1) <u>C language :-</u>

1) **What is C language?**

   Ans:- C language is programming language to develop various devices & to control the behaviour of devices.

2) **Why C language?**

   Ans:- C Language is important because

   1) Its middle level language and easy to understand
   2) The code written in C is run much faster as compare to assembly language.

3) **Give structure of C language?**

   Ans:- Structure of C program is as follows:-

   1) **<u>Documentation Section:-</u>**In this section we write description of program, for this we use command line

      // ←------------------------Single Command line

      /* */ ←-------------------Multiple Command line

   2) **<u>Definition Section:-</u>**In this section, header files, macro definition is available. For this # operator is used.
   3) **<u>Global variable Section:-</u>**In this section, global variables are define or declared in which we can use in whole the program.
   4) **<u>main() function section :-</u>**In this section, some executable text is written which implement our logic. It can be started with '{' and end with '}'.
   5) **<u>Subprogram section :-</u>** In this function user defined functions are available and to execute ,we have to call these function in main function.

4) Give compilation stages of C language?

Ans:- There are four stages of compiler

Pre-processor $\longrightarrow$ Compiler $\longrightarrow$ Assembler $\longrightarrow$ Linker

1) **<u>Pre-processor:-</u>** pre-processors are defined with #operator. It executed before actual program. These are very usefull in C program because it compressed C codes, it removes any comments and it add external C code of pre-processor directive. Pre-processor generate intermediate file i.e. "file_name.i".
2) **<u>Compiler:-</u>** Compiler act as converter, means code written by user is converted into binary format which is known by the system. Compiler converts intermediate file to source file i.e "file_name.i" to "file_name.s".
3) **<u>Assembler:-</u>** It convert source file into object file i.e. "file_name.s" to "file_name.o".
4) **<u>Linker:-</u>**It will link all object files & generate executable file i.e. "file.name.exe".

5) What are the escape sequence in C language?

Ans: Escape sequences are used to represent output of program in different way. Escape sequences are

Sr.No.	Code	Meaning
1	\b	Backspace
2	\n	New Line
3	\t	Horizontal tab
4	\v	Vertical Tab
5	\'	Single quote
6	\"	Double quote
7	\?	Question mark
8	\0	Null
9	\f	Form feed
10	\nnn	Octal number representation
11	\xhh	Hexadecimal number representation
12	\r	Carriage return : It will jump to first position of cursor in line
13	\\\\	Bacckslash

6) **What is Keywords?**

   Ans:- Keyword is the word which meaning is already explained to compiler. We can't use these as variables.

7) **Which Keyword are used in C language?**

   Ans:-There are total 32 keywords in C as follows,

   1)Storage class specifier:-

   a) auto b)extern c)static d)register

   2) Statement:-

   a)if b)else c)do d)while e)for f)switch g)continue h)break i)typedef j)goto k)case

   3)Type Specifier:-

   a)int b)char c)float d)double e)long f)short g)signed h)unsigned i)struct j)union k)void

   4)Storage class Modifier:-

   a)const b)volatile

   5) label :- default

   6)sizeof

   7)enum

   8)return

   Note:-All keywords are in lower case.

8) **Explain storage classes in c.**

Ans :- **a)auto :-** Its is default storage of local variable.

b)register :- This storage class used to define local variables that should be store in register instead of RAM. The register should only be used for variables that require quick access. It also noted that defining register does not mean that variable will store in register. It might be stores in register depending upon hardware & implementation restrictions.

c)Static :-

Global Static:- When variable declared as global static ,the scope of this is upto that file only. We can't use this in other file.

Local Static:- When variables declared as local static, the scope of this is upto that function only. We can't use it in other function.

d)extern :- When variable or function declared as extern, the scope of this is present as well as other file also.

9) **Explain actual & formal arguments(parameters) of function.**

Ans:-

Actual arguments :- At the time of function calling , when parameters are passed to function are called "Actual arguments(parameters)".

Formal arguments :- At the time of function declaration , parameters are declared are called 'Formal Arguments(parameters)".

Ex **:-**
```
int sum(int a, int b)  ←--------------- Formal arguments(parameters)
{
   int c;
   c=a+b;
   return c;
}
int main()
{
  int x=10, y=15;
 int result=sum(x,y); ←------------------Actual arguments(parameters)
printf("Addition of two number=%d\n",result);
}
```

10) Explain Memory Layout of C.

Ans:- There are four section in memory layout of C , as follows

1) **Stack Segment :-** In this segment, all local & auto variables, function arguments, return variable, expression & functions are stored. This segment size is variable as per local variables, function parameters, and function calls. This segment grows from a higher address to a lower address.
2) **Heap Segment :-** This is dynamically allocated memory to a process during its run time. This is area of memory allotted for dynamic memory storage such as for malloc() , calloc() & realloc() calls. This segment size is also variable as per user allocation. This segment grows from a lower address to a higher address. Segment size is known by executing the command "size".
3) **Data Segment :-** This section contains the global and static variables. It is represented by .data section and the .bss. The .data section is used to declare the memory region, where data elements are stored for the program. This section cannot be expanded after the data elements are declared, and it remains static throughout the program. The .bss section is also a static memory section that contains buffers for data to be declared later in the program. This buffer memory is zero-filled.

   Initialized Data Segment:- A data segment is a portion of the virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer.

   Uninitialized Data Segment:- This data segment is also called "Block Started by Symbol"[BSS] .It contain uninitialized static & global variables.
4) **Text Segment:-** A text segment, also known as a code segment or simply as text, is one of the sections of a program in an object file or in memory, which contains executable instructions.

   As a memory region, a text segment may be placed below the heap or stack in order to prevent heaps and stack overflows from overwriting it. The text segment is often read-only, to prevent a program from accidentally modifying its instructions.

11) What are data types in C?

a) Basic data types :-

Sr.No.	**Data type**	**Keyword**	**Memory Size**	**Range**
1	Integer	Int	2 or 4 bytes	Signed:- -32768 to +32767 Unsigned:-0 to 65535
2	Character	Char	1 byte	Signed: -128 to +127 Unsigned :- 0 to 255
3	Fractional values	Float	4 bytes	3.4E-38 to 3.4E+38
4	Double fractional values	Double	8 bytes	

b) Modified data types :-

Sr.No.	**Keyword**	**Memory Size**	**Description**
1	long	10 bytes	Double of original
2	short		Half of original
3	Signed		By default sign calue
4	unsigned		Only positive value

12) What are primary,derived & user dedfine data type?

Ans:-

Data Type		
Primary data type	**Derived data type**	**User defined data type**
Integer	Array	Structure
Character	Functions	Union
Fractional	Pointers	Enumeration
Double fractional	Reference	Typedef
Boolean		
Void		

13) What are formatted I/O Console?

Ans:-There are two formatted I/O console as follows,

1)printf() :- It will print the output

Syntax:- printf("statement"); or
printf("statement",varaible);

2)scanf() :-Its used to take input from user

Syntax:-scanf("Format specifier",&variable);

14) What are uformatted I/O Console?

Ans:- **<u>a)Unformatted Input Console:-</u>** getch() ,gectche() ,getchar() & gets() these all are used for taking input from user as character or string.

<u>b)Unformatted Output console:-</u> putchar(),puts() these are all used to print output in the form of character or string.

15) Which are format specifier?

Ans:-

Sr.No.	Format Specifier	Meaning
1	%c	Character
2	%d , %i	Integer
3	%x	Hexadecimal
4	%u	Unsigned int
5	%lu	Unsigned long int
6	%ld	Long integer
7	%lld	Long long integer
8	%f , %e ,%E	Floating point
9	%o	Octal
10	%s	String
11	%lf	Double float

16) What is Volatile?

Ans:- Volatile keyword tells the compiler that don't modify the variable. When variable declared as volatile its value can be change at any time.

17) What is pre-processor and macros? Explain it.

Ans:- Pre-processors & macros are same concept, it define at the beginning of program and can be used in whole program.

Pre-processor used in program because,

a)It improves code readability

b)Easy to modify

Pre-processors:-

Sr.No.	Pre-processor
1	#include
2	#define
3	#undef
4	#if,#elif,#else,#endif,#ifdef,#ifndef
5	#pragma , #line ,#error
6	_DATE_ , _FILE_ , _LINE_ , _TIME_ these are all predefined macros

18) What is Enumeration?

Ans:- Enumeration is user defined data type which is used to assign names to integral constants which makes program easy to read & maintain. Enumeration is also used to define variables of enum type.

The keyword “enum” is used for declare enumeration.

```
Syntax:- enum enum_name{Constant1,Constant2,……….Constant n };
              enum enum_variable;
```

19) What is typedef?

Ans:- To rename the existing data types, “typedef” is used.

```
Syntax:- typedef  existing_data type_name  new_name;
```

20) What is typecasting?

Ans:- To change the data type of variable, typecasting I used.

```
Syntax:- (data type) variable name;
```

21) What is '#' & '##' operator in c?

Ans:- **<u>a)'#' Operator:-</u>** Its also called "Stringize Operator". It sends commands to compiler to convert a token into string. We use this operator at the macro definition. Using stringize operator we can convert some text into string without using any quotes.

```
  e.g. #include<stdio.h>
       #define STR_PRINT(x) #x
       int  main()
      {
       printf(STR_PRINT(Mobiveil is Semiconductor Company));
       return 0;
      }

Output:- Mobiveil is Semiconductor Company
```

<u>b)'##' Operator:-</u> Its also called "Token Pasting Operator". It sends commands to compiler to add or concatenate two tokens into one string.

```
e.g. #include<stdio.h>
     #define STR_CONCAT(x, y) x##y
     main()
  {
   printf("%d", STR_CONCAT(20, 21));
   return 0;
  }

Output:- 2021
```

22) What is sizeof()?

Ans:- sizeof() is operator used to compute size of variable.

Note:-sizeof() is oprator not a function.

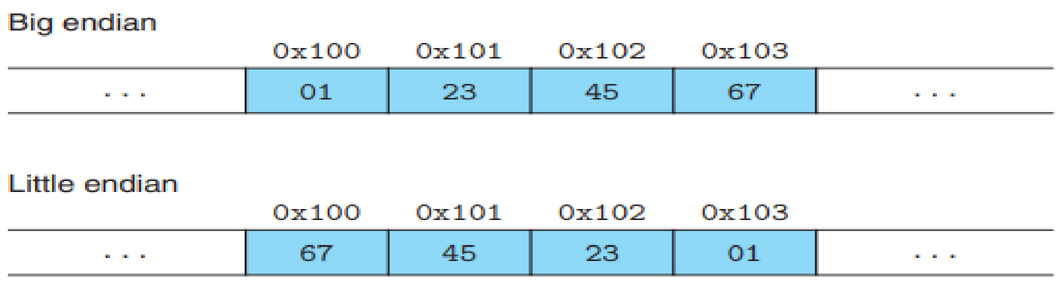
23) What is Little & Big Endian?

Ans:- Its is the process to store multibyte datatypes.

1)Little Endian :- In this process ,LSB byte store first in first memory location & MSB byte store last in last memory location.

2)Big Endian :-In this process, MSB byte store fisrt in fisrt memory location & LSB bytes store last in last memory location.

e.g. x=0X01234567



24) Explain Command Line Argument .

Ans:- It is possible to pass some values from the command line to C programs when it executed. These values are called **command line arguments** and many times they are important for program especially when to control program from outside instead of hard coding those values inside the code.

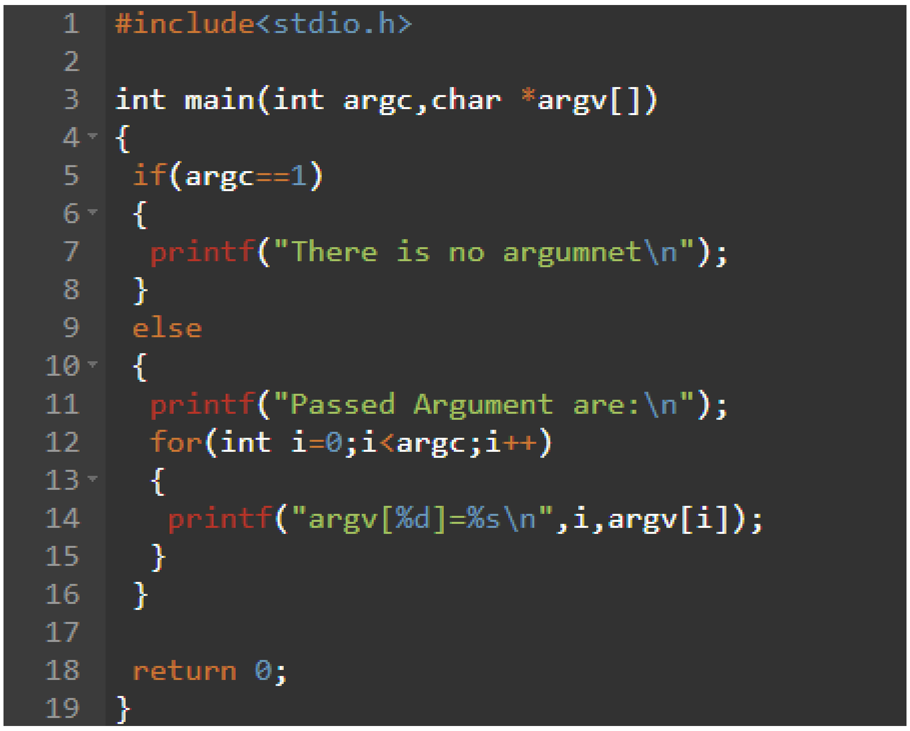
Synatx:-

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

- ➔ **argc(Argument Count)** = is int and stores number of command-line arguments passed by the user including the name of the program. argc should be non negatice.
- ➔ ***argv[] (Argument Vector)** = is array of character pointers listing all the arguments.
- ➔ They are passed to main() function.
- ➔ Arguments are passed using space.
- ➔ They are parameters/arguments supplied to the program when it is invoked.

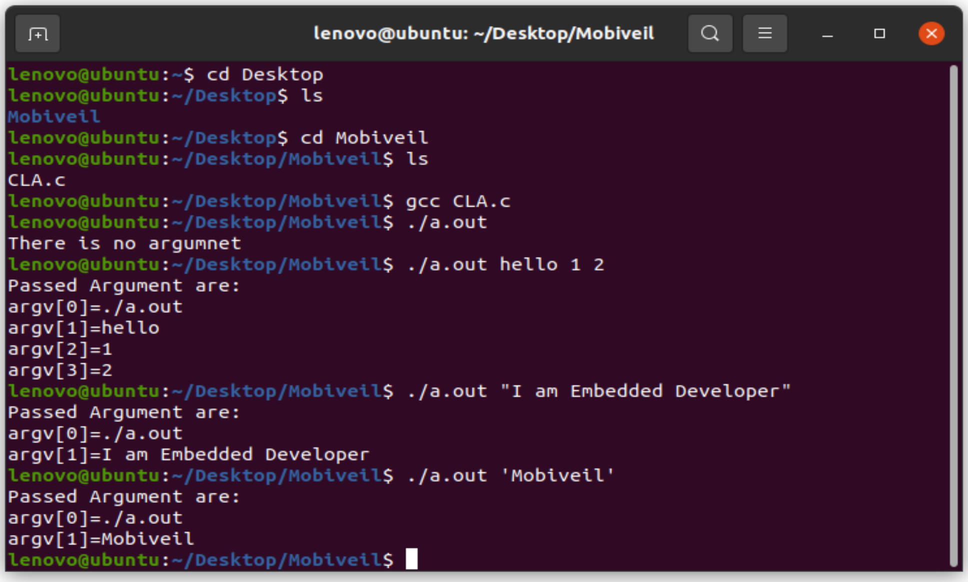
- ➔ They are used to control program from outside instead of hard coding those values inside the code.
- ➔ argv[argc] is a NULL pointer.
- ➔ argv[0] holds the name of the program.
- ➔ argv[1] points to the first command line argument and argv[n] points last argument.
- ➔ When argument is passed within double quote (“ “) or single quote (‘ ‘) , it consider as one argument.

e.g. **Program:-**



```c
#include<stdio.h>

int main(int argc,char *argv[])
{
 if(argc==1)
 {
  printf("There is no argumnet\n");
 }
 else
 {
  printf("Passed Argument are:\n");
  for(int i=0;i<argc;i++)
  {
   printf("argv[%d]=%s\n",i,argv[i]);
  }
 }

 return 0;
}
```

Output:-



```
lenovo@ubuntu:~$ cd Desktop
lenovo@ubuntu:~/Desktop$ ls
Mobiveil
lenovo@ubuntu:~/Desktop$ cd Mobiveil
lenovo@ubuntu:~/Desktop/Mobiveil$ ls
CLA.c
lenovo@ubuntu:~/Desktop/Mobiveil$ gcc CLA.c
lenovo@ubuntu:~/Desktop/Mobiveil$ ./a.out
There is no argumnet
lenovo@ubuntu:~/Desktop/Mobiveil$ ./a.out hello 1 2
Passed Argument are:
argv[0]=./a.out
argv[1]=hello
argv[2]=1
argv[3]=2
lenovo@ubuntu:~/Desktop/Mobiveil$ ./a.out "I am Embedded Developer"
Passed Argument are:
argv[0]=./a.out
argv[1]=I am Embedded Developer
lenovo@ubuntu:~/Desktop/Mobiveil$ ./a.out 'Mobiveil'
Passed Argument are:
argv[0]=./a.out
argv[1]=Mobiveil
lenovo@ubuntu:~/Desktop/Mobiveil$
```

25) What is operator,operand,expression,unary & binary operator?

Ans:- **<u>a)Operator:-</u>** It is the symbol to perform arithmetic or logical operation. Operator are used to manipulate data & varaibles.

<u>b)Operand:-</u>Data items that operators act upon are called operand.

<u>c)Expression:-</u>Operator & operand are combine together to form an expression.

<u>d)Unary Operator:-</u>If operator act upon single operand its called unary operator.

<u>e)Binary operator:-</u>If operator act upon two operand its called binary operator.

26) Which Operator in C?

Ans:- a)Arithmetic Operator:- +, - , *, /, %, a++, a--, ++a, --a

b)Relational Opeartor:- > , < , >=, <=, !=,==

c)Logical Operator :- && , || , !

d)Bitwise Operator :- | , & , ~ , ^

e)Assignment Operator:- += , -= , *= , /= ,%= ,|=, &=,>= ,<=, =

f)Condition Operator :- Its called ternary operator , syntax:- ? :

g)Special Operator:- *pointer to variable ,&returns addres of variable ,sizeof()-returns size of variable.

27) What are decision control statements in C?

Ans:- These statements check whether condition is true or not,if yes then execute the program else jump to next operation.

Decision Control Statements as follows:-

a) if:-

```
syntax:- if(condition)
          {
           Statements;
           }
```

b) if…else

```
syntax:- if(condition)
         {
          Statements;
          }
         else
         {
         Statements;
         }
```

c) if……else if……

```
syntax:- if(condition 1)
         {
          Statements;
          }
         else if(condition 2)
         {
         Statements;
         }
         else
        {
         Statements;
         }
```

d) Nested if

Syntax:- if(condiotion)
```
        {
           if(condition 1)
          {
           Statements;
           }
          else if(condition 2)
          {
          Statements;
          }
          else
         {
          Statements;
          }
        }
```

28) What are loop control statements in C?

Ans:- These execute statements until condition is true, if condition is false it will jump to next operation.

Control Statements are as follows:-

a) while :-

syntax:- while(condition)
```
         {
         Statements;
         }
```

b) do…while :-

syntax:- do
```
           {
         Statements;
           }  while(condition);
```

c) for loop :-
syntax:- for(initialization; loop condition; loop condition)

```
            {
             Statements;
            }
```

- ➔ for(; i<=n;i++):- loop is from zero to n and incerement the value of i till n.
- ➔ for(; i<=n ;):- loop is from zero to n and inrement the value of i till n.
- ➔ for(; ;) :- Infinite loop.

d) switch case:-It's a multi-way decision maker that test whether an expression matches one of number of ultimate values and branches accordingly.
Syntax:-

```
         switch(expression)
 {
  case  constant_1: statements;
                    break;
  case  constant_2: statements;
                    break;

  case  constant_n: statements;
                    break;

              default : statememts;
                    break;

 }
```

29) What is Array?

Ans:- **Definition:-**Array is group of data items having same name & same data type. Array index must start with zero(0).

Declaration of array:-

e.g. int a[5]={1,2,3,4,5};
int a[]={1,2,3,4,5};

Types of array:-

a)Single Dimension Array:- syntax:- data type array_name[array_size];

b)Multi-dimension Array:-Its array of array.
syntax:- data type array_name[size1][size2]…[sizen];

30) What is string?

Ans:- **Definition:-** It's a single dimensional array of characters terminated with '\0'(null) character. Characters within double quote considered as string.

Declaration of String:-

e.g. char a[6]={'H','E','L','L','O','\0'};
char a[6]="Hello";

31) What is structure?

Ans:-**Definition:-** Structure is collection of one or more variable having different data type grouped together under single name.

Syntax:-

```
1)  struct structure_name
  {
   data_type 1;                    }    members of structures
   data_type 2;                    }
    data type n;                   }
  } object name_1, object name_2……,object name_n;


2)  struct structure_name  object name_1,……object name_n
  {
   data_type 1;                    }    members of structures
   data_type 2;                    }
    data type n;                   }
  } ;
```

Variable access method :- To access variable of structure ,dot operator (.) is used.

Size of structure :- size of structure is equal to addition of size of its members.

32) What is structure padding?

Ans:- Structure padding is a concept in C that adds the one or more empty bytes between the memory addresses to align the data in memory.

33) What is union?

Ans:- **Definition:-** Union is collection of one or more variable having different data type grouped together under single name in same memory location.

Syntax:-

```
1)  union union_name
 {
  data_type 1;            }   members of union
  data_type 2;            }
   data type n;           }
 } object name_1, object name_2……,object name_n;

2)  union union_name  object name_1,……object name_n
 {
  data_type 1;            }   members of union
  data_type 2;            }
   data type n;           }
 } ;
```

Variable access method :- To access variable of union,dot operator (.) is used.

Size of union :- size of union is equal to largest size of its members.

34) What is difference between Structure & Union?

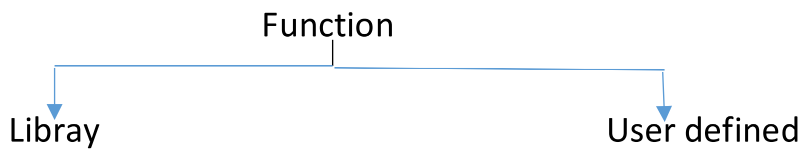
Sr.No.	Structure	Union
1	Size of structure is equal to sum of size of its members.	Size of union is equal to size of largest member.
2	Structure doesn't share memory within its members.	Union share memory within its members.
3	Memory is allocated to each member.	Memory is allocated as per size of largest member.

35) What is function ?explain in detail.

Ans:- **Definition:-** Function is block of statements which perform some kind of coherent task.

- ➔ Functions break large computing tasks into smaller ones. It help in modular development of programs.

Category of Function:-



Library functions:- These functions are pre-written & pre-defined by compiler.

User defined functions:- These functions are defined by programmers while develop the code.

Types of functions:-

a)call by value:- In this, user pass name of actual argument to function. If any changes make to these values within function definition will not affect the actual argument.

b)call by reference:- In this, user pass address of actual argument to function. If any changes make to these values within function definition will affect the actual argument.

c)Recusrion function:- It is the function that calls itself.

d)Functions with no argument & no return values.

e) Functions with argument & no return values.

f) Functions with no argument & return values.

g) Functions with argument & return values.

36) What is Pointer?

Ans:- Pointer is variable to store address of another variable.

37) Why we used Pointer or advantages of Pointer?

Ans:- a) Pointer saves the memory
b)Pointer reduces code length and complexity.
c)Pointer increases processor speed.

38) What is size of pointer or double pointer
(int*,char*,float,double*,long*,short* pointer, data_type)**

Ans:- 4 bytes or 8 bytes depending on Operating system.

39) What is void pointer?

Ans:- Its also called generic pointer and used to point any data type.Here typecasting is required while pointing to variable.
Syntax:- void *variable_name;

40) What is NULL pointer?

Ans:- Its always good practice to assign NULL value to pointer at time of initialization. A pointer that assigned with NULL value called "NULL Pointer".
Syntax:- data_type *variable_name=NULL;

41) What is Wild Pointer?

Ans:-Wild pointer points to unallocated memory or data value which has been deallocated. Wild pointer are declared but not initialized, so they can access random memory location.To avoid this pointer will initialized with NULL value.

42) What is dangling pointer?

Ans:- when memory is deallocated using free() function, but pointers still points to that memory then pointers act as "Dangling Pointer".To avoid this pointer will reinitialized with NULL value.

43) What is meaning of '*' and '&' in pointer?

Ans:- *** :-** It will return value of variable which pointed by pointer.

&:- it will return address of variable which pointed by pointer.

44) Explain arithmetic, increment & Decrement Operation of pointer.

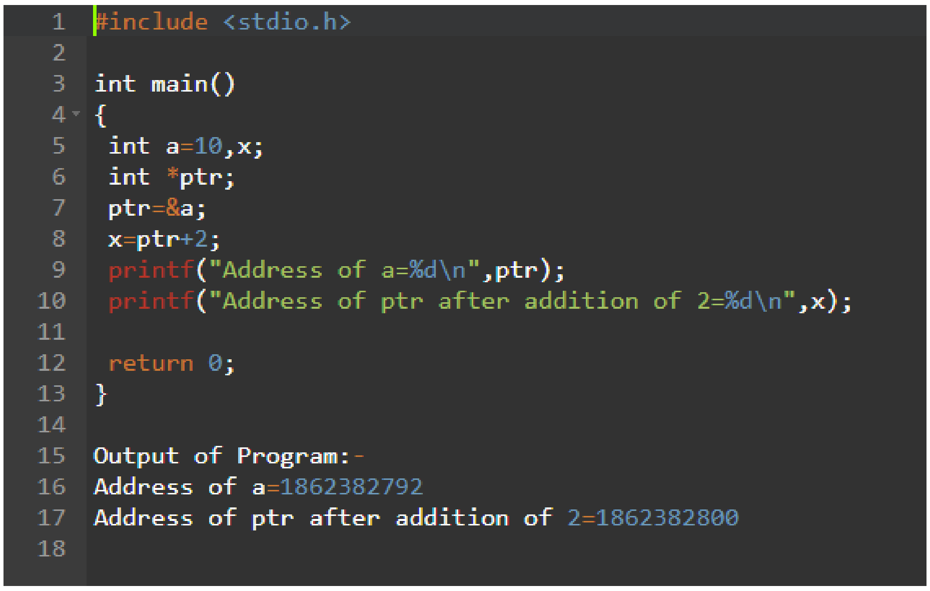
Ans:-

<u>**Arithmetic Operation of Pointer :-**</u>

1) Addition :- When pointer is add by N value , then it will increment address by N value (Pointer will increment depending upon compiler i.e. may be Pointer size 4 bytes or 8 bytes).

   We can't add two pointers

   e.g. Consider Pointer size is 4 bytes in below program.

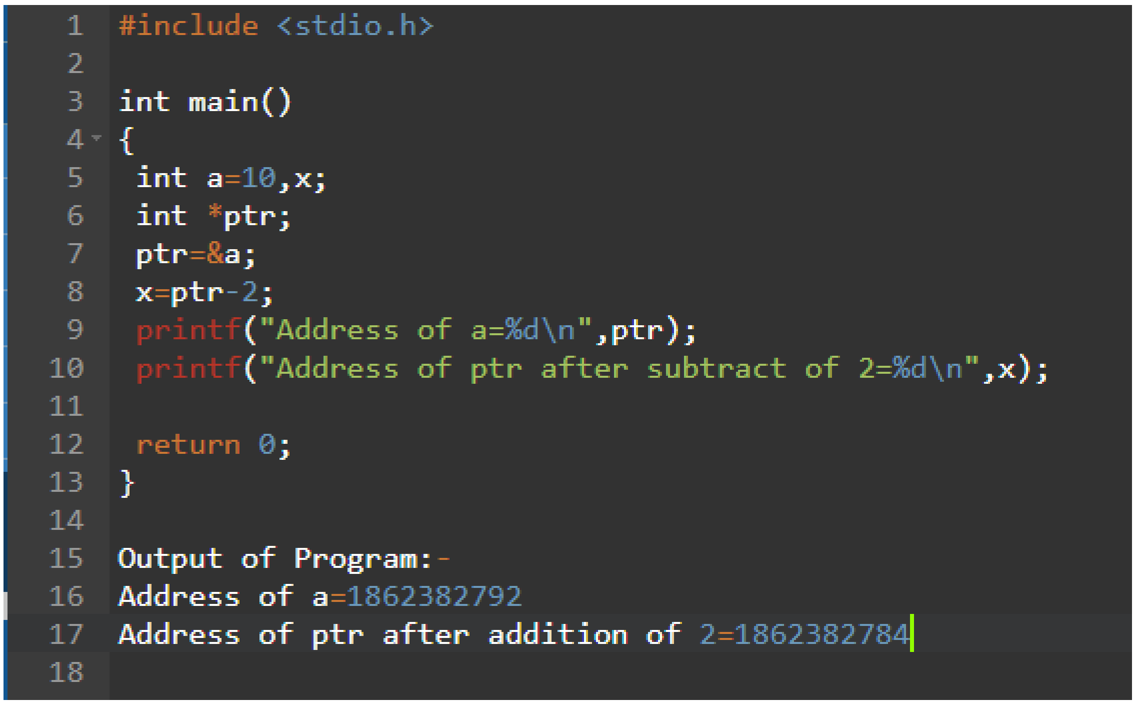


```
#include <stdio.h>

int main()
{
 int a=10,x;
 int *ptr;
 ptr=&a;
 x=ptr+2;
 printf("Address of a=%d\n",ptr);
 printf("Address of ptr after addition of 2=%d\n",x);

 return 0;
}

Output of Program:-
Address of a=1862382792
Address of ptr after addition of 2=1862382800
```

2) Subtraction :- When pointer is subtract by N value , then it will decrement address by N value (Pointer will decrememt depending upon compiler i.e. may be Pointer size 4 bytes or 8 bytes).

   We can't subtract one pointer from another.

   e.g. Consider Pointer size is 4 bytes in below program.



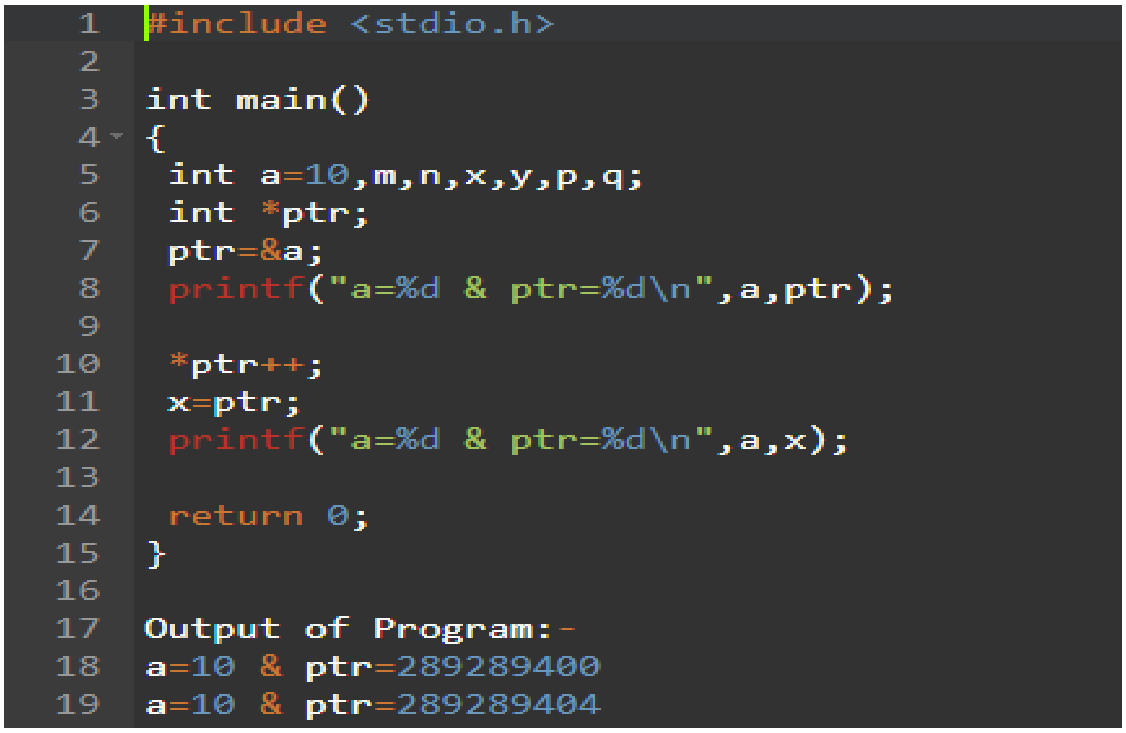
```
#include <stdio.h>

int main()
{
 int a=10,x;
 int *ptr;
 ptr=&a;
 x=ptr-2;
 printf("Address of a=%d\n",ptr);
 printf("Address of ptr after subtract of 2=%d\n",x);

 return 0;
}

Output of Program:-
Address of a=1862382792
Address of ptr after addition of 2=1862382784
```

3) We can't perform Division or Multiplication operation on pointer by N value or by two pointers.

Increment & Decrement Operation of Pointer :-

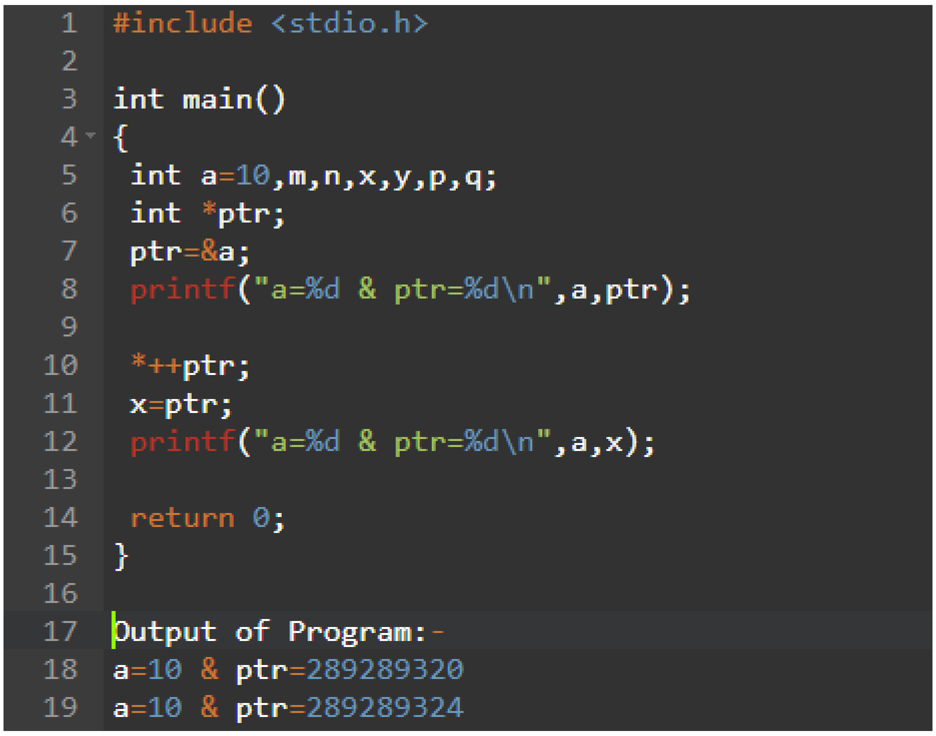
Lets consider ptr is a pointer.

1) ***ptr++** -> It will post increment address of pointer by pointer size.
   e.g. consider pointer size is 4 bytes



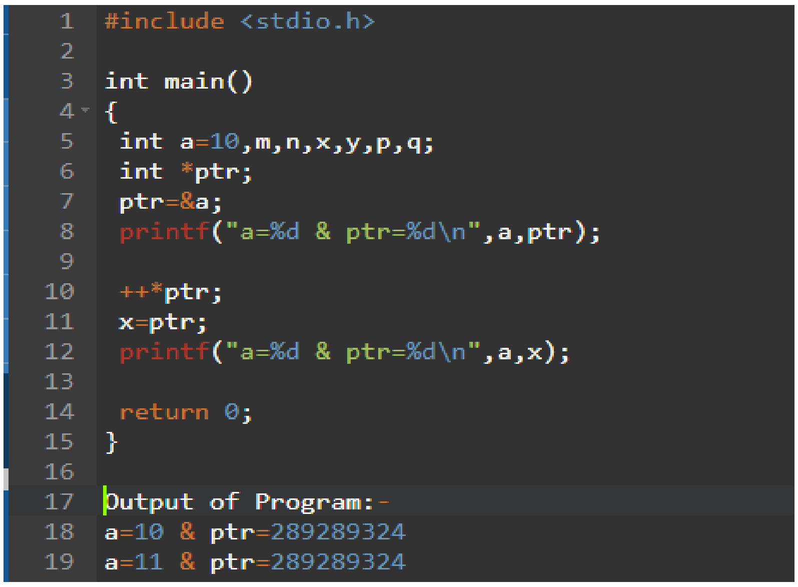
```
#include <stdio.h>

int main()
{
 int a=10,m,n,x,y,p,q;
 int *ptr;
 ptr=&a;
 printf("a=%d & ptr=%d\n",a,ptr);

 *ptr++;
 x=ptr;
 printf("a=%d & ptr=%d\n",a,x);

 return 0;
}

Output of Program:-
a=10 & ptr=289289400
a=10 & ptr=289289404
```

2) ***++ptr** -> It will pre-increment address of pointer by pointer size.
e.g. consider pointer size is 4 bytes



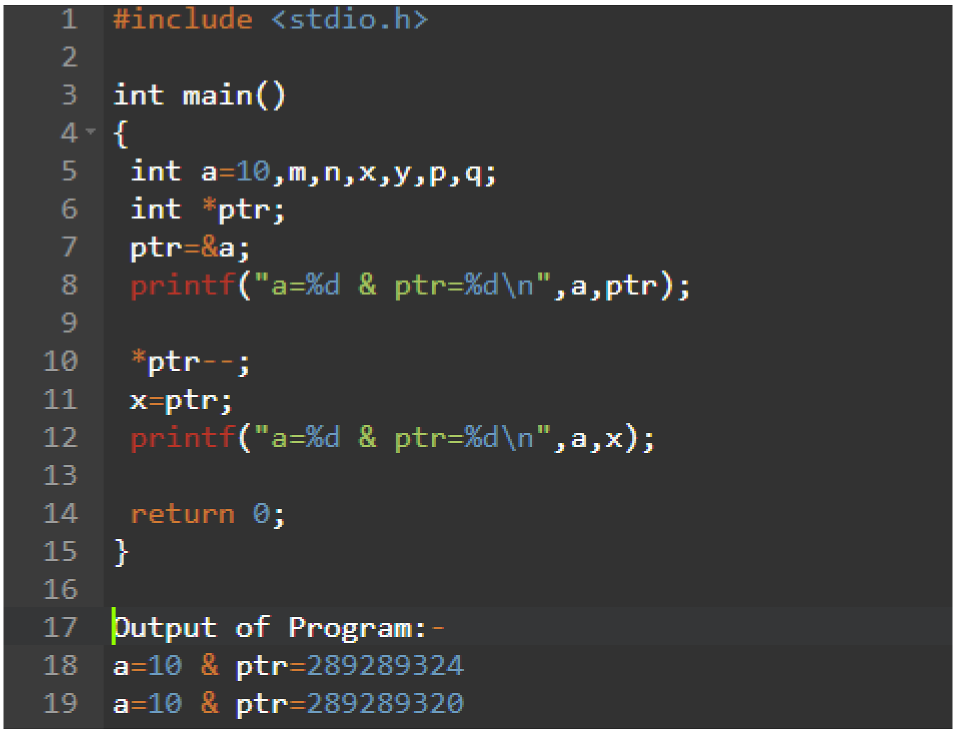
```
#include <stdio.h>

int main()
{
 int a=10,m,n,x,y,p,q;
 int *ptr;
 ptr=&a;
 printf("a=%d & ptr=%d\n",a,ptr);

 *++ptr;
 x=ptr;
 printf("a=%d & ptr=%d\n",a,x);

 return 0;
}

Output of Program:-
a=10 & ptr=289289320
a=10 & ptr=289289324
```

3) **++*ptr** -> It will pre-increment value pointed by pointer.
e.g. consider pointer size is 4 bytes



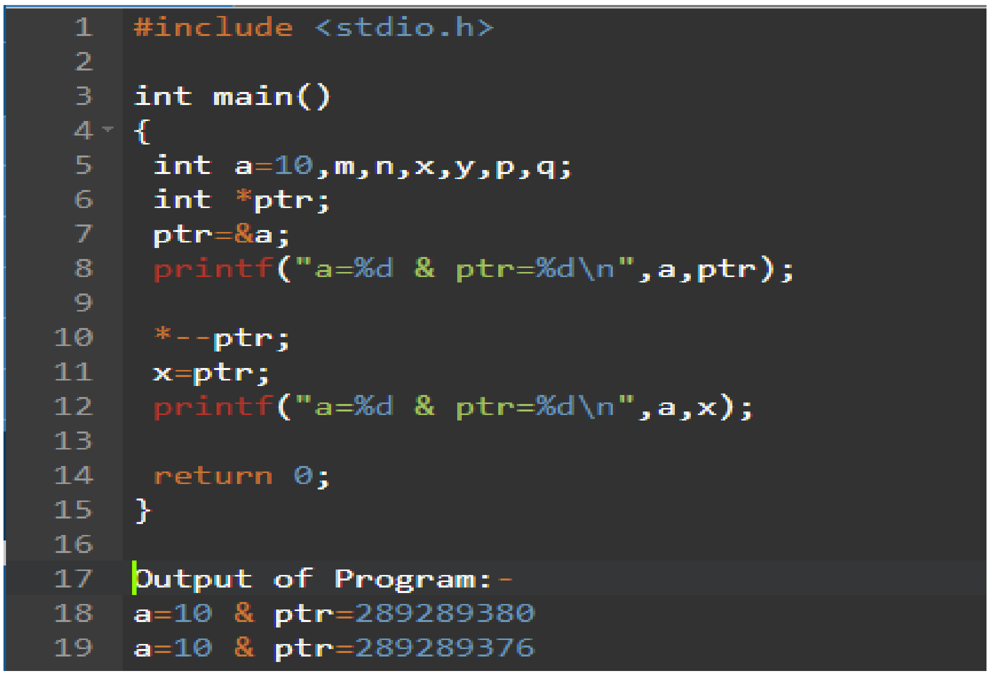
```
#include <stdio.h>

int main()
{
 int a=10,m,n,x,y,p,q;
 int *ptr;
 ptr=&a;
 printf("a=%d & ptr=%d\n",a,ptr);

 ++*ptr;
 x=ptr;
 printf("a=%d & ptr=%d\n",a,x);

 return 0;
}

Output of Program:-
a=10 & ptr=289289324
a=11 & ptr=289289324
```

4) ***ptr --** -> It will post decrement address of pointer by pointer size.
e.g. consider pointer size is 4 bytes



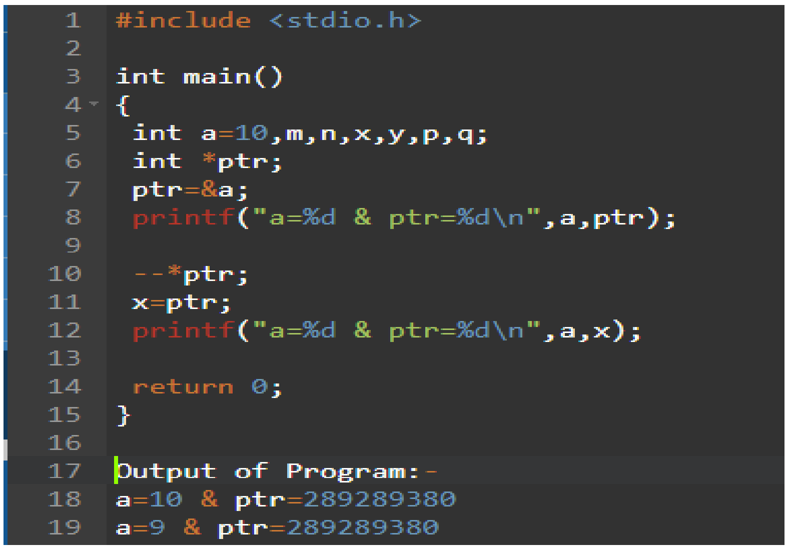
```
#include <stdio.h>

int main()
{
 int a=10,m,n,x,y,p,q;
 int *ptr;
 ptr=&a;
 printf("a=%d & ptr=%d\n",a,ptr);

 *ptr--;
 x=ptr;
 printf("a=%d & ptr=%d\n",a,x);

 return 0;
}

Output of Program:-
a=10 & ptr=289289324
a=10 & ptr=289289320
```

5) ***--ptr** -> It will pre-decrement address of pointer by pointer size.
e.g. consider pointer size is 4 bytes



```
#include <stdio.h>

int main()
{
 int a=10,m,n,x,y,p,q;
 int *ptr;
 ptr=&a;
 printf("a=%d & ptr=%d\n",a,ptr);

 *--ptr;
 x=ptr;
 printf("a=%d & ptr=%d\n",a,x);

 return 0;
}

Output of Program:-
a=10 & ptr=289289380
a=10 & ptr=289289376
```

6) **--*ptr** -> It will pre-decrememt value pointed by pointer.
e.g. consider pointer size is 4 bytes



```
#include <stdio.h>

int main()
{
 int a=10,m,n,x,y,p,q;
 int *ptr;
 ptr=&a;
 printf("a=%d & ptr=%d\n",a,ptr);

 --*ptr;
 x=ptr;
 printf("a=%d & ptr=%d\n",a,x);

 return 0;
}

Output of Program:-
a=10 & ptr=289289380
a=9 & ptr=289289380
```

45) Explain Pointer of constant.

Ans:- Suppose *ptr is pointer

1) **int const *ptr & const int *ptr->** pointer to constant interger. This means that the variable being declared is a pointer, pointing to a constant integer. Effectively, this implies that the pointer is pointing to a value that shouldn't be changed.
int const *ptr & const int *ptr both are same.
2) **int *const ptr ->** constant pointer to integer. This means that the variable being declared is a constant pointer pointing to an integer. Effectively, this implies that the pointer shouldn't point to some other address.
3) **const int *const ptr & int const *const ptr->** constant pointer to constant integer.
This means that the variable being declared is a constant pointer pointing to a constant integer. Effectively, this implies that a constant pointer is pointing to a constant value. Hence, neither the pointer should point to a new address nor the value being pointed to should be changed.
const int *const ptr & int const *const ptr both are same.

46) Explain dynamic memory allocation.

Ans:- When user needs more memory during runtime , then memory is allocated dymanically. There are four functions of dymanic memory allocation as follows.

a) **<u>malloc() :-</u>** memory allocation

➔ It reserve block of memory of specified number of bytes.It returns void pointer which can be casted into pointers of any form.

➔ Its used to allocate space in memory during execution of program.

➔ It doesn't initialize memory allocated during execution.It return or carries garbage value.

➔ It return null pointer if it couldn't able to allocate requested amount of memory.

➔ Syntax:- pointer_variable=(data_type*)malloc(size);

➔ E.g.:-ptr=(int*)malloc(5*sizeof(int));

b) **<u>calloc() :-</u>** continuous location

➔ It allocate memory & initializes all bits to zero.

➔ Its used to allocate space in memory during execution of program.

➔ It return null pointer if it couldn't able to allocate requested amount of memory.

➔ Syntax:- pointer_variable=(data_type*)calloc(n,size);

➔ E.g.:-ptr=(int*)calloc(5,sizeof(int));

c) **<u>realloc():-</u>** Reallocation

➔ If dynamically allocated memory is insufficient or more than required,we can change size of previously allocated memory using realloc() function.

➔ Syntax:-pointer_variable=realloc(pointer_variable,x) ,here x will be allocated using malloc() or calloc() syntax.

➔ E.g.ptr=realloc(ptr,5*sizeof(int());

d) **free():-** Dynamically allocated memory created with either malloc() or calloc() doesn't get free n their own. User need to use free() function to release space.

➔ Syntax:-free(pointer_variable).

47) What is difference between calloc and malloc?

Ans:- malloc() cant allocate continuous memory location and initialize with garbage value.While calloc() allocate continuous memory location with initialize value is zero.

48) What is memory leak ? How it will avoid?

Ans:- Memory leak occurs when programmers create a memory in heap and forget to delete it. To avoid memory leaks, memory allocated on heap should always be freed when no longer needed.

49) Why we use file handling, and how its used in C?

Ans:- File handling used to implement dfferent function using C program.

➔ **File Handling functions:-**

a) fopen() :- Open a file.
b) fprintf() :- printf o/p of file.
c) fscanf() :- read input from file
d) fclose() :- closing a file
e) getc() :- read character from file
f) putc() :-write character to file
g) getw() :-Read integer from file
h) putw() :-Write integer to file
i) fgets() :-Read string from file
j) fputs() :-Write string to file
k) feof() :- detects end of file marker in file.

➔ **File handling specifier:-**

Sr.No.	**Mode**	**Meaning**
1	r	Read a file
2	w	Write & Create a file
3	rb	Read binary file
4	wb	Write & Create binary file
5	a	Append content to file
6	ab	Append content to binary file
7	r+	Read & Write a file
8	w+	Read & Write a file

9	wb+	Read & Write binary file
10	rb+	Read & Write binary file
11	a+	Read & Append file
12	ab+	Read & Append binary file

50) What are inbuilt functions of C?

Ans:- **Arithmetic Functions**

Function	**Use**
abs	Returns the absolute value of an integer
cos	Calculates cosine
cosh	Calculates hyperbolic cosine
exp	Raises the exponential e to the xth power
fabs	Finds absolute value
floor	Finds largest integer less than or equal to argument
fmod	Finds floating-point remainder
hypot	Calculates hypotenuse of right triangle
log	Calculates natural logarithm
log10	Calculates base 10 logarithm
modf	Breaks down argument into integer and fractional parts
pow	Calculates a value raised to a power
sin	Calculates sine
sinh	Calculates hyperbolic sine
sqrt	Finds square root
tan	Calculates tangent
tanh	Calculates hyperbolic tangent

Character classification Functions

Function	**Use**
isalnum	Tests for alphanumeric character
isalpha	Tests for alphabetic character
isdigit	Tests for decimal digit
islower	Tests for lowercase character
isspace	Tests for white space character
isupper	Tests for uppercase character
isxdigit	Tests for hexadecimal digit
tolower	Tests character and converts to lowercase if uppercase
toupper	Tests character and converts to uppercase if lowercase

Data Conversion Functions

Function	**Use**
atof	Converts string to float
atoi	Converts string to int
atoll	Converts string to long
ecvt	Converts double to string
fcvt	Converts double to string
gcvt	Converts double to string
itoa	Converts int to string
ltoa	Converts long to string
strtod	Converts string to double
strtol	Converts string to long integer
strtoul	Converts string to an unsigned long integer
ultoa	Converts unsigned long to string

String Manipulation Functions

Function	**Use**
strcat	Appends one string to another
strchr	Finds first occurrence of a given character in a string
strcmp	Compares two strings
strcmpi	Compares two strings without regard to case
strcpy	Copies one string to another
strdup	Duplicates a string
stricmp	Compares two strings without regard to case(same as strcmpi)
strlen	Finds length of a string
strlwr	Converts a string to lowercase
strncat	Appends a portion of one string to another
strncmp	Compares a portion of one string with portion of another
strncpy	Copies a given number of characters of one string to another
strnicmp	Compares a portion of one string with a portion of another without regard to case
strrchr	Finds last occurrence of a given character in a string
strrev	Reverses a string
strset	Sets all characters in a string to a given character
strstr	Finds first occurrence of a given string in another string
strupr	Converts a string to uppercase

Searching and Sorting Functions

Function	**Use**
bsearch	Performs binary search
lfind	Performs linear search for a given value
qsort	Performs quick sort

I/O Functions

Function	**Use**
Close	Closes a file
fclose	Closes a file
feof	Detects end-of-file
fgetc	Reads a character from a file
fgetchar	Reads a character from keyboard (function version)
fgets	Reads a string from a file
fopen	Opens a file
fprintf	Writes formatted data to a file
fputc	Writes a character to a file
fputchar	Writes a character to screen (function version)
fputs	Writes a string to a file
fscanf	Reads formatted data from a file
fseek	Repositions file pointer to given location
ftell	Gets current file pointer position
getc	Reads a character from a file (macro version)
getch	Reads a character from the keyboard
getche	Reads a character from keyboard and echoes it
getchar	Reads a character from keyboard (macro version)
gets	Reads a line from keyboard
inport	Reads a two-byte word from the specified I/O port
inportb	Reads one byte from the specified I/O port
kbhit	Checks for a keystroke at the keyboard
lseek	Repositions file pointer to a given location
open	Opens a file
outport	Writes a two-byte word to the specified I/O port
outportb	Writes one byte to the specified I/O port
printf	Writes formatted data to screen
putc	Writes a character to a file (macro version)
putch	Writes a character to the screen
putchar	Writes a character to screen (macro version)
puts	Writes a line to file
read	Reads data from a file

rewind	Repositions file pointer to beginning of a file
scanf	Reads formatted data from keyboard
sscanf	Reads formatted input from a string
sprint	Writes formatted output to a string
tell	Gets current file pointer position
write	Writes data to a file

File Handling Functions

Function	**Use**
remove	Deletes file
rename	Renames file
unlink	Deletes file

Directory Control Functions

Function	**Use**
chdir	Changes current working directory
getcwd	Gets current working directory
fnsplit	Splits a full path name into its components
findfirst	Searches a disk directory
findnext	Continues *findfirst* search
mkdir	Makes a new directory
rmdir	Removes a directory

Buffer Manipulation Functions

Function	**Use**
memchr	Returns a pointer to the first occurrence, within a specified number of characters, of a given character in the buffer
memcmp	Compares a specified number of characters from two buffers
memcpy	Copies a specified number of characters from one buffer to another
memicmp	Compares a specified number of characters from two buffers without regard to the case of the characters
memmove	Copies a specified number of characters from one buffer to another
memset	Uses a given character to initialize a specified number of bytes in the buffer

Disk I/O Functions

Function	Use
absread	Reads absolute disk sectors
abswrite	Writes absolute disk sectors
biosdisk	Performs BIOS disk services
getdisk	Gets current drive number
setdisk	Sets current disk drive

Memory Allocation Functions

Function	Use
calloc	Allocates a block of memory
farmalloc	Allocates memory from far heap
farfree	Frees a block from far heap
free	Frees a block allocated with *malloc*
malloc	Allocates a block of memory
realloc	Reallocates a block of memory

Process Control Functions

Function	Use
abort	Aborts a process
atexit	Executes function at program termination
execl	Executes child process with argument list
exit	Terminates the process
spawnl	Executes child process with argument list
spawnlp	Executes child process using PATH variable and argument list
system	Executes an MS-DOS command

Time Related Functions

Function	Use
clock	Returns the elapsed CPU time for a process
difftime	Computes the difference between two times
ftime	Gets current system time as structure
strdate	Returns the current system date as a string
strtime	Returns the current system time as a string
time	Gets current system time as long integer
setdate	Sets DOS date
getdate	Gets system date

Graphics Functions

Function	Use
arc	Draws an arc
ellipse	Draws an ellipse
floodfill	Fills an area of the screen with the current color
getimage	Stores a screen image in memory
getlinestyle	Obtains the current line style
getpixel	Obtains the pixel’s value
lineto	Draws a line from the current graphic output position to the specified point
moveto	Moves the current graphic output position to a specified point
pieslice	Draws a pie-slice-shaped figure
putimage	Retrieves an image from memory and displays it rectangle Draws a rectangle
setcolor	Sets the current color
setlinestyle	Sets the current line style
putpixel	Plots a pixel at a specified point
setviewport	Limits graphic output and positions the logical origin within the limited area

Miscellaneous Functions

Function	Use
delay	Suspends execution for an interval (milliseconds)
getenv	Gets value of environment variable
getpsp	Gets the Program Segment Prefix
perror	Prints error message
putenv	Adds or modifies value of environment variable
random	Generates random numbers
randomize	Initializes random number generation with a random value based on time
sound	Turns PC speaker on at specified frequency
nosound	Turns PC speaker off

DOS Interface Functions

Function	Use
FP_OFF	Returns offset portion of a far pointer
FP_SEG	Returns segment portion of a far pointer
getvect	Gets the current value of the specified interrupt vector
keep	Installs terminate-and-stay-resident (TSR) programs
int86	Issues interrupts
int86x	Issues interrupts with segment register values
intdos	Issues interrupt 21h using registers other than DX and AL
intdosx	Issues interrupt 21h using segment register values
MK_FP	Makes a far pointer
segread	Returns current values of segment registers
setvect	Sets the current value of the specified interrupt vector