**Union assignments**

1. Refer the code below and comment on size of the given structure considering

A) The Job data structure has the following members:

* char name[32] (32 bytes)
* unsigned short ucount (typically 2 bytes)
* float salary (typically 4 bytes)
* int workerNo (typically 4 bytes)
* char \*orgname (typically 4 bytes on a 32-bit system, 8 bytes on a 64-bit system)

1. **Structure as union**

**A)** In the case of a union, all members share the same memory space. The size of the union is determined by the size of its largest member. Here, the largest member is name[32], which is 32 bytes, so the size of the union is 32 bytes.

**b. Structure as struct**

**A**) In the case of a struct, each member occupies its own space, and the total size of the structure is the sum of the sizes of its individual members**.**

**c. arr**

**A)** An array of structs will take 10 \* sizeof(struct Job) bytes. Since sizeof(struct Job) is 48 bytes, the size of the array will be:

sizeof(arr); // 10 \* 48 = 480 bytes

**d. uarr**

**A)** An array of unions will take 10 \* sizeof(union Job) bytes. Since sizeof(union Job) is 32 bytes, the size of the array will be:

sizeof(uarr); // 10 \* 32 = 320 bytes

\_\_\_ Job

{

char name[32];

unsigned short ucount;

float salary;

int workerNo;

char \*orgname;

};

\_\_\_ Job myvar; //could of union or of struct

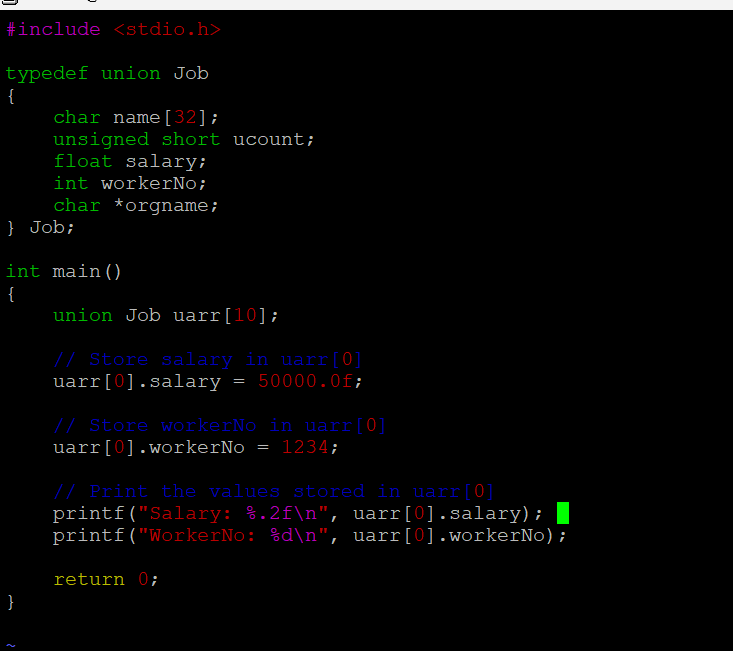
Struct Job arr[10];

Union Job uarr[10];

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.

A) we are using a **union**, the members salary and workerNo share the same memory. After assigning a value to salary, the salary field occupies the union's memory location. When we assign a value to workerNo, the memory of salary is overwritten by workerNo because they share the same space.

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

}

A computer screen shot of a black screen with colorful text

Description automatically generated