☰

(https://www.wisdomjobs.com/)

Search for Jobs...

C INTERVIEW QUESTIONS & ANSWERS

★★★★★ 5 avg. rating (100% score) - 5880 votes

Are you looking for bright career in the **C** Interview Questions? Then we have provided all the necessary things like **C Interview Questions Interview Question and Answers** on our site page, not only the Question and Answers we have also provided the various job roles in C Interview Questions. In order to clear the C Interview Questions interview in the first attempt one must prepare well on all the topics of C Interview Questions. There are numerous leading companies that offer jobs in various roles like Trainee Engineer, Application Programming / Maintenance, Technical Support Executive, Network Device Driver Developer - C/wlan along with these there are many other roles too in C Interview Questions. For any other details on C Interview Questions related topics and also for various leading **C Interview Questions job** positions visit our site **Wisdomjobs** C Interview Questions page.

C Interview Questions

Prev (https://www.wisdomjobs.com/e-university/c-tutorial-232/bitwise-operator-1558.html)

Next (https://www.wisdomjobs.com/e-university/c-practice-tests-232-327109)

C Interview Questions

Question 1. Explain The Purpose Of Main( ) Function?

Answer : The function main( ) invokes other functions within it.It is the first function to be called when the program starts execution.

()

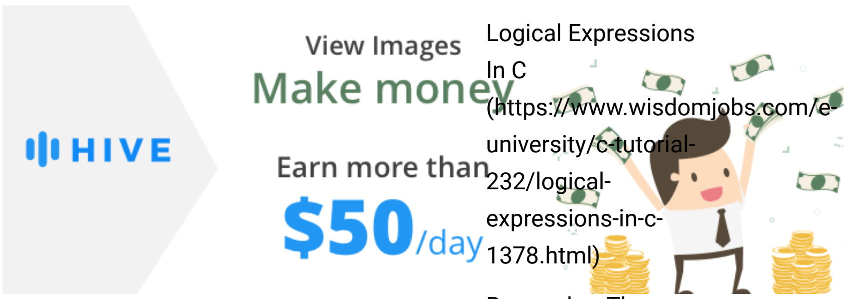
- It is the starting function.
- It returns an int value to the environment that called the program.
- Recursive call is allowed for main( ) also.
- It is a user-defined function.
- Program execution ends when the closing brace of the function main( ) is reached.
  - It has two arguments argument count and.
  - argument vector (represents strings passed).
- Any user-defined name can also be used as parameters for main( ) instead of argc and argv.

Question 2. Write The Equivalent Expression For X%8?

Answer : x&7.

Question 3. Why N++ Executes Faster Than N+1?

Answer : The expression n++ requires a single machine instruction such as INR to carry out the increment operation whereas n+1 requires more instructions to carry out this operation.



Question 4. Can The Sizeof Operator Be Used To Tell The Size Of An Array Passed To A Function?

Answer : No. There's no way to tell, at runtime, how many elements are in an array parameter just by looking at the array parameter itself. Remember, passing an array to a function is exactly the same as passing a pointer to the first element.

Question 5. Is Using Exit () The Same As Using Return?

Answer : No. The exit () function is used to exit your program and return control to the operating system. The return statement is used to return from a function and return control to the calling function. If you issue a return from the main () function, you are essentially returning

()

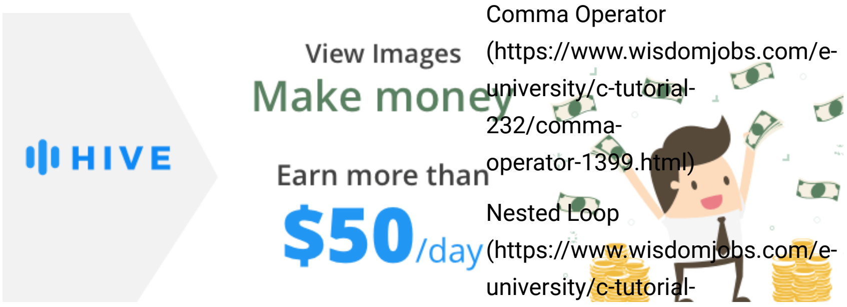
(https://www.wisdomjobs.com/)

Search for Jobs...

control to the calling function, which is the operating system. In this case, the return statement and exit () function are similar.

Question 6. What Is A Function And Built-in Function?

Answer : A large program is subdivided into a number of smaller programs or subprograms. Each subprogram specifies one or more actions to be performed for a large program. Such subprograms are functions. The function supports only static and extern storage classes. By default, function assumes extern storage class. Functions have global scope. Only register or auto storage class is allowed in the function parameters. Built-in functions that predefined and supplied along with the compiler are known as built-in functions. They are also known as library functions.



Question 7. Write About Modular Programming?

Answer : If a program is large, it is subdivided into a number of smaller programs that are called modules or subprograms. If a complex problem is solved using more modules, this approach is known as modular programming.

Question 8. When Does The Compiler Not Implicitly Generate The Address Of The First Element Of An Array?

Answer : Whenever an array name appears in an expression such as,

- array as an operand of the sizeof operator.
- array as an operand of & operator.
- array as a string literal initializer for a character array.

Then the compiler does not implicitly generate the address of the address of the first element of an array.

Question 9. Mention The Characteristics Of Arrays In C?

Answer :
- An array holds elements that have the same data type.
- Array elements are stored in subsequent memory locations.

expression-1395.html)

Indentation (https://www.wisdomjobs.com/e-university/c-tutorial-232/indentation-1396.html)

Iteration

Why Loop Control Structures Methods Of Looping (https://www.wisdomjobs.com/e-university/c-tutorial-232/why-loop-control-structures-methods-of-looping-1393.html)

Comma Operator (https://www.wisdomjobs.com/e-university/c-tutorial-232/comma-operator-1399.html)

Nested Loop (https://www.wisdomjobs.com/e-university/c-tutorial-232/nested-loop-1401.html)

Continue And Break Statements (https://www.wisdomjobs.com/e-university/c-tutorial-232/continue-and-break-statements-1402.html)

Preprocessor

Preprocessor Directives (https://www.wisdomjobs.com/e-university/c-tutorial-232/preprocessor-directives-1410.html)

Preprocessor Operators (https://www.wisdomjobs.com/e-university/c-tutorial-232/preprocessor-operators-1411.html)

Function

Function Basics (https://www.wisdomjobs.com/e-

()

☰

(https://www.wisdomjobs.com/)

Search for Jobs...

- Two-dimensional array elements are stored row by row in subsequent memory locations.
- Array name represents the address of the starting element.
- Array size should be mentioned in the declaration. Array size must be constant expression and not a variable.

Question 10. Differentiate Between A Linker And Linkage?

Answer : A linker converts an object code into an executable code by linking together the necessary build in functions. The form and place of declaration where the variable is declared in a program determine the linkage of variable.

Question 11. What Are The Advantages Of Auto Variables?

Answer :
- The same auto variable name can be used in different blocks.
- There is no side effect by changing the values in the blocks.
- The memory is economically used.
- Auto variables have inherent protection because of local scope.

Question 12. What Is Storage Class And What Are Storage Variable?

Answer : A storage class is an attribute that changes the behavior of a variable. It controls the lifetime, scope and linkage. There are five types of storage classes.

- auto.
- static.
- extern.
- register.
- typedef.

()

☰

(https://www.wisdomjobs.com/)

Search for Jobs...

Learn to Speak English Fluently with English Program.

Improve your spoken English skills & build success in your career & life

Ad eAgeTutor

Question 13. Which Expression Always Return True? Which Always Return False?

Answer :
- expression if (a=0) always return false.
- expression if (a=1) always return true.

Question 14. Is It Possible To Execute Code Even After The Program Exits The Main () Function?

Answer : The standard C library provides a function named at exit () that can be used to perform "cleanup" operations when your program terminates. You can set up a set of functions you want to perform automatically when your program exits by passing function pointers to the at exit() function.

Question 15. Why Should I Prototype A Function?

Answer : A function prototype tells the compiler what kind of arguments a function is looking to receive and what kind of return value a function is going to give back. This approach helps the compiler ensure that calls to a function are made correctly and that no erroneous type conversions are taking place.

Question 16. How Do You Print An Address?

Answer : The safest way is to use printf () (or fprintf() or sprintf()) with the %P specification. That prints a void pointer (void*). Different compilers might print a pointer with different formats. Your compiler will pick a format that's right for your environment.

If you have some other kind of pointer (not a void*) and you want to be very safe, cast the pointer to a void*:

```
printf (“%Pn”, (void*) buffer);
```

()

☰

(https://www.wisdomjobs.com/)

Search for Jobs...

Question 17. Can Math Operations Be Performed On A Void Pointer?

Answer : No. Pointer addition and subtraction are based on advancing the pointer by a number of elements. By definition, if you have a void pointer, you don't know what it's pointing to, so you don't know the size of what it's pointing to. If you want pointer arithmetic to work on raw addresses, use character pointers.

Question 18. How Can You Determine The Size Of An Allocated Portion Of Memory?

Answer : You can't, really free() can , but there's no way for your program to know the trick free() uses. Even if you disassemble the library and discover the trick, there's no guarantee the trick won't change with the next release of the compiler.

Question 19. What Is A "null Pointer Assignment" Error? What Are Bus Errors, Memory Faults, And Core Dumps?

Answer : These are all serious errors, symptoms of a wild pointer or subscript. Null pointer assignment is a message you might get when an MS-DOS program finishes executing. Some such programs can arrange for a small amount of memory to be available "where the NULL pointer points to" (so to speak). If the program tries to write to that area, it will overwrite the data put there by the compiler. When the program is done, code generated by the compiler examines that area. If that data has been changed, the compiler-generated code complains with null pointer assignment.

This message carries only enough information to get you worried. There's no way to tell, just from a null pointer assignment message, what part of your program is responsible for the error. Some debuggers, and some compilers, can give you more help in finding the problem. Bus error: core dumped and Memory fault: core dumped are messages you might see from a program running under UNIX. They're more programmers friendly. Both mean that a pointer or an array subscript was wildly out of bounds. You can get these messages on a read or on a write. They aren't restricted to null pointer problems.

The core dumped part of the message is telling you about a file, called core that has just been written in your current directory. This is a dump of everything on the stack and in the heap at the time the program was running. With the

()

help of a debugger, you can use the core dump to find where the bad pointer was used. That might not tell you why the pointer was bad, but it's a step in the right direction. If you don't have write permission in the current directory, you won't get a core file, or the core dumped message.

Question 20. What Is The Heap?

Answer : The heap is where malloc(), calloc(), and realloc() get memory.

Getting memory from the heap is much slower than getting it from the stack. On the other hand, the heap is much more flexible than the stack. Memory can be allocated at any time and deallocated in any order. Such memory isn't deallocated automatically; you have to call free ().

Recursive data structures are almost always implemented with memory from the heap. Strings often come from there too, especially strings that could be very long at runtime. If you can keep data in a local variable (and allocate it from the stack), your code will run faster than if you put the data on the heap. Sometimes you can use a better algorithm if you use the heap—faster, or more robust, or more flexible. It's a tradeoff.

If memory is allocated from the heap, it's available until the program ends. That's great if you remember to deallocate it when you're done. If you forget, it's a problem. A "memory leak" is some allocated memory that's no longer needed but isn't deallocated. If you have a memory leak inside a loop, you can use up all the memory on the heap and not be able to get any more. (When that happens, the allocation functions return a null pointer.) In some environments, if a program doesn't deallocate everything it allocated, memory stays unavailable even after the program ends.

Question 21. Difference Between Null And Nul?

Answer : NULL is a macro defined in for the null pointer. NUL is the name of the first character in the ASCII character set. It corresponds to a zero value. There's no standard macro NUL in C, but some people like to define it.

The digit 0 corresponds to a value of 80, decimal. Don't confuse the digit 0 with the value of '' (NUL)!

NULL can be defined as ((void*)0), NUL as ''.

Question 22. What Is The Stack?

Self-referential Structures (https://www.wisdomjobs.com/e-university/c-tutorial-232/self-referential-structures-1544.html)
List As A Data Structure (https://www.wisdomjobs.com/e-university/c-tutorial-232/list-as-a-data-structure-1546.html)
Creating A Node (https://www.wisdomjobs.com/e-university/c-tutorial-232/creating-a-node-1547.html)
Displaying A List (https://www.wisdomjobs.com/e-university/c-tutorial-232/displaying-a-list-1549.html)
Inserting A Node (https://www.wisdomjobs.com/e-university/c-tutorial-232/inserting-a-node-1551.html)
Deleting A Node (https://www.wisdomjobs.com/e-university/c-tutorial-232/deleting-a-node-1552.html)
Program Performing Operations On Linklist (https://www.wisdomjobs.com/e-university/c-tutorial-232/program-performing-operations-on-linklist-1553.html)
Miscellaneous
Enumerations (https://www.wisdomjobs.com/e-university/c-tutorial-232/enumerations-1554.html)
Type Casting (https://www.wisdomjobs.com/e-university/c-tutorial-232/type-casting-1555.html)

()

☰

(https://www.wisdomjobs.com/)

Search for Jobs...

Bitwise Operator (https://www.wisdomjobs.com/e-university/c-tutorial-232/bitwise-operator-1558.html)

C Practice Tests (https://www.wisdomjobs.com/e-university/c-practice-tests-232-327109)

Answer : The stack is where all the functions' local (auto) variables are created. The stack also contains some information used to call and return from functions.

A "stack trace" is a list of which functions have been called, based on this information. When you start using a debugger, one of the first things you should learn is how to get a stack trace. The stack is very inflexible about allocating memory; everything must be deallocated in exactly the reverse order it was allocated in. For implementing function calls, that is all tl
Allocating memory off the stack is extre
of the reasons C compilers generate su
their heavy use of a simple stack.

There used to be a C function that any p
use for allocating memory off the stack
automatically deallocated when the call
returned. This was a dangerous functior
available anymore.

Question 23. When Should A Far Pointe

Answer : Sometimes you can get aw
small memory model in most of a given
might be just a few things that don't fit i
and code segments. When that happens
explicit far pointers and function declara
rest of memory. A far function can be ou
segment most functions are shoehorne
code model. (Often, libraries are declare
they'll work no matter what code model

A far pointer can refer to information ou
data segment. Typically, such pointers are used with farmalloc () and such, to manage a heap separate from where all the rest of the data lives. If you use a small-data, large-code model, you should explicitly make your function pointers far.

Ad ✕ Microsoft | intel

Need to keep your customer's data safe?

Question 24. Differentiate Between Far And Near?

Answer : Some compilers for PC compatibles use two types of pointers. Near pointers are 16 bits long and can address a 64KB range. far pointers are 32 bits long and can address a 1MB range.

Near pointers operate within a 64KB segment. There's one segment for function addresses and one segment for data. far pointers have a 16-bit base (the segment address) and

()

a 16-bit offset. The base is multiplied by 16, so a far pointer is effectively 20 bits long. Before you compile your code, you must tell the compiler which memory model to use. If you use a small code memory model, near pointers are used by default for function addresses.

That means that all the functions need to fit in one 64KB segment. With a large-code model, the default is to use far function addresses. You'll get near pointers with a small data model, and far pointers with a large data model. These are just the defaults; you can declare variables and functions as explicitly near or far.

Far pointers are a little slower. Whenever one is used, the code or data segment register needs to be swapped out. Far pointers also have odd semantics for arithmetic and comparison. For example, the two far pointers in the preceding example point to the same address, but they would compare as different! If your program fits in a small-data, small-code memory model, your life will be easier.

Question 25. Is It Better To Use Malloc () Or Calloc ()?

Answer : Both the malloc() and the calloc() functions are used to allocate dynamic memory. Each operates slightly different from the other. malloc() takes a size and returns a pointer to a chunk of memory at least that big:

```
void *malloc( size_t size );
```

calloc() takes a number of elements, and the size of each, and returns a pointer to a chunk of memory at least big enough to hold them all:

```
void *calloc( size_t numElements,size_t sizeOfElemen
```

There's one major difference and one minor difference between the two functions. The major difference is that malloc () doesn't initialize the allocated memory. The first time malloc () gives you a particular chunk of memory, the memory might be full of zeros. If memory has been allocated, freed, and reallocated, it probably has whatever junk was left in it. That means, unfortunately, that a program might run in simple cases (when memory is never reallocated) but break when used harder (and when memory is reused). calloc() fills the allocated memory with all zero bits. That means that anything there you're going to use as a char or an int of any length, signed or unsigned, is guaranteed to be zero. Anything you're going to use as a

()

pointer is set to all zero bits. That's usually a null pointer, but it's not guaranteed. Anything you're going to use as a float or double is set to all zero bits; that's a floating-point zero on some types of machines, but not on all.

The minor difference between the two is that calloc () returns an array of objects; malloc () returns one object. Some people use calloc () to make clear that they want an array.

Question 26. Why Is That We Have To Assign Null To The Elements (pointer) After Freeing Them?

Answer : This is paranoia based on long experience. After a pointer has been freed, you can no longer use the pointed-to data. The pointer is said to "dangle"; it doesn't point at anything useful. If you "NULL out" or "zero out" a pointer immediately after freeing it, your program can no longer get in trouble by using that pointer. True, you might go indirect on the null pointer instead, but that's something your debugger might be able to help you with immediately. Also, there still might be copies of the pointer that refer to the memory that has been deallocated; that's the nature of C. Zeroing out pointers after freeing them won't solve all problems.

Question 27. When Would You Use A Pointer To A Function?

Answer : Pointers to functions are interesting when you pass them to other functions. A function that takes function pointers says, in effect, "Part of what I do can be customized. Give me a pointer to a function, and I'll call it when that part of the job needs to be done. That function can do its part for me." This is known as a "callback." It's used a lot in graphical user interface libraries, in which the style of a display is built into the library but the contents of the display are part of the application.

As a simpler example, say you have an array of character pointers (char*s), and you want to sort it by the value of the strings the character pointers point to. The standard qsort() function uses function pointers to perform that task. qsort() takes four arguments,

- a pointer to the beginning of the array,
- the number of elements in the array,
- the size of each array element, and,
- a comparison function, and returns an int.

()

Question 28. What Does It Mean When A Pointer Is Used In An If Statement?

Answer : Any time a pointer is used as a condition, it means "Is this a non-null pointer?" A pointer can be used in an if, while, for, or do/while statement, or in a conditional expression.

Question 29. Is Null Always Defined As 0?

Answer : NULL is defined as either 0 or (void*)0. These values are almost identical; either a literal zero or a void pointer is converted automatically to any kind of pointer, as necessary, whenever a pointer is needed (although the compiler can't always tell when a pointer is needed).

Question 30. What Is A Null Pointer?

Answer : 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:

- To stop indirection in a recursive data structure
- As an error value
- As a sentinel value

Question 31. Mention The Levels Of Pointers Can You Have?

Answer : The answer depends on what you mean by "levels of pointers." If you mean "How many levels of indirection can you have in a single declaration?" the answer is "At least 12."

```
int i = 0;
int *ip01 = & i;
int **ip02 = & ip01;
int ***ip03 = & ip02;
int ****ip04 = & ip03;
int *****ip05 = & ip04;
int ******ip06 = & ip05;
int *******ip07 = & ip06;
int ********ip08 = & ip07;
int *********ip09 = & ip08;
int **********ip10 = & ip09;
int ***********ip11 = & ip10;
int ************ip12 = & ip11;
************ip12 = 1; /* i = 1 */
```

The ANSI C standard says all compilers must handle at least 12 levels. Your compiler might support more.

()

Question 32. What Is Indirection?

Answer : If you declare a variable, its name is a direct reference to its value. If you have a pointer to a variable or any other object in memory, you have an indirect reference to its value.

Question 33. How Do You Print Only Part Of A String?

Answer : /* Use printf () to print the first 11 characters of source_str. */

```
printf (“First 11 characters: ‘%11.11s’n”, source_st
```

Question 34. How To Convert A String To A Number?

Answer : The standard C library provides several functions for converting strings to numbers of all formats (integers, longs, floats, and so on) and vice versa.

The following functions can be used to convert strings to numbers:
Function Name Purpose

- atof(): Converts a string to a double-precision floating-point value.
- atoi(): Converts a string to an integer.
- atol(): Converts a string to a long integer.

Question 35. How To Convert A Number To A String?

Answer : The standard C library provides several functions for converting numbers of all formats (integers, longs, floats, and so on) to strings and vice versa The following functions can be used to convert integers to strings:
Function Name Purpose

- iota(): Converts an integer value to a string.
- ltoa (): Converts a long integer value to a string.
- ultoa (): Converts an unsigned long integer value to a string.

The following functions can be used to convert floating-point values to strings:
Function Name Purpose

- ecvt() : Converts a double-precision floating-point value to a string without an embedded decimal point.
- fcvt(): Same as ecvt(), but forces the precision to a specified number of digits.

()

- gcvt(): Converts a double-precision floating-point value to a string with an embedded decimal point.
- strtod(): Converts a string to a double-precision floating-point value and reports any "leftover" numbers that could not be converted.
- strtol(): Converts a string to a long integer and reports any "leftover" numbers that could not be converted.
- strtoul(): Converts a string to an unsigned long integer and reports any "leftover" numbers that could not be converted.

Question 36. Differentiate Between A String Copy (strcpy) And A Memory Copy (memcpy)? When Should Each Be Used?

Answer : The strcpy() function is designed to work exclusively with strings. It copies each byte of the source string to the destination string and stops when the terminating null character () has been moved. On the other hand, the memcpy () function is designed to work with any type of data. Because not all data ends with a null character, you must provide the memcpy () function with the number of bytes you want to copy from the source to the destination.

Question 37. How Can You Check To See Whether A Symbol Is Defined?

Answer : You can use the #ifdef and #ifndef preprocessor directives to check whether a symbol has been defined (#ifdef) or whether it has not been defined (#ifndef).

Question 38. How Do You Override A Defined Macro?

Answer : You can use the #undef preprocessor directive to undefine (override) a previously defined macro.

Question 39. What Is #line Used For?

Answer : The #line preprocessor directive is used to reset the values of the _ _LINE_ _ and _ _FILE_ _ symbols, respectively. This directive is commonly used in fourth-generation languages that generate C language source files.

Question 40. What Is A Pragma?

Answer :

()

The #pragma preprocessor directive allows each compiler to implement compiler-specific features that can be turned on and off with the #pragma statement. For instance, your compiler might support a feature called loop optimization. This feature can be invoked as a command-line option or as a #pragma directive. To implement this option using the #pragma directive, you would put the following line into your code:

```
#pragma loop_opt(on).
```

Question 41. What Are The Standard Predefined Macros?

Answer : The ANSI C standard defines six predefined macros for use in the C language:
Macro Name Purpose

- _ _LINE_ _ Inserts the current source code line number in your code.
- _ _FILE_ _ Inserts the current source code filename in your code.
- _ _DATE_ _ Inserts the current date of compilation in your code.
- _ _TIME_ _ Inserts the current time of compilation in your code.
- _ _cplusplus Is defined if you are compiling a C++ program.

Question 42. How Many Levels Deep Can Include Files Be Nested?

Answer : Even though there is no limit to the number of levels of nested include files you can have, your compiler might run out of stack space while trying to include an inordinately high number of files. This number varies according to your hardware configuration and possibly your compiler.

Question 43. Can Include Files Be Nested?

Answer : 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.

()

Many programmers like to create a custom header file that has #include statements for every header needed for each module. This is perfectly acceptable and can help avoid potential problems relating to #include files, such as accidentally omitting an #include file in a module.

Question 44. Define Which Header File To Include At Compile Time?

Answer : Yes. This can be done by using the #if, #else, and #endif preprocessor directives. For example, certain compilers use different names for header files. One such case is between Borland C++, which uses the header file alloc.h, and Microsoft C++, which uses the header file malloc.h. Both of these headers serve the same purpose, and each contains roughly the same definitions. If, however, you are writing a program that is to support Borland C++ and Microsoft C++, you must define which header to include at compile time. The following example shows how this can be done:

```
#ifdef _ _BORLANDC_ _
#include  #else #include  #endif.
```

Question 45. Differentiate Between #include And #include "file"?

Answer : 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. This is because these headers are rarely (if ever) modified, and they should always be read from your compiler's standard include file directory.

The #include "file" method of file inclusion is often used to 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.

Question 46. Which Is Better To Use A Macro Or A Function?

Answer : The answer depends on the situation you are writing code for. Macros have the distinct advantage of being more efficient (and faster) than functions, because their corresponding code is inserted directly into your source code at the point where the macro is called. There is no overhead involved in using a macro like there is in placing a call to a function. However, macros are generally small and cannot handle large, complex coding constructs. A function is more suited for this type of situation. Additionally, macros are expanded inline, which means that the code is replicated for each occurrence of a macro. Your code therefore could be somewhat larger when you use macros than if you were to use functions. Thus, the choice between using a macro and using a function is one of deciding between the tradeoff of faster program speed versus smaller program size. Generally, you should use macros to replace small, repeatable code sections, and you should use functions for larger coding tasks that might require several lines of code.

Question 47. How Are Portions Of A Program Disabled In Demo Versions?

Answer : If you are distributing a demo version of your program, the preprocessor can be used to enable or disable portions of your program. The following portion of

()

code shows how this task is accomplished, using the preprocessor directives #if and #endif:

```
int save document(char* doc_name)
{
#if DEMO_VERSION
printf("Sorry! You can't save documents using the DE
version of this program!n");
return(0);
#endif
....
```

Question 48. What Is The Benefit Of Using An Enum Rather Than A #define Constant?

Answer : 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.

- The first advantage is that enumerated constants are generated automatically by the compiler. Conversely, symbolic constants must be manually assigned values by the programmer. For instance, if you had an enumerated constant type for error codes that could occur in your program, your enum definition could look something like this:

  ```
  enum Error_Code
  {
  OUT_OF_MEMORY,
  INSUFFICIENT_DISK_SPACE,
  LOGIC_ERROR,
  FILE_NOT_FOUND
  };
  ```

  In the preceding example, OUT_OF_MEMORY is automatically assigned the value of 0 (zero) by the compiler because it appears first in the definition. The compiler then continues to automatically assign numbers to the enumerated constants, making INSUFFICIENT_DISK_SPACE equal to 1, LOGIC_ERROR equal to 2, and FILE_NOT_FOUND equal to 3, so on. If you were to approach the same example by using symbolic constants, your code would look something like this:

  ```
  #define OUT_OF_MEMORY 0
  #define INSUFFICIENT_DISK_SPACE 1
  #define LOGIC_ERROR 2
  #define FILE_NOT_FOUND 3
  ```

()

values by the programmer. Each of the two methods arrives at the same result: four constants assigned numeric values to represent error codes. Consider the maintenance required, however, if you were to add two constants to represent the error codes DRIVE_NOT_READY and CORRUPT_FILE. Using the enumeration constant method, you simply would put these two constants anywhere in the enum definition. The compiler would generate two unique values for these constants. Using the symbolic constant method, you would have to manually assign two new numbers to these constants. Additionally, you would want to ensure that the numbers you assign to these constants are unique.

- 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.
- A third advantage to using enumeration constants is that some symbolic debuggers can print the value of an enumeration constant. Conversely, most symbolic debuggers cannot print the value of a symbolic constant. This can be an enormous help in debugging your program, because if your program is stopped at a line that uses an enum, you can simply inspect that constant and instantly know its value. On the other hand, because most debuggers cannot print #define values, you would most likely have to search for that value by manually looking it up in a header file.

Question 49. Can A File Other Than A .h File Be Included With #include?

Answer : The preprocessor will include whatever file you specify in your #include statement. Therefore, if you have the line

```
#include <macros.inc>
```

in your program, the file macros.inc will be included in your precompiled program. It is, however, unusual programming practice to put any file that does not have a .h or .hpp extension in an

```
#include statement.
```

()

You should always put a .h extension on any of your C files you are going to include. This method makes it easier for you and others to identify which files are being used for preprocessing purposes. For instance, someone modifying or debugging your program might not know to look at the macros.inc file for macro definitions. That person might try in vain by searching all files with .h extensions and come up empty. If your file had been named macros.h, the search would have included the macros.h file, and the searcher would have been able to see what macros you defined in it.

Question 50. Give The Benefit Of Using #define To Declare A Constant?

Answer : Using the #define method of declaring a constant enables you to declare a constant in one place and use it throughout your program. This helps make your programs more maintainable, because you need to maintain only the #define statement and not several instances of individual constants throughout your program. For instance, if your program used the value of pi (approximately 3.14159) several times, you might want to declare a constant for pi as follows:

```
#define PI 3.14159
```

Using the #define method of declaring a constant is probably the most familiar way of declaring constants to traditional C programmers. Besides being the most common method of declaring constants, it also takes up the least memory. Constants defined in this manner are simply placed directly into your source code, with no variable space allocated in memory. Unfortunately, this is one reason why most debuggers cannot inspect constants created using the #define method.

Question 51. How To Avoid Including A Header More Than Once?

Answer : One easy technique to avoid multiple inclusions of the same header is to use the #ifndef and #define preprocessor directives. When you create a header for your program, you can #define a symbolic name that is unique to that header. You can use the conditional preprocessor directive named #ifndef to check whether that symbolic name has already been assigned. If it is assigned, you should not include the header, because it

()

has already been preprocessed. If it is not defined, you should define it to avoid any further inclusions of the header. The following header illustrates this technique:

```
#ifndef _FILENAME_H
#define _FILENAME_H
#define VER_NUM “1.00.00”
#define REL_DATE “08/01/94”
#if _ _WINDOWS_ _
#define OS_VER “WINDOWS”
#else
#define OS_VER “DOS”
#endif
#endif
```

When the preprocessor encounters this header, it first checks to see whether _FILENAME_H has been defined. If it hasn’t been defined, the header has not been included yet, and the _FILENAME_H symbolic name is defined. Then, the rest of the header is parsed until the last #endif is encountered, signaling the end of the conditional #ifndef _FILENAME_H statement. Substitute the actual name of the header file for “FILENAME” in the preceding example to make it applicable for your programs.

Question 52. Differentiate Between Arrays And Pointers?

Answer : Pointers are used to manipulate data using the address. Pointers use * operator to access the data pointed to by them Arrays use subscripted variables to access and manipulate data. Array variables can be equivalently written using pointer expression.

Question 53. Mention The Purpose Of Realloc ( )?

Answer : The function realloc (ptr,n) uses two arguments. The first argument ptr is a pointer to a block of memory for which the size is to be altered. The second argument n specifies the new size. The size may be increased or decreased. If n is greater than the old size and if sufficient space is not available subsequent to the old region, the function realloc ( ) may create a new region and all the old data are moved to the new region.

Question 54. Describe Static Memory Allocation And Dynamic Memory Allocation?

Answer : Static memory allocation: The compiler allocates the required memory space for a declared variable. By using the address of operator, the reserved address is obtained and this address may be assigned to a pointer variable. Since most of the declared variable has static memory, this way of assigning pointer value to a

()

pointer variable is known as static memory allocation. Memory is assigned during compilation time. Dynamic memory allocation: It uses functions such as malloc ( ) or calloc ( ) to get memory dynamically. If these functions are used to get memory dynamically and the values returned by these functions are assigned to pointer variables, such assignments are known as dynamic memory allocation. Memory is assigned during run time.

Question 55. How Are Pointer Variables Initialized?

Answer : Pointer variable are initialized by one of the following two ways

- Static memory allocation
- Dynamic memory allocation

Question 56. What Is A Pointer Variable?

Answer : A pointer variable is a variable that may contain the address of another variable or any valid address in the memory.

Question 57. Differentiate Between Text And Binary Modes?

Answer : Streams can be classified into two types: text streams and binary streams. Text streams are interpreted, with a maximum length of 255 characters. With text streams, carriage return/line feed combinations are translated to the newline n character and vice versa. Binary streams are uninterpreted and are treated one byte at a time with no translation of characters. Typically, a text stream would be used for reading and writing standard text files, printing output to the screen or printer, or receiving input from the keyboard.

A binary text stream would typically be used for reading and writing binary files such as graphics or word processing documents, reading mouse input, or reading and writing to the modem.

Question 58. How To Restore A Redirected Standard Stream?

Answer : The preceding example showed how you can redirect a standard stream from within your program. But what if later in your program you wanted to restore the standard stream to its original state? By using the standard C library functions named dup() and fdopen(), you can restore a standard stream such as stdout to its original state.

()

The dup() function duplicates a file handle. You can use the dup() function to save the file handle corresponding to the stdout standard stream. The fdopen() function opens a stream that has been duplicated with the dup() function.

Question 59. How To Search For Data In A Linked List?

Answer : Unfortunately, the only way to search a linked list is with a linear search, because the only way a linked list's members can be accessed is sequentially. Sometimes it is quicker to take the data from a linked list and store it in a different data structure so that searches can be more efficient.

Question 60. How To Sort A Linked List?

Answer : Both the merge sort and the radix sort are good sorting algorithms to use for linked lists.

Question 61. What Do You Mean By Hashing?

Answer : 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.

If the data is expensive to compare, the number of comparisons used even by a binary search might be too many. So instead of looking at the data themselves, you'll condense (hash) the data to an integer (its hash value) and keep all the data with the same hash value in the same place. This task is carried out by using the hash value as an index into an array. To search for an item, you simply hash it and look at all the data whose hash values match that of the data you're looking for. This technique greatly lessens the number of items you have to look at. If the parameters are set up with care and enough storage is available for the hash table, the number of comparisons needed to find an item can be made arbitrarily close to one.

One aspect that affects the efficiency of a hashing implementation is the hash function itself. It should ideally distribute data randomly throughout the entire hash table, to reduce the likelihood of collisions. Collisions occur when two different keys have the same hash value. There

()

are two ways to resolve this problem. In "open addressing," the collision is resolved by the choosing of another position in the hash table for the element inserted later. When the hash table is searched, if the entry is not found at its hashed position in the table, the search continues checking until either the element is found or an empty position in the table is found.

The second method of resolving a hash collision is called "chaining." In this method, a "bucket" or linked list holds all the elements whose keys hash to the same value. When the hash table is searched, the list must be searched linearly.

Question 62. Which Is The Quickest Searching Method To Use?

Answer : A binary search, such as bsearch() performs, is much faster than a linear search. A hashing algorithm can provide even faster searching. One particularly interesting and fast method for searching is to keep the data in a "digital trie." A digital trie offers the prospect of being able to search for an item in essentially a constant amount of time, independent of how many items are in the data set.

A digital trie combines aspects of binary searching, radix searching, and hashing. The term "digital trie" refers to the data structure used to hold the items to be searched. It is a multilevel data structure that branches N ways at each level.

Question 63. What Is The Easiest Sorting Method To Use?

Answer : The answer is the standard library function qsort(). It's the easiest sort by far for several reasons:

- It is already written.
- It is already debugged.
- It has been optimized as much as possible (usually).

```
Void qsort(void *buf, size_t num, size_t size, int (
(const void *ele1, const void *ele2));
```

Question 64. Which Is The Quickest Sorting Method To Use?

Answer : The answer depends on what you mean by quickest. For most sorting problems, it just doesn't matter how quick the sort is because it is done infrequently or

()

other operations take significantly more time anyway. Even in cases in which sorting speed is of the essence, there is no one answer. It depends on not only the size and nature of the data, but also the likely order. No algorithm is best in all cases.

There are three sorting methods in this author's "toolbox" that are all very fast and that are useful in different situations. Those methods are quick sort, merge sort, and radix sort.

The Quick Sort :The quick sort algorithm is of the "divide and conquer" type. That means it works by reducing a sorting problem into several easier sorting problems and solving each of them. A "dividing" value is chosen from the input data, and the data is partitioned into three sets: elements that belong before the dividing value, the value itself, and elements that come after the dividing value. The partitioning is performed by exchanging elements that are in the first set but belong in the third with elements that are in the third set but belong in the first Elements that are equal to the dividing element can be put in any of the three sets—the algorithm will still work properly.

The Merge Sort: The merge sort is a "divide and conquer" sort as well. It works by considering the data to be sorted as a sequence of already-sorted lists (in the worst case, each list is one element long). Adjacent sorted lists are merged into larger sorted lists until there is a single sorted list containing all the elements. The merge sort is good at sorting lists and other data structures that are not in arrays, and it can be used to sort things that don't fit into memory. It also can be implemented as a stable sort.

The Radix Sort : The radix sort takes a list of integers and puts each element on a smaller list, depending on the value of its least significant byte. Then the small lists are concatenated, and the process is repeated for each more significant byte until the list is sorted. The radix sort is simpler to implement on fixed-length data such as ints.

Question 65. What Is The Benefit Of Using Const For Declaring Constants?

Answer : The benefit of using the const keyword is that the compiler might be able to make optimizations based on the knowledge that the value of the variable will not change. In addition, the compiler will try to ensure that the values won't be changed inadvertently.

()

Of course, the same benefits apply to #defined constants. The reason to use const rather than #define to define a constant is that a const variable can be of any type (such as a struct, which can't be represented by a #defined constant). Also, because a const variable is a real variable, it has an address that can be used, if needed, and it resides in only one place in memory.

Question 66. Is It Acceptable To Declare/define A Variable In A C Header?

Answer : A global variable that must be accessed from more than one file can and should be declared in a header file. In addition, such a variable must be defined in one source file.

Variables should not be defined in header files, because the header file can be included in multiple source files, which would cause multiple definitions of the variable. The ANSI C standard will allow multiple external definitions, provided that there is only one initialization. But because there's really no advantage to using this feature, it's probably best to avoid it and maintain a higher level of portability.

"Global" variables that do not have to be accessed from more than one file should be declared static and should not appear in a header file.

Question 67. When Should A Type Cast Be Used?

Answer : There are two situations in which to use a type cast. The first use is to change the type of an operand to an arithmetic operation so that the operation will be performed properly.

The second case is to cast pointer types to and from void * in order to interface with functions that expect or return void pointers. For example, the following line type casts the return value of the call to malloc() to be a pointer to a foo structure.

```
struct foo *p = (struct foo *) malloc(sizeof(struct
```

Question 68. How To Determine The Maximum Value That A Numeric Variable Can Hold?

Answer : For integral types, on a machine that uses two's complement arithmetic (which is just about any machine you're likely to use), a signed type can hold numbers from –2(number of bits – 1) to +2(number of bits – 1) – 1. An unsigned type can hold values from 0 to +2(number of bits) – 1. For instance, a 16-bit signed integer can hold numbers from –2^15 (–32768) to +2^15 – 1 (32767).

Question 69. Can A Variable Be Both Const And Volatile?

Answer : 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. For instance, in the example in FAQ 8, the timer structure was accessed through a volatile const pointer. 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.

Question 70. When Does The Register Modifier Be Used? Does It Really Help?

Answer : The register modifier hints to the compiler that the variable will be heavily used and should be kept in the CPU's registers, if possible, so that it can be accessed faster. There are several restrictions on the use of the register modifier.

First, the variable must be of a type that can be held in the CPU's register. This usually means a single value of a size less than or equal to the size of an integer. Some machines have registers that can hold floating-point numbers as well. Second, because the variable might not be stored in memory, its address cannot be taken with the unary & operator. An attempt to do so is flagged as an error by the compiler. Some additional rules affect how useful the register modifier is. Because the number of registers is limited, and because some registers can hold only certain types of data (such as pointers or floating-point numbers), the number and types of register modifiers that will actually have any effect are dependent on what machine the program will run on. Any additional register modifiers are silently ignored by the compiler.

Also, in some cases, it might actually be slower to keep a variable in a register because that register then becomes unavailable for other purposes or because the variable isn't

()

used enough to justify the overhead of loading and storing it.

So when should the register modifier be used? The answer is never, with most modern compilers. Early C compilers did not keep any variables in registers unless directed to do so, and the register modifier was a valuable addition to the language. C compiler design has advanced to the point, however, where the compiler will usually make better decisions than the programmer about which variables should be stored in registers. In fact, many compilers actually ignore the register modifier, which is perfectly legal, because it is only a hint and not a directive.

C Related Tutorials

C++ Tutorial (https://www.wisdomjobs.com/e-university/c-plus-plus-tutorial-219.html)	Java Tutorial (https://www.wisdomjobs.com/e-university/java-tutorial-1183.html)
Go (programming language) Tutorial (https://www.wisdomjobs.com/e-university/go-programming-language-tutorial-1306.html)	F Sharp (programming language) Tutorial (https://www.wisdomjobs.com/e-university/f-sharp-programming-language-tutorial-1421.html)
R Programming language Tutorial (https://www.wisdomjobs.com/e-university/r-programming-language-tutorial-1579.html)	D Programming Language Tutorial (https://www.wisdomjobs.com/e-university/d-programming-language-tutorial-1617.html)
Lua (programming language) Tutorial (https://www.wisdomjobs.com/e-university/lua-programming-language-tutorial-1757.html)	

C Related Interview Questions

DBMS Interview Questions (https://www.wisdomjobs.com/e-university/dbms-interview-questions.html)	C++ Interview Questions (https://www.wisdomjobs.com/e-university/c-plus-plus-interview-questions.html)
C & Data Structures Interview Questions (https://www.wisdomjobs.com/e-university/c-data-structures-interview-questions.html)	Java Interview Questions (https://www.wisdomjobs.com/e-university/java-interview-questions.html)

()

☰

(https://www.wisdomjobs.com/)

Search for Jobs...

C Related Interview Questions

Go (programming language) Interview Questions (https://www.wisdomjobs.com/e-university/go-programming-language-interview-questions.html)	F Sharp (programming language) Interview Questions (https://www.wisdomjobs.com/e-university/f-sharp-programming-language-interview-questions.html)
C preprocessor Interview Questions (https://www.wisdomjobs.com/e-university/c-preprocessor-interview-questions.html)	R Programming language Interview Questions (https://www.wisdomjobs.com/e-university/r-programming-language-interview-questions.html)
D Programming Language Interview Questions (https://www.wisdomjobs.com/e-university/d-programming-language-interview-questions.html)	Lua (programming language) Interview Questions (https://www.wisdomjobs.com/e-university/lua-programming-language-interview-questions.html)
Embedded C Interview Questions (https://www.wisdomjobs.com/e-university/embedded-c-interview-questions.html)	

C Related Practice Tests

DBMS Practice Tests (https://www.wisdomjobs.com/e-university/dbms-practice-tests-218-327348)	C++ Practice Tests (https://www.wisdomjobs.com/e-university/c-plus-plus-practice-tests-219-327076)
C & Data Structures Practice Tests (https://www.wisdomjobs.com/e-university/c-data-structures-practice-tests-321-327320)	Go (programming language) Practice Tests (https://www.wisdomjobs.com/e-university/go-programming-language-practice-tests-1306-327862)
C preprocessor Practice Tests (https://www.wisdomjobs.com/e-university/c-preprocessor-practice-tests-1532-327914)	

List of Tutorials

Developers Best Practices Tutorial NEW (https://www.wisdomjobs.com/e-university/developers-best-practices-tutorial-3129.html)

YAML Tutorial NEW (https://www.wisdomjobs.com/e-university/yaml-tutorial-3120.html)

Salesforce Tutorial NEW (https://www.wisdomjobs.com/e-university/salesforce-tutorial-3117.html)

Adobe Robohelp Tutorial (https://www.wisdomjobs.com/e-university/adobe-robohelp-tutorial-3114.html)

()

Sublime Text Tutorial (https://www.wisdomjobs.com/e-university/sublime-text-tutorial-3104.html)

GItlab Tutorial (https://www.wisdomjobs.com/e-university/gitlab-tutorial-3096.html)

(https://www.wisdomjobs.com/)

Adobe InDesign CC Tutorial (https://www.wisdomjobs.com/e-university/adobe-indesign-cc-tutorial-3092.html)

SaltStack Tutorial (https://www.wisdomjobs.com/e-university/saltstack-tutorial-3063.html)

Search for Jobs...

Read More (https://www.wisdomjobs.com/e-university/all-skillsets.html)

List of Topics

Summary NEW (https://www.wisdomjobs.com/e-university/developers-best-practices-tutorial-3129/summary-28360.html)

Career Planning in best practice NEW (https://www.wisdomjobs.com/e-university/developers-best-practices-tutorial-3129/career-planning-in-best-practice-28359.html)

Managing Managers NEW (https://www.wisdomjobs.com/e-university/developers-best-practices-tutorial-3129/managing-managers-28358.html)

Stress Management NEW (https://www.wisdomjobs.com/e-university/developers-best-practices-tutorial-3129/stress-management-28357.html)

Eager to Learn (https://www.wisdomjobs.com/e-university/developers-best-practices-tutorial-3129/eager-to-learn-28356.html)

Handy Tools & Techniques (https://www.wisdomjobs.com/e-university/developers-best-practices-tutorial-3129/handy-tools-techniques-28355.html)

Keep the Assets Safely (https://www.wisdomjobs.com/e-university/developers-best-practices-tutorial-3129/keep-the-assets-safely-28354.html)

Testing is the Religion (https://www.wisdomjobs.com/e-university/developers-best-practices-tutorial-3129/testing-is-the-religion-28353.html)

Read More (https://www.wisdomjobs.com/e-university)

Interview Questions

Cheque Truncation System Interview Questions NEW (https://www.wisdomjobs.com/e-university/cheque-truncation-system-interview-questions.html)

ECS Interview Questions NEW (https://www.wisdomjobs.com/e-university/ecs-interview-questions.html)

RTGS Interview Questions NEW (https://www.wisdomjobs.com/e-university/rtgs-interview-questions.html)

Private Equity Interview Questions NEW (https://www.wisdomjobs.com/e-university/private-equity-interview-questions.html)

Excel Formulas Interview Questions NEW (https://www.wisdomjobs.com/e-university/excel-formulas-interview-questions.html)

Infrared Sensor Interview Questions (https://www.wisdomjobs.com/e-university/infrared-sensor-interview-questions.html)

Sahi Interview Questions (https://www.wisdomjobs.com/e-university/sahi-interview-questions.html)

Riot Js Interview Questions (https://www.wisdomjobs.com/e-university/riot-js-interview-questions.html)

Read More (https://www.wisdomjobs.com/e-university/all-skillsets-interview-questions.html)

ABOUT US

About Wisdom Jobs

Contact US (https://www.wisdomjobs.com/contact-us.html)

Privacy Policy (https://www.wisdomjobs.com/privacy-and-policy.php)

Terms of Use (https://www.wisdomjobs.com/terms-and-conditions.php)

Report a problem (https://www.wisdomjobs.com/reportproblem.html)

Help (https://www.wisdomjobs.com/help)

TOP COMPANY JOBS

HDFC Careers (https://www.wisdomjobs.com/hdfc-jobs)

Infosys Careers (https://www.wisdomjobs.com/infosys-jobs)

Mphasis Careers (https://www.wisdomjobs.com/mphasis-jobs)

Axis Bank Careers (https://www.wisdomjobs.com/axis-bank-jobs)

Ashok Leyland Careers (https://www.wisdomjobs.com/ashok-leyland-jobs)

AEGIS Careers (https://www.wisdomjobs.com/aegis-jo

()

Press Corner
Html site Map (https://www.wisdomjobs.com/sitemap.html)
(https://www.wisdomjobs.com/)

Search for Jobs...

Convergys Careers (https://www.wisdomjobs.com/convergys-jobs)
Adobe Careers (https://www.wisdomjobs.com/adobe-jobs)
ICICI Bank Careers (https://www.wisdomjobs.com/icici-bank-jobs)
Indigo Careers (https://www.wisdomjobs.com/indigo-jobs)
Spicejet Careers (https://www.wisdomjobs.com/spicejet-jobs)
HSBC Careers (https://www.wisdomjobs.com/hsbc-jobs)

TOP CATEGORY JOBS

Govt Jobs (https://www.wisdomjobs.com/govtjobs/)
Freshers world (https://www.wisdomjobs.com/freshers-world)
Today walkins (https://www.wisdomjobs.com/today-walkins)
Sarkari Result (https://www.wisdomjobs.com/sarkari-result)
Agriculture Jobs (https://www.wisdomjobs.com/agriculture-jobs)
Defence Jobs (https://www.wisdomjobs.com/defence-jobs)
NGO Jobs (https://www.wisdomjobs.com/ngo-jobs)
Real Estate Jobs (https://www.wisdomjobs.com/real-estate-jobs)
Shipping Jobs (https://www.wisdomjobs.com/shipping-jobs)
Java Jobs (https://www.wisdomjobs.com/java-jobs)
Education Jobs (https://www.wisdomjobs.com/education-jobs)
Journalism Jobs (https://www.wisdomjobs.com/journalism-jobs)
SAP Jobs (https://www.wisdomjobs.com/sap-jobs)
IT Software Jobs (https://www.wisdomjobs.com/it-software-jobs)

JOBS IN TOP LOCATIONS

Jobs in Delhi (https://www.wisdomjobs.com/jobs-in-delhi)
Jobs in Bangalore (https://www.wisdomjobs.com/jobs-in-bangalore)
Jobs in Mumbai (https://www.wisdomjobs.com/jobs-in-mumbai)
Jobs in Pune (https://www.wisdomjobs.com/jobs-in-pune)
Jobs in Chennai (https://www.wisdomjobs.com/jobs-in-chennai)
Jobs in Hyderabad (https://www.wisdomjobs.com/jobs-in-hyderabad-secunderabad)
Jobs in Kolkata (https://www.wisdomjobs.com/jobs-in-kolkata)
Jobs in Chandigarh (https://www.wisdomjobs.com/jobs-in-chandigarh)
Jobs in Gurgaon (https://www.wisdomjobs.com/jobs-in-gurgaon)
Jobs in Noida (https://www.wisdomjobs.com/jobs-in-noida)
Jobs in Ahmedabad (https://www.wisdomjobs.com/jobs-in-ahmedabad)
Browse All Jobs (https://www.wisdomjobs.com/browse-alljobs)

ASSESSMENTS

Pragnya Meter (https://www.wisdomjobs.com/pragnyameter/)

TUTORIALS

E-University (https://www.wisdomjobs.com/e-university/aboutus.html)
Skill Sets (https://www.wisdomjobs.com/e-university/all-skillsets.html)
Practice Tests (https://www.wisdomjobs.com/e-university/onlineexam.html)

SERVICES

Resume Writing (https://www.wisdomjobs.com/resumewriting/resume-writing-for-freshers)

()

C Tutorial

()