Structures in C

Primitive type:

- It is a pre-defined data type.
- It can store only 1 value at a time.
- For example, int a = 10 ;

Array:

- It is a derived data type
- It can store more than 1 value but of same type.
- int a[5] = {10,20,30,40,50};

Structure:

- It is user defined data type.
- It can store more than 1 element of different data types.
- Structure is used to store Emp details, student details, account details.
- 'struct' is a keyword.
- 'struct' is used to define structures in C application.

Syntax:

```
        struct  identity
        {
                datatype ele-1;
                datatype ele-2;
                .....
                datatype  ele-n;
        };
```

Example:

```
        struct Emp
        {
                int no ;
                char name[20];
                float salary;
        };
```

- Above definition is represents Data type.
- We declare variables to allocate memory to that structure.
    - Example: struct Emp e1, e2 , e3 ;
- Structure variable stores base address of memory block
- We access the elements of structure using dot(.) operator.

Size of structure:

- Structure is a collection of elements.
- The size is equals to sum of sizes of individual elements.
- sizeof() function can be used to specify the size.

We can find the size either by using variable or data type

```
#include<stdio.h>
```

```
int main()
{
        int x ;
        printf("int size : %d \n", sizeof(x));
        printf("int size : %d \n", sizeof(int));
        return 0;
}
```

Finding Emp structure size:

```
#include<stdio.h>
struct Emp
{
        int no;
        char name[20];
        float salary;
};
int main()
{
        struct Emp x ;
        printf("Emp size : %d \n", sizeof(x));
        printf("Emp size : %d \n", sizeof(struct Emp));
        return 0;
}
```

- Structure is a data type but user-defined.
- Structure definition either Local or Global.

Local structure:

- Define structure inside the function.
- It can be accessed only from the same function in which it has defined.

```
#include<stdio.h>
void check();
int main()
{
        struct Local
        {
                int a,b;
        };
        struct Local x ;
        return 0;
}
void check()
{
        struct Local x ;   -> Error : unknown data type of 'x'
}
```

Global structure:

- Define structure outside to all functions.
- Global structure can be accessed through out the application.

```
#include<stdio.h>
void check();
struct Global
{
	int a,b;
};
int main()
{
	struct Global x ;
	return 0;
}
void check()
{
	struct Global x ;
}
```

- We can initialize the structure directly using assignment operator.
- How we assign values to array, in the same way we can assign values to structure.
- While assigning, we need to follow the order of structure elements.

```
#include<stdio.h>
struct Emp
{
	int no;
	char name[20];
	float salary;
};
int main()
{
	struct Emp x = {101, "amar", 50000};
	printf("No : %d \n", x.no);
	printf("Name : %s \n", x.name);
	printf("Salary : %f \n", x.salary);
	return 0;
}
```

Passing structure as an argument to function:

- Structure variable holds address of memory block.
- Like arrays and strings, we can pass structure as argument to function.
- To collect the address, we need to define same type of structure variable as an argument inside function.

```
#include<stdio.h>
struct Emp
{
	int no;
```

```
        char name[20];
        float salary;
};
void display(struct Emp);
int main()
{
        struct Emp x;
        printf("Enter Emp details : \n");
        scanf("%d%s%f", &x.no , x.name , &x.salary);
        display(x);
        return 0;
}
void display(struct Emp y)
{
        printf("No : %d \n", y.no);
        printf("Name : %s \n", y.name);
        printf("Salary : %f \n", y.salary);
}
```

Function returning structure:

```
#include<stdio.h>
struct Emp
{
        int no;
        char name[20];
        float salary;
};
struct Emp read();
int main()
{
        struct Emp y;
        y = read();
        printf("Name : %s\n", y.name);
        return 0;
}
struct Emp read()
{
        struct Emp x;
        printf("Enter Emp details : \n");
        scanf("%d%s%f", &x.no, x.name, &x.salary);
        return x;
}
```

Array of structures:

- Using structure variable, we can store only 1 record.
- To store multiple records, we need to declare an array of structure type
    - struct Emp e1 ; -> store 1 record
    - struct Emp e1, e2, e3 ; -> store 3 records

    - struct Emp arr[100] ; --> can store upto 100 records.
- Array elements can process using loops.
- We can read and process multiple records into array using loops only.

```
#include<stdio.h>
struct Emp
{
	int no;
	char name[20];
	float salary;
};
int main()
{
	struct Emp arr[3];
	int i;

	printf("Enter 3 Emp details : \n");
	for(i=0 ; i<3 ; i++)
	{
		printf("Enter Emp-%d details :\n", i+1);
		scanf("%d%s%f", &arr[i].no, arr[i].name, &arr[i].salary);
	}

	printf("Emp records :\n");
	for(i=0 ; i<3 ; i++)
	{
		printf("Emp-%d details :\n", i+1);
		printf("%d\n%s\n%f\n\n", arr[i].no, arr[i].name, arr[i].salary);
	}
	return 0;
}
```

Arrays in structure:

- We can declare array variable inside structure.
- Array elements we can access using loops.

Reading student details:

```
#include<stdio.h>
struct Student
{
	int no;
	char name[20];
	int marks[5];
};
int main()
{
	struct Student s;
	int i;
```

```
    printf("Enter student details :\n");
    scanf("%d%s", &s.no, s.name);
    for(i=0 ; i<5 ; i++)
    {
        scanf("%d", &s.marks[i]);
    }

    printf("Student details :\n");
    printf("%d\t%s\t", s.no, s.name);
    for(i=0 ; i<5 ; i++)
    {
        printf("%d\t", s.marks[i]);
    }
    return 0;
}
```

Reading more than 1 student details:

```
#include<stdio.h>
struct Student
{
    int sno ;
    char sname[20];
    int smarks[4];
};
void main()
{
    struct Student s[3];
    int i,j ;

    printf("Enter 3 student details :\n");
    for(i=0 ; i<3 ; i++)
    {
        printf("Enter student-%d details : \n", i+1);
        scanf("%d%s",&s[i].sno, s[i].sname);
        for(j=0 ; j<4 ; j++)
        {
            scanf("%d", &s[i].smarks[j]);
        }
    }

    printf("Student details : ");
    for(i=0 ; i<3 ; i++)
    {
        printf("Student-%d details : \n", i+1);
        printf("%d\t%s\t",s[i].sno, s[i].sname);
        for(j=0 ; j<4 ; j++)
        {
            printf("%d\t", s[i].smarks[j]);
        }
        printf("\n");
        }
}
```

Copying structure elements:

- Array elements need to copy using loops
- Structure element don’t have index.
- We can copy all the structure elements just by using assignment operator.

```
#include<stdio.h>
struct emp
{
        int eno;
        char ename[20];
        float esal;
};
void main()
{
        struct emp e1={1001,"amar",50000};
        struct emp e2,e3;

        //element by element copying
        e2.eno = e1.eno;
        strcpy(e2.ename, e1.ename);
        e2.esal = e1.esal;

        //all at once
        e3 = e2; //should be of same type

        printf("%d\t%s\t%f\n",e1.eno, e1.ename, e1.esal);
        printf("%d\t%s\t%f\n",e2.eno, e2.ename, e2.esal);
        printf("%d\t%s\t%f\n",e3.eno, e3.ename, e3.esal);
}
```

Nested structures:

- Define the structure inside another structure.
- Using nested structures we can define complex data types.
- Outer structure elements we can refer directly.
- Inner structure elements must be accessed with the help of Outer structure elements.

```
#include<stdio.h>
struct emp
{
        int eno;
        float esal;
};
struct employee
{
        struct emp e;
        char ename[20];
};
void main()
{
	struct employee e1;
	printf("Enter emp details\n");
	scanf("%d%s%f",&e1.e.eno, e1.ename, &e1.e.esal);
}
```