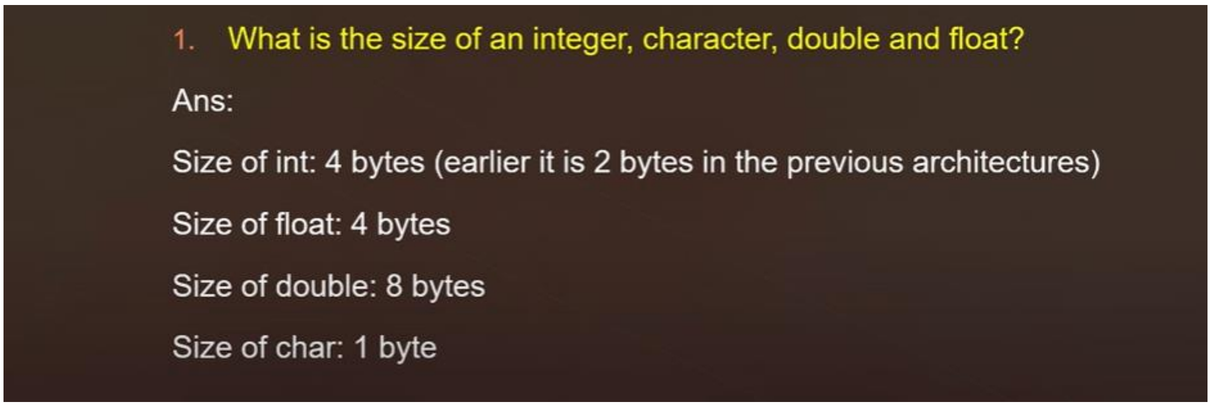
C programming for Embedded Systems



1. What is the size of an integer, character, double and float?

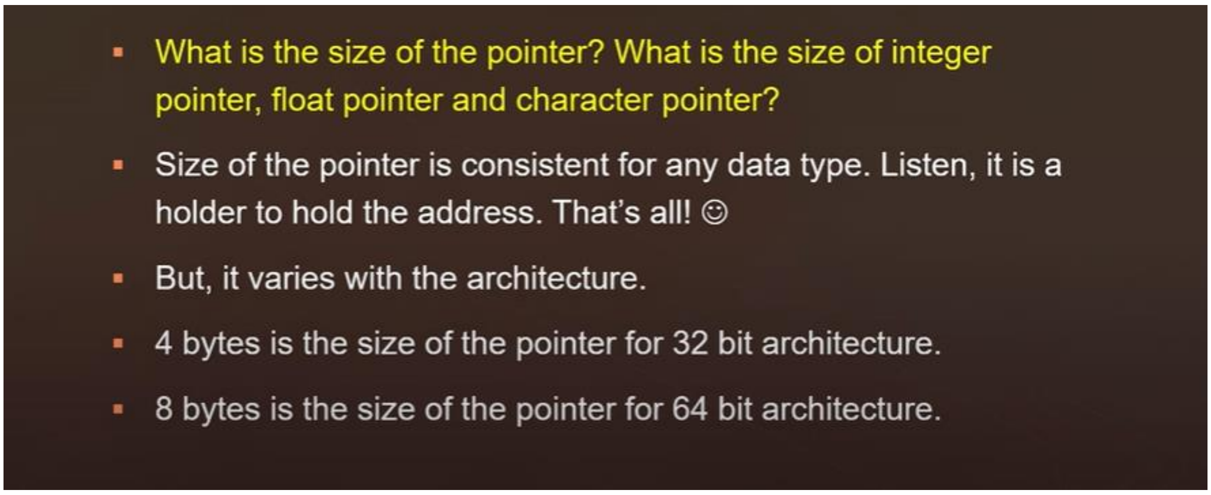
Ans:

Size of int: 4 bytes (earlier it is 2 bytes in the previous architectures)

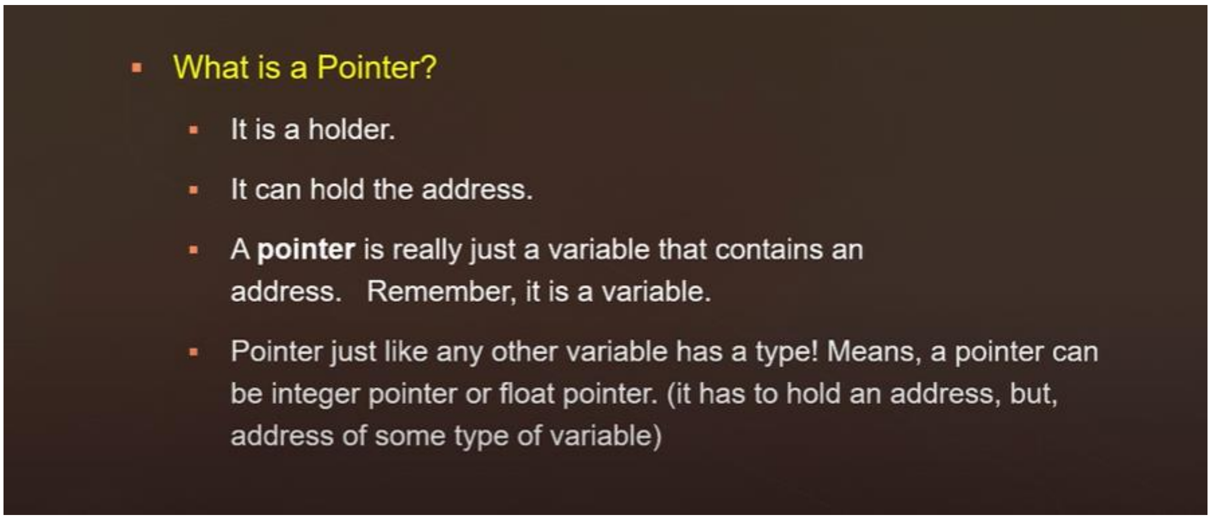
Size of float: 4 bytes

Size of double: 8 bytes

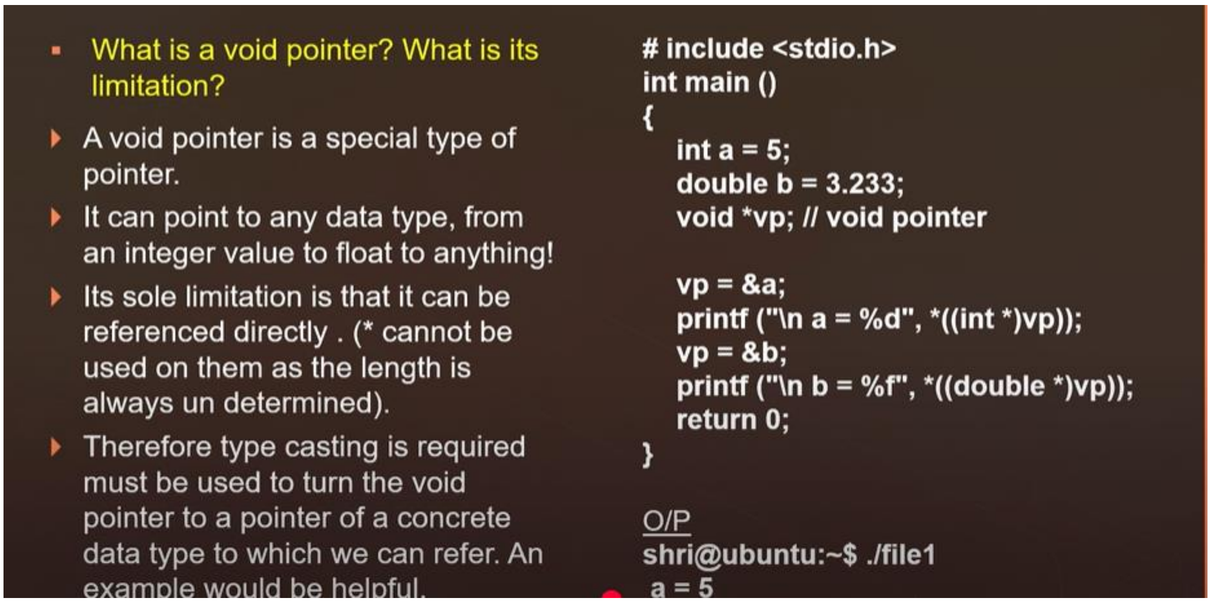
Size of char: 1 byte



- What is the size of the pointer? What is the size of integer pointer, float pointer and character pointer?
- Size of the pointer is consistent for any data type. Listen, it is a holder to hold the address. That’s all! ☺
- But, it varies with the architecture.
- 4 bytes is the size of the pointer for 32 bit architecture.
- 8 bytes is the size of the pointer for 64 bit architecture.



- What is a Pointer?
  - It is a holder.
  - It can hold the address.
  - A **pointer** is really just a variable that contains an address. Remember, it is a variable.
  - Pointer just like any other variable has a type! Means, a pointer can be integer pointer or float pointer. (it has to hold an address, but, address of some type of variable)

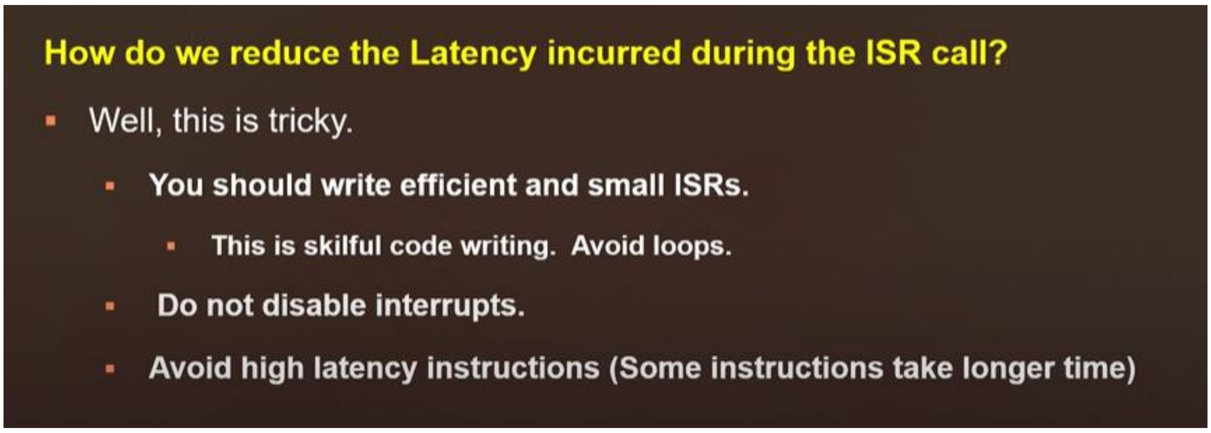


- What is a void pointer? What is its limitation?
  - A void pointer is a special type of pointer.
  - It can point to any data type, from an integer value to float to anything!
  - Its sole limitation is that it can be referenced directly . (* cannot be used on them as the length is always un determined).
  - Therefore type casting is required must be used to turn the void pointer to a pointer of a concrete data type to which we can refer. An example would be helpful.

```
# include <stdio.h>
int main ()
{
    int a = 5;
    double b = 3.233;
    void *vp; // void pointer

    vp = &a;
    printf ("\n a = %d", *((int *)vp));
    vp = &b;
    printf ("\n b = %f", *((double *)vp));
    return 0;
}
```

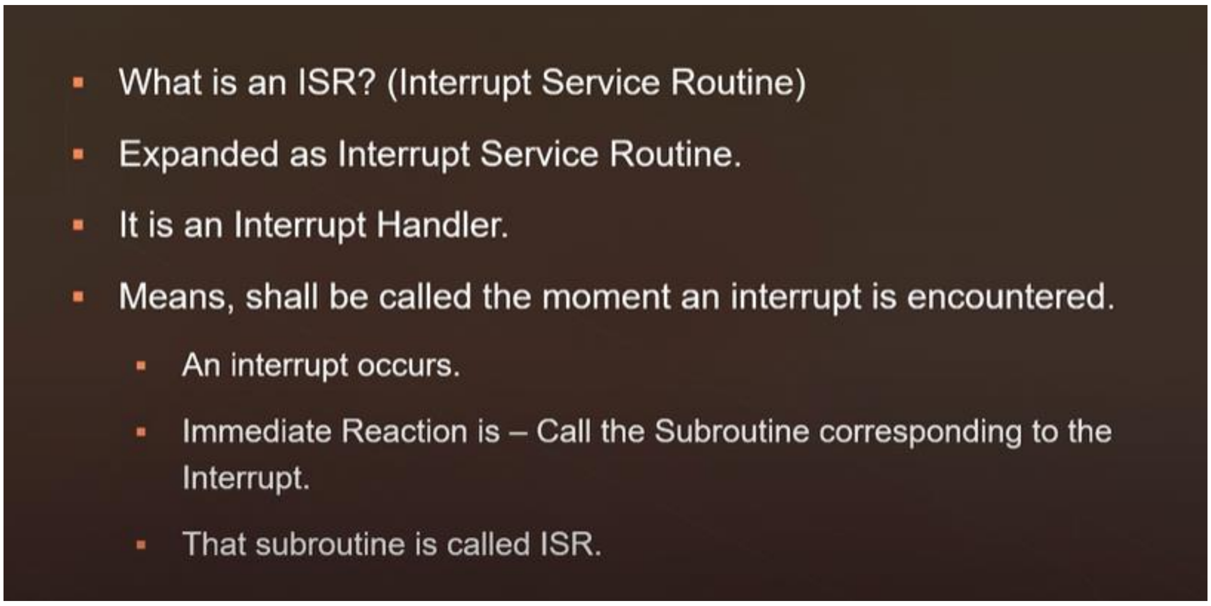
O/P

```
shri@ubuntu:~$ ./file1
 a = 5
```

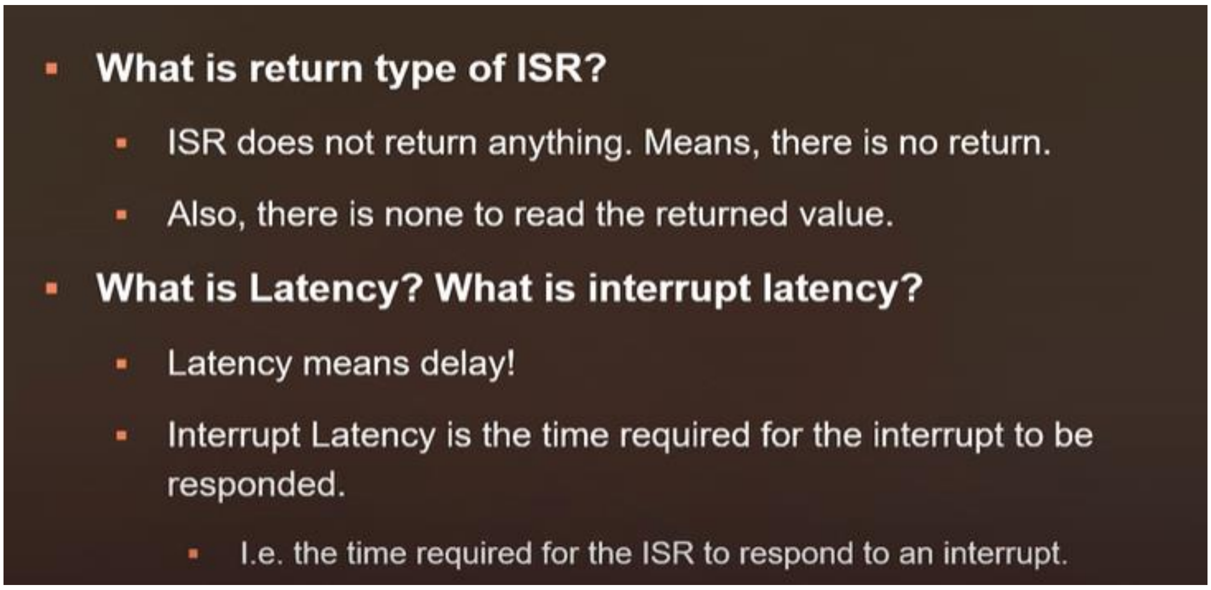


How do we reduce the Latency incurred during the ISR call?

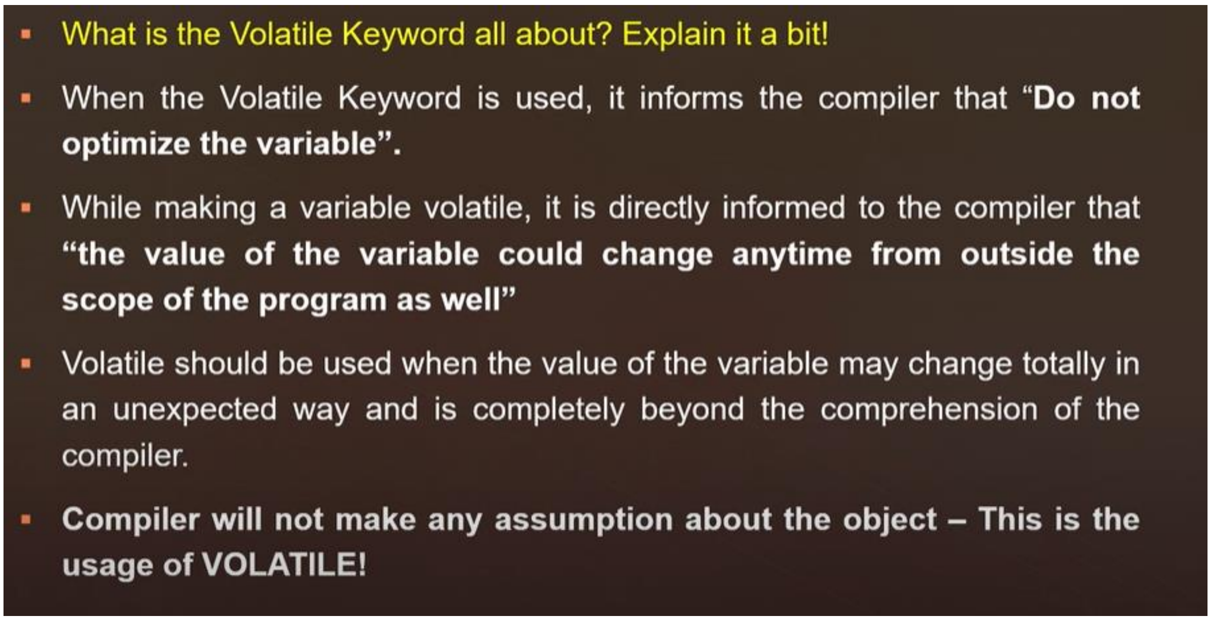
- Well, this is tricky.
  - **You should write efficient and small ISRs.**
    - **This is skilful code writing. Avoid loops.**
  - **Do not disable interrupts.**
  - **Avoid high latency instructions (Some instructions take longer time)**



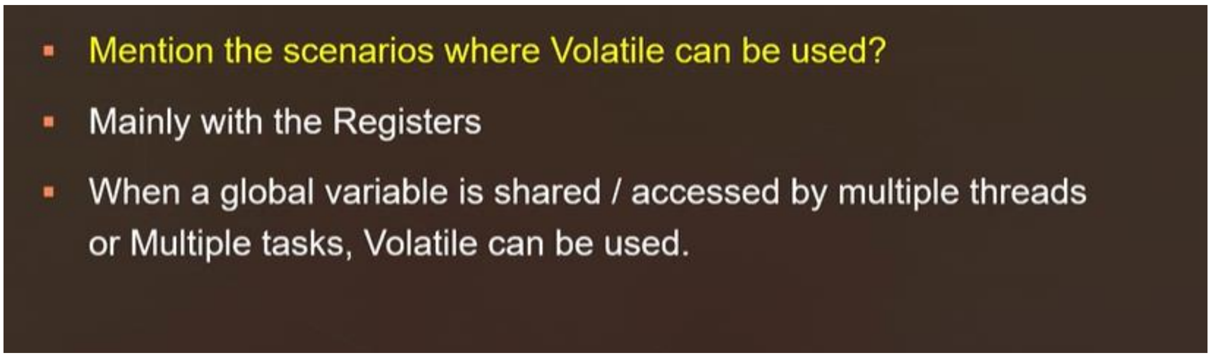
- What is an ISR? (Interrupt Service Routine)
- Expanded as Interrupt Service Routine.
- It is an Interrupt Handler.
- Means, shall be called the moment an interrupt is encountered.
  - An interrupt occurs.
  - Immediate Reaction is – Call the Subroutine corresponding to the Interrupt.
  - That subroutine is called ISR.



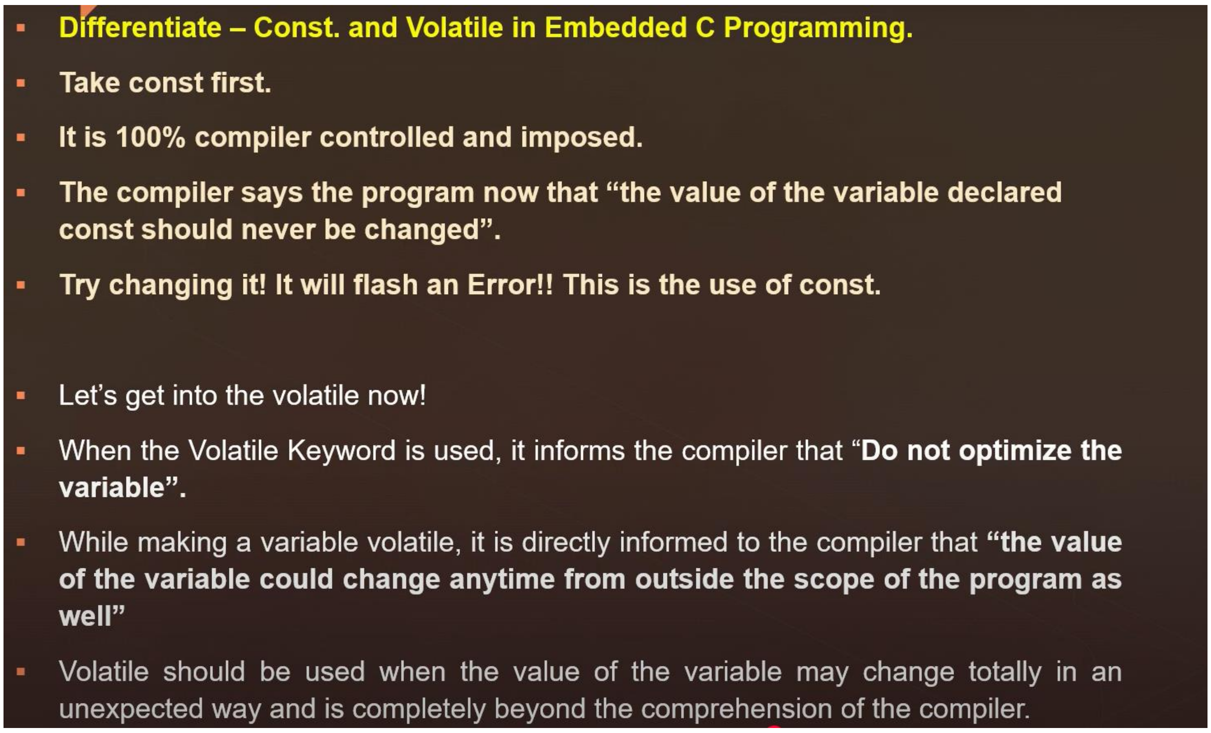
- **What is return type of ISR?**
  - ISR does not return anything. Means, there is no return.
  - Also, there is none to read the returned value.
- **What is Latency? What is interrupt latency?**
  - Latency means delay!
  - Interrupt Latency is the time required for the interrupt to be responded.
    - I.e. the time required for the ISR to respond to an interrupt.



- What is the Volatile Keyword all about? Explain it a bit!
- When the Volatile Keyword is used, it informs the compiler that "**Do not optimize the variable".**
- While making a variable volatile, it is directly informed to the compiler that **"the value of the variable could change anytime from outside the scope of the program as well"**
- Volatile should be used when the value of the variable may change totally in an unexpected way and is completely beyond the comprehension of the compiler.
- **Compiler will not make any assumption about the object – This is the usage of VOLATILE!**

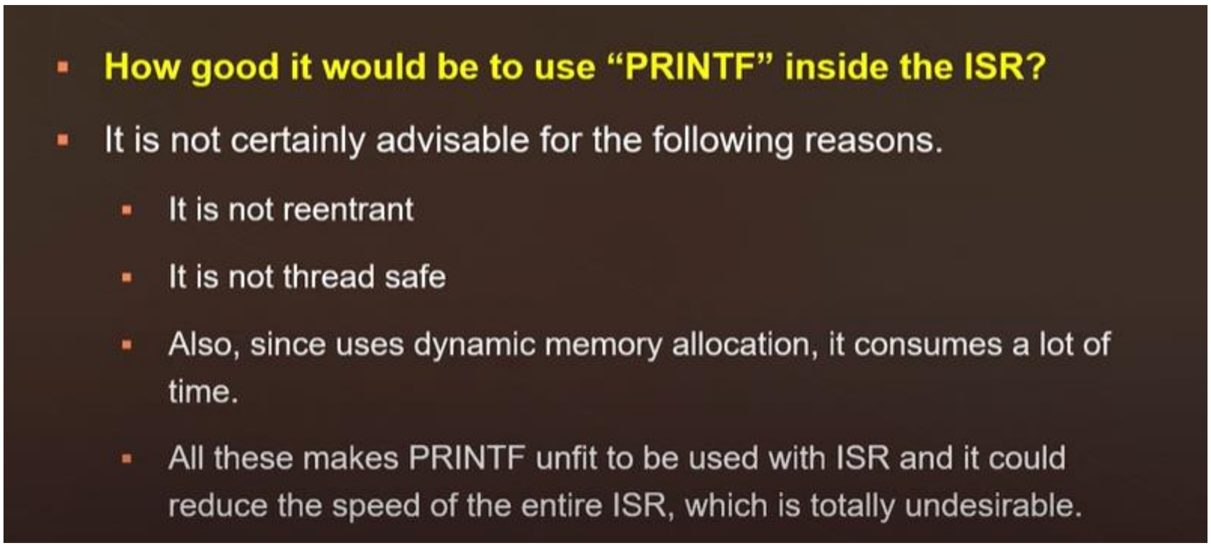


- Mention the scenarios where Volatile can be used?
- Mainly with the Registers
- When a global variable is shared / accessed by multiple threads or Multiple tasks, Volatile can be used.

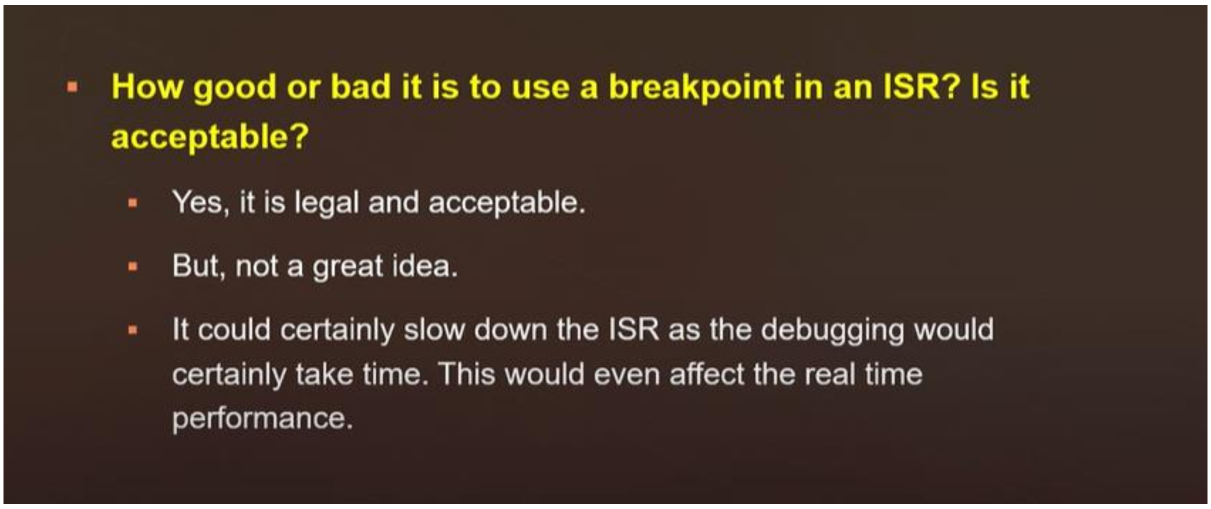


- **Differentiate – Const. and Volatile in Embedded C Programming.**
- **Take const first.**
- **It is 100% compiler controlled and imposed.**
- **The compiler says the program now that "the value of the variable declared const should never be changed".**
- **Try changing it! It will flash an Error!! This is the use of const.**

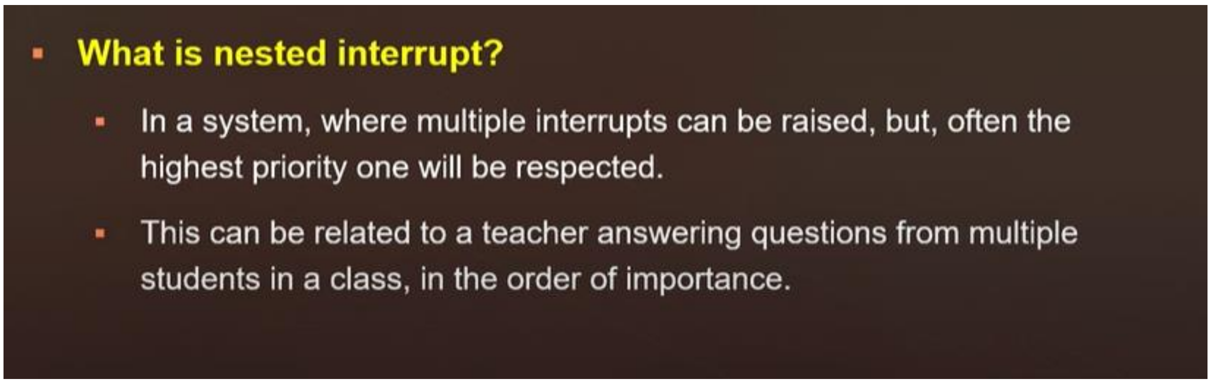
- Let's get into the volatile now!
- When the Volatile Keyword is used, it informs the compiler that "**Do not optimize the variable".**
- While making a variable volatile, it is directly informed to the compiler that **"the value of the variable could change anytime from outside the scope of the program as well"**
- Volatile should be used when the value of the variable may change totally in an unexpected way and is completely beyond the comprehension of the compiler.



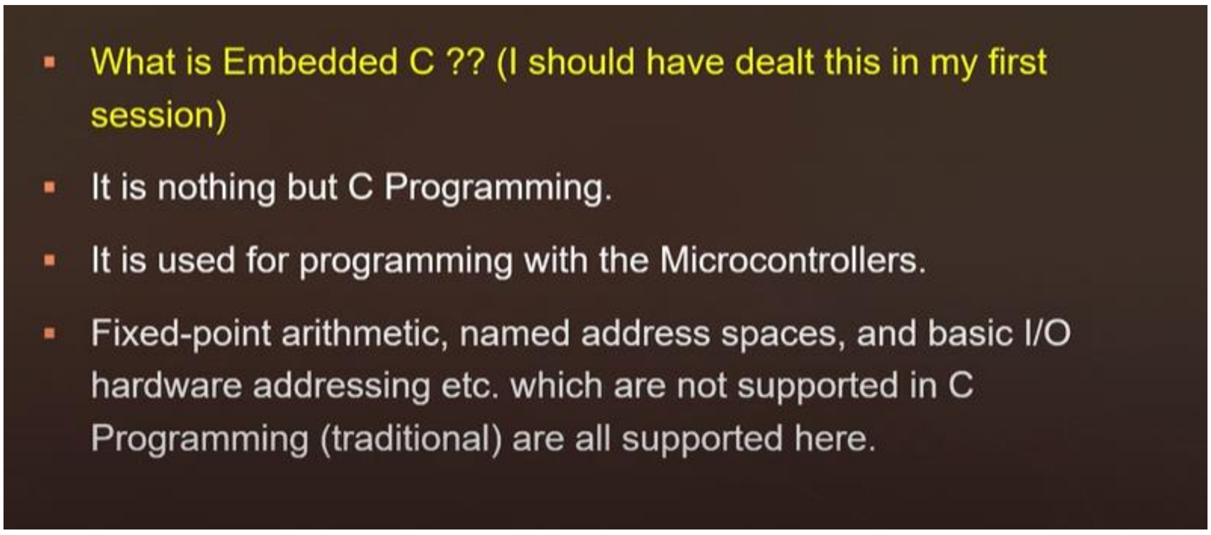
- **How good it would be to use "PRINTF" inside the ISR?**
- It is not certainly advisable for the following reasons.
  - It is not reentrant
  - It is not thread safe
  - Also, since uses dynamic memory allocation, it consumes a lot of time.
  - All these makes PRINTF unfit to be used with ISR and it could reduce the speed of the entire ISR, which is totally undesirable.



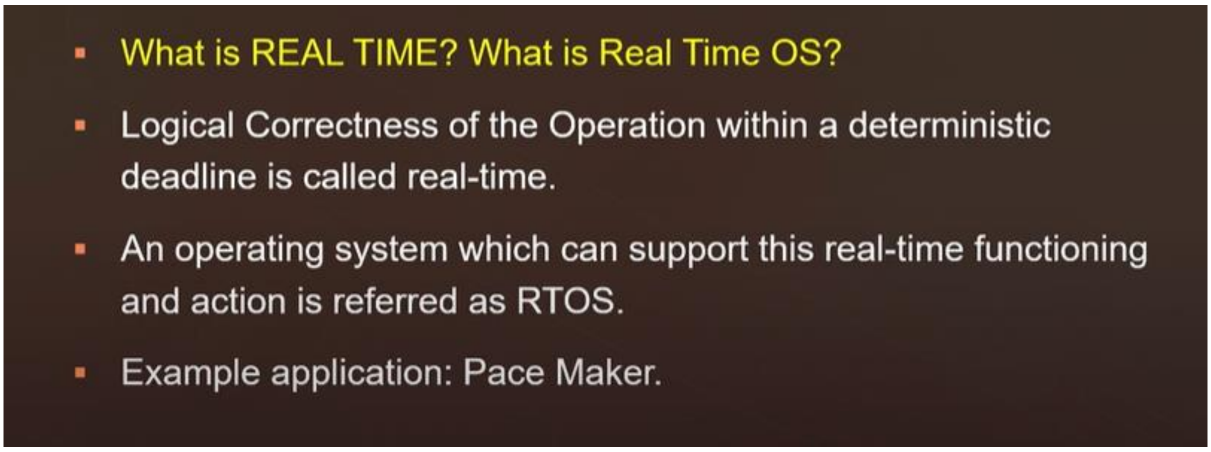
- **How good or bad it is to use a breakpoint in an ISR? Is it acceptable?**
  - Yes, it is legal and acceptable.
  - But, not a great idea.
  - It could certainly slow down the ISR as the debugging would certainly take time. This would even affect the real time performance.



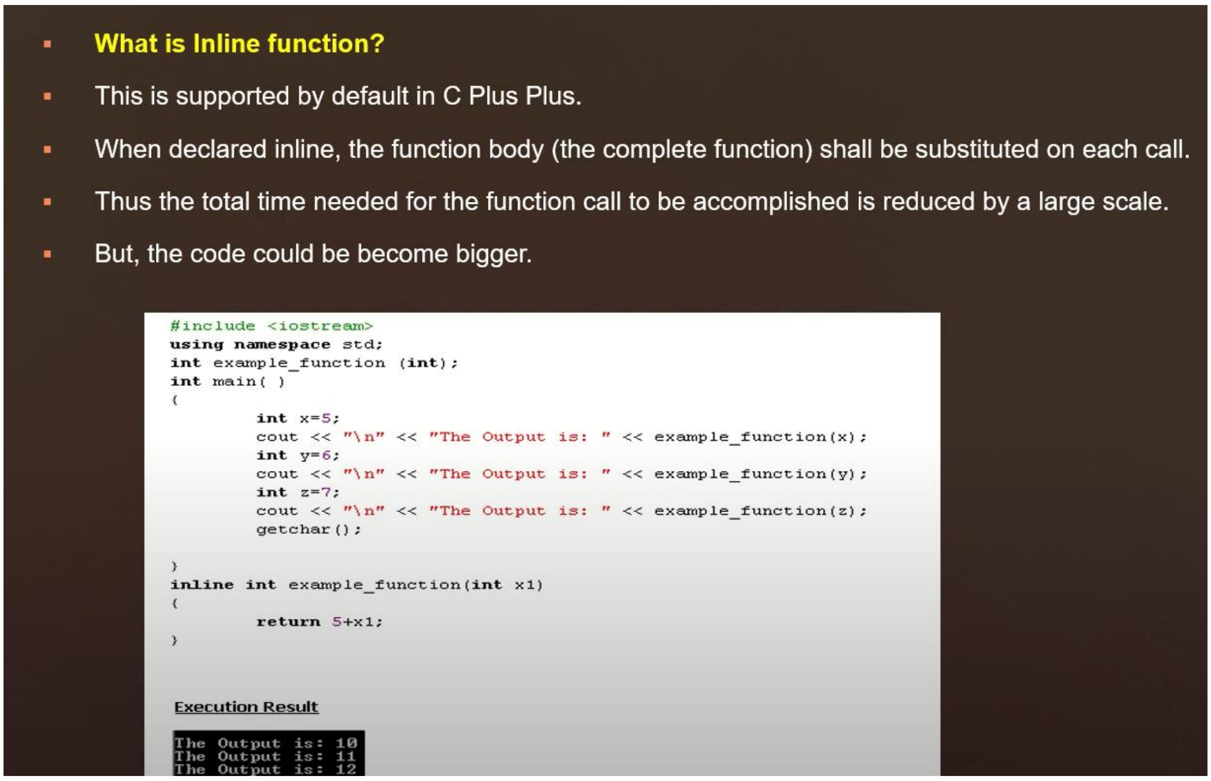
- **What is nested interrupt?**
  - In a system, where multiple interrupts can be raised, but, often the highest priority one will be respected.
  - This can be related to a teacher answering questions from multiple students in a class, in the order of importance.



- What is Embedded C ?? (I should have dealt this in my first session)
- It is nothing but C Programming.
- It is used for programming with the Microcontrollers.
- Fixed-point arithmetic, named address spaces, and basic I/O hardware addressing etc. which are not supported in C Programming (traditional) are all supported here.



- **What is REAL TIME? What is Real Time OS?**
- Logical Correctness of the Operation within a deterministic deadline is called real-time.
- An operating system which can support this real-time functioning and action is referred as RTOS.
- Example application: Pace Maker.

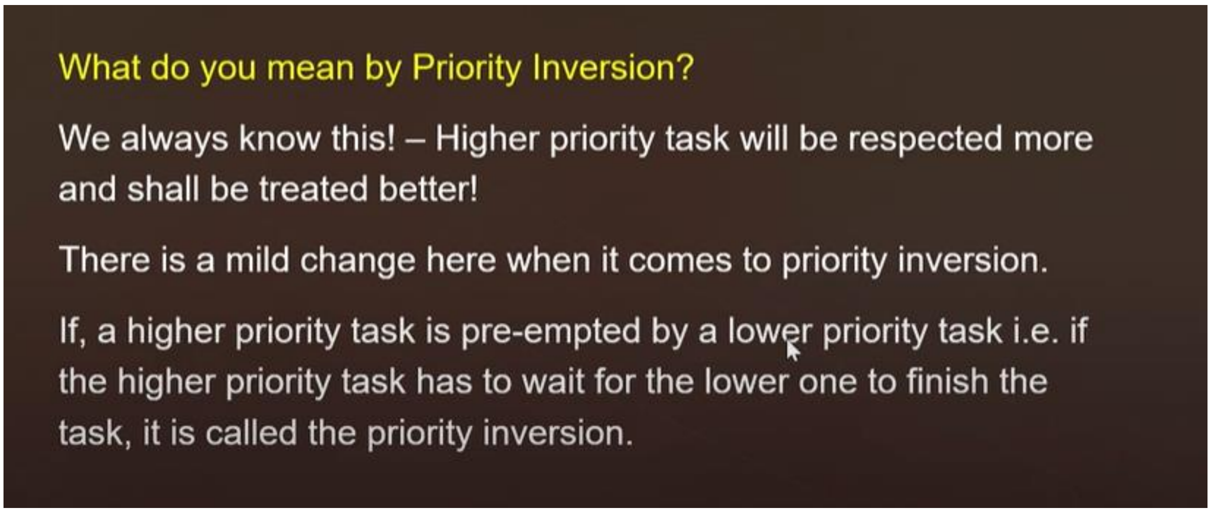


- **What is Inline function?**
- This is supported by default in C Plus Plus.
- When declared inline, the function body (the complete function) shall be substituted on each call.
- Thus the total time needed for the function call to be accomplished is reduced by a large scale.
- But, the code could be become bigger.

```
#include <iostream>
using namespace std;
int example_function (int);
int main( )
{
        int x=5;
        cout << "\n" << "The Output is: " << example_function(x);
        int y=6;
        cout << "\n" << "The Output is: " << example_function(y);
        int z=7;
        cout << "\n" << "The Output is: " << example_function(z);
        getchar();

}
inline int example_function(int x1)
{
        return 5+x1;
}
```

Execution Result

```
The Output is: 10
The Output is: 11
The Output is: 12
```

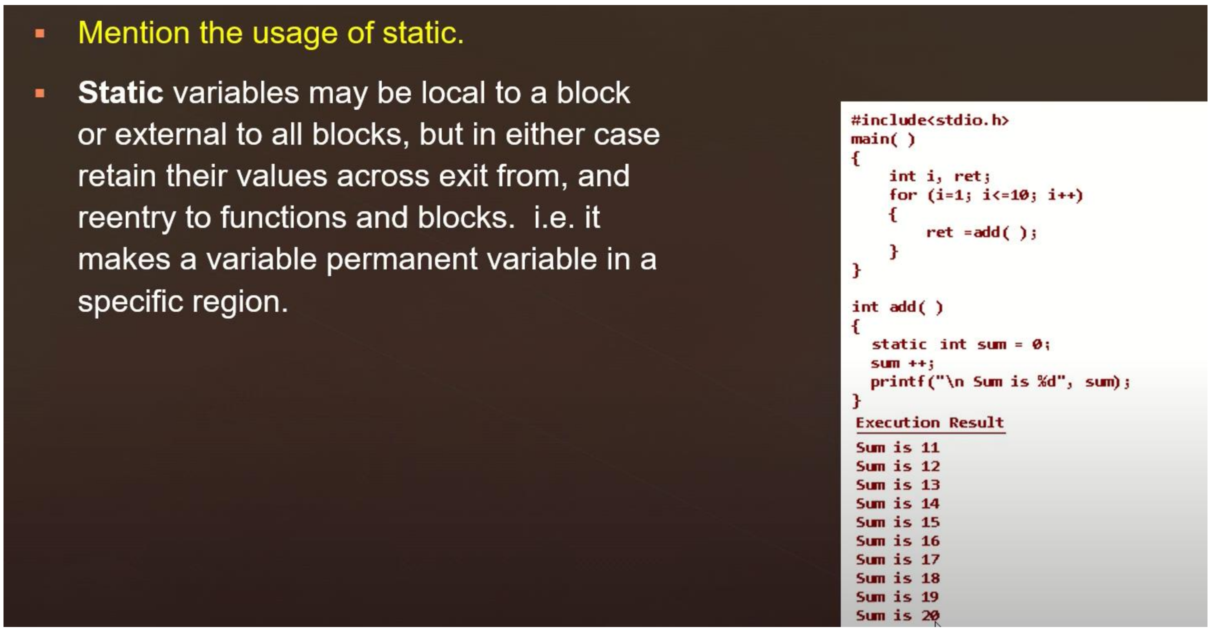


What do you mean by Priority Inversion?

We always know this! – Higher priority task will be respected more and shall be treated better!

There is a mild change here when it comes to priority inversion.

If, a higher priority task is pre-empted by a lower priority task i.e. if the higher priority task has to wait for the lower one to finish the task, it is called the priority inversion.

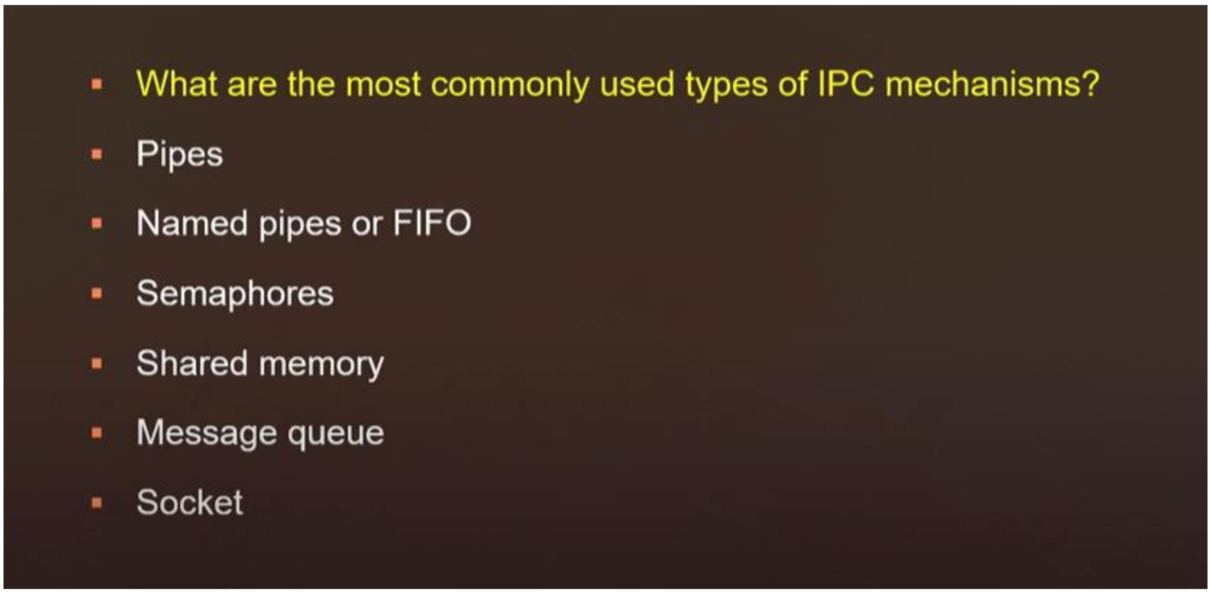


- Mention the usage of static.
- **Static** variables may be local to a block or external to all blocks, but in either case retain their values across exit from, and reentry to functions and blocks. i.e. it makes a variable permanent variable in a specific region.

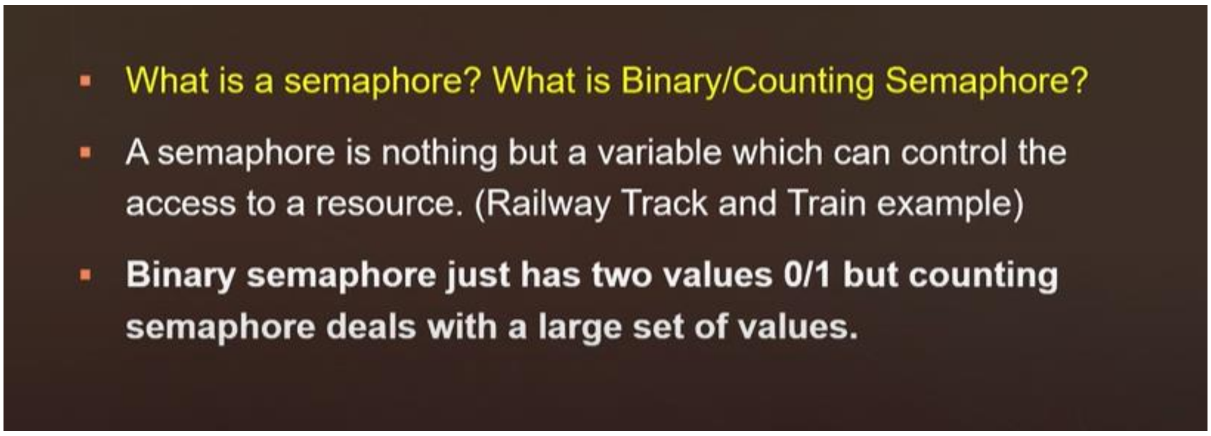
```
#include<stdio.h>
main( )
{
    int i, ret;
    for (i=1; i<=10; i++)
    {
        ret =add( );
    }
}

int add( )
{
  static int sum = 0;
  sum ++;
  printf("\n Sum is %d", sum);
}
```

Execution Result

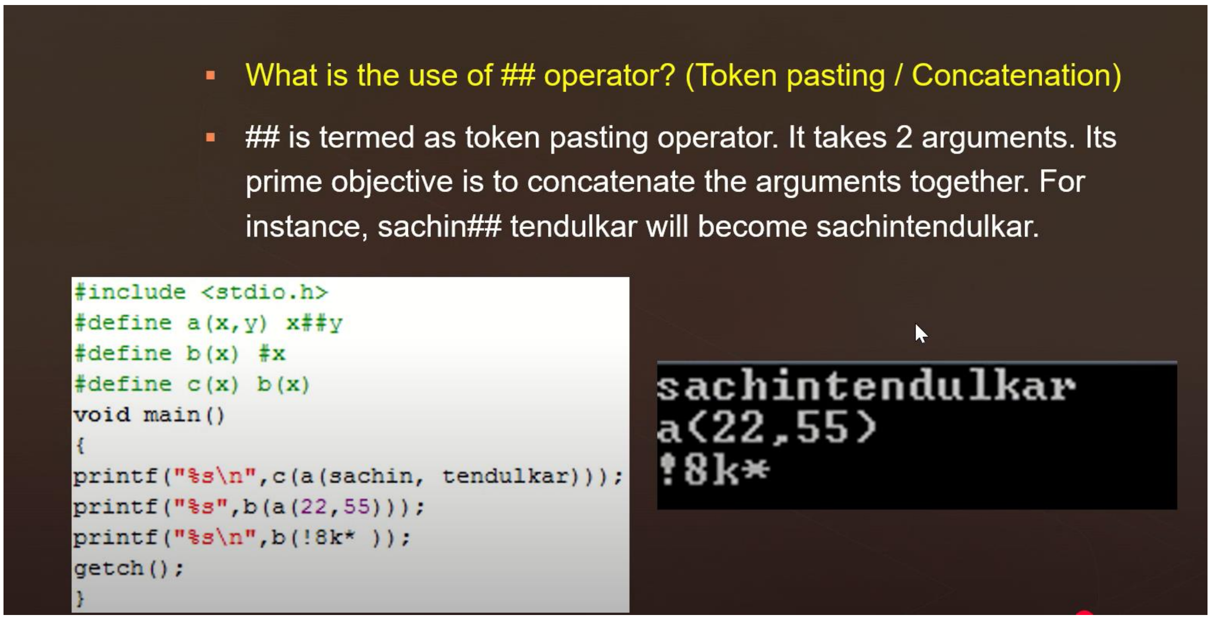
```
Sum is 11
Sum is 12
Sum is 13
Sum is 14
Sum is 15
Sum is 16
Sum is 17
Sum is 18
Sum is 19
Sum is 20
```



- What are the most commonly used types of IPC mechanisms?
- Pipes
- Named pipes or FIFO
- Semaphores
- Shared memory
- Message queue
- Socket



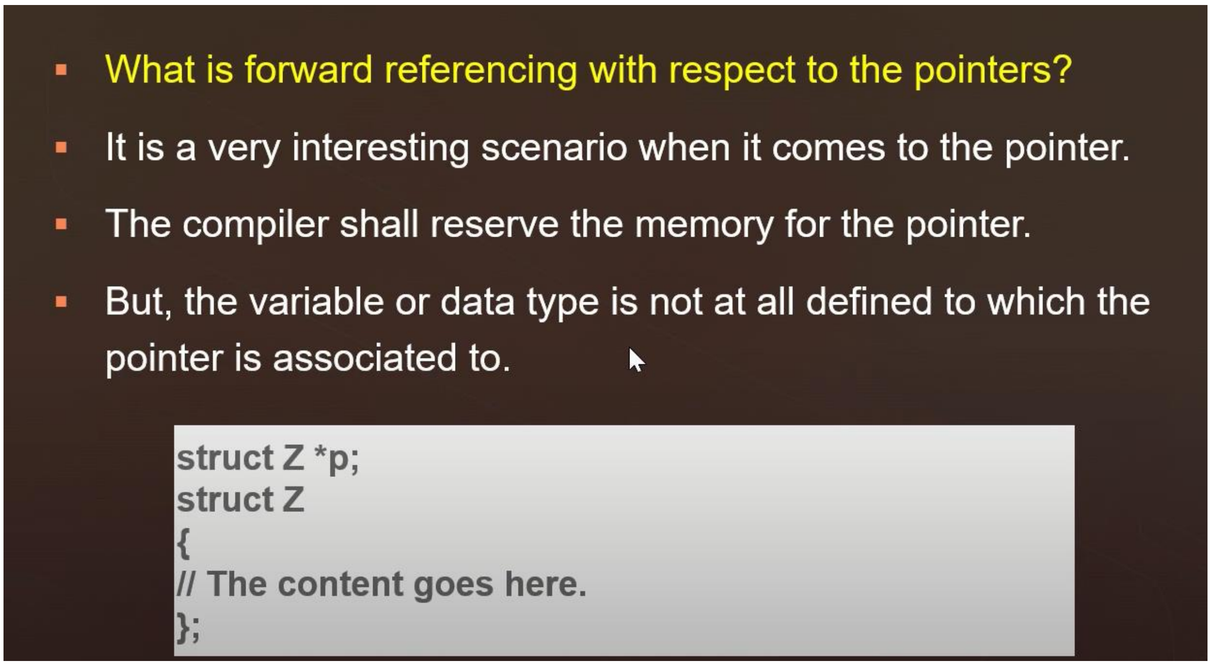
- What is a semaphore? What is Binary/Counting Semaphore?
- A semaphore is nothing but a variable which can control the access to a resource. (Railway Track and Train example)
- **Binary semaphore just has two values 0/1 but counting semaphore deals with a large set of values.**



- What is the use of ## operator? (Token pasting / Concatenation)
- ## is termed as token pasting operator. It takes 2 arguments. Its prime objective is to concatenate the arguments together. For instance, sachin## tendulkar will become sachintendulkar.

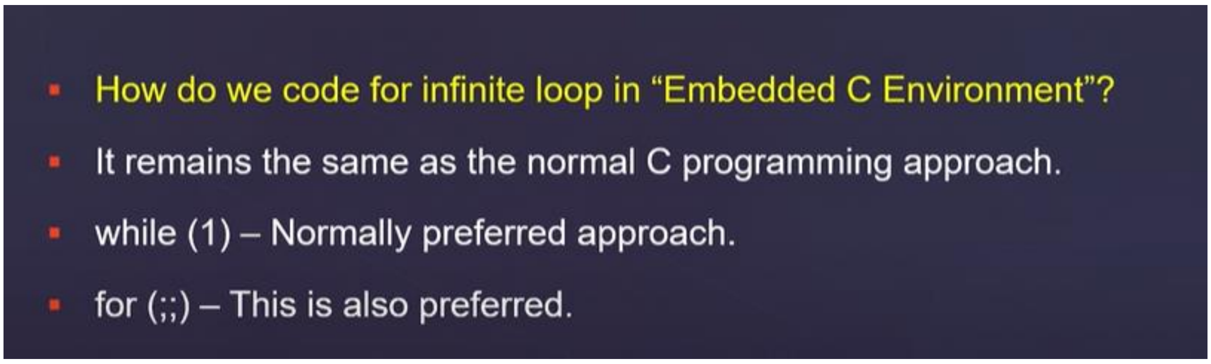
```
#include <stdio.h>
#define a(x,y) x##y
#define b(x) #x
#define c(x) b(x)
void main()
{
printf("%s\n",c(a(sachin, tendulkar)));
printf("%s",b(a(22,55)));
printf("%s\n",b(!8k* ));
getch();
}
```

```
sachintendulkar
a(22,55)
!8k*
```

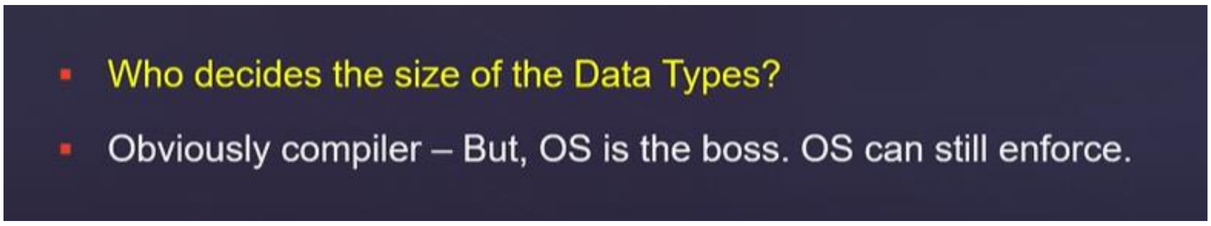


- What is forward referencing with respect to the pointers?
- It is a very interesting scenario when it comes to the pointer.
- The compiler shall reserve the memory for the pointer.
- But, the variable or data type is not at all defined to which the pointer is associated to.

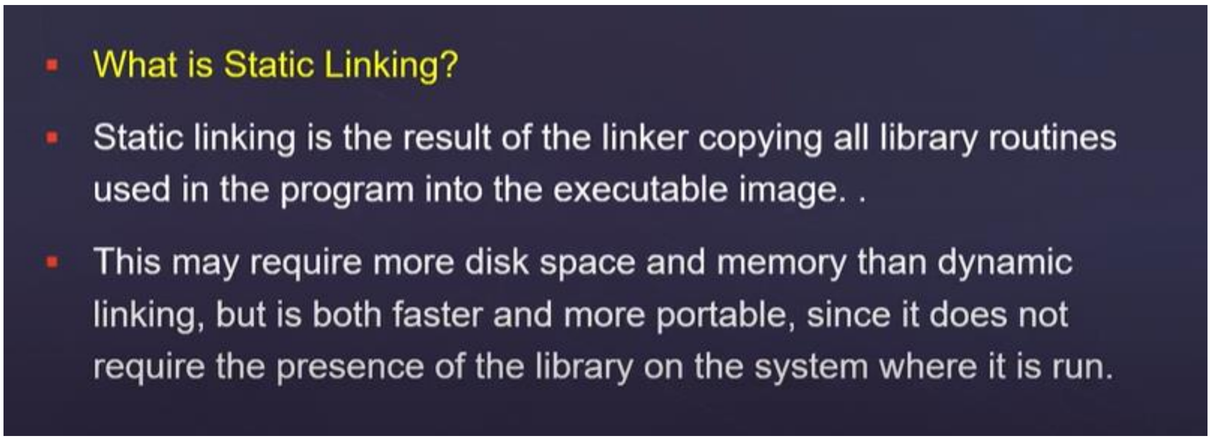
```
struct Z *p;
struct Z
{
// The content goes here.
};
```



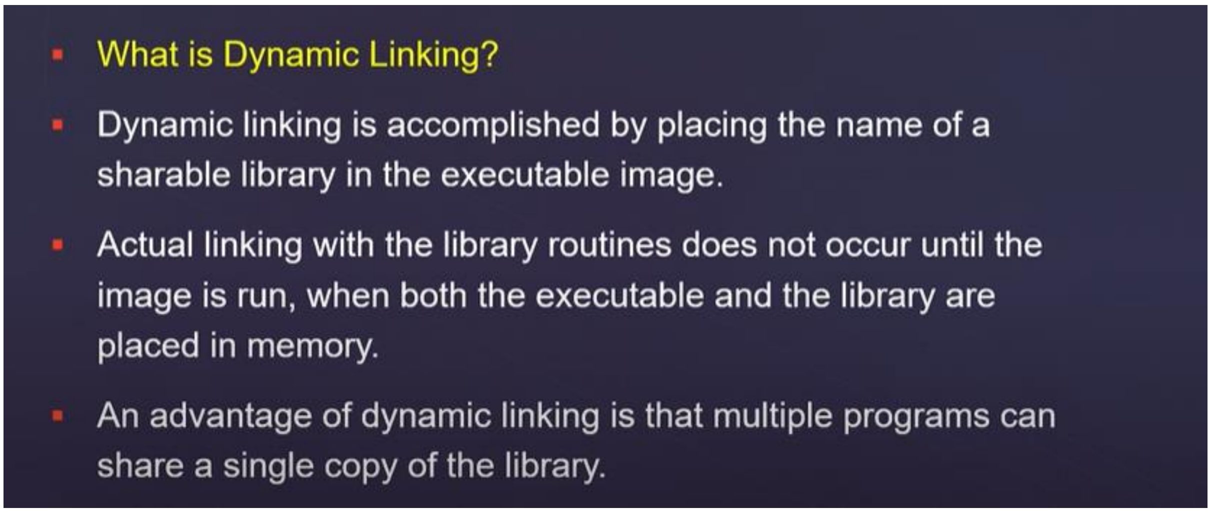
- How do we code for infinite loop in “Embedded C Environment”?
- It remains the same as the normal C programming approach.
- while (1) – Normally preferred approach.
- for (;;) – This is also preferred.



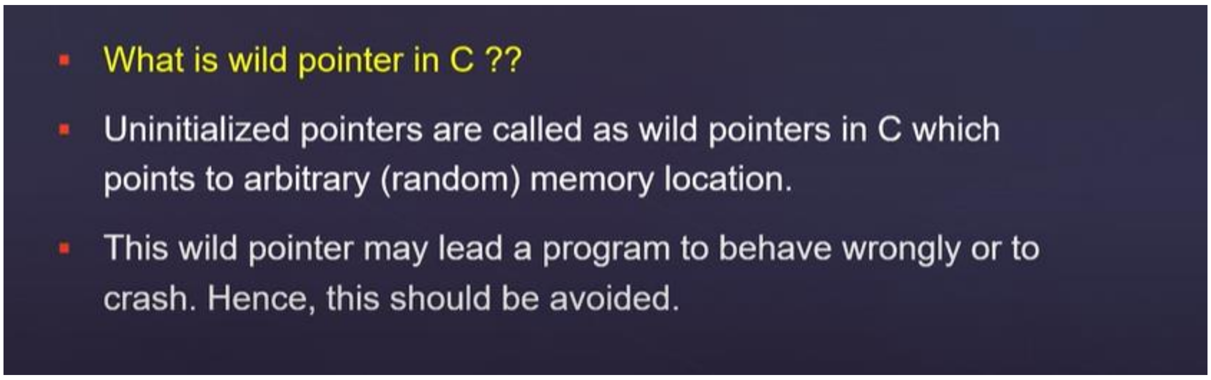
- Who decides the size of the Data Types?
- Obviously compiler – But, OS is the boss. OS can still enforce.



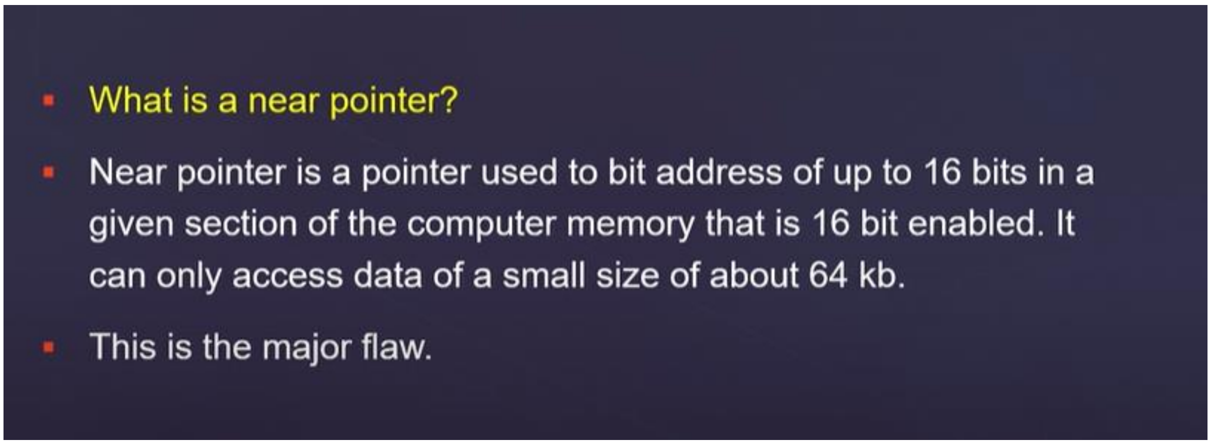
- What is Static Linking?
- Static linking is the result of the linker copying all library routines used in the program into the executable image. .
- This may require more disk space and memory than dynamic linking, but is both faster and more portable, since it does not require the presence of the library on the system where it is run.



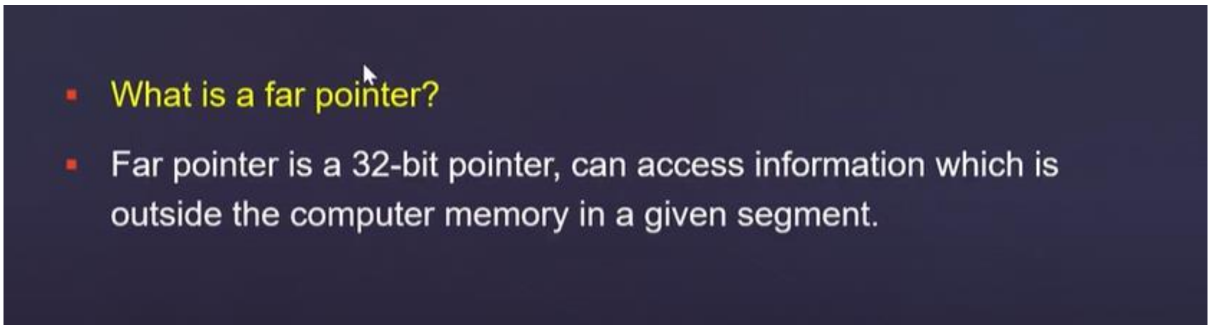
- What is Dynamic Linking?
- Dynamic linking is accomplished by placing the name of a sharable library in the executable image.
- Actual linking with the library routines does not occur until the image is run, when both the executable and the library are placed in memory.
- An advantage of dynamic linking is that multiple programs can share a single copy of the library.



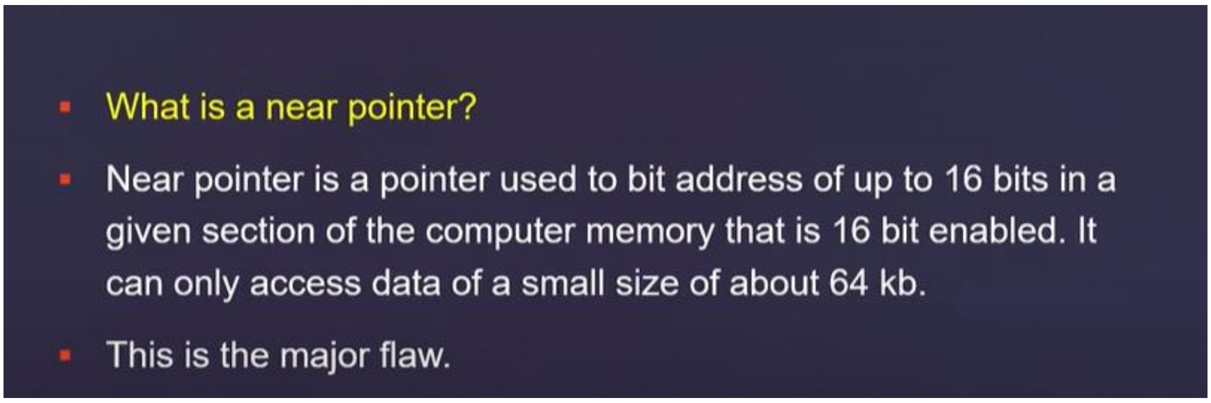
- What is wild pointer in C ??
- Uninitialized pointers are called as wild pointers in C which points to arbitrary (random) memory location.
- This wild pointer may lead a program to behave wrongly or to crash. Hence, this should be avoided.



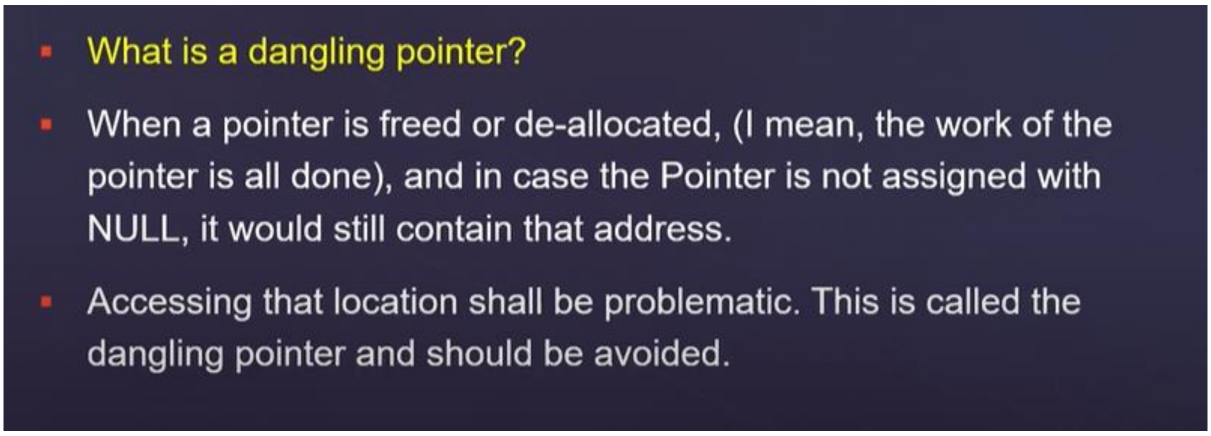
- What is a near pointer?
- Near pointer is a pointer used to bit address of up to 16 bits in a given section of the computer memory that is 16 bit enabled. It can only access data of a small size of about 64 kb.
- This is the major flaw.



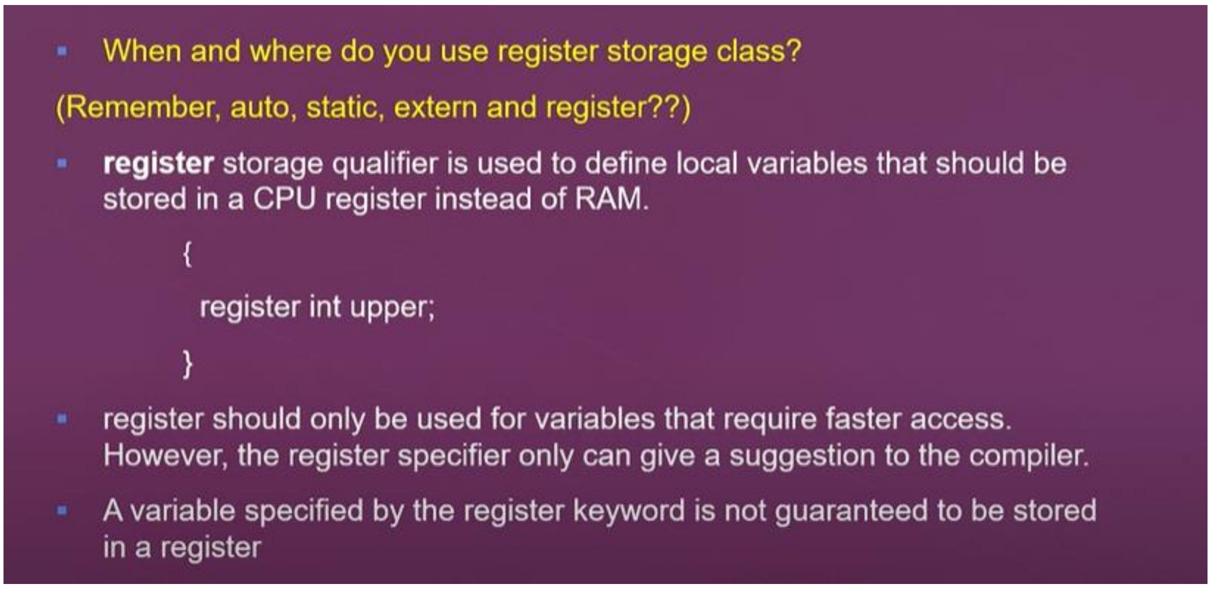
- What is a far pointer?
- Far pointer is a 32-bit pointer, can access information which is outside the computer memory in a given segment.



- What is a near pointer?
- Near pointer is a pointer used to bit address of up to 16 bits in a given section of the computer memory that is 16 bit enabled. It can only access data of a small size of about 64 kb.
- This is the major flaw.



- What is a dangling pointer?
- When a pointer is freed or de-allocated, (I mean, the work of the pointer is all done), and in case the Pointer is not assigned with NULL, it would still contain that address.
- Accessing that location shall be problematic. This is called the dangling pointer and should be avoided.



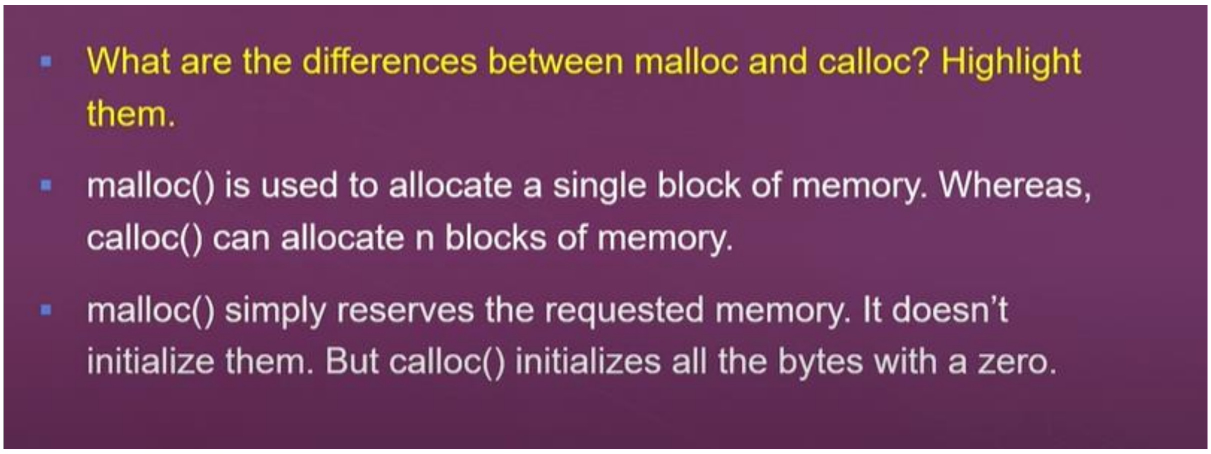
- When and where do you use register storage class?

(Remember, auto, static, extern and register??)

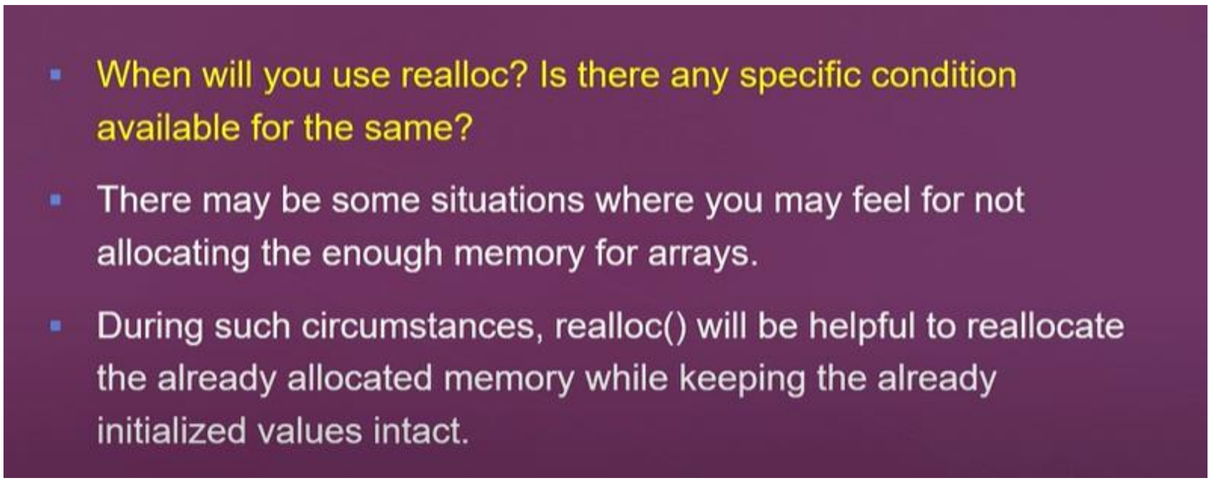
- **register** storage qualifier is used to define local variables that should be stored in a CPU register instead of RAM.

```
{
  register int upper;
}
```

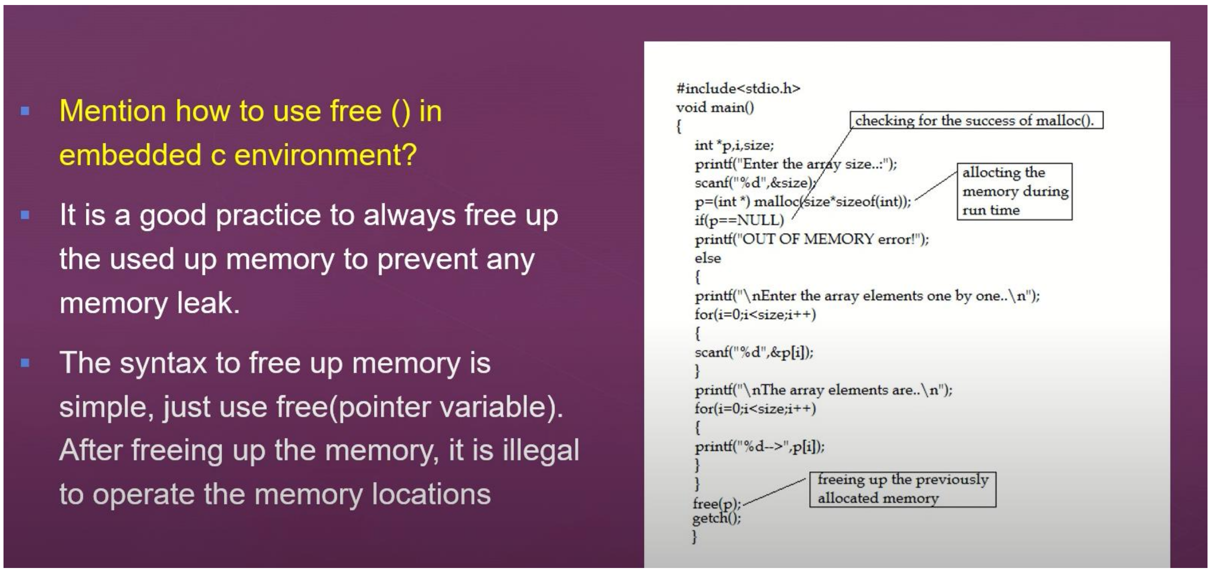
- register should only be used for variables that require faster access. However, the register specifier only can give a suggestion to the compiler.
- A variable specified by the register keyword is not guaranteed to be stored in a register



- What are the differences between malloc and calloc? Highlight them.
- malloc() is used to allocate a single block of memory. Whereas, calloc() can allocate n blocks of memory.
- malloc() simply reserves the requested memory. It doesn’t initialize them. But calloc() initializes all the bytes with a zero.



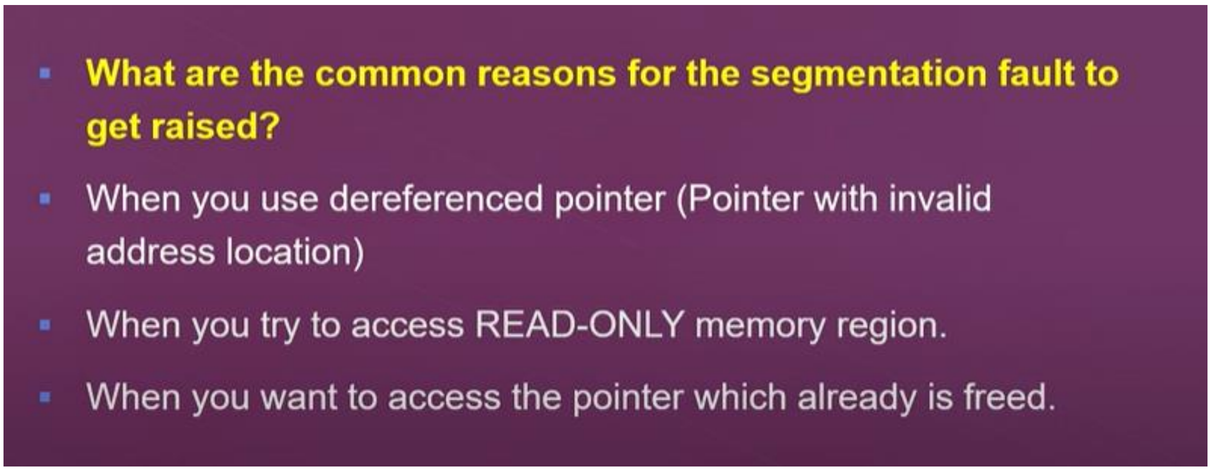
- When will you use realloc? Is there any specific condition available for the same?
- There may be some situations where you may feel for not allocating the enough memory for arrays.
- During such circumstances, realloc() will be helpful to reallocate the already allocated memory while keeping the already initialized values intact.



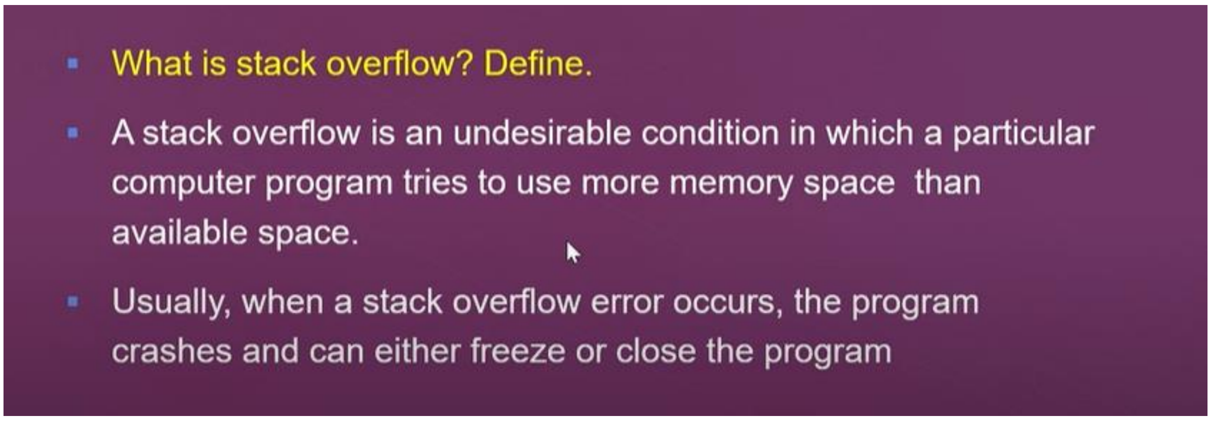
- Mention how to use free () in embedded c environment?
- It is a good practice to always free up the used up memory to prevent any memory leak.
- The syntax to free up memory is simple, just use free(pointer variable). After freeing up the memory, it is illegal to operate the memory locations

```
#include<stdio.h>
void main()
{
   int *p,i,size;
   printf("Enter the array size..:");
   scanf("%d",&size);
   p=(int *) malloc(size*sizeof(int));
   if(p==NULL)
   printf("OUT OF MEMORY error!");
   else
   {
   printf("\nEnter the array elements one by one..\n");
   for(i=0;i<size;i++)
   {
   scanf("%d",&p[i]);
   }
   printf("\nThe array elements are..\n");
   for(i=0;i<size;i++)
   {
   printf("%d-->",p[i]);
   }
   }
   free(p);
   getch();
   }
```

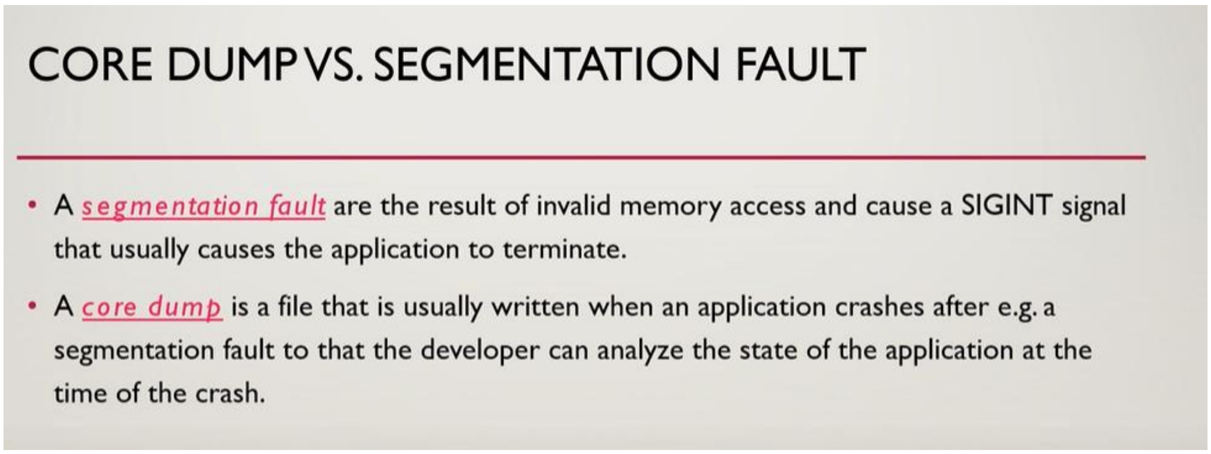
checking for the success of malloc().
allocting the memory during run time
freeing up the previously allocated memory



- **What are the common reasons for the segmentation fault to get raised?**
- When you use dereferenced pointer (Pointer with invalid address location)
- When you try to access READ-ONLY memory region.
- When you want to access the pointer which already is freed.



- What is stack overflow? Define.
- A stack overflow is an undesirable condition in which a particular computer program tries to use more memory space than available space.
- Usually, when a stack overflow error occurs, the program crashes and can either freeze or close the program



CORE DUMP VS. SEGMENTATION FAULT

- A *segmentation fault* are the result of invalid memory access and cause a SIGINT signal that usually causes the application to terminate.
- A *core dump* is a file that is usually written when an application crashes after e.g. a segmentation fault to that the developer can analyze the state of the application at the time of the crash.