machines character set. Certain character constants can be represented by escape sequences like '\n'. These sequences look like two characters but represent

only one. The following are the some of the examples of escape sequences :

Table : Escape Sequence and their Descriptions :

Escape Sequence	Description
\a	Alert
\b	Backspace
\f	Form feed
\n	New Line
\r	Carriage return
\t	Horizontal Tab
\v	Vertical Tab

String Constants : String literal is a sequence of zero or more characters surrounded by a double quote. Example, "I am a little boy". Quotes are not a part of the string.

To distinguish between a character constant and a string that contains a single character ex: 'a' is not same as "a". 'a' is an integer used to produce the numeric value of letter a in the machine character set, while "a" is an array of characters containing one character and a '\0' as a string in C is an array of characters terminated by NULL.

Enumeration Constant : It is a list of constant integer values.

Ex.: enum color { RED, Green, BLUE }

The first name in the enum has the value 0 and the next 1 and so on unless explicit values are specified.

If not all values specified, unspecified values continue the progression from the last specified value. For example

Enum months { JAN=1, FEB,MAR, ..., DEC –

Where the value of FEB is 2 and MAR is 3 and so on.

Enumerations provide a convenient way to associate constant values with names.

13. *Write short note on the variables in 'C' programming.*

Variables

A quantity that can vary during the execution of a program is known as a variable. To identify a quantity we name the variable.

For example if we are calculating a sum of two numbers we will name the variable that will hold the value of sum of two numbers as 'sum'.

14. ◆ ***What are basic data types available in "C"? Write the significance of each data type.*** **(2020-21)**

◆ ***Explain the various data types in 'C', giving suitable example of each with modifiers.***

Data Types

To represent different types of data in C program we need different data types. A data type is essential to identify the storage representation and the type of operations that can be performed on that data. There are two data types in C language, which are given below :

(1) Primary data types : int, char, float, double
(2) Derived data type : pointer, array, structure, union
(3) Void data type : void

Range of Primary Data Types : In the following table we have the range of the values of the primary or fundamental data type.

Primary data type is declared as per given below :

Example : int a, b;

Table : Data Type and Their Ranges

Data Type	Keyword	Equivalent Size (Bits)	Range
character	char	8	-128 to 127
signed character	signed char	8	-128 to 127
unsigned character	unsigned char	8	0 to 255
short integer	short int or short	8	-128 to 127
signed short integer	signed short int	8	-128 to 127
unsigned short integer	unsigned short int	8	0 to 255
integer	int	16	-32768 to 32767
signed integer	signed int	16	-32768 to 32767
unsigned integer	unsigned int	16	0 to 65535
long integer	long int or long	32	-2147483648 to 2147483647
signed long integer	signed long int	32	-2147483648 to 2147483647
unsigned long integer	unsigned long int	32	0 to 4294967295
floating point (6 decimal place)	float	32	3.4e-38 to 3.4e+38

double-precision floating point (12 decimal place)	double	64	1.7e-308 to 1.7e+308
extended double-precision floating point	long double	80	3.4e-4932 to 1.1e+4932

Derived Data Type : A derived data type is defined using combination of qualifiers along with the primitive data type. Derived types are created using basic data types with modified behavior and property.

Example : Array, Function, Pointer.

```
Declaration
int a[10];
char name [20];
```

Void Data Type : The void data type takes no value and is generally used with functions to denote that the function is not going to return any value.

Example : void main()

Here function main has no return data type.

15. *How to take input from the user and print output on the computer screen?*

Input and Output Statements

The simplest of input function is getchar to read a single character from the input device.

For Example : varname=getchar();

You need to declare varname.

The simplest of output function is putchar to output a single character on the output device.

For Example : putchar(varname)

The getchar() is used only for one input and is not formatted. Formatted input refers to an input data that has been arranged in a particular format, for that we have scanf function.

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

Control string specifies field format in which data is to be entered.

arg1, arg2... argn specifies address of location or variable where data is stored.

For Example : scanf("%d%d",&a,&b);

```
%d      used for integers
%f      floats
```



```
%l      long
%c      character
```

For formatted output you use printf function

For Example : printf("control string", arg1, arg2,...argn);

Example : Write a Program to Perform Input and Output Operation.

```
/* program to exhibit i/o */
#include<stdio.h>
main()
{
int a,b;
float c;
printf("Enter any number");
a=getchar();
printf("the char is ");
putchar(a);
printf("Exhibiting the use of scanf");
printf("Enter three numbers");
scanf("%d%d%f",&a,&b,&c);
printf("%d%d%f",a,b,c);
}
```

16. *What are types of error encountered while writing a 'C' program?*

Types of 'C' Program Errors

Error is an illegal operation performed by the user which results in abnormal working of the program.

There are mainly four types of errors :

(1) Syntax Errors : Errors that occur when you violate the rules of writing C syntax are known as syntax errors. This compiler error indicates something that must be fixed before the code can be compiled.

Most frequent syntax errors are :

(a) Missing Parenthesis (})

(b) Printing the value of variable without declaring it

(c) Missing semicolon

Example :

```
main()
{
  int x = 10;
  int y = 15;
  printf("%d", (x, y)) // semicolon missed
```

(2) **Run-time Errors :** Errors which occur during program execution (run-time) after successful compilation are called run-time errors. One of the most common run-time error is division by zero also known as Division error.

Example :

```
main()
    {
      int n = 9, div = 0;

      // wrong logic
      // number is divided by 0,
      // so this program abnormally terminates
      div = n/0;
       printf("result = %d", div);
    }
```

(3) **Linker Errors :** This error occurs when after compilation we link the different object files. These are errors generated when the executable of the program cannot be generated. This may be due to wrong function prototyping, incorrect header files.

Example :

```
void Main() // Here Main() should be main()
{
      int a = 10;
      printf("%d", a);
}
```

(4) **Logical Errors :** On compilation and execution of a program, desired output is not obtained when certain input values are given. These types of errors which provide incorrect output but appear to be error free are called logical errors.

Example :

```
int main()

    int i = 0;

    // logical error : a semicolon after loop
    for(i = 0; i < 3; i++);
    {
       printf("loop ");
       continue;
    }
    getchar();
    return 0;
}
```


17. ◆ *What is an operator?* (2020-21)
    ◆ *What are the various types of operators in 'C'?*

Operators

An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations. They form expressions.

C operators can be classified as

(1) Arithmetic operators
(2) Relational operators
(3) Logical operators
(4) Assignment operators
(5) Increment or Decrement operators
(6) Conditional operator
(7) Bit wise operators
(8) Special operators

18. ◆ *Explain the arithmetic operators in C language.* (2020-21)
    ◆ *What are the different bitwise operators used in C? Give an example of each.*

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable A holds 10 and variable B holds 20, then :

Table : Different Arithmetic Operator with Examples

Operator	Description	Example
+	Adds two operands.	a + b = 30
–	Subtracts second operand from the first.	a – b = -10
*	Multiplies both operands.	a * b = 200
/	Divides numerator by de-numerator.	b / a = 2
%	Modulus Operator and remainder of after an integer division.	b % a = 0
++	Increment operator increases the integer value by one.	a++
–	Decrement operator decreases the integer value by one.	a–

For Example : Write a 'C' Program to Demonstrate the Usability of Arithmetic Operator.

```
#include <stdio.h>
main()
{
  int a = 21;
  int b = 10;
  int c ;
  c = a + b;
  printf("Value of c is %d\n", c );
 c = a - b;
  printf("Value of c is %d\n", c );
  c = a * b;
  printf("Value of c is %d\n", c );
  c = a / b;
  printf("Value of c is %d\n", c );
  c = a % b;
  printf("Value of c is %d\n", c );
  c = a++;
  printf("Value of c is %d\n", c );
  c = a--;
  printf("Value of c is %d\n", c );
}
```

Output :

```
Value of c is 31
Value of c is 11
Value of c is 210
Value of c is 2
Value of c is 1
Value of c is 21
Value of c is 22
```

19. ◆ ***Explain the relational operators in C language.*** **(2020-21)**
    ◆ ***Discuss various relational operators in 'C' with examples.***

Relational Operators

We often compare two quantities and depending on their relation take certain decisions for that comparison we use relational operators.

Table : Different Relational Operators with Examples

Operator	Description	Example
==	It checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	It checks if the values of two operands	(A != B) is

	are equal or not. If the values are not equal, then the condition becomes true.	true.
>	It checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	It checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	It checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	It checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

For Example : Write a 'C' Program to Demonstrate the Usability of Relational Operator

```
#include <stdio.h>
main()
{
  int a = 21;
  int b = 10;
  int c ;
  if( a == b )
 {
    printf("a is equal to b\n" );
  }
 else
 {
    printf("a is not equal to b\n" );
  }
  if ( a < b )
 {
    printf("a is less than b\n" );
  }
 else
 {
    printf("a is not less than b\n" );
  }
  if ( a > b )
 {
    printf("a is greater than b\n" );
  }
 else
 {
    printf("a is not greater than b\n" );
  }
```

```
    a = 5;
    b = 20;
    if ( a <= b )
   {
      printf("a is either less than or equal to  b\n" );
    }
    if ( b >= a )
   {
      printf("b is either greater than  or equal to b\n" );
    }
  }
```

Output :

```
a is not equal to b
a is not less than b
a is greater than b
a is either less than or equal to  b
b is either greater than  or equal to b
```

20. ◆ ***Explain the logical operators in C language.*** **(2020-21)**

◆ ***Differentiate between bitwise operators and logical operators.***

Logical Operators

An expression of this kind which combines two or more relational expressions is termed as logical expressions. The following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0.

Table : Different Logical Operators with Examples

Operator	Description	Example
&&	It is called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	It is called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	It is called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

For Example : Write a 'C' Program to Demonstrate the Usability of Logical Operator.

```
#include <stdio.h>
main()
{
  int a = 5;
  int b = 20;
  int c ;
  if ( a && b )
  {
     printf("Condition is true\n" );
  }

  if ( a || b )
  {
     printf("Condition is true\n" );
  }

  a = 0;
  b = 10;

  if ( a && b )
  {
     printf("Condition is true\n" );
  }
  else
  {
     printf("Condition is not true\n" );
  }

  if ( !(a && b) )
  {
     printf("Condition is true\n" );
  }
}
```

Output :

```
Condition is true
Condition is true
Condition is not true
Condition is true
```

Bit-Wise Operators

'C' supports special operators known as bit wise operators for manipulation of data at bit level. They are not applied to float or double.

The following table lists the Bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13.

Table : Different Logical Operator with Example Space.

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e., -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Operator Precedence and Associativity

For Example : Write a Program to Demonstrate Bit-Wise Operator in 'C'.

```
#include <stdio.h>
main()
{
 unsigned int a = 60;                    /* 60 = 0011 1100 */
 unsigned int b = 13;                    /* 13 = 0000 1101 */
int c = 0;
 c = a & b;       /* 12 = 0000 1100 */
 printf("Value of c is %d\n", c );
 c = a | b;       /* 61 = 0011 1101 */
 printf("Value of c is %d\n", c );
 c = a ^ b;       /* 49 = 0011 0001 */
 printf("Value of c is %d\n", c );
 c = ~a;          /*-61 = 1100 0011 */
 printf("Value of c is %d\n", c );
 c = a << 2;      /* 240 = 1111 0000 */
 printf("Value of c is %d\n", c );
 c = a >> 2;      /* 15 = 0000 1111 */
 printf("Value of c is %d\n", c );
}
```

Output :

```
Value of c is 12
Value of c is 61
Value of c is 49
```



```
Value of c is -61
Value of c is 240
Value of c is 15
```

21. *Differentiate between increment and decrement operators in 'C'.*

Increment and Decrement Operators

Increment Operator (++) : Increment operator is used to increase the current value of a variable by 1. Increment operators are two types :

(1) Pre-Increment Operator : Pre-increment operator is used to increase the value of a variable by 1 before using in the expression.

For Example :

```
main()
{
int x= 4, y=5;
a=++x, b= ++y;      //Pre- increment operators (++x, ++y)
printf("%d%d",a,b);                    // printing value of a, b
}
```

Output : 5,6

(2) Post-Increment Operator : Post-increment operator is used to increase the value of a variable by 1 after using in the expression.

For Example :

```
main()
{
int x= 4, y=5;
a=x++, b= y++;                  // Post- increment operators (x++, y++)
printf("%d%d",a,b);                    // printing value of a, b
}
```

Output : 4,5

Decrement Operator (–) : Decrement operator is used to decrease the current value of a variable by 1. Decrement operators are two types :

(1) Pre-Decrement Operator : Pre-decrement operator is used to decrease the value of a variable by 1 before using in the expression.

For Example :

```
main()
{
int x= 4, y=5,a,b;
a=--x, b= --y;                  // Pre- decrement operators (--x, --y)
```

```
printf("%d%d",a,b);                    // printing value of a, b
}
```

Output : 3,4

(2) **Post-Decrement Operator** : Post-decrement operator is used to decrease the value of a variable by 1 after using in the expression.

```
main()
{
int x= 4, y=5,a,b;
a=x--, b= y--;          // Post- decrement operators (x--, y--)
printf("%d%d",a,b);                    // printing value of a, b
}
```

Output : 4,5

Note : In the previous examples, suppose we want to print value of x and y then what will be the result?

```
main()
{
int x= 4, y=5,a,b;
a=++x, b= ++y;  //Pre- increment operators (++x, ++y)
printf("%d%d",a,b);                    // printing value of a, b
printf("/n");                          // print for new line
 printf("%d%d",x,y);                   // printing value of x, y
}
```

Output : 5,6 5,6

```
main()
{
int x= 4, y=5,a,b;
a=x++, b= y++;          //Post-increment operators(x++, ++y)
printf("%d%d",a,b);                    // printing value of a, b
printf("/n");                          // print for new line
printf("%d%d",x,y);                    // printing value of x, y
}
```

Output : 4,5
5,6

```
main()
{
int x= 4, y=5,a,b;
a= --x, b= --y,      //Pre-decrement operators (--x, --y)
printf("%d%d",a,b);                    // printing value of a, b
printf("/n");                          // print for new line
printf("%d%d",x,y);                    // printing value of x, y
}
```

Output : 3,4
3,4

```
main()
{
int x= 4, y=5,a,b;
a=x--, b= y--;                 //Post-decrement operators(x--, --y)
```



```
printf("%d%d",a,b);                    // printing value of a, b
printf("/n");                          // print for new line
printf("%d%d",x,y);                    // printing value of x, y
}
```

Output : 4,5
3,4

22. *Explain the concept of conditional operator in 'C' with example.*

Conditional Operator

A ternary operator pair "?:" is available in C to construct conditional expressions of the form

expression1? expression 2 : expression 3;

It works as "if expression 1 is true then expression 2 will be the output else expression 3 will be the output".

For Example : Write a Program to Demonstrate Conditional Operator in 'C'.

```
#include <stdio.h>
int main ()
{
  // Local variable declaration:
  int x, y = 10;
  x = (y < 10)? 30: 40;
  printf( "value of x: ", x);
   return 0;
}
```

Output :
value of x : 40

23. *Explain the usability of special operators like sizeof, Pointer operators (& and *) in 'C' with example.*

Special Operators

These operators which do not fit in any of the above classification are, (comma), sizeof, Pointer operators (& and *) and member selection operators (. and ->). The comma operator is used to link related expressions together.

For Example : Write a Program to Explore Concept of Pointer Variable.

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

```
{
int *ptr, q;
q = 50;
/* address of q is assigned to ptr */
ptr = &q;
/* display q's value using ptr variable */
printf("%d", *ptr);
return 0;
}
```

Output : 50

Example : Write a Program to Explore sizeof Operator in 'C'.

```
#include <stdio.h>
int main()
{
int a;
char b;
float c;
double d;
printf("Storage size for int data type:%d \n",sizeof(a));
printf("Storage size for char data type:%d \n",sizeof(b));
printf("Storage size for float data type:%d \n",sizeof(c));
printf("Storage size for double data type:%d\n",sizeof(d));
return 0;
}
```

Output :

```
Storage size for int data type:4
Storage size for char data type:1
Storage size for float data type:4
Storage size for double data type:8
```

24. ◆ ***What do you mean by operator precedence?*** **(2020-21)**

◆ ***What do you mean by operator precedence in 'C'? Also explain associativity of different operators.***

Operator Precedence and Associativity

Operator precedence determines which operator is evaluated first when an expression has more than one operator.

For Example : 100-2*30 would yield 40, because it is evaluated as 100 – (2*30) and not (100-2)*30. The reason is that multiplication * has higher precedence than subtraction (-).

Associativity is used when there are two or more operators of same precedence is present in an expression. For example multiplication and division arithmetic operators have same precedence, let's say we have an expression 5*2/10, this expression would be evaluated as (5*2)/10 because the associativity is left to right for these operators. Similarly 20/2*5 would be calculated as. (20*2)/5.

The following table explores the operator precedence and their associativity :

Table : Operator Precedence and Associativity

Operator	Associativity
( )	Left to Right
[]	Left to Right
->	Left to Right
. –	Right to Left
— ++	Right to Left
~	Right to Left
& *	Right to Left
(data type)	Right to Left
sizeof	Right to Left
>> <<	
*, /, %	Left to Right
+, –	Left to Right
<	Left to Right
>	Left to Right
<=	Left to Right
>=	Left to Right
==	Left to Right
!=	Left to Right
&&	Left to Right
||	Left to Right
!	Right to Left
&	Left to Right
^	Left to Right
|	Left to Right
=	Right to Left
*=	Right to Left
/=	Right to Left
%=	Right to Left
+=	Right to Left

-=	Right to Left
&=	Right to Left
^=	Right to Left
|=	Right to Left
<<=	Right to Left
>>=	Right to Left
,	Right to Left
?:	Right to Left

25. ***Design an algorithm and draw a corresponding flow chart to convert a decimal number to its binary equivalent.*** **(2020-21)**

Decimal Number System

A number system with a base 10 is known as decimal number system. Hence, decimal numbers are denoted with a base 10.

This number system consists of 10 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Each digit in the decimal system has a position and every digit is ten times more significant than the previous digit.

For example: (461)10, (370)10, (890)10, (400)10, etc.

Binary Number System

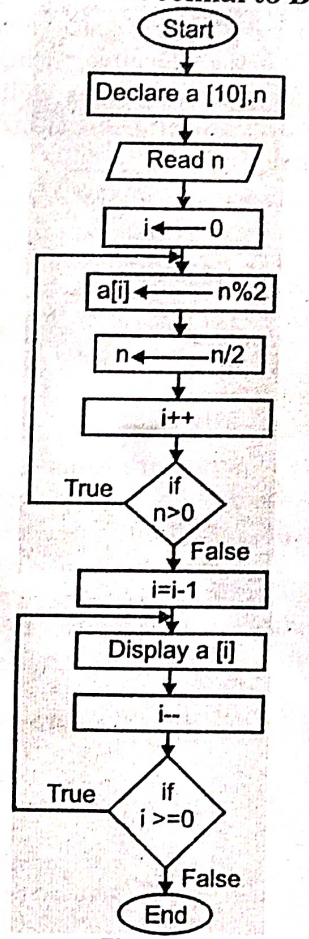
A number system with a base 2 is known as binary number system. Hence, binary numbers are denoted with a base 2. It consists of two values: 0 and 1.

Each digit in this system is said to be a bit. For example: (110101)2, (10101101)2, (10000)2, (10)2, etc.

Algorithm to Convert Decimal to Binary :

Input : Decimal number (n);
Output : Binary format of "n";
Step-1 : Start
Step-2 : Declare a[10] , n , i;
Step-3 : Read n (Decimal value);
Step-4 : for(i = 0 ; n > 0 ; i++){ //repeat till n > 0
Step-4.1 : a[i] ← n % 2;
Step-4.2 : n ← n / 2;
}
//Printing the output in Binary format
Step-5 : for(i = i – 1 ; i >= 0 ; i++){ //repeat till i >= 0
Step-5.1 : Display a[i];
}
Step-6 : End

Flowchart to Convert Decimal to Binary :



Figure

26. ***Differentiate between Global variable & extern variable.*** **(2020-21)**

Global Variable : There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions. Global variables are automatically initialized to 0 at the time of declaration.

External Variable : The variables that have to be used by many functions and in different files can be declared as

external variables. The initial value of an uninitialized external variable is zero.

The declaration of an external variable declares the type and name of the variable, while the definition reserves storage for the variable as well as behaves as a declaration. The keyword extern is specified in declaration but not in definition.

❑❑

CONDITIONAL PROGRAM EXECUTION

1. ◆ *What are Selection statements?*
   ◆ *Differentiate between if-else, ladder if and nested if statements.*

Selection Statements

if statements : We have a number of situations where we may have to change the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met.

The if statement is a two way decision statement and is used in conjunction with an expression. It takes the following form :

if(test expression)

If the test expression is true then the statement block after if is executed otherwise it is not executed

Syntax :

```
if (test expression)
{
statement block;
}
```

For Example

```
/* program for it */
#include<stdio.h>
main()
{
int a,b;
printf("Enter two numbers");
scanf("%d%d",&a,&b);
if (a>b)
printf(" a is greater");
if (b>a)
printf("b is greater");
}
```

Output :

```
Enter two numbers
5 10
b is greater
```

The if -else statement : If you have another set o statement to be executed if condition is false then if-else i used.

Syntax :

```
if (test expression)
{
statement block1;
}
else
{
statement block2;
}
For example:

/* program for if-else */
#include<stdio.h>
main()
{
int a,b;
printf("Enter two numbers");
scanf("%d%d",&a,&b):
if a>b
printf(" a is greater");
else
printf("b is greater");
}
```

Output :

```
Enter two numbers
5 10
b is greater
```

Nesting of if..else statement : If more than one if els statement exist then

Syntax :

```
if(text cond1)
if (test expression2)
{
statement block1;
}
else
{
statement block 2;
}
}
else
{
statement block2;'
}
{
```


if else ladder statement :

Syntax :

```
if(condition1)
{
statement1;
}
else if(condition2)
{
statement 2;
}
else if(condition3)
{
statement n;
}
else
{
default statement.
}
```

Note : The nesting of if-else depends upon the conditions with which we have to deal.

2. ◆ ***Write and explain about switch statement.*** **(2020-21)**

◆ ***Write the syntax of switch statement. Explain with the help of an example.***

Switch Statement

If for suppose we have more than one valid choices to choose from then we can use switch statement in place of if statements.

Syntax :

```
switch(expression)
{
case value-1 : statement-1 ;
break;
case value-2 : statement-2;
break;
------
------
default: statement;
}
/* Program to implement switch */
#include<stdio.h>
main()
{
int marks,index;
```

```
char grade[10];
printf("Enter your marks");
scanf("%d",&marks);
index=marks/10;
switch(index)
{
case 10 :
case 9:
case 8:
case 7:
case 6: grade="first";
break;
case 5 : grade="second";
break;
case 4 : grade="third";
break;
default : grade ="fail";
}
printf("%s",grade);
}
```

Limitations of Switch :

(1) Logical operators cannot be used with switch.

(2) Switch can only have int or char data type. No fl
data type.

3. *Explain the purpose of default in switch-case.*

Purpose of Default in Switch-Case

Default clause in switch statement used to indic
that the desired option is not available with the sw
case statement. It is similar to else statement o
statement which is used when the condition does
satisfy.

**4. *How does switch statement differ from nested
Show using an example.***

Switch Statement

The switch statement is a multi-branch lang
construct. We can create more than two branche
the switch statement. This is in contrast to the statem

The general format of a switch statement
follows :

```
switch (Expression)
{
```



```
case Constant :
statement ;
break ;
...
default :
statement ;
break ;
}
```

The switch expression is evaluated and checked against each of the case compile-time constants. If a constant matches the switch expression, the case statement is executed. If the case also contains a break statement, the program then jumps out of the switch. If there is no break statement, the program continues evaluating the other case statements.

If no matches are found, the default statement is executed. If there are no matches and no default, none of the statements inside the switch are executed.

Each of the previous statement lines can be replaced with a block of statements by enclosing the block in {..} braces.

Examples :

```
switch (Debtor. Account No)
{       case "1000" :
        do_something;
        break;
        case "2000"
          do_something_else;
        break;
        default :
        default_statement:
        break; }
```

It is possible to make the execution drop through case branches by omitting a break statement. For example :

```
switch (x)
{case 10:
a = b;
case 11:
c = d;
break;
case 12:
e = f;
break;}
```

Here, if x is 10, b is assigned to a, and d is assigned to c, the break statement is omitted after the case 10:

statement. If x is 11, d is assigned to c. If x is 12, f is assigned to e.

Nested if-else : An entire if-else construct within either the body of the if statement or the body of an else statement. This is called nesting of if.

```
if (condition)
{
// statements;
}
else
{
if (condition)
{
// statements;
}
else
{
// statements;
}
```

5. *Differentiate between break and continue statements.*

Break and Continue Statements

Break : The break statement terminates the loop (for, while and do...while loop) immediately when it is encountered. Its syntax is :

```
break;
```

Example 1 : break statement

```
// Program to calculate the sum of maximum of 10 numbers
// If negative number is entered, loop terminates and sum is displayed
# include <stdio.h>
int main()
{
   int i;
   double number, sum = 0.0;
   for(i=1; i <= 10; ++i)
   {
      printf("Enter a n%d: ",i);
      scanf("%lf",&number);
      // If user enters negative number, loop is terminated
      if(number < 0.0)
      {
         break;
      }
```



```
      sum += number; // sum = sum + number;
   }
   printf("Sum = %.2lf",sum);
   return 0;
}
```

Output :

```
Enter a n1: 2.4
Enter a n2: 4.5
Enter a n3: 3.4
Enter a n4: -3
Sum = 10.30
```

Continue : The continue statement skips statements after it inside the loop. Its syntax is :

```
continue;
```

Example 2 : continue statement

```
// Program to calculate sum of maximum of 10 numbers
// Negative numbers are skipped from calculation
# include <stdio.h>
int main()
{
   int i;
   double number, sum = 0.0;
   for(i=1; i <= 10; ++i)
   {
      printf("Enter a n%d: ",i);
      scanf("%lf",&number);
      if(number < 0.0)
      {
         continue;
      }
      sum += number; // sum = sum + number;
   }
   printf("Sum = %.2lf",sum);
   return 0;
}
```

Output :

```
Enter a n1: 1.1
Enter a n2: 2.2
Enter a n3: 5.5
Enter a n4: 4.4
Enter a n5: -3.4
Enter a n6: -45.5
Enter a n7: 34.5
Enter a n8: -4.2
Enter a n9: -1000
Enter a n10: 12
Sum = 59.70
```

6. *Write a program that accepts marks of 5 subjects and finds total and percentage of student.*

```
# include <stdio.h>
# include <conio.h>
void main()
{   int n1,n2,n3,n4,n5;
 int tot, per;
 printf ("enter marks of 5 subjects\n");
 scanf ("%d%d%d%d%d", &n1, &n2, &n3, &n4, &n5);
tot= n1+n2+n3+n4+n5;
 per= ( tot*100)/500;
 printf ("Total = %d", tot);
printf ("\n Percentage = %d", per);
getch ();
}
```

7. *Write a program that accepts temperature in celsius and converts it into fahrenheit. F = (9C/5) + 32*

```
#include<stdio.h>
#include<conio.h>
void main()
{
 float F,C;
 clrscr( );
 printf ("\n enter temprature in centigrade");
 scanf ("%f",&C);
 F = 32 + (9 * C) /5 ;
 printf(" temp in centigrade =%f, \n temp in farenheit is %f",C,F);
 getch();
}
```

8. *Write a program that swaps values of two variable using third variables.*

```
# include <stdio.h>
# include <conio.h>
void main()
{
int a, b, temp ;
clrscr( );
printf ("\n enter number1");
scanf ("d",&a);
printf ("\n enter number2");
```



```
 scanf ("d",&b);
 printf("\n BEFORE SWAPPING");
 printf("a=%d and b=%d",a,b);
 // swapping code
 temp = a;
 a=b;
 b=temp;
 printf("\n AFTER SWAPPING");
 printf("a=%d and b=%d",a,b);
 getch();
 }
```

9. *Write a program that finds greatest of two numbers.*

```
# include<stdio.h>
# include<conio.h>
void main()
{   int a,b;
clrscr( );
printf ("\n enter two numbers");
scanf ("%d %d",&a, &b);
if (a>b)
{ printf(" a is greater than b \n"); }
else
{ printf(" b is greater than a \n"); }
getch( );
}
```

10. *Write a program that finds whether the year is leap year or not.*

```
# include<stdio.h>
# include<conio.h>
void main()
{
 int year;
 clrscr( );
 printf ("enter the year\n");
 scanf ("%d",&year);

if( (year % 4==0) && (year % 100!=0) ||(year % 400==0) )
    printf(" IT IS A LAEP YEAR ");
else
   printf(" IT IS NOT A LEAP YEAR " );
getch();
}
```

11. ***Write a program that takes two operands and one operator from the user and perform the operation and print the result. (or simulate calculator using switch).***

```
# include<stdio.h>
# include<conio.h>
void main ( )
{  int a ,b ,res ;
  char op;
  clrscr( );
  printf ( "enter value of a and b operand\n");
  scanf ( " %d %d ", &a, &b);
  printf ( " enter operator \n );
  op= getchar (op);
  switch (op )
  {   case '+' :     res= a+b;
                     printf (" result =%d ", res);
                     break;
      case '-' :     res= a - b;
                     printf (" result =%d ", res);
                     break;
      case '* ' :    res= a * b;
                     printf (" result =%d ", res);
                     break;
      case ' / ' :   res= a / b;
                     printf (" result =%d ", res);
                     break;
      default :      printf (" you have entered wrong operator \n");
  }
 getch ( );
 }
```

12. ***Write a program to compute the area and perimeter of a various geometrical figures such as square, circle, rectangle, triangle, depending on the choice, input the required parameter and computes its area and perimeter.***

e.g.- enter 1 for finding area of a rectangle
enter 2 for finding area of a square
enter 3 for finding area and circumference of a circle.

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



```
{
 float l,b , area, cir ,r, s;
 float pi= 3.14;
char choice;
 clrscr ( );
 printf("enter 1- finding area of a rectangle \n");
 printf("enter 2- finding area of a Square \n");
 printf("enter 3- finding area and circumference of a circle \n");
 printf("enter 4- for exit ");
 printf( "enter your choice \n");
 scanf( " %c ", &choice);
 switch (choice )
 {   case 1 :  printf ("enter length and width of rect ");
               scanf( " %f %f " , &l ,&b);
               area= l * b;
               printf ( "area = %f ", area );
               break;
   case 2 :    printf ("enter side of square ");
               scanf( " %f " , &s);
               area= s *s ;
               printf ( "area = %f ", area );
               break;
   case 3 :    printf ("enter radius of circle ");
               scanf( " %f " , &r);
               area= pi * r * r ;
               cir = 2 * pi * r ;
               printf ( "area = %f ", area );
               printf ( "circunference = % f ", cir );
               break;
   case 4 :    exit (0);
   default :   printf (" you have entered wrong choice \n");
    }
  getch();
}
```

13. ***What do you mean by looping and its type?***

Looping

Sometimes we require a set of statements to be executed repeatedly until a condition is met. We have two types of looping structures. One in which condition is tested before entering the statement block called entry control. The other in which condition is checked at exit called exit controlled loop.

There are three types of loop, which is given below :

(1) for loop
(2) while loop
(3) do-while loop

14. ◆ *Write and explain syntax of "for" loop.* (2020-21)

◆ *Write the syntax of for loop in 'C' programming and also write a program to print numbers from 1 to 10 using for loop.*

"for" Loop

It is also an entry control loop that provides a more concise structure.

Syntax :

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

Program to print numbers from 1 to 10

```
#include <stdio.h>
 void main ()
{
  int i;
  /* for loop execution */
  for( i = 1; i<=10; i++ )
{
    printf("%d\n", i);
  }
 }
```

Output :

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

15. ◆ *List the differences between while loop and do – while loop. Write a C program to find sum of Natural Numbers from 1 to N using for loop.* (2020-21)

◆ *Differentiate between while and do-while loop.*

"while" Loop

It is an entry controlled loop. The condition is evaluated and if it is true then body of loop is executed. After execution of body the condition is once again evaluated and if true body is executed once again. This goes on until test condition becomes false.

Syntax :

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

For example : Write a program to print numbers from 1 to 10

```
#include <stdio.h>
 int main ()
{
  int i=1;;
  /* while loop execution */
  while (i<=10)
{
    printf("%d\n", i);
    i++;
  }
   return 0;
}
```

Output :

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

"do-while" Loop

The while loop does not allow body to be executed if test condition is false. The do while is an exit controlled loop and its body is executed at least once.

Syntax :

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

For Example : Write a program to print numbers from 1 to 10

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

```
int main ()
{
  int i=1;;
  /* do-while loop execution */
  do
{
    printf("%d\n",*i);
    i++;
  }
While (i<=10);
   return 0;
}
```

Output :

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

C Program to find Sum of Natural Numbers from 1 to N Using for Loop :

```
#include <stdio.h>
#include <conio.h>
void main()
{
intnum, i, sum = 0;
printf(" Enter a positive number: ");
scanf("%d", &num);

// executes until the condition remains true.

  {
// at each iteration the value of i is added to the sum variable
sum = sum + i;
  }

// display the sum of natural number
printf("\n Sum of the first %d number is: %d", num, sum);
getch();
}
```

Output :

```
Enter a positive number: 25
Sum of the first 25 number is: 325
```


16. *What is the use of "goto" and "labels" in 'C'?*

"goto" and "labels" Statement

In C, it is used to escape from multiple nested loops, or to go to an error handling exit at the end of a function. You will need a label when you use a goto; this example shows both.

```
goto L1;
/* whatever you like here.*/
L1: /* anything else */
```

A label is an identifier followed by a colon. Labels have their own „name space" so they can't clash with the names of variables or functions. The name space only exists for the function containing the label, so label names can be re-used in different functions. The label can be used before it is declared, too, simply by mentioning it in a goto statement. Labels must be part of a full statement, even if it's an empty one. These usually only matters when you're trying to put a label at the end of a compound statement—like this.

```
label_at_end: ; /* empty statement */
}
```

The goto works in an obvious way, jumping to the labeled statements. Because the name of the label is only visible inside its own function, you can't jump from one function to another one.

For Example :

```
// Program to calculate the sum and average of maximum of 5 numbers
// If user enters negative number, the sum and average of previously entered positive numbers are displayed
# include <stdio.h>
int main()
{
   const int maxInput = 5;
   int i;
   double number, average, sum=0.0;
   for (i=1; i<=maxInput; ++i)
   {
      printf("%d Enter a number: ", i);
      scanf("%lf",&number);
   // If user enters negative number, flow of program moves to label jump
      if(number < 0.0)
         goto jump;
```

```
            sum += number; // sum = sum+number;
        }
        jump:
        average=sum/(i-1);
        printf("Sum = %.2f\n", sum);
        printf("Average = %.2f", average);
        return 0;
    }
```

Output :

```
Enter a number: 3
Enter a number: 4.3
Enter a number: 9.3
Enter a number: -2.9
Sum = 16.60
```

17. *What is the role of assignment operator in 'C'?*

Assignment Operators :

They are used to assign the result of an expression to a variable. The assignment operator is '='.

For Example :

v op = exp ;
v is variable
op binary operator
exp expression
op= short hand assignment operator

Short Hand Assignment Operators :

Simple Assignment Operators	Short Hand Assignment Operators
a = a + 1	a + =1
a = a - 1	a - =1
a = a % b	a % =b

18. *Define functions and its syntax with its advantages.*

Functions

A function is a self contained program segment that carries out some specific well defined tasks.

Advantages of Functions :

(1) Write your code as collections of small functions to make your program modular
(2) Structured programming
(3) Code easier to debug
(4) Easier modification
(5) Reusable in other programs

Syntax :

```
return_type func_name( parameter list )
{
Declarations;
Statements;
}
```

19. *Explain standard header files and functions of each header file in 'C'.*

Standard Library Functions and Header Files

C functions can be classified into two categories, namely, library functions and user-defined functions. Main is the example of user-defined functions. printf and scanf belong to the category of library functions. The main difference between these two categories is that library functions are not required to be written by us where as a user-defined function has to be developed by the user at the time of writing a program. However, a user-defined function can later become a part of the c program library.

'C' Standard Library Functions : The C language is accompanied by a number of library functions that perform various tasks. The ANSI committee has standardized header files which contain these functions.

Some of the Header files are :

<math.h> Mathematical functions
<stdio.h> standard I/O library functions
<stdlib.h> Utility functions such as string conversion routines memory allocation routines , random number generator etc.
<string.h> string Manipulation functions
<time.h> Time Manipulation functions

math.h Functions : The math library contains functions for performing common mathematical operations. Some of the functions are :

abs : returns the absolute value of an integer x
cos : returns the cosine of x, where x is in radians
exp : returns "e" raised to a given power
fabs : returns the absolute value of a float x
log : returns the logarithm to base e
log10 : returns the logarithm to base 10
pow : returns a given number raised to another number
sin : returns the sine of x, where x is in radians
sqrt : returns the square root of x

tan : returns the tangent of x, where x is in radians
ceil : The ceiling function rounds a number with a decimal part up to the next highest integer (written mathematically as $\lceil x \rceil$)
floor : The floor function rounds a number with a decimal part down to the next lowest integer (written mathematically as $\lfloor x \rfloor$)

string.h Functions : There are many functions for manipulating strings. Some of the more useful are :
strcat : Concatenates (i.e., adds) two strings
strcmp : Compares two strings
strcpy : Copies a string
strlen : Calculates the length of a string (not including the null)
strstr : Finds a string within another string
strtok : Divides a string into tokens (i.e., parts)

stdio.h Functions :
Printf : Formatted printing to stdout
Scanf : Formatted input from stdin
Fprintf : Formatted printing to a file
Fscanf : Formatted input from a file
Getc : Get a character from a stream (e.g, stdin or a file)
putc : Write a character to a stream (e.g, stdout or a file)
fgets : Get a string from a stream
fputs : Write a string to a stream
fopen : Open a file
fclose : Close a file
stdlib.h functions :
Atof : Convert a string to a double (not a float)
Atoi : Convert a string to an int
Exit : Terminate a program, return an integer value
free : Release allocated memory
malloc : Allocate memory
rand : Generate a pseudo-random number
system : Execute an external command
time.h Functions : This library contains several functions for getting the current date and time.
Time : Get the system time
Ctime : Convert the result from time() to something meaningful

20. *What are the different types of parameter passing mechanism? Explain in detail.*

Function Calling

There are different types of function calling. Depending on the number of parameters it can accept, function can be classified into following 4 types :

Table : Different Types of Parameters Passing Mechanism in the Function

Function Type	Parameter	Return Value
Type 1	Accepting Parameter	Return Value
Type 2	Accepting Parameter	Not Return Value
Type 3	Not Accepting Parameter	Return Value
Type 4	Not Accepting Parameter	Not Return Value

Example :
Type 1 :
Method definition :

```
int add(int i, int j)
{
 return i + j;
}
```

Function Calling :

```
int answer = sum(2,3);
```

Type 2 :
Method Definition :

```
void add(int i, int j)
{
 printf("%d",i+j);
}
```

Function Calling :

```
sum(2,3);
```

Type 3 :
Method Definition :

```
int add()
{
 int i, int j;
 i = 10;
 j = 20;
 return i + j;
}
```

Function Calling :

```
result = add();
```

Type 4 :

Method definition :

```
void add()
{
 int i, int j;
 i = 10;
 j = 20;
 printf("Result : %d",i+j);;
}
```

Function Calling :

```
add();
```

21. *Differentiate between call by value method and call by reference in 'C' with the help of an example.*

Difference between Function Call by Value and Call by Reference Methods.

Call by Value	Call by Reference
In call by value, a copy of actual arguments is passed to formal arguments of the called function and any change made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function.	In call by reference, the location (address) of actual arguments is passed to formal arguments of the called function. This means by accessing the addresses of actual arguments we can alter them within from the called function.
In call by value, actual arguments will remain safe, they cannot be modified accidentally.	In call by reference, alteration to actual arguments is possible within from called function; therefore the code must handle arguments carefully else you get unexpected results.

For Example : Write a Program to Swap Two Numbers using Call by Value Method.

```
#include <stdio.h>
 void swapByValue(int, int); /* Function Prototype */
 int main()
 {
  int n1 = 10, n2 = 20;
   /* actual arguments will be as it is */
  swapByValue(n1, n2);
 }
 void swapByValue(int a, int b)
 {
      int t;
      t = a;
a = b;
```



```
b = t;
printf("n1: %d, n2: %d\n", a, b);
}
```

Output :

```
n1: 20, n2: 10
```

22. *Write a Program to Swap Two Numbers using Call by Reference Method.*

```
#include <stdio.h>
 void swapByReference(int*, int*); /* Function Prototype */
 int main() /* Main function */
 {
  int n1 = 10, n2 = 20;
   /* actual arguments will be altered */
  swapByReference(&n1, &n2);
  printf("n1: %d, n2: %d\n", n1, n2);
 }
 void swapByReference(int *a, int *b)
 {
  int t;
  t = *a; *a = *b; *b = t;
printf("n1: %d, n2: %d\n", a, b);
}
```

Output :

```
n1: 20, n2: 10
```

23. *Differentiate between formal parameter and actual parameter in the function.*

Formal Parameters vs. Actual Parameters

There are two other categories that you should know about that are also referred to as "parameters". They are called "parameters" because they define information that is passed to a function.

(1) Actual parameters are parameters as they appear in function calls.

(2) Formal parameters are parameters as they appear in function declarations.

For Example : Write a program to calculate total bill of the expenses done by fred, franck and franny.

```
#include <stdio.h>
int calculate_bill (int, int, int);
int main()
{
 int bill;
```

```
int fred = 25;
int frank = 32;
int franny = 27;
bill = calculate_bill (fred, frank, franny);
printf("The total bill comes to $%d.00.\n", bill);
return 0;
}
int calculate_bill (int diner1, int diner2, int diner3)
{
int total;
total = diner1 + diner2 + diner3;
return total;
}
```

In the function main in the example above, fred, frank, and franny are all actual parameters when used to call calculate_bill. On the other hand, the corresponding variables in calculate_bill (namely diner1, diner2 and diner3, respectively) are all formal parameters because they appear in a function definition.

24. ***How did you pass an array to a function? Explain with the help of an example.***

Array Passing to a Function

We pass the address of an array while calling a function. We will only send in the name of the array as argument, which is nothing but the address of the starting element of the array, or we can say the starting memory address.

For Example : Write a function to find out average of all the elements of the array and print it.

```
#include<stdio.h>
float findAverage(int marks[]);
void main()
{
  float avg;
  int marks[] = {99, 90, 96, 93, 95};
  avg = findAverage(marks);        // name of the array is
passed as argument.-
  printf("Average marks = %.1f", avg);
}
float findAverage(int marks[])
{
  int i, sum = 0;
  float avg;
```



```
for (i = 0; i <= 4; i++) {
  sum += marks[i];
}
avg = (sum / 5);
return avg;
}
```

25. ◆ ***What is Recursion?*** **(2020-21)**
◆ ***What is recursion and recursive function?***

Recursive Functions

A function called by itself is called recursive function and this process is known as recursion.

For Example : Write a function to print Fibonacci series using recursive function.

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

26. ***Write a program to print sum of all number upto given number.***

```
# include <stdio.h>
# include <conio.h>
void main()
{
int sum, num, i;
sum= 0;
clrscr( );
printf ("enter the target number\n ");
scanf ("%d",&num);
for (i=1; i<=num; i++)
{
 Sum=sum + i ;
}
printf("\n result is %d", sum);
getch();
}
```

27. ***Write a program to print " Fibonacci series ".. ex =0 1 1 2 3 5 8 13 ....upto n terms.***

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

```
void main( )
{
 int f, s, t, term;
 int i;
 clrscr();
 printf("Enter number upto which you want the series\n");
 scanf("%d", & term);
 f=0;
 s=1;
 printf ("%d ",f);
 printf ("%d ",s);
 for ( i=3; i<=term; i++)
    {    t=f+s;
         printf ("%d ", t);
         f=s;
         s=t;
    }
 getch( );
 }
```

28. ***Write a program that checks whether the number is prime or not.***

```
# include<stdio.h>
# include<conio.h>
void main( )
{
int n, count =0, i;
clrscr( );
printf ("enter number\n");
scanf ("%d", &n);
//checking for prime or not
for( i=1; i<=n; i++ )
{
  if (n%i = = 0)
      { count++; }
}
if (count = =2)
  printf("number is prime");
else
  printf("number is not prime");
getch();
}
```

29. ***Write a program to find sum of digit (eg- number is 234 then output is 2 + 3 + 4 = 9).***

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



```
void main()
{
  int num, i, digit;
  int sum= 0;
  clrscr( );
  printf ("enter any number\n");
  scanf ("%d", &num);
 //code for accessing each digit of a number
  while (num!=0)
  {
    digit =num%10 ;
    sum =sum+digit ;
    num =num/10 ;
  }
  printf(" Sum of digits =%d",sum);
  getch ( );
}
```

30. ***Write a program to print " armstrong series" from 1 to 1000 // eg = 153 is the armstrong num...then 13 + 53 + 33 = 1 + 125 + 27= 153.***

```
# include<stdio.h>
# include<conio.h>
void main ( )
{   int sum ,n ,num , cube, digit;
  clrscr ( );
  for ( num=1; num<=1000; num++)
  {   n = num;
      sum = 0;
      while ( n!=0)
       {
          digit= n%10;
          cube= digit *digit *digit;
          sum= sum + cube;
          n= n/10;
       }
      if (sum == num)
      printf ("%d ", num);
  }
 getch();
}
```

31. ◆ ***Write a program in C to calculate the factorial of given number using recursion.*** **(2020-21)**

◆ *Write a program to find factorial of a number using recursive function.*

```
#include<stdio.h>
long factorial(int);
 int main()
{
  int n;
  long f;
   printf("Enter an integer to find its factorial\n");
  scanf("%d", &n);

  if (n < 0)
    printf("Factorial of negative integers isn't defined.\n");
  else
  {
    f = factorial(n);
    printf("%d! = %ld\n", n, f);
  }
   return 0;
}
```

32. *Write a program to find maximum of two number using function.*

```
#include <stdio.h>
 /* function declaration */
int max(int num1, int num2);
 int main () {
  /* local variable definition */
  int a = 100;
  int b = 200;
  int ret;
   /* calling a function to get max value */
  ret = max(a, b);
  printf( "Max value is : %d\n", ret );
   return 0;
}
 /* function returning the max between two numbers */
int max(int num1, int num2) {
  /* local variable declaration */
  int result;
   if (num1 > num2)
     result = num1;
   else
     result = num2;
   return result;
}
```


33. *Write a program to find GCD of Two Numbers using Recursion.*

```
#include <stdio.h>
int hcf(int n1, int n2);
int main()
{
  int n1, n2;
  printf("Enter two positive integers: ");
  scanf("%d %d", &n1, &n2);
  printf("G.C.D of %d and %d is %d.", n1, n2, hcf(n1,n2));
  return 0;
}
int hcf(int n1, int n2)
{
  if (n2 != 0)
    return hcf(n2, n1%n2);
  else
    return n1;
}
```

34. *Write a program to convert binary number to decimal system.*

```
#include <stdio.h>
#include <math.h>
int convertBinaryToDecimal(long long n);
int main()
{
  long long n;
  printf("Enter a binary number: ");
  scanf("%lld", &n);
  printf("%lld in binary = %d in decimal", n,
convertBinaryToDecimal(n));
  return 0;
}
int convertBinaryToDecimal(long long n)
{
  int decimalNumber = 0, i = 0, remainder;
  while (n!=0)
  {
    remainder = n%10;
    n /= 10;
    decimalNumber += remainder*pow(2,i);
    ++i;
  }
  return decimalNumber;
}
```

35. *Write a program to find sum of the given series :*
1 + 2 + 3 + 4 +.....N

```
#include<stdio.h>
int main()
{
   int i,N,sum;
   /*read value of N*/
   printf("Enter the value of N: ");
   scanf("%d",&N);
   /*set sum by 0*/
   sum=0;
   /*calculate sum of the series*/
   for(i=1;i<=N;i++)
      sum= sum+ i;
   /*print the sum*/
   printf("Sum of the series is: %d\n",sum);
   return 0;
}
```

36. *Write a program to print the following pattern :*

```
*
***
*****
*******
*********
```

```
#include <stdio.h>
void main()
{
 int row, c, n, s;
 printf("Enter the number of rows in pyramid of stars \n");
 scanf("%d", &n);
 s = n;
 for (row = 1; row <= n; row++)  // Loop to print rows
 {
  for (c = 1; c < s; c++)  // Loop to print spaces in a row
    printf(" ");
  s--;
  for (c = 1; c <= 2*row - 1; c++) // Loop to print stars in a row
    printf("*");
  printf("\n");
 }
}
```


37. *What are the types of Functions?*

Types of function

There are two types of function in C programming :

(1) Standard library functions

(2) User-defined functions

Standard Library Functions : The standard library functions are built-in functions in C programming.

These functions are defined in header files. For example,

(1) The printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the stdio.h header file.

Hence, to use the printf()function, we need to include the stdio.h header file using #include <stdio.h>.

(2) The sqrt() function calculates the square root of a number. The function is defined in the math.h header file.

User-defined Function : You can also create functions as per your need. Such functions created by the user are known as user-defined functions.

38. *How user-defined function works?*

```
#include <stdio.h>
void functionName()
{
   ... .. ...
   ... .. ...
}

int main()
{
   ... .. ...
   ... .. ...

   functionName();

   ... .. ...
   ... .. ...
```

The execution of a C program begins from the main() function.

When the compiler encounters functionName();, control of the program jumps to

```
void functionName()
```

And, the compiler starts executing the codes inside functionName().

The control of the program jumps back to the main() function once code inside the function definition is executed.

Note, function names are identifiers and should be unique.

Advantages of User-defined Function :

(1) The program will be easier to understand, maintain and debug.

(2) Reusable codes that can be used in other programs

(3) A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

39. ◆ ***What will be the value of the following function calls?*** **(2020-21)**

(1) ***divide (10,2)***

(2) ***divide (9,2)***

(3) ***divide (4.5, 1.5)***

(4) ***divide (2.0,3.0)***

(5) ***divide (4,5)***

◆ ***A function to Divide two Floating Point Numbers is as Follows :***

```
divide (float x, float y)
{
return (x/y);
}
```

(1) divide(10, 2) = 5.0

(2) divide(9 , 2) = 4.5

(3) divide(4.5 , 1.5) = 3.0

(4) divide(2.0 , 3.0) = 0.667

(5) divide(4 , 5) = 0.800

❑❑

ARRAYS/POINTERS/ STRINGS

1. ***Define array and its types.***

Array

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is a group of related data items that share a common name.

There are two types of array, which is given below :

(1) One Dimensional Array

(2) Two Dimensional Array

(3) Multi Dimensional Array

2. ◆ ***What are arrays? How can they be initialized? Give the memory representation of arrays.***

◆ ***What do you mean by an array? In what way array is different from an ordinary variable?***

Array and Its Initialization

The array is a collection of similar elements. These similar elements could be all ints, or all floats or all chars etc. Usually the array of character is called a 'string' whereas an array of ints or floats is called simply an array. All elements of any given array must be of the same type, i.e. we cannot have an array of 10 numbers, of which 5 are ints and 5 are floats.

To begin with, like other variables an array needs to be declared so that the compiler will know what kind of an array and how large an array we want. In our program we have done this with the statement :

```
int marks[30];
```

Here, int specifies the type of the variable, just as it does with ordinary variables and the word marks specify the name of the variable. The [30] however is now new. The number 30 tells how many elements of the type int will be in our array. This number is often called dimension

of the array. The bracket ([ ]) tells the compiler that we are dealing with an array.

An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.

Declaration of Array : Data_type<array name>[expression];

Example : int arr[10];

Initialization of an Array : int arr[5]={2,5,8,6,3};

Difference between Array and Ordinary Variable in C :

(1) Array is the set of a multiple values where as variable can store single value at a time.

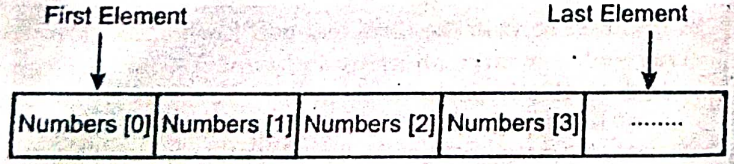
(2) The difference between the definition of array and ordinary variable is the, array is always declared, initialized, and accessed using subscript whereas ordinary variable does not have any subscript.

(3) The syntax for ordinary variable definition is data_type *v*1, *v*2, ...;

And the syntax for array variables is data type *v*1[*N*1],*v*2[*N*2],...; where *v*1,*v*2 are name of variable and *N*1, *N*2 are the integer constants indicating the maximum size of array.

Memory Representation of Arrays : Instead of declaring individual variables, such as number 0, number1, ..., and number 99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



(Figure)

3. *How to declare and initialize one dimensional array and two dimensional array.*

Declaration of One - Dimensional Arrays

Syntax : Type variable - name [sizes];

Type : data type of all elements Ex: int, float etc.,

Variable : name - is an identifier

Size : is the maximum no of elements that can be stored.

For Example : int a[50];

This array is of type int. Its name is avg. and it can contain 50 elements only. The range starting from 0 - 49 elements.

Initialization of One-Dimensional Arrays : Initialization of elements of arrays can be done in the same way as ordinary variables are done when they are declared.

Syntax : Type array_name[size] = {List of Value};

For Example : int number[3]={5,6,7};

Declaration of Two-Dimensional Arrays : To store tables we need two dimensional arrays. Each table consists of rows and columns. Two dimensional arrays are declared as :

Syntax : type array_name [row-size][col-size];

For example: int a[3][2];

Here, a is an integer array with row size is 3 and column size is 2. It looks like

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

Initialization of Two Dimensional Arrays : Initializes the elements of first row to zero and second row to one. The initialization is done by row by row. The above statement can be written as :

int a[ ][ ] = {{0,0,0},{1,1,1}};

4. ◆ ***What is multi – dimensional array?*** **(2020-21)**

◆ ***How are Two – Dimensional Array and multidimensional arrays defined? Compare with the manner in which one-dimensional arrays are defined.***

Two - Dimensional Array

Let *M* be a two-dimensional $R \times C$ array. Although *M* is represented as a rectangular array of elements with *R* rows and *C* columns, The array will be represented in the

memory by a block of $R \times C$ sequential memory locations. The M array may be stored in the memory in one of the following ways :

(1) Column by column that is in column major order.

(2) Row by row that is in row major order.

M

	R	C	Info
0	0	0	11
	1	0	12
	2	0	13
	3	0	14
1	0	1	21
	1	1	22
	2	1	23
	3	1	24

Represents column number

M

	R	C	Info
0	0	0	11
	0	1	21
	0	2	31
	0	3	41
1	1	0	12
	1	1	22
	1	2	32
	1	3	42

Represents row number

Example :

```
#include<stdio.h>
#include<conio.h>
#define ROWS 5
#define COLUMNS 5
void main()
{
 int row,column,product[ROWS][COLUMNS];
 clrscr();
 int i,j;
 printf("MULTIPLICATION TABLE\n\n");
 printf("  ");
 for(j=1;j<=COLUMNS;j++)
 printf("%4d",j);
 printf("\n");
 printf("---------------------\n");
 for(i=0;i<ROWS;i++)
 {
 row=i+1;
 printf("%2d |",row);
 for(j=1;j<=COLUMNS;j++)
 {  column=j;
 product[i][j]=row*column;
```



```
 printf("%4d",product[i][j]);  }
 printf("\n");
 }
 }
```

Output :

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

Multidimensional Arrays

C allows arrays of three or more dimensions. The limit is determined by the compiler. The general form of a multidimensional arrays is :

type array_name [s][s2]][s3]...[sm];

Wheres is the size of the dimension. some example are :

Int survey[3][5][12];
float table[5][4][5][3];

Survey is a three-dimensional array declared to contain 180 integer type elements. Similarly table is a four-dimensional array containing 300 elements of floating point type.

```
/*************************************************/
/*PROGRAM TO TABULATE SURVEY DATA*/
/*************************************************/
#include<stdio.h>
#include<conio.h>
void main( )
{
int  i, j, car;
clrscr();
static int frequency[5][5] = { {0}, {0}, {0}, {0}, {0} };
char city;
printf("For each person, enter the city code \n");
printf("followed by the car code. \ n");
printf("Enter the letter x to indicate end. \n");
/*------------TABULATION BEGINS----------------------*/
for(i = 1 ; i < 100 ; i++)
{
      scanf("%c", &city);
      if(city == 'X')
      break;
      scanf("%d", &car);
      switch(city)
{
```

```
case 'B':frequency[1][car]++;
break;
case 'C':frequency[2][car]++;
break;
case 'D':frequency[3][car]++;
break;
case 'M':frequency[4][car]++;
break;
}
}
/*TABULATION COMPLETED AND PRINTING BEGINS*/
  printf("\n\n");
  printf("POPULARITY TABLE \n\n");
  printf("-------------------------------------------\n");
  printf("City   Ambassador  Fiat          Dolphin  Maruti\n");
  printf("-------------------------------------------\n");
for(i = 1; i <=4; i++)
{
      switch(i)
      {
      case 1 :printf("Bombay");
      break;
      case 2 :printf("Calcutta");
      break;
      case 3 :printf("Delhi");
      break;
      case 4 :printf("Madras");
      break;
      }
      for(j=1 ; j<= 4; j++)
      printf("%6d", frequency[i][j]);
      printf("\n") ;
      }
      printf("-------------------------------------------\n");
/*-------------PRINTING ENDS-------------------------*/
}
```

Output :

```
For each person, enter the city code
followed by the car code.  nEnter the letter x to indicate end.
M 1 C 2 B 1 D 1 M 2 B 4
C 1 D 3 M 4 B 4 D 1 C 3
X

                    POPULARITY TABLE

City      Ambassador      Fiat    Dolphin      Maruti
-------------------------------------------
Bombay          1         0       0       2
Calcutta        1         1       1       0
Delhi           2         0       1       0
Madras          1         1       0       1
```


Program to Add Two 4 × 4 Matrices using 2 D Array and Function

```
#include<stdio.h>
#include<conio.h>
void read_arr(int a[10][10],int row,int col)
{   int i,j;
   for(i=1;i<=row;i++)
   {   for(j=1;j<=col;j++)
   {      printf("Enter Element %d %d : ",i,j);
      scanf("%d",&a[i][j]);}}}
 void  add_arr(int  m1[10][10],int  m2[10][10],int  m3[10][10],int
row, int col)
{   int i,j;
   for(i=1;i<=row;i++)
   {   for(j=1;j<=col;j++)
   {   m3[i][j] =  (m1[i][j] + m2[i][j]);  }}}
void print_arr(int m[10][10],int row,int col)
{  int i,j;
   for(i=1;i<=row;i++)
      {      for(j=1;j<=col;j++)
      {      printf("%d ",m[i][j]);   }
      printf("\n");   }}
main()
{   int m1[10][10],m2[10][10],m3[10][10],row,col;
   clrscr();
   printf("Enter number of rows :");
   scanf("%d",&row);
   printf("Enter number of colomns :");
   scanf("%d",&col);
   read_arr(m1,row,col);
   read_arr(m2,row,col);
   add_arr(m1,m2,m3,row,col);
   print_arr(m3,row,col);
   getch();}
```

Multidimensional Array Passed To A Function

```
#include <stdio.h>
void displayNumbers(int num[2][2]);
int main() {
   int num[2][2], i, j;
   printf("Enter 4 numbers:\n");
   for (i = 0; i < 2; ++i)
      for (j = 0; j < 2; ++j)
         scanf("%d", &num[i][j]);
   // passing multi-dimensional array to a function
   displayNumbers(num);
   return 0;
}
void displayNumbers(int num[2][2]) {
```

```
    int i, j;
    printf("Displaying:\n");
    for (i = 0; i < 2; ++i)
       for (j = 0; j < 2; ++j)
          printf("%d\n", num[i][j]);
}
#include <stdio.h>
void display(int age) {
   printf("%d", age);
}
int main() {
   int ageArray[ ] = {2, 3, 4};
   display(ageArray[2]); //Passing array element ageArray[2]
   return 0;
}
```

Differences Multidimensional Array With One-Dimensional Arrays

	1D Array	Multi D Array
(1)	These are also called single subscripted variable.	These are called multi-subscripted variable
(2)	Syntax : <type of><identifier>[<index>] Example : int mat[10];	Syntax : <type of><identifier>[index][index] Example : int mat[3][3][3]

5. ***State the rules that determine the order in which initial values are assigned to multi-dimensional array elements.***

With normal variables, we could declare on one line then initialize on the next :

```
int x;
x = 0;
```

Or, we could simply initialize the variable in the declaration statement itself :

```
int x = 0;
```

Can we do the same for arrays? Yes, for the built-in types. Simply list the array values (literals) in set notation { } after the declaration. Here are some examples :

```
int list[4] = {2, 4, 6, 8};
char letters[5] = {'a', 'e', 'i', 'o', 'u'};
double numbers[3] = {3.45, 2.39, 9.1};
int table[3][2] = {{2, 5} , {3,1} , {4,9}};
```


C-Style Strings : Arrays of type char are special cases.

(1) We use strings frequently, but there is no built-in string type in the language

(2) A C-style string is implemented as an array of type char that ends with a special character, called the "null character".

(a) The null character has ASCII value 0.

(b) The null character can be written as a literal in code this way :'\0'

(3) Every string literal (something in double-quotes) implicitly contains the null character at the end

Since character arrays are used to store C-style strings, you can initialize a character array with a string literal (i.e. a string in double quotes), as long as you leave room for the null character in the allocated space.

```
char name[7] = "Johnny";
```

Notice that this would be equivalent to :

```
char name[7] = {'J', 'o', 'h', 'n', 'n', 'y', '\0'};
```

6. ◆ ***Write a program in C for Multiplication of two matrices. The result must be stored in a third matrix.*** **(2020-21)**

◆ ***Write a program to multiply any two 2 × 2 matrices.***

◆ ***Write a C program to multiplication of two arrays.***

```
#include<stdio.h>
#include<conio.h>
void main()
{int a[10][10],b[10][10],c[10][10],i,j,k;
int sum=0;
clrscr();
printf("nEnter First Matrix : n");
for(i=0;i<3;i++)
{for(j=0;j<3;j++)
scanf("%d",&a[i][j]);}
printf("nEnter Second Matrix:n");
for(i=0;i<3;i++)
{for(j=0;j<3;j++)
scanf("%d",&b[i][j]);}
printf("The First Matrix Is:n");
for(i=0;i<3;i++)//print the first matrix
{for(j=0;j<3;j++)
```

```
printf(" %d ",a[i][j]);
printf("n");
}
printf("The Second Matrix Is:n");
for(i=0;i<3;i++) // print the second matrix
{for(j=0;j<3;j++)
printf(" %d ",b[i][j]);
printf("n");}
for(i=0 i<=2;i++)
for(j=0 j<=2;j++)
{sum = 0;
for(k=0;k<=2;k++)
sum = sum + a[i][k] * b[k][j];
c[i][j]=sum;}
printf("nMultiplication Of Two Matrices :nn");
for(i=0;i<3;i++) .
{for(j=0;j<3;j++)
printf(" %d ",c[i][j]);
printf("n");}
getch();
}
```

7. ◆ ***What are Pointers? Why are they required? How do you declare and initialize them? Write a program to read two integers x & y and swap the contents of two variables x & y using pointers.*** **(2020-21)**

◆ ***What is meant by Pointer? Write down the use of pointer. Explain the following :***

(1) ***Pointer and function***

(2) ***Pointer and array***

(3) ***Pointer and string***

Pointer

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

```
type *var-name;
```

Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable. The asterisk * you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a

variable as a pointer. Following are the valid pointer declaration :

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

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

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

(1) We define a pointer variable

(2) Assign the address of a variable to a pointer and

(3) Finally access the value at the address available in the pointer variable.

This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. Following example makes use of these operations:

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

(1) Pointer and Function : Just like pointer to characters, integers etc, we can have pointers to functions. A function pointer can be declared as :

<return type of function> (*<name of pointer>) (type of function arguments)

For Example :

```
int (*fptr)(int, int)
```

The above line declares a function pointer 'fptr' that can point to a function whose return type is 'int' and takes two integers as arguments. Take anexample :

```
#include<stdio.h>
int func (int a, int b)
{printf("\n a = %d\n",a);
printf("\n b = %d\n",b);
return 0;}
int main(void)
{int(*fptr)(int,int); // Function pointer
fptr = func; // Assign address to function pointer
func(2,3);
fptr(2,3);
return 0;}
```

(2) Pointer and Array : An array name is a constant pointer to the first element of the array. Therefore, in the declaration :

```
double balance[50];
```

Balance is a pointer to & balance[0], which is the address of the first element of the array balance. Thus, the following program fragment assigns p the address of the first element of balance:

```
double *p;
double balance[10];
p = balance;
```

It is legal to use array names as constant pointers, and vice versa. Therefore, *(balance + 4) is a legitimate way of accessing the data at balance [4].

Once you store the address of first element in p, you can access array elements using *p, *(p+1), *(p+2) and so on. Below is the example to show all the concepts of pointer to array:

```
#include <stdio.h>
int main ()
{
/* an array with 5 elements */
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
double *p;
int i;
p = balance;
/* output each array element's value */
printf( "Array values using pointer\n");
for ( i = 0; i < 5; i++ )
{
printf("*(p + %d) : %f\n", i, *(p + i) );
}
printf( "Array values using balance as address\n");
for ( i = 0; i < 5; i++ )
{
```



```
printf("*(balance + %d) : %f\n", i, *(balance + i) );
}
return 0;
}
```

(3) Pointer and Strings : In C language Strings are defined as an array of characters or a pointer to a portion of memory containing ASCII characters. A string in C is a sequence of zero or more characters followed by a NULL '\0' character.

It is important to preserve the NULL terminating character as it is how C defines and manages variable length strings. All the C standard library functions require this for successful operation. All the string handling functions are prototyped in: string.h or stdio.h standard header file. So while using any string related function, don't forget to include either stdio.h or string.h. May be your compiler differs so please check before going ahead. If you were to have an array of characters WITHOUT the null character as the last element, you'd have an ordinary character array, rather than a string constant. String constants have double quote marks around them, and can be assigned to char pointers as shown below. Alternatively, you can assign a string constant to a char array - either with no size specified, or you can specify a size, but don't forget to leave a space for the null character!

```
#include <stdio.h>
int main() {
char array1[50];
char *array2;
printf("Now enter another string less than 50");
printf(" characters with spaces: \n");
gets(array1);
printf("\nYou entered: ");
puts(array1);
printf("\nTry entering a string less than 50");
printf(" characters, with spaces: \n");
scanf("%s", array2);
printf("\nYou entered: %s\n", array2);
return 0;
}
```

C Program to Read Two Integers *x* and *y* and Swap the Contents of the two Variables *x* and *y* using Pointers :

```
#include <stdio.h>
int main()
{
int x, y, *a, *b, temp;
printf("Enter the value of x and y\n");
scanf("%d%d", &x, &y);
printf("Before Swapping\nx = %d\ny = %d\n", x, y);
   a = &x;
   b = &y;
temp = *b;
  *b = *a;
  *a = temp;
printf("After Swapping\nx = %d\ny = %d\n", x, y);
return 0;
}
```

8. *Write a program in 'C' find out the average of 4 integers entered by the user.*

```
#include <stdio.h>
void main()
{
   int avg = 0;
   int sum =0;
   int x=0;
   int num[4];
    // storing the entered values in the array
   for (x=0; x<4;x++)
   {
      printf("Enter number %d \n", (x+1));
      scanf("%d", &num[x]);
   }
// Perform sum of the array elements
   for (x=0; x<4;x++)
   {
      sum = sum+num[x];
   }
   avg = sum/4;
   printf("Average of entered number is: %d", avg);
}
```

9. *Write a program in 'C' to find out addition of two matrices enter by the user.*

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



```
void main( )
{   int A[10][10], B[10][10], C[10][10] ;
  int m, n ;
  int p, q ;
  int i ,j ;
  printf ("Enter row size and col size of first matrix A (less then 10)
\n");
  scanf ("%d%d", &m, &n);
  printf ("Enter row size and col size of first matrix B (less then 10)
\n");
  scanf ("%d%d", &p, &q);
  // condition for checking whether matrices can be added
  if ( m==p && n==q )
     {  printf("matrices can be added\n");   }
  else
     {  printf("matrices can not be added\n") ;
        exit(0);
     }
  // Reading first matrix
  printf("enter elements of matrix A\n");
  for( i=0; i<m; i++)
   {  for( j=0; j<n; j++)
       scanf("%d", &A[i][j]) ;
   }
 // Reading second matrix
 printf("enter elements of matrix B\n");
 for( i=0; i<p; i++)
   {  for( j=0; j<q; j++)
      scanf ("%d", &B[i][j]);
   }
 //Printing matrices A and B in a proper format
  printf("\n MATRIX A \n");
  for ( i=0; i<m; i++)
   {  printf ("\n");
      for(j=0;j<n;j++)
      printf ("%d", A[i][j]);
   }
  printf("\n MATRIX B \n");
  for( i=0; i<p; i++)
    {  printf( "\n");
       for( j=0; j<q; j++)
       printf(" %d", B[i][j]);
    }
  // loop to add both matrices
  for( i=0; i<m; i++)
    {   for( j=0; j<n; j++)
          C[i][j] =A[i][j] + B[i][j];
    }
  //printing sum Matrix C
```

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

10. *Write a program to reads 10 integers and print them in descending order using bubble sort technique.*

```
# include <stdio.h>
# include <conio.h>
void main ( )
{ int arr[50], size, i ,temp , j;
  clrscr();
  printf("enter size of the array\n");
  scanf("%d", &size);
  printf("Enter elements of the array\n");
  for( i=0; i<size; i++)
  {  scanf("%d", &arr[i]);  }
  // sorting function
  for(i=0; i<size; i++)
  {
    for ( j=0 ; j<(size-1) – i ; j++)
    {   if ( arr[j] < arr[j+1])
                {  temp=arr[j];
                   arr[j]=arr[j+1];
                   arr[j+1]=temp;
                }
    }
  }
  // printing sorted array
  printf("array after sorting\n");
  for( i=0; i<size; i++)
    {   printf("\n %d", arr[i]);   }
  getch();
}
```

11. *Write a program to print the transpose of a square matrix*

```
# include<stdio.h>
# include<conio.h>
void main ( )
{   int A[3][3], B[3][3], i, j;
```



```
clrscr( );
//reading elements of the matrix
printf("Enter elements of square matrix\n");
for( i=0; i<3; i++)
 {  for( j=0; j<3; j++)
      scanf("%d", &A[i][j]);
 }
// printing original square matrix
for( i=0; i<3; i++)
{   printf("\n");
    for( j=0; j<3; j++)
    printf("%d  ", A[i][j]) ;
}
// Finding transpose of a matrix
for ( i=1; i<3; i++)
 {  for( j=0; j< 3; j++)
    B[ j][i] = A[i] [j] ;
 }
//printing resultant matrix B - transpose of a matrix
printf ("\n transpose of a matrix \n");
for ( i=0; i<3; i++)
{   printf("\n");
    for( j=0; j<3; j++)
    printf("%d  ", B[i][j]);
}
getch();
}
```

12. *Write a program to find sum of diagonal elements of a two dimensional array.*

```
# include<stdio.h>
# include<conio.h>
void main ( )
{
 int A[4][4],  i, j, sum=0;
 clrscr();
 //reading elements of the matrix
 printf("Enter elements of square matrix\n");
 for( i=0; i<4; i++)
  {  for( j=0; j<4; j++)
       scanf("%d", & A[i][j]);
  }
 // printing original square matrix
 for ( i=0; i<4; i++)
 {   printf("\n");
     for( j=0; j<4; j++)
     printf("%d  ", A[i][j]) ;
```

```
    }
    // Finding diagonal elements
    for ( i=1; i<3; i++)
    {   for( j=0; j< 3; j++)
          { if ( i = = j)
              sum= sum + A[i][j];
          }
    }
    printf( "sum of diagonal elements =%d ", sum );
    getch( );
}
```

13. *Write a program to reads n numbers and print them in increasing order using selection sort technique.*

```
#include <stdio.h>
 int main()
{
  int array[100], n, c, d, position, swap;
  printf("Enter number of elements\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  for (c = 0; c < (n - 1); c++)
  {
    position = c;
    for (d = c + 1; d < n; d++)
    {
      if (array[position] > array[d])
        position = d;
    }
    if (position != c)
    {
      swap = array[c];
      array[c] = array[position];
      array[position] = swap;
    }
  }
  printf("Sorted list in ascending order:\n");
  for (c = 0; c < n; c++)
    printf("%d\n", array[c]);
  return 0;
}
```


14. *Write a program to store marks of n students and display marks with their details.*

```
#include <stdio.h>
struct student
{
   char name[50];
   int roll;
   float marks;
} s;
int main()
{
   printf("Enter information:\n");
   printf("Enter name: ");
   scanf("%s", s.name);
   printf("Enter roll number: ");
   scanf("%d", &s.roll);
   printf("Enter marks: ");
   scanf("%f", &s.marks);
   printf("Displaying Information:\n");
   printf("Name: ");
   puts(s.name);
   printf("Roll number: %d\n",s.roll);
   printf("Marks: %.1f\n", s.marks);
   return 0;
}
```

15. *Write a program to count the number of vowels, consonants in a word.*

```
#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;
}
```

16. *Write a program to sort elements in lexicographical order (dictionary order)*

```
#include<stdio.h>
#include <string.h>
int main()
{
    int i, j;
    char str[10][50], temp[50];
    printf("Enter 10 words:\n");
    for(i=0; i<10; ++i)
        scanf("%s[^\n]",str[i]);
    for(i=0; i<9; ++i)
        for(j=i+1; j<10 ; ++j)
        {
            if(strcmp(str[i], str[j])>0)
            {
                strcpy(temp, str[i]);
                strcpy(str[i], str[j]);
                strcpy(str[j], temp);
            }
        }
    printf("\nIn lexicographical order: \n");
    for(i=0; i<10; ++i)
    {
        puts(str[i]);
    }
    return 0;
}
```


17. *Define pointers and its advantages.*

Pointers

A pointer is a variable that can store an address of a variable (i.e., 112300).We say that a pointer points to a variable that is stored at that address. A pointer itself usually occupies 4 bytes of memory.

Advantages of Pointers :

(1) A pointer enables us to access a variable that is defined out side the function.

(2) Pointers are more efficient in handling the data tables.

(3) Pointers reduce the length and complexity of a program. They increase the execution speed.

18. *How to declare pointer? Explain with the help of examples.*

Declaration of Pointers

```
Data_type * Variable-name;
Eg:-  int *ad;        /* pointer to int */
char *s;      /* pointer to char */
float *fp;    /* pointer to float */
char **s;     /* pointer to a variable that is a pointer to char */
```

Consider the Statement : p=&i;

Here & is called address of a variable. p contains the address of a variable i.

19. *What are the characteristics of C pointers?*

Characteristics of C Pointers

(1) Pointers are special variables that store the memory address, instead of value like in usual variables.

(2) Pointers always hold addresses as a whole number.

(3) Pointers in C are always initialized to NULL. For example, int *ptr=null;

(4) Null pointer symbol is "0".

(5) The asterisk symbol, * is used to retrieve the value of the variable; & ampersand symbol is used for retrieving the address of a variable.

(6) If a C pointer is assigned to the null value, it points nothing.

(7) Only subtraction between pointers are allowed. Addition, division, and multiplication operations are not allowed.

(8) The memory size of a C pointer for a 16-bit system is 2 bytes.

20. *How Pointers Work in C Programming Language?*

We must understand the use of two operators (& and *) for using pointers in C.

Unary Ampersand (&) Operator : The unary operator, & is used for getting the address of the variable. If you use '&' before a variable name, it returns the address of that variable. For example, &y will return the address of the variable y.

/*Example C Program*/

```
(1) #include<stdio.h>
(2) #include<conio.h>
(3) int main()
(4) {
(5) int y;
(6) printf("%p", &y); /*prints the address of y */
(7) return 0;
(8) }
```

Unary Asterisk Operator (*) : There are two ways to use the * operator in C language.

(1) **Pointer Declaration :** When a pointer is declared, there must be an asterisk operator placed before the pointer name.

```
/*Example C Program*/
#include<stdio.h>
void main()
{
  int y=5;
      int *pt; /* when * is used, pt is considered as a pointer variable */
  pt = &y; /* pt holds the address of the variable y */
}
```

(2) **Accessing Pointer Variables :** We can also use the * operator for accessing the value of stored in any address/memory-location.

```
/*Example C Program*/
#include<stdio.h>
void main()
{
```



```
int x = 6;
int *pt;
pt = &x;
printf("The value of x = %d \n:", *pt); /*prints 6 as x value*/
    printf("The address of x = %p\n", pt); /*returns the address of variable x */
*pt= 10;
    printf("After changing the value of pt, the new value is %d", *pt); /*prints 10 as pt value */
}
```

Pointer Applications in C Programming

Pointer is used for different purposes. Pointer is low level construct in programming which is used to perform high level task. Some of the pointer applications are listed below :

(1) **Passing Parameter by Reference :** First pointer application is to pass the variables to function using pass by reference scheme.

```
void interchange(int *num1,int *num2)
{
  int temp;
  temp  = *num1;
  *num1 = *num2;
  *num2 = *num1;
}
```

Pointer can be used to simulate passing parameter by reference. Pointer is used to pass parameter to function. In this scheme we are able to modify value at direct memory location.

(2) **Accessing Array Element :**

```
int main()
{
int a[5] = {1,2,3,4,5};
int *ptr;
ptr = a;

for(i=0;i<5;i++) {
 printf("%d",*(ptr+i));
}

return(0);
}
```

We can access array using pointer. We can store base address of array in pointer.

```
    ptr = a;
```

Now we can access each and individual location using pointer.

```
for(i=0;i<5;i++) {
    printf("%d",*(ptr+i));
}
```

(3) Dynamic Memory Allocation : Another pointer application is to allocate memory dynamically.

We can use pointer to allocate memory dynamically. Malloc and calloc function is used to allocate memory dynamically.

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

int main()
{
  char *str;

  str = (char *) malloc(15);
  strcpy(str, "mahesh");

  printf("String = %s,  Address = %u\n", str, str);
  free(str);

  return(0);
}
```

consider above example where we have used malloc() function to allocate memory dynamically.

(4) Reducing Size of Parameter :

```
struct student {
  char name[10];
  int rollno;
};
```

Suppose we want to pass the above structure to the function then we can pass structure to the function using pointer in order to save memory.

Suppose we pass actual structure then we need to allocate (10 + 4 = 14 Bytes(*)) of memory. If we pass pointer then we will require 4 bytes(*) of memory.

(5) Some other Pointer Applications :

(a) Passing Strings to function

(b) Provides effective way of implementing the different data structures such as tree,graph,linked list

21. *Write a note on Pointer Arithmetic.*

Pointer Arithmetic

It is common to do basic addition or subtraction of pointer values. In C we can just add or subtract values from a pointer to cause the pointer to point to other data. Pointer arithmetic is most often done with arrays. When pointer arithmetic is performed, a sizeof operation is implicit in terms of the resultant memory address. If a is a pointer to an integer, then (a+i) can be thought to mean (a + sizeof(int)*i).

```
#include <stdio.h>

int main(void)
{
  int i=2,j=3,k=4;
  int a[5];
  int *p, *q;

  p = &k;
  printf( "i = %d\n", *(p+2));          /* i = 2 */

  q = a + 2;
  k = (int)a + sizeof(int)*2;
  printf( "q - k = %d\n", (int)q-k );   /* q - k = 0 */
  return 0;
}
```

22. *Differentiate between referencing and de-referencing operator.*

Referencing Operator : The operator & returns the memory address of variable on which it is operated, this is called Referencing.

De-Referencing Operator : The * operator is called an indirection operator or dereferencing operator which is used to display the contents of the Pointer Variable.

Consider the following Statements :

```
int *p,x;
x =5;
p= &x;
```

Assume that x is stored at the memory address 2000. Then the output for the following printf statements is :

```
printf("The Value of x is %d",x);
```

```
printf("The Address of x is %u",&x);
printf("The Address of x is %u",p); Printf("The Value of x is %d",*p);
printf("The Value of x is %d",*(&x));
```

Output :

```
The Value of x is 5
The Address of x is 2000
The Address of x is 2000
The Value of x is 5
The Value of x is 5
```

23. ***Write a program in 'C' to read an array and print that array using pointer only.***

```
#include<stdio.h>
main ()
{
int *a,i;
printf("Enter five elements:");
for (i=0;i<5;i++)
scanf("%d",a+i);
printf("The array elements are:");
for (i=0;i<5;i++)
printf("%d", *(a+i));
}
```

Note : In one dimensional array, a[i] element is referred by (a+i) is the address of ith eth lement.
* (a+i) is the value at the i element.

In two-dimensional array, a[i][j] element is represented as
((a+i)+j)

24. ***Write a program to change the value of constant integer using pointers***

```
#include <stdio.h>
int main()
{
   const int a=10;     //declare and assign constant integer
   int *p;             //declare integer pointer
   p=&a;               //assign address into pointer p
   printf("Before changing - value of a: %d",a);//assign value
   *p=20;
   printf("\nAfter  changing - value of a: %d",a);
  printf("\nWauuuu... value has changed.");
   return 0;
}
```


25. ***Write a program to explore value of NULL pointer.***

```
#include <stdio.h>
int main(void) {
        int num = 10;
        int *ptr1 = &num;
        int *ptr2;
        int *ptr3=0;
        if(ptr1 == 0)
                printf("ptr1: NULL\n");
        else
                printf("ptr1: NOT NULL\n");
        if(ptr2 == 0)
                printf("ptr2: NULL\n");
        else
                printf("ptr2: NOT NULL\n");
        if(ptr3 == 0)
                printf("ptr3: NULL\n");
        else
                printf("ptr3: NOT NULL\n");
        return 0;
}
```

26. ◆ ***What is string? Write a C program that reads a sentence and prints the frequency of each of the vowels and total count of consonants.*** **(2020-21)**

◆ ***Define string. How to declare and initialize string?***

Strings (Character Arrays) : A String is an array of characters. Any group of characters (except double quote sign) defined between double quotes is a constant string.

Ex : "C is a great programming language".

Note : If we want to include double quotes.

Ex : "\"C is great \" is norm of programmers ".

Declaring and Initializing Strings : A string variable is any valid C variable name and is always declared as an array.

Syntax : char string name [size];

Size determines the number of characters in the string name. When the compiler assigns a character string to a character array, it automatically supplies a null character ('\0') at end of String. Therefore, size should be equal to the maximum number of characters in String plus one.

String can be initialized when declared as
char greeting[ ] = "Hello";
char city[10]= ,'N','E','W','','Y','O','R','K','/0';

C Program that Reads a Sentence and Prints the Frequency of the Each Vowels and Total Count of Consonants :

```
#include <string.h>

int main()
{
char s[1000];
inti,vowels=0,consonants=0;

printf("Enter  the string : ");
gets(s);

for(i=0;s[i];i++)
   {
/* To check ASCII values of capital letters A-Z range 65-90,
small letters a-z range 97-122 following condition is written. */

        if((s[i]>=65 && s[i]<=90)|| (s[i]>=97 && s[i]<=122))
        {
              // To check vowels
          if(s[i]=='a'||
s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u'||s[i]=='A'||s[i]=='E'||s[i]=='I'||s[i]=='O'
||s[i]=='U')
        vowels++;
    else
    consonants++;
        }

        }


    printf("vowels = %d\n",vowels);
    printf("consonants = %d\n",consonants);

    return 0;
    }
```

Output :

```
Enter  the string : hello world
vowels = 3
consonants = 7
```


27. *What are the different types of string functions in 'C'?*

String Functions :

strcat( )	Concatenates two Strings
strcmp( )	Compares two Strings
strcpy( )	Copies one String Over another
strlen( )	Finds length of String

strcat( ) Function : This function adds two strings together.

Syntax : char *strcat(const char *string1, char *string2);
strcat(string1,string2);
string1 = VERY
string2 = FOOLISH
strcat(string1,string2);
string1=VERY FOOLISH
string2 = FOOLISH

Strncat( ) : Append n characters from string2 to stringl.
char *strncat(const char *string1, char *string2, size_t n);

strcmp() Function : This function compares two strings identified by arguments and has a value 0 if they are equal. If they are not, it has the numeric difference between the first non-matching characters in the Strings.

Syntax : int strcmp (const char *string1,const char *string2);
strcmp(string1,string2);

Ex : strcmp(name1,name2);
strcmp(name1,"John"); strcmp("ROM","Ram");

Strncmp( ) : Compare first n characters of two strings.
int strncmp(const char *string1, char *string2, size_t n);

strcpy( ) Function : It works almost as a string assignment operators. It takes the form .

Syntax : char *strcpy(const char *string1,const char *string2);
strcpy(string1,string2); string2 can be array or a constant.

Strncpy( ) : Copy first n characters of string2 to stringl . char *strncpy(const char *string1,const char *string2, size_t n);

strlen() Function : Counts and returns the number of characters in a string.

Syntax : int strlen(const char *string);
n= strlen(string);
n is an integer variable which receives the value of length of string.

/* Illustration of String-Handling */

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

```
char s1[20],s2[20],s3[20];
int X,L1,L2,L3;
printf("Enter two string constants\n");
scanf("%s %s",s1,s2);
X=strcmp(s1,s2);
if (X!=0)
{
printf("Strings are not equal\n");
strcat(s1,s2);
}
else
printf("Strings are equal \n");
strcpy(s3,s1);
L1=strlen(s1);
L2=strlen(s2);
L3=strlen(s3);
printf("s1=%s\t length=%d chars \n",s1,L1);
printf("s2=%s\t length=%d chars \n",s2,L2);
printf("s3=%s\t length=%d chars \n",s3,L3);
}
```

28. *Discriminate puts () and gets().* **(2020-21)**

Difference between puts () and gets()

gets()	puts()
It is a C library function that reads a line from stdio header file and stores it in the pointed string.	It is a C library function that writes a string to stdout or standard output.
Declaration is: char *gets (char *str)	Declaration is: int puts (const char *str)
Helps to scan a line of text fro1n a standard input device	Helps to display a string on a standard output device
Returns string on success; however, it will returnNULL or EOF if there are no characters to read	Returns a non-negative value if successful; if unsuccessful, it will return EOF

❑❑

STRUCTURE/UNION/ STORAGE CLASSES

1. *Explain structure with its declaration and initialization.*

Structures

A Structure is a collection of elements of dissimilar data types. Structures provide the ability to create user defined data types and also to represent real world data.

Suppose if we want to store the information about a book, we need to store its name (String), its price (float) and number of pages in it(int). We have to store the above three items as a group then we can use a structure variable which collectively store the information as a book.

Structures can be used to store the real world data like employee, student, person etc.

Declaration : The declaration of the Structure starts with a Key Word called Struct and ends with ; . The Structure elements can be any built in types.

Syntax :

```
struct <Structure name>
{
Structure element 1; Structure element 2;
---
Structure element n;
};
```

Then the Structure Variables are declared as :

```
struct < Structure name > <Var1,Var2>;
```

For Example :

```
struct emp
{
int empno.
char ename[20];
float sal;
};
struct emp e1,e2,e3;
```

Initialization : Structure Variables can also be initialised where they are declared.

```
struct emp
```

```
{
int empno;
char ename[20];
float sal;
};
struct emp e1 = { 123,"Kumar",5000.00};
```

To access the Structure elements we use the .(dot) operator.
To refer empno we should use e1.empno
To refer sal we whould use e1.sal

Structure elements are stored in contiguous memory locations as shown below. The above Structure occupies totally 26 bytes.

e1.empno	e1.ename	e1.sal
123	Kumar	5000.00
2000	2002	2022

2. *Write a program in 'C' to explore structure.*

```
#include <stdio.h>
#include <string.h>
void main( )
{
struct emp
{
int empno;
char ename[20];
float sal;
};
struct emp e;
printf (" Enter Employee number: \n");
scanf("%d",&e.empno);
printf (" Enter Employee Name: \n");
scanf("%s",&e.empname);
printf (" Enter the Salary: \n");
scanf("%f",&e.sal);
printf (" Employee No = %d", e.empno);
printf ("\n Emp Name = %s", e.empname);
printf ("\n Salary = %f", e.sal);
}
```

3. *Declare a structure of a student. Declare an array of structure for a class strength of 60 students.*

```
#include <stdio.h>
struct student
{
```



```
char name[50];
int roll;
float marks;
} s;
int main()
{
printf("Enter information:\n");
printf("Enter name: ");
scanf("%s", s.name);
printf("Enter roll number: ");
scanf("%d", &s.roll);
printf("Enter marks: ");
scanf("%f", &s.marks);
printf("Displaying Information:\n");
printf("Name: ");
puts(s.name);
printf("Roll number: %d\n",s.roll);
printf("Marks: %.1f\n", s.marks);
return 0;
}
```

Array of Structure for a Class Strength of 60 Students

```
include <stdio.h>
struct student
{
    char name[50];
    int roll;
    float marks;
} s[10];
int main()
{
    int i;
    printf("Enter information of students:\n");
    // storing information
    for(i=0; i<10; ++i)
{
    s[i].roll = i+1;
    printf("\nFor roll number%d,\n",s[i].roll);
    printf("Enter name: ");
    scanf("%s",s[i].name);
    printf("Enter marks: ");
    scanf("%f",&s[i].marks);
    printf("\n");
}
    printf("Displaying Information:\n\n");
    // displaying information
    for(i=0; i<10; ++i)
{
```

```
        printf("\nRoll number: %d\n",i+1);
        printf("Name: ");
        puts(s[i].name);
        printf("Marks: %.1f",s[i].marks);
        printf("\n");
        }
      return 0;
}
```

4. ◆ ***Differentiate between the Structure & Union using suitable example.*** **(2020-21)**
   ◆ ***What is Union? How is it different from structures? Explain in detail the memory representation of structure and union. Use examples to justify your answers.***
   ◆ ***How a union is different from a structure? Declare a structure called date with the following members :***
   (1) ***Date***
   (2) ***Month***
   (3) ***Year***

Union

A union is a memory location that is shared by two or more different type of variables generally these variables can be used for different data types at different times.

The syntax is :

```
union< Union Name>
{Datatype   field1;
Datatype   field2;
Datatype   field3;
...   ...
...   ...
...   ...};
```

Difference between Union and Structure

Union	Structure
Data type in which allocation of the memory (largest data type) taken place and each and every variable shared this memory allocation.	In this data type, each component of the structure allocates memory differently and in used by the user.
It is based upon the memory saving technique called sharing.	Not any memory saving technique.
The syntax begins with union keyword.	The syntax involves struct keyword.

Memory Representation of Union

A union is a special data type available in C that enables you to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multi-purpose.

To define a union, you must use the union statement in very similar was as you did while defining structure. The union statement defines a new data type, with more than one member for your program. The format of the union statement is as follows :

```
union [union tag]
{
  member definition;
  member definition;
     ...
  member definition;
} [one or more union variables];
```

The union tag is optional and each member definition is a normal variable definition, such as int *i*; or float *f*; or any other valid variable definition. At the end of the union's definition, before the final semicolon, you can specify one or more union variables but it is optional. Here is the way you would define a union type named Data which has the three members *i*, *f*, and str :

```
union Data
{
  int i;
  float f;
  char  str[20];
} data;
```

Now, a variable of Data type can store an integer, a floating-point number, or a string of characters. This means that a single variable i.e. same memory location can be used to store multiple types of data. You can use any built-in or user defined data types inside a union based on your requirement.

The memory occupied by a union will be large enough to hold the largest member of the union. For example, in above example Data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by character string.

Memory Representation of Structure : There are two ways to initialize them : all the fields in one statement or one field at a time.

To initialize all fields in one statement, we can declare variable as usual, but assign it a list of variables enclosed in brackets, like this :

```
struct human TR = {"Tony", 12, 1.5};
```

This is similar to the method of initializing arrays.

Now the struct human variable, TR has all three fields initialized : the name field holds the string, "Tony", the age field holds the integer, 12, and the height is 1.5.

To reference an individual field, we can use the dot operator to separate the struct variable name and the field name.

You can use the dot operator to initialize each field individually :

```
struct human TR;
TR.name = "Tony";
TR.age = 12;
TR.height = 1.5;
```

To access each field, use the dot operator again.

For example, TR.age will return 12

Declare a Structure Called Date :

```
#include <stdio.h>
int main (void)
{
struct date
{
int month;
  int day;
  int year;
};
struct date today;
today.month = 9;
today.day = 25;
today.year = 2015;
printf ("Today's date is %i/%i/%.2i\n", today.month, today.day,
today.year %100);
}
return 0;
}
```

Output :

```
Today's date is 9/25/15.
```


5. ***WAP in C to enter full name (with Fname, Name) and date of birth (day, month, year) and display the same. Use the nested structure.***

```
#include <stdio.h>
struct student{
        char name[30];
        int rollNo;
        struct dateofBirth{
                int dd;
                int mm;
                int yy;
        }DOB; /*created structure variable DOB*/
};
int main()
{
        struct student std;
        printf("Enter name: "); gets(std.name);
        printf("Enter roll number: ");scanf("%d",&std.rollNo);
        printf("Enter Date of Birth [DD MM YY] format: ");
        scanf("%d%d%d",&std.DOB.dd,&std.DOB.mm,&std.DOB.
yy);
        printf("\nName : %s \nRollNo : %d \nDate of birth :
        %02d/%02d/%02d\n",std.name,std.rollNo,std.DOB.dd,std.DO
        B.mm,std.DOB.yy);
        return 0;
}
```

6.
- ***Declare a structure pointer to structure book with attributes bname, author, pages with proper data type declarations.***
- ***Using the above declaration, WAP to display the content of structure above.***

```
#include <stdio.h>
#include <string.h>
struct Books {
  char  title[50];
  char  author[50];
  char  subject[100];
  int   book_id;
};
int main( ) {
  struct Books Book1;        /* Declare Book1 of type Book */
  struct Books Book2;        /* Declare Book2 of type Book */
  /* book 1 specification */
  strcpy( Book1.title, "C Programming");
```

```
    strcpy( Book1.author, "Nuha Ali");
    strcpy( Book1.subject, "C Programming Tutorial");
    Book1.book_id = 6495407;
    /* book 2 specification */
    strcpy( Book2.title, "Telecom Billing");
    strcpy( Book2.author, "Zara Ali");
    strcpy( Book2.subject, "Telecom Billing Tutorial");
    Book2.book_id = 6495700;
    /* print Book1 info */
    printf( "Book 1 title : %s\n", Book1.title);
    printf( "Book 1 author : %s\n", Book1.author);
    printf( "Book 1 subject : %s\n", Book1.subject);
    printf( "Book 1 book_id : %d\n", Book1.book_id);
    /* print Book2 info */
    printf( "Book 2 title : %s\n", Book2.title);
    printf( "Book 2 author : %s\n", Book2.author);
    printf( "Book 2 subject : %s\n", Book2.subject);
    printf( "Book 2 book_id : %d\n", Book2.book_id);
    return 0;
}
```

7. ***Can a structure variable be defined as a member of another structure? Can an array be included as a member of a structure? Can an array have structures as elements? If yes, then place suitable block codes for respectively.***

Nested structure in C is nothing but 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. You can learn below concepts in this section.

```
#include <stdio.h>
#include <string.h>
struct student_college_detail
{
   int college_id;
   char college_name[50];
};
struct student_detail
{
   int id;
   char name[20];
   float percentage;
   // structure within structure
```



```
   struct student_college_detail clg_data;
}stu_data;
int main()
{
   struct student_detail stu_data = {1, "Raju", 90.5, 71145,
                                     "Anna University"};
   printf(" Id is: %d \n", stu_data.id);
   printf(" Name is: %s \n", stu_data.name);
   printf(" Percentage is: %f \n\n", stu_data.percentage);
   printf(" College Id is: %d \n",
            stu_data.clg_data.college_id);
   printf(" College Name is: %s \n",
            stu_data.clg_data.college_name);
   return 0;
}
```

8. ***Design a structure to store a length in yards, feet and inches (for example 7 yards, 2 feet, 3 inches). Write a function to find difference between two measurements as represented by these structures.***

```
#include <stdio.h>
struct Distance
{
  int yards;
  int feet;
   float inch;
} d1, d2, sumofDistances;
int main()
{
   printf("Enter information for 1st distance\n");
   printf("Enter feet: ");
   scanf("%d", &d1.feet);
   printf("Enter inch: ");
   scanf("%f", &d1.inch);
   printf("\nEnter information for 2nd distance\n");
   printf("Enter feet: ");
   scanf("%d", &d2.feet);
   printf("Enter inch: ");
   scanf("%f", &d2.inch);
   sumofDistances.feet = d1.feet+d2.feet;
   sumofDistances.inch = d1.inch+d2.inch;
   // If inch is greater than 12, changing it to feet.
   if (sumofDistances.inch>12.0)
   {
      sumofDistances.inch = sumOfDistances.inch-12.0;
      ++sumofDistances.feet;
   }
```

```
    printf("\nSum of distances = %d\'-%.1f\"",sumofDistances.feet,
sumofDistances.inch);
    return 0;
}
```

9. ◆ *What are the various parts of a structure declaration? Explain each of them.*
◆ *What are structures? How are they stored in memory? What is the meaning of → in structures?*
◆ *What is structure? Define self referential structure with example.*

Structure

A structure gathers together different atoms of information that comprise a given entity. A structure contains a number of data types grouped together. These types may or may not be of the same type. The keyword struct is used to declare a structure then syntax is :

```
Struct < Structure Name >
{
Datatype field 1;
Datatype field 2;
Datatype field 3;
...    ...
...    ...
...    ...
};
```

Such structure which contains a number of fields that point to the same structure type are called self-referential structure.

A node may be represented in general form as follows :

```
Struct tag-name
{
#include<stdio.h>
#include<conio.h>
struct link_list
{
float price;
struct link_list *next;
};
struct link_list node1,node2;
void main()
{
clrscr();
node1.next=&node2;
```



```
node1.price=35.50;
node2.price=49.00;
printf("%f\n",node1.next->price);
}
```

Output : 49.000000

Arrow operator (→) : Arrow operator is used for accessing members of structure using pointer variable, below is the syntax of arrow operator in c programming.

Syntax of Arrow Operator :

```
struct student
{ char name[20],
  int roll; }*
ptr;
```

10. ***Write a program to create a structure and store its variable value in file.***

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
int main( )
{
struct bank
{
char name[10];
int acc;
double bal;
};
int n,i;
int ac;
printf("\nEnter the record size:");
scanf("%d",&n);
s=(struct bank*)malloc(n*sizeof(struct bank));
FILE *f;
f=fopen("bdata.txt","w");
for(i=0;i<n;i++)
{
printf("\nName:");
scanf("%s",&s[i].name);
printf("\nAcc no.:");
scanf("%d",&s[i].acc);
printf("\nBalance:");
scanf("%ld",&s[i].bal);
}
for(i=0;i<n;i++)
{
```

```
fprintf(f,"%s %d %ld",s[i].name,s[i].acc,s[i].bal);
}
fclose(f);
f=fopen("bdata.txt","r");
while(!feof(f))
{
for(i=0;i<n;i++)
{
fscanf(f,"%s %d %ld",&s[i].name,&s[i].acc,&s[i].bal);
printf("\nNAME=%s,ACCOUNT NO.%d",s[i].name,s[i].acc);
}
}
printf("\nENTER ACCNO.TO SEARCH:");
scanf("%d",&ac);
for(i=0;i<n;i++)
{
if(s[i].acc==ac)
{
printf("\n\nWELCOME %s,TO ADF BANK LTD.",s[i].name);
printf("\nCURRENT BALANCE: %ld",s[i].bal);
}
}
fclose(f);
getch( );
}
```

11. (1) Find Bugs or error in the following code of program (with mentioning line number).
(2) Explain how all bugs are corrected to generate compiled version of codes.
(3) After correction what is output of the code.

```
Void main ()
{
  struct book
  {
  int pages;
   float price;
  }
        * b1;
       b1.pages = 500;
       b1.price = 25.5;
       clrscr ();
          printf ("In pages :%ud", b1.pages);
       printf ("In price :%g", b1.price);
     }
```

(1) line 7 Structure required on left side of . or .*
line 8 Structure required on left side of . or .*

line 10 Structure required on left side of . or .*
line 11 Structure required on left side of . or .*

(2) Using accessing operator → (arrow) in place of . (dot)

(3) Output :

pages 500
price 25.5

12. *When do you use pointers to structure? Explain with an example.*

Passing and returning structures to functions may not be efficient, particularly if the structure is large. We can eliminate this excessive data movement by passing pointers to the structures to the function, and access them indirectly through the pointers.

```
#include<stdio.h>
#include<conio.h>
struct st{
int a;
char ch;
};
int main(void)
{
struct st obj;
struct st *stobj = &obj;
      stobj->a = 5;
stobj->ch = 'a';
printf("\n [%d] [%c]\n", stobj->a, stobj->ch);
return 0;
      }
```

13. *Write a C program to read and write the details of 10 employees using an array of structure :*

Field Name	Data type
emp-name	String
emp_no	Integer
address	String
pay	float

```
#include<stdio.h>
#include<conio.h>
struct emp
{
char emp_name[20],addr[50];
int emp_no;
```

```
float pay;
};
void main()
{
struct emp ee[3];
int i;
float x;
for(i=0;i<3;i++)
{
printf("\nEnter the details of the employee");
printf("\nENTER THE NAME OF THE EMPLOYEE");
fflush(stdin);
gets(ee[i].emp_name);
printf("\nENTER THE COMPLETE ADDRESS") ;
fflush(stdin);
gets(ee[i].addr);
printf("\nENTER THE EMPLOYEE NUMBER ");
scanf("%d",&ee[i].emp_no);
printf("\nENTER THE EMPLOYEE PAY");
fflush(stdin);
scanf("%f",&x);
ee[i].pay=x;
}
getch();
for(i=0;i<3;i++)
{
printf("\nDETAILS \n") ;
printf("\nTHE NAME OF THE EMPLOYEE=%s",ee[i].emp_name);
printf("\nENTER THE COMPLETE
    ADDRESS=%s",ee[i].addr);
        printf("\nENTER THE EMPLOYEE NUMBER
       =%d",ee[i].emp_no);
printf("\nENTER THE EMPLOYEE PAY=%f",ee[i].pay);
    getch();
}
getch();
}
```

Output :

```
Enter the details of the employee
ENTER THE NAME OF THE EMPLOYEE Ram

ENTER THE COMPLETE ADDRESS 112/56 Gumti

ENTER THE EMPLOYEE NUMBER 1

ENTER THE EMPLOYEE PAY 3000

Enter the details of the employee
ENTER THE NAME OF THE EMPLOYEE Rohit

ENTER THE COMPLETE ADDRESS 23/89 Govind Nagar

ENTER THE EMPLOYEE NUMBER 2

ENTER THE EMPLOYEE PAY 40000
```



```
Enter the details of the employee
ENTER THE NAME OF THE EMPLOYEE Rahul

ENTER THE COMPLETE ADDRESS 34/78 R.K.Nagar

ENTER THE EMPLOYEE NUMBER 3

ENTER THE EMPLOYEE PAY 4000

DETAILS
THE NAME OF THE EMPLOYEE= Ram
ENTER THE COMPLETE ADDRESS= 112/56 Gumti
ENTER THE EMPLOYEE NUMBER =1
ENTER THE EMPLOYEE PAY=3000.000000
DETAILS
THE NAME OF THE EMPLOYEE= Rohit
ENTER THE COMPLETE ADDRESS= 23/89 Govind Nagar
ENTER THE EMPLOYEE NUMBER =2
ENTER THE EMPLOYEE PAY=40000.000000
DETAILS
THE NAME OF THE EMPLOYEE= Rahul
ENTER THE COMPLETE ADDRESS= 34/78 R.K.Nagar
ENTER THE EMPLOYEE NUMBER =3
ENTER THE EMPLOYEE PAY=4000.000000
```

14. ***Write a program for constructing a union of structure.***

```
//PROGRAM FOR CONSTRUCTING A UNION OF STRUCTURE
#include<stdio.h>
#include<conio.h>
void main()
{ clrscr();
 struct ab
 { int x;
  char m[2]; };
 struct bc
 { int y;
  char n[2]; };
 union cd
 { struct ab key;
  struct bc data; };
 union cd strange;
 strange.key.x=510;
 strange.data.n[0]=0;
 strange.data.n[1]=33;
 printf("\n%d",strange.key.x);
 printf("\n%d",strange.data.y);
 printf("\n%d",strange.key.m[0]);
 printf("\n%d",strange.data.n[0]);
 printf("\n%d",strange.key.m[1]);
 printf("\n%d",strange.data.n[1]);
 getch();}
```

Output :

15. *Write a note on size of structure.*

We normally use structures, unions and arrays to create variables of large sizes. The actual size of these variables in terms of bytes may change from machine to machine. We may use the unary operator size of to tell us the size of a structure (or any variable). The expression

sizeof (struct *x*)

Will evaluate the number of *bytes* required to hold all the members of the structure *x*. If y is a simple structure variable of type struct *x*, then the expression

sizeof(*y*)

Would also give the same answer. However, if *y* is an array variable of type struet *x*, then

sizeof(*y*)

Would give the total number of *bytes* the array *y* requires.

This kind of information would be useful to determine the number of records in a database. For example, the expression :

sizeof(*y*)/sizeof(*x*)

would give the number of elements in the array *y*.

16. *Create a structure to specify data on students given below :*
Roll number, Name, Department, Course, Year of joining.
Assume that there are not more than 100 students in the college.
(1) *Write a function to print names of all students who joined in particular year.*
(2) *Write a function to print the data of a student whose roll number is given.*

Structure to Specify Data on Students

```
void main ( )
{
char j [1];
```

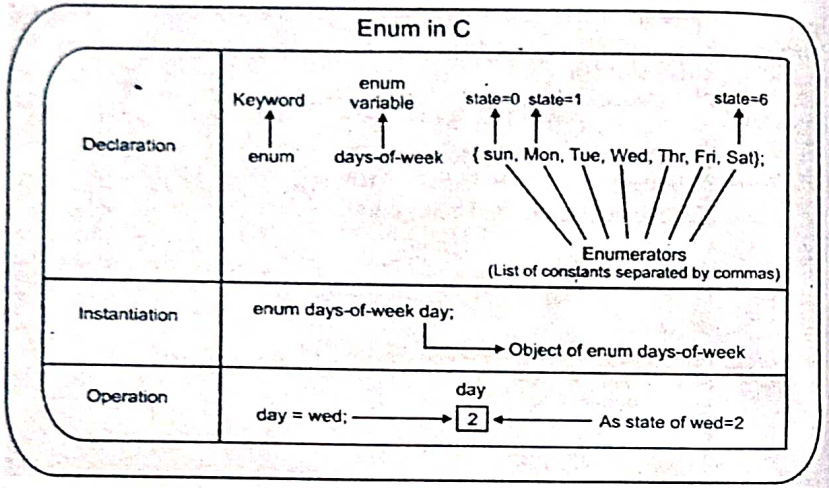


```
int year c, roll c, i;
struct student
{
int roll;
char name [20];
char department [20];
char course [20];
int year;
.};
struct student s[5] = {
{1, "Shambhavi" "C.S", "B.Tech", 2009},
{2, "Parul", "C.S" "B.Tech", 2008},
{3, Vineet", "E.C", "B.Tech", 2009},
{4, Ashish", "E.C", "B.Tech", 2010},
{5, Manish", "E.N", "B.Tech", 2009},
};
while (1)
{
clrscr ( );
printf ("\n enter a year (2000 or 2001)\n");
scanf ("%d", & year C);
for (i = 0, i<5, i++)
{
if (year c==S[i].year)
printf ("\n Name : % S year : % d", S[i]. name, S[i].year);
}
printf ("\n enter a roll number (1-5)\n");
scanf ("%d", & roll c);
for (i = 0; i <5; i ++)
{
        if ("\n Roll number : %d, Name :%S, Department :%S\n
Course :%S year : %d",  S[i]. roll. S[i]. name, S[i]. department, S[i].
course, S[i]. year),
break;
}
}
printf ("\n press q to quit or any key to continue \n");
scanf ("%s", j);
if (j[0]==99)
break;
}
}
link float ( )
{
float a = 0, *b;
b = & a;
a = *b;
}
```

17. What is enum in C?

Enumeration (or enum) in C

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.



(Figure)

```
enum State {Working = 1, Failed = 0};
```

The keyword 'enum' is used to declare new enumeration types in C and C++. Following is an example of enum declaration.

```
// The name of enumeration is "flag" and the constant
// are the values of the flag. By default, the values
// of the constants are as follows:
// constant1 = 0, constant2 = 1, constant3 = 2 and
// so on.
enum flag{constant1, constant2, constant3, ..... };
```

Variables of type enum can also be defined. They can be defined in two ways :

```
// In both of the below cases, "day" is
// defined as the variable of type week.

enum week{Mon, Tue, Wed};
enum week day;

// Or

enum week{Mon, Tue, Wed}day;
```



```
// An example program to demonstrate working
// of enum in C
#include<stdio.h>

enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};

int main()
{
   enum week day;
   day = Wed;
   printf("%d",day);
   return 0;
}
```

Output :

```
2
```

In the above example, we declared "day" as the variable and the value of "Wed" is allocated to day, which is 2. So as a result, 2 is printed.

18. Explain union with the help of an example.

Union

Union is also a user defined data type. All the members of union share the same memory location. Size of union is decided by the size of largest member of union. If you want to use same memory location for two or more members, union is the best for that. Unions are similar to the structure. Union variables are created in same manner as structure variables. The keyword "union" is used to define unions in C language.

Here is the syntax of unions in C language,

```
union union_name {
  member definition;
} union_variables;
```

Here,

union_name – Any name given to the union.
member definition – Set of member variables.
union_variable – This is the object of union.

Here is an example of unions in C language,

```
#include <stdio.h>
#include <string.h>
union Data {
  int i;
  float f;
} data, data1;
int main( ) {
```

```
  printf( "Memory size occupied by data : %d\t%d", sizeof(data),
sizeof(data1));
  return 0;
}
```

Output : Memory size occupied by data: 4 4

19. ◆ ***What is meant by the storage class of a variable? Name the four storage class specifications included in C. Discuss with example.*** **(2020-21)**
    ◆ ***What are the storage classes in 'C'? Explain with the help of examples.***

Storage Classes

Storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify. We have four different storage classes in a C program :

(1) Automatic Storage Class
(2) Register Storage Class
(3) Static Storage Class
(4) External Storage Class

Automatic Storage Class : A variable defined within a function or block with auto keyword belongs to automatic storage class. All variables defined within a function or block by default belong to automatic storage class if no storage class is mentioned. Variables having automatic storage class are local to the block which they are defined in.

Example :

```
#include <stdio.h>
int main( )
{
 auto int i = 1;
      {
       auto int i = 2;
       {
        auto int i = 3;
       printf ( "\n%d ", i);
      }
       printf ( "%d ", i);
      } ,
      printf( "%d\n", i);
}
```

Output : 3 2 1

Register Storage Class : The register keyword declares a variable of register storage class. Variables belonging to register storage class are local to the block which they are defined in. A register declaration is equivalent to an auto declaration, but they are placed in CPU registers, not in the memory. Only a few variables are actually placed into registers, and only certain types are eligible; the restrictions are implementation-dependent. However, if a variable is declared register, the unary & (address of) operator may not be applied to it, explicitly or implicitly. Register variables are also given no initial value by the compiler.

Example :

```
#include <stdio.h>
 int main()
 {
  register int i = 10;
  int *p = &i; //error: address of register variable requested

  printf("Value of i: %d", *p);
  printf("Address of i: %u", p);
 }
```

Static Storage Class : The static keyword gives the declared variable static storage class. Static variables can be used within function or file. Unlike global variables, static variables are not visible outside their function or file, but they maintain their values between calls.

Example :

```
#include <stdio.h>
      void staticDemo()
{
       static int i;
       {
        static int i = 1;
        printf("%d ", i);
        i++;
       }
       printf("%d\n", i);
        i++;
}
 int main()
 {
  staticDemo();
  staticDemo();
 }
```

Output : 1 0
2 1

External Storage Class : The extern keyword gives the declared variable external storage class. The principal use of extern is to specify that a variable is declared with external linkage elsewhere in the program.

Example :

```
#include <stdio.h>
 extern int x;
int main()
{
  printf("x: %d\n", x);
}
```

Output : int x = 10

20. *Differentiate between the Array & Structure using suitable example.* (2020-21)

Difference between Array and Structure

Points	Array	Structure
Basic	An array is a collection of variables of same data type.	A structure is a collection of variables of different data type.
Syntax	type array_name[size];	structsruct_name{ type element1; type element1; . . } variable1, variable2, . .;
Memory	Array elements are stored in contiguous memory location.	Structure elements may not be stored in a contiguous memory location.
Access	Array elements are accessed by their index number.	Structure elements are accessed by their names.
Operator	Array declaration and element accessing operator is "[ ]" (square bracket).	Structure element accessing operator is "." (Dot operator).
Pointer	Array name points to the first element in that array so, array name is a pointer.	Structure name does not point to the first element in that structure so, structure name is not a pointer.
Objects	Objects (instances) of an array cannot be created.	Structure objects (instance or structure variable) can be created.
Size	Every element in array is of same size.	Every element in a structure is of different data type.

Example of Array : In C programming to find out the average of 4 integers :

```
#include <stdio.h>
int main()
{
int avg = 0;
int sum =0;
int x=0;
   /* Array- declaration – length 4*/
int num[4];
   /* We are using a for loop to traverse through the array
    * while storing the entered values in the array
    */
for (x=0; x<4;x++)
   {
printf("Enter number %d \n", (x+1));
scanf("%d", &num[x]);
   }
for (x=0; x<4;x++)
   {
sum = sum+num[x];
   }
avg = sum/4;
printf("Average of entered number is: %d", avg);
return 0;
}
```

Output :

```
Enter number 1
10
Enter number 2
10
Enter number 3
20
Enter number 4
40
Average of entered number is: 20
```

Example of Structure In C programming to Store Employee Details.

```
#include<stdio.h>
#include<conio.h>
struct emp
{
int id;
```

```
        char name[36];
        float sal;
        };
        void main()
        {
        struct emp e;
        clrscr();
        printf("Enter employee Id, Name, Salary: ");
        scanf("%d",&e.id);
        scanf("%s",&e.name);
        scanf("%f",&e.sal);
        printf("Id: %d",e.id);
printf("\nName: %s",e.name);
printf("\nSalary: %f",e.sal);
getch();
}
```

Output :

```
Enter employee Id, Name, Salary: 5 ABC 45000
Id : 05 Name: ABC Salary: 45000.00
```

21. ***What do you understand by C preprocessor? Explain the purpose of at least four C – preprocessors.*** **(2020-21)**

C-Preprocessors

(1) A unique feature of c language is the preprocessor. A program can use the tools provided by preprocessor to make his program easy to read, modify, portable and more efficient.

(2) The C preprocessor is a collection of special statements, called directives that are executed at the beginning of the compilation process.

(3) It operates under the control of preprocessor command lines and directives.

(4) Preprocessor directives are placed in the source program before the main line before the source code passes through the compiler it is examined by the preprocessor for any preprocessor directives.

(5) If there is any appropriate actions are taken then the source program is handed over to the compiler.

(6) Preprocessor directives follow the special syntax a. rules that are listed below.

(a) Executed by the pre-processor.

(b) Occurs before a program is compiled.

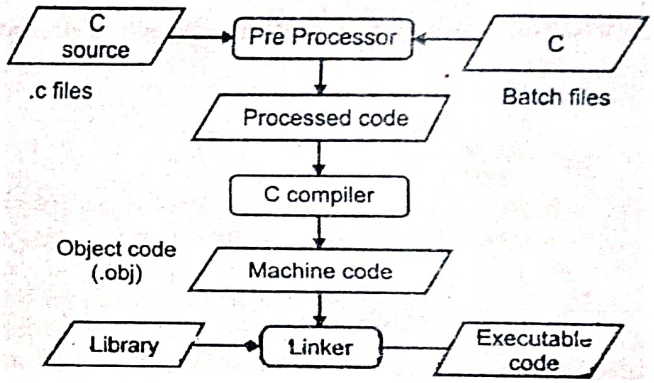
(c) Begin with #.

(d) Would not end with semicolon.

(e) Can be placed anywhere in the program.

(f) Normally placed at the beginning of the h. program or before any particular function.

(7) The compilation process can be diagrammatically given as below



(Figure)

Purpose of Four C-preprocessors :

(1) **#include :** The #include preprocessor directive is used to paste code of given file into current file. It is used include system-defined and user-defined header files. If included file is not found, compiler renders error.

```
#include <file> / #include "file"
```

(2) **Macro's (#define) :** A macro is a segment of code which is replaced by the value of macro. Macro is defined by #define directive.

Syntax :

```
#define token value
```

There are two types of macros :

(a) Object-like Macros

(b) Function-like Macros

(3) **#undef :** To undefine a macro means to cancel its definition. This is done with the #undef directive.

Syntax :

```
#undef token

define and undefine example
#include <stdio.h>
```

```
#define PI 3.1415
#undef PI
main() {
printf("%f",PI);
}
```

Output :

Compile Time Error: 'PI' undeclared

(4) #ifdef : The #ifdef preprocessor directive checks if macro is defined by #define. If yes, it executes the code.

Syntax :

```
#ifdef MACRO
//code
#endif
```

(5) #ifndef : The #ifndef preprocessor directive checks if macro is not defined by #define. If yes, it executes the code.

Syntax :

```
#ifndef MACRO
//code
#endif
```

❑❑

DYNAMIC MEMORY ALLOCATION

1. ◆ ***What is Dynamic Memory Allocation? Explain malloc() and calloc() function with suitable example.*** **(2020-21)**
   ◆ ***What is dynamic memory allocation? Explain various dynamic memory allocation functions.***

Dynamic Memory Allocation

When you declare a variable using a basic data type, the C compiler automatically allocates memory space for the variable in a pool of memory called the stack. For example, a float variable takes typically 4 bytes (according to the platform) when it is declared. However, there is a process for allocating memory which will permit you to implement a program in which the array size is undecided until you run your program (runtime). This process is called "Dynamic memory allocation."

Functions of Dynamic Memory Allocation

There are 4 functions which help in allocating dynamic memory :

malloc()	allocates single block of requested memory.
calloc()	allocates multiple block of requested memory.
realloc()	reallocates the memory occupied by malloc() or calloc() functions.
free()	frees the dynamically allocated memory.

2. ***Write a program in 'C' to explore malloc function.***

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  int n,i,*ptr,sum=0;
    printf("Enter number of elements: ");
    scanf("%d",&n);
    ptr=(int*)malloc(n*sizeof(int));  //memory allocated using malloc
    if(ptr==NULL)
    {
        printf("Sorry! unable to allocate memory");
        exit(0);
    }
```

```
    printf("Enter elements of array: ");
    for(i=0;i<n;++i)
    {
       scanf("%d",ptr+i);
       sum+=*(ptr+i);
    }
    printf("Sum=%d",sum);
    free(ptr);
 return 0;
}
```

3. *Write a program in 'C' to explore calloc function.*

```
#include<stdio.h>
#include<stdlib.h>
int main(){
 int n,i,*ptr,sum=0;
   printf("Enter number of elements: ");
   scanf("%d",&n);
   ptr=(int*)calloc(n,sizeof(int));  //memory allocated using calloc
   if(ptr==NULL)
   {
      printf("Sorry! unable to allocate memory");
      exit(0);
   }
   printf("Enter elements of array: ");
   for(i=0;i<n;++i)
   {
      scanf("%d",ptr+i);
      sum+=*(ptr+i);
   }
   printf("Sum=%d",sum);
   free(ptr);
 return 0;
}
```

4. *Write a program in 'C' to explore malloc, realloc and free function.*

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int *ptr, i , n1, n2;
   printf("Enter size of array: ");
   scanf("%d", &n1);
   ptr = (int*) malloc(n1 * sizeof(int));
   printf("Addresses of previously allocated memory: ");
```



```
   for(i = 0; i < n1; ++i)
      printf("%u\n",ptr + i);
   printf("\nEnter new size of array: ");
   scanf("%d", &n2);
   ptr = realloc(ptr, n2 * sizeof(int));
   printf("Addresses of newly allocated memory: ");
   for(i = 0; i < n2; ++i)
      printf("%u\n", ptr + i);
   return 0;
free(ptr);
}
```

5. ◆ ***What are the different file operations?*** **(2020-21)**

◆ ***Explain basic file handling operation performed in C.***

File is a collection of bytes that is stored on secondary storage devices like disk. There are 4 basic operations that can be performed on any files in C programming language. They are :

(1) Opening/Creating a file
(2) Closing a file
(3) Reading a file
(4) Writing in a file

6. *What are the various file handling functions in C.*

Table : Different File Handling Functions

File Operation	Declaration & Description
fopen() – To open a file	Declaration: FILE ***fopen** (const char *filename, const char *mode) fopen() function is used to open a file to perform operations such as reading, writing etc. In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist. FILE *fp; fp=**fopen** ("filename", "mode"); Where, fp – file pointer to the data type "FILE". filename – the actual file name with full path of the file.mode – refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+. Please refer below the description for these mode of operations.

fclose() – To close a file	Declaration : int fclose(FILE *fp); fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below. fclose (fp);
fgets() – To read a file	Declaration: char *fgets(char *string, int n, FILE *fp) fgets function is used to read a file line by line. In a C program, we use fgets function as below. **fgets** (buffer, size, fp), where, buffer – buffer to put the data in. size – size of the buffer fp – file pointer
fprintf() – To write into a file	**Declaration** : int **fprintf**(FILE *fp, const char *format, ...);fprintf() function writes string into a file pointed by fp. In a C program, we write string into a file as below. fprintf (fp, "some data"); or fprintf (fp, "text %d", variable_name);

There are many modes in opening a file. Based on the mode of a file, it can be opened for reading or writing or appending the texts. They are listed below.

(1) **r** : Opens a file in read mode and sets pointer to the first character in the file. It returns null if the file does not exist.

(2) **w** : Opens a file in write mode. It returns null if the file could not be opened. If file exists, data is overwritten.

(3) **a** : Opens a file in append mode. It returns null if the file couldn't be opened.

(4) **r+** : Opens a file for read and write mode and sets pointer to the first character in the file.

(5) **w+** : opens a file for read and write mode and sets pointer to the first character in the file.

(6) **a+** : Opens a file for read and write mode and sets pointer to the first character in the file. But, it can't modify existing contents.

7. ◆ ***What is a file? What is the benefit of using file?***
◆ ***What is meant by opening a data file? How is this accomplished?***

Defining and Opening a File

If we want to store data in a file in the secondary memory, we must specify certain things about the file, to the operating system. They include :

(1) File name.
(2) Data structure.
(3) Purpose.

Data structure of a file is defined as FILE in the library of standard I/O function definitions. Therefore, all files should be declared as type FILE before they are used. FILE is a defined data type.

When we open a file, we must specify what we want to do with the file. For example, we may write data to the file or read the already existing data.

Following is the general format for declaring and opening a file :

```
File *fp;
fp = fopen("filename," "mode");
```

The first statement declares the variable fp as a "pointer to the data type FILE". As stated earlier, FILE is a structure that is defined in the I/O library. The second statement opens the file named filename and assigns an identifier to the FILE type pointer fp. This pointer which contains all the information about the file is subsequently used as a communication link between the system and the program.

The second statement also specifies the purpose of opening this file. The mode does this job. Mode can be one of the following :

r	Open the file for reading only.
w	Open the file for writing only.
a	Open the file for appending (or adding) data to it.

Filename and mode are specified as strings. They should be enclosed in double quotation marks.

Consider the following statements :

```
File  *p1, *p2;
p1    = fopen("data", 'r');
p2    = fopen("results", "w");
```

The file data is opened for reading and results is opened for writing.

Many recent compliers include additional modes of operation. They include.

r+	The existing file is opened to the beginning for both reading and writing.
w+	Same is w except both for reading and writing.
a+	Same as a except both for reading and writing.

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

```
#include<conio.h>
void main()
{
 FILE *f1;
 char c;
 printf("Data Input \n\n");
 f1=fopen("INPUT","w");
 while((c=getchar())!=EOF)
 putc(c,f1);
 fclose(f1);
 printf("\nData Output\n\n");
 f1=fopen("INPUT","r");
 while((c=getc(f1))!=EOF)
 printf("%c",c);
 fclose(f1);
}
```

Output :

Benefit of Using File : A binary file is no different to a text file but it is a collection of bytes rather than characters. In C Programming Language a byte and a character are equivalent. Hence a binary file is also referred to as a character stream, but there are two basic differences.

(1) No special processing of the data occurs and each byte of data is transferred to or from the disk unprocessed.

(2) C Programming Language places no constructs on the file, and it may be read from, or written to, in any manner chosen by the programmer.

8. ***WAP to write data to text file "abc.txt" then read and display the content of the file.***

```
#include<stdio.h>
#include<string.h>
int main()
{
    FILE *fp;    //File Pointer
```



```
fp=fopen("abc.txt","w");//Opens files (write mode) abc.txt or creates it
ifits not found in the specified directory
    if(fp=NULL)
        printf("Error\n");
        char sen[100];
    printf("Enter Sentence\n");
    gets(sen);
    //printf("%s",sen);fprintf(fp,"%s",sen);
    char* sen2
    fscanf(fp,"%s", &sen2);
    printf("%s",sen2);
    fclose(fp);
    return 0;
}
```

9. ***Explain file handling in C. Explain in detail about the following standard library function with examples :***

(1) fopen()
(2) fseek()
(3) fscanf()
(4) fprintf()
(5) fgetc()
(6) fputc()

File Handling in C

(1) Creating a Text File and Output Some Data : We will start this section with an example of writing data to a file. We begin as before with the include statement for stdio.h, then define some variables for use in the example including a rather strange looking new type.

```
#include "stdio.h"
 main( )
 {
  FILE *fp;
  char stuff[25];
  int index;
  fp = fopen("TENLINES.TXT","w"); /* open for writing */
  strcpy(stuff,"This is an example line.");
  for (index = 1; index <= 10; index++)
  fprintf(fp,"%s Line number %d\n", stuff, index);
  fclose(fp); /* close the file before ending program */
 }
```

The type FILE is used for a file variable and is defined in the stdio.h file. It is used to define a file

pointer for use in file operations. The definition of C contains the requirement for a pointer to a FILE, and as usual, the name can be any valid variable name.

(2) **Opening a File :** Before we can write to a file, we must open it. What this really means is that we must tell the system that we want to write to a file and what the filename is. We do this with the fopen function illustrated in the first line of the program. The file pointer, fp in our case, points to the file and two arguments are required in the parentheses, the filename first, followed by the file type.

The filename is any valid DOS filename, and can be expressed in upper or lower case letters, or even mixed if you so desire. It is enclosed in double quotes. For this example we have chosen the name TENLINES.TXT. This file should not exist on your disk at this time. If you have a file with this name, you should change its name or move it because when we execute this program, its contents will be erased. If you don't have a file by this name, that is good because we will create one and put some data into it. You are permitted to include a directory with the filename. The directory must, of course, be a valid directory otherwise an error will occur. Also, because of the way C handles literal strings, the directory separation character '\' must be written twice. For example, if the file is to be stored in the \PROJECTS subdirectory then the filename should be entered as "\\PROJECTS\\TENLINES.TXT"

(3) **Reading (*r*) :** The second parameter is the file attribute and can be any of three letters, *r*, *w*, or *a*, and must be lower case. When an *r* is used, the file is opened for reading, a *w* is used to indicate a file to be used for writing, and indicates that you desire to append additional data to the data already in an existing file. Most C compilers have other file attributes available; check your Reference Manual for details. Using the r indicates that the file is assumed to be a text file. Opening a file for reading requires that the file already exist. If it does not exist, the file pointer will be set to NULL and can be checked by the program.

(4) **Writing (*w*) :** When a file is opened for writing, it will be created if it does not already exist and it will be reset if it does, resulting in the deletion of any data already there. Using the *w* indicates that the file is assumed to be a text file.

(5) **Appending (*a*) :** When a file is opened for appending, it will be created if it does not already exist and it will be initially empty. If it does exist, the data input point will be positioned at the end of the present data so that any new data will be added to any data that already exists in the file. Using the *a* indicates that the file is assumed to be a text file.

(6) **Outputting to the File :** The job of actually outputting to the file is nearly identical to the outputting we have already done to the standard output device. The only real differences are the new function names and the addition of the file pointer as one of the function arguments. In the example program, fprintf replaces our familiar printf function name, and the file pointer defined earlier is the first argument within the parentheses. The remainder of the statement looks like, and in fact is identical to, the printf statement.

(7) **Closing a File :** To close a file you simply use the function fclose with the file pointer in the parentheses. Actually, in this simple program, it is not necessary to close the file because the system will close all open files before returning to DOS, but it is good programming practice for you to close all files in spite of the fact that they will be closed automatically, because that would act as a reminder to you of what files are open at the end of each program.

You can open a file for writing, close it and reopen it for reading, then close it, and open it again for appending, etc. Each time you open it, you could use the same file pointer, or you could use a different one. The file pointer is simply a tool that you use to point to a file and you decide what file it will point to. Compile and run this program. When you run it, you will not get any output to the monitor because it doesn't generate any. After running it, look at your directory for a file named TENLINES.TXT and type it; that is where your output will be. Compare the output

with that specified in the program; they should agree! Do not erase the file named TENLINES.TXT yet; we will use it in some of the other examples in this section.

Reading From a Text File : Now for our first program that reads from a file. This program begins with the familiar include, some data definitions, and the file opening statement which should require no explanation except for the fact that an *r* is used here because we want to read it.

```
#include "stdio.h"
main( )
{
 FILE *funny;
 char c;
 funny = fopen("TENLINES.TXT", "r");
 if (funny == NULL) printf("File doesn't exist\n");
 else {
  do {
   c = getc(funny); /* get one character from the file
   */
    putchar(c); /* display it on the monitor
   */
   } while (c != EOF); /* repeat until EOF (end of file)
  */
 }
 fclose(funny);
}
```

In this program we check to see that the file exists, and if it does, we execute the main body of the program. If it doesn't, we print a message and quit. If the file does not exist, the system will set the pointer equal to NULL which we can test. The main body of the program is one do while loop in which a single character is read from the file and output to the monitor until an EOF (end of file) is detected from the input file. The file is then closed and the program is terminated. At this point, we have the potential for one of the most common and most perplexing problems of programming in C. The variable returned from the getc function is a character, so we can use a char variable for this purpose. There is a problem that could develop here if we happened to use an unsigned char however, because C usually returns a minus one for an EOF - which an unsigned char type variable is not capable of containing. An unsigned char type variable can only have the values of

zero to 255, so it will return a 255 for a minus one in C. This is a very frustrating problem to try to find. The program can never find the EOF and will therefore never terminate the loop. This is easy to prevent : always have a char or int type variable for use in returning an EOF. There is another problem with this program but we will worry about it when we get to the next program and solve it with the one following that.

After you compile and run this program and are satisfied with the results, it would be a good exercise to change the name of TENLINES.TXT and run the program again to see that the NULL test actually works as stated. Be sure to change the name back because we are still not finished with TENLINES.TXT.

(1) fseek() & ftell() : In C there are two I/O functions, fseek() and ftell(), that are designed to deal with random access. One of the members in the FILE structure is called the file position indicator. The file position indicator has to point to the desired position in a file before data can be read from or written to there. You can use the fseek() function to move the file position indicator to the spot you want to access in a file.

The syntax for the fseek() function is :

```
#include<stdio.h>
int fseek(FILE *stream, long offset, int whence);
```

Here stream is the file pointer associated with an opened file. Offset indicates the number of bytes from a fixed position, specified by whence, that can have one of the following integral values represented by SEEK_SET, SEEK_CUR, and SEEK_END. If it is successful, the fseek() function returns 0; otherwise, the function returns a nonzero value.

You can find the values represented by SEEK_SET, SEEK_CUR, and SEEK_END in the header file stdio.h.

If SEEK_SET is chosen as the third argument to the fseek() function, the offset is counted from the beginning of the file and the value of the offset is greater than or equal to zero. If, however, SEEK_END is picked up, then the offset starts from the end of the file; the value of the offset should be

negative. When SEEK_CUR is passed to the fseek() function, the offset is calculated from the current value of the file position indicator.

You can obtain the value of the current file position indicator by calling the ftell() function.

The syntax for the ftell() function is :

```
#include<stdio.h>
long ftell(FILE *stream);
```

(2) **The fscanf() and fprintf() Functions :** The two C library functions scanf() and printf() can be used to read or write formatted data through the standard I/O (that is, stdin and stdout). Among the C disk file I/O functions, there are two equivalent functions, fscanf() and fprintf(), that can do the same jobs as the scanf() and printf() functions. In addition, the fscanf() and fprintf() functions allow the programmer to specify I/O streams. The syntax for the fscanf() function is

```
#include<stdio.h>
int fscanf(FILE *stream, const char *format, ...);
```

Here stream is the file pointer associated with an opened file format, whose usage is the same as in the scanf() function, is a char pointer pointing to a string that contains the format specifiers. If successful, the fscanf() function returns the number of data items read. Otherwise, the function returns EOF.

The syntax for the fprintf() function is :

```
#include<stdio.h>
int fprintf(FILE *stream, const char *format, ...);
```

Here stream is the file pointer associated with an opened file format, whose usage is the same as in the printf() function, is a char pointer pointing to a string that contains the format specifiers. If successful, the fprintf() function returns the number of formatted expressions. Otherwise, the function returns a negative value.

(3) **fopen() :** The C I/O function fopen() gives you the ability to open a file and associate a stream to the opened file. You need to specify the way to open a file and the filename with the fopen() function.

The syntax for the fopen() function is :

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



```
FILE *fopen(const char *filename, const char *mode);
```

Here filename is a char pointer that references a string of a filename. The filename is given to the file that is about to be opened by the fopen() function mode points to another string that specifies the way to open the file. The fopen() function returns a pointer of type FILE. If an error occurs during the procedure to open a file, the fopen() function returns a null pointer.

(4) **getc() :** The getc() function reads the next character from a file stream, and returns the character as an integer.

The syntax for the getc() function is :

```
#include<stdio.h>
int getc(FILE *stream);
```

Here FILE *stream declares a file stream (that is, a variable). The function returns the numeric value of the character read. If an end-of-file or error occurs, the function returns EOF.

The C language provides another function, getchar(), to perform a similar operation to getc(). More precisely, the getchar() function is equivalent to getc(stdin).

The syntax for the getchar() function is :

```
#include<stdio.h>
int getchar(void);
```

Here void indicates that no argument is needed for calling the function. The function returns the numeric value of the character read. If an end-of-file or error occurs, the function returns EOF.

(5) **putc() :** The putc() function writes a character to the specified file stream, which is the standard output pointing to your screen.

The syntax for the putc() function is :

```
#include<stdio.h>
int putc(int c, FILE *stream);
```

Like putc(), putchar() can also be used to put a character on the screen. The only difference between the two functions is that putchar() needs only one argument to contain the character. You don't need to specify the file stream, because the standard output (stdout) is the default file stream to putchar().

The syntax for the putchar() function is :

```
#include<stdio.h>
int putchar(int c);0
```

Here int c is the argument that contains the numeric value of a character. The function returns EOF if an error occurs; otherwise, it returns the character that has been written.

(6) **fgets & fputs :** Besides reading or writing one character at a time, you can also read or write one character line at time. There is a pair of C I/O functions, fgets() and fputs(), that allows you to do so.

The syntax for the fgets() function is :

```
#include<stdio.h>
char *fgets(char *s, int n, FILE *stream);
```

The fgets() function can read up to n-1 characters, and can append a null character after the last character fetched, until a newline or an EOF is encountered. Note that if a newline is encountered during the reading, the fgets() function includes the newline in the array. This is different from what the gets() function does. The gets() function just replaces the newline character with a null character.

The syntax for the fputs() function is :

```
#include<stdio.h>
int fputs(const char *s, FILE *stream);
```

Here s points to the array that contains the characters to be written to a file associated with the file pointer stream. The const modifier indicates that the content of the array pointed to by s cannot be changed. If it fails, the fputs() function returns a nonzero value; otherwise, it returns zero.

(7) **fclose :** After a disk file is read, written, or appended with some new data, you have to disassociate the file from a specified stream by calling the fclose() function.

The syntax for the fclose() function is :

```
#include<stdio.h>
int fclose(FILE *stream);
```

Here stream is a file pointer that is associated with a stream to the opened file. If fclose() closes a file successfully, it returns 0. Otherwise, the function

returns EOF. Normally, the fclose() function fails only when the disk is removed before the function is called or there is no more space left on the disk.

Since all high-level I/O operations are buffered, the fclose() function flushes data left in the buffer to ensure that no data will be lost before it disassociates a specified stream with the opened file.

Note that a file that is opened and associated with a stream has to be closed after the I/O operation. Otherwise, the data saved in the file may be lost; some unpredictable errors might occur during the execution of your program.

(8) **ferror :** Any file maintained by the operating system has a particular size. As you write a file it is made bigger until you close it, so the only problems that you have when sending output to a file are concerned with what happens when you run out of disk space. The standard C input/output functions can tell you how many items they have successfully transferred; you should always use the value they give back to test that your dealings with files are going properly.

When you are reading from a file it is useful to be able to check whether or not you have reached the end, without just failing to read. The function feof can be used to find out if you have hit the end :

```
if( feof(input_file ) ) printf( "BANG!\n" ) ;
```

This function returns the value 0 if the end of the file has not been reached and another value if it has. It takes a FILE pointer as a parameter.

Note that binary and text files have different methods of determining the end of a file. If you are having problems because you keep reaching the end of the file before you think you should it may be because you have opened the file in the wrong mode.

Feof has a twin brother called ferror who can be called to find out if an error has been caused due to a file operation. It is worth calling this if you find that fewer items have been transferred by a read or a write than you expected. The error number that you get back is specific to the operating system you are using but it can be used to make program friendlier, for example your program could tell the difference

between "no disks in the drive" and "the disk has completely failed".

Rewind() :

Declaration : void rewind(FILE *fp) rewind function is used to move file pointer position to the beginning of the file. In a C program, we use rewind() as below.

```
rewind(fp);
#include<stdio.h>
#include<conio.h>
void main(){
FILE *fp;
char c;
clrscr();
fp=fopen("file.txt","r");

while((c=fgetc(fp))!=EOF){
printf("%c",c);
}
rewind(fp);//moves the file pointer at beginning of the file
while((c=fgetc(fp))!=EOF){
printf("%c",c);
}

fclose(fp);
getch();
}
```

10. *Describe two different methods for creating a stream-oriented data file. Can both methods be used with unformatted data files?*

Two Different Methods of Stream Oriented Data File

In the first category, the data file comprises consecutive characters. These characters can be interpreted as individual data items or as components of strings or numbers. These are called text files.

In the second category, often called as unformatted data files, organizes data into blocks containing contiguous bytes of information. These blocks represent more complex data structures, such as arrays and structures. These files are called binary files.

No, second category is used for unformatted data files.

11. *Give a brief description of how a file is organized in the form of tables.*

File Organization

Data files are organized so as to facilitate access to records and to ensure their efficient storage. A tradeoff between these two requirements generally exists: if rapid access is required, more storage is required to make it possible.

Access to a record for reading it is the essential operation on data. There are two types of access :

(1) Sequential Access : It is performed when records are accessed in the order they are stored. Sequential access is the main access mode only in batch systems, where files are used and updated at regular intervals.

(2) Direct Access : On-line processing requires direct access, whereby a record can be accessed without accessing the records between it and the beginning of the file. The primary key serves to identify the needed record.

There are three methods of file organization :

(a) Sequential organization
(b) Indexed-sequential organization
(c) Direct organization

Sequential Organization : In sequential organization records are physically stored in a specified order according to a key field in each record.

Indexed-Sequential Organization : In the indexed-sequential files method, records are physically stored in sequential order on a magnetic disk or other direct access storage device based on the key field of each record. Each file contains an index that references one or more key fields of each data record to its storage location address.

Direct Organization : Direct file organization provides the fastest direct access to records. When using direct access methods, records do not have to be arranged in any particular sequence on storage media. Characteristics of the direct access method include :

(1) Computers must keep track of the storage location of each record using a variety of direct organization

methods so that data can be retrieved when needed.

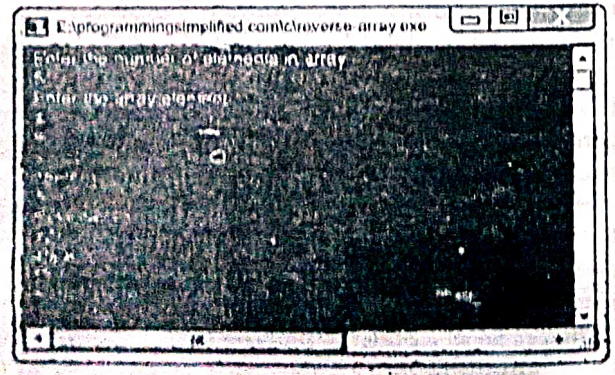
(2) New transactions' data do not have to be sorted.

(3) Processing that requires immediate responses or updating is easily performed.

12. *Write a program that displays the contents of a file in reverse order.*

```
#include <stdio.h>
int main()
{ int n, c, d, a[100], b[100];
  printf("Enter the number of elements in array\n");
  scanf("%d", &n);
  printf("Enter the array elements\n");
  for (c = 0; c < n ; c++)
    scanf("%d", &a[c]);
  /*
   * Copying elements into array b starting from end of array a
   */
  for (c = n - 1, d = 0; c >= 0; c--, d++)
    b[d] = a[c];
  /*
   * Copying reversed array into original.
   * Here we are modifying original array, this is optional.
   */
  for (c = 0; c < n; c++)
    a[c] = b[c];
  printf("Reverse array is\n");
  for (c = 0; c < n; c++)
    printf("%d\n", a[c]);
  return 0;}
```

Output of Program :



(Figure)

13. *Name the function to write a block of data to a file in standard I/O. Write a program to check the number is prime or not.*

Writing a File

Following is the simplest function to write individual characters to a stream

```
int fputc( int c, FILE *fp );
```

- The function fputc() writes the character value of the argument c to the output stream referenced by fp. It returns the written character written on success otherwise EOF if there is an error. You can use the following functions to write a null-terminated string to a stream

```
int fputs( const char *s, FILE *fp );
```

The function fputs() writes the string s to the output stream referenced by fp. It returns a non-negative value on success, otherwise EOF is returned in case of any error. You can use int fprintf(FILE *fp,const char *format, ...) function as well to write a string into a file. Try the following example.

Make sure you have /tmp directory available. If it is not, then before proceeding, you must create this directory on your machine.

```
#include <stdio.h>
main() {
  FILE *fp;
  fp = fopen("/tmp/test.txt", "w+");
  fprintf(fp, "This is testing for fprintf...\n");
  fputs("This is testing for fputs...\n", fp);
  fclose(fp);
}
```

When the above code is compiled and executed, it creates a new file test.txt in /tmp directory and writes two lines using two different functions.

Program for Checking the Prime Number

```
#include <stdio.h>
#include<conio.h>
main() {
  int n, i, c = 0;
  printf("Enter any number n:");
  scanf("%d", &n);
  /*logic*/
```

```
    for (i = 1; i <= n; i++) {
      if (n % i == 0) {
        c++;
      }
    }
    if (c == 2) {
        printf("n is a Prime number");
    }
    else {
        printf("n is not a Prime number");
    }
    return 0;
}
```

Program Output :

Enter any number n: 7
n is Prime

14. *Write the use of library function rename ( ). Write a program to delete a file using remove ( ) function.*

Use of Library Function Rename( )

The C library function int rename (const char *old_filename, const char *new_filename) causes the filename referred to by old_filename to be changed to new_filename.

Declaration : Following is the declaration for rename() function.

```
int rename(const char *old_filename, const char *new_filename)
```

Parameters :

(1) Old_filename : This is the C string containing the name of the file to be renamed and/or moved.

(2) New_filename : This is the C string containing the new name for the file.

Return Value : On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

Example : The following example shows the usage of rename() function.

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



```
  int ret;
  char oldname[] = "file.txt";
  char newname[] = "newfile.txt";
  ret = rename(oldname, newname);
  if(ret == 0)
  {
    printf("File renamed successfully");
  }
  else
  {
    printf("Error: unable to rename the file");
  }
  return(0);
}
```

Let us assume we have a text file file.txt, having some content. So, we are going to rename this file, using the above program. Let us compile and run the above program to produce the following message and the file will be renamed to newfile.txtfile.

Program to Delete a File Using Remove ( ) Function

```
#include<stdio.h>
main()
{
  int status;
  char file_name[25];
  printf("Enter the name of file you wish to delete\n");
  gets(file_name);
  status = remove(file_name);
  if( status == 0 )
    printf("%s file deleted successfully.\n",file_name);
  else
  {    printf("Unable to delete the file\n");
    perror("Error");  }
  return 0;}
```

Download Delete file program executable.

Output :

```
Enter the name of file you wish to delete
leap-year.c
leap-year.c file deleted successfully

Process returned 0(0x0) execution time : 9.829 s
Press any key to continue.
```

15. *Explain the Object file.*

An object file is the real output from the compilation phase. It's mostly machine code, but has info that allows a linker to see what symbols are in it as well as symbols it requires in order to work.

16. *Explain the Binary File.*

A binary file is no different to a text file. It is a collection of bytes. In C Programming Language a byte and a character are equivalent. Hence, a binary file is also referred to as a character stream, but there are two essential differences.

(1) No special processing of the data occurs and each byte of data is transferred to or from the disk unprocessed.

(2) C Programming Language places no constructs on the file, and it may be read from, or written to, in any manner chosen by the programmer.

Binary files can be either processed sequentially or, depending on the needs of the application, they can be processed using random access techniques. In C Programming Language, processing a file using random access techniques involves moving the current file position to an appropriate place in the file before reading or writing data. This indicates a second characteristics of binary files-they a generally processed using read and write operations simultaneously.

For example, a database file will be created and processed as a binary file. A record update operation will involve locating the appropriate record, reading the record into memory, modifying it in some way, and finally writing the record back to disk at its appropriate location in the file. These kinds of operations are common to many binary files, but are rarely found in applications that process text files.

17. *Write a program to search a word in file and replace with the word input by the user.*

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



```
#include<conio.h>
#include<stdlib.h>
count_data( );
void main( )
{ // calling function
  count_data( );
  getch( );}
count_data( ) // function for count no of words,lines & characters.
{ FILE *fp,*fp_rep;
  char ch,ch1,temp_str[50],rep_str[10],new_str[10];
  int count=0; // counter
  clrscr();
  fp=fopen("c:\\windows\\desktop\\input.txt","r");
  fp_rep=fopen("c:\\windows\\desktop\\input1.txt","w");
  printf("\nEnter String to find:");
  scanf("%s",rep_str);
  printf("\nEnter String to replace:");
  scanf("%s",new_str);
  while((ch=getc(fp))!=EOF)
  { if(ch==' ')
    { temp_str[count]='\0';
      if(strcmp(temp_str,rep_str)==0)
    { printf("Do u want to replace(y/n):");
     ch1=getche( );
     if(ch1=='y')
     {fprintf(fp_rep," %s",new_str);
      count=0;}
     else
      { fprintf(fp_rep," %s",temp_str);count=0;}  }
   Else {
fprintf(fp_rep," %s",temp_str);
     count=0; } }
else {
 temp_str[count++]=ch; } }
    if(strcmp(temp_str,rep_str)==0)
   { printf("Do u want to replace(y/n):");
    ch1=getche( );
    if(ch1=='y')
    {  fprintf(fp_rep,"%s ",new_str);
     count=0; }
    else
     {  fprintf(fp_rep,"%s ",temp_str);
      count=0;  } }
else
    {  fprintf(fp_rep,"%s ",temp_str);  }
   fclose(fp);
```

```
    fclose(fp_rep);
    remove("c:\\windows\\desktop\\input.txt");
    rename ("c:\\windows\\desktop\\ input1.txt","c:\\windows\\ desktop
\\input.txt");
    fflush(stdin);}
```

18. *Write a program to copy a file into another but in reverse order.*

```
#include<stdio.h>
#include<conio.h>
void main( )
{FILE *in,*out;
char ch,f1[80],f2[80];
long loc;
clrscr( );
printf("\n Enter Name of Source File:");
scanf("%s",&f1);
printf("\n Enter Name of Target File:");
scanf("%s",&f2);
in=fopen(f1,"w");
if(f1==NULL)
{printf("\nCannot open file");
exit(0);}
Else
{printf("\nEnter data in file %s(Press q to stop):",f1);
while(1)
{ch=getchar( );
if(ch=='q')
break;
else
fputc(ch,in);}
fclose(in);}
in=fopen(f1,"r");
if(f1==NULL)
{printf("\nfile does not exist");
exit(0);}
out=fopen(f2,"w");
if(f2==NULL)
{printf("\ncannot open file");
exit(0);}
fseek(in, 0, SEEK_END);
loc = ftell(in);
loc = loc-1;
while(loc >= 0)
```



```
{fseek(in, loc, SEEK_END);
ch = fgetc(in);
fputc(ch, out);
loc--;}
printf("\nfile copied in reverse order successfully");
fcloseall( );
getch( );}
```

19. *List any three library functions for files and illustrate them with suitable examples.*

Library Functions for Files

Following are the functions defined in the header stdio.h :

(1) Fopen – opens a text file.

(2) Fclose – closes a text file.

(3) Fgets – reads a string from a file.

Program :

```
#include<stdio.h>
int main()  {
   FILE *ptr_file;
   char buf[1000];
   ptr_file =fopen("input.txt","r");
   if (!ptr_file)
     return 1;
   while (fgets(buf,1000, ptr_file)!=NULL)
     printf("%s",buf);
fclose(ptr_file);
   return 0;
}
```

20. *Write a program in 'C' for file open, file write and file close.*

```
# include <stdio.h>
# include <string.h>
 int main( )
{
   FILE *fp ;
   char data[50];
   // opening an existing file
   printf( "Opening the file test.c in write mode" ) ;
   fp = fopen("test.c", "w") ;
   if ( fp == NULL )
   {
      printf( "Could not open file test.c" ) ;
      return 1;
   }
```

```
    printf( "\n Enter some text from keyboard" \
          " to write in the file test.c" ) ;
    // getting input from user
    while ( strlen ( gets( data ) ) > 0 )
    {
       // writing in the file
       fputs(data, fp) ;
       fputs("\n", fp) ;
    }
    // closing the file
    printf("Closing the file test.c") ;
    fclose(fp) ;
    return 0;
}
```

21. *Write a program to update details of employee using files.*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct emp
{
   int empid;
   char *name;
};
int count = 0;
void add_rec(char *a);
void display(char *a);
void update_rec(char *a);
void main(int argc, char *argv[])
{
   int choice;
   while (1)
   {
      printf("MENU:\n");
      printf("1.Add a record\n");
      printf("2.Display the file\n");
      printf("3.Update the record\n");
      printf("Enter your choice:");
      scanf("%d", &choice);
      switch(choice)
      {
      case 1 :
         add_rec(argv[1]);
         break;
```



```
      case 2 :
         display(argv[1]);
         break;
      case 3 :
         update_rec(argv[1]);
         break;
      case 4 :
         exit(0);
      default:
         printf("Wrong choice!!!\nEnter the correct choice\n");
      }
   }
}
void add_rec(char *a)
{
   FILE *fp;
   fp = fopen(a, "a+");
   struct emp *temp = (struct emp *)malloc(sizeof(struct emp));
   temp->name = (char *)malloc(50*sizeof(char));
   if (fp == NULL)
      printf("Error!!!");
   else
   {
      printf("Enter the employee id\n");
      scanf("%d", &temp->empid);
      fwrite(&temp->empid, sizeof(int), 1, fp);
      printf("enter the employee name\n");
      scanf(" %[^\n]s", temp->name);
      fwrite(temp->name, 50, 1, fp);
      count++;
   }
   fclose(fp);
   free (temp);
   free(temp->name);
}
void display(char *a)
{
   FILE *fp;
   char ch;
   int rec = count;
   fp = fopen(a, "r");
   struct emp *temp = (struct emp *)malloc(sizeof(struct emp));
   temp->name = (char *)malloc(50*sizeof(char));
   if (fp == NULL)
      printf("Error!!");
   else
   {
      while (rec)
      {
```

```
            fread(&temp->empid, sizeof(int), 1, fp);
            printf("%d", temp->empid);
            fread(temp->name, 50, 1, fp);
            printf(" %s\n", temp->name);
            rec--;
          }
        }
        fclose(fp);
        free(temp);
       free(temp->name);
}
void update_rec(char *a)
{
    FILE *fp;
    char ch, name[5];
    int rec, id, c;
    fp = fopen(a, "r+");
    struct emp *temp = (struct emp *)malloc(sizeof(struct emp));
    temp->name = (char *)malloc(50*sizeof(char));
    printf("Enter the employee id to update:\n");
    scanf("%d", &id);
    fseek(fp, 0, 0);
    rec = count;
    while (rec)
    {
       fread(&temp->empid, sizeof(int), 1, fp);
       printf("%d", temp->empid);
       if (id == temp->empid)
       {
          printf("Enter the employee name to be updated");
          scanf(" %[^\n]s", name);
          c = fwrite(name, 50, 1, fp);
          break;
       }
       fread(temp->name, 50, 1, fp);
       rec--;
    }
    if (c == 1)
       printf("Record updated\n");
    else
       printf("Update not successful\n");
    fclose(fp);
    free(temp);
    free(temp->name);
}
```


22. *What is Graphics in C?*

Graphics programming in C used to drawing various geometrical shapes(rectangle, circle eclipse etc), use of mathematical function in drawing curves, coloring an object with different colors and patterns and simple animation programs like jumping ball and moving cars.

(1) First Graphics Program (Draw a Line) :

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

void main(void) {
   int gdriver = DETECT, gmode;
   int x1 = 200, y1 = 200;
   int x2 = 300, y2 = 300;
   clrscr();

   initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");
   line(x1, y1, x2, y2);
   getch();
   closegraph();
}
```

(2) Explanation of Code : The first step in any graphics program is to include graphics.h header file. The graphics.h header file provides access to a simple graphics library that makes it possible to draw lines, rectangles, ovals, arcs, polygons, images, and strings on a graphical window.

The second step is initialize the graphics drivers on the computer using initgraph method of graphics.h library.

void initgraph(int *graphicsDriver, int *graphicsMode, char *driverDirectoryPath);

It initializes the graphics system by loading the passed graphics driver then changing the system into graphics mode. It also resets or initializes all graphics settings like color, palette, current position etc, to their default values. Below is the description of input parameters of initgraph function.

(a) graphicsDriver : It is a pointer to an integer specifying the graphics driver to be used. It tells

the compiler that what graphics driver to use or to automatically detect the drive. In all our programs we will use DETECT macro of graphics.h library that instruct compiler for auto detection of graphics driver.

(b) **graphicsMode :** It is a pointer to an integer that specifies the graphics mode to be used. If *gdriver is set to DETECT, then initgraph sets *gmode to the highest resolution available for the detected driver.

(c) **driverDirectoryPath :** It specifies the directory path where graphics driver files (BGI files) are located. If directory path is not provided, then it will search for driver files in current working directory directory. In all our sample graphics programs, you have to change path of BGI directory accordingly where you Turbo C++ compiler is installed.

We have declared variables so that we can keep track of starting and ending point.

```
int x1=200, y1=200;
int x2=300, y2=300;
```

No, We need to pass just 4 parameters to the line function.

```
line(x1,y1,x2,y2);
```

line Function Draws Line From (x1,y1) to (x2,y2) .

Syntax : line(x1,y1,x2,y2);

Parameter Explanation :

(1) x1 - X Co-ordinate of First Point
(2) y1 - Y Co-ordinate of First Point
(3) x2 - X Co-ordinate of Second Point
(4) y2 - Y Co-ordinate of Second Point

At the end of our graphics program, we have to unloads the graphics drivers and sets the screen back to text mode by calling closegraph function.

23. *Write a program to draw a circle.*

Circle Graphics :

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



```
#include<graphics.h>
#include<dos.h>

void main()
{
int gd=DETECT, gm; int i;
initgraph (&gd,&gm,"c:\tc\\bgi");

clrscr();

for(i=0;i<500;i++)
{
      setcolor(i);
      //coordinates of center from x axis, y axis, radius
      circle(100,100,30+i);
      delay(30);
      //cleardevice(); //try with and without cleardevice();
}

getch();

closegraph();
}
```

Output :

The circles will keep on growing till the assigned number.

(Figure)

24. *Write a program in C to copy content from one file to another.* **(2020-21)**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  FILE *fptr1, *fptr2;
char filename[100], c;
printf("Enter the filename to open for reading \n");
scanf("%s",filename);
  // Open one file for reading
  fptr1 = fopen(filename, "r");
```

```
if (fptr1 == NULL)
{
printf("Cannot open file %s \n", filename);
exit(0);
  }
printf("Enter the filename to open for writing \n");
scanf("%s", filename);
  // Open another file for writing
  fptr2 = fopen(filename, "w");
if (fptr2 == NULL){
printf("Cannot open file %s \n", filename);
exit(0);
  }
  // Read contents from file
  c = fgetc(fptr1);
while (c != EOF)
{
fputc(c, fptr2);
    c = fgetc(fptr1);
  }
printf("\nContents copied to %s", filename);
fclose(fptr1);
fclose(fptr2);
return 0;
}
```

Output : When the above program is executed, it produce the following result :

```
Enter the filename to open for reading
file3.txt
Enter the filename to open for writing
file1.txt
Contents copied to file1.txt
```

25. *List and discuss various Library functions used in drawing.* **(2020-21)**

There are so many built in graphics functions define in C library. The basic graphics functions are described a below. Graphics functions are text mode graphics functions as well as graphical mode functions

Text Mode Graphics Functions : The text mode graphics functions place the text in certain area of the screen. Those functions are included in the header file <conio.h>. Some of those text mode graphics functions are :

(1) **window() :** It sets particular area on the output screen and displays the output text on that area.
The syntax of calling this window function is
window(10,10, 80,25);
where co-ordinate (10,10) represents the upper left corner of displaying window and (80,25) is the lower right corner of the display window.

(2) **clrscr() :** It is used to clear the screen and locates the curser at the beginning of the screen.

(3) **gotoxy(x,y) :** It moves the curser at specified co-ordinates (x,y) position.

(4) **cputs(string) :** It writes a string given to function to the user defined window screen.

(5) **putch(char) :** It writes a character given to function to the user defined area of window.

(6) **initgraph() :** It is one of the function that is used to initialize the computer in graphics mode. Its syntax is as:
initgraph(&gdriver, &gmode, "graphics driver path");

(7) **closegraph() :** It is the graphical function that is used to close the graphics mode initialized by the initgraph() function.

(8) **graphresult() :** It is a graphical function that returns the value 0 if the graphics is detected correctly and the driver is correctly initialized to correct graphics mode otherwise it returns some error code than 0.

(9) **grapherrormsg(errorcode) :** This function returns the message string corresponding to the errorcode returned by the graphresult() function.

(10) **cleardevice() :** This function is used to clear the screen in graphical mode as clrscr() in text mode.

(11) **setcolor(color) :** It is a function that is used to set the color of the drawing object with given color. The color is indicated by the integer from 0 to 15 (16 color)

(12) **moveto(x, y) :** It is used to move the cursor in display screen at specified co-ordinates by the value (x,y).

(13) **outtext(" text string ") :** Prints the text string in the current position of screen.

(14) **outtextxy(x, y, "string")** : Prints the string from the co-ordinate (x,y) position.

(15) **lineto(x, y)** : Draws the line from current position to (x,y) position.

(16) **putpixel(x, y, color)** : Displays a pixel(point) at (x,y) with given color.

(17) **line(a, b, c, d)** : Draws line from (a,b) to (c,d).

(18) **circle(x, y, r)** : Prints circle with center (x,y) and radius r.

(19) **rectangle(a, b, c, d)** : Prints rectangle where (a,b) is upper left coordinate and (c,d) is lower right co-ordinates of rectangle.

❑❑

MCA

(SEM. I) MODEL PAPER

KCA – 102 : PROBLEM SOLVING USING C

Time : Three Hours *Maximum Marks : 70*

Note : 1. Attempt *all* Sections. If require any missing data; then choose suitably.

SECTION – A

1. Attempt *all* questions in brief. **(2 ×7 = 14)**

(a) Write short note on the variables in 'C' programming.
(b) Explain the purpose of default in switch-case.
(c) Write a program in 'C' find out the average of 4 integers entered by the user.
(d) Explain structure with its declaration and initialization.
(e) Write a program in 'C' to explore malloc function.
(f) What are the various types of operators in 'C'?
(g) Write a program that swaps values of two variable using third variables.

SECTION – B

2. Attempt *any three* of the following : **(3 ×7 = 21)**

(a) What are the different bitwise operators used in C? Give an example of each.
(b) What is the use of "goto" and "labels" in 'C'?
(c) Define string. How to declare and initialize string?
(d) Using the above declaration, WAP to display the content of structure above.
(e) What is meant by opening a data file? How is this accomplished?

SECTION – C

3. Attempt *any one* part of the following : **(1 ×7 = 7)**

(a) What is concept of problem solving? Explain different problem solving techniques.
(b) Explain standard header files and functions of each header file in 'C'.

4. Attempt *any one* part of the following : **(1 ×7 = 7)**

(a) What do you mean by an array? In what way array is different from an ordinary variable?
(b) What is Union? How is it different from structures? Explain in detail the memory representation of structure and union. Use examples to justify your answers.

5. **Attempt *any one* part of the following :** **(1 ×7 = 7)**
   (a) What is meant by opening a data file? How is this accomplished?
   (b) Briefly discuss the history of 'C' programming.

6. **Attempt *any one* part of the following :** **(1 ×7 = 7)**
   (a) Write the syntax of switch statement. Explain with the help of an example.
   (b) What is enum in C?

7. **Attempt *any one* part of the following :** **(1 ×7 = 7)**
   (a) What is meant by opening a data file? How is this accomplished?
   (b) State the rules that determine the order in which initial values are assigned to multi-dimensional array elements.

❑❑

MCA

(SEM. I) THEORY EXAMINATION, 2020–21
KCA – 102 : PROBLEM SOLVING USING C

Time : Three Hours *Maximum Marks : 100*

Note : Attempt *all* Sections. If require any missing data, then choose suitably.

SECTION – A

1. **Attempt *all* questions in brief.** **(2 ×10 = 20)**
   (a) What are salient features of C Language?
   (b) What is Token? What are the different types of token available in C language?
   (c) What is an algorithm? Give the characteristics of an algorithm.
   (d) Write and explain syntax of "for" loop.
   (e) Write and explain about switch statement.
   (f) What is multi – dimensional array?
   (g) Discriminate puts () and gets().
   (h) What do you mean by operator precedence?
   (i) Differentiate between Global variable & extern variable.
   (j) What are the different file operations?

SECTION – B

2. **Attempt *any three* of the following :** **(3 ×10 = 30)**
   (a) What is an operator? Explain the arithmetic, relational, logical, and assignment operators in C language.
   (b) What is Recursion? Write a program in C to calculate the factorial of given number using recursion.
   (c) What is string? Write a C program that reads a sentence and prints the frequency of each of the vowels and total count of consonants?
   (d) What is meant by the storage class of a variable? Name the four storage class specifications included in C. Discuss with example.
   (e) What is Dynamic memory Allocation? Explain *malloc() and* calloc() function with suitable example.

SECTION – C

3. **Attempt *any one* part of the following :** **(1 ×10= 10)**
   (a) What are basic data types available in "C"? Write the significance of each data type.
   (b) Design an algorithm and draw a corresponding flow chart to convert a decimal number to its binary equivalent.

4. **Attempt *any one* part of the following :** **(1 ×10= 10)**
   (a) List the differences between while loop and do – while loop. Write a C program to find sum of Natural Numbers from 1 to N