01 How to measure the size of any variable without sizeof operator?

The Solution:

```
# define SIZE_OF(X) ((char*)(&X+1)-(char*)&x))
void main()
{
     short int x;
     printf("%d",SIZE_OF(x));
}
```

Ans : 2

The Explanation :

Lets look how the answer is coming without sizeof(). First declare a short int x variable. By using &x we get the base address of the variable x and by adding 1 to it we get the base address of next short int type. Hence the resulting address of (&x + 1) will be 2 bytes more than the base address of the variable x. But if we just display the difference between the base address of x and the incremented address, then the difference will be 1 means"1 block of short int type has been added" but we need the result of size of variable in terms of bytes not in terms of blocks. This can be achieved if we typecast the address into char *, because address of char datatype will always be in blocks of 1 byte, hence if the difference between the base address of x and the incremented address is displayed in terms of char type , then the difference will be displayed as 2, because the difference is actually 2 blocks or 2 bytes in terms of char type representation.

02 **How can you measure the size of a type without using sizeof operator?**

The Solution:

```
#define SIZE_OF(T) (((T*)0)+1)
void main( )
{
     clrscr();
     printf("%d", SIZE_OF(long int));
}
```

Ans : 4

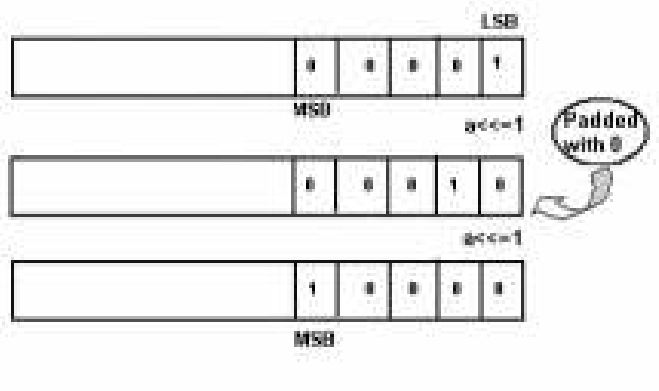
The explanation:

Whenever we typecast any constant, it converts the constant into a base address of specified data type and as we know address + 1 will always return the next address of its type or we can say the address of next block of memory of that type, so accordingly if 1 will be added with the address 0 (as specified in (long int *)0) then it will return the size of any data type in bytes. For example 1 for char, 2 for short integer, 4 for long integer or float and so on.

03 **How can you measure the size of a bit field?**

The Solution:

```
struct xx
{
     char x:5;
};
void main()
{
     struct xx a;
```



```
int counter=0;
     clrscr();
     a.x=1;
     while(a.x) // till 0
     {
a.x<<=1;
counter++;
}
     printf("%d",counter);
     getch();
}
```

Ans : 5

The explanation:

The sizeof operator can't measure the size of a bit field.

So to get the output, here we have assigned 1 to the variable x which can assign 1 only in 5 bits out of 8 bits. When the value of a.x is shifted towards left by 1, the corresponding memory representation is depicted in Figure 2, where 1 has shifted to the left by one place and 0 gets padded to the LSB. Figure 3 shows the memory representation of a.x after getting shifted 4 times and if you notice the loop, the value of counter is 4 as it gets incremented by 1 in each shift. And finally the value of a.x becomes 0 in 5th shift, which makes the loop to terminate. At this stage the value in the counter variable is 5 which is nothing but the size of the bit field.

04 How can you measure the size of a variable in terms of bits?

The Solution:

```
void main()
{
     int i=0,j=0;
```

```
        float a;
        clrscr();
        ((unsigned char*)&a)[j]=1;
        while(a)
        {
            i++;
            ((unsigned char*)&a)[j]<<=1;
            if(i%8==0)
            {
                j++;
                ((unsigned char*)&a)[j]=1;
            }
            if(a==0 && ((unsigned char*)&a)[j]==128)
                i++;
        }
        printf("%d",i);
        getch();
    }
```

Ans : 32

The Explanation:

Size of a variable cannot be calculated using sizeof operator because it returns the value in terms of bytes. Still then we can calculate the size of the variable in terms of bits by checking each individual bit and increasing the counter. This is possible by the value 1. As a bit can store only two values which is either 0 or 1. So we check the value of the variable repeatedly for each bit with a value 1 and the rest of the bit to zero. The moment the value of the variable becomes 0 the current value in the variable counter contains the number of bits.

How many different algorithms that we can use to swap between two variables?

Lets see….

The Solution:

Logic 1:

```
int a=5,b=10;
int c;            //using extra memory space

    c=a;
    a=b;
    b=c;
```

The above example shows the swapping of two variables a and b by using extra memory space for variable c. In the first step the value of a is assigned to c, thus the value of c now becomes 5, after that b is assigned to a and the value of a now becomes 10 and finally value of c is assigned to b. Thus the values of a and b have been swapped.

Logic 2:

```
        a=a+b;        // Using arithmetic operators
        b=a-b;
        a=a-b;
```

In the above code snippet an interchange in the values of a and b is done but without using any extra memory space. In the first statement a is assigned the value of (a+b)i.e., 15. In the 2nd statement b is assigned the value of (a – b), i.e., (15-10) equal to 5 and in the 3rd statement a is replaced with (a – b), i.e. 10.

Logic 3:

```
    a=a^b;            // Using Bit wise operators
    b=b^a;
    a=a^b;
```

In line 1, a=a^b results in the value of a becoming 15 and b becoming 10. In the 2nd line b=b^a, means that a has the same 15 but the value of b is now 5. In the subsequent line 3, the expression a=a^b results in varible a becoming 10 and b becoming 5, thereby completing the swap.

line	operation	value of a	Value of b
1	a=a^b	15	10
2	b=b^a	15	5
3	a=a^b	10	5

Logic 4:

```
asm mov ax,a;
asm mov bx,b;
    asm mov cx,bx;
    asm mov bx,ax;
    asm mov ax,cx;
    asm mov a,ax;
    asm mov b,bx;
```

Above example shows the swapping of two variable using assembly programming and without using assignment operator.

Logic 5:

```
a^=b^=a^=b;              // Using one line statement
```

In the above example a and b is interchanged in a single line statement using the Compound assignment operator, where its order of evaluation is from right to left. So the value b is XORed with a and the result is assigned to a (i,e a=15) . In the second compound assignment operator a is XORed with b and the result is assigned to b (i,e b=5). Finally b is again XORed with a and result is assigned to a and a becomes 10. So both are swapped.

Logic: 6

```
a=(a+b)-(b=a);           // Using One line statement
```

In the above example, parenthesis operator enjoys the highest priority and the order of evaluation is from left to right. Hence a+b is evaluated first and replaced with 15. Then (b=a) is evaluated and the value of a is assigned to b, which is 5. Finally a is replaced with 15 – 5 , i.e 10.

How to swap between first and second byte of an integer in one line statement.

The solution:

```
x=x<<8 | x>>8;
```

The explanation:

Let value of x be 300.

x = 00000001 **00101100**

Fig.1

x<<8 = 00101100 **00000000**

Fig.2

x>>8 = 00000000 **00000001**

Fig.3

x<<8 | x>>8 = 00101100 00000001

Fig.4

In Figure 1, we have the binary representation of x, depicting the value 300. In Figure 2, the value of the 1st byte of x is shifted to the 2nd byte. In Figure 3, the value of the 2nd byte of x is shifted to the 1st byte. Once we use the bitwise OR, the resultant binary representation is depicted in Figure 4.

How do we write a general swap macro in C, a macro which can swap any type of data whether it is int, char, float, struct, etc.

The Solution:

```
#define SWAP(a, b, type) { type t = a; a = b; b = t; }
```

The Explanation:

You have to call the macro SWAP with 3 parameters. We have to provide the two variables to be swapped along with the type they belong to.

For example:

```
float f1 = 10.3, f2 = 12.7;
SWAP(f1,f2 float);
```

WAP using C to swap between two bits of a particular number.

The Solution:

```
void main()
{
        int number, ithbit, jthbit;
        clrscr();
        printf("\nEnter a no. : " );
        scanf("%d",&number);
        printf("\nEnter the two positions : " );
        scanf("%d%d",&ithbit,&jthbit);

        ithbit--;
        jthbit--;
```

```
        if( ((number & (1<<ithbit)) >>ithbit)  !=
((number&(1<<jthbit))>>jthbit) )
        {
        number = number ^ (1<<jthbit);
             number = number ^ (1<< ithbit);
        }
        printf("The resultant no is %d",number);
        getch();
}
```

The Explanation:

let's have a look at the program. Immediately after getting the bit number from the user we have decremented each bit by one. It's only because the actual bit number starts from 0 where as for the user it can be considered as the 1st bit. When the user will give to interchange the ith and jth bit, we have changed it to (i-1)th and (j-1)th bit.

Here we want to swap the contents of two bits of a particular number. So the contents of a bit either may be 1 or 0. So before going to swap, we first check whether the content of the given bits are equal or not. If the content is not equal then we will go for interchange otherwise not.

(number & (1 << ithbit)) >> ithbit gives the content of ithbit.

Let's take an example as the user wants to interchange the content of 3rd bit and 6th bit of 33.

So for us its 2nd and 5th bit.

Binary of 33	0000	0000	0010	0001 ->	content of 3rd bit is 0
1 << 2	0000	0000	0000	0100 ->	by shifting 1 twice towards left.
&					
33&(1<<2)	0000	0000	0000	0000	->

33&(1<<2)>>2	0000	0000	0000	0000	now the 3rd is in the LSB.———Eq.1
Again					
Binary of 33		0000	0000	0010	0001
1 << 5		0000	0000	0010	0000
&					
33 &(1<<5)		0000	0000	0010	0000 ->all are reset expect 5th bit
33 &(1<<5) >> 5	0	000	0000	0000	0001 -> now the 5th bit is in the LSB———Eq. 2

Now equation 1 and 2 are not equal. So interchange takes place.

Lets consider the two statements in the if structure.

1st statement -> number = number ^ (1 << jthbit)

Here jthbit = 5.

1 << jthbit		0000	0000	0010	0000
number		0000	0000	0010	0001
	^				
number =		0000	0000	0000	0001 -> the new value of number after 1st statement

2nd statement -> number = number ^ (1<< ithbit);

ithbit = 2.

1 << 2		0000	0000	0000	0100
number		0000	0000	0000	0001
	^				
number		0000	0000	0000	0101 -> the new value of number after 2nd statement.

The given value is 33 0000 0000 0010 0001 -> 3rd bit = 0 and 5th bit = 1

Now the new value is 0000 0000 0000 0101 -> 3rd bit = 1 and 5th bit = 0 which were interchanged and the new value of number is 5.

In reality,the bit number actually starts from 0th bit but for the user it can be considered to start from the 1st bit.

09 How to convert a long int to its binary format without using modulo(%) operator and division (/) operator.

The Solution:

Logic 1:

```
#include<limits.h>
int convert(long int n,long int m)
{
     if(m==1<<31)
     {
          return;
     }
     (n & m)?convert(n,m*2),printf("1
"):(convert(n,m*2),printf("0 "));
     return 1;
}

void convert(long int  m)
{
     int i;
     for(i=31;i>=0;i–)
          (m&(1L<<i))?printf("1 "):printf("0 ");
}
```

```
void main()
{
     long int n;
     printf("\n\nEnter a no.  :");
     scanf("%ld",&n);
     convert(n,1);
     getch();
}

Case 2:
void main()
{
     unsigned long int x;
     int i;
     printf("\nEnter a long integer : ");
     scanf("%ld",&x);
     for(i=0;i<32;i++)
         printf("%d",x<<i>>31);
     getch();
}
```

Ans : Enter a no. : 65536

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

The Explanation:

Case 1:

Here we are calling the function convert having two parameters, one is the number which is to be converted to binary along with one. The function is called recursively and each time the second parameter is multiplied by two until it reaches the LONG_MAX . Simultaneously the corresponding bit of the number is checked using bitwise & operator and the value is printed on screen.

Convert function is an iterative one . Here we directly check the bits of the number starting from 31 to 0 by setting the 1 in the corresponding bit using left shift operator (<<).

Case 2:

Here we shift each bit to the MSB(starting from the MSB towards the LSB) by left shifting it by i and then that bit is shifted to the LSB by right shifting it by 31 and after shifting rest bits are padded to zero. So that the required value starting from 31st bit of the number towards the 0th bit is stored in the LSB of the number in the corresponding iteration.

10 What is the most efficient (fastest) way to divide any number by 4?

The Solution:

```
void main( )
{
     int x ;
     printf("\nEnter any number : ");
     scanf("%d",&x);
     printf("%d",x>>2);
}
```

Ans :
Enter any number : 16
4

The explanation:

We know that if we right shift a number by 1 we get the result which is same as the number divided by 2. So to get the result of dividing a number by 4 we have to left-shift the number by 2. This is a more efficient or faster way of getting the result because it directly interacts with the bits of the number.