1. Introduction to Structure and Union

In C language, **structure** and **union** are **user-defined data types** that allow grouping of **different data types** under a single name.
They are used to represent **complex data** efficiently.

- **Structure** → Stores multiple values with **separate memory**
- **Union** → Stores multiple values with **shared memory**

---

2. Need for Structure and Union

They are required to:

- Store related data together
- Improve data organization
- Represent real-world entities
- Simplify complex programs
- Reduce memory usage (union)

---

3. Structure in C Language

A **structure** is a collection of variables of **different data types**, grouped under one name.

Definition

A structure is a user-defined data type that allows storing multiple related values of different types.

---

4. Declaration of Structure

Syntax

```
struct structure_name
{
   data_type member1;
   data_type member2;
   ...
};
```

Example

```
struct Student
```

```
{
   int roll;
   char name[20];
   float marks;
};
```

---

5. Declaring Structure Variables

Method 1

```
struct Student s1, s2;
```

Method 2 (With Definition)

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

---

6. Accessing Structure Members

The **dot operator (.)** is used to access structure members.

Example

```
s1.roll = 101;
printf("%d", s1.roll);
```

---

7. Initialization of Structure

Example

```
struct Student s1 = {101, "Ravi", 85.5};
```

---

8. Structure and Arrays

An **array of structures** is used to store multiple records.

Example

```
struct Student s[3];
```

Access:

```
printf("%s", s[0].name);
```

---

9. Structure and Functions

Structures can be:

- Passed to functions
- Returned from functions

Passing Structure

```
void display(struct Student s)
{
   printf("%d", s.roll);
}
```

---

10. Pointer to Structure

Pointers can point to structures.

Syntax

```
struct Student *p;
```

Access Members Using Arrow Operator (->)

```
p->roll;
```

---

11. Nested Structure

A structure inside another structure is called **nested structure**.

Example

```
struct Date
{
   int day, month, year;
};

struct Student
{
   int roll;
   struct Date dob;
};
```

---

12. typedef with Structure

`typedef` is used to create an alias for structure.

Example

```
typedef struct Student
```

```
{
   int roll;
   float marks;
} STU;
```

---

13. Advantages of Structure

- Groups related data
- Improves readability
- Supports complex data handling
- Useful for records and databases

---

14. Limitations of Structure

- Memory consumption is high
- No direct memory sharing
- Slow comparison

---

15. Union in C Language

A **union** is a user-defined data type in which **all members share the same memory location**.

Definition

Union is a data type that stores different data types in the same memory location.

---

16. Declaration of Union

Syntax

```
union union_name
{
   data_type member1;
   data_type member2;
};
```

Example

```
union Data
{
   int i;
   float f;
   char c;
};
```

17. Accessing Union Members

Same dot operator is used.

```
union Data d;
d.i = 10;
```

□□ Only **one member** holds a valid value at a time.

18. Memory Allocation in Union

- Memory size = size of **largest member**
- All members share same memory

Example

```
sizeof(union Data);
```

19. Difference Between Structure and Union

Feature	Structure	Union
Memory	Separate	Shared
Size	Sum of all members	Largest member
Access	All at a time	One at a time
Data Safety	High	Low

20. Structure vs Union (Example)

```
struct A
{
   int a;
   float b;
};

union B
{
   int a;
   float b;
};
```

21. Applications of Structure

- Student records
- Employee database
- File handling
- Networking packets
- Operating systems

22. Applications of Union

- Memory-efficient programs
- Embedded systems
- Device drivers
- Interpreters

23. Common Errors

1. Wrong member access
2. Forgetting structure keyword
3. Misuse of union members
4. Incorrect pointer usage

24. Best Practices

- Use structure for safety
- Use union for memory optimization
- Use typedef for readability
- Initialize structures properly

25. Conclusion

Structure and union are powerful features of C language that help in organizing and managing complex data. While **structures** provide safety and clarity, **unions** offer memory efficiency. Understanding both is essential for system-level and real-world programming.