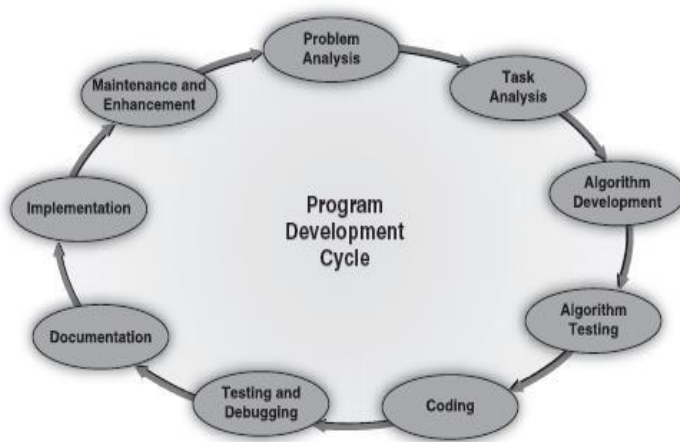


Question Bank
Problem Solving Using C
Paper Code: 21CAC-1/AI

Unit 1

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 step 4
 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.

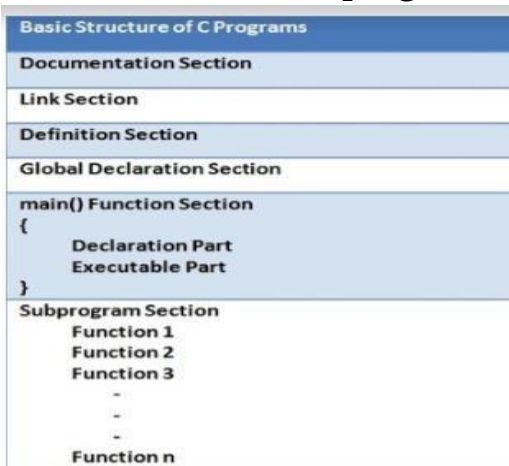
There are some standard graphics that are used in flowchart as following

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.



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'

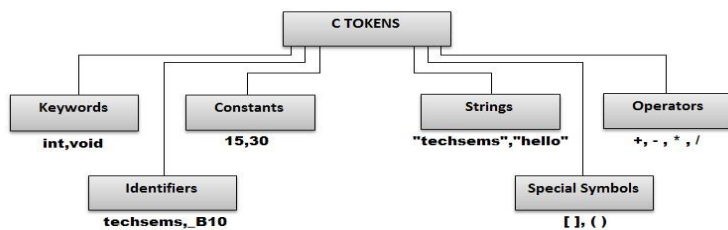
The smallest individual elements or units in a program are called as Tokens. C has following tokens. Identifiers

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 consists of a series of digits representing the whole part of the number, followed by a decimal point, followed by a series of digits representing the fractional part.

Valid Real constants (Fractional): 0.0 -0.1 +123.456 .2 2.

Invalid Real constant: - 1 - a decimal point is missing
1, 2.3 - Illegal character (.)

Single Character Constants

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

Created by RAJKISHOR

Floating point Data Type:

- 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

Created by RAJKISHOR

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

Created by RAJKISHOR

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

Created by RAJKISHOR

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_type***variable_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/expre  
ssion; 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.

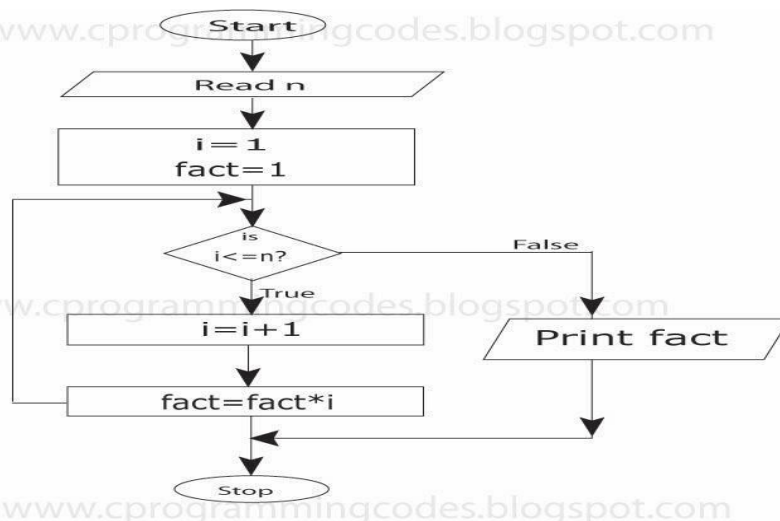
15. Explain the Limitations of Flowcharts

- **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

Descriptive questions

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

Arithmetic Operators:

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
>=	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 is 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
<code>&</code>	Bitwise And
<code> </code>	Bitwise Or
<code>^</code>	Bitwise exclusive Or
<code><<</code>	Shift left
<code>>></code>	Shift Right
<code>~</code>	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= 9-12/3+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,

$(a > (b + c))$ executes then, $(a > (b + c)) \&\& d$ 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 rounding 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 or 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* operand in 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 prints the 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 `true` evaluated to , 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",&var
```

```
2); 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) {
    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

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:

- 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	1
1 2	8
1 2 3	35
1 2 3 4	35

```

#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 );
}

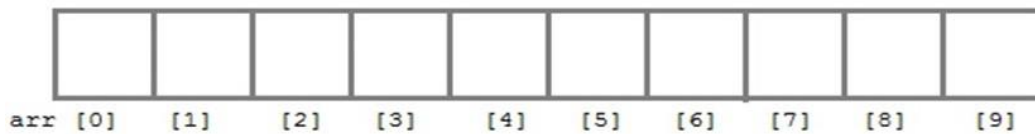
```

6. Differentiate entry-controlled and exit-controlled loops
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,

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.

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*/
```

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 l checks condition at last.

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<stdi
o.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 $M \times N$ (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



Fig. Two - dimensional array of the order $M \times N$

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[0][0] = 1` `mat[0][1] = 2` `mat[1][0] = 3` `mat[1][1] = 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<stdi
o.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, character 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, character 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 justified 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 (string1, 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 str1[50], str2[25];
clrscr();
printf("Enter the first
string\n"); gets( str1);
printf("Enter the second string\n");
gets(str2);
strcat(str1, str2); /* concatenate str1 and str2 */
printf("Concatenated string :\t %s",
str1); 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
(string1, string2);

The strcmp() function compares two strings, character by character, (ASCII comparison) 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 Str1 [] = "aaa", Str2[] = "bbb",
Str3[] = "ccc"; int ptr; clrscr();
ptr = strcmp(Str2, Str1);
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 str1 [ ] = "Computer
Science"; clrscr( );
strcpy(string, str1);
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 str1 [ ] = "Computer
Science"; clrscr( );
strcpy(string, str1);
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 str1 [50], str2[25];
clrscr();
printf("Enter the first
string\n"); gets( str1);
printf("Enter the second string\n");
gets(str2); strcat(str1, str2); 1*
concatenate str1 and str2 */
printf("Concatenated string :\t %s",
str1); 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 <code>str1</code> is greater (in ASCII) than that of <code>str2</code> .
Return Value	Remarks
<0	if the first non-matching character in <code>str1</code> is lower (in ASCII) than that of <code>str2</code> .

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);  
}
```