STRUCTURES:

Structure is a user defined data type that can store related information of different data types. The major difference between array and structure is array can store only information of same data type.

Declaration of Structure:

```
struct point
{
int x,y;
}p1;
struct point
{
int x,y;
};
int main(
{
struct point p1;
}
```

Initialization of Structure:

Structure members cannot be initialized with declaration

```
struct point
{
int x=0;        //compiler error
int y=0;        //compiler error
};
```

- Memory is allocated when variables are created struct members can be initialized using curly braces {}.

```
struct point
{
int x,y;
}
int main()
{
struct point p1={0,1}
}
```

The structure member are accessed using dot(.) operator

```
struct point
{
int x,y;
};
int main ()
{
struct point p1={0,1};
p1.x=20;
printf("x=%d, y=%d,p1.x,p1.y");
return 0;
}
```

Example:

```
#include<stdio.h>
#include<conio.h>
struct point
{
int x,y;
};
int main()
{
```

Designated Initialization:

It allows structure members to be initialized in any order.

Array of Structure:

Like other primitive data types we can create an array of structures.

```
struct point arr[10]
arr[0].x=10;
arr[0].y=20;
printf(%d%d",arr[0].x,arr[0].y);
return 0;
}
```

Structure using Pointer:

Like primitive data type we can have pointer to a structure member are accessed using arrow (->) operator.

```
#include<stdio.h>
#include<conio.h>
struct point
{
int main()
{
struct point*p1={1,2};
struct point*p2=&p1;
printf("%d%d",p2->x,p2->y);
return0;
}
}
```

Output 1 2