Contents

[What is the start-up code? 5](#_heading=h.gjdgxs)

[What are the start-up code steps? 6](#_heading=h.30j0zll)

[What is the cause of the stack overflow? 6](#_heading=h.1fob9te)

[What is ISR? 6](#_heading=h.3znysh7)

[What is segmentation fault in C? 7](#_heading=h.2et92p0)

[What are the common causes of segmentation fault in C? 7](#_heading=h.tyjcwt)

[What is the difference between a macro and a function? 8](#_heading=h.3dy6vkm)

[What is the difference between const and macro? 8](#_heading=h.1t3h5sf)

[What is the typedef and enum? 9](#_heading=h.4d34og8)

[What is a NULL pointer? 9](#_heading=h.2s8eyo1)

[What is the meaning of below declarations? 9](#_heading=h.17dp8vu)

[What is the difference between global and static global variables? 9](#_heading=h.3rdcrjn)

[Differentiate between an internal static and external static variable? 10](#_heading=h.26in1rg)

[What is the difference between pass by value by reference in c and pass by reference in c? 10](#_heading=h.lnxbz9)

[What is a reentrant function? 10](#_heading=h.35nkun2)

[What is the inline function? 11](#_heading=h.44sinio)

[What is the advantage and disadvantage of the inline function? 11](#_heading=h.2jxsxqh)

[What is the volatile keyword? 12](#_heading=h.z337ya)

[What is the use of volatile keyword? 12](#_heading=h.3j2qqm3)

[Can a variable be both constant and volatile in C? 12](#_heading=h.1y810tw)

[What is interrupt latency? 12](#_heading=h.4i7ojhp)

[How do you measure interrupt latency? 12](#_heading=h.2xcytpi)

[How to reduce the interrupt latency? 13](#_heading=h.1ci93xb)

[What are the causes of Interrupt Latency? 13](#_heading=h.3whwml4)

[What is NVIC in ARM Cortex? 14](#_heading=h.2bn6wsx)

[What is the difference between Bit Rate and Baud Rate? 15](#_heading=h.qsh70q)

[The Proper place to use the volatile keyword? 15](#_heading=h.3as4poj)

[Difference between RISC and CISC processor? 16](#_heading=h.1pxezwc)

[What is the difference between Asynchronous and Synchronous Communication? 18](#_heading=h.49x2ik5)

[What is the difference between RS232 and RS485? 19](#_heading=h.2p2csry)

[What is the difference between malloc and calloc? 20](#_heading=h.147n2zr)

[What is the purpose of realloc()? 20](#_heading=h.3o7alnk)

[What are dangling pointers? 21](#_heading=h.23ckvvd)

[What is the wild pointer? 21](#_heading=h.ihv636)

[When should we use const in a C program? 21](#_heading=h.32hioqz)

[What is the advantage of a void pointer in C? 22](#_heading=h.1hmsyys)

[Explain the Memory layout of a C and Embedded C program? 22](#_heading=h.41mghml)

[A simple program and its analysis: 25](#_heading=h.vx1227)

[**Other C Interview questions** 27](#_heading=h.3fwokq0)

[ What](#_heading=h.1v1yuxt) is virtual memory? 27

[ Where can the function pointers be used? 27](#_heading=h.4f1mdlm)

[ What is dynamic memory fragmentation? 27](#_heading=h.2u6wntf)

[ What is the return value of malloc (0)? 27](#_heading=h.19c6y18)

[ What is the memory leak in C? 27](#_heading=h.3tbugp1)

[ What is meant by structure padding? 27](#_heading=h.28h4qwu)

[ Write a multiline macro statement? 27](#_heading=h.nmf14n)

[ How can object realization achieved in C? 27](#_heading=h.37m2jsg)

[ Can math operations be performed on a void pointer? 27](#_heading=h.1mrcu09)

[ Can Module operation be performed on floating point numbers? 27](#_heading=h.46r0co2)

[**C++ Interview questions** 30](#_heading=h.2lwamvv)

[ What is difference between in C and C++? 30](#_heading=h.111kx3o)

[ Why is the size of an empty class not zero in C++? 30](#_heading=h.3l18frh)

[ What are C++ access, specifiers? 30](#_heading=h.206ipza)

[ What are the various OOPs concepts in C++? 30](#_heading=h.4k668n3)

[ What are the differences between a C++ struct and C++ class? 30](#_heading=h.2zbgiuw)

[ What is difference between new and malloc? 30](#_heading=h.1egqt2p)

[ What is a namespace? 30](#_heading=h.3ygebqi)

[ What is Overriding? 30](#_heading=h.2dlolyb)

[ How to create .dll in C++ and how to link .dll in your code? 30](#_heading=h.sqyw64)

[ What is overloading? 30](#_heading=h.3cqmetx)

[ How to create and use a reference variable in c++? 30](#_heading=h.1rvwp1q)

[ Can you overload a function based only on whether a parameter is a value or a reference? 30](#_heading=h.4bvk7pj)

[ What is difference between function overloading and Operator Overloading? 30](#_heading=h.2r0uhxc)

[ Can we access private data members of a class without using a member or a friend function? 30](#_heading=h.1664s55)

[ What is the use of inline function? 30](#_heading=h.3q5sasy)

[ how to access derived class function from base class object without using virtual function? 30](#_heading=h.25b2l0r)

[ Distinguish between shallow copy and deep copy. 30](#_heading=h.kgcv8k)

[ Friend class and function in C++ 30](#_heading=h.34g0dwd)

[ what is difference between constructor and destructor? 30](#_heading=h.1jlao46)

[ How are .h files loaded and linked with their .c files? 30](#_heading=h.43ky6rz)

[ What Is Inheritance? 30](#_heading=h.2iq8gzs)

[ What is “this” pointer? 30](#_heading=h.xvir7l)

[ What is difference between delete and free? 30](#_heading=h.3hv69ve)

[ What is encapsulation? 30](#_heading=h.1x0gk37)

[ What is static Member in C++? 30](#_heading=h.4h042r0)

[ What is a pure virtual function in C++? 30](#_heading=h.2w5ecyt)

[ Count the number of words, characters, and lines in a file? 30](#_heading=h.1baon6m)

[ Any fundamental difference between source and header files in C? 31](#_heading=h.3vac5uf)

[ Can a C++ class member function template be virtual? 31](#_heading=h.2afmg28)

[ What is the Diamond problem? How can we get around it? 31](#_heading=h.pkwqa1)

[ What are the advantages of inheritance? 31](#_heading=h.39kk8xu)

[ why virtual functions cannot be static in C++? 31](#_heading=h.1opuj5n)

[ How does the compilation/linking process work? 31](#_heading=h.48pi1tg)

[ Can you explain the order of execution in constructor initialization list? 31](#_heading=h.2nusc19)

[ How to make a C++ class whose objects can only be dynamically allocated? 31](#_heading=h.1302m92)

[ Is it possible to have Virtual Constructor? If yes, how? If not, Why not possible? 31](#_heading=h.3mzq4wv)

[ What is a constructor? Is default constructor exist in C++? 31](#_heading=h.2250f4o)

[ Can a constructor throw an exception? How to handle the error when the constructor fails? 31](#_heading=h.haapch)

[ What is the difference between a copy constructor and an overloaded assignment operator? 31](#_heading=h.319y80a)

[ When are copy constructors called in C++? 31](#_heading=h.1gf8i83)

[ Why copy constructor takes the parameter as a reference in C++? 31](#_heading=h.40ew0vw)

[ What do you mean by inline function? 31](#_heading=h.2fk6b3p)

[ What is a template function? 31](#_heading=h.upglbi)

[ Can we combine c and C++ code? 31](#_heading=h.3ep43zb)

[ What is shared pointer in C++? 31](#_heading=h.1tuee74)

[ What is typecasting? 31](#_heading=h.4du1wux)

[ Why is a pure virtual function initialized by 0? 31](#_heading=h.2szc72q)

[ How to create multidimensional arrays using new? 31](#_heading=h.184mhaj)

[ When should static\_cast, dynamic\_cast, const\_cast and reinterpret\_cast be used? 31](#_heading=h.3s49zyc)

[ Can I free () pointers allocated with new? 31](#_heading=h.279ka65)

[ How do I convert an integer to a string in C++? 31](#_heading=h.meukdy)

[ What is the difference between a pointer and a reference? 31](#_heading=h.36ei31r)

[ When should I use references, and when should I use pointers? 31](#_heading=h.1ljsd9k)

[ What is the assignment operator in C++? 31](#_heading=h.45jfvxd)

[ What does the explicit keyword mean? 31](#_heading=h.2koq656)

[ When do we need to create virtual destructor in our class? 32](#_heading=h.zu0gcz)

[ What is Polymorphism? 32](#_heading=h.3jtnz0s)

[ What is the effect of extern “C” in C++? 32](#_heading=h.1yyy98l)

[ What are the different types of polymorphism in C++? 32](#_heading=h.4iylrwe)

[ Can a copy constructor accept an object of the same class as a parameter, in place of reference of the object? If No, why not possible? 32](#_heading=h.2y3w247)

[ What about Virtual Destructor? 32](#_heading=h.1d96cc0)

[ What is conversion constructor? 32](#_heading=h.3x8tuzt)

[ How do you access the static member of a class? 32](#_heading=h.2ce457m)

[ When do we use the Initializer List in C++? 32](#_heading=h.rjefff)

[ Is it possible to overload the destructor the class? 32](#_heading=h.3bj1y38)

[ Which is faster: Stack allocation or Heap allocation 32](#_heading=h.1qoc8b1)

[ What is auto pointer in C++? 32](#_heading=h.4anzqyu)

[ What is the smart pointer in C++? 32](#_heading=h.2pta16n)

[ Can I delete pointers allocated with malloc()? 32](#_heading=h.14ykbeg)

[ Can I use realloc() on pointers allocated via new? 32](#_heading=h.3oy7u29)

[ Can I call the destructor explicitly? 32](#_heading=h.243i4a2)

[ Can one constructor of a class call another constructor of the same class to initialize the this object? 32](#_heading=h.j8sehv)

[ What is the difference between an array and a list? 32](#_heading=h.338fx5o)

[ What is a “pure virtual” member function and when it is used? 32](#_heading=h.1idq7dh)

[ What are VTABLE and VPTR? 32](#_heading=h.42ddq1a)

[ How to handle the exception in C++? 32](#_heading=h.2hio093)

[ What is a Memory Leak? 32](#_heading=h.wnyagw)

[ why static functions cannot access non-static variables? 32](#_heading=h.3gnlt4p)

[ What is a dangling pointer? 32](#_heading=h.1vsw3ci)

[ What is “mutable” keyword in C++? 32](#_heading=h.4fsjm0b)

[ What are the debugging methods you use when came across a problem? 32](#_heading=h.2uxtw84)

[ STL Containers – What are the types of STL containers? 32](#_heading=h.1a346fx)

[**RTOS:** 33](#_heading=h.3u2rp3q)

[What are the important criterions for selecting an RTOS? 33](#_heading=h.2981zbj)

[What are the rules to follow when writing critical instructions? 33](#_heading=h.odc9jc)

[How two or more processes or threads communicate with each other? 33](#_heading=h.38czs75)

[What is the scheduling used in RTOS? 33](#_heading=h.1nia2ey)

[What the 3 essential things that an RTOS provide? 33](#_heading=h.47hxl2r)

[What are the types of scheduling? 33](#_heading=h.2mn7vak)

[What is a Process, Thread, and tasks? 34](#_heading=h.11si5id)

**Embedded C Interview questions**

# What is #pragma Directive in C/C++?

The preprocessor directive #pragma is used to provide the additional information to the compiler in C/C++ language. This is used by the compiler to provide some special features.

Here is the syntax of #pragma directive in C/C++ language,

#pragma token\_name

# What is the start-up code?

A start-up code is called prior to the main function, it creates a basic platform for the application. It is a small block of code that is written in assembly language.

**There are following parts of the start-up code.**

* Declaration of the Stack area.
* Declaration of the Heap area.
* Vector table.
* Reset handler code.
* Other exception handler code.

# What are the start-up code steps?

Start-up code for C programs usually consists of the following actions, performed in the order described:

* Disable all interrupts.
* Copy any initialized data from ROM to RAM.
* Zero the uninitialized data area.
* Allocate space for and initialize the stack.
* Initialize the processor’s stack pointer.
* Create and initialize the heap.
* Enable interrupts.
* Call main.

# What is the cause of the stack overflow?

In the embedded application we have a little amount of stack memory as compare to the desktop application. So we have to work on embedded application very carefully either we can face the stack overflow issues that can be a cause of the application crash.

**Here, I have mentioned some causes of unwanted use of the stack.**

* Improper use of the recursive function.
* Passing to much arguments in the function.
* Passing a structure directly into a function.
* Nested function calls.
* Creating a huge size local array.

# What is ISR?

An ISR refers to the Interrupt Service Routines. These are procedures stored at specific memory addresses which are called when a certain type of interrupt occurs. The Cortex-M processors family has the NVIC that manage the execution of the interrupt.

# What is segmentation fault in C?

A segmentation fault is a common problem that causes programs to crash. A core file (core dumped file) also associated with segmentation fault that is used by the developer to find the root cause of the crashing (segmentation fault).

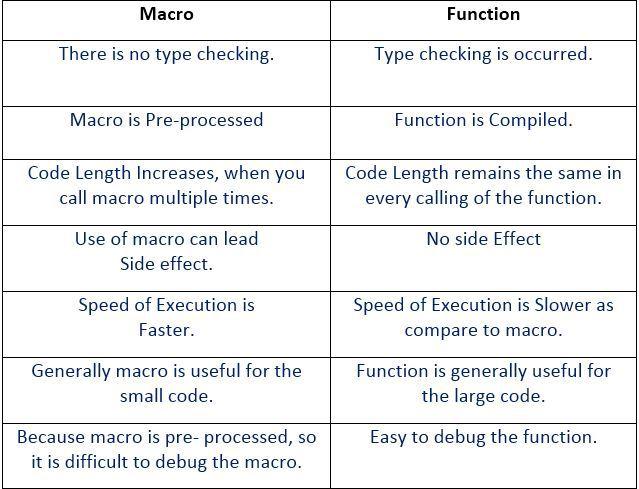
Generally, the segmentation fault occurs when a program tried to access a memory location that it is not allowed to access or tried to access a memory location in a way that is not allowed (tried to access read-only memory).

# What are the common causes of segmentation fault in C?

There are many reasons for the segmentation fault, here I am listing some common causes of the segmentation fault.

* Dereferencing NULL pointers.
* Tried to write read-only memory (such as code segment).
* Trying to access a nonexistent memory address (outside process’s address space).
* Trying to access memory the program does not have rights to (such as kernel structures in process context).
* Sometimes dereferencing or assigning to an uninitialized pointer (because might point an invalid memory) can be the cause of the segmentation fault.
* Dereferencing the freed memory (after calling the free function) can also be caused by the segmentation fault.
* A stack overflow is also caused by the segmentation fault.
* A buffer overflow (try to access the array beyond the boundary) is also cause of the segmentation fault.

# What is the difference between a macro and a function?



# What is the difference between const and macro?

* The const keyword is handled by the compiler (The const keyword is compiler-enforced and says that program could not change the value of the object that means it makes the object nonmodifiable type.), in another hand, a macro is handled by the preprocessor directive.
* const is a qualifier that is modified the behavior of the identifier, but macro is preprocessor directive.
* There is type checking is occurred with const keyword but does not occur with #define.
* const is scoped by C block, #define applies to a file.
* const can be passed as a parameter (as a pointer) to the function. In case of call by reference, it prevents to modify the passed object value.

# What is the typedef and enum?

The C language provides a very important keyword **typedef** for defining a new name for existing types. The typedef is the compiler directive mainly use with user-defined data types (structure, union or enum) to reduce their complexity and increase the code readability and portability.

In C language [enum](https://aticleworld.com/seven-important-points-enum-c-language/) is user-defined data type and it consists a set of named constant integer.

# What is a NULL pointer?

According to C standard, an integer constant expression with the value 0, or such an expression cast to type void \*, is called a null pointer constant. If a **null pointer** constant is converted to a pointer type, the resulting pointer, called a null pointer.

# What is the meaning of below declarations?

1. const int a;
2. int const a;
3. const int \*a;
4. int \* const a;
5. int const \* a const;
6. The “a” is a constant integer.
7. Like first, “a” is a constant integer.
8. Here “a” is a pointer to a const integer, the value of the integer is not modifiable, but the pointer
9. Here “a” is a const pointer to an integer, the value of the pointed integer is modifiable, but the pointer is not modifiable.
10. Here “a” is a const pointer to a const integer that means the value of pointed integer and pointer both are not modifiable.

# What is the difference between global and static global variables?

In simple word, they have different linkage.

A static global variable            ===>>> **internal linkage.**  
A non-static global variable ===>>> **external** **linkage.**

So global variable can be accessed outside of the file, but the static global variable only accesses within the file in which it is declared.

# Differentiate between an internal static and external static variable?

In C language, the external static variable has the internal linkage and internal static variable has no linkage. So, the life of both variable throughout the program but scope will be different.

An external static variable ===>>> **internal linkage.**  
A internal static variable   ===>>>**none**

# What is the difference between pass by value by reference in c and pass by reference in c?

**Pass By Value:**

* In this method value of the variable is passed. Changes made to formal will not affect the actual parameters.
* Different memory locations will be created for both variables.
* Here there will be temporary variable created in the function stack which does not affect the original variable.

**Pass By Reference:**

* In Pass by reference, an address of the variable is passed to a function.
* Whatever changes made to the formal parameter will affect the value of actual parameters (a variable whose address is passed).
* Both formal and actual parameter shared the same memory location.
* it is useful when you required to returns more than 1 values.

# What is a reentrant function?

In computing, a computer program or subroutine is called reentrant if it can be interrupted in the middle of its execution and then safely be called again (“re-entered”) before its previous invocations’ complete execution. The interruption could be caused by an internal action such as a jump or call, or by an external action such as an interrupt or signal. Once the reentered invocation completes, the previous invocations will resume correct execution.

# What is the inline function?

An inline keyword is a compiler directive that only suggests the compiler to substitute the body of the function at the calling the place. It is an optimization technique used by the compilers to reduce the overhead of function calls.

# What is the advantage and disadvantage of the inline function?

There are few important advantage and disadvantage of the inline function.

**Advantages:-**  
1) It saves the function calling overhead.  
2) It also saves the overhead of variables push/pop on the stack, while function calling.  
3) It also saves the overhead of return call from a function.  
4) It increases locality of reference by utilizing instruction cache.  
5) After inlining compiler can also apply intraprocedural optimization if specified. This is the most important one, in this way compiler can now focus on dead code elimination, can give more stress on branch prediction, induction variable elimination etc.

**Disadvantages:-**  
1) May increase function size so that it may not fit in the cache, causing lots of cache miss.  
2) After inlining function, if variables number which are going to use register increases than they may create overhead on register variable resource utilization.  
3) It may cause compilation overhead as if somebody changes code inside an inline function then all calling location will also be compiled.  
4) If used in the header file, it will make your header file size large and may also make it unreadable.  
5) If somebody used too many inline functions resultant in a larger code size than it may cause thrashing in memory. More and number of page fault bringing down your program performance.  
6) It’s not useful for an embedded system where large binary size is not preferred at all due to memory size constraints.

# What is the volatile keyword?

The volatile keyword is a type qualifier that prevents the objects from the compiler optimization. According to C standard, an object that has volatile-qualified type may be modified in ways unknown to the implementation or have other unknown side effects. You can also say that the value of the volatile-qualified object can be changed at any time without any action being taken by the code. If an object is qualified by the **volatile qualifier**, the compiler reloads the value from memory each time it is accessed by the program that means it prevents from to cache a variable into a register. Reading the value from the memory is the only way to check the unpredictable change of the value.

# What is the use of volatile keyword?

The volatile keyword is mainly used where we directly deal with GPIO, interrupt or flag Register. It is also used where a global variable or buffer is shared between the threads.

# Can a variable be both constant and volatile in C?

**Syntax of declaration,**

**int volatile \* const PortRegister;**

# What is interrupt latency?

It is an important question that is asked by the interviewer to test the understanding of Interrupt. Basically, interrupt latency is the number of clock cycles that is taken by the processor to respond to an interrupt request. These number of the clock cycle is count between the assertions of the interrupt request and first instruction of the interrupt handler.

# How do you measure interrupt latency?

With the help of the oscilloscope, we can measure the interrupt latency. You need to take following steps.

* First takes two GPIOs.
* Configure one GPIO to generate the interrupt and second for the toggling (if you want you can attach an LED).
* Monitor the PIN (using the oscilloscope or analyzer) which you have configured to generate the interrupt.
* Also, monitor (using the oscilloscope or analyzer) the second pin which is toggled at the beginning of the interrupt service routine.
* When you will generate the interrupt then the signal of the both GPIOs will change.

The interval between the two signals (interrupt latency) may be easily read from the instrument.

# How to reduce the interrupt latency?

The interrupt latency depends on many factors, some factor I am mentioning in below statements.

* Platform and interrupt controller.
* CPU clock speed.
* Timer frequency
* Cache configuration.
* Application program.

So, using the proper selection of platform and processor we can easily reduce the interrupt latency. We can also reduce the interrupt latency by making the ISR shorter and avoid to calling a function within the ISR.

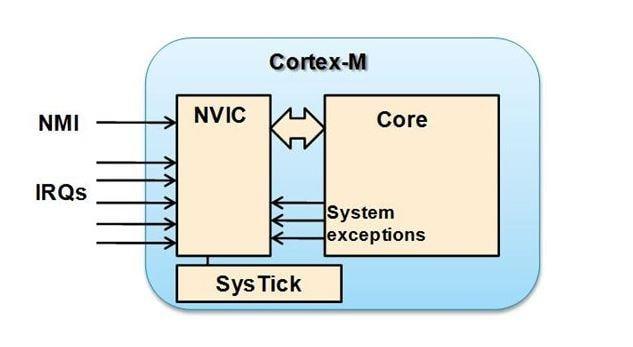
# What are the causes of Interrupt Latency?

* The first delay is typically caused by hardware: The interrupt request signal needs to be synchronized to the CPU clock. Depending on the synchronization logic, up to 3 CPU cycles may expire before the interrupt request has reached the CPU core.
* The CPU will typically complete the current instruction, which may take several cycles. On most systems, divide, push-multiple or memory-copy instructions are the most time-consuming instructions to execute. On top of the cycles required by the CPU, additional cycles are often required for memory accesses. In an ARM7 system, the instruction STMDB SP!,{R0-R11, LR} typically is the worst case instruction, storing 13 registers of 32-bits each to the stack, and takes 15 clock cycles to complete.
* The memory system may require additional cycles for wait states.
* After completion of the current instruction, the CPU performs a mode switch or pushes registers on the stack (typically PC and flag registers). Modern CPUs such as ARM generally perform a mode switch, which takes fewer CPU cycles than saving registers.
* Pipeline fill: Most modern CPUs are pipelined. Execution of an instruction happens in various stages of the pipeline. An instruction is executed when it has reached its final stage of the pipeline. Since the mode switch has flushed the pipeline, a few extra cycles are required to refill the pipeline.

# What is NVIC in ARM Cortex?

The Nested Vector Interrupt Controller (NVIC) in the Cortex-M processor family is an example of an interrupt controller with extremely flexible interrupt priority management. It enables programmable priority levels, automatic nested interrupt support, along with support for multiple interrupt masking, whilst still being very easy to use by the programmer.

The Cortex-M3 and Cortex-M4 processors the NVIC supports up to 240 interrupt inputs, with 8 up to 256 programmable priority levels.



# What is the difference between Bit Rate and Baud Rate?

|  |  |
| --- | --- |
| **Bit Rate** | **Baud Rate** |
| Bit rate is the number of bits per second. | Baud rate is the number of signal units per second. |
| It determines the number of bits traveled per second. | It determines how many times the state of a signal is changing. |
| Cannot determine the bandwidth. | It can determine how much bandwidth is required to send the signal. |
| This term generally used to describe the processor efficiency. | This term generally used to describe the data transmission over the channel. |
| Bit rate = baud rate x the number of bits per signal unit | Baud rate = bit rate / the number of bits per signal unit |

# The Proper place to use the volatile keyword?

Here I am pointing some important places where we need to use the volatile keyword.

* Accessing the memory-mapped peripherals register or hardware status register
* Sharing the global variables or buffers between the multiple threads.
* Accessing the global variables in an interrupt routine or signal handler.

# Difference between RISC and CISC processor?

The RISC (reduced instruction set computer) and CISC (Complex instruction set computer) are the processors ISA (instruction set architecture).

|  |  |  |
| --- | --- | --- |
|  | **RISC** | **CISC** |
| Acronym | It stands for ‘Reduced Instruction Set Computer’. | It stands for ‘Complex Instruction Set Computer’. |
| Definition | The RISC processors have a smaller set of instructions with few addressing nodes. | The CISC processors have a larger set of instructions with many addressing nodes. |
| Memory unit | It has no memory unit and uses a separate hardware to implement instructions. | It has a memory unit to implement complex instructions. |
| Program | It has a hard-wired unit of programming. | It has a micro-programming unit. |
| Design | It is a complex complier design. | It is an easy complier design. |
| Calculations | The calculations are faster and precise. | The calculations are slow and precise. |
| Decoding | Decoding of instructions is simple. | Decoding of instructions is complex. |
| Time | Execution time is very less. | Execution time is very high. |
| External memory | It does not require external memory for calculations. | It requires external memory for calculations. |
| Pipelining | Pipelining does function correctly. | Pipelining does not function correctly. |
| Stalling | Stalling is mostly reduced in processors. | The processors often stall. |
| Code expansion | Code expansion can be a problem. | Code expansion is not a problem. |
| Disc space | The space is saved. | The space is wasted. |
| Applications | Used in high-end applications such as video processing, telecommunications and image processing. | Used in low-end applications such as security systems, home automations, etc. |

# What is the difference between Asynchronous and Synchronous Communication?

|  |  |
| --- | --- |
| **Asynchronous Communication** | **Synchronous Communication** |
| There is no common clock signal between the sender and receivers. | Communication is done by a shared clock. |
| Sends 1 byte or character at a time. | Sends data in the form of blocks or frames. |
| Slow as compare to synchronous communication. | Fast as compare to asynchronous communication. |
| Overhead due to start and stop bit. | Less overhead. |
| Ability to communicate long distance. | Less as compared to asynchronous communication. |
| A start and stop bit used for the data synchronization. | A shared clock is used for the data synchronization. |
| Economical | Costly |
| RS232, RS485 | I2C, SPI. |

# What is the difference between RS232 and RS485?

|  |  |  |
| --- | --- | --- |
| **Parameter** | **RS232** | **RS485** |
| Line configuration | Single –ended | differential |
| Numbers of devices | 1 transmitter 1 receiver | 32 transmitters 32 receivers |
| Mode of operation | Simplex or full duplex | Simplex or half duplex |
| Maximum cable length | 50 feet | 4000 feet |
| Maximum data rate | 20 Kbits/s | 10 Mbits/s |
| signaling | unbalanced | balanced |
| Typical logic levels | +-5 ~ +-15V | +-1.5 ~ +-6V |
| Minimum receiver input impedance | 3 ~ 7 K-ohm | 12 K-ohm |
| Receiver sensitivity | +-3V | +-200mV |

# What is the difference between malloc and calloc?

The malloc and calloc are memory management functions. They are used to allocate memory dynamically. Basically, there is no actual difference between calloc and malloc except that the memory that is allocated by calloc is initialized with 0.

In C language, calloc function initialize the all allocated space bits with zero but malloc does not initialize the allocated memory. These both functions also has a difference regarding their number of arguments, malloc take one argument but calloc takes two.

# What is the purpose of realloc()?

The realloc function is used to resize the allocated block of memory. It takes two arguments first one is a pointer to previously allocated memory and the second one is the newly requested size.

The calloc function first deallocates the old object and allocates again with newly specified size. If the new size is lesser to the old size, the contents of the newly allocated memory will be same as prior but if any bytes in the newly created object goes beyond the old size, the values of the exceeded size will be indeterminate.

***Syntax:***  
void \*realloc(void \*ptr, size\_t size);

# What are dangling pointers?

Generally, **daggling pointers** arise when the referencing object is deleted or deallocated, without changing the value of the pointers. It creates the problem because the pointer is still pointing the memory that is not available. When the user tries to dereference the daggling pointers than it shows the undefined behavior and can be the cause of the segmentation fault.

# What is the wild pointer?

A pointer that is not initialized properly prior to its first use is known as the **wild pointer**. Uninitialized pointers behavior is totally undefined because it may point some arbitrary location that can be the cause of the program crash, that’s is the reason it is called a wild pointer.

In the other word, we can say every **pointer in programming languages** that are not initialized either by the compiler or programmer begins as a wild pointer.

***Note:****Generally, compilers warn about the wild pointer.*

***Syntax,***  
int \*piData; //piData is wild pointer.

# When should we use const in a C program?

There are following places where we need to use the const keyword in the programs.

* In call by reference function argument, if you don’t want to change the actual value which has passed in function.  
  Eg.  
  **int PrintData ( const char \*pcMessage);**
* In some places, const is better then macro because const handle by the compiler and have a type checking.  
  Eg.  
  **const int ciData = 100;**
* In the case of I/O and memory mapped register const is used with the volatile qualifier for efficient access.  
  Eg.  
  **const volatile uint32\_t \*DEVICE\_STATUS = (uint32\_t \*) 0x80102040;**
* When you don’t want to change the value of an initialized variable.

# What is the advantage of a void pointer in C?

There are following advantages of a void pointer in c.

* Using the void pointer we can create a generic function that can take arguments of any data type. **The memcpy and memmove library function** are the best examples of the generic function, using these function we can copy the data from the source to destination.  
  e.g.  
  **void \* memcpy ( void \* dst, const void \* src, size\_t num );**
* We have already know that void pointer can be converted to another data type that is the reason malloc, calloc or realloc library function return void \*. Due to the void \* these functions are used to allocate memory to any data type.
* Using the void \* we can create a generic linked list. For more information see this link: How to create generic Link List.

# Explain the Memory layout of a C and Embedded C program?

A screenshot of a cell phone

Description automatically generated

Segment Memory Contents

.data RAM Explicitly initialized variables with static storage duration

.bss RAM Zero-initialized variables with static storage duration

.stack RAM Local variables and function call parameters

.heap RAM Dynamically allocated variables (usually not used in embedded systems)

.rodata ROM const variables with static storage duration. String literals.

.text ROM The program. Integer constants. Initializer lists. Contains Compiled Machine code instructions, operating system, kernel image, BIOS, Bootloader

BSS stands for Block Started by Symbols.

A screenshot of a cell phone

Description automatically generated



A simple program and its analysis:

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA close up of a logo

Description automatically generatedA close up of a logo

Description automatically generatedA screenshot of a cell phone

Description automatically generated

# **Other C Interview questions**

## What is virtual memory?

## Where can the function pointers be used?

## What is dynamic memory fragmentation?

## What is the return value of malloc (0)?

## What is the memory leak in C?

## What is meant by structure padding?

## Write a multiline macro statement?

## How can object realization achieved in C?

## Can math operations be performed on a void pointer?

## Can Module operation be performed on floating point numbers?

we cannot perform a modulo operation on floating points in C.

But we can. There is a function in “math.h” named fmod which has three prototypes namely:

1. double fmod(double x, double y);
2. float fmodf(float x , float y);
3. long double fmodl(long double x, long double y);

# **C++ Interview questions**

## What is difference between in C and C++?

## Why is the size of an empty class not zero in C++?

## What are C++ access, specifiers?

## What are the various OOPs concepts in C++?

## What are the differences between a C++ struct and C++ class?

## What is difference between new and malloc?

## What is a namespace?

## What is Overriding?

## How to create .dll in C++ and how to link .dll in your code?

## What is overloading?

## How to create and use a reference variable in c++?

## Can you overload a function based only on whether a parameter is a value or a reference?

## What is difference between function overloading and Operator Overloading?

## Can we access private data members of a class without using a member or a friend function?

## What is the use of inline function?

## how to access derived class function from base class object without using virtual function?

## Distinguish between shallow copy and deep copy.

## Friend class and function in C++

## what is difference between constructor and destructor?

## How are .h files loaded and linked with their .c files?

## What Is Inheritance?

## What is “this” pointer?

## What is difference between delete and free?

## What is encapsulation?

## What is static Member in C++?

## What is a pure virtual function in C++?

## Count the number of words, characters, and lines in a file?

## Any fundamental difference between source and header files in C?

## Can a C++ class member function template be virtual?

## What is the Diamond problem? How can we get around it?

## What are the advantages of inheritance?

## why virtual functions cannot be static in C++?

## How does the compilation/linking process work?

## Can you explain the order of execution in constructor initialization list?

## How to make a C++ class whose objects can only be dynamically allocated?

## Is it possible to have Virtual Constructor? If yes, how? If not, Why not possible?

## What is a constructor? Is default constructor exist in C++?

## Can a constructor throw an exception? How to handle the error when the constructor fails?

## What is the difference between a copy constructor and an overloaded assignment operator?

## When are copy constructors called in C++?

## Why copy constructor takes the parameter as a reference in C++?

## What do you mean by inline function?

## What is a template function?

## Can we combine c and C++ code?

## What is shared pointer in C++?

## What is typecasting?

## Why is a pure virtual function initialized by 0?

## How to create multidimensional arrays using new?

## When should static\_cast, dynamic\_cast, const\_cast and reinterpret\_cast be used?

## Can I free () pointers allocated with new?

## How do I convert an integer to a string in C++?

## What is the difference between a pointer and a reference?

## When should I use references, and when should I use pointers?

## What is the assignment operator in C++?

## What does the explicit keyword mean?

## When do we need to create virtual destructor in our class?

## What is Polymorphism?

## What is the effect of extern “C” in C++?

## What are the different types of polymorphism in C++?

## Can a copy constructor accept an object of the same class as a parameter, in place of reference of the object? If No, why not possible?

## What about Virtual Destructor?

## What is conversion constructor?

## How do you access the static member of a class?

## When do we use the Initializer List in C++?

## Is it possible to overload the destructor the class?

## Which is faster: Stack allocation or Heap allocation

## What is auto pointer in C++?

## What is the smart pointer in C++?

## Can I delete pointers allocated with malloc()?

## Can I use realloc() on pointers allocated via new?

## Can I call the destructor explicitly?

## Can one constructor of a class call another constructor of the same class to initialize the this object?

## What is the difference between an array and a list?

## What is a “pure virtual” member function and when it is used?

## What are VTABLE and VPTR?

## How to handle the exception in C++?

## What is a Memory Leak?

## why static functions cannot access non-static variables?

## What is a dangling pointer?

## What is “mutable” keyword in C++?

## What are the debugging methods you use when came across a problem?

## STL Containers – What are the types of STL containers?

# **RTOS:**

# What are the important criterions for selecting an RTOS?

* Memory footprint
* Interrupt latency
* API support
* Architecture/HW support

# What are the rules to follow when writing critical instructions?

* 1. Use atomic instructions
  2. Disable interrupts
  3. Keep nesting of functions as it may cause overheads

# How two or more processes or threads communicate with each other?

IPC techniques are - Semaphores, Mutex, Shared Memory, Signal, Message passing, pipes, Sockets, rpc calls

# What is the scheduling used in RTOS?

RTOS have a deterministic scheduler. For the given set of tasks your process will always execute every number of usecs. OS services consume only known/expected amount of time.

# What the 3 essential things that an RTOS provide?

* Task scheduler: determine which task to run and when
* Task dispatcher – handles necessary operation to get the task ready to go
* Intertask communication

# What are the types of scheduling?

Round robin – all tasks are of equal importance. Tasks execute sequentially. However, can be preempted by an interrupt.

Time-sliced scheduling – Similar to round robin robin except that a task can only run for a predefined number of ticks.

Preemptive – Scheduler gives control to the highest priority task i.e. preempt a low priority running process.

Non-preemptive – Once the process enters the running state, it cannot be preempted until it completes its allocated time.

# What is a Process, Thread, and tasks?

A process is an instance of an application/program. Threads are subsets of a process. Process have only one state information – Running, Wait, stopped, Queued, Stopped. Threads share state as well as Memory and other resources. Context switching between threads is faster than context switching between processes. Sometimes Tasks and Processes are used interchangeably in many modern RTOSes.

Tasks are smallest unit of a process.

# What is BSP?

BSP includes Device drivers, Utility SW for application programs.

BSP is a SW code for a given board that conforms to a given OS. BSP is commonly built with a bootloader that contains minimal device support to load the OS and device driver support for devices on the board.

BSP performs:

* Initialize the processor
* Initialize the bus
* Initialize the interrupt controller
* Initialize the clock
* Initialize the RAM settings
* Run bootloader from Flash

A screenshot of a cell phone

Description automatically generated

# Explain Hardware Initialization?

Boot image executes after the CPU begins executing instructions from the reset vector. Typically, at this stage, the minimum hardware initialization required to get the boot image to execute is performed, which includes:

1. starting execution at the reset vector
2. putting the processor into a known state by setting the appropriate registers:
   * getting the processor type
   * getting or setting the CPU’s clock speed
3. disabling interrupts and caches
4. initializing memory controller, memory chips, and cache units:
   * getting the start addresses for memory
   * getting the size of memory
   * performing preliminary memory tests, if require

A screenshot of a cell phone

Description automatically generated

# Explain RTOS initialization sequences?

* Initializing different RTOS objects and services, if present (usually controlled with a user-configurable header file):
  + Task objects
  + Semaphore objects
  + Message-queue objects
  + Timer services
  + Interrupt services
  + Memory-management services
* Creating necessary stacks for RTOS
* Initializing additional RTOS extensions, such as:
* TCP/IP stack
* file systems
* Starting the RTOS and its initial tasks