- What is structure
- Initializations and declarations
- Memory allocation functions
- Pointers with structures
- Array with structures
- User defined function with structures
- Nested structures
- Introduction to union
- Difference between Structure & Union
- Enumerated Type

- **WHY STRUCTURE ? :**

In C, there are cases where we need to store multiple attributes of an entity.

It is not necessary that an entity has all the information of same type.

It can have different attributes of different data types.

For example, an entity Student may have its name (string), roll number (int), marks (float).

To store such type of information regarding an entity student, we have the following approaches:

Construct individual arrays for storing names, roll numbers, and marks.

Use a special data structure to store the collection of different data types.

Example of Array :

```
#include<stdio.h>
#include<conio.h>
void main()  {
        char name[3][10];
        int a[3],b[3],i;
        clrscr();
        for(i=0;i<3;i++)
        {
                printf("Enter Your Name :\n");
                scanf("%s",&name[i]);
                scanf("%d",&a[i]);
                scanf("%d",&b[i]);
        }
```

```
        for(i=0;i<3;i++)
        {
                printf("Your Name :\n");
                printf("%s",name[i]);
                printf("%d",a[i]);
                printf("%d",b[i]);
        }
        getch();
}
```

❖ **What is a structure?**

A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.

Arrays allow to define type of variables that can hold several data items of the same kind. Similarly **structure** is another user defined data type available in C that allows to combine data items of different kinds.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −

- Title
- Author
- Subject
- Book ID

- **How to create a structure?**

'struct' keyword is used to create a structure. Following is an example.

Syntax :

```
struct <tag_name/Structure name>
{
        ……
        ……
        Declaration of variable
        ……
        ……
        ……
};
```

```
struct address
{
    char name[50];
    char street[100];
    char city[50];
    char state[20];
    int pin;
};
```

- **How to declare structure variables/object variable?**
    1. **Before semicolon at structure terminates .**
    2. **At global declaration section (Global Scope) .**
    3. **Inside the main function (Local Scope) .**

A structure variable can either be declared with structure declaration or as a separate declaration like basic types.

Syntax :

struct <tag_name> <object_name>,[obj2,3,4….];

```
// A variable declaration like basic data types
structPoint
{
    intx, y;
}p1;  //before semicolon
Struct Point p2 //global declaration

void main()
{
    structPoint p3;   // Local Variable -> The variable p3 is declared like a normal variable
    struct Point p4={10,20};
}
```

Here, p1,p2,p3,p4 are structure variable or object variable

- **How to initialize structure members?**

Structure members **cannot be** initialized with declaration. For example the following C program fails incompilation.

```
struct Point
{
    intx = 0;  // COMPILER ERROR: cannot initialize members here
    inty = 0;  // COMPILER ERROR: cannot initialize members here
};
```

Structure members can be initialized using curly braces '{ }'. For example, following is a valid initialization

```
struct Point
{
    intx, y;
};
```

```
void main()
  {
      structPoint p1 = {0, 1};
      // A valid initialization. member x gets value 0 and y
      // gets value 1.   The order of declaration is followed.
  }
```

- **How to access structure elements?**

Structure members are accessed using dot (.) operator.

Syntax :

Object_name.variable_name;

Example :

```
#include<stdio.h>

struct Point
{
    intx, y;
};
void main()
{
    structPoint p1 = {0, 1};      // local object variable
    printf("x = %d, y = %d", p1.x, p1.y);
    // Accesing members of point p1
    p1.x =20;
    printf("\nx = %d, y = %d", p1.x, p1.y);
    getch();
}
```

Output:

x=0, y=1
x = 20, y = 1

❖ Array with Structure :

Like other primitive data types, we can create an array of structures.
We can implement array concept in two different ways.

1. **Array Within Structure**
2. **Array Of Structure**

1. Array Within Structure

We can declare array of any type of array like 1D,2D and multi dimensional array inside structure **as a member variable**.

Example :

Write a c program to get name of student and 5 subject marks using structure.

```
#include <stdio.h>
struct student
{
        char name[50];   //array within structure
        int mark[5];       //array within structure
};
int main()
{
        struct student s1;
        int i;
        clrscr();
        printf("Enter name: ");
        scanf("%[^\n]%c", s1.name);
        for(i=0;i<5;i++)
        {
                printf("\nEnter Student Mark :\n");
                scanf("%d",&s1.mark[i]);
        }
        printf("Student Name : %s",s1.name);
        for(i=0;i<5;i++)
        {
                printf("\nStudent Mark is : ");
                printf("%d",s1.mark[i]);
        }
        getch();
        return 0;
}
```

2. Array Of Structure

We can declare array of structure similar like ordinary array of integer values.

Whenever you want to declare many structure variable in program then declare individual many variable is difficult to manage.

We can create an array of structure variable for better management.

Example :

```
#include<stdio.h>
struct Point
{
    intx, y;
};
void main()
{
    // Create an array of structures
    structPoint arr[10];

    // Access array members
    arr[0].x = 10;
    arr[0].y = 20;
    //arr[1].x=34;
    //arr[1].y=56;

    printf("%d %d", arr[0].x, arr[0].y);
    printf("\n%d %d", arr[1].x, arr[1].y);
    getch();
}
```

Output:

10 20
34 56

❖ What is a structure pointer?

We can also use the pointer with structure like other inbuilt data type which hold the address of other variable.

Same way structure pointer object points to address of other structure variable.

If we have a pointer to structure, members areaccessed using arrow (->) operator.

Syntax :

Pointer_object->variable_name;

Example:

```
#include<stdio.h>
struct Point
{
    intx, y;
};
void main()
{
    structPoint p1 = {1, 2};
```

```
    // p2 is a pointer to structure p1
    structPoint *p2 = &p1;

    // Accessing structure members using structure pointer
    printf("%d %d", p2->x, p2->y);
    getch();
}
```

Output:

1 2

❖ Udf with structures

We can also use structure with user defined function. Same as normal variable, we can pass member of structure in user defined function as an arguments.

There are four technique to passed the structure member to UDF.

1. The individual member of structure passed to UDF using call by value.
2. The individual member of structure passed to UDF using call by reference.
3. The entire structure passed to UDF using call by value.
4. The entire structure passed to UDF using call by reference.

- **This Example of entire structure passed to UDF using call by value.**

```
#include <stdio.h>
struct student
 {
            char name[50];
            int age;
};

     // function prototype
 void display(struct  student );

void main()
{
               struct student s1;
               printf("Enter name: ");
               scanf("%[^\n]%c", s1.name);
               printf("Enter age: ");
               scanf("%d", &s1.age);
               display(s1);            // passing struct as an argument
               getch();
}

void display(struct  student  s)
{
           printf("\nDisplaying information\n");
           printf("Name: %s", s.name);
           printf("\nAge: %d", s.age);
}
```

Output :

Enter name: Bond
Enter age: 13

Displaying informationName:
Bond
Age: 13

❖ Nested Structure in C

C provides us the feature of nesting one structure within another structure by using which, complex data types are created.
For example, we may need to store the address of an entity employee in a structure.
The attribute address may also have the subparts as street number, city, state, and pin code. Hence, to store the address of the employee, we need to store the address of the employee into a separate structure and nested the structure address into the structure employee.

```
#include<stdio.h>
struct address
{
        char city[20];
        int pin;
        char phone[14];
};
struct employee
{
        char name[20];
        struct address add;
};
void main()
{
        struct employee emp;
        printf("Enter employee information :\n");
        scanf("%s %s %d %s", &emp.name, &emp.add.city, &emp.add.pin, &emp.add.phone);
        printf("Printing the employee information.        \n");
        printf("name: %s\nCity: %s\nPincode: %d\nPhone: %s", emp.name, emp.add.city,
        emp.a dd.pin, emp.add.phone);
}
```

Output

Enter employee information?
Arun
Delhi 110001
1234567890
Printing the employee information....
name: Arun
City: Delhi
Pincode: 110001
Phone: 1234567

❖ Union in C

Union can be defined as a user-defined data type which is a collection of different variables of different data types in the same memory location. The union can also be defined as many members, but only one member can contain a value at a particular point in time.

The basic difference between structure and union is it's memory allocation mechanism.
Structure members are allocated in sequential memory and the union allocates memory of it's largest member based.

- **How to define a union?**

We use the unionkeyword to define unions. Here's an example:

Syntax :

```
union <tag_name/Structure name>
{
        ……..……
        ……..……
        Declaration of variable
        ……..……
        ……..……
        ……..……
};
```

Example :

```
union car
{
        char name[50];
        int price;
};
```

- **Create union variables**

When a union is defined, it creates a user-defined type. However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.

Here's how we create union variables.

```
union car
{
        char name[50];
        int price;
};
void main()
{
```

```
        union car car1, car2, *car3;
}
```

For example in the following C program, both x and y share the same location. If we change x, we can see the changes being reflected in y.

Example :

```
#include <stdio.h>

//Declation of union same as structure

union test {
        int x, y;
};
void main()
{
        union test t;

        t.x = 2; // t.y also gets value 2
        printf("After making x = 2:\n x = %d, y = %d\n\n", t.x, t.y);

        t.y = 10; // t.x is also updated to 10
        printf("After making y = 10:\n x = %d, y = %d\n\n",
        t.x, t.y);
}
```

Output :

```
After making x = 2  :
x = 2, y = 2

After making y = 10:
x = 10, y = 10
```

❖ Structure vs. Union

There are some common points between structure and union are :

1. Both are user defied data type.
2. Both have same definitions and syntax.
3. Both have same way to declare variable/object.
4. Both have same technique for initialization of variables.

There are some different points between structure and union are :

Points	Structure	Union
Keyword	The structure is declared using **struct** keyword.	The structure is declared using **union** keyword.
Storage	In structure, Each members are hold **separately** storage location.	In union, Each members are hold **same** storage location.
Size	In structure, Each member has its own storage location on memory. So, there is not required maximum size location.	In union, Large size member occupies in memory and all member use this location. So, at a time one member can accessed.

Syntax	struct tagname
	{
	Data_type variable_name;
	Data_type variable_name;
	Data_type variable_name;
	
	};
		union tagname
		{
		Data_type variable_name;
		Data_type variable_name;
		Data_type variable_name;
		
		};

Example	struct tagname
	{
	int a;
	float b;
	};
	
	In this example size of structure is 2 bytes for variable a and 4 bytes for variable b.
	So, total size of structure is 6 bytes.
		union tagname
		{
		int a;
		float b;
		};
		
		In this example size of structure is 2 bytes for variable a and 4 bytes for variable b.
		So, total size of structure is 4 bytes because it occupies the large size of variable.

❖Enumeration In C Language

Enumeration data type used to symbolize a list of integral constants with valid meaningful names, so that, it makes the program easy to read and modify.
Enumeration has advantage of generating the values automatically for given list of names.
Keyword enum is used to define enumerated data type.

Syntax :

```
enum type_name
{
        value1, value2... valueN
};
```

Here, type_name is the name of enumerated data type or tag. And value1, value2 ...valueN are values of type type_name.
By default, value1 will be equal to 0, value2 will be 1 and so on but, the programmer can change the default value.

Example:

```
#include <stdio.h>

enum week{ sunday, monday, tuesday, wednesday, thursday, friday, saturday};

void main()

{

        enum week today;

         today=wednesday;

        printf("%d th day",today+1);

}
```

Output :
4th day