perplexity

Structures - Defining a Structure variable, Accessing members, Array of

structures, Passing structure to function; Union.

teach me all this with appropriate points, differences to note, syntax and an example

Here's a comprehensive guide to **structures** and **unions** in C, covering definitions, syntax, examples, and key differences:

1. Structures

Defining a Structure

- **Syntax**:

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

- **Example**:

```
struct Student {
    int roll_no;
    char name[^50];
    float marks;
};
```

Declaring Structure Variables

- **Syntax**:

```
struct structure_name variable_name;
```

- **Example**:

```
struct Student s1; // Declare a Student variable
```

Accessing Structure Members

- Use the **dot (.) operator** for structure variables.
- **Example**:

```
s1.roll_no = 101;
strcpy(s1.name, "Alice");
s1.marks = 85.5;
```

Array of Structures

- **Syntax**:

```
struct structure_name array_name[size];
```

- **Example**:

```
struct Student class[^30]; // Array of 30 Student structures
class[^0].roll_no = 101;   // Access first student's roll number
```

Passing Structures to Functions

By Value (Copy of structure passed):

```
void printStudent(struct Student s) {
    printf("Roll: %d\nName: %s\nMarks: %.2f\n", s.roll_no, s.name, s.marks);
}
```

By Reference (Pointer to structure passed):

```
void updateMarks(struct Student *s, float new_marks) {
    s->marks = new_marks; // Use -> operator for pointers
}
```

2. Unions

Defining a Union

- **Syntax**:

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

```
    // ...
};
```

- **Example**:

```
union Data {
    int int_val;
    float float_val;
    char str[^20];
};
```

Using Unions

- All members share the **same memory location**.
- **Example**:

```
union Data d;
d.int_val = 10;       // Store integer
d.float_val = 3.14;   // Overwrite memory (int_val is now garbage)
strcpy(d.str, "Hi");  // Overwrite again
```

Key Differences: Structure vs Union

Feature	Structure	Union
Memory Allocation	Each member has separate memory.	All members share the same memory.
Size	Sum of all members' sizes.	Size of the largest member.
Usage	Store multiple values simultaneously.	Store one value at a time (memory-efficient).
Access	All members retain their values.	Only the last assigned member holds a valid value.

Examples

Structure Example

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

struct Book {
    char title[^100];
    char author[^50];
    float price;
};
```

```
int main() {
    struct Book b1;
    strcpy(b1.title, "C Programming");
    strcpy(b1.author, "Dennis Ritchie");
    b1.price = 450.50;
    printf("Title: %s\nPrice: ₹%.2f\n", b1.title, b1.price);
    return 0;
}
```

Union Example

```
#include <stdio.h>

union Number {
    int i;
    float f;
};

int main() {
    union Number num;
    num.i = 10;
    printf("Integer: %d\n", num.i); // Output: 10
    num.f = 3.14;
    printf("Float: %.2f\n", num.f); // Output: 3.14
    printf("Integer (garbage): %d\n", num.i); // Undefined
    return 0;
}
```

When to Use

- **Structure**: Group related data (e.g., student records).
- **Union**: Save memory when only one member is needed at a time (e.g., parsing different data types from a network packet).

⁂