Question Bank
Problem Solving Using C
Paper Code: 21CAC-1/AI

Unit 1

MCQ QUESTIONS:

1. Select the correct answer for the question.
1) Who invented C Language.?
A) Charles Babbage
B) Grahambel
C) Dennis Ritchie
D) Steve Jobs

2. ____________ it is a step-by-step representation of a solution to a given problem
A) algorithm
B) flowchart
C) Pseudocode
D) None of the above

3. The ________ provides pictorial representation of given problem.
A. Algorithm
B. Flowchart
C. Pseudocode
D. All of these

4. Limitation of flowchart ___________
A. Complex
B. Costly
C. Difficult to modify
D. all of the above

5. Find a correct C Keyword below.
A) work
B) case
C) constant
D) permanent

6. Every C program must have only one _________function.
A. **main()**
B. User defined
C. Built in
D. none of the above

7. Every statement in this Section are terminated by a _________
A. Dot (.)
B. colon (:)
C. semicolon(;)
D. Question mark (?)

8. The Sequence of the Instruction written to perform specific task is called the__________.
A. Statement
B. Program
C. Algorithm
D. None of the above

9. Types of Integers are.?
A) short
B) int
C) long
D) All the above

10. Size of float, double and long double in Bytes are.?
A) 4, 8, 16
B) 4, 8, 10
C) 2, 4, 6
D) 4, 6, 8

11. Keywords are also known as_________.
A. Keypads
B. reserve words
C. Characters
D. words

12. C supports a set of ___________keywords
A. 32
B. 18
C. 34
D.64

13. An Identifier may contain?
A) Letters a-z, A-Z in Basic character set. Unicode alphabet characters other languages
B) Underscore _ symbol
C) Numbers 0 to 9 Unicode Numbers in other languages
D) All the above

14. An Identifier can start with?
A) Alphabet
B) Underscore (_) sign
C) Any character that can be typed on a keyboard

D) Option A & Option B

15. Choose correct statements
A) A constant value does not change. A variable value can change according to needs.
B) A constant can change its values. A variable can have one constant value only.
C) There is no restriction on number of values for constants or variables.
D) Constants and Variables can not be used in a singe main function.

16. Find an integer constant.
A) 3.145
B) 34
C) "125"
D) None of the above

17. A Variable of a particular type can hold only a constant of the same type. Choose right answer.
A) TRUE
B) FALSE
C) It depends on the place the variable is declared.
D) None of the above.

18. Each statement in a C program should end with?
A) Semicolon ;
B) Colon :
C) Period . (dot symbol)
D) None of the above.

19. ______ is a procedure or step by step process for solving a problem.
A. Algorithm
B. Flowchart
C. Pseudocode
D. All of these

20. The _______ symbol is used to represent decision in flowchart.
A. Circle
B. Rectangle
C. Diamond
D. None of these

21. Which of the following is not a valid C variable name?
a) int number;
b) float rate;
c) int variable_count;
d) int $main;

22. A ___________ is a data name that may be used to store a data value

A. Keyword
B. Constant
C. Variable
D. Data type

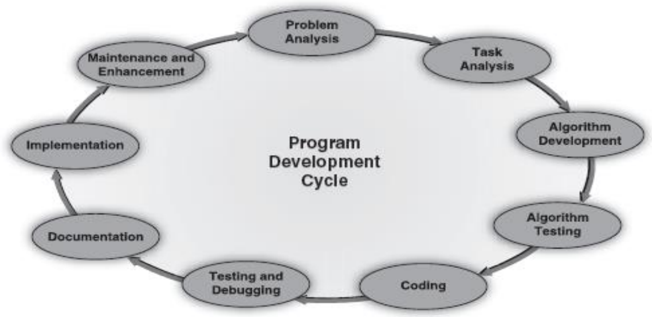
23. Symbolic constant can be defined using ___________
A. #define statement.
B. #include
C. #definition
D. #Declaration

24. A single character can be defined as a character data type using the keyword _________
A. String
B. char
C. character
D. int

25. Data types are derived from the fundamental data types are called as ____________
A. Derived data types
B. Detailed data types.
C. Defined data types
D. None of the above

Descriptive Questions:

1) Explain any two phases of Program Development Cycle with diagram



Problem Analysis: The problem is analysed precisely (exactly) and completely. Based on understanding, the developer knows about the scope within which the problem needs to be developed.
Task Analysis: After analysing the problem, the developer needs to develop various solutions to solve the given problem.
From these solutions, the optimum solution is chosen, which can solve the problem comfortably and economically.

2) Define algorithm and give the advantages and disadvantages of algorithm.
An *algorithm* is defined as a finite sequence of explicit instructions that when provided with a set of input values produces an output and then terminates
Advantages:

- it is a step-by-step representation of a solution to a given problem ,which is very easy to understand
- it has got a definite (clearly stated or decided) procedure.
- it easy to first develop an algorithm, then convert it into a flowchart &then into a computer program.
- it is independent of programming language.
- it is easy to debug as every step is got its own logical sequence

Disadvantages:

- It is time consuming & cumbersome (complicated) as an algorithm is developed first which is converted into flowchart &then into computer program

3) Write an algorithm check for palindrome

```
Step 1:start
Step 2:read a number num
Step 3 :set  rev=0,rem=0
Step 4:while num>0 true continue else goto step 8
Step 5 :set rem=num%10
Step 6:set rev=rev*10+rem
step 7:set num=num/10 go to step 4
Step 8:print rev
Step 9:stop
```

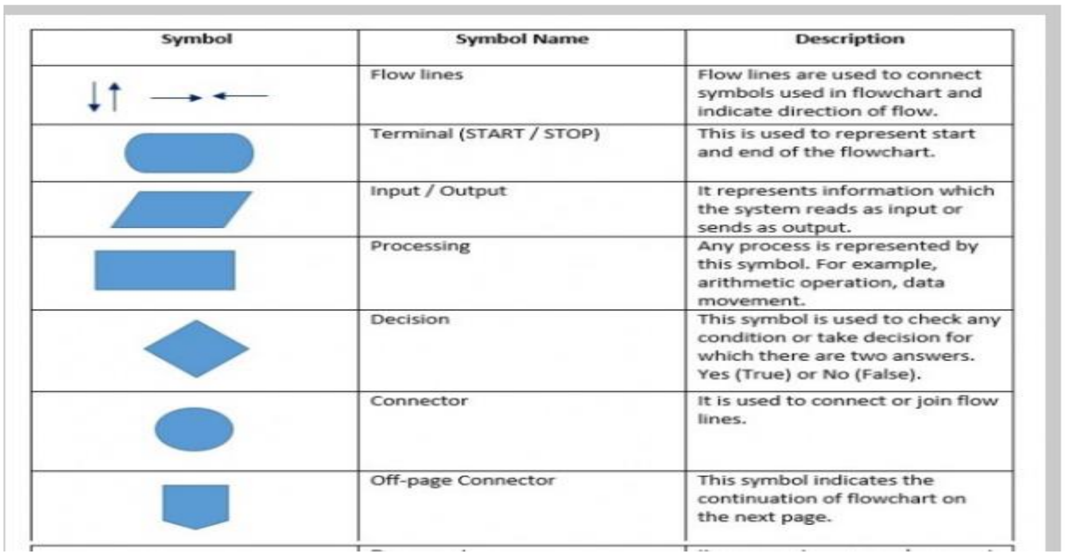
4) **Write algorithm to find largest among three number.**

```
Step 1:start
Step 2:read a,b,c
Step 3:if(a>b)and(a>c)then print a goto step 6  else goto step4
Step 4:if(b>c) then print b and goto step 6
else goto step 5
Step 5:print c
Step 6:stop
```

5) Define flowchart and explain any 6 symbols using in flowchart.

Graphical representation of any program is called flowchart.

There are some standard graphics that are used in flowchart as following

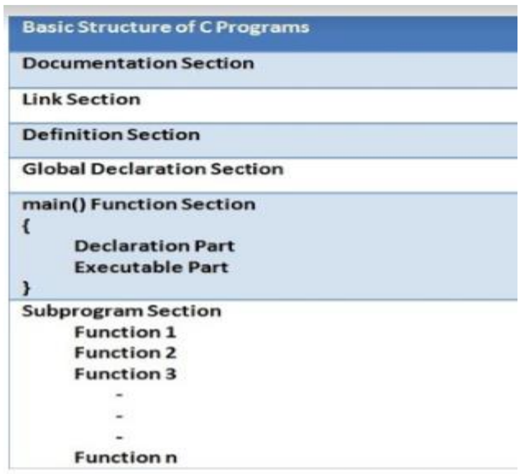


Symbol	Symbol Name	Description
↓↑ → ←	Flow lines	Flow lines are used to connect symbols used in flowchart and indicate direction of flow.
⬭	Terminal (START / STOP)	This is used to represent start and end of the flowchart.
▱	Input / Output	It represents information which the system reads as input or sends as output.
▭	Processing	Any process is represented by this symbol. For example, arithmetic operation, data movement.
◆	Decision	This symbol is used to check any condition or take decision for which there are two answers. Yes (True) or No (False).
●	Connector	It is used to connect or join flow lines.
⬟	Off-page Connector	This symbol indicates the continuation of flowchart on the next page.

6) List and explain features of C

- It is robust.
- C has the advantage of assembly level programming such as bit manipulation and all the significant features of high level language such as easy debugging, compactness etc. Therefore most of the C compilers are written in C.
- C is also called as a middle level language since it combines the features of high level as well as low level programming.
- It is highly suited for writing system software and application packages.
- C consists of a rich variety of data types and powerful operators. This makes C programs much more efficient and fast.
- It is platform independent and highly portable i.e., C can be run on almost any operating system.
- C is a structured language, wherein the program is subdivided into a number of modules. Each of these modules performs a specific task. Further structured programming helps in making program debugging, testing and maintenance, easier.
- One of the salient feature of C is its ability to add on to its library. User-defined functions can be added to the C library making the programs simpler.
- C provides manipulation of internal processor registers.
- It allows pointer arithmetic and pointer manipulation.
- Expressions can be represented in compact form using C.

7) Give Structure of C program and Explain each section.



Basic Structure of C Programs
Documentation Section
Link Section
Definition Section
Global Declaration Section

main() Function Section
{
Declaration Part
Executable Part
}

Subprogram Section Function 1 Function 2 Function 3 - - - Function n

Documentation Section

This section consists of comment lines which include the name of programmer, the author and other details like time and date of writing the program. Documentation section helps anyone to get an overview of the program.

Link Section

The link section consists of the header files of the functions that are used in the program. It provides instructions to the compiler to link functions from the system library.

Definition Section

All the symbolic constants are written in definition section. Macros are known as symbolic constants.

Global Declaration Section

The global variables that can be used anywhere in the program are declared in global declaration section. This section also declares the user defined functions.

main() Function Section

It is necessary have one main() function section in every C program. This section contains two parts, declaration and executable part. The declaration part declares all the variables that are used in executable part. These two parts must be written in between the opening and closing braces. Each statement in the declaration and executable part must end with a semicolon (;). The execution of program starts at opening braces and ends at closing braces.

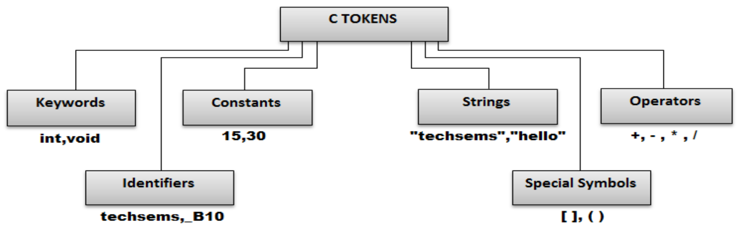
Subprogram Section

The subprogram section contains all the user defined functions that are used to perform a specific task. These user defined functions are called in the main() function.

8) Write a note on Tokens in 'C'

The smallest individual elements or units in a program are called as Tokens. C has following tokens.

Identifiers
Keywords
Constants
Operators
Special characters



Identifiers :

Identifiers refer to the name given to the programming elements such as variables, functions, arrays,etc.

Eg:- *page_nograde , mark1 , Mark*

Constants refer to fixed values that the program may not alter during its execution.
Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are enumeration constants as well.
Constants are treated just like regular variables except that their values cannot be modified after their definition.

Variables

Variables are identifiers whose value can change during execution of the Program. Variables used to store values. A data type is associated with each variable & it decides what values the variable can take. Rules for declaring variable are same as that of identifier.
In C, a variable must be declared before it can be used. Variables are declared in the declaration section of function.

Examples of legal variable names include:

x	result	outfilebestyet
x1	x2	out_filebest_yet

9) Write a note on Constants in 'C'

Constants refer to fixed values that the program may not alter during its execution.
Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are enumeration constants as well.
Constants are treated just like regular variables except that their values cannot be modified after their definition.

Integer Constants

It's referring to a sequence of digits. Integers are of three types namely

1. Decimal Integer
2. Octal Integer
3. Hexadecimal Integer

Example:

Decimal Constants : 15, -265, 0, 99818, +25,
Octal Constants : 045 , 077
HexaDecimal Constants : 0X6 , 0X88

Real constant

The numbers containing fractional parts like 99.25 are called real or floating points constant.

The Real or Floating-point constants can be written in **two** forms:

1. **Fractional** or **Normal** form
2. **Exponential or Scientific** form

Express a Real constant in fractional form
A real constant consistsyy for a series of digits representing the whole part of the number, followed by a decimal pointy, followed by a series of representing the fractional part.
Valid Real constants (yyFractional): 0.0 -0.1 +123.456 .2 2.
Invalid Real constant: - 1 - a decimal point is missing
1, 2.3 - Illegal character (.)

Single Character Constantsy
It simply contains a single character enclosed within ' and ' (a pair of single quote). It is to be noted that the character '8' is not the same as 8. Character constants have a specific set of integer values known as ASCII values (American Standard Code for Information Interchange).
Example:
'X', '5', ';'

String Constants
These are a sequence of characters enclosed in double quotes, and they may include letters, digits, special characters, and blank spaces. It is again to be noted that "G" and 'G' are different - because "G" represents a string as it is enclosed within a pair of double quotes whereas 'G' represents a single character.
Example:
"Hello!", "2015", "2+1"

Backslash character constant

- There are some characters which have special meaning in C language.
- They should be preceded by backslash symbol to make use of special function of them.

Given below is the list of special characters and their purpose
Ex:

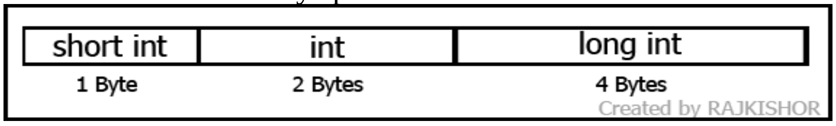
Backslash character	**Meaning**
\b	Backspace
\f	Form feed

10) Explain in brief classification of a) integer types and b) floating point types

Integer Data Type:

- Integers are whole numbers with a range of values. Integers occupy one word of storage.
- In a 16-bit machine, the range of integers is -32768 to+32767
- In order to provide some control over a range of numbers and storage space, C has 3 classes of storage for integers namely, short int, int and long int in both signed and unsigned format
- Short int is used for small range of values and requires half the amount of storage as a regular int number uses.
- Long int requires double the storage as an int value.

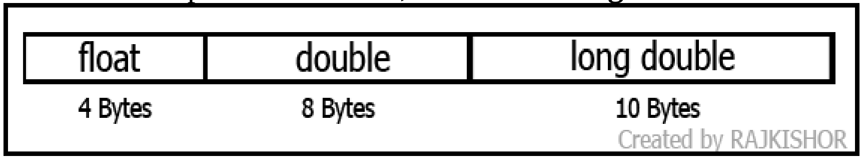
- In unsigned integers, all the bits in the storage are used to store the magnitude and are always positive.



short int	int	long int
1 Byte	2 Bytes	4 Bytes

Floating point DataType:

- All floating point numbers require 32 bits with 6 digits of precision
- They are defined in C using the keyword float
- For higher precision, double data type can be used.
- A **double type number uses 64 bits (8 bytes) giving a precision of 14 digits.**
- Range of values for double type is **1.7E-308 to 1.7E+308**
- To extend the precision further, we can use **long double which uses 80 bits.**



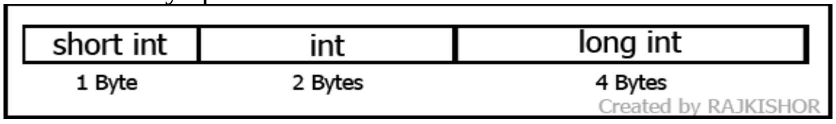
float	double	long double
4 Bytes	8 Bytes	10 Bytes

11) List and explain primary data types available in 'C'

- All C compilers support this data type. Primary data types can be classified as follows
  1. Integer
  2. Floating point
  3. Character

Integer Data Type:

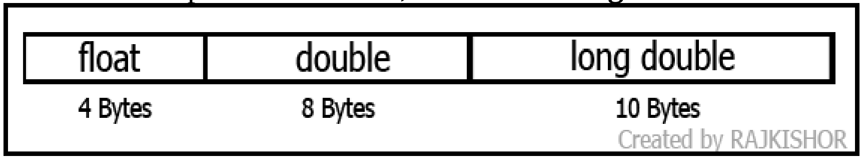
- Integers are whole numbers with a range of values. Integers occupy one word of storage.
- In a 16-bit machine, the range of integers is -32768 to+32767
- In order to provide some control over a range of numbers and storage space, C has 3 classes of storage for integers namely, short int, int and long int in both signed and unsigned format
- Short int is used for small range of values and requires half the amount of storage as a regular int number uses.
- Long int requires double the storage as an int value.
- In unsigned integers, all the bits in the storage are used to store the magnitude and are always positive.



short int	int	long int
1 Byte	2 Bytes	4 Bytes

Floating point DataType:

- All floating point numbers require 32 bits with 6 digits of precision
- They are defined in C using the keyword float
- For higher precision, double data type can be used.
- A **double type number uses 64 bits (8 bytes) giving a precision of 14 digits.**
- Range of values for double type is **1.7E-308 to 1.7E+308**
- To extend the precision further, we can use **long double which uses 80 bits.**



float	double	long double
4 Bytes	8 Bytes	10 Bytes

Character DataType:

- A single character can be defined as a character data type using the keyword char
- Characters usually need 8 bits for storage
- The qualifier **signed or unsigned can be explicitly applied to char.**
- While **unsigned characters have values between 0 and 255,**
- **signed characters have values from -128 to 127.**

12) Explain with examples declaring, initializing and assigning value to variable

Variables are identifiers whose value can change during execution of the Program. Variables used to store values. A data type is associated with each variable & it decides what values the variable can take. Rules for declaring variable are same as that of identifier.
In C, a variable must be declared before it can be used. Variables are declared in the declaration section of function.

Examples of legal variable names include:

x	result	outfilebestyet
x1	x2	out_filebest_yet

Declaration of Variable

Declaration of variable in c can be done using following syntax:

data_typevariable_name;
or
data_type variable1, variable2,…,variablen;

where *data_type* is any valid c data type and *variable_name* is any valid identifier.

For example,yyy

```
int a;
float variable;
float a, b;
```

Initialization of Variable

Assigning initial value to the variable is called variable initialization. C variables declared can be initialized with the help of assignment operator '='.

Syntax
data_typevariable_name=constant/literal/expression;
or
variable_name=constant/literal/expression;
Example

```
1	int a=10;
2	 int a=b+c;
3	 a=10;
4	 a=b+c;
```

Multiple variables can be initialized in a single statement by single value, for example,
a=b=c=d=e=10;

13) Explain with example #define statement in 'C. List rules apply to it.

The preprocessor #define is another more flexible (see Preprocessor Chapters) method to define constants in a program.

```
#define TRUE            1
#define   FALSE         0
#define NAME_SIZE       20
```

The const keyword is to declare a constant, as shown below:

```
int const a = 1;
const int a =2;
```

Symbolic Constant in C Language :

A symbolic constant is a name that substitutes for a sequence of characters. The characters may represent numeric constant, a character constant or a string constant. Thus, a symbolic constant allows a name to appear in place of a numeric constant, a character constant or a string.

Constants defined using #define pre-processor directive is called symbolic constant.

Syntax: #define symbolic name value

For example

#define PI 3.141593

#define TRUE 1

#define FALSE 0

#define PI 3.141593 defines a symbolic constant PI whose value is 3.141593. When the program is preprocessed, all occurrences of the symbolic constant PI are replaced with the replacement text 3.141593.

Note that the pre-processor statements begin with a #symbol, and are not end with a semicolon. By convention, preprocessor constants are written in UPPERCASE.

14) Explain the any 4 advantages of flowchart.

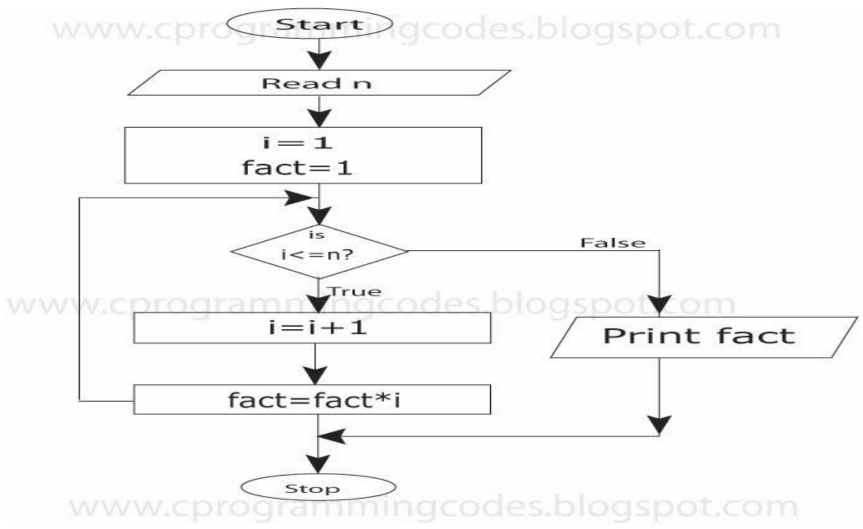
- **Makes Logic Clear:** The main advantage of using a flowchart to plan a task is that it provides a pictorial representation of the task, which makes the logic easier to follow. The symbols are connected in such a way that they show the movement (flow) of information through the system visibly. The steps and how each step is connected to the next can be clearly seen. Even less experienced personnel can trace the actions represented by a flowchart, that is, flowcharts are ideal for visualizing fundamental control structures employed in computer programming.
- **Communication:** Being a graphical representation of a problem-solving logic, flowcharts are a better way of communicating the logic of a system to all concerned. The diagrammatical representation of logic is easier to communicate to all the interested parties as compared to actual program code as the users may not be aware of all the programming techniques and jargons.
- **Effective Analysis:** With the help of a flowchart, the problem can be analysed in an effective way. This is because the analysing duties of the programmers can be delegated to other persons, who may or may not know the programming techniques, as they have a broad idea about the logic. Being outsiders, they often tend to test and analyse the logic in an unbiased manner.

- **Useful in Coding:** The flowcharts act as a guide or blueprint during the analysis and program development phase. Once the flowcharts are ready, the programmers can plan the coding process effectively as they know where to begin and where to end, making sure that no steps are omitted. As a result, error-free programs are developed in HLL and that too at a faster rate.
- **Proper Testing and Debugging:** By nature, a flowchart helps in detecting the errors in a program, as the developers know exactly what the logic should do. Developers can test various data for a process so that the program can handle every contingency.
- **Appropriate Documentation:** Flowcharts serve as a good program documentation tool. Since normally programs are developed for novice users, they can take the help of the program documentation to know what the program actually does and how to use the program.

15. Explain the Limitations of Flowcharts

- **Complex:** The major disadvantage in using flowcharts is that when a program is very large, the flowcharts may continue for many pages, making them hard to follow. Flowcharts tend to get large very quickly and it is difficult to follow the represented process. It is also very laborious to draw a flowchart for a large program. You can very well imagine the nightmare when a flowchart is to be developed for a program, consisting of thousands of statements.
- **Costly:** Drawing flowcharts are viable only if the problem-solving logic is straightforward and not very lengthy. However, if flowcharts are to be drawn for a huge application, the time and cost factor of program development may get out of proportion, making it a costly affair.
- **Difficult to Modify:** Due to its symbolic nature, any changes or modification to a flowchart usually requires redrawing the entire logic again, and redrawing a complex flowchart is not a simple task. It is not easy to draw thousands of flow lines and symbols along with proper spacing, especially for a large complex program.
- **No Update:** Usually programs are updated regularly. However, the corresponding update of flowcharts may not take place, especially in the case of large programs. As a result, the logic used in the flowchart may not match with the actual program's logic. This inconsistency in flowchart update defeats the main purpose of the flowcharts, that is, to give the users the basic idea about the program's logic.

16. **Draw the flowchart to find factorial of n number**



UNIT 2

Multiple choice questions

1. Among unary operation which operator represents increment?
A) --
B) ++
C) –
D) +

2.When one of the operands is real and the other is an integer, the expression is called ___________ arithmetic expression.
A) mixed mode
B) yreal mode
C) integer mode
D) expression mode

3.Comparisons can be done using_____________.
A) Arithmetic operator
B) Logical operators
C) Conditional operator
D) relational operators

4.____________are used to assign the result of an expression to variable
A) Assignment operators
B) Conditional operator
C) Arithmetic operator
D) None of the above

5. Bitwise operators are used to manipulate data at __________
A) byte level
B) yyybit level
C) number level

D) bitwise level

6. The function scanf is used to ___
A) To take logical decisions
B) Input a set of values
C) Print a set of values
D) Do mathematical manipulations

7. The _________ function reads the character from the keyboard.
A) getchar()
B) putchar()
C) scan()
D) none of the above

8. The ___________ function is used for formatted
A) prints()
B) put()
C) printf()
D) scanf()

9. Operator % in C Language is called.?
A) Percentage Operator
B) Quotient Operator
C) Modulus
D) Division

10. Choose a right statement.
int a = 10 + 4.867;
A) a = 10
B) a = 14.867
C) a = 14
D) compiler error.

11. What is the priority of operators *, / and % in C language.?
A) * > / > %
B) % > * > /
C) Both % = / , * are same
D) All three operators *, / and % are same.

12. Choose a C Conditional Operator from the list.
A) ?:
B) :?
C) :<
D) <:

13. What is the other name for C Language ?: Question Mark Colon Operator.?
A) Comparison Operator
B) If-Else Operator
C) Binary Operator
D) Ternary Operator

14. Choose a syntax for C Ternary Operator from the list.
A) condition ? expression1 : expression2
B) condition : expression1 ? expression2
C) condition ? expression1 < expression2
D) condition < expression1 ? expression2

15. which of the following operators takes only integer operands?
A) +
B) *
C) /
D) %

16. What is the output of the C statement.?

```
int main()
{
   int a=0;
   a = 5<2 ? 4 : 3;
   printf("%d",a);
   return 0;
}
```

A) 4
B) 3
C) 5
D) 2

17. If you have to make decision based on multiple choices, which of the following is best suited?
a) if
b) if-else
c) if-else-if
d) All of the above

18. Which of the following cannot be checked in a switch - case statement?
A) Character
B) Integer
C) Float
D) enum

19. Which of the following is branching statement of C language?
A) if statement
B) if…else statement
C) switch statement
D) All of these

20. If the Boolean expression of if statement evaluates to ________, then the block of code inside the if statement will be executed.
A. True
B. False

C. True and False
D. None of these

21. What will be the output of the following C code?
```
#include <stdio.h>
   void main()
   {
      int x = 5;
      if (x < 1)
         printf("hello");
      if (x == 5)
         printf("hi");
      else
         printf("no");
   }
```
A)hi
b) hello
c) no
d) error

22. What is the output of C Program with switch statement.?
```
int main()
{
   int a=3;

   switch(a)
   {
      case 2: printf("ZERO "); break;
      case default: printf("RABBIT ");
   }
   }
```
A) RABBIT
B) ZERO RABBIT
C) No output
D) Compiler error

23. The _________statement terminates the loop immediately when it is encountered.
A) break
B) goto
C) continue
D) none

24. The ___________statement skips some statements inside the loop and goes back to beginning of loop for executing next iteration.
A. if
B. continue
C. goto
D. break

25. A loop becomes an infinite loop if a condition never becomes_______

A. false
B. true
C. either true or false
D. none of the above

Descriptive questions

1. List and explain arithmetic, logical and relational operators available in C

ArithmeticOperators:

All the basic arithmetic operators are supported by C. These operators can manipulate with any built in data types supported by C. The various operators are tabulated as follows:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Division

Comparisons can be done using relational operators. A relational expression contains a combination of arithmetic expressions, variables or constants along with relational operators. A relational expression can contain only two values i.e. true or false. When the expression is evaluated as true then the compiler assigns a non zero(1) value and 0 otherwise. These expressions are used in decision statements to decide the course of action of a running program.
Syntax: ae1 relational operator ae2
where ae1 and ae2 are arithmetic expressions

Operators	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
y>=	Greater than or equal to
==	Equal to
!=	Not equal to

Logical Operators:

An expression that combines two or more relational expressions is called a logical expression and the operators used to combine them are called logical operators.
Syntax: R1 operator R2
where R1 and R2 are relational expressions

Operator	Meaning
&&	Logical And
||	Logical Or
!	Logical Not

Example :

if(agey > 55 && salary < 1000)

if(number < 0 || number > 100)

2) List and explain a) conditional operator, b) increment & decrement operators available in C

Conditional Operator:

Conditional operators are also called ternary operators. Conditional expressions can be constructed in C using the operator pair '?:'.

Syntax: expr?expr1:expr2;

Here expr is evaluated first. If the value is true, then expr1 is evaluated and becomes the value of the expression. If the expression is false then, expr2 iyyyys evaluated and becomes the value of the expression.

E.g. a=5;b=10;
    x=(a>b)?a:b;

The output of the above example is as follows- The value of b i.e. 10 is assigned to x.

Increment/Decrement Operators:

C has 2 very powerful operators that are not found in any other language. They are increment/decrement operators i.e. ++ and --. The ++ operator is used to increment value of a variable by 1. The -- operator is used to decrement value of a variable by 1.

Both are unary operators and can be written as follows: ++m, m++ , m—and –m. Both m++ and ++m mean the same thing when they form statements independently. But, they behave differently when used in expression on the right hand side of assignment operator.

E.g. Let a=5; x=a++; Here, this statement can be broken into 3 statements
    as follows a=5; x=a; a=a+1;

In the above example, the value of a is assigned to x and then its value is incremented by 1**. A prefix operator first adds 1 to the operand and then the result is assigned to the variable on the left. A postfix operator first assigns its value to the variable and then increments its value by 1.**

E.g. m=10;y
    y=--m

Here the value of m is decremented by 1 and then assigned to y. Hence the value of y is 9.

3) Write a note on bitwise operators available in C

These operators are used to manipulate data at bit level. These operators are used for testing the bits or shifting the bits either to the left or right.

Operator	Meaning
&	Bitwise And
|	Bitwise Or
^	Bitwise exclusive Or
<<	Shift left
>>	Shift Right
~	One's complement

4) Explain with example evaluation of arithmetic expression in C

Expressions are always evaluated from left to right, using the rules of precedence of operators, if parenthesis are missingy

High Precedence- * / %

Low Precedence - + -

Basically the evaluation procedure involves 2 left to right passes. During the first pass the high priority operators are applied as they are encountered and during the second pass, lower priority operators are applied as they are encountered.

Example: Consider the following expression

x= a-b/3+c*2-1 If the values of a=9, b=12,c=3

Now x= 9-$\underline{12/3}$+$\underline{3*2}$-1

Pass 1: x=9-4+6-1

Pass 2: x=5+6-1=10

Example 2:9-12/(3+3)(2-1)*

Whenever parentheses are used, the expressions within parentheses assume highest priority. If two or more sets of parentheses appear one after another as shown above, the expression contained in the left-most set is evaluated first and the right-most in the last. Given below are the new steps.

First pass

Step 1: 9-12/6 * (2-1)

Step 2 : 9-12/6 * 1

Second pass

Step 3 : 9-2* 1

Step 4 : 9-2

Third pass

Step 5 :7

5) Explain precedence of arithmetic operators with the help of an example expression

If more than one operators are involved in an expression then, C language has predefined rule of priority of operators. This rule of priority of operators is called operator precedence.

In C, precedence of arithmetic operators(*,%,/,+,-) is higher than relational

operators(==,!=,>,<,>=,<=) and precedence of relational operator is higher than logical operators(&&, || and !).
Suppose an expression:
(a>b+c&&d)
This expression is equivalent to ((a>(b+c))&&d)
i.e. (b+c) executes first then,
(a>(b+c)) executesthen, (a> (b=c)) && d) executes

6) Explain the concept of type conversion in C, Give examples

Type casting is a way to convert a variable from one data **type** to another data **type**.

There are two types of the type conversions:

- Implicit Type Conversion
- Explicit Type Conversion

Implicit Type Conversion :

- When data value automatically convert from one type to another type is called the Implicit type conversion.
- If the operands are of different types, the lower type is automatically converted to the higher type before the operation proceeds.

1.float to int causes truncation of the fractional part.
2.double to float cause rouding of digits.
3.longint to int cause dropping of the excess higher order bits.
Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
float c;
a= 12,b=13;
c= a+b;    //automatically conversion   done
printf("%f",c);
getch();
}
```

7) Explain in brief implicit and explicit type conversion with example

Explicit Type Conversion:

When a process of the conversion done manually of locally then this process is called the explicit type conversion process or casting operation.

Syntax :(type_name) expression

example:

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

```
{
floata,b;
int c;
clrscr();
 a=12.3;
b=13.3;
c= (int) (a+b);
printf("%f",c);
getch();
}
/* Output 25 */
```

8)Explain with example formatted Input in C

scanf() function

Syntax :

scanf("format string", arg1,arg2…argn);

- This function is usually used as an input statement.
- The format string must be a text enclosed in double quotes. It contains type of data to input. Example: integer (%d) , float (%f) , character (%c) or string (%s).
- The arg1,arg2..argn contains a list of variables each preceded by & (to get address of variable) and separated by comma.

Examples :-

```
int i, d ;
char c ;
float f ;
scanf( "%d", &i ) ;  /* input integer data*/
scanf( "%d %c %f", &d, &c, &f ) ; /* input int , char and float */
```

The & character is the *address of* operandin C, it returns the address in memory of the variable it acts on.

9) Explain with example formatted Output in C
printf() function

Syntax :
printf("format string", v1,v2…vn);

The printf() function is used for formatted output and uses a format string which is made up of a series of format specifiers to govern how it prints out the values of the variables or constants required. The more common format specifiers are given below

%c character	%f floating point
%d integer	%lf double floating point
%i integer	%e exponential notation
%u unsigned integer	%s string
%ld signed long	%x hexadecimal
%lu unsigned long	%o octal

```
#include<stdio.h>
void main()
{
  int i;
printf("Please enter a value...");
scanf("%d", &i);
printf( "\nYou entered: %d", i);
}
```

When you will compile the above code, it will ask you to enter a value. When you will enter the value, it will display the value you have entered on screen.

10) Explain unformatted input/ output in C.

The getchar() and putchar() Functions [Unformatted I/O]

The **getchar()** function reads the character from the keyboard. This function reads only single character at a time. To continuously read the characters use getchar() within looping statement.

Syntax : c=getchar() where c is character variable.

The **putchar(int c)** function printsthe given character on the screen . This function puts only single character at a time. You can use this method in the loop in case you want to display more than one character.

```
#include <stdio.h>
main( )
{
   int c;
printf( "Enter a value :");
   c = getchar( );
printf( "\nYou entered: ");
putchar(c);
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads only a single character and displays it

11) Explain simple if statement and else..if with syntax and example.

Simple if Statement

The basic format of if statement is:

```
if(test_expression)
{
    statement 1;
    statement 2;
...
}
```

Here if the test expression is evaluated to true, the statement block will get executed, or it will get skipped.

Example :

```
#include<stdio.h>
main()
{
  int a = 15, b = 20;
  if (b > a) {
  printf("b is greater");
  }
}
```

If ..else Statement

The syntax of an **if...else** statement in C programming language is –

```
if(test expression)

{
   Statement block-1
}
 else
{
   Statement block-2
}
```

If the test expression evaluates to true, then the if block will be executed, otherwise, the else block will be executed.

Example:

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

12) Explain nested if statement with the help of a program example

A nested if is an if statement that is inside another if statement .

Syntax:

```
if (condition1)

{

/* Executes when condition1 is true*/

  if (condition2)

  {

/* Executes when condition2 is true*/

  }

}
```

Example

```
#include <stdio.h>
int main()
{
  int var1, var2;
printf("Input the value of var1:");
scanf("%d", &var1);
printf("Input the value of var2:");
scanf("%d",&var2);
  if (var1 != var2)
  {
          printf("var1 is not equal to var2\n");
          /*Nested if else*/
          if (var1 > var2)
          {
                    printf("var1 is greater than var2\n");
          }
          else
          {
                    printf("var2 is greater than var1\n");
          }
  }
  else
  {
          printf("var1 is equal to var2\n");
  }
  return 0;
}
```

13) Explain else..if ladder with the help of a program example
if-else-if ladder

Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with

that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.
Syntax:

```
if (condition-1)
{
   Statement block-1;
}
else if (condition-2)
{
   Statement block-2;
}
.
.
Else
{
Default block of  statement(s);
}
```

The if else ladder statement in C is used to test set of conditions in sequence. An if condition is tested only when all previous if conditions in if-else ladder is false. If any of the conditional expression evaluates to true, then it will execute the corresponding code block and exits whole if-else ladder.

- First of all condition1 is tested and if it is true then statement block-1 will be executed and control comes out of whole if else ladder.
- If condition1 is false then only condition2 is tested. Control will keep on flowing downward, If none of the conditional expression is true.
- The last else is the default block of code which will gets executed if none of the conditional expression is true.

14) Explain switch statement with syntax and a program example

Switch Statement

Syntax :

```
switch (variable or integer expression)
{
   case constant1:
    Statement(s) ;
break;
case constant2:
           Statement(s) ;
 break;
        .
        .
        .
   default:
default statement(s);
}
```

Switch statement is a control statement that allows us to choose only one choice among the many given choices. The expression in switch evaluates to return an integral value, which is then compared to the values present in different cases. It executes that block of code which matches the case value. If there is no match, then **default** block is executed(if present).

```
#include <stdio.h>
main()
{
   int x = 2;
   switch (x)
   {
       case 1:
printf("Choice is 1");
break;
       case 2:
printf("Choice is 2");
 break;
       case 3: printf("Choice is 3");
               break;
       default: printf("Choice other than 1, 2 and 3");
                break;
   }
   return 0;
}
```

15) Explain while statement with syntax and example

While Loop

The syntax of a while loop is:

```
while (testExpression)
{
  Statement block
}
```

where, test Expression checks the condition is true or false before each loop.The while loop evaluates the test expression. If the test expression is true (nonzero), codes inside the body of while loop are executed. The test expression is evaluated again. The process goes on until the test expression is false.When the test expression is false, the while loop is terminated. While loop is entry controlled loop.

Example

```
/* Programm to print numbers from 1 to 9*/:
            #include <stdio.h>
            main()
            {
              int i=1;

              while(i<10)
              {
                 printf("%d\n",i);
                 i++;
              }

            }
```

16) Explain do..while statement with syntax and example

do...while loop in C programming checks its condition at the bottom of the loop.

A do...while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax :

```
do {
  statement(s);

} while(condition );
```

if the condition is true, the flow of control jumps back up to do, and the statement(s) in the

example

```
 main ()
{
   int a = 10;
  /* do loop execution */
  do {
printf("  %d\n", a);
     a = a + 1;
}while( a < 20 );
 }
```

When the above code is compiled and executed, it prints values from 10 to 19

17) Explain for statement with syntax and example

for loop

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of a **for** loop in C programming language is −

```
for ( init_exp; test_exp; update_exp )
{
  statement(s);
}
```

Example

```
#include <stdio.h>

main ()
 {
int  i;
  /* for loop execution */
for( i = 1;i < 10;i++ ){
printf("%d\n",,i);
  }
}
```

When the above code is compiled and executed, it prints values from 1 to 9

18) Explain the following: (6)

i. Nested loops
ii. Infinite loops

i) **Nesting of Loops**
C programming allows to use one loop inside another loop. The following section shows a few examples to illustrate the concept.

The syntax for a **nested for loop** statement in C is as follows −

```
for ( init_exp; test_exp; update_exp )
 {
for(init_exp; test_exp; update_exp)
{
    statement(s);
  }
  statement(s);
}
```

The syntax for a **nested while loop** statement in C is as follows −

```
while(condition) {

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

ii) The Infinite Loop
A loop becomes an infinite loop if a condition never becomes false. Loop will not terminate

Example

```
int i = 1
while(i<10)
{
printf("%d\n", i);
}
```

Here we are not updating the value of i. So after each iteration value of i remains same. As a result, the condition (i<10) will always be true. For the loop to work correctly add i++;

UNIT-3

Multiple Choice Questions:

1. To perform a set of instructions repeatedly which of the following can be used?
A.for
B.while
C.if-else-if
D.both i & ii

2.
```
int main() {
int i = 1 ;
while( i<=2)
   printf("%d",i);
return 0;
   }
```
What is output of above code

A.12
B.i
C.Compilation error
D.indefinite loop

3. When __________ is encountered inside any loop, Control automatically passes to the first statement after loop.
A.Continue
B.break
C.goto
D.return

4. This Loop tests the condition after having executed the Statements within the Loop.
A.while
B.do-while
C.for Loop
D.if-else-if

Choose a right C Statement.
A) Loops or Repetition block executes a group of statements repeatedly.
B) Loop is usually executed as long as a condition is met.
C) Loops usually take advantage of Loop Counter
D**) All the above.**

5. What is the output of C Program.?
```
int main()
{
    while(true)
    {
        printf("RABBIT");
        break;
    }

    return 0;
}
```
A) RABBIT
B) RABBIT is printed unlimited number of times.
C) No output
D) Compiler error.

6. What is the way to suddenly come out of or Quit any Loop in C Language.?
A) continue; statement
B) break; statement
C) leave; statement
D) quit; statement

7. Choose facts about continue; statement is C Language.
A) continue; is used to take the execution control to next iteration or sequence
B) continue; statement causes the statements below it to skip for execution
C) continue; is usually accompanied by IF statement.
D) All the above.

8. Choose a correct statement about C break; statement.?
A) break; statement can be used inside switch block
B) break; statement can be used with loops like for, while and do while.
C) break; statement causes only the same or inner loop where break; is present to quit suddenly.
D) All the above.

9. Choose a correct C Statement regarding for loop.
for(; ;);
A) for loop works exactly first time
B) for loop works infinite number of times
C) Compiler error
D) None of the above

10. What is an Array in C language.?
A) A group of elements of same data type.
B) An array contains more than one element
C) Array elements are stored in memory in continuous or contiguous locations.
D) All the above.

11. Choose a correct statement about C language arrays.
A) An array address is the address of first element of array itself.
B) An array size must be declared if not initialized immediately.
C) Array size is the sum of sizes of all elements of the array.
D) All the above

12. An array Index starts with.?
A) -1
B) 0
C) 1
D) 2

13. What is an array Base Address in C language.?
A) Base address is the address of 0th index element.
B) An array b[] base address is &b[0]
C) An array b[] base address can be printed with printf("%d", b);
D) All the above

14. What is the size of an array in the below C program statement.?
```
int main()
{
   int ary[9];
   return 0;
}
```
A) 8
B) 9
C) 10
D) None of the above

15. What is the minimum and maximum Indexes of this below array.?
```
int main()
{
   int ary[9];
   return 0;
}
```
A) -1, 8
B) 0, 8
C) 1,9
D) None of the above

16. What is the dimension of the C array int ary[10][5].?
A) 1
B) 2
C) 5
D) 10

17. What is the dimension of the below C Array.?

```
int ary[]={1,3,5,7};
```

A) 1
B) 2
C) 3
D) 5

18. Array of Arrays is also called.?
A) Multi Data Array
B) Multi Size Array
C) Multi Dimensional Array
D) Multi Byte Array

19. What is a String in C Language.?
A) String is a new Data Type in C
B) String is an array of Characters with null character as the last element of array.
C) String is an array of Characters with null character as the first element of array
D) String is an array of Integers with 0 as the last element of array.

20. Choose a correct statement about C String.

```
char ary[]="Hello..!";
```

A) Character array, ary is a string.
B) ary has no Null character at the end
C) String size is not mentioned
D) String can not contain special characters.
What is the Format specifier used to print a String or Character array in C Printf or Scanf function.?
A) %c
B) %C
C) %s
D) %w

21. What is the output of C Program with Strings.?

```
int main()
{
   char str[]={'g','l','o','b','e'};
   printf("%s",str);
   return 0;
}
```

A) g
B) globe
C) globe\0
D) None of the above

22. How do you accept a Multi Word Input in C Language.?
A) SCANF

B) GETS
C) GETC
D) FINDS

23. A character constant is enclosed by.?
A) Left Single Quotes
B) Right Single Quotes
C) Double Quotes
D) None of the above

24. A C string elements are always stored in.?
A) Random memory locations
B) Alternate memory locations
C) Sequential memory locations
D) None of the above

25. Choose a correct C statement about String functions.?
A) int n=strlen("abc") returns 3.
B) strupr("abc") returns ABC
C) strlwr("Abc") returns abc
D) All the above

26. What is the output of C program.?
```
int main()
{
   char str1[]="JAMES,";
   char str2[15]="BOND ";
   strcat(str2,str1);
   printf("%s",str2);
   printf("%s",str1);
}
```
A) JAMES BOND,JAMES,
B) JAMES,JAMES,
C) **BOND JAMES,JAMES,**
D) None of the above

27. Choose a correct C statement about String functions.?
A) toupper('a') returns A
B) tolower('D') returns d.
C) strcmp("123","12345") returns a negative number
D) All the above

28. What is the output of C program.?
```
int main()
{
   printf("%c","HUMPTY"[2]);
```

}
A) U
B) M
C) HUMPTY
D) None of the above

29. What is the output of C program with String arrays.?
```
int main()
{
   char code[3][4]={"IN","USA","K"};
   printf("%s", code[1]);
   return 0;
}
```
A) IN
B) USA
C) K
D) Compiler error

Descriptive Questions:

1. Explain while statement with syntax and example
   The syntax of a while loop is:
   ```
   while (testExpression)
   {
     Statement block
   }
   ```
   where, test Expression checks the condition is true or false before each loop.
   The while loop evaluates the test expression.If the test expression is true (nonzero), codes inside the body of while loop are executed. The test expression is evaluated again. The process goes on until the test expression is false. When the test expression is false, the while loop is terminated. While loop is entry controlled loop.
   Example
   ```
   /* Programm to print numbers from 1 to 9*/:
   #include <stdio.h>
   main()
   {
     int i=1;

     while(i<10)
     {
        printf("%d\n",i);
        i++;
     }

   }
   ```
2. Explain do..while statement with syntax and example
   do...while loop in C programming checks its condition at the bottom of the loop.

A do...while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax :

```
do {
       statement(s);

     } while (condition );
```

if the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

Example

```
 main ()
{
    int a = 10;
   /* do loop execution */
   do {
      printf("  %d\n", a);
      a = a + 1;
   }while( a < 20 );
 }
```

When the above code is compiled and executed, it prints values from 10 to 19

3. Explain for statement with syntax and example

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of a for loop in C programming language is −

```
for ( init_exp; test_exp; update_exp )
{
   statement(s);
}
```

- The init_exp step is executed first, and only once. This step allows you to initialize loop control variables.
- Next, the test_exp is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the update_exp This statement allows you to update any loop control variable
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself. After the condition becomes false, the 'for' loop terminates.

Example

```
#include <stdio.h>
main ()
 {
   int  i;
   /* for loop execution */
```

```
  for( i = 1;i < 10;i++ ){
    printf("%d\n",,i);
  }
}
```

When the above code is compiled and executed, it prints values from 1 to 9

4. Explain the following:

i. Nested loops

C programming allows to use one loop inside another loop. The following section shows a few examples to illustrate the concept.

The syntax for a nested for loop statement in C is as follows −

```
for ( init_exp; test_exp; update_exp )
 {

  for( init_exp; test_exp; update_exp)
{
    statement(s);
  }
  statement(s);
}
```

The syntax for a nested while loop statement in C is as follows −

```
while(condition) {

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

The syntax for a nested do...while loop statement in C is as follows −

```
do {
  statement(s);

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

}while( condition );
```

Example

C program to print the number pattern.

```
1
```

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

```
#include <stdio.h>
main()
{
   int i=1,j;
   while (i <= 5)
   {
      j=1;
      while (j <= i )
      {
         printf("%d ",j);
         j++;
      }
      printf("\n");
      i++;
   }
   }
```

ii. Infinite loops

A loop becomes an infinite loop if a condition never becomes false. Loop will not terminate

Example

```
int i = 1
while(i<10)
{
   printf("%d\n", i);
}
```

Here we are not updating the value of i. So after each iteration value of i remains same. As a result, the condition (i<10) will always be true. For the loop to work correctly add i++;

5. Differentiate break and continue statements

The break statement terminates the loop (for, while and do...while loop) immediately when it is encountered. The break statement is used with decision making statement such as if...else.

Syntax of break statement

```
break;
```

How break statement works?

Example

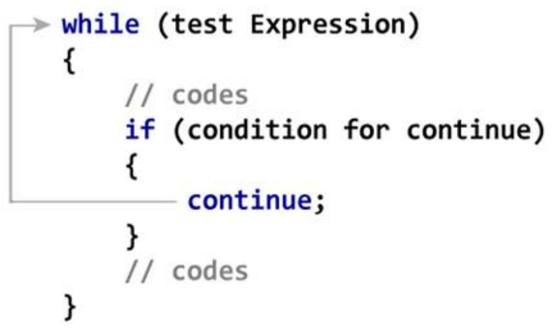
```
 main ()
{

    int a = 1 ;
   /* while loop execution */
   while( a < 10 )
   {
     printf("value of a: %d\n", a);
     a++;
     if( a >  5)
       {

         /* terminate the loop using break statement */
         break;
      }
   }


}
```

The continue statement skips some statements inside the loop and goes back to beginning of loop for executing next iteration.. The continue statement is used with decision making statement such as if...else.

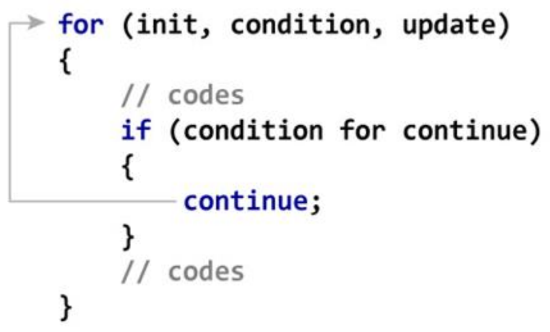
Syntax of continue Statement

```
continue;
```

How continue statement works?



```
while (test Expression)
{
    // codes
    if (condition for continue)
    {
        continue;
    }
    // codes
}
```



```
for (init, condition, update)
{
    // codes
    if (condition for continue)
    {
        continue;
    }
    // codes
}
```

Example

```
#include <stdio.h>
  main ()
{
  int a = 1 ;
   /* do loop execution */
   do {
         if( a ==  5)
       {
         a ++;
         continue;
      }
      printf("value of a: %d\n", a);
      a++;
      } while( a < 10 );
}
```

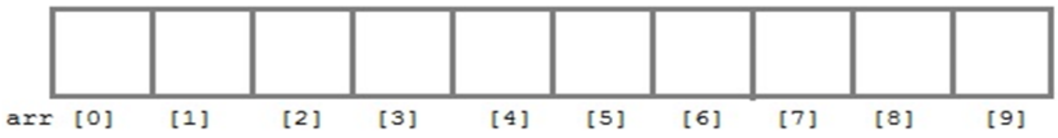
6. Differentiate entry-controlled and exit-controlled loops

Entry Control Loop	**Exit Control Loop**
Entry control loop checks condition first and then body of the loop will be executed.	The exit control loop first executes the body of the checks condition at last.

Entry Control Loop	**Exit Control Loop**
The body of the loop may or may not be executed at all.	The body of the loop will be executed at least once condition is checked at last
for, while are an example of an entry control loop	Do…while is an example of an exit control loop.

7. Explain one-dimensional array with the help of suitable code example .
   Like any other variable, arrays must be declared before they are used. General form of array declaration is,
   data-type variable-name[size];

   int arr[10];



•	Here int is the data type, arr is the name of the array and 10 is the size of array. It means array arr can only contain 10 elements of int type.
•	Index of an array starts from 0 to size-1 i.e first element of arr array will be stored at arr[0]address and the last element will occupy arr[9].

Initialization of an Array

After an array is declared it must be initialized. Otherwise, it will contain garbage value . An array can be initialized at either compile time or at runtime.
Compile time initialization of array elements is same as ordinary variable initialization. The general form of initialization of array is,

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

```
/* Here are a few examples */
int marks[4]={ 67, 87, 56, 77 };    /* integer array initialization*/
float area[5]={ 23.4, 6.8, 5.5 };   /* float array initialization*/

int marks[4]={ 67, 87, 56, 77, 59 };    /* Compile time error*/
```

If we omit the size of the array, an array size is equal to number elements assigned.
int balance[] = {1000 , 200, 300, 700, 500};
in the above example size of array is 5.

```
char b[]={'C','O','M','P','U','T','E','R'};    /*character Array*/
```

Example:
/* Program to initialize array at run time.*/

```
#include<stdio.h>
main()
{
   int a[4];
   int i, j;
   printf("Enter array element:");
   for(i=0; i<4; i++)
   {
      scanf("%d",&a[i]);    //Run time array initialization
   }
   printf("Value in Array:");
   for(j=0; j<4; j++)
   {
      printf("%d \n",a[j]);
   }
   getch();
}
```

8. Explain two-dimensional array with the help of suitable code example .
   It is an ordered table of homogeneous elements. It is generally referred to as matrix, of some rows and some columns.
   The general form of two dimensional array is
   storage_class data_type array~name[rows][columns]
   for example

   ```
   int Mat[ 3 ][ 4 ];
   ```

   declares that "Mat" is a two dimensional array of type int with 3 rows and 4 columns. The subscript value ranges from 0 to 2 for rows and for columns it range from 0 to 3 (4 columns). Storage class is optional.

   In general an array of the order M X N (read as M by N) consists of M rows, N columns and MN elements. It may be depicted as shown below.

   0 1 2 3 n-1

   1
   2
   3
   4

   m-1

   Fig. Two - dimensional array of the order M x N

Initialization of two dimensional array

The general form of two - dimensional array initialization is .
data_type array _name[row][column]={ value 1, value2, ..
valuen}
where storage_class is optional.
Data_type is the basic data type, array_name is the name of a two dimensional array. value 1, value2, ….. valuen are the initial values to be assigned to a two dimensional array.
Example:
int mat [2][2] = { 1,2,3,4}; then the elements of matrix will be
mat[O][O] = 1 mat[O][1] = 2 mat[l][O] = 3 mat[l][l] = 4 If the number of elements to be assigned are less than the total number of elements, that two dimensional array contained, then all the remaining~ elements are assigned zeros.
The statement
int mat [2][2] = { 1,2,3,4 };
can be equivalently written as
int mat [2][2] = { { 1,2}, {3, 4 } };
by surrounding the elements of each row by braces.
If the values are missing in an initialization, then that is automatically set to zero.
For example the statement
int mat [2][3] = { { 1, 1 } }, { 2 } }; will initialize the first two elements of the first row to one, the first element of the second row to two, and all other elements to zero.
Example:

```
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
for(i=0;i<4;i++){
 for(j=0;j<3;j++){
   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
return 0;
}
```

9. Explain the different ways available for the initialization of string variable .
Character arrays can be initialized in two ways. That is, characterr
arrays may be initialized when they are declared.
char subject[9] = {'c',' o' ,'m' ,'p','u' ,'t',' e' ,'r' ,'\0' }

Even though there are eight characters in the word "computer", the character array "subject" is initialized to 9. This is because to store the null ('\0') character at the end

of the string. When character array is initialized by listing its elements, null character must be supplied explicitly as shown in the above example.
Consider another example

```
  char subject[9]={ "computer"};
```

Here each character of the string "computer" is stored in individual elements of a character array subject, but it automatically adds null character at the end of the string. It is also possible to initialise a character array without specifying the number of elements. In such cases, the size of the array will be determined automatically, based on the number of elements in the given string. .
For example

```
static char month[ ] = "April"
```

Defines the character array month as a 6 element array.

Reading String From Terminal

The input function scanf ( ) with %s format specification can be used to read an array of characters.
Example

```
char Subject[10];
scanf("%s" ,Subject);
```

If we enter subject name as "Computer Science" then only the first word "Computer" will be assigned to the variable Subject This is because, the scanf( ) function terminates its input on the first occurrence of the white space.
In such cases, when line of text has to be read from the terminal one can make use of gets() function or one can make use of getchar() function repeatedly to read successive single characters from the keyboard and place them into a character array.
¬¬
Example Program 1

```
# include <stdio.h>
        # include <string.h>
main()
{
char text[50], ch;
int i=0;
printf("Enter line of text and press enter key at the end\n");
while((ch=getchar())!=’\n’)
(
text[i]=ch;
i++;
}
text[i]='\0' /*put null at the end*/
printf("Entered line of the text:\n%s" ,text);
}
```

10. Write a program to sort elements of an integer array.
    Program to input n numbers and sort it in ascending order using bubble sort.

```
#include<stdio.h>
main()
 {
   int a[10],n,i,j,temp;
   clrscr();
    printf("Enter number of    elements:");
   scanf("%d",&n);
   printf("Enter array elements:");
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);
   for(i=0;i<n;i++)
   for(j=0;j<n-i-1;j++)
  if(a[j]>a[j+1])
  {
    temp=a[j];
    a[j]=a[j+1];
    a[j+1]=temp;
   }
  printf("Sorted array is\n");
  for(i=0;i<n;i++)
  printf("%d\t",a[i]);
  getch();
 }
```

11. Write a program to search for an element in an integer array.
    Program to search a number in a list, using linear search technique. If present print its position(s).

```
#include<stdio.h>
main()
 {
   int a[10],n,x,i,flag=0;
   clrscr();
   printf("Enter number of elements:");
   scanf("%d",&n);
   printf("Enter array elements:");
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);
   printf("Enter search element:");
   scanf("%d",&x);
   for(i=0;i<n;i++)
   {
if(a[i]==x)
 {
   printf("%d is found at the location   %d\n",x,i);
   flag=1;
 }
}
 if(flag==0)
 printf("Given element not found");
 getch();
}
```

12. Explain with example how to declare, initialize and use strings in C.

Declaring String Variables

The general form of declaration of a string variable is

```
char string_name[ size];
```

Where string_name is a valid variable name and the size determines' the number of characters in the string_name.

For example

```
char name [20] ;
```

When the compiler assigns a string data to character array, it automatically supplies a null character ('\0') at the end of the string. Therefore the array size should be one more than maximum number of characters in the string.

Initializing Strings

Character arrays can be initialized in two ways. That is, characterr arrays may be initialized when they are declared.

```
char subject[9] = {'c',' o' ,'m' ,'p','u' ,'t',' e' ,'r' ,'\0' }
```

Even though there are eight characters in the word "computer", the character array "subject" is initialized to 9. This is because to store the null ('\0') character at the end of the string. When character array is initialized by listing its elements, null character must be supplied explicitly as shown in the above example.

Consider another example

```
char subject[9]={ "computer"};
```

Here each character of the string "computer" is stored in individual elements of a character array subject, but it automatically adds null character at the end of the string. It is also possible to initialise a character array without specifying the number of elements. In such cases, the size of the array will be determined automatically, based on the number of elements in the given string. .

For example

```
static char month[ ] = "April"
```

Defines the character array month as a 6 element array.

Reading String From Terminal

The input function scanf ( ) with %s format specification can be used to read an array of characters.

Example

```
char Subject[10];
scanf("%s" ,Subject);
```

If we enter subject name as "Computer Science" then only the first word "Computer" will be assigned to the variable Subject This is because, the scanf( ) function terminates its input on the first occurrence of the white space.
In such cases, when line of text has to be read from the terminal one can make use of gets() function or one can make use of getchar() function repeatedly to read successive single characters from the keyboard and place them into a character array.
¬¬
Example Program 1

```
# include <stdio.h>
        # include <string.h>
main()
{
char text[50], ch;
int i=0;
printf("Enter line of text and press enter key at the end\n");
while((ch=getchar())!='\n')
(
text[i]=ch;
i++;
}
text[i]='\0' /*put null at the end*/
printf("Entered line of the text:\n%s" ,text);
}
```

Writing Strings To Screen

The printf function with %s format specifier can be used to display the strings on the screen. For example the following statement

```
printf ("%s", Subject);
```

Can be used to display the contents of the character array Subject.
The %s format conversion character with an optional w.d specification can be used to format the output. In this case 'd' number of characters from the beginning of the string will be printed in a field of width w.

Example Program 2

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

```
{
clrscr();
printf ("%20s", "Left justified printing.\n");
printf ("%-40.24s", "Left justified printing.\n");
printf ("%-40.16s", "Left justified printing.\n");
printf ("%-40.12s", "Left justified printing.\n");
printf ("%-40.8s", " Left justified printing.\n");
printf ("%-40s", "Left justified printing.\n");
printf ("%-40.0s", " Left justified printing.\n");
printf ("%40.25s", "Right justified printing.\n");
printf ("%40.20s", "Right justified printing.\n");
printf ("%40.5s", "Right justified printing.\n");
 printf ("%40.0s", "Right jusdfied printing.\n”);
}
```

The output of the program illustrates the following features of the %s specifications.

1. When the field width is less than the length of the string, the entire string is printed.
2. The integer value on the right side of the decimal point specifies the number of characters to be printed.
3. When the number of characters to be printed is specified as zero, nothing is printed.
4. The minus (-) sign in the specification causes the string to be printed left justified.

13. List and explain any four-string handling functions available in C

1. strcat( ) function

This function is used to concatenate two strings, i.e., it appends one string at the end of another string. This function accepts two strings as parameters and adds the contents of the second string at the end of the first string. Its syntax is follows
strcat (string!, string2);
here, the string string2 is appended to the end of string1.

Example Program 3

```
/* Program to concatenate two given strings */
#include <stdio.h>
#include <string.h>
main()
{
char strl [50], str2[25];
clrscr();
printf("Enter the first string\n");
gets( strl);
printf("Enter the second string\n");
gets(str2);
strcat(strl, str2);          1* concatenate strl and str2 */
printf("Concatenated string :\t %s", strl);
 getch();
}
```

2. strlen() function

strlen() function returns the number of characters in the given string. Here, the length does not include the terminating character '\0' (NULL). Its syntax is as follows.
Len = strlen(string);
Where Len is an integer type variable and string is an one dimensional array of characters.

Example Program 4

```
/* Program to find the length of the entered string */
#include <stdio.h>
#include <string.h>
main()
{
char str[50];
clrscr();
printf("Enter the string\n");
gets( str) ;
printf("Length of the entered string: %d", strlen(str));
getch();
        }
```

3. strcmp () function

This function is used to compare two strings. The function accepts two strings as parameters and returns an integer, whose value is:
Less than 0, if the first string is less than the second string Equal to 0, if both are identical
Greater than 0, if the first string is greater than the. second string. The general form of strcmp() function is as follows

        strcmp (stringl, string2);

The strcmp() function compares two strings, character by character, (ASCII comparision) to decide the greatest one. Whenever two characters in the string differ, there is no need to compare the other characters of the strings.

Example Program 5

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

```
#include <stdio.h>
main()
 {
char Strl [] = "aaa", Str2[] = "bbb", Str3[] = "ccc";
int ptr;
clrscr();
ptr = strcmp(Str2, Str]);
if (ptr > 0)
            printf("String 2 is greater than String 1 \n");
            else
printf("String 2 is less than String 1 \n");
ptr = strcmp(Str2, Str3);
if (ptr > 0)
   printf("String 2 is greater than String 3\n");
else
printf("String 2 is less than String 3\n");
 getch();
 }
```

4. strcpy() function

This function copies one string to another. The strcpy() function accepts two strings as parameters and copies the second string, character by character into the first string, upto and including null character of the second string. Its syntax is as follows
strcpy (string 1, string2);

Example Program 6

```
#include <stdio.h>
#include <string.h>
main()
{
char string[25];
char strl [ ] = "Computer Science";
 clrscr( );
strcpy(string, strl);
printf("Copied string: %s\n", string);
getch();
}
```

14. Mention suitable string functions to do the following:
    a. To find length of the string
       strlen() function returns the number of characters in the given string. Here, the length does not include the terminating character '\0' (NULL). Its syntax is as follows.
       Len = strlen(string);
       Where Len is an integer type variable and string is an one dimensional array of characters.

Example Program 4

```
/* Program to find the length of the entered string */
#include <stdio.h>
#include <string.h>
main()
{
char str[50];
clrscr();
printf("Enter the string\n");
gets( str) ;
printf("Length of the entered string: %d", strlen(str));
getch();
      }
```

b. To copy one string to another string

This function copies one string to another. The strcpy() function accepts two strings as parameters and copies the second string, character by character into the first string, upto and including null character of the second string. Its syntax is as follows

strcpy (string 1, string2);

Example Program 6

```
#include <stdio.h>
#include <string.h>
main()
{
char string[25];
char strl [ ] = "Computer Science";
 clrscr( );
strcpy(string, strl);
printf("Copied string: %s\n", string);
getch();
}
```

c. To add two strings

This function is used to concatenate two strings, i.e., it appends one string at the end of another string. This function accepts two strings as parameters and adds the contents of the second string at the end of the first string. Its syntax is follows

strcat (string!, string2);

here, the string string2 is appended to the end of string1.

Example Program 3

```
/* Program to concatenate two given strings */
#include <stdio.h>
#include <string.h>
```

```
main()
{
char strl [50], str2[25];
clrscr();
printf("Enter the first string\n");
gets( strl);
printf("Enter the second string\n");
gets(str2);
strcat(strl, str2);     1* concatenate strl and str2 */
printf("Concatenated string :\t %s", strl);
 getch();
}
```

d. To reverse given string
strrev() is a non-standard C library function, sometimes found in <string.h>, that is used to reverse a string.
Example:

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

int main() {
   char str[20] = "coffee";

   printf("String before strrev(): %s\n",str);

   strrev(str);

   printf("String after strrev(): %s\n",str);

   return 0;
}
```

15. What are the possible values that the function strcmp( ) can return? What do they mean?
This function is used to compare two strings. The function accepts two strings as parameters and returns an integer
Return Value from strcmp()

Return Value	Remarks
0	if strings are equal
>0	if the first non-matching character in str1 is greater (in ASCII) than that of str2.

Return Value	Remarks
<0	if the first non-matching character in str1 is lower (in ASCII) than that of str2.

Example:

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

int main () {
  char str1[15];
  char str2[15];
  int ret;


  strcpy(str1, "abcdef");
  strcpy(str2, "ABCDEF");

  ret = strcmp(str1, str2);

  if(ret < 0) {
    printf("str1 is less than str2");
  } else if(ret > 0) {
    printf("str2 is less than str1");
  } else {
    printf("str1 is equal to str2");
  }

  return(0);
}
```

UNIT-4

Multiple Choice Questions:

1. Choose correct statement about Functions in C Language.
A) A Function is a group of c statements which can be reused any number of times.
B) Every Function has a return type.
C) Every Function may no may not return a value.
D) All the above.

2. Choose a correct statement about C Language Functions.
A) A function name can not be same as a predefined C Keyword.
B) A function name can start with an Underscore(_) or A to Z or a to z.

C) Default return type of any function is an Integer.
D) All the above.

3. Choose a correct statement about C Function.?

```
main()
{
   printf("Hello");
}
```

A) "main" is the name of default must and should Function.
B) main() is same as int main()
C) By default, return 0 is added as the last statement of a function without specific return type.
D) All the above

4. A function which calls itself is called a ___ function.
A) Self Function
B) Auto Function
C) Recursive Function
D) Static Function

5. What is the output of C Program with functions.?

```
int main()
{
   show();
   printf("BANK ");
   return 0;
}

void show()
{
   printf("CURRENCY ");
}
```

A) CURRENCY BANK
B) BANK CURRENCY
C) BANK
D) Compiler error

6. How many values can a C Function return at a time.?
A) Only One Value
B) Maximum of two values
C) Maximum of three values
D) Maximum of 8 values

7. What is the output of C Program with functions.?

```
int show();
```

```
void main()
{
   int a;
   printf("PISTA COUNT=");
   a=show();
   printf("%d", a);
}

int show()
{
   return 10;
}
```

A) PISTA COUNT=
B) PISTA COUNT=0
C) PISTA COUNT=10
D) Compiler error

8. What are types of Functions in C Language.?
A) Library Functions
B) User Defined Functions
C) Both Library and User Defined
D) None of the above

9. What is the limit for number of functions in a C Program.?
A) 16
B) 31
C) 32
D) None of the above

10. Every C Program should contain which function.?
A) printf()
B) show()
C) scanf()
D) main()

11. What is the minimum number of functions to be present in a C Program.?
A) 1
B) 2
C) 3
D) 4

12. What characters are allowed in a C function name identifier.?
A) Alphabets, Numbers, %, $, _
B) Alphabets, Numbers, Underscore (_)
C) Alphabets, Numbers, dollar $

D) Alphabets, Numbers, %

13. Arguments passed to a function in C language are called ___ arguments.
A) Formal arguments
B) Actual Arguments
C) Definite Arguments
D) Ideal Arguments

14. Arguments received by a function in C language are called ___ arguments.
A) Definite arguments
B) Formal arguments
C) Actual arguments
D) Ideal arguments

15. Choose a corrects statement about C language function arguments.
A) Number of arguments should be same when sending and receiving
B) Type of each argument should match exactly
C) Order of each argument should be same
D) All the above

16. Choose a non Library C function below.
A) printf()
B) scanf()
C) fprintf()
D) printf2()

17. What is the default return value of a C function if not specified explicitly.?
A) -1
B) 0
C) 1
D) None of the above

18. A recursive function can be replaced with __ in c language.
A) for loop
B) while loop
C) do while loop
D) All the above

19. A recursive function without If and Else conditions will always lead to.?
A) Finite loop
B) Infinite loop
C) Incorrect result
D) Correct result

20. What is the C keyword that must be used to achieve expected result using Recursion.?
A) printf

B) scanf
C) void
D) return

21. How many functions are required to create a recursive functionality.?
A) One
B) Two
C) More than two
D) None of the above

22. Choose a correct statement about Recursive Function in C language.
A) Each recursion creates new variables at different memory locations
B) There is no limit on the number of Recursive calls
C) Pointers can also be used with Recursion but with difficulty.
D) All the above

23. Identify wrong C Keywords below.
A) auto, double, int, struct
B) break, else, long, switch
C) case, enum, register, typedef
D) char, extern, intern, return

24. What is a C Storage Class.?
A) C Storage decides where to or which memory store the variable.
B) C Storage Class decides what is the default value of a variable.
C) C Storage Class decides what is the Scope and Life of a variable.
D) All the above.

25. Find a C Storage Class below.
A) static
B) auto
C) register & extern
D) All the above

26. What is the default C Storage Class for a variable.?
A) static
B) auto
C) register
D) extern

27. Variables of type auto, static and extern are all stored in .?
A) ROM
B) RAM
C) CPU
D) Compiler

28. Which among the following is a Local Variable.?
A) register
B) auto
C) static
D) extern

29. which among the following is a Global Variable.?
A) auto
B) register
C) static
D) extern

30. Choose a correct statement about static variable.
A) A static global variable can be accessed in other files.
B) A static global variable can be used only in a file in which it is declared.
C) A static global variable can not be declared without extern keyword.
D) Default value of a static variable is -1.

Descriptive Questions:

1) Explain with example how to declare, define and call a function in C

Function prototype (Declaration)

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.

A function prototype gives information to the compiler that the function may later be used in the program.

The function prototype is not needed if the user-defined function is defined before the main() function.

Syntax of function prototype

returnType functionName(type1 argument1, type2 argument2,...);

Function definition

Function definition contains the block of code to perform a specific task.

Syntax of function definition

```
return type functionName(data-type  argument1, data-type  argument2, ...)
{   Local Variable Declarations

    Execution statement(s)

   [return  value/expression]
}
```

Where .

Return data type- type of the return value by the functions.

If nothing is returned to the calling function, then data type is void. Function_name - It is the user defined function name.

Argument(s) - The argument list contains valid variable name separated by commas. .

return statement- It is used to return value to the calling function. A function can have multiple return statements . But only one return statement is executed (A function can return only single value). Execution of return statement causes execution to be transferred back to calling function.

When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function.

Calling a Function:

A function can be accessed or called by specifying its name, followed by a list of arguments enclosed in parentheses and separated by commas. If the function call does not require any arguments, an empty pair of parentheses must follow the function name.

2) Explain with the help of a code example declaring, defining and calling a function in C

Function prototype (Declaration)

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.

A function prototype gives information to the compiler that the function may later be used in the program.

The function prototype is not needed if the user-defined function is defined before the main() function.

Syntax of function prototype

returnType functionName(type1 argument1, type2 argument2,...);

Function definition

Function definition contains the block of code to perform a specific task.

Syntax of function definition

```
return type functionName(data-type  argument1, data-type  argument2, ...)
{   Local Variable Declarations

    Execution statement(s)

   [return  value/expression]
}
```

Where .

Return data type- type of the return value by the functions.

If nothing is returned to the calling function, then data type is void. Function_name - It is the user defined function name.

Argument(s) - The argument list contains valid variable name separated by commas. .

return statement- It is used to return value to the calling function. A function can have multiple return statements . But only one return statement is executed (A function can return only single value). Execution of return statement causes execution to be transferred back to calling function.

When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function.

Calling a Function:

A function can be accessed or called by specifying its name, followed by a list of arguments enclosed in parentheses and separated by commas. If the function call does not require any arguments, an empty pair of parentheses must follow the function name.

Example Program 1

```
    # include <stdio.h>
int Add(int x, int y);  /*function prototype or function declaration*/
main()
      {
        int a, b, sum;
       printf("enter two number");
        scanf("%d %d", &a, &b);
       sum=Add(a,b );
       printf("sum of %d and %d is %d", a, b, sum);
       }/*End of the main*/

int Add (int x, int y) /*Function to add two numbers */
{
  return (x + y);
}
```

3) Explain with example function with no arguments and no return values
   Such functions can either be used to display information or they are completely dependent on user inputs.
   Below is an example of a function, which takes 2 numbers as input from user, and display which is the greater number.

```
#include<stdio.h>
void greatNum();       // function declaration
int main()
{
    greatNum();        // function call
    return 0;
}

void greatNum()        // function definition
{
    int i, j;
    printf("Enter 2 numbers that you want to compare...");
    scanf("%d%d", &i, &j);
    if(i > j) {
        printf("The greater number is: %d", i);
    }
    else {
        printf("The greater number is: %d", j);
    }
}
```

4) Explain with example function with arguments and no return value
   We are using the same function as example again and again, to demonstrate that to solve a problem there can be many different ways.

This time, we have modified the above example to make the function greatNum() take two integer values as arguments, but it will not be returning anything.

```
#include<stdio.h>
void greatNum(int a, int b);       // function declaration
int main()
{
    int i, j;
    printf("Enter 2 numbers that you want to compare...");
    scanf("%d%d", &i, &j);
    greatNum(i, j);        // function call
    return 0;
}

void greatNum(int x, int y)        // function definition
{
    if(x > y) {
        printf("The greater number is: %d", x);
    }
    else {
        printf("The greater number is: %d", y);
    }
}
```

5) Explain with example function with no arguments but a return value

We have modified the above example to make the function greatNum() return the number which is greater amongst the 2 input numbers.

```
#include<stdio.h>

int greatNum();       // function declaration

int main()
{
    int result;
    result = greatNum();        // function call
    printf("The greater number is: %d", result);
    return 0;
}

int greatNum()        // function definition
{
    int i, j, greaterNum;
    printf("Enter 2 numbers that you want to compare...");
    scanf("%d%d", &i, &j);
    if(i > j) {
        greaterNum = i;
    }
    else {
        greaterNum = j;
    }
    // returning the result
    return greaterNum;
}
```

6) Explain with example functions with arguments and a return value

This is the best type, as this makes the function completely independent of inputs and outputs, and only the logic is defined inside the function body.

```
#include<stdio.h>
int greatNum(int a, int b);       // function declaration
int main()
{
    int i, j, result;
    printf("Enter 2 numbers that you want to compare...");
    scanf("%d%d", &i, &j);
    result = greatNum(i, j); // function call
    printf("The greater number is: %d", result);
    return 0;
}

int greatNum(int x, int y)        // function definition
{
    if(x > y) {
        return x;
    }
    else {
        return y;
    }
}
```

7) Write a note on a) nesting of functions b) recursive functions

Nesting of Functions

C language also allows nesting of functions i.e to use/call one function inside another function's body. We must be careful while using nested functions, because it may lead to infinite nesting.

```
function1()
{
    // function1 body here
        function2();
        // function1 body here
}
```

If function2() also has a call for function1() inside it, then in that case, it will lead to an infinite nesting. They will keep calling each other and the program will never terminate.

Not able to understand? Lets consider that inside the main() function, function1() is called and its execution starts, then inside function1(), we have a call for function2(), so the control of program will go to the function2(). But as function2() also has a call to function1() in its body, it will call function1(), which will again call function2(), and this will go on for infinite times, until you forcefully exit from program execution.

Recursive functions are functions that call themselves, so a recursion is process of calling function by itself.

Example program 7
/*Recursive function program to find the factorial of a number*/

```
#include<stdio.h>
int fact(int n);
main( )
(
 int num;
clrscr( );
 printf("\nEnter a number :");
 scanf("%d" ,&num);

/* processing and output part */
 clrscr( );
 printf("Entered number: %d",num);
 printf("\nFactorial of the entered number: %d", fact(num));
 getch( );
}
int fact(int num)
{
if(num==0)
 return (1);
else
 return (num * fact(num-1);
}
```

Suppose the definition fact is called with the value 2, then the value of num in the function fact is 2, and the condition in the if statement is false, and the function gets called again with the value num - 1, which is 1. In the second invocation of fact, the value of num is 1. The condition in the if statement is again false, and a third invocation of fact results, passing 0 to it. In the third invocation, the value of num 0, and the condition of fact returns 1 to the second invocation.
Hence, the statement
return (num * fact (num-1);
Multiplies 1 by 1 and causes the number 1 to be returned to the first invocation; where the value of num was 2. This brings back to the statement
return (num * fact(num-l));

Which is now executing in the first invocation. The return value of the second invocation, 1 is multiplied by 2 and the value 2 is returned. Two important conditions must be satisfied by any recursive function:

i) Each time when a function calls itself, it must be closer, in some sense to a solution.

ii) There must be a decision criterion for stopping the process or computation.

For the function factorial, each time when the function calls itself, its argument is decremented by one. The stopping criterion is the if statement that checks for the zero argument.

8) What are the actual and formal arguments?

Function parameters are the means of communication between the calling and the called functions. They can be classified into formal parameters and actual parameters. The formal parameters are the parameters given in the function declaration and function definition. The actual parameters(commonly called arguments), are specified in the function call.

```
main ( )
{
   ---------
   ---------
                 actual parameters
  function1(a1, a2,  a3   am);   /* function call*/
¬  ---------
  ---------
}


function1  (fI,   f2,    f3 fn)     /* called function*/
{           formal parameters
  --------
  --------
}
```

Arguments matching between the function call and the called function.

The actual and formal arguments should match in number, type and order. The values of actual arguments are assigned to the formal arguments on a one to one basis starting with the first argument as shown in figure. In case, the actual arguments are more than the formal parameters (m > n), the extra actual arguments are discarded. But if the actual arguments are less than the fomal arguments, the unmatched formal arguments are initialized to some garbage values. Any mismatch in data type may also result in passing of garbage values.

The formal parameters must be valid variable names, but the actual arguments may be variable names, expressions or constants. The variables used in actual arguments must be assigned values before the function call is made.
Example

```
   # include <stdio.h>
int Add(int x, int y);  /*function prototype or function declaration*/
main()
     {
  int a, b, sum;
 printf("enter two number");
  scanf("%d %d", &a, &b);
 sum=Add(a,b );
 printf("sum of %d and %d is %d", a, b, sum);
 }/*End of the main*/

int Add (int x, int y) /*Function to add two numbers */
{
  return (x + y);
}
```

9) What is a recursive function explain with example?
Recursive functions are functions that call themselves, so a recursion is process of calling function by itself.

Example program 7
/*Recursive function program to find the factorial of a number*/

```
#include<stdio.h>
int fact(int n);
main( )
(
 int num;
clrscr( );
 printf("\nEnter a number :");
 scanf("%d" ,&num);

/* processing and output part */
 clrscr( );
 printf("Entered number: %d",num);
 printf("\nFactorial of the entered number: %d", fact(num));
 getch( );
}
int fact(int num)
{
if(num==0)
```

```
 return (1);
else
 return (num * fact(num-1);
}
```

Suppose the definition fact is called with the value 2, then the value of num in the function fact is 2, and the condition in the if statement is false, and the function gets called again with the value num - 1, which is 1. In the second invocation of fact, the value of num is 1. The condition in the if statement is again false, and a third invocation of fact results, passing 0 to it. In the third invocation, the value of num 0, and the condition of fact returns 1 to the second invocation.
Hence, the statement
return (num * fact (num-1);
Multiplies 1 by 1 and causes the number 1 to be returned to the first invocation; where the value of num was 2. This brings back to the statement
return (num * fact(num-l));
Which is now executing in the first invocation. The return value of the second invocation, 1 is multiplied by 2 and the value 2 is returned. Two important conditions must be satisfied by any recursive function:

i) Each time when a function calls itself, it must be closer, in some sense to a solution.
ii) There must be a decision criterion for stopping the process or computation.

For the function factorial, each time when the function calls itself, its argument is decremented by one. The stopping criterion is the if statement that checks for the zero argument.

10) Write recursive function to calculate factorial of a given number.

```
/*Recursive function program to find the factorial of a number*/

#include<stdio.h>
int fact(int n);
main( )
(
 int num;
clrscr( );
 printf("\nEnter a number :");
 scanf("%d" ,&num);

/* processing and output part */
 clrscr( );
 printf("Entered number: %d",num);
 printf("\nFactorial of the entered number: %d", fact(num));
 getch( );
}
int fact(int num)
```

```
{
if(num==0)
 return (1);
else
 return (num * fact(num-1);
}
```

11)Write a note on automatic variable.

The variables in main() are private or local to main(). Because they are declared within main, no other function can have direct access to them. The same is true of the variables in other functions. Each local variable in a function comes into existence only when the function is called and disappeared when the function is exited. Such variables are usually known as automatic variables.We can make use of the optional keyword in the declaration, auto for Example

```
Auto int  x;
```

auto has been the default class for all the variables we have used so for. The value of the automatic variables cannot be changed in some other function in the program. So, we can declare and use the same variable name in different functions in the same program.

Example:

```
#include<stdio.h>
int Add (int a,int b);
 main( )
{
auto int a=10,b=15,c;
clrscr( );
c= a+b;
printf("c= %d", c);
c= Add(a,b);
printf("\nc= %d", c);
      }
int Add( int a, int b)
      {
      int c;
     c= a+b;
     return c;
  }
```

12) Write a note on register variable.

Register variables are local variables (similar to automatic variables) except that they are placed in machine registers. A register declaration advises the compiler that the variable in question will be heavily used. The idea is that register variables are to be placed in machine registers, which may result in smaller and faster programs. But compilers are free to ignore the advice.

The register declaration looks like

register int i;

Then the compiler may try to place the variable 'i' in a machine register if available.

The register declaration can only be applied to automatic variables and the parameters of a function. Since only a few variables can be placed in the register, 'C' will automatically convert register variables into auto variables once the limit is reached. Loop indices, accessed more frequently, can be declared as register variables. For example 'index' is a register variable in the program given below.

Example:

```
# inc1ude<stdio.h>
main( )
 {

   register index, Sum = 0;
for (index== 1; index<==100; index++)
Sum == Sum + index;
 printf(" Sum == %d", Sum);
}
```

Finally one cannot assign large or aggregate data types, such as doubles or arrays to registers and the & operator cannot be applied to register variables.

13) Write a note on external variable.

. External variables are defined outside of any function, and are thus available to many functions. Hence, they are also known as global variables. By default, external variables and functions have the property that all references to them by the same name.

Because external variables are globally accessible, they provide an attention to function arguments and return values for communicating data between functions. Any function may access an external variable by referring to it by name. If a number of variables must be shared among functions, external variables are more convenient and efficient than long argument lists. External variables are also useful because of their greater scope and lifetime. Automatic(local) variables are internal to a function; they come into existence when the function is entered, and disappeared (destroyed) when it is left. External variable on the other hand, are permanent, so they retain values from one function invocation to the next. Thus if functions must share some data, yet neither calls the other, it is often most convenient if the shared data is kept in external variables rather than passed in and out via arguments.

One important feature of global variable is that if a function has a local variable, of the same name as a global variable, the local variable has precedence over the global variable. Analogously, a variable declared within a block has precedence over an external variable of the same name.

In order to be able to use an externally declared variable inside' a function, such a variable may explicitly redeclared inside the function with the extern qualifier, for example:

extern int X;

Example:

```
#include <stdio.h>
int Sum;
void Inc( );
main( )
{
 int a,b;
 clrscr( );
 priiltf("Enter two numbers");
 scanf("%d %d", &a,&b);
 Sum = a+b;
printf("Sum of %d and %d is %d", a,b,Sum);
Inc( );
printf("\nAfter incrementing the value of Sum is %d",Sum);
getch( );
}
 void Inc( )
{
Sum ++;
}
```

14) Write a note on static variable.

Another C storage declaration is static. The keyword static is used to declare variables which must not lose their storage locations or their values when control leaves the functions or blocks where in they are defined. In C, static variables retain their values between function calls. The initial value assigned to a static variable must be a constant, or an expression involving constants. But static variables are by default initialized to 0, in contrast to autos.

For example

```
 Static int i;
```

Initalises variable i to 0

¬Example Program 13

```
 #include<stdio.h>
void Display(void);
main( )
{
int i;
clrscr( );
for ( i=1; i<=4; i++)
Display( );
}
void Display(void)
```

```
 {
   static int x= 0;
    x++;
  printf("\nx=%d" ,x);
}
```

output:

```
 x=1
x=2
x=3
x=4
```

A static variable is initialized only once, when the program is compiled. It is never initialized again. During the first call to Display( ), x is incremented to I. Because x is static, this value retains and therefore, the next call adds another I to x giving its value equal to 2. The value of x becomes 3 when third call is made and 4 when fourth call is made.

UNIT – 5

Multiple Choice Questions:

1. Which of the following are themselves a collection of different data types?

   A. String
   B. Structures
   C. Char
   D. None of the above

2. Which operator connects the structure name to its member name?

   A. -
   B. ->
   C. .
   D. both . and ->

3. Which of the following cannot be a structure member?
   A. Function
   B. Array
   C. Structure
   D. None of the above

4. Union differs from structure in the following way

   A. All members are used at a time
   B. Only one member can be used at a time
   C. Union cannot have more members
   D. Union initialized all members as structure

5. size of union is size of the longest element in the union

   A. Yes
   B. No
   C. May Be
   D. Can't Say

6. Which of the following are themselves a collection of different data types?

   a. string
   b. **structures**
   c. char
   d. all of the mentioned

7. Which of the following comment about Union is false?

   A. Union is a structure whose members share same memory area

   B. The compiler will keep track of what type of information is currently stored

   C. Only one of the members of union can be assigned a value at particular time

   D. Size allocated for Union is the size of its member needing the maximum storage

6. Which of the following comment about the usage of structures in true?
   A. Storage class can be assigned to individual member
   B. **The scope of the member name is confined to the particular structure, within which it is defined**
   C. Individual members can be initialized within a structure type declaration
   D. None of above

9. Which of the following statements correct about the below code?
   maruti.engine.bolts=25;
   A: Structure bolts is nested within structure engine.
   B: Structure engine is nested within structure maruti
   C: Structure maruti is nested within structure engine
   D: Structure maruti is nested within structure bolts.

10. Size of a union is determined by size of the
    A: First member in the union
    B: Last member in the union
    C: **Biggest member in the union**
    D: Sum of the sizes of all members

11. Members of a union are accessed as________________.
    A: union-name.member
    B: union-pointer->member
    C: Both a & b
    D: None of the mentioned

12. Which of the following share a similarity in syntax?
    A: 3 and 4
    B: 1 and 2
    C: 1 and 3
    D: 1, 3 and 4

13. What is the output of this C code?
```
struct student
{
};
void main()
{
struct student s[2];
printf("%d", sizeof(s));
}
```
    A: 2
    B: 4
    C: 8
    D: 0

14. What will happen when the structure is declared?
    A: it will not allocate any memory
    B: it will allocate the memory
    C: it will be declared and initialized
    D: none of the above.

15. Which of the following comments about union are true?
    A:Union is a structure whose members share the same storage area
    B:Size allocated for union is the size of its member needing the maximum storage
    C:Only one of the members of union can be assigned a value at a particular time
    D:All of these

16. Which of the following accesses a variable in structure b?
    A: b->var
    B: b.var
    C: b-var
    D: b>var

17. Union differs from structure in the following way
    A:All members are used at a time
    B:Only one member can be used at a time
    C:Union cannot have more members
    D:Union initialized all members as structure

18. Which of the following true about FILE *fp?
    a. FILE is a keyword in C for representing files and fp is a variable of FILE type.

b. FILE is a stream
c. FILE is a buffered stream
d. FILE is a structure and fp is a pointer to the structure of FILE type

19**.** The first and second arguments of fopen() are

a. A character string containing the name of the file & the second argument is the mode
b. A character string containing the name of the user & the second argument is the mode
c. A character string containing file pointer & the second argument is the mode
d. None of the mentioned

20. Which type of files can't be opened using fopen()?

a. .txt
b. .bin
c. .c
d. **None of the above**

21. If there is any error while opening a file, fopen will return?

a. Nothing
b. EOF
c. NULL
d. Depends on compiler

22. It is not possible to combine two or more file opening mode in open () method.

a. True
b. False
c. May be True or False
d. Can't Say

23. Which files will get closed through the fclose() in the following program?

```
void main()
{
    FILE *fp, *ft;
    fp = fopen("a.txt", "r");
    ft = fopen("b.txt", "r");
    fclose(fp,ft);
}
```

a. a, b
b. a
c. b
d. **Error in fclose**

24. The first and second arguments of fopen() are

A. A character string containing the name of the file & the second argument is the mode
B. A character string containing the name of the user & the second argument is the mode

C. A character string containing file pointer & the second argument is the mode
D. None of the mentioned

25. When a C program is started, O.S environment is responsible for opening file and providing pointer for that file?
A. Standard input
B. Standard output
C. Standard error
D. All of the above

26. A mode which is used to open an existing file for both reading and writing _______
a) ”W”
b) ”W+”
c) ”R+”
d) ”A+”

27. Select a function which is used to write a string to a file _______
a) pits()
b) putc()
c) fputs()
d) fgets()

28. Select a function which is used to read a single character from a file at a time?
a) fscanf()
b) getch()
c) fgetc()
d) fgets()

29. Which is data type of file pointer is ________
a) int
b) double
c) void
d) File

30. getc() returns EOF when
a) When getc() fail to read the character
b) When end of file is reached
c) Both A and B
d) None of the above

Descriptive Questions:

1. What is structure? Why it is required?

Arrays provide a means to homogenous data items into a single named unit. But most of the applications require the grouping together of heterogeneous data items. Such an entity is called structure and the related data items use in it are referred to as members. Thus a single structure might contain integer elements, floating point elements and character elements. Pointers, arrays and other structures can also be included as elements with in a structure.

Give the general format used for defining a structure

DEFINING A STRUCTURE

Every structure must be defined and declared before it appears in a C program .The general form of structure definition and declaration is as follows,

```
                    struct tagname
                    {
                      data-type member_1 ;
                      data-type member_2;
                      -----------
                      -----------
                 data-type member_n;
                      } ;
```

In this declaration struct is a required keyword ,tagname is a name of the structure and member_1 ,member_2, ...member_n are individual member declarations. The individual members can be ordinary variables, pointers, arrays or other structures. The member names with in a particular structure must be distinct from one another though a member name can be same as the name of a variable defined outside of the structure.

Example

```
struct Student
        {
            int Regno;
            char Name[20);
            int m1,m2,m3,Total_marks;
             float Avg;
        };
```

This structure named Student (i.e, the tag is student) contains seven members, an integer variable Regno ,a 20-element character array (Name [20]). integer variables m1,m2,m3,Total_marks and a floating type variable Avg.

The general form of declaring structure variable is

```
                         struct structure-type variable- list;
```

The following declaration declares student to be of type Std.

```
                    struct Student Std;
```

2. Give the general format used for defining a union?

The general form of union is

```
union tagnmae
 {
Data-type member__1;
Data-type member_2;
-----------¬
Data- type member n;
};
```

Where union is a required keyword and the tag is optional.
A union may be initialized by value of the type of its first member. If a union has been assigned a value of certain type then it is not correct to attempt to retrieve from it a value of a different type.
Example:

```
union Item
{ int x;
  float y;
  char z;
        } Alpha;
```

This declares a variable Alpha of type union Item. The union contains three members each with a different data type. But it is possible to use only one of them at a time. This is due to the fact that only one location is allocated for a union variable irrespective of size.

3. How to Access individual members of structure? Give example.

THE DOT OPERATOR

In order to gain access to a member of a structure, the dot operator "." is used. That is a structure member can be accessed by writing.

structure_name. member_name

Where structure_name refers to the name of a structure *type* variable and member _name refers to the name of member with in the structure. The period (.) is an operator and it is a member of the highest precedence group and it groups from left to right.

Example

Example:

```
#include<stdio.h>
#include<conio.h>
void main( )
{
struct employee {
                        long int code;
                        char name[20];
                        float salary;
                  } ;
struct employee emp;
clrscr( );
printf(" Enter employee code:");
scanf("%d" ,&emp.code);
fflush(stdin);
printf("Enter employee name:");
gets(emp.name);
fflush(stdin);
printf("Enter salary:");
scanf("%f' ,&emp.salary);
printf("EMPLOYEE INFORMATION SYSTEM\n\n\n");
printf("CODE NAME SALARY\n");
printf("%d\t %s\t %f\n",emp.code, emp.name, emp.salary);
getch( );
 }
```

4. What is union? How it is declared

Union like structures, contains members whose individual data type may differ from one another. The difference between union and structure the member that compose a union all share the same storage area within the computer's memory. Where as in structures, each member has its own storage location. This implies that, although a union may contain many members of different types, it can handle only one member at a time. They are useful for applications involving multiple members where values need not be assigned to all of the members at anyone time. Thus unions are used to conserve memory.

The general form of union is

```
union tagnmae
 {
Data-type member__1;
Data-type member_2;
-----------
Data- type member n;
};
```

Where union is a required keyword and the tag is optional.

A union may be initialized by value of the type of its first member. If a union has been assigned a value of certain type then it is not correct to attempt to retrieve from it a value of a different type.

Example:

```
union Item
{ int x;
  float y;
  char z;
} Alpha;
```

This declares a variable Alpha of type union Item. The union contains three members each with a different data type. But it is possible to use only one of them at a time. This is due to the fact that only one location is allocated for a union variable irrespective of size.

5. Differentiate structure and Union.

Structure	Union
You can use a struct keyword to define a structure.	You can use a union keyword to define a union.
Every member within structure is assigned a unique memory location.	In union, a memory location is shared by all the data members.
Changing the value of one data member will not affect other data members in structure.	Changing the value of one data member will change the value of other data members in union.
It enables you to initialize several members at once.	It enables you to initialize only the first member of union.

The total size of the structure is the sum of the size of every data member.	The total size of the union is the size of the largest data member.
It is mainly used for storing various data types.	It is mainly used for storing one of the many data types that are available.
It occupies space for each and every member written in inner parameters.	It occupies space for a member having the highest size written in inner parameters.
You can retrieve any member at a time.	You can access one member at a time in the union.

6. Explain with the help of code example array of structures

ARRAY OF STRUCTURES

Whenever some structure that is to be applied to a group of people or items, array of structures can be used.

Example

```
struct Stud
{
int Regno;
char Name[20];
int mark 1, mark2, mark3, Total;
float Avg;
};
struct Stud Student[10];
```

Here Student is an array of 10 elements of type Stud. Thus Student [0] will contain the first set of values, Student[1] will contain the second set of values and so on.

Example:

```
#include<stdio.h>
#include<string.h>
struct student
   {
    int regno;
    char name[15], grade[15];
    int m1, m2, m3, total;
    float avg;
   };
  main()
  {
    struct student s[5];
    int i, n;
    clrscr();
    printf("Enter number of students: ");
    scanf("%d",&n);
for(i=0;i<n;i++)
{
  printf("Enter information about student %d\n",i+1);
  printf("Enter regno\n");
  scanf("%d",&s[i].regno);
  fflush(stdin);
  printf("Enter the name ");
  gets(s[i].name);
  printf("Enter the marks in 3 subjects ");
  scanf("%d%d%d",&s[i].m1, &s[i].m2, &s[i].m3);
  s[i].total=s[i].m1+s[i].m2+s[i].m3;
  s[i].avg=s[i].total/(3.0);
if(s[i].m1>=35&&s[i].m2>=35&&s[i].m3>=35)
 {
  if(s[i].avg>=70)
  strcpy(s[i].grade,"Distinction");
  else
  if(s[i].avg>=60)
  strcpy(s[i].grade,"First class");
  else
  if(s[i].avg>=50)
  strcpy(s[i].grade,"Second class");
  else
  strcpy(s[i].grade,"Pass class");
 }
else
 strcpy(s[i].grade,"Fail");
}
printf("---------------------------------------------------------\n");
printf("regno\tname\tm1\tm2\tm3\ttotal\tavg\tgrade\n");
printf("---------------------------------------------------------\n");
 for(i=0;i<n;i++)
 {
  printf("%d\t%s\t",s[i].regno, s[i].name);
  printf("%d\t%d\t%d\t",s[i].m1, s[i].m2, s[i].m3);
  printf("%d\t%0.2f\t%s\n",s[i].total, s[i].avg, s[i].grade);
 }
getch();
}
```

7. Write a note on nesting of structures

A structure may be defined as a member of another structure. In such cases ,the declaration of the embedded structure must appear before the declaration of the outer structure.

<u>Example</u>

```
    struct date
           {int day;
            int month;
            int year;
          } ;
   struct student
   {
    int Regno;
    char Name[20];
    struct date DOB
  } std;
```

or
it can also be written as

```
       struct Student
    {
           int Regno;
            char Name[20];
            struct date
      struct date
           {int day;
            int month;
            int year;
          }DOB ;
     }  ;
```

If a structure member is itself a structure then a member of the embedded structure can be accessed by writing

    variable. member.submember

Where member refers to the name of the member within the outer structure, and submember refers to the name of the member within the embedded structure.

In the above example the last member of std is std.DOB which is itself a structure of type date. To access the year of date of birth, we have to write as

```
std.DOB.year;
```

8. Explain with example defining a union, declaring union variable and accessing union members

The general form of union is

```
union tagnmae
 {
Data-type member__1;
Data-type member_2;
-----------
Data- type member n;
};
```

Where union is a required keyword and the tag is optional.

A union may be initialized by value of the type of its first member. If a union has been assigned a value of certain type then it is not correct to attempt to retrieve from it a value of a different type.

Example:

```
union Item
{ int x;
  float y;
  char z;
} Alpha;
```

This declares a variable Alpha of type union Item. The union contains three members each with a different data type. But it is possible to use only one of them at a time. This is due to the fact that only one location is allocated for a union variable irrespective of size.

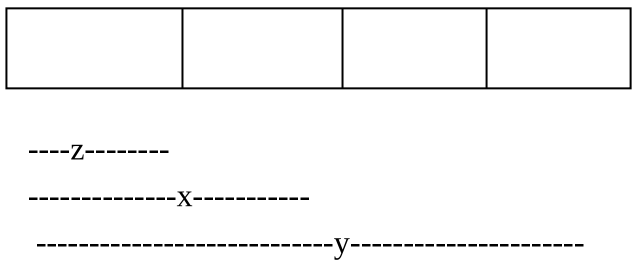


Fig. Sharing of storage locations by Union members

The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union. In the declaration above the member (float type) requires 4 bytes which is the largest among the members. Figures shows how all the three variables share the same address. To access union members we have to use dot operator .

Example :-

```
Alpha.x
Alpha.y
Alpha.z
```

9. What are different basic file operations in C?

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

Let us see the syntax for each of the above operations in a table:

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);

10. What are different mode of operations performed on file?

There are many modes in opening a file. Based on the mode of file, it can be opened for reading or writing or appending the texts. They are listed below.

- r – Opens a file in read mode and sets pointer to the first character in the file. It returns null if file does not exist.
- w – Opens a file in write mode. It returns null if file could not be opened. If file exists, data are overwritten.
- a – Opens a file in append mode. It returns null if file couldn’t be opened.
- r+ – Opens a file for read and write mode and sets pointer to the first character in the file.
- w+ – opens a file for read and write mode and sets pointer to the first character in the file.
- 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.

11. Write a C program to open, write and close the file?

```
/ * Open, write and close a file : */
# 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;
}
```

12. Write a C program to open, read and close the file?

```
/* Open, Read and close a file: reading string by string */

# include <stdio.h>
int main( )
{
         FILE *fp ;
         char data[50] ;
         printf( "Opening the file test.c in read mode" ) ;
         fp = fopen( "test.c", "r" ) ;
         if ( fp == NULL )
         {
                 printf( "Could not open file test.c" ) ;
                 return 1;
         }
         printf( "Reading the file test.c" ) ;
         while( fgets ( data, 50, fp ) != NULL )
         printf( "%s" , data ) ;
         printf("Closing the file test.c") ;
         fclose(fp) ;
         return 0;
}
```

13. Explain getc and putc functions?

The getc and putc Functions

The simplest file *I/O* functions are getc and pute. These are analogous to getehar and putchar functions and handle one character at a time. Assume that a file is opened with mode w and file pointer fp1. Then, the statement

putc(c, fp1);

writes the, character contained in the character variable c to the file associated with FILE pointer fpl. Similarly, getc is used to read a character from a file that has been opened in read mode. For example, the statement

c = getc(fp2);

would read a character from the file whose file pointer is fp2.

The file pointer moves by one character position *for* every operation of getc(). The getc() will return an end-of-file marker EOF, when end of the file has been reached. Therefore, the reading should be terminated when EOF is encountered.

14. Write a C program for using getc and putc functions wrt files?

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

FILE *f1;
 char c;
printf("Data Input\ n\ n");
f1 = fopen("INPUT", “w");        /* Open the file INPUT */
while((c=getchar()) != EOF)    /* Get a character from keyboard*/
putc(c,f1);                      /* Write a character to INPUT file */
fclose(f1);
printf("\nData Output\n\n");
f1 = fopen("'NPUT","r");                   /*.Reopen the file INPUT */
while((c=getc(f1) != EOF)     /* Read a character from INPUT*/
printf("%c",c);
fclose(f1);                    /* Close the file INPUT

}
```

15. Write a note on getw and putw functions wrt files?

The getw and putw are integer-oriented functions. They ,are similar to the getc and putc functions and are used to read and write integer values. These functions would be useful when we deal with only integer data. The general forms of getw and putw are:

putw(integer_variable ,fp);

integer_variable =getw(fp);

<u>Example:</u>

```
#include <stdio.h>
main( )
{
FILE *f1, *f2, *f3;
int number, i;
printf(“Contents of DATA file(Input -1 to terminate)\n\n”);
f1 =fopen(“DATA”, “w”);
while(1)
{
scanf(U%d", &number);
if(number = = -1)
break;
putw(number,f1 );
}
fclose(f1 );
f1 = fopen(“DATA”, “r");
f2 = fopen(“ODD", “w");
f3 = fopen(“EVEN", “w");
while((number = getw(f1)).! = EOF)
 {
  if(number %2 == 0)
  putw(number, f3);
  else
            putw(number, f2);
  fclose(f1 );
  fclose(f2);
  fclose(f3);
  f2 = fopen(“ODD",”r");
  f3 = fopen(“EVEN", “r");
  printf(“\n\nContents of ODD file\n\n");
  while((number = getw(f2)) ! = EOF)
printf("%4d", number);
printf("\n\nContents of EVEN file\n\n");
while((number = getw(f3)) ! = EOF)
  printfr%4d" number);
  fclose(f2);
  fClose(f3);
}
```

16. Write a note on fprinf() and fscanf() function wrt files.

The functions fprintf and fscanf perform I/O operations that are identical to the familar printf and scanf functions, except of course that they work on files. The first argument of these functions is a file pointer which specifies the file to be used. The general form of fprintf is

fprintf(fp, “control string” ,list)

Where fp is pointer associated with a file that has been opened for writing. The control string contains output specifications for variables specified in the list. The list may contain variables , constants and strings.

For example

fprintf(f1, "%S %d %f", name, age, 7.5);

Here, name is an array variable of type char and age is an int variable.

The general format of fscanf is

fscanf(fp, *"control string", list);*

This statement would cause the reading of the items in the *list* from the file specified by *fp,* according to the specifications contained in the *control string.* Example:

fscanf(f2, "%s %d", Item, &quantlty);

Like scanf, fscanf also returns the number of items that are successfully read. When the end of file is reached, it returns the value EOF.

17. Write a C program to illustrate fprintf() and fscanf() function?

```
#include <stdio.h>
main( )
{
FILE *fp;
int number, quantity, i;
float price, value;
char item[10], filename[10];
printf("lnput file name\n");
scanf("%s", filename);
fp = fopen(filename, "w");
printf("lnput inventory data\n\n");
printf("ltem name Number     Price    Quantity\n");
for(i = 1; i <= 3; i++)
{
fscanf(stdin, "%s %d %f %d",
item, &number, &price, &quantity);
fprintf(fp, "%s %d %.2f %d",
item, number, price, quantity);
}
fclose(fp);
fprintf(stdout, "\n\ n");
fp = fopen(filename. "r");
printf("ltem name Number     Price    Quantity          Value\n");
for(i = 1; i <= 3; i++)
{
fscanf(fp, "%s %d %f %d",item,&number,&price,&quantity); value = price' quantity;
fprintf(stdout, "% -8s % 7d %8.2f %8d % 11.2f\n",
item, number. price, quantity, value);
}
 fclose(fp );
}
```