DSA SERIES

- Learn Coding

Topic to be Covered today

Recursion

LETS START TODAY'S LECTURE

Lecture-41

Recursion

↳ It is a programming technique.

↳ A function calling itself. to solve a problem.

↳ Recursion : { → A base case
→ Recursive step that leads to base case.

```
solve () {
  // Base Case

  // Recursive step
}
```

Instead of solving the whole problem at once, recursion breaks it down to smaller sub problems of the same type until it reaches a simple base case that can be solved directly.

⇒ The function is divided into two parts :-

i) Base case : The condition under which the function stops calling itself.

(ii) Recursive case : The part of the function where the problem is broken into smaller instances.

For example : ① Find factorial of a number

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$4! = 4 \times 3 \times 2 \times 1$

$3! = 3 \times 2 \times 1$

$2! = 2 \times 1$

$\boxed{0! = 1}$

```
int factorial (int n) {
    // Base case
    if (n==0) {
        return 1;
    }
    return n * factorial(n-1);
}
```

② Reverse a string

Str = "Ankit"

```
void solve (str, idx){
    if (idx >= str.length()){
        return;
    }
    solve (str, idx+1) ;
    cout<< str [idx]<<endl;
}
```

Recursion Tree

factorial problem

fib(5) ⇒ 5×4×3×2×1

↓

fib(4) ⇒ 4×3×2×1

↓

fib(3) ⇒ 3×2×1

↓

fib(2) → 2×1

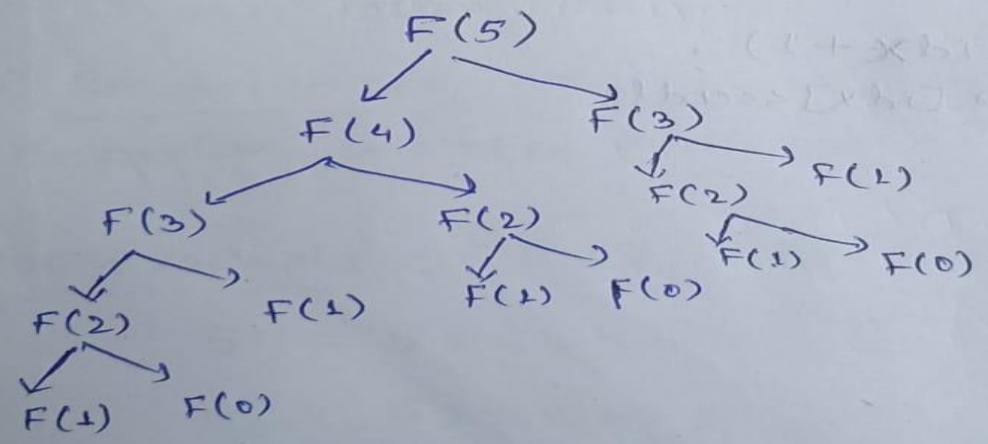
↓

fib(1) ⇒ 1

(3) Find the nth fibonacci number.

0 1 1 2 3 5 ?

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

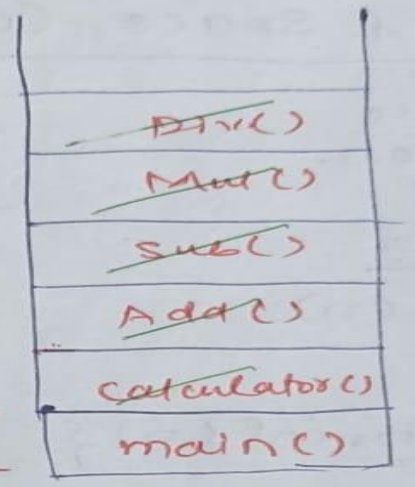


Recursion Call stack

```
int
void add () {
  }
void sub () {
  }
void mul() { }
void divide() { }

int calculator () {
    add ();
    sub ();
    mul ();
    divide();
  }

int main() {
 calculator (a, b);
 }
```



stack frame ← {

Memory (RAM)

→ Stores information about active function calls.

→ Each function calls adds a new frame to the call stack.

→ Each frame has its own set of local variables. (variable scoping)

Different scenario at :-

→ pass by value
→ pass by reference.

Time & Space Complexity of Recursive function.

Recurrence Relation.

Example:-

Factorial (n)

```
int factorial (n) {
 if (n <= 1) {
   return 1;
 }
return n* factorial (n-1);
}
```

$$T(n) = T(n-1) + 2$$
$$= \{T((n-1)-1) + 2\} + 2$$
$$= T(n-2) + 4$$
$$= T((n-2)-1) + 2) + 4$$
$$= T(n-3) + 6$$
$$T(n) = T(n-k) + 2 \times k$$
$$= T$$
$$\vdots$$
$$T(0)$$

$n - k = 0$
$n = k$

$$\Rightarrow T(n) = T(0) + 2 \times n$$
$$= 1 + 2 \times n$$
$$\boxed{T(n) = O(n)}$$

Space complexity

↳ It means how many times, the function was called.

F(n)
↓
F(n-1)
↓
F(n-2)
⋮
F(1)

n

$\boxed{O(n) = S.C}$

Tail Recursion

Form of recursion, where the recursive call is the last operation performed in the function.

Types of Recursion

① Direct Recursion

↳ When a function calls itself directly.

Example:-

```
void Recursion (n) {
    if(n==0) return;
    cout << n << " ";
    ~~direct~~Recursion(n-1);
}
```

② Indirect Recursion

↳ When a function calls another function, and that function calls the first one back.

Example:-

```
void functionA(int n) {
  if (n <= 0) return;
  cout << n << " ";
  funcB(n-1);   // calls another function.
}

void functionB(int n){
  if (n <=0) return;
  cout << n << " ";
  funcA (n-2);
}

int main (){
  funcA (5);
  return 0;
}
```

③ Tail Recursion

↳ When the recursive call is the last statement in the function.

⇒ No computation is pending after the recursive call.

```
int factorial (int n) {
    if (n<=1) {
        return 1;
    }
    return n*factorial(n-1);
}
```

✗ Not a tail Recursion.

↳ Transforming this to tail Recursion

```
int Fact (int n, int product){
    if (n<=1) {
        return Product;
    }
    return Fact((n-1), product*n);
}
```

Advantage of Tail Recursion

↳ Faster Computation

↳ Space optimisation

Printing Vs Returning in Recursion

(1) Printing Recursion

- The function directly prints values at each ste
- Output happens during the recursive calls execution.
- Once printed, values cannot be reused Later

Example:

```
void printNumbers(int n){
    if (n==0) return;
    cout << n << " ";
    printNumbers(n-1);
}
```

(2) Returning in Recursion

- The function returns a value instead of printing it.
- Returned values can be collected, modified, or used further.
- Printing can still be done later in main() or another function.

Example :

```
int func(n){
    if (n==0) return 0;
    return n+ sum(n-1);
}
```

Basic Recursion problems

1. factorial ✓
2. fibonacci ✓
3. Printing Numbers from 1 to N
4. Sum of Numbers from 1 to N.
5. Printing Numbers from N to 1.
6. Reverse a string
7. Reverse a number.

Learn coding

THANK YOU