# C INTERVIEW QUESTIONS

**1. What is C language?**

The C programming language is a standardized programming language developed in the early 1970s by Ken Thompson and Dennis Ritchie for use on the UNIX operating system. It has since spread to many other operating systems, and is one of the most widely used programming languages. C is prized for its efficiency, and is the most popular programming language for writing system software, though it is also used for writing applications.

**2. What does static variable mean?**

There are 3 main uses for the static.

1. If you declare within a function: It retains the value between function calls

2. If it is declared for a function name: By default function is extern..so it will be visible from other files if the function declaration is as static..it is invisible for the outer files

3. Static for global variables: By default we can use the global variables from outside files If it is static global..that variable is limited to with in the file.

```
#include

int t = 10;

main(){

int x = 0;

void funct1();

funct1();

printf("After first call \n");

funct1();

printf("After second call \n");

funct1();
```

```
printf("After third call \n");

}

void funct1()

{

   static int y = 0;

   int z = 10;

   printf("value of y %d z %d",y,z);

   y=y+10;

}
```

value of y 0 z 10 After first call

value of y 10 z 10 After second call

value of y 20 z 10 After third call

**3. What are the different storage classes in C?**

C has three types of storage: automatic, static and allocated. Variable having block scope and without static specifier have automatic storage duration.

Variables with block scope, and with static specifier have static scope. Global variables (i.e, file scope) with or without the the static specifier also have static scope. Memory obtained from calls to malloc(), alloc() or realloc() belongs to allocated storage class.

**4. What is hashing?**

To hash means to grind up, and that□s essentially what hashing is all about. The heart of a hashing algorithm is a hash function that takes your nice, neat data and grinds it into some random-looking integer.

The idea behind hashing is that some data either has no inherent ordering (such as images) or is expensive to compare (such as images). If the data has no inherent ordering, you can□t perform comparison searches.

**5. Can static variables be declared in a header file?**

You can't declare a static variable without defining it as well (this is because the storage class modifiers static and extern are mutually exclusive). A static variable can be defined in a header file, but this would cause each source file that included the header file to have its own private copy of the variable, which is probably not what was intended.

**6. Can a variable be both constant and volatile?**

Yes. The const modifier means that this code cannot change the value of the variable, but that does not mean that the value cannot be changed by means outside this code.

The function itself did not change the value of the timer, so it was declared const. However, the value was changed by hardware on the computer, so it was declared volatile. If a variable is both const and volatile, the two modifiers can appear in either order.

**7. Can include files be nested?**

Yes. Include files can be nested any number of times. As long as you use precautionary measures, you can avoid including the same file twice. In the past, nesting header files was seen as bad programming practice, because it complicates the dependency tracking function of the MAKE program and thus slows down compilation. Many of today's popular compilers make up for this difficulty by implementing a concept called precompiled headers, in which all headers and associated dependencies are stored in a precompiled state.

**8. What is a null pointer?**

There are times when it's necessary to have a pointer that doesn't point to anything. The macro NULL, defined in , has a value that's guaranteed to be different from any valid pointer. NULL is a literal zero, possibly cast to void* or char*.

Some people, notably C++ programmers, prefer to use 0 rather than NULL.

The null pointer is used in three ways:

1) To stop indirection in a recursive data structure.

2) As an error value.

3) As a sentinel value.

**9. What is the output of printf("%d") ?**

When we write printf("%d",x); this means compiler will print the value of x. But as here, there is nothing after %d so compiler will show in output window garbage value.

**10. What is the difference between calloc() and malloc() ?**

calloc(...) allocates a block of memory for an array of elements of a certain size. By default the block is initialized to 0. The total number of memory allocated will be (number_of_elements * size).

malloc(...) takes in only a single argument which is the memory required in bytes. malloc(...) allocated bytes of memory and not blocks of memory like calloc(...).

malloc(...) allocates memory blocks and returns a void pointer to the allocated space, or NULL if there is insufficient memory available.

calloc(...) allocates an array in memory with elements initialized to 0 and returns a pointer to the allocated space. calloc(...) calls malloc(...) in order to use the C++ _set_new_mode function to set the new handler mode.

**11. What is the difference between printf() and sprintf() ?**

sprintf() writes data to the character array whereas printf(...) writes data to the standard output device.

**12. How to reduce a final size of executable?**

Size of the final executable can be reduced using dynamic linking for libraries.

**13. Can you tell me how to check whether a linked list is circular?**

Create two pointers, and set both to the start of the list. Update each as follows:

```
while (pointer1) {

pointer1 = pointer1->next;

pointer2 = pointer2->next;

if (pointer2) pointer2=pointer2->next;

if (pointer1 == pointer2) {

print ("circular");

}

}
```

If a list is circular, at some point pointer2 will wrap around and be either at the item just before pointer1, or the item before that. Either way, its either 1 or 2 jumps until they meet.

**14. Advantages of a macro over a function?**

Macro gets to see the Compilation environment, so it can expand __ __TIME__ __FILE__ #defines. It is expanded by the preprocessor.

For example, you can't do this without macros

```
#define PRINT(EXPR) printf( #EXPR "=%d\n", EXPR)

PRINT( 5+6*7 ) // expands into printf("5+6*7=%d\n", 5+6*7 );
```

You can define your mini language with macros:

```
#define strequal(A,B) (!strcmp(A,B))
```

**15. What is the difference between strings and character arrays?**

A major difference is: string will have static storage duration, whereas as a character array will not, unless it is explicity specified by using the static keyword.

Actually, a string is a character array with following properties:

* the multibyte character sequence, to which we generally call string, is used to initialize an array

of static storage duration. The size of this array is just sufficient to contain these characters plus the terminating NUL character.

* it not specified what happens if this array, i.e., string, is modified.

* Two strings of same value[1] may share same memory area.

**16. Write down the equivalent pointer expression for referring the same element a[i][j][k][l] ?**

a[i] == *(a+i)

a[i][j] == *(*(a+i)+j)

a[i][j][k] == *(*(*(a+i)+j)+k)

a[i][j][k][l] == *(*(*(*(a+i)+j)+k)+l)

**17. Which bit wise operator is suitable for checking whether a particular bit is on or off?**

The bitwise AND operator. Here is an example:

enum {

KBit0 = 1,

KBit1,



KBit31,

};

if ( some_int & KBit24 )

printf ( Bit number 24 is ON\n);

else

printf ( Bit number 24 is OFF\n);

**18. Which bit wise operator is suitable for turning off a particular bit in a number?**

The bitwise AND operator, again. In the following code snippet, the bit number 24 is reset to zero.

some_int = some_int & ~KBit24;

**19. Which bit wise operator is suitable for putting on a particular bit in a number?**

The bitwise OR operator. In the following code snippet, the bit number 24 is turned ON:

some_int = some_int | KBit24;

**20. Does there exist any other function which can be used to convert an integer or a float to a string?**

Some implementations provide a nonstandard function called itoa(), which converts an integer to string.

#include

char *itoa(int value, char *string, int radix);

DESCRIPTION

The itoa() function constructs a string representation of an integer.

PARAMETERS

value: Is the integer to be converted to string representation.

string: Points to the buffer that is to hold resulting string.

The resulting string may be as long as seventeen bytes.

radix: Is the base of the number; must be in the range 2 - 36.

A portable solution exists. One can use sprintf():

char s[SOME_CONST];

int i = 10;

float f = 10.20;

sprintf ( s, %d %f\n, i, f );

**21. Why does malloc(0) return valid memory address ? What's the use?**

malloc(0) does not return a non-NULL under every implementation. An implementation is free to behave in a manner it finds suitable, if the allocation size requested is zero. The implmentation may choose any of the following actions:

* A null pointer is returned.

* The behavior is same as if a space of non-zero size was requested. In this case, the usage of return value yields to undefined-behavior.

Notice, however, that if the implementation returns a non-NULL value for a request of a zero-length space, a pointer to object of ZERO length is returned! Think, how an object of zero size should be represented

For implementations that return non-NULL values, a typical usage is as follows:

```
void

func ( void )

{

int *p; /* p is a one-dimensional array, whose size will vary during the the lifetime of the program */

size_t c;

p = malloc(0); /* initial allocation */

if (!p)

{

perror (##FAILURE##);

return;

}

/* 👽*/
```

```
while (1)

{

c = (size_t) 👽; /* Calculate allocation size */

p = realloc ( p, c * sizeof *p );

/* use p, or break from the loop */

/* 👽*/

}

return;

}
```

Notice that this program is not portable, since an implementation is free to return NULL for a malloc(0) request, as the C Standard does not support zero-sized objects.

**22. Difference between const char* p and char const* p**

In const char* p, the character pointed by 👉is constant, so u cant change the value of character pointed by p but u can make 👉refer to some other location.

In char const* p, the ptr 👉is constant not the character referenced by it, so u cant make 👉to reference to any other location but u can change the value of the char pointed by 👉

**23. What is the result of using Option Explicit?**

When writing your C program, you can include files in two ways. The first way is to surround the file you want to include with the angled brackets < and >. This method of inclusion tells the preprocessor to look for the file in the predefined default location. This predefined default location is often an INCLUDE environment variable that denotes the path to your include files.

For instance, given the INCLUDE variable

INCLUDE=C:\COMPILER\INCLUDE;S:\SOURCE\HEADERS; using the #include version of file inclusion, the compiler first checks the C:\COMPILER\INCLUDE directory for the specified file. If the file is not found there, the compiler then checks the S:\SOURCE\HEADERS directory. If the

file is still not found, the preprocessor checks the current directory.

The second way to include files is to surround the file you want to include with double quotation marks. This method of inclusion tells the preprocessor to look for the file in the current directory first, then look for it in the predefined locations you have set up. Using the #include file version of file inclusion and applying it to the preceding example, the preprocessor first checks the current directory for the specified file. If the file is not found in the current directory, the C:COMPILERINCLUDE directory is searched. If the file is still not found, the preprocessor checks the S:SOURCEHEADERS directory.

The #include method of file inclusion is often used to include standard headers such as stdio.h or stdlib.h.

The #include file include nonstandard header files that you have created for use in your program. This is because these headers are often modified in the current directory, and you will want the preprocessor to use your newly modified version of the header rather than the older, unmodified version.

**24. What is the benefit of using an enum rather than a #define constant?**

The use of an enumeration constant (enum) has many advantages over using the traditional symbolic constant style of #define. These advantages include a lower maintenance requirement, improved program readability, and better debugging capability.

1) The first advantage is that enumerated constants are generated automatically by the compiler. Conversely, symbolic constants must be manually assigned values by the programmer.

2) Another advantage of using the enumeration constant method is that your programs are more readable and thus can be understood better by others who might have to update your program later.

3) A third advantage to using enumeration constant