Assignment No-10

Q1] What is the concept of structure? explain with example.

→ Structure is a user-defined data type that allows grouping of different data types ~~that~~ under one name.
- Structures are particularly useful for representing real-world entities, like a student, employee, or product, that have multiple attributes.
- Structures are ~~dif~~ defined using the struct keyword.
- A structure can have multiple variables of different types.
- Members of a structure are accessed using the dot operator (.).

e.g.

```
#include <stdio.h>
// A structure
struct Student {
    int rollNumber;
    char name[50];
    float marks;
};
int main() {
// Declare and initialize a structure
struct Student student1 = {1, "John", 87.5};
// access and print structure variable
printf("Roll Number : %d\n", student1.rollNumber);
printf("Name : %s\n", student1.name);
printf("Marks : %f\n", student1.marks);
```

```
// modifying a member
student1.masks = 90.0;
printf("Updated Masks : %f\n", student1.masks);
return 0;
}
```

Output - Roll Number - 1
Name - John
Masks - 87.50
Updated masks - 90.00

Q2) Explain the concept of padding in memory allocation of structure.

→ Padding in memory allocation of structure is a concept in C where the compiler adds extra bytes of memory between the structure members to align them according to the architecture's requirements.
- This ensures efficient memory access and avoids performance penalties due to misaligned data.
- Each member is aligned to its size or the largest size supported by the architecture (eg. 4bytes or 8 bytes).
- The compiler adds padding between members to align each to its required boundary.
- Padding may also be added at the end of the structure to make its total size a multiple of the largest member's alignment.
- Padding increases the size of the structure.
- To minimize padding, arrange members in descending order of size.

Q3] Write the difference between structure and Union.

→ 1] Structure - In case of structure memory gets allocated to each an every member separately.
Union - In case of union there is no separate memory allocation for each member.

2] Structure - size is equal to or greater than total size of its member is allocate
Union - Size of ~~or~~ union is size of its largest member.

3] Structure - It can contain data for multiple member at same time.
Union - It can contain data only for one member at a same time.

4] Structure - Struct keyword is used to create structure.
Union - union keyword is used to create union.

Q4] Which ~~can~~ type of data can we store in structure? explain with program.

→ 1] Data Types like integers, float, characters

e.g.

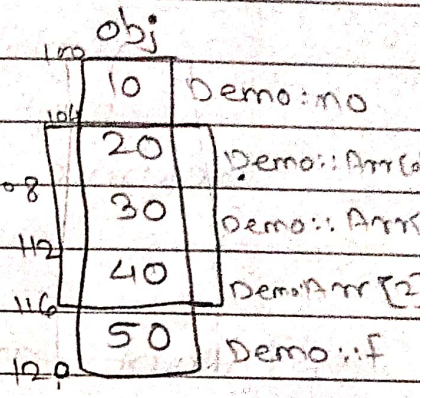
```
struct BasicData {
    int id;
    float salary;
    char grade;
};
```

2] We can create an array as a member of structure.

eg.

```
#include<stdio.h>
Struct Demo {
   int no;
   int Arr[3];  ← Array declaration
   float f;
};
int main()
{ struct Demo obj;
   obj.no = 10;
   obj.Arr[0] = 20;
   obj.Arr[1] = 30;
   obj.Arr[2] = 40;
   obj.f = 50.0;
return 0;
}
```

member by member initialization



3] Inside structure we can also stored pointer as a member.

→

```
#include<stdio.h>
Struct Demo {
   int *p;
   float *q;
   double d;
};
int main()
{ struct Demo obj1;
   int i = 10;
   float f = 90.90f;

   obj1.p = &i;
```

```
    obj1.q = &f;
    obj1.d = 90.9999;

    printf("%d\n", *(obj1.p));
    printf("%d\n", *(obj1.q));
    printf("%d\n", *(obj1.d));
return 0;
}
```

Q5] What are the different ways to initialize the members of structure.

→ 1] Member by member designated initialization list

eg.
```
Struct Demo
  {  int no;
     int x;
     float d;
  }
```

2] We can initialized the list by member initialization list

eg.

```
Struct Demo strobj3 = {2, 60.0f, 700, 700.5f};
```

3] We can create an array as a member of structure.

eg. -
```
Struct demo obj;
    obj.no = 10
    obj.Arr[0] = 20;
    obj.Arr[1] = 30;
    obj.Arr[2] = 40;
    obj.f = 50;
```

Q) Inside structure we can also stored pointer as a member.

e.g.

```
#include<stdio.h>
struct Demo
{
   int *p;
   float *q;
   double d;
};
int main()
{
   struct Demo obj1;
   int i = 10;
   float f = 90.90f;

   obj1.p = &i;
   obj1.q = &f;
   obj.d = 90.999;
printf("%d\n", *(obj1.p));
printf("%d\n", *(obj1.q));
printf("%d\n", *(obj1.d));
return 0;
}
```

Q6) Why we can't initialize members of structure at time of declaration?

→ In C structure member cannot be initialized at the time of their declaration with in the structure definition.

This limitation arises because a structure

Serves as a blueprint for creating multiple instances, each potentially requiring different initial value.
Allowing initialization within the structure definition would enforce the same initial values for all instances, reducing flexibility.

Q7) Write difference between structure and Array.

Array - It refers to a collection consisting of elements of homogeneous data type.
Structure - It refers to a collection consisting of elements of heterogeneous data type.

Array - Array usses [] subscript operator for element access
Structure - Structure uses . Dot operator for element access.

Array - Array internally considered as a pointer
Structure - structure is not a pointer.

Array - Array declaration is done simply using [] and not use any keyword.
Structure - structure declaration is done with the help of struct keyword.

Q8] What is the difference between direct access (.) & indirect access (->) operator explain with example and program.

→ Direct Operator -
When we have object of structure or union then we can access its value ~~using~~ directly using dot operator called as direct accessing operator.
(box हातात असणे)

Indirect Operator
When we have a pointer which points to the object of structure and Onion then access to a member then called indirect access operator.
(kichen मध्ये box असणे)

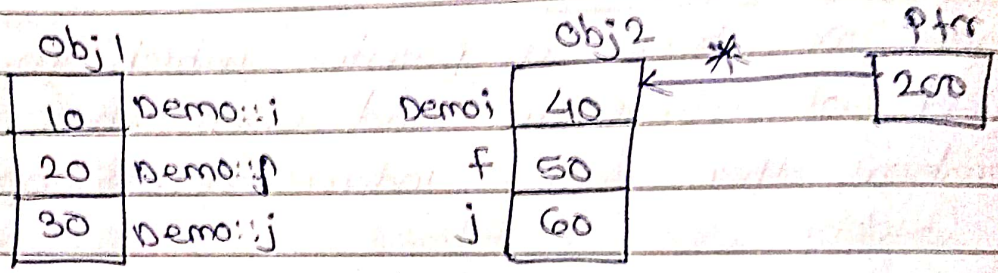
eg.

```
#include<stdio.h>
Struct Demo
{ int i;
  float f;
  int j;
};
int main()
{ struct Demo obj1;
  Struct Demo obj2;

  Struct Demo *ptr = NULL;
  ptr = & obj2;

  obj1.i = 10;
  obj1.f = 20
  obj1.j = 30
```

} Direct access

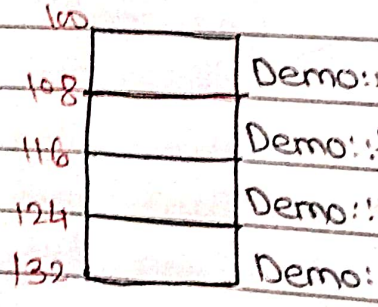
```
ptr -> i = 40;
ptr -> f = 50;
ptr -> j = 60;
```
} Indirect access

```
printf ("obj1.i %d\n", obj1.i);
printf (" obj2.i %d\n", obj2.i);
printf (" ptr->i %d\n", ptr->i);
return 0;
}
```



Q9] Draw the memory layout for the below Struc

→ Struct Demo
{ int i;
float f;
double d;
int j;
};



Q10] Find out the problem in the below syntax and correct it and draw the memory layout for beb structures.

→ { Struct Demo
{ int i;
float f = 10.5;
double d;
}

→ Given syntax is incorrect because At the time of creating/ defer declaration of structure we can't initialize its member with its value. Initialization is not allow because at that point there is no memory allocation for structure.

Correct Syntax -

```
struct Demo
{ int i;
  float f;
  double d;
};
int main()
```

// modifying a member
student1.marks = 90.0;
printf("Updated Marks : %f\n", student1.marks);
return 0;
}

Output - Roll Number - 1
Name - John
Marks - 87.50
Updated marks - 90.00

2) Explain the concept of padding in memory allocation of structure.

→ Padding in memory allocation of structure is a concept inc.
where the compiler adds extra bytes of memory between the structure members to align them according to the architecture's requirements
- This ensures efficient memory access and avoids performance penalties due to misaligned data.
- Each member is aligned to its size or the largest size supported by the architecture (eg. 4bytes or 8 bytes).
- The compiler adds padding between members to align each to its required boundary.
- Padding may also be added at the end of the structure to make its total size a multiple of the largest member's alignment.
- Padding increases the size of the structure.
To minimize padding, arrange members in descending order of size.