Unit 7 Structure and Union

- Structure: definition, declaration, initialization, size of structure
- Accessing member of Structure
- Array of Structure
- Nested Structure
- Union: definition, declaration, size of union
- Structure Vs. Union

Structure

- In C programming, a struct (or structure) is a collection of variables (can be of different types) under a single name.
- Unlike an array, a structure can contain many different data types (int, float, char, etc.).

Defining Structure

To define a struct, the struct keyword is used.

Syntax

```
struct structureName {
  dataType member1;
  dataType member2;
  ...
};
```

Example

```
struct Person {
  char name[50];
  int citNo;
  float salary;
};
```

Creating struct variable

When a struct, type is declared, no storage or memory is allocated. To allocate memory of a given structure type and work with it, we need to create variables.

Example

```
struct Person {
  // code
};

int main() {
  struct Person person1, person2, p[20];
  return 0;
}
```

Another way of creating a struct variable is:

```
struct Person {
   // code
} person1, person2, p[20];
```

Access Members of a Structure

To access members of a structure, use the dot syntax (.)

Example:

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

// create struct with person1 variable
struct Person
{
    char name[50];
    int citNo;
    float salary;
} person1;

int main()
{

    // assign value to name of person1
    strcpy(person1.name, "George Orwell");

    // assign values to other person1 variables
    person1.citNo = 1984;
    person1.salary = 2500;

    // print struct variables
    printf("Name: %s\n", person1.name);
    printf("Citizenship No.: %d\n", person1.citNo);
    printf("Salary: %.2f", person1.salary);

    return 0;
}
```

Example 2:

```c
#include <stdio.h>

struct Car
{
    char brand[50];
    char model[50];
    int year;
};

int main()
{
    struct Car car1 = {"BMW", "X5", 1999};
    struct Car car2 = {"Ford", "Mustang", 1969};
    struct Car car3 = {"Toyota", "Corolla", 2011};

    printf("%s %s %d\n", car1.brand, car1.model, car1.year);
    printf("%s %s %d\n", car2.brand, car2.model, car2.year);
    printf("%s %s %d\n", car3.brand, car3.model, car3.year);

    return 0;
}
```

Array of Structures

As a structure in C is a user-defined data type, we can also create an array of it, same as other data types.

Syntax:

```c
struct structure_name array_name[size_of_array];
```

Example:

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

struct Student
{
    char name[50];
    char section;
    int class;
};
```

```c
int main()
{
    // creating an array of structures

    struct Student arr[5];

    for (int i = 0; i < 5; i++)
    {
        scanf("%s", arr[i].name);
        arr[i].section = 'A' + i;
        arr[i].class = i + 1;
        printf("name: %s section: %c class: %d\n", arr[i].name,
arr[i].section, arr[i].class);
    }

    return 0;
}
```

Nested structure

C provides us the feature of nesting one structure within another structure.

Example:

```c
#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.add.pin, emp.add.phone);
}
```

Unions

- A union is a user-defined type similar to structs in C.
- The main difference between structure and union is: Structures allocate enough space to store all their members, whereas **unions can only hold one member value at a time**.

Defining Union

Creating union variables

Access member of union

Size of structure vs union

```c
#include <stdio.h>
union unionJob
{
    // defining a union
    char name[32];
    float salary;
    int workerNo;
} uJob;

struct structJob
{
    char name[32];
    float salary;
    int workerNo;
} sJob;

int main()
{
    printf("size of union = %d bytes", sizeof(uJob));
    printf("\nsize of structure = %d bytes", sizeof(sJob));
    return 0;
}
```

Output:

size of union = 32 bytes

size of structure = 40 bytes

In the above example it is clear that unions take less memory space compared to structures.

Structure vs Union

Feature	Structures	Unions
Purpose	Used to group related variables of different types together	Used to save memory by sharing the same memory location for different variables
Memory Allocation	Each member has its own memory space	Members share the same memory space
Size Calculation	Sum of sizes of all members	Size of the largest member
Memory Efficiency	Less memory efficient as each member has its own memory space	More memory efficient as members share memory space
Initialization	Individual member initialization	Only one member can be initialized at a time

Feature	Structures	Unions
Access	Individual members can be accessed independently	Only one member can be accessed at a time
Member Overlap	Members do not overlap in memory	Members can overlap in memory
Memory Usage	Uses memory for each member, regardless of usage	Uses memory only for the largest member
Type Safety	Provides type safety for each member	No type safety, can lead to type-related errors
Usage	Suitable when different types of data need to be stored together	Suitable when memory efficiency is a priority and only one member is accessed at a time