Recursion - 1

Lecture-27

Raghav Garg

What and Why?

Recursion → recurrence relation

What ? → • Loop ki replacement

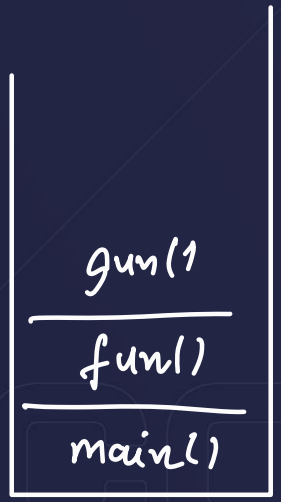
• problem = something + subproblem

→ function calling itself

Why → To solve problems

Function calls

```cpp
#include<iostream>
using namespace std;
void gun(){
    return; // khatam
     cout<<"Hello CW"<<endl;

}
void fun(){
    cout<<"Hello PW"<<endl;
    gun();
    return;
}
int main(){
    fun();
}
```



Call Stack

- Hello PW
-

Function calling itself

```cpp
#include<iostream>
using namespace std;
void fun(){
    cout<<"Hello PW"<<endl;
    fun();
}
int main(){
    fun();
}
```

- Hello PW
- Hello PW
- Hello PW

Function calling itself

```cpp
#include<iostream>
using namespace std;
void fun(int n){
    if(n==0) return;
    cout<<"Hello PW"<<endl;
    fun(n-1);
}
int main(){
    fun(3);
}
```

```cpp
void fun(int n){
    if(n==0) return;
    cout<<"Hello PW"<<endl;
    fun(n-1);
}
```

```cpp
void fun(int n){
    if(n==0) return;
    cout<<"Hello PW"<<endl;
    fun(n-1);
}
```

```cpp
void fun(int n){
    if(n==0) return;
    cout<<"Hello PW"<<endl;
    fun(n-1);
}
```

Output

- Hello PW
- Hello PW
- Hello PW

Function calling itself

Classwork : Print goodmorning 'n' no. of times, where 'n' is user input.

Ques : Make a function which calculates the factorial of n using recursion.

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

$= 5 \times 4!$

$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$

$= 7 \times 6!$

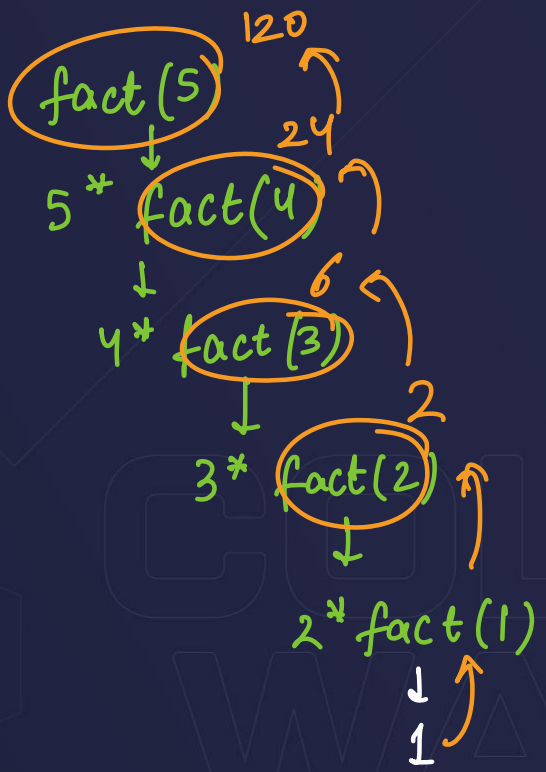
$n! = n \times n-1 \times n-2 \times n-3 \times \ldots \ldots 3 \times 2 \times 1$

$n! = n \times (n-1)!$

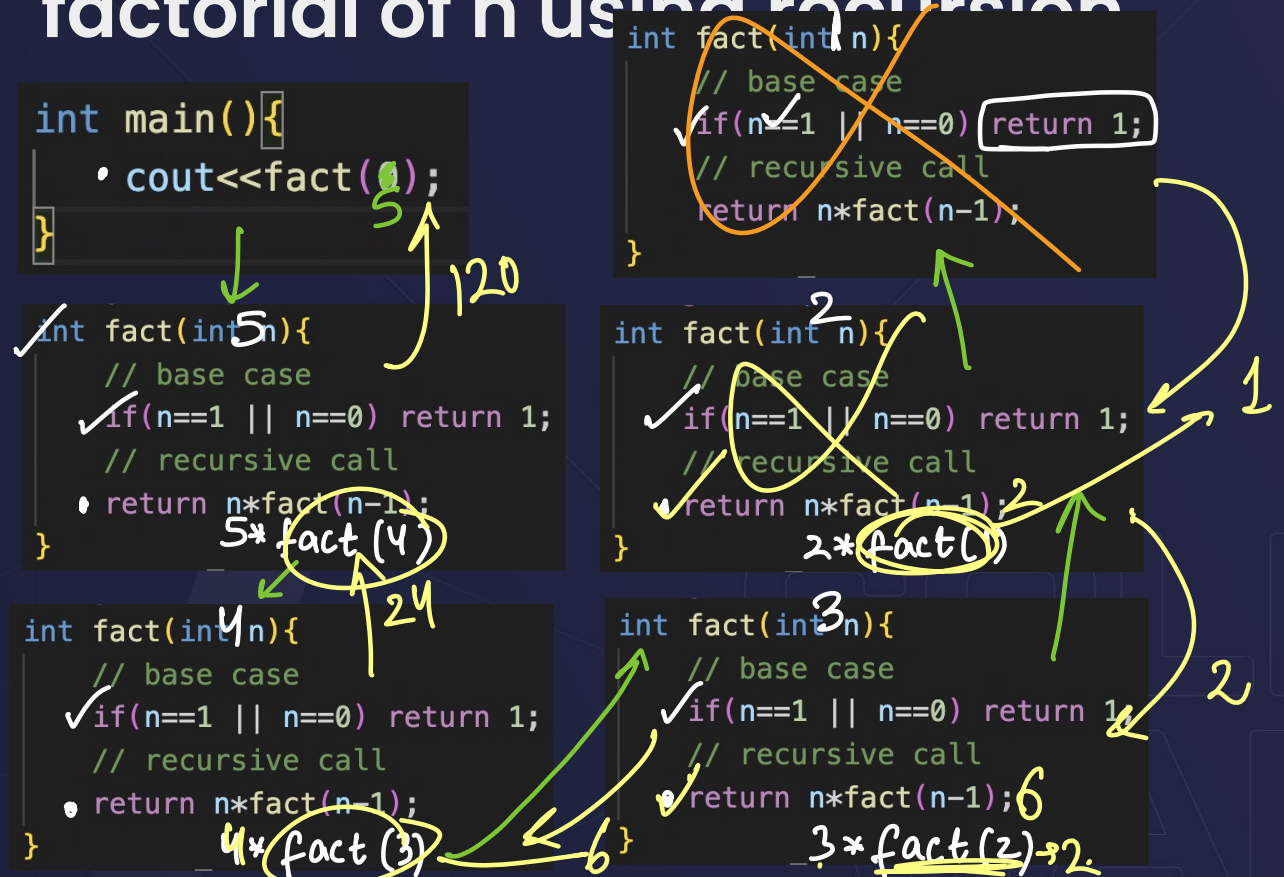
factorial of n = 1 to n numbers ka product

Ques : Make a function which calculates the factorial of n using recursion.

```cpp
#include<iostream>
using namespace std;
int fact(int n){
    return n*fact(n-1);
}
int main(){
    cout<<fact(5);
}
```



Ques : Make a function which calculates the factorial of n using recursion



Ques : Make a function which calculates the factorial of n using

```cpp
int main(){
    cout<<fact(5);
}
```

```cpp
int fact(int n){
    // base case
    if(n==1 || n==0) return 1;
    // recursive call
    return n*fact(n-1);
}
```

$n! \rightarrow 2n$ ops

$\downarrow$

$T.C. = O(n)$

$S.C. = O(n)$ stack frames

$\rightarrow O(n)$


```cpp
int fact(int n){
    // base case
    if(n==1 || n==0) return 1;
    // recursive call
    return n*fact(n-1);
}
```

```cpp
int fact(int n){
    // base case
    if(n==1 || n==0) return 1;
    // recursive call
    return n*fact(n-1);
}
```

```cpp
int fact(int n){
    // base case
    if(n==1 || n==0) return 1;
    // recursive call
    return n*fact(n-1);
}
```

```cpp
int fact(int n){
    // base case
    if(n==1 || n==0) return 1;
    // recursive call
    return n*fact(n-1);
}
```

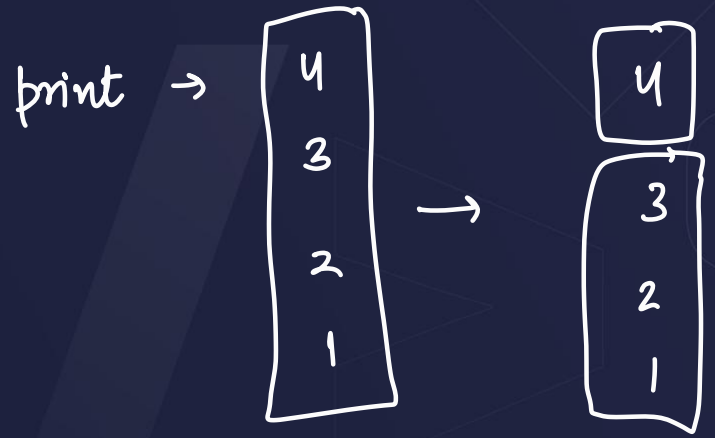
Ques : Print n to 1

```
int n;
cin>>n;
print(n);
```

Output

n = 4

4
3
2
1



Ques : Print 1 to n (extra parameter)

```
void print(int i,int n){
    if(i>n) return;
    cout<<i<<endl;
    print(i+1,n);
}
```

```
void print(int i,int n){
    if(i>n) return;
    cout<<i<<endl;
    print(i+1,n);
}
```

```
void print(int i,int n){
    if(i>n) return;
    cout<<i<<endl;
    print(i+1,n);
}
```

```
void print(int i,int n){
    if(i>n) return;
    cout<<i<<endl;
    print(i+1,n);
}
```

```
void print(int i,int n){
    if(i>n) return;
    cout<<i<<endl;
    print(i+1,n);
}
```

output

- 1
- 2
- 3
- 4
- 5

Ques : Print 1 to n (after recursive call)

```
int main(){
    print(4);
}
```

```
void print(int n){
    if(n==0) return; // base case
    print(n-1); // call
    cout<<n<<endl; // kaam
}
```

```
void print(int n){
    if(n==0) return; // base case
    print(n-1); // call
    cout<<n<<endl; // kaam
}
```

```
void print(int n){
    if(n==0) return; // base case
    print(n-1); // call
    cout<<n<<endl; // kaam
}
```

```
void print(int n){
    if(n==0) return; // base case
    print(n-1); // call
    cout<<n<<endl; // kaam
}
```

```
void print(int n){
    if(n==0) return; // base case
    print(n-1); // call
    cout<<n<<endl; // kaam
}
```

Output

- 1
- 2
- 3
- 4
-

Recursive Function

Base Case

Kaam

Call

Kaam

return

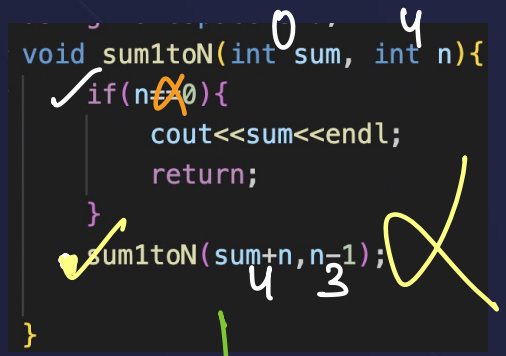
Ques : Print sum from 1 to n (Parameterised)

0

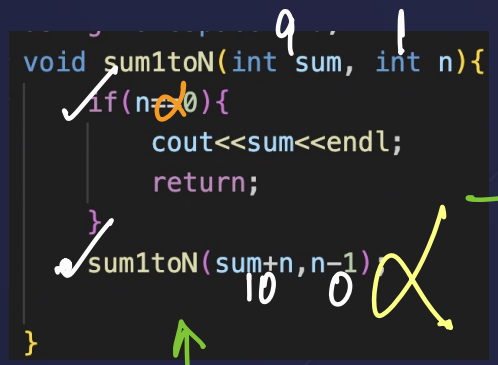
```
void sum(int sum, int n){
    if(n==0) return;
    sum(sum+n, n-1);

}
```

Ques : Print sum from 1 to n (Parameterised)

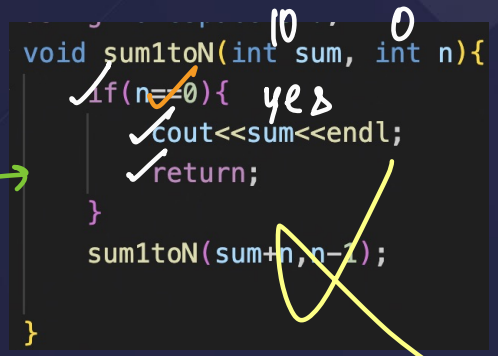
```
void sum1toN(int sum, int n){
    if(n==0){
        cout<<sum<<endl;
        return;
    }
    sum1toN(sum+n,n-1);
}
```



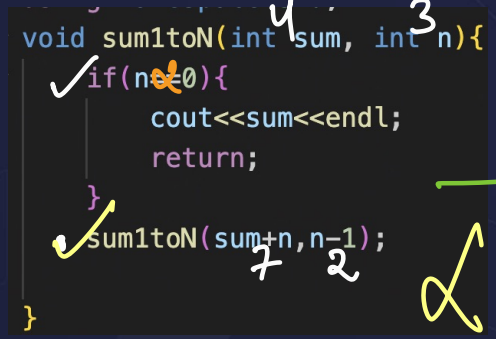

```
void sum1toN(int sum, int n){
    if(n==0){
        cout<<sum<<endl;
        return;
    }
    sum1toN(sum+n,n-1);
}
```



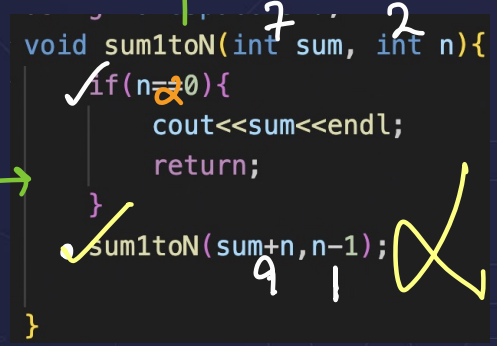

```
void sum1toN(int sum, int n){
    if(n==0){
        cout<<sum<<endl;
        return;
    }
    sum1toN(sum+n,n-1);
}
```




```
void sum1toN(int sum, int n){
    if(n==0){
        cout<<sum<<endl;
        return;
    }
    sum1toN(sum+n,n-1);
}
```




```
void sum1toN(int sum, int n){
    if(n==0){
        cout<<sum<<endl;
        return;
    }
    sum1toN(sum+n,n-1);
}
```



Ques : Print sum from 1 to n (Return type)

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

```
int fact (int n){
  if(n==1 || n==0) return 1;
  return n* fact(n-1);
}
```

sum(n) = n + sum(n-1);

fact(n) = n * fact(n-1)

1 to 6 → 6 to 1 = 6 + 5 + 4 + 3 + 2 + 1 = sum(6)

6 + sum(5) = sum(6)

Ques : Make a function which calculates 'a' raised to the power 'b' using recursion.

Input → a , b

Cout << pow(a,b);

$a^b = a^1 \times a^{b-1}$

power(a,b) = a * power(a,b-1)

a^b = a x a x a x a x a . . . .

b times

2^4 = 2 x 2 x 2 x 2

4 times

Ques : Make a function which calculates 'a' raised to the power 'b' using recursion.

$a^0 = 1$

```
if (b==0) return 1;
return  =  a * power(a,b-1);
```

Calculating Time and Space complexity

Recursive Solution

```
int pow(int a, int b){
    if(b==0) return 1;
    return a*pow(a,b-1);
}
```

T.C. = O(b)

S.C. = O(b) stack frames

Iterative Solution

```
int power(int a, int b){
    int p = 1;
    for(int i=1;i<=b;i++){
        p *= a;
    }
    return p;
}
```

T.C. = O(b)

S.C. = O(1)

Calculating Time and Space complexity

```
int pow(int a, int b){
    if(b==0) return 1;
    return a*pow(a,b-1);
}
```

```
int pow(int a, int b){
    if(b==0) return 1;
    return a*pow(a,b-1);
}
```

```
int pow(int a, int b){
    if(b==0) return 1;
    return a*pow(a,b-1);
}
```

```
int pow(int a, int b){
    if(b==0) return 1;
    return a*pow(a,b-1);
}
```

```
int pow(int a, int b){
    if(b==0) return 1;
    return a*pow(a,b-1);
}
```

$2^*b \rightarrow$ space khaleh

Next Lecture

Fibonacci Series

Understanding multiple calls in Recursion

More problems on Recursion

1 1 2 3 5 8 13 21 34 55 89 ...

1 2 3 4 5 6 7 8 9 10 11