C- Interview Questions

1. **What are the features of C language?**

It is middle level structured language developed by Denis Ritchie in 1972 at AT&T Bell Lab.

Features of C:

1. Programs Written in C are efficient and fast.
2. It is a robust language with rich set of built-in functions and operators that can be used to write any complex program.
3. The C compiler combines the capabilities of an assembly language with features of a high-level language.
4. C is highly portable
5. A C program is basically a collection of functions that are supported by C library. We can also create our own function and add it to C library.
6. C language is the most widely used language in operating systems, device drivers and embedded system development today.

2. **Differentiate between C and C++, or Structured and Object oriented language?**

Following are the differences Between C and C++ :

C	C++
1. C is Procedural Language.	**1.** C++ is non Procedural i.e Object oriented Language.
2. No virtual Functions are present in C	**2.** The concept of virtual Functions are used in C++.
3. In C, Polymorphism is not possible.	**3.** The concept of polymorphism is used in C++. Polymorphism is the most Important Feature of OOPS.
4. Operator overloading is not possible in C.	**4.** Operator overloading is one of the greatest Feature of C++.
5. Top down approach is used in Program Design.	**5.** Bottom up approach adopted in Program Design.
6. No namespace Feature is present in C Language.	**6.** Namespace Feature is present in C++ for avoiding Name collision.
7. Multiple Declaration of global variables are allowed.	**7.** Multiple Declaration of global varioables are not allowed.

8. In C • scanf() Function used for Input. • printf() Function used for output.	**8.** In C++ • Cin>> Function used for Input. • Cout<< Function used for output.
9. Mapping between Data and Function is difficult and complicated.	**9.** Mapping between Data and Function can be used using "Objects"
10. In C, we can call main() Function through other Functions	**10.** In C++, we cannot call main() Function through other functions.
11. C requires all the variables to be defined at the starting of a scope.	**11.** C++ allows the declaration of variable anywhere in the scope i.e at time of its First use.
12. No inheritance is possible in C.	**12.** Inheritance is possible in C++
13. In C, malloc() and calloc() Functions are used for Memory Allocation and free() function for memory Deallocating.	**13.**In C++, new and delete operators are used for Memory Allocating and Deallocating.
14. It supports built-in and primitive data types.	**14.** It support both built-in and user define data types.
15. In C, Exception Handling is not present.	**15.** In C++, Exception Handling is done with Try and Catch block.

3. Difference between syntax and logical error.

Ans:

Syntax Error

1-These involves validation of syntax of language.
2-compiler prints diagnostic message.

Logical Error

1-logical error are caused by an incorrect algorithm or by a statement mistyped in such a way that it doesn‘t violet syntax of language.
2-difficult to find.

4. Differentiate between loader and linker.

5. What is the difference between declaring a variable and defining a variable?

ANS: Declaration of a variable in C hints the compiler about the type and size of the variable in compile time. Similarly, declaration of a function hints about type and size of function parameters. No space is reserved in memory for any variable in case of declaration.

Example: int a;
Here variable 'a' is declared of data type 'int'

Defining a variable means declaring it and also allocating space to hold it.
We can say "Definition = Declaration + Space reservation".

Example: int a = 10;
Here variable "a" is described as an int to the compiler and memory is allocated to hold value 10.

6. **What is the difference between initialization and assignment?**

Ans: Initialisation is, you give a value during the creation of the variable, like this:
int i = 10;

Assignment is that you give the variable some value after it got created, like this:

int i; /*Creating the variable, but may have garbage value */
i = 10; /* Assigning 10 to the integer variable i*/

7. **How to write a running C code without main()?**

Ans: Logically it's seems impossible to write a C program without using a main() function. Since every program must have a main() function because:-

- It's an entry point of every C/C++ program.
- All Predefined and User-defined Functions are called directly or indirectly through the main.

Therefore we will use preprocessor(a program which processes the source code before compilation) directive #define with arguments to give an impression that the program runs without main. But in reality it runs with a hidden main function. Let's see how the preprocessor doing their job:-

Hence it can be solved in following ways:-

1. **Using a macro that defines main**

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

 Output: Hello World
 Using Token-Pasting Operator

The above solution has word 'main' in it. If we are not allowed to even write main, we can use token-pasting operator.

```
#include<stdio.h>
#define fun m##a##i##n
```

```
int fun()
{
   printf("Hello World");
   return 0;
}
```

 Output: Hello World

8. Can you modify the constant variable in c?

***YES* ,** we can modify constant variable in C language with the help of Pointers

```
#include<stdio.h>
int main()
{
   int var = 10;
   int *ptr = &var;
   *ptr = (int *)20;
   printf("%d",var);
   return 0;
}
```

This will modify the value of (var) form 10 to 20 .

9. What do you mean by L-Value and R-Value?

Ans: Expressions that refer to memory locations are called "l-value" expressions. An l-value represents a storage region's "locator" value, or a "left" value, implying that it can appear on the left of the equal sign (=). L-values are often identifiers.

- An identifier of integral, floating, pointer, structure, or union type
- A subscript (**[ ]**) expression that does not evaluate to an array
- A member-selection expression (–> or **.**)
- A unary-indirection (*****) expression that does not refer to an array
- An l-value expression in parentheses
- A **const** object (a nonmodifiable l-value)

The term "r-value" is sometimes used to describe the value of an expression and to distinguish it from an l-value. All l-values are r-values but not all r-values are l-values.

10. Differentiate between signed and unsigned variables.

11. How floating point numbers are stored in the memory?

Explanation:

IEEE 754 standard:

- **Float (single precesion i.e 4 bytes)**

(biased with 127 magic no.)

Sign, EEEEEE……E, FFFFFFF…….……F

(1 bit sign) (8 bits for Exponent) (23 bits for mantissa)

- **double (double precesion i.e 8 bytes)**

(biased with 1023 magic no.)

Sign, EEEEEE……E, FFFFFFFF…….……F

(1 bit sign) (11-bits for Exponent) (52-bits for mantissa)

12. What are Bitwise operators? Where are they used?

ANS: In C, following 6 operators are bitwise operators (work at bit-level)

1. **& (bitwise AND)** Takes two numbers as operand and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
2. **| (bitwise OR)** Takes two numbers as operand and does OR on every bit of two numbers. The result of OR is 1 any of the two bits is 1.
3. **^ (bitwise XOR)** Takes two numbers as operand and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
4. **<< (left shift)** Takes two numbers, left shifts the bits of first operand, the second operand decides the number of places to shift.
5. **>> (right shift)** Takes two numbers, right shifts the bits of first operand, the second operand decides the number of places to shift.
6. **~ (bitwise NOT)** Takes one number and inverts all bits of it

Following is example C program.

```
/* C Program to demonstrate use of bitwise operators */
#include<stdio.h>
int main()
{
    unsigned char a = 5, b = 9; // a = 4(00000101), b = 8(00001001)
    printf("a = %d, b = %d\n", a, b);
    printf("a&b = %d\n", a&b); // The result is 00000001
    printf("a|b = %d\n", a|b);  // The result is 00001101
    printf("a^b = %d\n", a^b); // The result is 00001100
    printf("~a = %d\n", a = ~a);   // The result is 11111010
    printf("b<<1 = %d\n", b<<1);  // The result is 00010010
    printf("b>>1 = %d\n", b>>1);  // The result is 00000100
    return 0;
}
```

Output:

a = 5, b = 9

a&b = 1

a|b = 13

a^b = 12

~a = 250

b<<1 = 18

b>>1 = 4

Following are interesting facts about bitwise operators.

1) The left shift and right shift operators should not be used for negative numbers The result of << and >> is undefined behabiour if any of the operands is a negative number. For example results of both -1 << 1 and 1 << -1 is undefined. Also, if the number is shifted more than the size of integer, the behaviour is undefined. For example, 1 << 33 is undefined if integers are stored using 32 bits.

2) The bitwise XOR operator is the most useful operator from technical interview perspective. It is used in many problems. A simple example could be "Given a set of numbers where all elements occur even number of times except one number, find the odd occuring number" This problem can be efficiently solved by just doing XOR of all numbers.

```
// Function to return the only odd occurring element
int findOdd(int arr[], int n) {
   int res = 0, i;
   for (i = 0; i < n; i++)
     res ^= arr[i];
   return res;
}

int main(void) {
   int arr[] = {12, 12, 14, 90, 14, 14, 14};
   int n = sizeof(arr)/sizeof(arr[0]);
   printf ("The odd occurring element is %d ", findOdd(arr, n));
   return 0;
}
// Output: The odd occurring element is 90
```

3) The bitwise operators should not be used in-place of logical operators.
The result of logical operators (&&, || and !) is either 0 or 1, but bitwise operators return an integer value. Also, the logical operators consider any non-zero operand as 1. For example consider the following program, the results of & and && are different for same operands.

```
int main()
{
  int x = 2, y = 5;
  (x & y)? printf("True ") : printf("False ");
  (x && y)? printf("True ") : printf("False ");
  return 0;
}
// Output: False True
```

4) The left-shift and right-shift operators are equivalent to multiplication and division by 2 respectively.
As mentioned in point 1, it works only if numbers are positive.

```
int main()
{
   int x = 19;
```

```
   printf ("x << 1 = %d\n", x << 1);
   printf ("x >> 1 = %d\n", x >> 1);
   return 0;
}
// Output: 38 9
```

5) The & operator can be used to quickly check if a number is odd or even

The value of expression (x & 1) would be non-zero only if x is odd, otherwise the value would be zero.

```
int main()
{
   int x = 19;
   (x & 1)? printf("Odd"): printf("Even");
   return 0;
}
// Output: Odd
```

6) The ~ operator should be used carefully

The result of ~ operator on a small number can be a big number if result is stored in a unsigned variable. And result may be negative number if result is stored in signed variable (assuming that the negative numbers are stored in 2's complement form where leftmost bit is the sign bit)

```
// Note that the output of following program is compiler dependent
int main()
{
   unsigned int x = 1;
   printf("Signed Result %d \n", ~x);
   printf("Unsigned Result %ud \n", ~x);
   return 0;
}
/* Output:
Signed Result -2
Unsigned Result 42949
67294d */
```

13. Which bitwise operator is suitable for checking whether a particular bit is ON or OFF? Bitwise AND operator.

AND operator.

Suppose in byte that has a value 10101101 . We wish to check whether bit number 4 is ON (1) or OFF(0) .

Since we want to check the bit number 4,

1<<(bit-1) i.e 1<<(4-1) 1000

now take bitwise and of num and bit

10101101	**original bit pattern**

00001000	**AND mask**

00001000	**resulting bit pattern**
---------	

14. Which bitwise operator is suitable for turning OFF a particular bit in a number?
Bitwise AND operator (&), one's complement operator(~)

Bitwise AND operator (&), one's complement operator(~)

Explanation:

Consider,
int x=23; ---------- 000000000010111

suppose bit=3
1<<(bit-1)---------1<<(3-1)---------- 000000000000100
~(1<<(bit-1) ---------- 111111111111011
Do x=(x) & ~ (1<< (bit-1));

000000000000010111
&111111111111111011

10011

Thus the 3rd bit is unset.

15. Subtract two numbers without using '-' operator.

Ans: c= a+(~b)

16. Write a c program without using any semicolon to print Hello world.

Solution: 1
void main(){ if(printf("Hello world")){ } }
Solution: 2
void main(){ while(!printf("Hello world")){ } }
Solution: 3
void main(){ switch(printf("Hello world")){ } }

17. What is the meaning of scope of a variable?

Ans: Meaning of scope is to check either variable is alive or dead. Alive means data of a variable has not destroyed from memory. Up to which part or area of the program a variable is alive, that area or part is known as scope of a variable. In the above figure scope of variable a represented outer red box i.e. whole program. Note: If any variable is not visible it may have scope i.e. it is alive or may not have scope. But if any variable has not scope i.e. it is dead then variable must not to be visible. There are four type of scope in c: 1. Block scope. 2. Function scope. 3. File scope. 3. Program scope.

18. What is void data type? Tell me any three use of void data type.

Linguistic meaning of void is nothing. Size of void data type is meaningless question.

What will be output of following c code?

```
#include<stdio.h>
 int main(){
 int size;
 size=sizeof(void);
 printf("%d",size);
 return 0; }
```

Output: Compilation error

If we will try to find the size of void data type complier will show an error message "not type allowed". We cannot use any storage class modifier with void data type so void data type doesn't reserve any memory space. Hence we cannot declare any variable as of void type i.e. void num; //compilation error

Use of void data type

- To declare generic pointer
- As a function return type
- As a function parameter.

19. What is difference between implicit and explicit type conversions?

ANS: When variables and constants of different types are combined in an expression then they are converted to same data type. The process of converting one predefined type into another is called type conversion.

Type conversion in c can be classified into the following two types:

Implicit Type Conversion

- When the type conversion is performed automatically by the compiler without programmers intervention, such type of conversion is known as **implicit type conversion** or **type promotion**.
- The compiler converts all operands into the data type of the largest operand.
- It should be noted that the final result of expression is converted to type of variable on left side of assignment operator before assigning value to it.
- Also, conversion of float to int causes truncation of fractional part, conversion of double to float causes rounding of digits and the conversion of long int to int causes dropping of excess higher order bits.

Explicit Type Conversion

- The type conversion performed by the programmer by posing the data type of the expression of specific type is known as explicit type conversion.
- The explicit type conversion is also known as **type casting**.

Type casting in c is done in the following form:

(data_type)expression;

where, *data_type* is any valid c data type, and *expression* may be constant, variable or expression.

For example,

```
1       x=(int)a+b*d;
```

The following rules have to be followed while converting the expression from one type to another to avoid the loss of information:

1. All integer types to be converted to float.
2. All float types to be converted to double.
3. All character types to be converted to integer.

20. Do you know any post tested loop in c?
Ans: do while loop.

21. What is cyclic property of data type in c?

```
#include<stdio.h> int main()
{     signed char c1=130;
 signed char c2=-130;
printf("%d  %d",c1,c2);
```

```
return 0; }
 Output: -126   126 (why?)
```

This situation is known as overflow of signed char.

Range of unsigned char is -128 to 127. If we will assign a value greater than 127 then value of variable will be changed to a value if we will move clockwise direction as shown in the figure according to number. If we will assign a number which is less than -128 then we have to move in anti-clockwise direction.

22. Differentiate between if and switch statement. Which one is better?

Basis for Comparison	if-else	Switch
Basic	Which statement will be executed depend upon the output of the expression inside if statement.	Which statement will be executed is decided by user.
Expression	if-else statement uses multiple statement for multiple choices.	switch statement uses single expression for multiple choices.
Testing	if-else statement test for equality as well as for logical expression.	switch statement test only for equality.
Evaluation	if statement evaluates integer, character, pointer or floating-point type or boolean type.	switch statement evaluates only character or integer value.
Sequence of Execution	Either if statement will be executed or else statement is executed.	switch statement execute one case after another till a break statement is appeared or the end of switch statement is reached.
Default Execution	If the condition inside if statements is false, then by default the else statement is executed if created.	If the condition inside switch statements does not match with any of cases, for that instance the default statements is executed if created.
Editing	It is difficult to edit the if-else statement, if the nested if-else statement is used.	It is easy to edit switch cases as, they are recognized easily.

23. Why should we avoid goto statement in C program?

Reasons to avoid goto statement:

- Though, using goto statement give power to jump to any part of program, using goto statement makes the logic of the program complex and tangled.
- In modern programming, goto statement is considered a harmful construct and a bad programming practice.

- The goto statement can be replaced in most of C program with the use of break and continue statements.
- In fact, any program in C programming can be perfectly written without the use of goto statement.
- All programmer should try to avoid goto statement as possible as they can.

24. Differentiate between break and continue.

break	continue
A break can appear in both switch and loop (for, while, do) statements.	A continue can appear only in loop (for, while, do) statements.
A break causes the switch or loop statements to terminate the moment it is executed. Loop or switch ends abruptly when break is encountered.	A continue doesn't terminate the loop, it causes the loop to go to the next iteration. All iterations of the loop are executed even if continue is encountered. The continue statement is used to skip statements in the loop that appear after the continue.
The break statement can be used in both switch and loop statements.	The continue statement can appear only in loops. You will get an error if this appears in switch statement.
When a break statement is encountered, it terminates the block and gets the control out of the switch or loop.	When a continue statement is encountered, it gets the control to the next iteration of the loop.
A break causes the innermost enclosing loop or switch to be exited immediately.	A continue inside a loop nested within a switch causes the next loop iteration.

25. What is modular programming and its significant advantages?

Ans: Modular programming also called as **stepwise refinement** or **top-down design** is a programming approach that breaks down program functions into modules.

Advantages:

1. Faster development.
2. Several programmers can work on individual programs at the same time.
3. Easy debugging and maintenance.
4. Easy to understand as each module works independently to another module.
5. Less code has to be written.
6. The scoping of variables can easily be controlled.
7. Modules can be re-used, eliminating the need to retype the code many times.

26. What is the purpose of main() function?

Ans:

The main() function is :

- The first function to start a program

- Returns int value to the environment which called the program
- It can be called recursively.
- It is a user defined function, except the name
- Like other functions, main(0 function can receive arguments. It has a) argument count and b) argument vector(string argument)

27. Explain command line arguments of main function? **What do the 'c' and 'v' in argc and argv stand for?**

Ans: We can also give command-line arguments in C and C++. Command-line arguments are given after the name of the program in command-line shell of Operating Systems.

To pass command line arguments, we typically define main() with two arguments :

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

The c in argc(argument count) stands for the number of command line argument the program is invoked with and v in argv(argument vector) is a pointer to an array of character string that contain the arguments.

28. What is default return type of a function?
Ans: int

29. What are the differences between formal arguments and actual arguments of a function?

Ans: F**ormal parameter** — the identifier used in a method to stand for the value that is passed into the method by a caller.

For example, amt is a formal parameter of function void FUN(int amt)

actual parameter — the actual value that is passed into the method by a caller.

For example, the 200 used when FUN() is called is an actual parameter. Actual parameters are often called **arguments**

30. Difference between pass by reference and pass by value?

Ans: Pass by reference passes a pointer to the value. This allows the callee to modify the variable directly.Pass by value gives a copy of the value to the callee. This allows the callee to modify the value without modifying the variable. (In other words, the callee simply cannot modify the variable, since it lacks a reference to it.)

31. In header files whether functions are declared or defined?

Ans: Functions are declared within header file. That is function **prototypes** exist in a header file,not function bodies. (header file also contains some built in constants, macros ans structres)

- function bodies are defined in library (.DLL)

32. Can a function return more than one value/address by using return keyword? If No, then how it is possible.

Ans: No, a function cannot return multiple values by using return keyword. But is can be possible using call by reference technique.

Example:

```
int sum_diff(int a,int b,int *s,int *d)
  {
*s=a+b;
*d=a-b;
   }
main()
 {   int a=20,b=10,sum,diff;
sum_diff(a,b,&sum,&diff);
printf("Sum = %d and Difference = %d\n",sum,diff);
 }
```

- **Output: Sum=30 and Difference=10**

33. What is function pointer?

Function pointer, as the name suggests, points to a function. We can declare a function pointer and point to a function. After that using that function pointer we can call that function. Let's say, we have a function Hello which has definition like this -

```
void Hello(int)
```

The pointer to that function will look like -

```
void (*ptr)(int);
```

Here, ptr is a function pointer that can point to function with no return type and one integer argument. After declaration, we can point to function Hello like this -

```
void (*ptr)(int) = NULL;
ptr = Hello;
```

After that we can call that function like this -

```
(*ptr)(30);
```

34. Describe about storage allocation and scope of global, extern, static, local and register variables?

Ans:

Globals have application-scope. They‘re available in any compilation unit that includes an appropriate declaration (usually brought from a header file). They‘re stored wherever the linker puts them, usually a place called the ―BSS segment.‖ Extern? This is essentially ―global.‖

Static: Stored the same place as globals, typically, but only available to the

compilation unit that contains them. If they are block-scope global, only available within that block and its subblocks.
Local: Stored on the stack, typically. Only available in that block and its subblocks. (Although pointers to locals can be passed to functions invoked from within a scope where that local is valid.)
Register: See tirade above on ―local‖ vs. ―register.‖ The only difference is that the C compiler will not let you take the address of something you've declared as ―register.‖

35. What is recursion?

Ans: A recursion function is one which calls itself either directly or indirectly it must halt at a definite point to avoid infinite recursion.

- Types of Recursion:
- Direct
- Indirect
- Tail (No pending operation after recursive call)
- Non tail (some pending operation after recursive call)
- Binary (e.g fibonacci series program)

36. What do you mean by Stack Overflow?

```
void add ( int a, int b )
{
    int c;
    c =  a + b;
    add (1,1);
}
```

What is the result of above function?
a. Sum of a,b,1
b. Results in Buffer Overflow
c Results in Compiler Error
d Results in Stack Overflow

When a function is called recursively, sometimes infinite recursions occur which results in STACK OVERFLOW. What does it mean? Well when a function is called,

1. **First it will evaluate actual parameter expressions.**
2. Then, memory is allocated to local variables.
3. Store caller's current address of execution (return address of the current function) and then continue execute the recursive call.
4. Then it executes rest of the function body and reaches end and returns to the caller's address.

Now when a function is infinitely called (recursively) without a proper condition to check its recursive, then only first 3 steps keep executing and function will never reach step 4 to finish execution and return to previous function. In this way, function will keep allocating memory

and at some point of time it will go out of memory or reaches stack limit and will never be able to accommodate another function and hence crashes. This is called stack overflow.

37. What is array? Discuss its types.

Ans: Array is an homogeneous data structures in which same types of data elements are stored in a contiguous memory locations.

Types: 1-D, 2-D, multi-D

38. How to pass an array to a function?

Passing an entire one-dimensional array to a function

While passing arrays as arguments to the function, only the name of the array is passed (,i.e, starting address of memory area is passed as argument).

C program to pass an array containing age of person to a function. This function should find average age and display the average age in main function.

```
#include <stdio.h>
float average(float age[]);

int main()
{
   float avg, age[] = { 23.4, 55, 22.6, 3, 40.5, 18 };
   avg = average(age); /* Only name of array is passed as argument. */
   printf("Average age=%.2f", avg);
   return 0;
}

float average(float age[])
{
   int i;
   float avg, sum = 0.0;
   for (i = 0; i < 6; ++i) {
      sum += age[i];
   }
   avg = (sum / 6);
   return avg;
}
```

Output

```
Average age=27.08
```

39. Is there any bound checking in array in C?

Ans: No. C does not provide any bound checking of array, neither at compile time nor at run time.

40. What is string? Discuss various ways for string handling.

Recall that a C-string is implemented as a null-terminated array of type char

No built-in string type in C. Must use character arrays

NOT every character array is a C-string. Only when terminated with the null-character

String literals in code ("Hello World", "John Smith") are implemented as constant C-strings.

String in C can be handled by using two ways:

1. Using character arry.
   Char str[]=”Hello”;
2. Using character pointer.
   Char *str=”Hello”;

41. Discuss any five string handling functions.

Function	Work of Function
strlen()	Calculates the length of string
strcpy()	Copies a string to another string
strcat()	Concatenates(joins) two strings
strcmp()	Compares two string
strlwr()	Converts string to lowercase
strupr()	Converts string to uppercase

42. What is Pointer?

Ans: Pointers are variables which stores the address of another variable of its specified type.

Syntax to declare:

DataType *PointerNmae;

Eg. Int *ptr;

Here (*) is called indirection operator.

43. When should we use pointers in a C program?

Ans:

1. To get address of a variable

2. For achieving pass by reference in C: Pointers allow different functions to share and modify their local variables.

3. To pass large structures so that complete copy of the structure can be avoided.

4. To implement "linked" data structures like linked lists and binary trees.

5. String Handling.

44. What is NULL pointer?

Ans: NULL is used to indicate that the pointer doesn't point to a valid location. Ideally, we should initialize pointers as NULL if we don't know their value at the time of declaration. Also, we should make a pointer NULL when memory pointed by it is deallocated in the middle of a program.

int *ptr=NULL;

int *ptr=0;

45. What is Wild Pointer?

Ans: The pointer which is not initialized is wild pointer.

```
main()
{
int *p;
printf("%d",*p);
}
```

Output: garbage

46. What is Dangling pointer?

Ans: Dangling Pointer is a pointer that doesn't point to a valid memory location. Dangling pointers arise when an object is deleted or deallocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the deallocated memory. Following are examples.

```
// EXAMPLE 1
int*ptr = (int*)malloc(sizeof(int));
.....
.....
free(ptr);

// ptr is a dangling pointer now and operations like following are invalid
*ptr = 10;  // or printf("%d", *ptr);
// EXAMPLE 2
int*ptr = NULL
```

```
{
  intx  = 10;
  ptr = &x;
}
// x goes out of scope and memory allocated to x is free now.
// So ptr is a dangling pointer now.
```

47. What is memory leak? Why it should be avoided

Ans: Memory leak occurs when programmers create a memory in heap and forget to delete it. Memory leaks are particularly serious issues for programs like daemons and servers which by definition never terminate.

```
/* Function with memory leak */
#include <stdlib.h>

voidf()
{
  int*ptr = (int*) malloc(sizeof(int));

  /* Do some work */

  return; /* Return without freeing ptr*/
}
```

48. In C, why is the void pointer useful? When would you use it?

Ans: The void pointer is useful because it is a generic pointer that can hold the address of any type of variable.

But before dereferencing a void pointer type casting is required because it does not point to any object, just holds the base address of the memory.

Eg.
```
char ch='A';
int i=10;
void *ptr;
ptr=&ch;
printf("%c", *(char*)ptr);
ptr=&i;
printf("%d", *(int*)ptr);
```

49. Discuss on pointer arithmetic?

Ans:

1. **Two pointers can't be added multiplied and divided.**
2. **Subtraction of two pointers is allowed but only in case of array.**
3. **A constant value can be added or subtracted(but it will be multiplied by scale factor of the pointer)**
4. **Ptr++, --ptr, comparison i.e ptr1==ptr2 are allowed.**

50. What is a far pointer? Where we use it?

Ans: In large data model (compact, large, huge) the address B0008000 is acceptable because in these

model all pointers to data are 32bits long. If we use small data model(tiny, small, medium) the above address won‘t work since in these model each pointer is 16bits long. If we are working in a small data model and want to access the address B0008000 then we use far pointer. Far pointer is always treated as a 32bit pointer and contains a segment address and offset address both of 16bits each. Thus the address is represented using segment : offset format B000h:8000h. For any given memory address there are many possible far address segment : offset pair. The segment register contains the address where the segment begins and offset register contains the offset of data/code from where segment begins.

51. What is a huge pointer?

Ans: Huge pointer is 32bit long containing segment address and offset address. Huge pointers are normalized pointers so for any given memory address there is only one possible huge address segment: offset pair. Huge pointer arithmetic is doe with calls to special subroutines so its arithmetic slower than any other pointers.

52. What is near pointer?

Ans: A near pointer is 16 bits long. It uses the current content of the CS (code segment) register (if the pointer is pointing to code) or current contents of DS (data segment) register (if the pointer is pointing to data) for the segment part, the offset part is stored in a 16 bit near pointer. Using near pointer limits the data/code to 64kb segment.

53. Discuss its significant advantages over static memory allocation.

•Static MA: working with variables arrays whose size have been fixed at compile time.

It has problem of array bound and memory leak.

•DMA: Creating memories as well as releasing memories at run time (in Heap) using DMA functions (malloc, calloc, realloc, free).

Better utilization of memory.

54. Difference between array name and a pointer variable?

Ans: A pointer variable is a variable where as an array name is a fixed address and is not a variable. A

pointer variable must be initialized but an array name cannot be initialized. An array name being a constant value , ++ and — operators cannot be applied to it.

55. Differentiate between a constant pointer and pointer to a constant?

Ans: const char *p; //pointer to a const character. char const *p; //pointer to a const character. char * const p; //const pointer to a char variable.

const char * const p; // const pointer to a const character.

56. What is pointer to a pointer?

Ans: If a pointer variable points another pointer value. Such a situation is known as a pointer to a pointer.
Example:

```
int
*p1,**p2,
v=10;
P1=&v;
p2=&p1;
```

Here p2 is a pointer to a pointer.

57. What is a structure?

Ans: Structure is an user defined data type which represents several different data members in a single unit. It represents a record.

e.g: Student={Rollno, name, branch, marks}
Employee={Id,name, dept, salary}

Syntax:

```
struct Students
{
int Rollno;
char name[20];
};
```

58. What is a union?

Ans: Union is a collection of heterogeneous data type but it uses efficient memory utilization technique by allocating enough memory to hold the largest member. Here a single area of memory contains values of different types at different time. A union can never be initialized.

59. What are the differences between structures and union?

Ans:

A structure variable contains each of the named members, and its size is large enough to hold all the members. Structure elements are of same size.

A union contains one of the named members at a given time and is large enough to hold the largest member. Union element can be of different sizes.

60. What are the differences between structures and arrays?

Ans: Structure is a collection of heterogeneous data type but array is a collection of homogeneous data types.

Array

1-It is a collection of data items of same data type. 2-It has declaration only
3-.There is no keyword.
4- array name represent the address of the starting element.

Structure

1-It is a collection of data items of different data type.
2- It has declaration and definition

3- keyword struct is used
4-Structure name is known as tag it is the short hand notation of the declaration.

61. Why can't we compare structures?
62. How are structure passing to a function?
63. What do you mean by self referential structure?

```
struct Node
{
        int rollno;
char name[20];
struct Node *ptr;
};
```

64. What are enumerations?

Ans: They are a list of named integer-valued constants. Example:enum color { black , orange=4, yellow, green, blue, violet };This declaration defines the symbols ―black‖, ―orange‖, ―yellow‖, etc. to have the values ―1,‖ ―4,‖ ―5,‖ … etc. The difference between an enumeration and a macro is that the enum actually declares a type, and therefore can be type checked.

65. What is difference between enumeration and macro?

66. What are macros? What are its advantages and disadvantages?

Ans: Preprocessor is a program or software which processes the source code before it passes through compiler. Preprocessor directive are commands given to preprocessor to do something before compilation(always starts with #)
The various preprocessor directives available in C language :

- Macro replacement directive (#define directive, #undef directive)
- File inclusion directive (#include directive)
- Line directive (#line directive)
- Error directive (#error directive)
- Pragma directive (#pragma directive)
- Conditional compilation directives (#if, #else, #elif, #endif, #ifdef, #ifndef)
- Null directive (# new-line)

67. What do the functions atoi(), itoa() and gcvt() do? Ans:

atoi() is a macro that converts integer to character.
itoa() It converts an integer to string
gcvt() It converts a floating point number to string

68. What is conditional compilation?
There are total six conditional compilation directives. There are:

(a)#if

(b)#elif

(c)#else

(d)#endif

(e)ifdef

(f)ifndef

#if directive :

- **It is conditional compilation directive. That is if condition is true then it will compile the c programming code otherwise it will not compile the c code.**

Syntax 1:

```
#if <Constant_expression>
   ----------
   ----------
#endif
```

69. What is File? Discuss types of files.

- When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
- If you have to enter a large number of data, it will take a lot of time to enter them all. However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.
- You can easily move your data from one computer to another without any changes.

Types of Files

When dealing with files, there are two types of files you should know about:

1. Text files
2. Binary files

1. Text files

- Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.

- When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.
- They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.

2. Binary files

- Binary files are mostly the .bin files in your computer.
- Instead of storing data in plain text, they store it in the binary form (0's and 1's).
- They can hold higher amount of data, are not readable easily and provides a better security than text files.

70. What is FILE pointer in C?

ANS: FILE pointer is struct data type which has been defined in standard library stdio.h. This data type points to a stream or a null value. It has been defined in stdio.h as

```
typedef struct{
   short         level;
   unsigned      flags;
   char          fd;
   unsigned char  hold;
   short         bsize;
   unsigned char  *buffer, *curp;
   unsigned      istemp;
   short         token;
} FILE;
```

Syntax to declare a FILE pointer:

```
FILE *fp;
```

71. What are the various file opening modes in C? Explain their significant difference.

ANS: Opening a file is performed using the library function in the **"stdio.h"** header file: fopen().

The syntax for opening a file in standard I/O is:

ptr = fopen("fileName","mode");

Opening Modes in Standard I/O

File Mode	Meaning of Mode	During Inexistence of file
R	Open for reading.	If the file does not exist, fopen() returns NULL.
Rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
W	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.

Opening Modes in Standard I/O

File Mode	Meaning of Mode	During Inexistence of file
Wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
A	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
Ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exists, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exists, it will be created.

72. Differentiate between printf() and fprintf() functions.

ANS:

printf:

printf function is used to print character stream of data on stdout console.

Syntax:

```
int printf(const char* str, ...);
```

Example:

```
// simple print on stdout
#include<stdio.h>
int main()
{
   printf("hello geeksquiz");
   return 0;
}
```

Output:

```
hello geeksquiz
```

sprintf:

Syntax:

```
int sprintf(char *str, const char *string,...);
```

String print function it is stead of printing on console store it on char buffer which are specified in sprintf

Example:

```
// Example program to demonstrate sprintf()
```

```
#include<stdio.h>
int main()
{
   char buffer[50];
   int a = 10, b = 20, c;
   c = a + b;
   sprintf(buffer, "Sum of %d and %d is %d", a, b, c);

   // The string "sum of 10 and 20 is 30" is stored
   // into buffer instead of printing on stdout
   printf("%s", buffer);

   return 0;
}
```

Output:

Sum of 10 and 20 is 30

fprintf:

fprintf is used to print the sting content in file but not on stdout console.

```
int fprintf(FILE *fptr, const char *str, ...);
```

Example:

```
#include<stdio.h>
int main()
{
   int i, n=2;
   char str[50];

   //open file sample.txt in write mode
   FILE *fptr = fopen("sample.txt", "w");
   if (fptr == NULL)
   {
      printf("Could not open file");
      return 0;
   }

   for (i=0; i<n; i++)
   {
      puts("Enter a name");
      gets(str);
      fprintf(fptr,"%d.%s\n", i, str);
   }
   fclose(fptr);

   return 0;
}
```

Input: GeeksforGeeks
GeeksQuiz

Output: sample.txt file now having output as
0. GeeksforGeeks
1. GeeksQuiz

73. Discuss fseek() function.

The C library function **int fseek(FILE *stream, long int offset, int whence)** sets the file position of the **stream** to the given **offset**.

Declaration

Following is the declaration for fseek() function.

int fseek(FILE *stream, long int offset, int whence)

Parameters

- **stream** − This is the pointer to a FILE object that identifies the stream.
- **offset** − This is the number of bytes to offset from whence.
- **whence** − This is the position from where offset is added. It is specified by one of the following constants −

Constant	Description
SEEK_SET	Beginning of file
SEEK_CUR	Current position of the file pointer
SEEK_END	End of file

Return Value

This function returns zero if successful, or else it returns a non-zero value.

Example

The following example shows the usage of fseek() function.

```
#include <stdio.h>

int main ()
{
  FILE *fp;
```

```
   fp = fopen("file.txt","w+");
   fputs("This is tutorialspoint.com", fp);
   fseek( fp, 7, SEEK_SET );
   fputs(" C Programming Language", fp);
   fclose(fp);
   return(0);
}
```

Let us compile and run the above program that will create a file **file.txt** with the following content. Initially program creates the file and writes *This is tutorialspoint.com* but later we had reset the write pointer at 7th position from the beginning and used puts() statement which over-write the file with the following content −

This is C Programming Language

74. What is role of rewind() function in C?

The C library function **void rewind(FILE *stream)** sets the file position to the beginning of the file of the given **stream**.

Declaration

Following is the declaration for rewind() function.

void rewind(FILE *stream);

eg. Rewind(fp);

Parameters

- **stream** − This is the pointer to a FILE object that identifies the stream.

Return Value

This function does not return any value.

75. What is ftell() function in C?

In the C Programming Language, the **ftell function** returns the current file position indicator for the stream pointed to by *stream*.

Syntax

The syntax for the ftell function in the C Language is:

long int ftell(FILE *stream);

Parameters or Arguments

stream

The stream whose file position indicator is to be returned.

Returns
The ftell function returns the current file position indicator for *stream*. If an error occurs, the ftell function will return -1L.

76. What is difference between algorithm and pseudo code?
ANS:

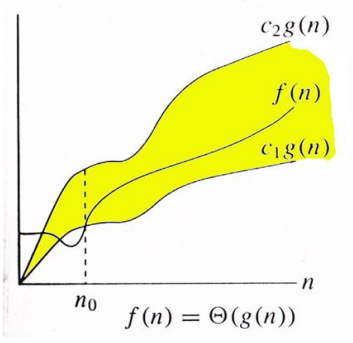
- An algorithm is just a sequence of steps with no fixed representation. It can be described in a high-level description, pseudocode or code in any language.
- More generically, any program written in any language, any pseudocode or really any concrete sequence of steps can be considered an algorithm.
- There's no fixed format for pseudocode - it can look very similar to any language or combination of languages, or it can simply be a natural language description of the algorithm.

77. What is complexity of an algorithm?

- Time complexity of an algorithm signifies the total time required by the program to run till its completion. The time complexity of algorithms is most commonly expressed using the **big O notation**.
- Time Complexity is most commonly estimated by counting the number of elementary functions performed by the algorithm. And since the algorithm's performance may vary with different types of input data, hence for an algorithm we usually use the **worst-case Time complexity** of an algorithm because that is the maximum time taken for any input size.

78. What is asymptotic notation? Discuss various notations and their significance.

ANS: The main idea of asymptotic analysis is to have a measure of efficiency of algorithms that doesn't depend on machine specific constants, and doesn't require algorithms to be implemented and time taken by programs to be compared. Asymptotic notations are mathematical tools to represent time complexity of algorithms for asymptotic analysis. The following 3 asymptotic notations are mostly used to represent time complexity of algorithms.

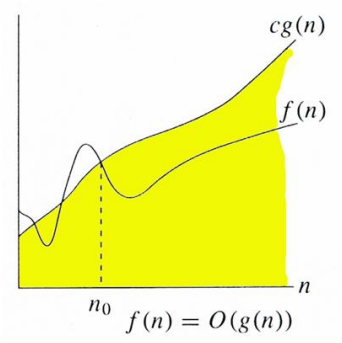


1) Θ Notation: The theta notation bounds a functions from above and below, so it defines exact asymptotic behavior.
A simple way to get Theta notation of an expression is to drop low order terms and ignore leading constants. For example, consider the following expression.

$3n^3 + 6n^2 + 6000 = \Theta(n^3)$
Dropping lower order terms is always fine because there will always be a n0 after which $\Theta(n^3)$ has higher values than Θn^2) irrespective of the constants involved.
For a given function g(n), we denote Θ(g(n)) is following set of functions.

Θ(g(n)) = {f(n): there exist positive constants c1, c2 and n0 such
that 0 <= c1*g(n) <= f(n) <= c2*g(n) for all n >= n0}

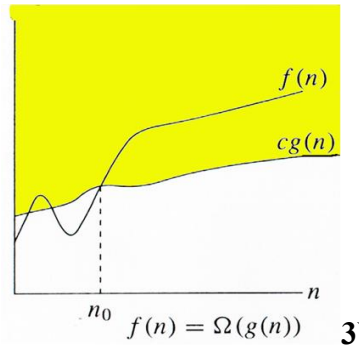
The above definition means, if f(n) is theta of g(n), then the value f(n) is always between c1*g(n) and c2*g(n) for large values of n (n >= n0). The definition of theta also requires that f(n) must be non-negative for values of n greater than n0.



2) Big O Notation: The Big O notation defines an upper bound of an algorithm, it bounds a function only from above. For example, consider the case of Insertion Sort. It takes linear time in best case and quadratic time in worst case. We can safely say that the time complexity of Insertion sort is O(n^2). Note that O(n^2) also covers linear time.
If we use Θ notation to represent time complexity of Insertion sort, we have to use two statements for best and worst cases:
1. The worst case time complexity of Insertion Sort is Θ(n^2).
2. The best case time complexity of Insertion Sort is Θ(n).

The Big O notation is useful when we only have upper bound on time complexity of an algorithm. Many times we easily find an upper bound by simply looking at the algorithm.

O(g(n)) = { f(n): there exist positive constants c and
n0 such that 0 <= f(n) <= cg(n) for
all n >= n0}



3) Ω Notation: Just as Big O notation provides an asymptotic upper bound on a function, Ω notation provides an asymptotic lower bound.

Ω Notation< can be useful when we have lower bound on time complexity of an algorithm. As discussed in the previous post, the best case performance of an algorithm is generally not useful, the Omega notation is the least used notation among all three.

For a given function g(n), we denote by Ω(g(n)) the set of functions.

Ω (g(n)) = {f(n): there exist positive constants c and
n0 such that 0 <= cg(n) <= f(n) for
all n >= n0}.

Let us consider the same Insertion sort example here. The time complexity of Insertion Sort can be written as Ω(n), but it is not a very useful information about insertion sort, as we are generally interested in worst case and sometimes in average case.

79. Differentiate between linear search and binary search.

ANS: A linear search scans one item at a time, without jumping to any item .

1. The worst case complexity is O(n), sometimes known an O(n) search
2. Time taken to search elements keep increasing as the number of elements are increased.

A binary search however, cut down your search to half as soon as you find middle of a sorted list.

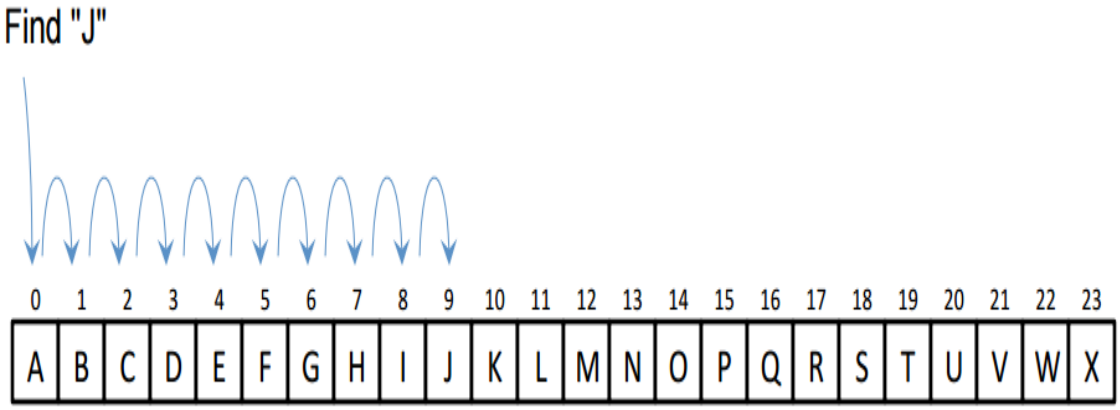
1. The middle element is looked to check if it is greater than or less than the value to be searched.
2. Accordingly, search is done to either half of the given list

Important Differences

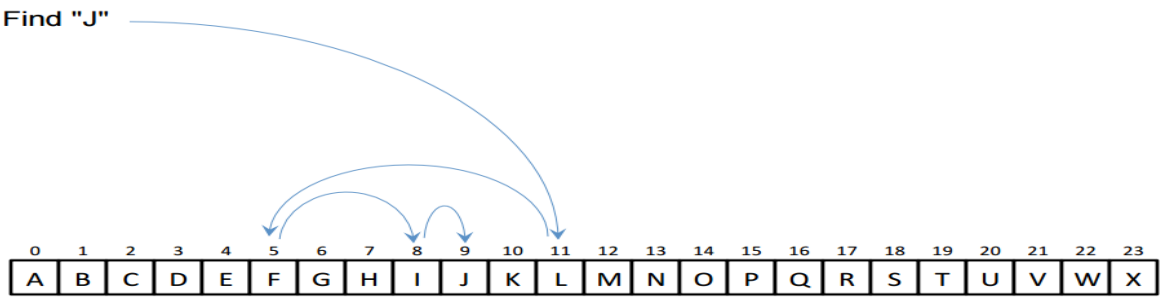
- Input data needs to be sorted in Binary Search and not in Linear Search
- Linear search does the sequential access whereas Binary search access data randomly.
- Time complexity of linear search -O(n) , Binary search has time complexity O(log n).
- Linear search performs equality comparisons and Binary search performs ordering comparisons

Let us look at an example to compare the two:

Linear Search to find the element “J” in a given sorted list from A-X



Binary Search to find the element “J” in a given sorted list from A-X



80. Discuss the complexity of the followings:

Bubble sort
Insertion Sort
Selection sort
Quick sort
Merge sort
Linear search
Binary search
