🧠✨ Lecture 06 — FUNCTIONS IN C++

> “Functions are not a syntax feature.
> They are how large systems stay alive.”

🔴 FUNCTION KYU AYA? (FIRST THOUGHT PRINCIPLE)

❌ Life without Functions

Socho ek real app (Instagram / YouTube / Amazon):

• Same logic 1000 jagah
• Ek bug → 1000 files
• Change = nightmare
• Testing impossible

👉 **Ye software nahi, disaster hota hai**

✅ Function ne kya problem solve ki?

🧠 **Function = Logic ka container**

• Ek kaam
• Ek jagah
• Baar-baar reuse

📌 Real-life analogy
📱 **Camera Button**
Tum photo ka algorithm nahi likhte
👉 sirf **button press (call)** karte ho

🧠 C++ ME FUNCTION KYA HOTA HAI?

```
return_type functionName(parameters){

    // logic

    return value;

}
```

🔍 Breakdown (Yaad rakhne layak)

• **return_type** → kya output milega
• **functionName** → logic ka naam

- **parameters** → input
- **return** → result

📌 One-liner

> Function = **Input** → **Processing** → **Output**

🚀 PROGRAM EXECUTION ORDER (MOST IMPORTANT)

❓ Program kaha se start hota hai?

➡️ **Hamesha `main()` se**

```
main()
 ├─ function call
 |     ├─ function executes
 |     └─ returns value
 └─ main continues
```

📌 **Golden Rule (Interview favourite)**

> Function ka order important nahi
> **Declaration ka hona important hai**

⚠️ BIG ERROR — Function bina declare kiye call

❌ Wrong

```
int main(){
    cout << factorial(5);
}

int factorial(int n){ }
```

❗ Error:

```
not declared in this scope
```

✅ Fix

• Function upar define karo
• Ya declaration upar likho

👉 Beginner ke liye: **function hamesha main se upar**

🔢 FACTORIAL — WHY FUNCTION EXISTS

❌ Manual approach (BAD)

Same loop 3 baar likhna
Bug aaye → 3 jagah fix

✅ Function approach (GOOD)

```
int factorial(int n){

    int fact = 1;

    for(int i = 1; i <= n; i++){

        fact *= i;

    }

    return fact;

}
```

🧪 Example

```
factorial(5) → 120

factorial(7) → 5040

factorial(8) → 40320
```

📌 **SDE Rule**

DRY — Don't Repeat Yourself

➕ SUM — FUNCTION CONCEPT KA BASE

```
int sum(int a, int b){

    return a + b;

}
```

🧪 Example
`sum(3,4) → 7`

📌 Ye hi concept:
• APIs
• Backend services
• Microservices

🎓 GRADING SYSTEM (REAL LIFE)

```
void findGrade(int marks){

    if(marks > 90) cout<<"A+";

    else if(marks > 80) cout<<"A";

    else if(marks > 70) cout<<"B+";

    else if(marks > 60) cout<<"B";

    else cout<<"C";

}
```

📌 **Why else-if?**
Ek student → ek hi grade

📺 YOUTUBE EXAMPLE (INDUSTRY GOLD)

```
void youtube(string photo, string title, int views, int time){

    cout << photo << title << views << time << endl;

}
```

📌 Same UI
📌 Different data

👉 **Real apps isi tarah bante hain**

🧬 FUNCTION OVERLOADING

```
int sum(int a,int b);

int sum(int a,int b,int c);

float sum(float a,float b);
```

📌 Decision **compile time** par
📌 Return type se ❌ overloading

🔄 PASS BY VALUE vs REFERENCE

❌ Pass by value

Copy → original safe

✅ Pass by reference

Original variable modify

```
void swap(int &a,int &b){

    int t = a;

    a = b;

    b = t;

}
```

📌 Interview line

Reference passes memory address

🔢 DEFAULT PARAMETER

```
void print(int x = 5){

    cout << x;
}
```

📌 `print()` → 5
📌 `print(10)` → 10

Used in:
• APIs
• Config files
• Frameworks

🔁 SWAP TWO NUMBERS (XOR — ADVANCED)

🧠 XOR Rules (VERY IMPORTANT)

• `x ^ x = 0`
• `x ^ 0 = x`
• XOR reversible hota hai

✅ Full XOR Swap Code

```
void swapXOR(int &a, int &b){


    a = a ^ b;

    // a = (a ^ b)


    b = a ^ b;

    // b = (a ^ b) ^ b = a


    a = a ^ b;

    // a = (a ^ b) ^ a = b

}
```

🧠 Kaise kaam kar raha hai?

Initial:

```
a = 10, b = 20
```

}

After step 1:

```
a = 10 ^ 20
```

After step 2:

```
b = (10 ^ 20) ^ 20 = 10
```

After step 3:

```
a = (10 ^ 20) ^ 10 = 20
```

Final:

```
a = 20, b = 10
```

📌 **Why use XOR?**
• Extra memory nahi
• Low-level systems
• Embedded / kernel code

🔢 PRIME NUMBER (FUNCTION)

```
bool isPrime(int n){

    if(n < 2) return false;

    for(int i = 2; i * i <= n; i++){

        if(n % i == 0) return false;

    }

    return true;

}
```

📌 `sqrt(n)` tak check = optimization

🔢 ARMSTRONG NUMBER

```
bool isArmstrong(int n){

    int o = n, s = 0;

    while(n){

        int d = n % 10;

        s += d*d*d;

        n /= 10;

    }

    return s == o;

}
```

🧠 WHAT FUNCTIONS REALLY GIVE YOU

Problem	Solution
Repetition	Reuse
Bugs	Isolation
Messy code	Clean design
Scaling	Modularity

🏁 FINAL SUMMARY

✓ Program starts from `main()`
✓ Function = logic capsule
✓ Parameters = input
✓ Return = output
✓ Reference = original change
✓ Overloading = compile time
✓ Default params = flexibility
✓ XOR = memory optimized
✓ Functions = real software foundation

🧠🔥 LIFE-TIME TIPS

- Logic repeat ho → function banao
- Ek function = ek responsibility
- Readability > cleverness
- Interview me **WHY explain karo**

✨ Made By — Harshal Chauhan ✨
