**Union assignments**Mandatory  
1. Refer the code below and comment on size of the given structure considering  
a. Structure as union  
b. Structure as struct  
c. arr  
d. uarr  
\_\_\_ Job  
{  
char name[32];  
unsigned short ucount;  
float salary;  
int workerNo;  
char \*orgname;  
};  
\_\_\_ Job myvar; //could of union or of struct  
Struct Job arr[10];  
Union Job uarr[10];

Ans:

**a. Structure as Union**

In a union, the size is determined by the largest member. The largest member is char name[32] which is 32 bytes. However, due to alignment requirements, the size might be larger.

**b. Structure as Struct**

In a struct, the size is the sum of the sizes of all members, considering alignment:

* char name[32]: 32 bytes
* unsigned short ucount: 2 bytes (aligned to 2 bytes)
* float salary: 4 bytes (aligned to 4 bytes)
* int workerNo: 4 bytes (aligned to 4 bytes)
* char \*orgname: 8 bytes (assuming 64-bit system)

**c. arr**

arr is an array of 10 Job structs. If each struct is 56 bytes, arr would be 10 \* 56 = 560 bytes.

**d. uarr**

uarr is an array of 10 Job unions. If each union is 32 bytes, uarr would be 10 \* 32 = 320 bytes.

2. Refer Job datastructure in Q#1 above. Using uarr, perform below operations.  
a. Read and store salary  
b. Read and store workerNo  
Comment on values of output if salary and workerNo are printed in order.  
Justify your statement.

Ans:

**a. Read and Store Salary**

uarr[0].salary = 50000.0;

float storedSalary = uarr[0].salary;

**b. Read and Store WorkerNo**

uarr[0].workerNo = 12345;

int storedWorkerNo = uarr[0].workerNo;

Since uarr is a union, all members share the same memory location. Writing to salary and then reading workerNo (or vice versa) will result in undefined behavior because the data will overlap. The output values will not be meaningful.

3. Refer Job datastructure in Q#1 above. Assume that myvar is a structure  
variable. If I need to place 2 bytes (i.e 0x0102) as ucount using a char \*ptr  
then list all possible statements that can be used in \_\_\_\_\_.  
[Let solutions include cases such as  
i. using base address of ucount  
ii. using relative address of ucount w.r.t to base address of  
myvar]  
int main()  
{  
char \*ptr = &myvar;  
\_\_\_\_\_\_\_\_\_\_\_ = 0x01;  
\_\_\_\_\_\_\_\_\_\_\_ = 0x02  
}

Ans:

**i. Using Base Address of ucount:** To place the bytes 0x01 and 0x02 into ucount using its base address, you can directly manipulate the memory location of ucount.

\*ptr =0x01;

\*(ptr+1)= 0x02

**ii. Using Relative Address of ucount w.r.t Base Address of myvar:** To place the bytes using the relative address of ucount with respect to the base address of myvar, you need to calculate the offset of ucount within the structure**.**

As ucount is at 32 we directly access that bit

\*(ptr+32)=0x01; and \*(ptr+33)=0x02;