Assignment No. - 10

1. What is concept of structure in c programming ? Explain with example.

→ structure is a concept which comes under a userdefined datatype.

- structure is a collection of heterogeneous elements i.e as per user need he can take data type in structure, so structure is called as userdefined datatype.

- The concept of structure can store primitive, userdefined, derived data type in it.

- structure in c and it is used in c++ & Java as class.

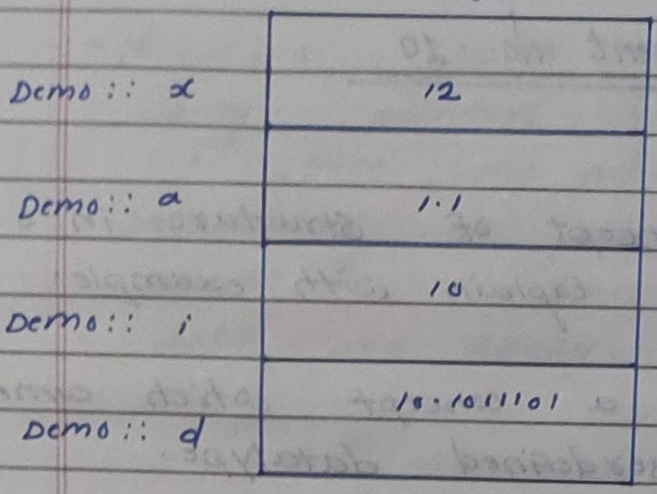
e.g.

keyword

```
struct Demo
{
    int x;
    float a;
    int i;
    double d;
};
```

structure members (Body)

obj1



- The structure will get memory when we define object / structure variable. now each member of structure gets seperate memory to each data member.

- Here total 120 byte mumory gets allocated to structure.

- we cannot initialize the data elements in structure, for that we need to use direct accessing (. dot) operator.

- struct Demo obj1 ;

```
obj1.x = 12;
obj1.a = 1.1;
obj1.i = 10;
obj1.d = 10.1011101;
```

Q.2. What is the difference between Structure & union.

—

- Structure and union are considered as user defined data types.

- Structure and union can hold any heterogeneous datatype based on programmer requirement.

- Structure can store any primitive, derived & user defined data type in it.

- In case of union memory gets allocated for only the largest member of a union.

- As the memory gets allocated for only one member we can store only one value at a time.

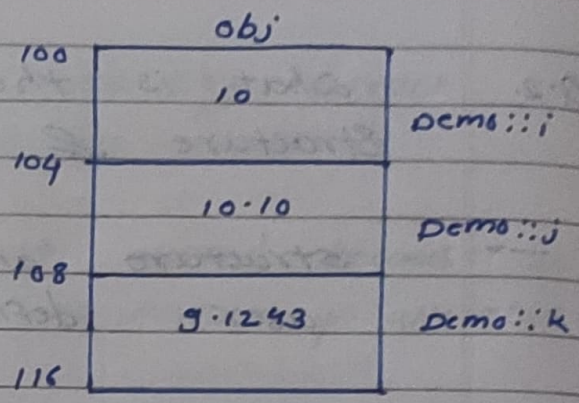
- But in case of Structure we can store all the values of all the members at a time.

Example -

1. Structure

```
struct Demo
{
    int i;
    float j;
    double k; };
```

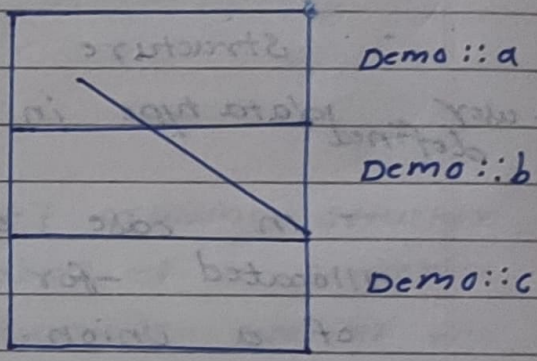
```
struct Demo obj;

obj.i = 10;
obj.j = 10.10;
obj.k = 9.1243;
```

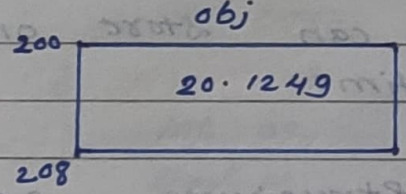


"16 bytes"

2. Union

```
union Demo
{
    float a;
    int b;
    double c;
} obj;

obj.c = 20.1249;
```





"8 bytes"

Q.3. Which type of data we can store in Structure ?

→ • We can store any of the data type in structure.

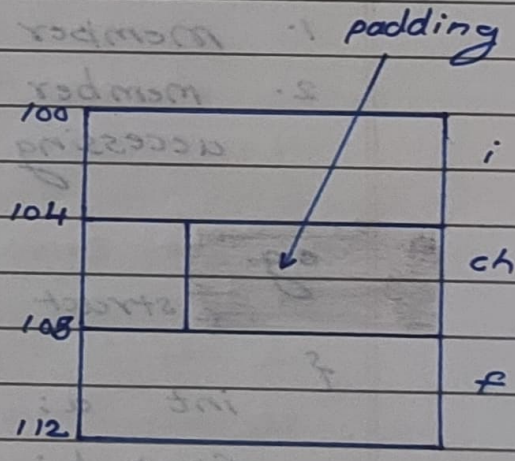
i.e 1. Primitive
2. derived
3. Userdefined

Q.4. What is meant by padding in memory allocation of structure.

→ Structure padding is defined as the process of adding one or more empty bytes between datatypes to allign data in memory.

for e.g. -

```
struct Demo
{
    int i ;
    char ch;
    float f;
};
```



- In above example we expected 12 bytes of memory ; but when we execute it we get 12 bytes.

- These 3 bytes are considered as padding.

- Padding is considered as allocated memory inside the object that we cannot access.

- It is considered as wastage of memory.

- To access element in faster way compiler assign some extra spaces that space is called as padding.

Q.5. How many ways we can initialize the member of structure.

—> - There are 3 ways to initialize members of structure:

1. member initialization list
2. member by member using direct accessing operator.

e.g.

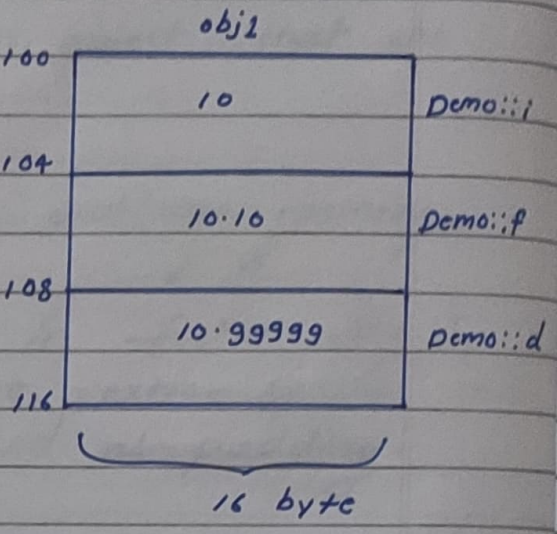
```
struct Demo
{
    int a;
    float b;
    char c;
} obj, obj1;
```

```
1. list
    obj = { 10, 1.1, 'A' };

2. obj.a = 10;
   obj.b = 1.1;
   obj.c = 'A';
```

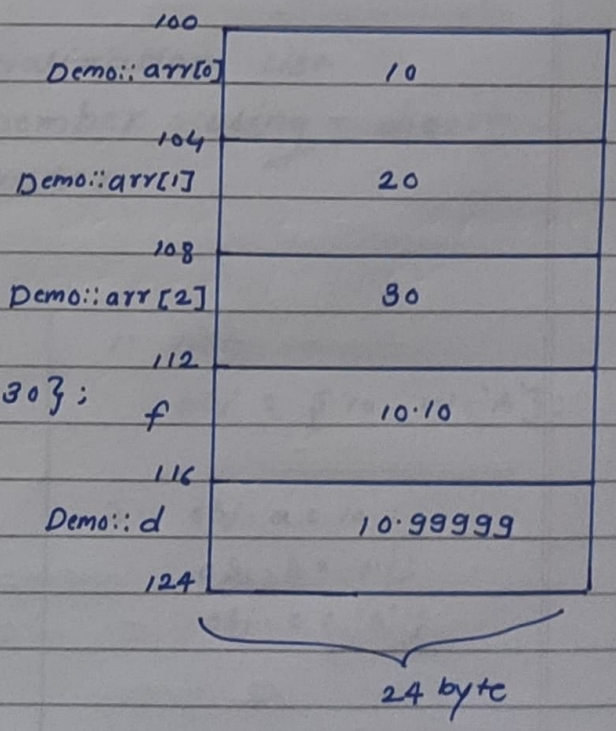
Q.6. what is array between array & structure.

Q.7. Draw the memory layout.

1.
```
struct Demo
{
    int i;
    float f;
    double d;
};

struct Demo obj1;

obj1.i = 10;
obj1.f = 10.10;
obj1.d = 10.99999;
```



2.
```
struct Demo
{
    int arr[3];
    float f;
    double d;
};

struct Demo obj;

obj.arr[3] = {10,20,30};
obj.f = 10.10;
obj.d = 10.99999;
```

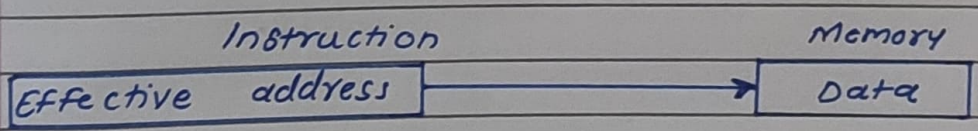


Q.8. Why we can not initialise members of structure at time of declaration?

--> • Structure members can not be initialized like othe variables inside the structure defination.
This is because when a structure is defined no memory is allocated to the structures data members at this point.
• Memory is not allocated when a structure variable is ~~de not~~ declared.
• Memory allocated to the structure when object of that structure created.

Q.9. What is difference between direct access and indirect access operator.

--> • Direct accessing :
In direct accessing mode, the address field in the instruction contains the effective address of the operand & no intermediate memory access is required.



• Indirect accessing :
The address field in the instruction contains the memory location

or register where the effective address of operant is present.

Q.10. Detect the problem.

—> 1. struct Demo
{
int i;
float f = 10.0; // cannot initialize variable while declaring structure.
double d;
};

2. struct Demo
{
int i;
float f;
double i;
};

// we cannot declare two variables with same identifier / name.