✅ UNIT – IV : Recursion, Structures & Unions

PART – A : RECURSION

---

1. Recursion – Introduction

Definition

Recursion is a programming technique in which a function calls itself to solve a problem.

Key Idea

A large problem is broken into smaller sub-problems of the same type.

---

2. Nature of Recursion

Characteristics

- Function calls itself
- Must have a base condition
- Uses stack memory
- Stops when base condition becomes true

Basic Structure

```
function() {
   if(condition)
      return value;
   else
      function();
}
```

---

3. Tracing a Recursive Function

Example – Factorial

```
int fact(int n) {
   if(n == 0)
      return 1;
```

```
    else
        return n * fact(n-1);
}
```

Tracing for fact(3)

fact(3) = 3 * fact(2)

fact(2) = 2 * fact(1)

fact(1) = 1 * fact(0)

fact(0) = 1

Result

fact(3) = 3 × 2 × 1 = 6

---

4. Recursive Mathematical Functions

Example: Fibonacci Series

```
int fib(int n) {
    if(n == 0)
        return 0;
    else if(n == 1)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

---

5. Recursive Functions with Array Parameters

Example: Sum of Array Elements

```
int sum(int a[], int n) {
    if(n == 0)
        return 0;
    return a[n-1] + sum(a, n-1);
}
```

---

6. Recursive Functions with String Parameters

Example: String Length

```
int length(char str[]) {
    if(str[0] == '\0')
        return 0;
    return 1 + length(str + 1);
}
```

Text Diagram (Recursion Flow)

Function Call

↓

Checks Condition

↓

Calls Itself

↓

Base Case Reached

↓

Returns Result

Advantages of Recursion

- Simplifies complex problems
- Reduces code size
- Useful for tree and graph problems

Disadvantages

- Uses more memory
- Slower than loops
- Risk of stack overflow

PART – B : STRUCTURES AND UNIONS

---

7. User-Defined Structure Types

Definition

A structure is a user-defined data type that groups different data types.

Syntax

```
struct student {
    int id;
    char name[20];
    float marks;
};
```

---

8. Declaring Structure Variables

```
struct student s1;
```

---

9. Accessing Structure Members

```
s1.id = 101;
strcpy(s1.name, "Ram");
s1.marks = 85.5;
```

---

10. Structure Type Data as Input and Output Parameters

Passing Structure to Function

```
void display(struct student s) {
    printf("%d %s %.2f", s.id, s.name, s.marks);
}
```

---

11. Functions with Structured Result Values

Returning Structure from Function

```
struct student getData() {
    struct student s;
    s.id = 1;
    strcpy(s.name, "Ravi");
    s.marks = 90;
    return s;
}
```

---

12. Union Types

Definition

A union allows storing different data types in the same memory location.

Syntax

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

---

13. Difference Between Structure and Union

Structure	Union
Each member has separate memory	All share same memory
More memory usage	Less memory usage
All values stored	Only one value at a time

---

14. Example of Union

```
union data d;
d.i = 10;
printf("%d", d.i);
```

Text Diagram: Memory Allocation

Structure:

| int | float | char |

Union:

| int / float / char |

Important Keywords (For Exams)

- Recursion
- Base condition
- Stack memory
- Structure
- Union
- Self-referencing
- Function call