

# **EMBEDDED C**

## **Interview Questions**

1) Does Preprocessing happens in case of Assembly Language Code?  
If "Yes" then exactly How??

**Ans:** it really depends on the processor directives. If u write a assembly code and save it with .s (small s) the compiler wont perform the preprocessing but if u store it as .S(capital S) then it tells compiler to perform the preprocessing on assembly code as well. I dont understand much about it but armclang is mostly used as comandline tool to compile assembly codes for ARM processors/microcontrollers.

## 2) Macros for Data Manipulations in 'C'

**Ans:**

>>> Macro to Swap Nibbles in a Byte

```
#define SwapNibbles(data) (((data & 0x0F)<<4) | ((data & 0xF0)>>4))
```

>>> Macro to Swap Two Bytes

```
#define SwapTwoBytes(data) (((data & 0x00FF)<<8) | ((data & 0xFF00)>>8))
```

>>> Macro to Swap two numbers

```
#define SWAP(x,y) (x^=y^=x^=y)
```

>>> Macro to Convert one Endian into another Endian

```
#define SwapEndians(value) ((value & 0x000000FF) << 24) | ((value & 0x0000FF00) << 8) | \
((value & 0x00FF0000) >> 8) | ((value & 0xFF000000) >> 24)
```

**#S**

### >>> Macro to Swap Eight Bytes of Data

**#define** SwapEightBytes(data)

```
((data) >> 56) & 0x00000000000000FF | (((data) >> 40) & 0x000000000000FF00) |  
(((data) >> 24) & 0x0000000000FF0000) | (((data) >> 8) & 0x00000000FF000000) |  
(((data) << 8) & 0x000000FF00000000) | (((data) << 24) & 0x0000FF0000000000) |  
(((data) << 40) & 0x00FF000000000000) | (((data) << 56) & 0xFF00000000000000))
```

### >>> Macros to Get HIGH and LOW Bytes from a word

**#define** LOW\_BYTE(x) ((unsigned char)(x)&0xFF)

**#define** HIGH\_BYTE(x) ((unsigned char)(x>>8)&0xFF)

//\*\*\*\*\*//

### 3) StartUp Code

**Ans:**

It is a Small Block of Assembly Language code that prepares the way for execution of Software written in High Level language

It is used to Setup data memory such as Global data variables. It also zero initializes part of Data memory for variables that are uninitialized at load time.

#### Following actions are performed by StartUp Code

- 1) Disable All Interrupts
- 2) Copy any initialized data from ROM to RAM
- 3) Zero the uninitialized data area
- 4) Enable the Interrupts
- 5) Initialize Processor's Stack Pointer
- 6) Initialize the Heap
- 7) Execute the Constructors and initializers for All Global variables( C++ only )
- 8) Call main( ) function i.e., the Entry point to our Program

//\*\*\*\*\*//

### 4) BUFFER OVERFLOW IN C

**Ans:**

What is Buffer?

A Buffer, in terms of a program in execution, can be thought of as a region of computer's main memory that has certain boundaries in context with the program variable that references this memory.

**#S**

For example :

```
char buff[10]
```

In the above example, 'buff' represents an array of 10 bytes where buff[0] is Left boundary and buff[9] is Right boundary of the Buffer

### **BUFFER OVERFLOW**

A buffer is said to be overflown when the data(meant to be written into memory buffer) gets written past the left or right boundary of Buffer.

This way the data gets written to a portion of memory which does not belong to program variable that references the buffer.

Here is an example :

```
char buff[10];  
buff[10]='a';
```

In the above example,we declared an array of size 10 bytes. Please note that index 0 to index 9 can used to refer these 10 bytes of buffer.

But,in next line,we used Index 10 to store the value 'a'.

This is the point where buffer overrun happens because data gets written beyond Right boundary of the Buffer.

### **Why Buffer Overflows are Harmful?**

1)Buffer overflows,if undetected,can cause your program to crash or produce unexpected results

2)An attacker can cause the program to crash,make data corrupt,steal some private information or run his/her own code.

```
//*****//
```

## **5) CALLBACK FUNCTIONS IN C**

**Ans:**

"Callback" is basically any Executable Code that is passed as an argument to other code,that is expected to call back or execute the Argument at a given time(On Occurence of some Event...Eg:Interrupt)

In simple language,If a reference of a function is passed to another function as an argument to call it,then it is called as a Callback function

In C,a Callback function is a function that is called through a Function pointer

**#S**

In most instances,a Callback will contain three Steps:

- The Callback function
- A Callback Registration
- Callback execution

Below is the Simplest Example of Callback Function

```
#include<stdio.h>
void PrintMsg(void);
void RunCallback(void(*fptr)(void));
int main(void)
{
//Address of the Function PrintMsg() is passed as an argument to function RunCallBack()
RunCallBack(&PrintMsg);
return 0;
}
void PrintMsg()
{
printf("Hello World");
}
//This function takes in a function as an Argument
void RunCallBack(void(*fptr)(void))
{
(*fptr)();
}
```

O/P:Hello World

Advantages of using Callback Functions:

Calling function can pass whatever parameters it wishes to the Called functions

It allows correct information hiding:code that passes a callback to a calling function does not need to know parameter values that will be passed to Function.

```
//*****//
```

**#S**

6) C program to Pass Individual Members of a Structure to Function as Arguments  
[ Method:Call by Value ]

```
#include <stdio.h>
struct student
{
    char name[50];
    int age;
    int roll_no;
    float marks;
};
void Print(char name[],int,int,float);
int main()
{
    struct student s1={"Abhishek",18,50,72.5};
    Print(s1.name,s1.age,s1.roll_no,s1.marks);
    return 0;
}
void Print(char name[],int age,int Roll_no,float marks)
{
    printf("%s %d %d %.2f",name,age,Roll_no,marks);
}
-----
OUTPUT:Abhishek 18 50 72.50
```

#S

## 7) C program for Passing Structure to a Function by Reference

```
#include <stdio.h>
struct Employee
{
    char name[20];
    int age;
    char Doj[10];           //Date of Joining
    char designation[20];
};
void Print(struct Employee *);
int main()
{
    struct Employee data={"Abhishek Mane",27,"2/5/2018","Systems Engineer"};
    Print(&data);
    return 0;
}
void Print(struct Employee *ptr)
{
    printf("Name:%s\n",ptr->name);
    printf("Age:%d\n",ptr->age);
    printf("Date of joining:%s\n",ptr->Doj);
    printf("Designation:%s\n",ptr->designation);
    printf("\n");
}
```

-----

OUTPUT

Name:Abhishek Mane

Age:27

Date of joining:2/5/2018

Designation:Systems Engineer

#S

## 8) Difference between Memory mapped I/O and I/O mapped I/O

Memory mapped I/O	I/O mapped I/O
I/O Devices and Memory share the same Address space	I/O Devices and Memory have separate Address spaces
"MEMR" and "MEMW" Control Signals are used	"IOR" and "IOW" Control Signals are used
16 Bit Address Lines (A0 to A15)	8 Bit Address Lines(A0 to A7)
Shared in 64KB Memory	256 I/P lines and 256 O/P lines
Data transfer takes place between any Register and I/O device	Data transfer is possible between the Accumulator and I/O device only
More Hardware is required to decode 16 Bit Address	Less Hardware is required to decode 8 Bit Address
Instructions Example: LDA,STA  LDA 8000H=Load A with Data read from Input device with 16 Bit Address 8000H STA 8001H=Store contents of A to Output device with 16 Bit Address 8001H	I/O mapped I/O uses "IN" and "OUT" instructions for access

## 9) Some Excellent Websites for Learning Linux,"RTOS","MATLAB" & also Lots of Interview questions related to Embedded 'C'

🔗 <https://embetronicx.com>

🔗 <https://aticleworld.com>

🔗 <https://www.embhack.com>

🔗 [deepblueembedded.com](http://deepblueembedded.com)

🔗 [Hackaday.com](http://Hackaday.com)

🔗 [maxembedded.com](http://maxembedded.com)

🔗 [www.piembstech.com](http://www.piembstech.com)

🔗 <https://lnkd.in/e4KBTvt>

🔗 <https://lnkd.in/ecch5BX>

🔗 [www.engineersgarage.com](http://www.engineersgarage.com)

//\*\*\*\*\*//

#S



## 10) List of some Important Embedded System Interview Questions

- ❑ What is the Significance of "Watchdog Timer" in Embedded Systems?
- ❑ Explain Read Modify Write technique?
- ❑ Explain LCALL
- ❑ What is the Advantage of "ACALL" over "LCALL"?
- ❑ What is Virtual Memory?
- ❑ What is "Interrupt Latency" & how we can reduce it?
- ❑ What are the different activities which a STARTUP Code performs?
- ❑ Explain Boot process of a Microcontroller
- ❑ What is the function of DMA Controller in Embedded Systems?
- ❑ How is a program executed "Bit by bit" or "Byte by Byte"?
- ❑ Difference between Memory Mapped I/O and I/O mapped I/O
- ❑ How "Interrupts" are handled in a Microcontroller?
- ❑ What is the Role of a "Bootloader" in Embedded Systems?
- ❑ Difference between "Bit Rate" and "Baud Rate"
- ❑ What is Ist File?
- ❑ What is EQU?
- ❑ What is the function of Simple Thread Poll in Embedded Systems?
- ❑ What are the different types of customizations that are used with "Volatile" Keyword?

//\*\*\*\*\*//

## 11) List of some Tricky 'C' Interview Questions

- ➡ What would malloc(0) return?
- ➡ How will you write a Function which takes Two Arguments --> a Byte and field in the Byte and returns value of field in that Byte?
- ➡ Declare a manifest constant that returns number of Seconds in a year using Preprocessor? Disregard Leap Years in your answer
- ➡ Write standard "MIN" macro that is a macro that takes Two arguments and returns smaller of the two arguments
- ➡ What is the problem associated with gets( ) function and how to avoid it?
- ➡ How "Free" function knows Size of memory on Heap to be de-allocated?
- ➡ How to Call a function before main( )?
- ➡ Create a function that accepts different types of Data arguments & returns an integer
- ➡ What are various reasons for "Segmentation Fault" in C?
- ➡ How will you override an existing Macro in C?
- ➡ How to print a string containing '%' in printf?

//\*\*\*\*\*//

#S

### 13 ) TIMERS

ANS:

**METHOD 1: Generating Time delay by Loading value in Timer Register & simply Polling Timer Interrupt Flag**

A Timer in a simplest term is a Register.

Timer inside a microcontroller is a Free running Binary Counter.

Timer increments for each pulse applied to it. It counts continuously from 0 to  $(2^n - 1)$  where n is no of Timer Bits

8 Bit Timer Range 0 to 255

16 Bit Timer Range 0 to 65535

PIC Microcontroller takes 4 Clock Cycles to Execute 1 single instruction Hence  $F_{osc}/4$

#### \* TIMER TICKS \*

A "Timer Tick" is simply 1 Clock Cycle or the time between a Timer increment or decrement

So if Timer is clocked at 1MHz, then each Timer Tick would be 1us

It means value of Timer will be incremented every 1us in our Code

#### CALCULATIONS:

$F_{osc} \Rightarrow 4\text{MHz}$

Desired Delay = 50ms

Prescaler  $\rightarrow 1$

Timer Tick =  $(\text{Prescaler} / (F_{osc}/4))$

$= 4 / F_{osc}$

$= 4 / 4\text{MHz} = 1\mu\text{s}$

Timer Count = Desired Delay / Timer tick time

$= 50\text{ms} / 1\mu\text{s}$

$= 50000$

Timer is 16 Bit Hence  $65536 - \text{Timercount} = 65536 - 50000 = 15536 (0x3CB0)$

$0x3CB0$ ) and will go upto  $65535 (0xFFFF)$

After that Timer will Overflow & Rollback to 0. When it overflows, Timer Interrupt Flag gets

Timer will Start incrementing from 15536 (Set

#S

```

MICROCONTROLLER --> PIC18F4520
XTAL Frequency used --> 4 MHz

#include<p18f4520.h>
/***** CONFIGURATION BITS SETTINGS *****/
#pragma config OSC=HS
#pragma config PWRT=OFF
#pragma config WDT=OFF
#pragma config DEBUG=OFF,LVP=OFF
#pragma config FCMEN=OFF
/***** MACROS *****/
#define LED PORTBbits.RB0
/***** FUNCTION PROTOTYPES *****/
void delay_50ms(void);
/*****
void main()
{
    ADCON1=0x0F;           //All Port Pins are configured as Digital I/O's
    TRISBbits.TRISB0=0;    //Configure Port Pin as Output
    LED=0;                 //LED is initially OFF
    while(1)
    {
        LED=1;             //LED "ON"
        delay_50ms();
        LED=0;             //LED "OFF"
        delay_50ms();
    }
}
//Function to Generate 50 milliseconds Delay using Timer1
void delay_50ms(void)
{
    T1CON=0x08;            //No Prescaler is used
    TMR1H=0x3C;
    TMR1L=0xB0;
    T1CONbits.TMR1ON=1;    //Start Timer
    while(PIR1bits.TMR1IF==0); //Poll Timer Interrupt Flag
    T1CONbits.TMR1ON=0;    //Turn OFF Timer
    PIR1bits.TMR1IF=0;     //Clear Timer Interrupt Flag
}

-----
TIP:If you want Larger Delay then iterate Function "delay_50ms" in For Loop

```

#S

## 16) TIMERS

**Ans:**

METHOD 2: Generating Time Delay by Keeping Track of Number of Timer Overflows.

This Method is little Tricky.

I am using 8 Bit Timer here

In this method, we are going to declare a Variable "count"

Each time Timer Overflows we are going to increment this variable "count"

We are also going to Monitor value of this "count" variable simultaneously

Once value of this "count" variable reaches predefined calculated value which we calculated, then we are going to perform some action like Toggling LED

**PREREQUISITES:**

Fosc = 20 MHz

Fcpu = Fosc/4 = 5 Mhz

Prescaler: 2

TimerTick = (Prescaler/(Fosc/4))

=  $(2/(20 \times 10^6/4))$

=  $(8/20 \times 10^6)$

= 0.4us

**How to Calculate Timer OverflowCount?**

Timer is 8 Bit

Time taken by Timer for single Overflow

= 256 x TimerTick

= 256 x 0.4us

= 102.4us

We want 1 second delay

So what I am going to do is I am going to divide 1 second by 102.4us to get desired Overflow count

OverflowCount = 1sec/102.4us

= 9766 This means to achieve 1 second Delay, Timer should Overflow 9766 times...!!

**#S**

Below is the "Embedded C" Code

```
MICROCONTROLLER --> PIC18F4520

#include<p18f4520.h>
/***** CONFIGURATION BITS SETTINGS *****/
#pragma config OSC=HS
#pragma config PWRT=OFF
#pragma config WDT=OFF
#pragma config DEBUG=OFF,LVP=OFF
#pragma config BOREN=OFF
#pragma config FCMEN=OFF
/***** MACROS *****/
#define LED PORTBbits.RB0
/*****

void main()
{
    unsigned int count=0;    //Variable that holds Timer Overflow count
    ADCON1=0x0F;             //All Port Pins are configured as Digital I/O's
    TRISBbits.TRISB0=0;     //Configure Port Pin as Output
    LED=0;                   //LED is initially OFF
    T0CON=0x40;              //Timer OFF, 8 Bit Timer mode, Prescaler:2
    while(1)
    {
        T0CONbits.TMR0ON=1; //Turn ON Timer0
        while(INTCONbits.TMR0IF==0); //Wait for Timer0 Overflow
        INTCNbits.TMR0IF=0; //Clear Timer0 Interrupt Flag
        if(count!=9766)
        {
            count++; //Increment count
        }
        else
        {
            count=0; //Reset Count to Zero
            LED=~LED; //Toggle LED
            T0CONbits.TMR0ON=0; //Turn OFF Timer0
        }
    }
}
```

#S

## **17) 'C' function to Transmitt Decimal value from USART onto PC Without using Library Function .**

**This is a Little weird way....But it works Absolutely Fine [100%]**

**This is Very Handy when someone doesn't want to use a Library function**

**All you have to do is Pass the Integer to the Function and Rest all work is done by the Function itself**

### **How does this Function works?**

**Basically what this Function does is that it Splits the Passed Integer into it's Individual Digits in the form of "TEN\_THOUSAND", "THOUSAND", "HUNDRED", "TENS" and "UNIT" places...!!!**

**For example:** If the Passed Integer is 65978...then it will be splitted as '6' '5' '9' '7' '8'

**If you pass "1234" then it will be splitted as '0' '1' '2' '3' '4'**

**So in this way you can Transmitt Decimal value from USART onto PC and Observe it on "Hyperterminal"**

**#S**

```

    %(Modulo Operator) --> Remainder
    /(Dividend) --> Quotient

void Transmitt_Decimalvalue(unsigned int Data)
{
    unsigned char digit1,digit2,digit3,digit4,digit5;
    digit1=Data/10000U;
    digit2=(Data%10000U)/1000U;
    digit3=(Data%1000U)/100U;
    digit4=(Data%100U)/10U;
    digit5=Data%10U;
    if(Data>=10000)
    {
        TXREG=digit1+0x30;           //Convert into "ASCII"
        while(PIR1bits.TXIF==0);
        TXREG=digit2+0x30;
        while(PIR1bits.TXIF==0);
        TXREG=digit3+0x30;
        while(PIR1bits.TXIF==0);
        TXREG=digit4+0x30;
        while(PIR1bits.TXIF==0);
        TXREG=digit5+0x30;
        while(PIR1bits.TXIF==0);
    }
    else if((Data<10000)&&(Data>=1000))
    {
        TXREG=digit2+0x30;
        while(PIR1bits.TXIF==0);
        TXREG=digit3+0x30;
        while(PIR1bits.TXIF==0);
        TXREG=digit4+0x30;
        while(PIR1bits.TXIF==0);
        TXREG=digit5+0x30;
        while(PIR1bits.TXIF==0);
    }
    else if((Data<1000)&&(Data>=100))
    {
        TXREG=digit3+0x30;
        while(PIR1bits.TXIF==0);
        TXREG=digit4+0x30;
        while(PIR1bits.TXIF==0);
        TXREG=digit5+0x30;
        while(PIR1bits.TXIF==0);
    }
}

```

#S

## 18) Very Important and Commonly asked 'C' program in Technical 'C' Interviews

C program to Count number of Set Bits in an Integer

✱ NOTE: This is Brian Kernighan's Method

```
#include <stdio.h>
unsigned int CountSetBits(unsigned int);
int main()
{
    unsigned int number=15;
    printf("SetBits:%d",CountSetBits(number));
    return 0;
}
unsigned int CountSetBits(unsigned int n)
{
    unsigned int count=0;
    while(n!=0)
    {
        n&=(n-1);
        count++;
    }
    return count;
}
-----
OUTPUT --> SetBits:4
```

#S



## 19) Two Most Important Concepts related to "SPI" Protocol & are also Commonly asked in "Embedded C" Interviews

### CLOCK POLARITY AND CLOCK PHASE

#### >>> CLOCK POLARITY(CPOL)

Clock Polarity determines "IDLE" state of the Clock  
If "IDLE" state of Clock is "LOW" then Clock Polarity=0  
If "IDLE" state of Clock is "HIGH" then Clock Polarity=1

#### What is "IDLE" state of Clock?

The "IDLE" state is defined as the period when Chip Select[Active Low Signal] is "HIGH" and transitioning to "LOW" at Start of the transmission and when Chip Select(CS) is "LOW" and transitioning to "HIGH" at End of transmission.

#### >>> CLOCK PHASE(CPHA)

Clock Phase determines the Edge[Rising/Falling Edge] at which Data Read/Write(R/W) occurs  
Depending on the CPHA Bit, the Rising or Falling Clock Edge is used to Sample and/or Shift the Data

//\*\*\*\*\*//

## 20) Setter and Getter Functions

### Ans:

"Setter" function is used to Set or Update value to the variable

"Getter" function is used to retrieve value of the variable


### Where these can be used in "Embedded C"?

For Example: PWM Application Code

We can use "Setter" function to Set the PWMDutycycle and "Getter" function to retrieve the PWMDutycycle

#S

Below 'C' program illustrates How Setter and Getter functions work



```
#include<stdio.h>
void Setvalue(int);
int Getvalue(void);
static int x;
int main()
{
    Setvalue(10);
    printf("x=%d",Getvalue());
    return 0;
}
/****Setter Function****/
void Setvalue(int value)
{
    x=value;
}
/****Getter Function****/
int Getvalue(void)
{
    return x;
}
-----
OUTPUT --> x=10
```

#S

## 21) MPORTANT 'C' PROGRAM COMMONLY ASKED IN "EMBEDDED C" INTERVIEW

### C program to Copysting using Pointers

```
#include<stdio.h>
#include<string.h>
void Copysting(char *,char *);
int main(void)
{
    char sourcestring[15]="Helloworld!!";
    char Targetstring[15];
    printf("Sourcestring=%s\n",sourcestring);
    Copysting(Targetstring,sourcestring);
    printf("Targetstring=%s",Targetstring);
    return 0;
}
void Copysting(char *target,char *source)
{
    while(*source!='\0')
    {
        *target=*source;
        source++;           //Move Pointer to next location
        target++;           //Move Pointer to next location
    }
    *target='\0';           //Append "NULL" character at end
}

-----
OUTPUT
Sourcestring=Helloworld!!
Targetstring=Helloworld!!
```

#S

## 22) C' function to Display ADCResult(10 Bit ADC)[0 to 1023] on 16x2 LCD

**Ans:**

This function is used to Display ADCResult of a 10 Bit ADC(values between 0 to 1023) on a 16x2 LCD

All you have to do is Pass the variable to this function in which your Converted "ADCResult" is stored....Remaining stuff is handled by this function itself

If you Find this Interesting/Exciting then Please Let me know in the Comments

```
void DisplayADCresult(unsigned int adc_count)
{
    unsigned int x=0;
    unsigned char temp[4];
    temp[0]=((adc_count/1000U)+48);           //Extract Thousands Digit
    temp[1]=(((adc_count/100U)%10U)+48);      //Extract Hundreds Digit
    temp[2]=(((adc_count/10U)%10U)+48);       //Extract Tens Digit
    temp[3]=((adc_count%10U)+48);             //Extract Ones digit
    for(x=0;x<4;x++)
    {
        lcddata(temp[x]);                    //Send values to LCD Data Port one-by-one
    }
}
```

**#S**

### 23) IMPORTANT PROGRAMMING INTERVIEW QUESTIONS ON ARRAYS

- > How do you Find the missing number in a given integer Array of 1 to 100?
- > Find the Duplicate number on a given integer Array
- > All pairs of Integer array whose sum is equal to a given number
- > Find How do you Find the Largest and smallest number in an unsorted Integer Array
- > Find Duplicate numbers in an Array if it contains multiple duplicates
- > Remove Duplicates from an Array in a given place
- > Find multiple missing numbers in a given Integer Array with duplicates
- > How to rotate an Array Left and Right by a given number K?
- > Given an Array of Integers sorted in Ascending Order, Find the Starting and Ending position of a given value

//\*\*\*\*\*//

### 24) IMPORTANT "EMBEDDED C" PROGRAMMING INTERVIEW QUESTION \*□□

C Program to Swap Endians[Convert One Endian into another Endian]

```
#include<stdio.h>
#include<inttypes.h>
uint32_t SwapEndians(uint32_t);
int main()
{
    uint32_t Data=0x12345678;
    uint32_t SwappedData=0;
    SwappedData=SwapEndians(Data);
    printf("SwappedData:0x%x\n",SwappedData);
    return 0;
}
uint32_t SwapEndians(uint32_t Value)
{
    uint32_t Converted_value=0;
    Converted_value|=(Value & 0xFF000000)>>24;
    Converted_value|=(Value & 0x00FF0000)>>8;
    Converted_value|=(Value & 0x0000FF00)<<8;
    Converted_value|=(Value & 0x000000FF)<<24;
    return Converted_value;
}
```

-----  
OUTPUT

SwappedData:0x78563412

#S

## 25) IMPORTANT 'C' INTERVIEW QUESTIONS 2



-> IMPORTANT AND RARELY ASKED 'C' INTERVIEW QUESTIONS <-

- 1)What happens when we Call a Function in C?
- 2)How we can return Multiple values from a Function in C?
- 3)What will happen if Memory regions "Stack" and "Heap" Overlap with each other?
- 4)How "Free" function knows Exact Size of memory on "Heap" to be deallocated?
- 5)Explain "Segmentation Fault" in 'C'
- 6)Why "String Literals" cannot be modified?
- 7)How will you Call a Function before main() in C?
- 8)How will you Override an Existing MACRO in 'C'
- 9)What are "Memory Leaks"?
- 10)How to prevent "Memory Leaks"?
- 11)What is NULL Pointer assignment error in C?
- 12)What is NULL Pointer Exception in C?
- 13)Why Pointer Arithmetic cannot be performed on "Void Pointer"?
- 14)How "Dynamic Linking" is different from "Static Linking"?

HAPPY LEARNING...!!!

---

## 26) How to Solve Dangling Pointer Problem?

**Ans:**

**There are 2 Ways to Solve this issue**

**1) There is No Point in Returning Address of a "Local Variable" from a Function because as We Exit from the function, Variable will go to "Out of Scope"**

**So Declare the Variable as "Static"**

**2) Convert the "Dangling Pointer" into "NULL Pointer"**

**Means???**

**Means Assign "NULL" to the Pointer**

**Let's understand it from Simple Example below**

```
int *ptr=(int *)malloc(5*sizeof(int));
```

```
free(ptr); //ptr now Becomes Dangling Pointer
```

```
ptr=NULL; //Now ptr is NULL pointer
```

**So After Assigning "NULL" to the Pointer It will not Point to Any Memory location further**

**#S**



**27) IMPORTANT "EMBEDDED SYSTEM" INTERVIEW QUESTION**  
**HOW "INTERRUPTS" ARE HANDLED BY MICROCONTROLLER?**

**BELOW ARE THESE STEPS**

- 1. MICROCONTROLLER FINISHES THE CURRENT INSTRUCTION IT IS EXECUTING AND SAVES ADDRESS OF NEXT INSTRUCTION (PROGRAM COUNTER) ON THE STACK**
- 2. IT ALSO SAVES CURRENT STATUS OF ALL THE INTERRUPTS INTERNALLY (I.E., NOT ON THE STACK)**
- 3. IT JUMPS TO THE MEMORY LOCATION OF "INTERRUPT VECTOR TABLE". INTERRUPT VECTOR TABLE CONTAINS ADDRESSES OF ALL ISR'S**
- 4. MICROCONTROLLER FETCHES ADDRESS OF THE ISR FROM "INTERRUPT VECTOR TABLE" AND JUMPS TO IT. IT STARTS EXECUTING THE INTERRUPT SERVICE SUBROUTINE UNTIL IT REACHES LAST INSTRUCTION OF SUBROUTINE, WHICH IS RETFIE (RETURN FROM INTERRUPT EXIT)**
- 5. UPON EXECUTING RETFIE INSTRUCTION, MICROCONTROLLER RETURNS TO THE PLACE WHERE IT WAS INTERRUPTED  
FIRST IT GETS THE PROGRAM COUNTER (PC) ADDRESS FROM STACK BY POPPING TOP BYTES OF THE STACK INTO PC. THEN IT STARTS TO EXECUTE FROM THAT ADDRESS**



## 28) Technical Questions

1. Add function, here you have to care about the data type range, for example

```
uint8 x,y,z;
```

```
x = 255;
```

```
y = 255;
```

```
z = x+y; // overflow will occur cause x+y = 510 > 2^8 - 1
```

2. factorial function (normal way or using recursion)

3. Matrix operations, multiplication, addition

4. sine function without math.h.

5. searching and sorting

6. knowledge about UART, CAN, ADC, SPI, Timers, Interrupts.

## 29) some functions to comment on it as:

- > switch case(one of the cases without break)
- > postincrement and preincrement(++i,i++)->Result
- > #define?? #include?? #pragma?? #asm?? #ifdef...#endif?
- > const?? extern?? volatile?? static?? register??
- > 'int' type?? 'unsigned char' type?? (size)
- > static and dynamic variables
- > diff. bet function and Macros
- > some MISRA rules
- > specifications of embedded SW

#s

### 30)What are Lvalues and Rvalues?

An object is a contiguous region of memory storage. An lvalue (pronounced: L value) is an expression that refers to such an object. The original definition of lvalue referred to "an object that can appear on the left-hand side of an assignment." However, const objects are lvalues, and yet they cannot appear on the left-hand side of an assignment. An expression that can appear in the right-hand side of an expression (but not in the left-hand side of an expression) is an rvalue. For example:

```
#include <string>
using namespace std;
int& f();

void func()
{
    int n;
    char buf[3];
    n = 5; // n is an lvalue; 5 is an rvalue
    buf[0] = 'a'; //buf[0] is an lvalue, 'a' is an rvalue
    string s1 = "a", s2 = "b", s3 = "c"; // "a", "b", "c" are rvalues
    s1 = // lvalue
        s2 +s3; //s2 and s3 are lvalues that are implicitly converted to rvalues
    s1 =
        string("z"); // temporaries are rvalues
    int * p = new int; // p is an lvalue; 'new int' is an rvalue
    f() = 0; // a function call that returns a reference is an lvalue
    s1.size(); // otherwise, a function call is an rvalue expression
}
```

An lvalue can appear in a context that requires an rvalue; in this case, the lvalue is implicitly converted to an rvalue. An rvalue cannot be converted to an lvalue. Therefore, it is possible to use every lvalue expression in the example as an rvalue, but not vice versa.

31) How is a signed negative number stored in memory?

A:

In most intel architectures, signed negative numbers are stored in memory as two's Complement.

0101 = +5

1011 = -5

32 ) Which is the best way to write Loops?

Q: Is Count Down\_to\_Zero Loop better than Count\_Up\_Loops?

A: Always, count Down to Zero loops are better.

This is because, at loop termination, comparison to Zero can be optimized by compiler. (Eg using SUBS)

Else, At the end of the loop there is ADD and CMP.

33) What is loop unrolling?

A: Small loops can be unrolled for higher performance, with the disadvantage of increased code size. When a loop is unrolled, a loop counter needs to be updated less often and

fewer branches are executed. If the loop iterates only a few times, it can be fully unrolled,

so that the loop overhead completely disappears.

```
int countbit1(uint n)
```

```
{ int bits = 0;
```

```
while (n != 0)
```

```
{
```

```
if (n & 1) bits++;
```

```
n >>= 1;
```

```
}
```

```
#s
```

```

return bits;
}

int countbit2(uint n)
{ int bits = 0;
  while (n != 0)
  {
    if (n & 1) bits++;
    if (n & 2) bits++;
    if (n & 4) bits++;
    if (n & 8) bits++;
    n >>= 4;
  }
  return bits;
}

```

34) How does, taking the address of local variable result in unoptimized code?

A: The most powerful optimization for compiler is register allocation. That is it operates the variable from register, than memory. Generally local variables are allocated in registers. However if we take the address of a local variable, compiler will not allocate the variable to register.

35) How does global variable result in unoptimized code?

A: For the same reason as above, compiler will never put the global variable into register. So its bad.

#s

**36) Which is better a char, short or int type for optimization?**

A: Where possible, it is best to avoid using char and short as local variables. For the types char and short the compiler needs to reduce the size of the local variable to 8 or 16 bits after each assignment. This is called sign-extending for signed variables and zeroextending for unsigned variables. It is implemented by shifting the register left by 24 or 16 bits, followed by a signed or unsigned shift right by the same amount, taking two instructions (zero-extension of an unsigned char takes one instruction). These shifts can be avoided by using int and unsigned int for local variables. This is particularly important for calculations which first load data into local variables and then process the data inside the local variables. Even if data is input and output as 8- or 16-bit quantities, it is worth considering processing them as 32-bit quantities.

**37) Can structures be passed to the functions by value?**

Yes , will be waste of memory because call by value mean the function will allocate locale memory in stack to save this local variables which in this case be the hole structure

The best way is to pass by address

**38) Why cannot arrays be passed by values to functions?**

When a array is passed to a function, the array is internally changed to a 'pointer'. And pointers are always passed by reference.

**39) Advantages and disadvantages of using macro and inline functions?**

Advantage: Macros and Inline functions are efficient than calling a normal function. The times spend in calling the function is saved in case of macros and inline functions as these are included directly into the code.

#s

Disadvantage: Macros and inline functions increased the size of executable code.

Difference in inline functions and macro

- 1) Macro is expanded by preprocessor and inline function are expanded by compiler.
- 2) Expressions passed as arguments to inline functions are evaluated only once while `_expression` passed as argument to inline functions are evaluated more than once.

#### 40) What happens when recursion functions are declared inline?

inline function's property says whenever it will be called, it will copy the complete definition of that function. Recursive function declared as inline creates the burden on the compiler's execution.

The size of the stack may/may not be overflow if the function size is big.

#### 41) Scope of static variables?

if we want the value to be extent throughout the life of a program, we can define the local variable as "static."

#### 42) Which way of writing infinite loops is more efficient than others?

there are 3 ways.

**While(1)**

```
{  
statements  
}
```

**or**

```
do  
{  
statements  
}  
while(1);  
or  
for(;;)
```

**#s**

#### 43) What is interrupt latency?

In real-time operating systems, interrupt latency is the time between the generation of an interrupt by a device and the servicing of the device which generated the interrupt.

#### 44) What are the different storage classes in C?

There are four type of storage classes in C. These are used to store variables. These are Extern, Static, Register and Auto.

1. Auto is the default class assigned to any variable.  
eg. if we define `int x=10;` then it means `auto int x=10`
2. register and static differ in only one grounds that register is there in the cpu and static in the main memory.
3. extern is global and can be accessed by any program and anywhere.

#### 45) What are the different qualifiers in C?

**Type qualifiers can be characterized by the restrictions they impose on access and cacheing.**

**\* const**

**No writes through this lvalue. In the absence of this qualifier, writes may occur through this lvalue.**

**\* volatile**

**No cacheing through this lvalue: each operation in the abstract semantics must be performed (that is, no cacheing assumptions may be made, since the location is not guaranteed to contain any previous value). In the absence of this qualifier, the contents of the designated location may be assumed to be unchanged except for possible aliasing.**

**\* restrict**

**Objects referenced through a restrict-qualified pointer have a special association with that pointer. All references to that object must directly or indirectly use the value of this pointer.**

**#s**

**In the absence of this qualifier, other pointers can alias this object. Cacheing the value in an object designated through a restrict-qualified pointer is safe at the beginning of the block in which the pointer is declared, because no pre-existing aliases may also be used to reference that object. The cached value must be restored to the object by the end of the block, where pre-existing aliases again become available. New aliases may be formed within the block, but these must all depend on the value of the restrict-qualified pointer, so that they can be identified and adjusted to refer to the cached value. For a restrict-qualified pointer at file scope, the block is the body of each function in the file.**



**THANK  
YOU  
GUYS**

**#S**