Page No.
Date

⊗ Recursion

When a funcⁿ call itself until a specific condⁿ is met.

```
void f() {
    print(1);
    f();
}
```

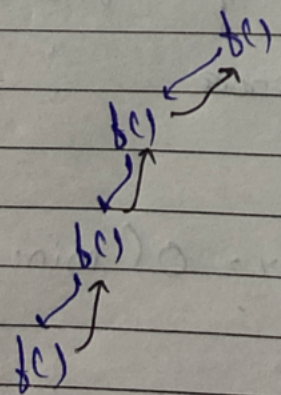
Observation:
1) funcⁿ will keep calling itself
2) After a certain time, the memory will be full and this is known as Stack overflow
3) that's why we define a condⁿ to overcome the chance of Stack overflow.

```
int cnt = 0;
f() {
  if (cnt == 4) {
    return ;
  } else {
    print(cnt);
    cnt++;
    f();
  }
}
```

Observation:
1) When the funcⁿ is called for first time, it will mismatch with if condⁿ and goes to else
2) now in else it will keep calling it again but when cnt = 4 reaches. if condⁿ will execute and the funcⁿ will be terminated.

⊗ Recursion tree

in the above example, instead of writing for every iteration



Basic recursion problems

1) Print name 5 times
2) Print linearly from 1 to N
3) Print from N to 1
4) Print linearly from 1 to N } by backtrack
5) Print from N to 1

Page No.

1) Print name 5 times

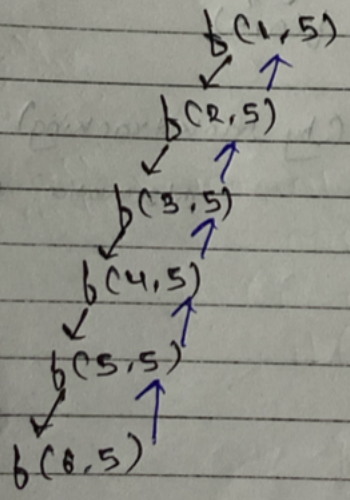
```
void f(i, n){
    if (i > n){
        return ;
    }
    print("ray");
    f(i+1, n);
}
```

```
int main(){
    int n;
    cin >> n;
    f(1, n);
}
```

let n = 5

Output:
ray
ray
ray
ray
ray

recursion tree:



2) Print linearly from 1 to N (let N = 4)

Output:
1
2
3
4

```
void f(i, N){
    if (i > N){
        return ;
    }
    print(i);
    f(i+1, N);
}
```

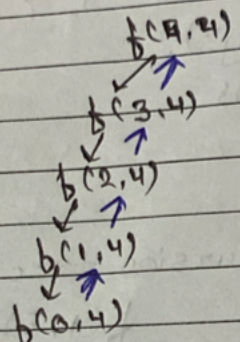
```
int main(){
    int n;
    cin >> n;
    f(1, N);
}
```

3) print from N to 1 (let N=4)

```
void f(i,N){
    if(i<1){
        return;
    }
    print(i);
    f(i-1,N);
}
```

```
int main(){
    int N;
    cin>>N;
    f(N,N);
}
```

Output:
4
3
2
1



4) print from 1 to N (by backtracking)

```
void f(i,N){
    if(i<1){
        return;
    }
    f(i-1,N);
    print(i);
}
```

```
int main(){
    int N;
    cin>>N;
    f(N,N);
}
```

N=3

Observation: 1) when it execute for f(3,3), the if statement will be ignored and it will again call the funcn but by decrementing i

2) then execute for f(2,3) still same till f(1,3)

3) when f(0,3) is executing 'if' will execute and return the funcn to f(1,3) and the print line will execute. then for (2,3)

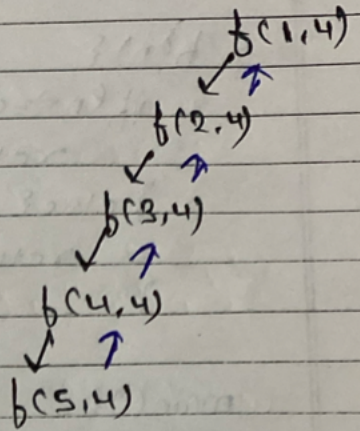
Output
1
2
3

5) print from N to 1 (using backtrack)

```
void f(i,N){
  if(i>N){
    return;
  }
  f(i+1,N);
  print(i);
}

int main(){
  int N;
  cin>>N;
  f(1,4);
}
```

Output
4
3
2
1



i) Jaha pe 'if' statement execute hoyega waha pe funcn return hoke print statement ko execute karta chalega.

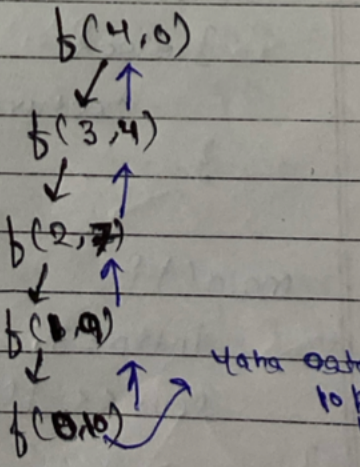
Q- Sum of first N natural number using recursion
(parameter, functional)

→ Parameter approach

```
f(i,Sum){
  if(i<1){
    print(Sum);
    return;
  }
  f(i-1,Sum+i);
}

main(){
  int n;
  cin>>n;
  f(n,0);
}
```

let n=4



Output : 10

logic we are going to use { f(3) → 3 + f(2)
f(2) → 2 + f(1)
f(1) → 1 + f(0) }

Page No.
Date

~→ functional approach

```
f(n){
  if(n==0){
    return 0;
  }else{
    return n + f(n-1);
  }

main(){
  int n=3;
  cout<< f(n);
}
```

Observation
1) when funcn is called for first time, it will execute else statement
2) after 1st execution, it will return 3+f(2)
it will again call f(n) and else condn again execute until the 'if' is executed.

Q- factorial of N number

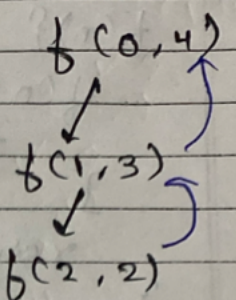
```
f(n){
  if(n==0 || n==1){
    return 1;
  }else{
    return n* f(n-1)
  }

main(){
  int n=3;
  cout<< f(3);
}
```

Page No.
Date

Reverse an array using recursion

```
f(l,r){
    if(l>=r){
        return;
    }
    swap(a[l], a[r]);
    f(l+1, r-1);
}
```




```
main(){
    arr = {1,3,2,5,4};
    f(0, n-1);
}
```

Output :
{4, 5, 2, 3, 1}

To check if string is palindrome or not

logic : 1) reverse the string
2) compare it with original string

```
f(l,r){
    if(l>=r){
        return;
    }
    swap(a[l], a[r]);
    f(l+1, r-1);
}
```

```
bool chk(l,r){
    if(l>=r){
        return true;
    }
    if(a[l] != a[r]{
        return false;
    }
    return chk(l+1, r-1);
```

Fibonacci (we know 0 and 1 are 1st and 2nd Fib. number)

0 1 1 2 3 5 8 13 21 34 ~~~

Sum Sum Sum Sum Sum

$f(n) = f(n-1) + f(n-2)$

```
f(n){
  if(n<=1){
    return n;
  }
  return f(n-1) + f(n-2);
}

main(){
  int n=4;
  cout << f(4);
```

Observation:
1) here 2 recursion calls are taking place simultaneously
2) when we run for n=4 → it will again run funcn for f(3) → as well as for f(2)
3) f(n-1) will execute first completely. Till then f(n-2) will not be in between
4) Stack space will keep f(n-2) to wait.

How code is running

n=4

f(4){
  if(n<=1){ (will not execute)
    return n;
  }
return f(n-1) + f(n-2)

f(3) → again it will not execute → f(3) = f(2) + f(1)

f(2) → again it will not execute (Now it will execute) → f(2) = f(1) + f(0)

f(2) = f(1) + f(0)