STRUCTURE AND UNION

Introduction to Structure

- Structure is a user-defined datatype in C language which allows us to combine data of different types together.
- It is somewhat similar to an array, but an array holds data of similar type only. But structure on the other hand, can store data of any type, which is practical more useful.
- In structure, data is stored in form of records.
- For eg: If I have to write a program to store Student information, which will have Student's name, age, marks, branch, permanent address, father's name etc, which included string values, integer values, float values etc. how can I use arrays for this problem, I will require something which can hold data of different types together.

Defining a Structure

struct keyword is used to define a structure.

Syntax:

```
struct structure_name
{
    data_type member_variable1;
    data_type member_variable2;
    .   .   .
    .   .   .
};
```

Note: The closing curly brace in the structure type declaration must be followed by a semicolon (;).

Example of Structure:

```
struct student
{
    char name[25];
    int age;
    char branch[15];
    float marks;
};
```

Here, *struct Student* declares a structure to hold the details of a student which consists of 4 data fields, namely name, age, branch, and marks. These fields are called structure elements or members.

Each member can have different datatype, like in this case, name is an array of char type and age is of int type etc. Student is the name of the structure and is called as the structure tag.

Declaring Structure Variables

Once structure_name or structure_tag is declared as new data type, then variables of that type can be declared.

i. Declaring structure variables separately:

```
struct student
{
        char name[25];
        int age;
        char branch[15];
        float marks;
};
struct student s1, s2;                    //declaring variables of structure student
```

ii. Declaring structure variables with structure definition:

```
struct student
{
        char name[25];
        int age;
        char branch[15];
        float marks;
}s1, s2;
```

Here, s1 and s2 are variables of structure student.

Structure Initialization

C does not allow the initialization of individual structure members within its definition.

Syntax:

```
struct structure_name structure_variable = {value1, value2, value3,…};
```

Example:

```
        struct student
```

```
{
        char name[25];
        int age;
        char branch[15];
        float marks;
}s1, s2;
struct student s1 = {“Saugat”, 14, “Computer”, 80.0};
```

This line is equivalent to:

```
struct student s1;
s1.name = “Shyam”;
s1.age = 14;
s1.branch = “Computer”;
s1.marks = 80.0;
```

Accessing member of structure

A structure member can be accessed by using a dot(.) operator also called period or member access operator.

Syntax:

```
structure_variable.member
structure_variable.member.submember          //in case of nested structure
```

Example of structure:

```
#include<stdio.h>
#include<string.h>
struct student
{
        char name[20];
        int roll;
        float marks;
        char remark;                    //P for pass and F for fail
};
int main(){
        struct student s;
        printf(“Enter name:\t”);
        gets(s.name);
```

```
        printf("\nEnter roll no:\t");
        scanf("%d", &s.roll);
        printf("\nEnter marks:");
        scanf("%f", &s.marks);
        printf("\nEnter remark p for pass or f for fail:\t ");
        s.remark = getche();
        printf("\n\nStudent Name\t\t\tRoll\tMarks\tRemarks");
        printf("\n-------------------------------------------------------\n");
        printf("%s\t\t\t%d\t%.2f\t%c", s.name, s.roll, s.marks, s.remark);
        return 0;
}
```

Difference between array and structure

Array	Structure
1. An array is a collection of related data elements of the same type.	1. Structure can have elements of different types.
2. An array is a derived data type.	2. A structure is a programmerdefined data type.
4. Array allocates static memory and uses index/subscript for accessing elements of the array.	4. Structures allocate dynamic memory and uses (.) operator for accessing the member of a structure.
5. An array is a pointer to the first element of it.	5. Structure is not a pointer.
6. Element access takes relatively less time.	6. Property access takes relatively large time.

Introduction to Union

- Unions are similar to structure. Its syntax and use is similar to structure.
- It also contains members whose individual data types may differ form one another.
- All members within union share the same storage area of computer memory thus, one variable can reside into memory at a time.

Defining a Union

union keyword is used to define a union.

Syntax:

```
union union_name
{
```

```
        data_type member_variable1;
        data_type member_variable2;
        .       .       .
        .       .       .
    };
```

Declaring Union Variable

```
union student
{
    int roll;
    float marks;
};
union student st;
```

Union Initialization

```
union student
{
    int roll;
    float marks;
};
union student st;
st.roll = 13;
```

Accessing member of union

A union member can be accessed by using a dot(.) operator also called period or member access operator.

Syntax:

```
    union_variable.member
```

Example1:

```
#include<stdio.h>
union student
{
    int roll;
```

```
        float marks;
};
int main(){
        union student st;
        st.roll = 13;
        printf("\nRoll=%d", st.roll);
        st.marks = 65.0;
        printf("\nMarks=%.2f", st.marks);
        return 0;
}
```

Output

Roll=13

Marks=65.00

Example1:

```
#include<stdio.h>
union student
{
        int roll;
        float marks;
};
int main(){
        union student st;
        st.marks = 65.0;
        st.roll = 13;
        printf("\nRoll=%d", st.roll);
        printf("\nMarks=%.2f", st.marks);
        return 0;
}
```

Output

Roll=13

Marks=0.00

Here, Marks is zero as memory is replaced by another variable st.roll

Difference between structure and union

Structure	Union
1. Each member within a structure is assigned its own unique storage. It takes more memory than union.	1. All members within union share the same storage area of computer memory. It takes less memory than structure.

2. The amount of memory required to store a structure is the sum of the sizes of all members.	2. The amount of memory required to store an union is same as member that occupies largest memory.
3. All the structure members can be accessed at any point of time.	3. Only one member of union can be accessed at any given time.
4. Structure is defined as: struct student { char name[30]; int roll; float marks; }st;	4. Structure is defined as: union student { char name[30]; int roll; float marks; }st;

Some Programs

1. Write a program to read the name, address, and salary of 3 employee using array of structure and display information of each employee in ascending order of their name.

```
#include<stdio.h>
#include<string.h>
#define size 3
struct employee
{
    char name[20];
    char address[20];
    float salary;
};
int main(){
    struct employee e[size], temp;
    int i, j;
    float sal;
    for(i=0; i<size;i++){
            printf("Enter information of employee no.%d", i+1);
            printf("\nName:");
            scanf("%s", e[i].name);
            printf("\nAddress:");
            scanf("%s", e[i].address);
            printf("\nSalary:");
            scanf("%f", &sal);
            e[i].salary=sal;
    }
    for(i=0; i<size-1; i++){
            for(j=j+1; j<size;j++){
                    if(strcmp(e[i].name, e[j].name)>0){
                            temp=e[i];
                            e[i]=e[j];
                            e[j]=temp;
                    }
            }
    }
    printf("\nEmployee list in ascending order of their name");
    printf("\nEmployee Name\t\tAddress\t\tSalary");
    printf("\n-----------------------------------------------------------------\n");
    for(i=0; i<size; i++){
            printf("%s\t\t\t%s\t\t%f\n", e[i].name, e[i].address, e[i].salary);
    }
```

```
    return 0;
}
```