

STRUCTURES

Structure is a collection of elements of different data types.

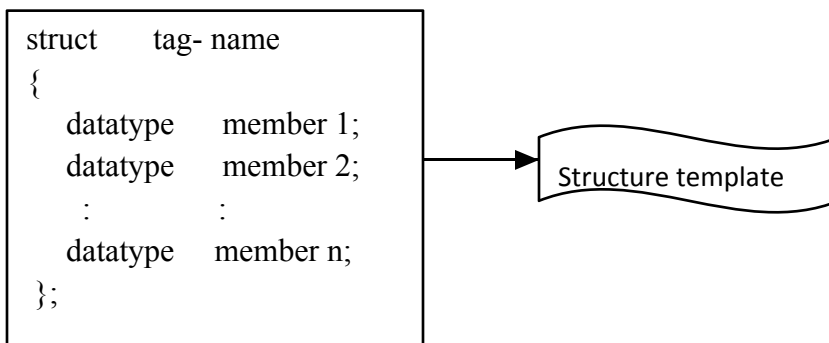
Keyword of structure is : **struct**

Note : structures, arrays ,pointers, functions are called as DERIVED DATATYPES since, all these will be using fundamental datatypes like int, char float etc.,,

Uses of structures:

- A structure is convenient tool for handling a group of logically related data items.
- The concept of structure is analogous to that of a “record” in many other languages.
- Structures help to organize complex data in a more meaningful way that we often need to use in our program design.

Syntax of a structure:



Example:

```
struct student
{
    char name[20];
    int rollno;
    int marks;
    float average;
};
```

In the above example, **student** is called **structure-tag**.

name[20], rollno, marks, average are called as **structure elements** or **members**.

struct student is like a datatype. Now variables can be declared of this type and access the members of structure.

While declaring structure :

- Structure template should be terminated with a semi-colon.
- Tag-name is used to declare variables for the structure in later part of the program.

Declaring variables for the structure :

```
struct student
{
    char name[20];
    int rollno;
    int marks;
    float average;
} s1, s2 ;
```

s1 ,s2 are variables. Now input can be given as :

student-1 :

```
s1.name= " anand ";
s1.rollno= 102 ;
s1.marks= 555 ;
s1.average=68.4;
```

student-2 :

```
s2.name= " sanjay "
s2.rollno= 134 ;
s2.marks= 756 ;
s2.average= 81.3;
```

operators used with structures :

1. Dot or Member operator (.) – used to access members of structure
2. Arrow operator (->) - used to access the value stored at address of variable

Why to write tag-name for structure ?

Tag-name is optional for a structure.

```
struct
{
    char name[20];
    int rollno;
    int marks;
    float average;
} s1, s2 ;
```

We cannot declare variables in later part of program and can't access the structure members.

```
struct student
{
    char name[20];
    int rollno;
    int marks;
```

Memory of a structure :

When ever we declare variables for structure then only memory is allocated for it.

```
struct student
{
    char name[20];
    int rollno;
    int marks;
    float average;
};
```

No memory allocation is made.

```
struct student
{
    char name[20];
    int rollno;
    int marks;
    float average;
};
void main()
{
    struct student s1,s2 ;
}
```

(s1)
 $20 + 2 + 2 + 4 = 28$ bytes
(s2)
 $20 + 2 + 2 + 4 = 28$ bytes

Note :

- Memory allocated need not to be continuous for variables s1 and s2.
- Structure can be declared before (global) or after main() (local)

Assigning values to structures :

Method-1 :

```
struct student
{
    char name[20];
    int rollno;
    int marks;
    float average;
};
void main()
{
    struct student s1 ;
    strcpy( s1.name, " amith ");
    s1.rollno= 98;
    s1.marks=678;
    s1.avg= 65.78;
}
```

Method-2 :

```
struct student
{
    char name[20];
    int rollno;
    int marks;
    float average;
};
void main()
{
    struct student s1={"amith", 98, 678, 65.78 };
}
```

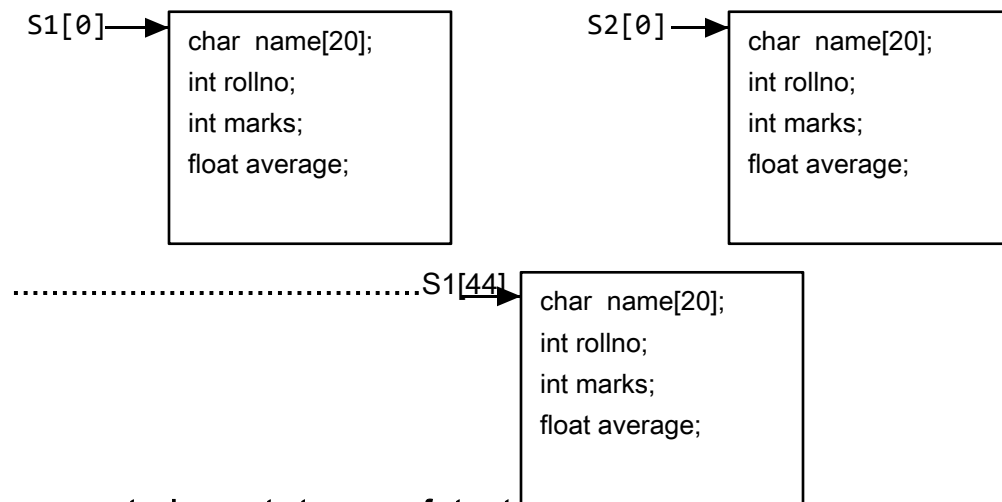
Method-3 :

```
struct student
{
    char name[20];
    int rollno;
    int marks;
    float average;
};
void main()
{
    struct student s1;
    scanf("%s %d %d %f", s1.name, &s1.rollno, &s1.marks, &s1.average);
}
```

For example, `struct student s[60];` defines, an array called that consists of 60 elements, each element is defined to be of the type `struct student`.

```
struct student
{
    char name[20];
    int rollno;
    int marks;
    float average;
};
void main()
{
    struct student s1[45];
}
```

Now array of structures can be represented as :



program to demonstrate array of structures

```
#include<stdio.h>

struct student
{
    char name[20];
    int rollno;
    float per;
};

int main( )
```

```
{
struct student s[60];
int i;
printf("enter the student details\n");
for(i=1;i<=60;i++)
{
scanf("%s%d%f",s[i].name,&s[i].rollno,&s[i].per);
}

printf("student details are\n");
for(i=1;i<=60;i++)
{
printf("%s \t %d\t %f\n",s[i].name,s[i].rollno,s[i].per);
}
}
```

Structures and functions

In functions:

- we pass values as arguments to function.
- we pass arrays as arguments to function.

Similarly, structures can also be passed as arguments to functions.

It is done in 3 ways:-

1. Passing individual elements of structure to a function
2. Passing entire structure to function
3. Passing address of structure to function

```
/* Passing individual elements of structure to a function */
```

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

```
struct student  
{  
char name[20];  
int rollno;  
float per;  
};
```

```
void display(char [],int, float);
```

```
int main()
```

```
{  
struct student s;  
printf("enter name rollno and per\n");  
scanf("%s%d%f",s.name,&s.rollno,&s.per);  
display(s.name,s.rollno,s.per);  
print("%s %d %f ",s.name,s.rollno,s.per);  
}
```

output:

```
enter name rollno and per  
xyz 10 80.2  
abc 12 90.2  
abc 10 80.2
```

```
void display(char name[],int rollno,float per)
```

```
{  
strcpy(name,"abc");  
rollno=12;  
per=90.2;  
printf("%s %d %f ",name,rollno,per);  
}
```

In the above program

- We are passing members to structure individually.
- array name it self acts as a base pointer. Modification done to array in display() or string will effect to main() also. (refer the output)
- When size of structure is large that is, if structure has more number of variables then this method is not effective.

```
/* Passing entire structure to function */
```

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

```
struct student
```

```
{  
char name[20];  
int rollno;
```

```

float per;
};

int main()
{
struct student s;
void display(struct student);
printf("enter name rollno and per\n");
scanf("%s%d%f",s.name,&s.rollno,&s.per);
printf("before function call the structure data\n");
print("%s %d %f ",s.name,s.rollno,s.per);
display(s);
printf("after function call the structure data\n");
print("%s %d %f ",s.name,s.rollno,s.per);
}

void display(struct student s)
{
strcpy(s.name,"abc");
s.rollno=12;
s.per=90.2;
printf("%s %d %f ",s.name,s.rollno,s.per);
}

```

output:

```

enter name rollno and per
xyz 10 80.2
before function call the structure
data
xyz 10 80.2
abc 12 90.2
after function call the structure
data
abc 10 80.2

```

/* Passing address of structure to function */

```

struct student
{
char name[20];
int rollno;
float per;
};
int main()
{
struct student s;
void display(struct student * );
printf("enter name rollno and per\n");
scanf("%s %d %f",s.name,&s.rollno,&s.per);
}

```



```

printf("before function call the structure data\n");
print("%s %d %f ",s.name,s.rollno,s.per);
display(&s);
printf("after function call the structure data\n");
print("%s\t%d\t%f\n",s.name,s.rollno,s.per);
}
void display(struct student *p)
{
strcpy(p->name,"abc");
p->rollno=12;
p->per=90.2;
printf("%s %d %f ",p->name,p->rollno,p->per);
}

```

output:

```

enter name rollno and per
xyz 10 80.2
before function call the structure data
xyz 10 80.2
abc 12 90.2
after function call the structure data
abc 12 90.2

```

Self referential structure (NESTED STRUCTURES):

A Structure inside another structure is called as nested structure.
A structure referring to itself is called as Self Referential Structure.

```

int main()
{
    struc t dob
    {
        int d, m, y;
    };
    struct student
    {

```

```

        char name[20];
        struct dob b; // creating variable for "dob" structure
    };
    struct student a;
    printf("enter name and date of birth:");
    scanf("%s %d %d %d",s.n, &s.b.d, &s.b.m, &s.b.y);
    printf("student name =%s\n",s.n);
    printf("\ndob=%d-%d-%d", s.b.d, s.b.m, s.b.y);
}

```

Output:

```

enter name and date of birth:
RAM 5 7 94
student name =RAM
dob = 5-7-94

```

C gives us a concept of copying the details of one structure variable to another structure variable. This can be done in 2 ways

- 1. Element by element**
- 2. Entire structure in single step**

Ex: WAP for copying structure into another structure

```

int main()
struct a
{
int no; char n[20]; float s;
} e2, e3;

```

```
struct a e1={a,"sam",7500}; //initialisation
```

```
//First method
```

```
e2.no=e1.no  
strcpy(e2.n, e1.n);  
e2.s=e1.s;
```

```
//Second method
```

```
e3=e1;
```

```
printf("first emp \n");  
printf("no=%d, Name=%s, Sal=%f\n", e1.no, e1.n, e1.s);  
printf("second emp \n");  
printf("no=%d, Name=%s, Sal=%f\n", e2.no, e2.n, e2.s);  
printf("third emp \n");  
printf("No=%d, Name=%s, Sal=%f",e3.no, e3.n, e3.s);  
}
```

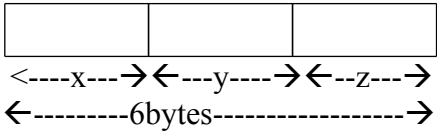
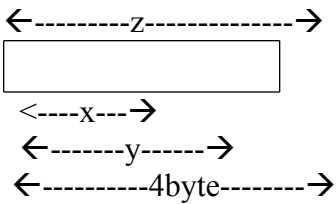
UNION

Union is another data type with two or more members, similar to structure.

General form of union:

```
union  union_name  
{  
data type  member 1;  
data type  member 2;  
  :      :  
data type  member n;  
} var1, var2;
```

Difference between structure and union:

Structures	Unions
<p>1. Structure is a collection of elements of different data types.</p> <p>2. Syntax: Struct structure name { Datatype member 1; Data type member 2; : Data type member n; }var1,var2;</p> <p>3. Ex: Struct abc { Char x; Int y; Float z; }s;</p>  <p>4. Each member at structure will have its own memory space.</p> <p>5. We can access all the members of a structure at a time.</p> <p>6. Structure occupies more memory compare to union.</p>	<p>1. Union is a collection of elements of different data types.</p> <p>2. Syntax: Union union name { Datatype member 1; Data type member 2; : Data type member n; }var1,var2;</p> <p>3. Ex: Union abc { Char x; Int y; Float z; }s;</p>  <p>4. All members of union shares the memory space.(allocates memory for large enough to hold the largest member)</p> <p>5. Only one member of union can be accessed at a time.</p> <p>6. Union saves memory space.</p>

Union follows the same syntax as structures. But the major difference between structure and union is for structures the compiler allocated memory for all members in contiguous memory allocation where as in unions it allocate memory of size large enough to hold the largest variable

type in the union.