

Unit-7

Structure and Union

(Marks -8)

CTEVT Diploma in Computer Engineering

Subject : C Programming

Shree Sarwajanik Secondary Technical School

Prepared By: Er. Abinash Adhikari

C Structures

A structure in C is a derived or user-defined data type. We use the keyword `struct` to define a custom data type that groups together the elements of different types. The difference between an array and a structure is that an array is a homogenous collection of similar types, whereas a structure can have elements of different types stored adjacently and identified by a name.

We are often required to work with values of different data types having certain relationships among them. For example, a book is described by its title (string), author (string), price (double), number of pages (integer), etc. Instead of using four different variables, these values can be stored in a single `struct` variable.

Declare (Create) a Structure

- ❖ You can create (declare) a structure by using the "struct" keyword followed by the structure_tag (structure name) and declare all of the members of the structure inside the curly braces along with their data types.
- ❖ To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member.
- ❖ **Syntax of Structure Declaration**

```
struct [structure tag]{  
    member definition;  
    member definition;  
    ...  
    member definition;  
} [one or more structure variables];
```

- ❖ The structure tag is optional and each member definition is a normal variable definition, such as "int i;" or "float f;" or any other valid variable definition.
- ❖ At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional.

❖ **Example:**

```
struct book{  
    char title[50];  
    char author[50];  
    double price;  
    int pages;  
} book1;
```

- ❖ Here, we declared the structure variable book1 at the end of the structure definition. However, you can do it separately in a different statement.

Structure Variable Declaration

To access and manipulate the members of the structure, you need to declare its variable first. To declare a structure variable, write the structure name along with the "struct" keyword followed by the name of the structure variable. This structure variable will be used to access and manipulate the structure members.

Example

```
struct book book1;
```

The following statement demonstrates how to declare (create) a structure variable

Usually, a structure is declared before the first function is defined in the program, after the include statements. That way, the derived type can be used for declaring its variable inside any function.

Structure Initialization

The initialization of a struct variable is done by placing the value of each element inside curly brackets.

Example

The following statement demonstrates the initialization of structure

```
struct book book1 = {"Learn C", "Dennis Ritchie", 675.50, 325};
```

Accessing the Structure Members

To access the members of a structure, first, you need to declare a structure variable and then use the dot (.) operator along with the structure variable.



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

```
struct book{  
    char title[10];  
    char author[20];  
    double price;  
    int pages;  
};
```

```
int main(){  
    struct book book1 = {"Learn C", "Dennis Ritchie", 675.50, 325};  
  
    printf("Title:  %s \n", book1.title);  
    printf("Author: %s \n", book1.author);  
    printf("Price:  %lf\n", book1.price);  
    printf("Pages:  %d \n", book1.pages);  
    printf("Size of book struct: %ld", sizeof(struct book));  
    return 0;  
}
```

Output:

```
Title:  Learn C  
Author: Dennis Ritchie  
Price:  675.500000  
Pages:  325  
Size of book struct: 48
```



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

Output:

```
struct book{
    char title[10];
    char author[20];
    double price;
    int pages;
} book1;
```

Title: Learn C
Author: Dennis Ritchie
Price: 675.500000
Pages: 325

```
int main(){
    strcpy(book1.title, "Learn C");
    strcpy(book1.author, "Dennis Ritchie");
    book1.price = 675.50;
    book1.pages = 325;

    printf("Title: %s \n", book1.title);
    printf("Author: %s \n", book1.author);
    printf("Price: %lf \n", book1.price);
    printf("Pages: %d \n", book1.pages);
    return 0;
}
```


Array of Structures in C

- ❖ An array of structures is simply an array where each element is a structure. It allows you to store several structures of the same type in a single array.
- ❖ **Array of Structure Declaration**
 - Once you have already defined structure, the array of structure can be defined in a similar way as any other variable.
*struct struct_name **arr_name** [size];*
- ❖ **Need for Array of Structures**
 - ❖ Suppose we have 50 employees, and we need to store the data of 50 employees. So for that, we need to define 50 variables of struct Employee type and store the data within that. However, declaring and handling the 50 variables is not an easy task. Let's imagine a bigger scenario, like 1000 employees.
 - ❖ So, if we declare the variable this way, it's not possible to handle this.

struct Employee emp1, emp2, emp3, emp1000;
- ❖ For that, we can define an array whose data type will be struct Employee so that will be easily manageable.
 - Arrays of structures allow you to group related data together and handle multiple instances of that data more efficiently.
 - It is particularly useful when you want to manage large datasets (such as a list of employees, students, or products) where each element has a consistent structure.

```

#include <stdio.h>
// Structure definition
struct A {
    int var;
    char c;
};
int main() {
    // Declaration and initialization using nested
    // initializer list
    struct A arr1[2] = { {1, 'a'}, {2, 'b'} };

    // Declaration and initialization using non-nested
    // initializer list
    struct A arr2[2] = { 10, 'A', 20, 'B' };

    // Designated initialization
    struct A arr3[2] = { {.c = 'A', .var = 10}, {.var = 2, .c = 'b'} };
    //Driver Code Starts{
    for (int i = 0; i < 2; i++){
        printf("%d %c", arr1[i].var, arr1[i].c);
        printf("\n");
    }
    printf("\n");
    for (int i = 0; i < 2; i++){
        printf("%d %c", arr2[i].var, arr2[i].c);
        printf("\n");
    }
    printf("\n");
    for (int i = 0; i < 2; i++){
        printf("%d %c", arr3[i].var, arr3[i].c);
        printf("\n");
    }
    return 0;
}

```

Output:

1 a

2 b

10 A

20 B

10 A

2 b

Nested Structures

A nested structure in C is a structure where one of the members is itself another structure. This allows us to group related data in a hierarchical format, making it easier to represent and manipulate complex data models.

Why Use Nested Structures?

1. To organize data more logically.
2. To create reusable and modular data structures.
3. To represent real-world entities that have hierarchical relationships (e.g., an employee with an address or a student with exam results).

Syntax of Nested Structures

```
struct OuterStruct {  
    struct InnerStruct {  
        // Inner structure members  
    } innerMember; // Declare an inner structure instance  
    // Outer structure members  
};
```

```

#include <stdio.h>
#include <string.h>

// Inner Structure
struct Address {
    char street[50];
    char city[50];
    int zipCode;
};

// Outer Structure
struct Employee {
    int empID;
    char name[50];
    struct Address address; // Nested structure
};

int main() {
    // Declare an Employee structure variable
    struct Employee emp;

    // Assign values to the Employee structure
    emp.empID = 101;
    strcpy(emp.name, "John Doe");

    // Assign values to the nested Address structure
    strcpy(emp.address.street, "123 Maple Street");
    strcpy(emp.address.city, "New York");
    emp.address.zipCode = 10001;

    // Print the details
    printf("Employee ID: %d\n", emp.empID);
    printf("Employee Name: %s\n", emp.name);
    printf("Address: %s, %s, %d\n", emp.address.street, emp.address.city, emp.address.zipCode);

    return 0;
}

```

Output:



```

Employee ID: 101
Employee Name: John Doe
Address: 123 Maple Street, New York, 10001

```

Unions in C

A union is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple purpose.

All the members of a union share the same memory location. Therefore, if we need to use the same memory location for two or more members, then union is the best data type for that. The largest union member defines the size of the union.

Defining a Union

Union variables are created in same manner as structure variables. The keyword union is used to define unions in C language.

Syntax

```
union [union tag]{  
    member definition;  
    member definition;  
    ...  
    member definition;  
} [one or more union variables];
```

The "union tag" is optional and each member definition is a normal variable definition, such as "int i;" or "float f;" or any other valid variable definition.

At the end of the union's definition, before the final semicolon, you can specify one or more union variables.

Accessing the Union Members

To access any member of a union, we use the member access operator (.). The member access operator is coded as a period between the union variable name and the union member that we wish to access. You would use the keyword union to define variables of union type.

Syntax

```
union_name.member_name;
```

Initialization of Union Members


You can initialize the members of the union by assigning the value to them using the assignment (=) operator.

Syntax

```
union_variable.member_name = value;
```

Example

```
data.i = 10;
```

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

```
union Data{
    int i;
    float f;
    char str[20];
};
```

```
int main(){
    union Data data;

    data.i = 10;
    data.f = 220.5;
    strcpy(data.str, "C Programming");

    printf("data.i: %d \n", data.i);
    printf("data.f: %f \n", data.f);
    printf("data.str: %s \n", data.str);
    return 0;
}
```

Output:

```
data.i: 1917853763
data.f: 4122360580327794860452759994368.000000
data.str: C Programming
```

Here, we can see that the values of i and f (members of the union) show garbage values because the final value assigned to the variable has occupied the memory location and this is the reason that the value of str member is getting printed very well.

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

union Data{
    int i;
    float f;
    char str[20];
};

int main(){

    union Data data;

    data.i = 10;
    printf("data.i: %d \n", data.i);

    data.f = 220.5;
    printf("data.f: %f \n", data.f);

    strcpy(data.str, "C Programming");
    printf("data.str: %s \n", data.str);
    return 0;
}
```

Output:

Size of a: 4 bytes

Size of b: 4 bytes

Size of c: 20 bytes

Size of union: 20 bytes

Structure Vs. Union

Aspect	Structure	Union
Memory Allocation	Allocates separate memory for each member.	All members share the same memory location.
Size	Sum of sizes of all members (with padding).	Size of the largest member.
Access	All members can be accessed simultaneously.	Only one member can be accessed at a time.
Use Case	Use when all members need to hold values together.	Use when only one member is used at a time.
Example Use	Employee data, Student records.	Type conversion, Memory-critical applications.