🔢 Lecture 04 — Pattern Advanced

✨ 0–1 Alternate Pattern

🎯 Problem Statement

Hume ek pattern print karna hai jisme **0 aur 1 alternate ho**,
aur **har row ka starting digit row number par depend kare.**

```
1

01

101

0101

10101
```

🧠 First Principle Thinking

Pattern kabhi direct code se nahi banta,
pehle soch samajh ke rule nikala jata hai.

🔍 Teen basic questions pucho:

- **Rows kitni?** → 5
- **Har row me elements?** → row number ke barabar
- **Digits ka rule?** → 0 aur 1 alternate

🔑 Key Observation

Row number sab kuch decide kar raha hai 👇

- **Odd Row (1, 3, 5 …)** → start with **1**
- **Even Row (2, 4 …)** → start with **0**

👉 Is rule ko ek line me likh sakte hain:

```
row % 2
```

🧩 Why Boolean?

Boolean sirf **2 values** rakhta hai:

`true  → 1`

`false → 0`

Aur `!` operator value ko **flip** kar deta hai:

`1 → 0`

`0 → 1`

👉 Alternate pattern ke liye **perfect fit.**

🧪 Dry Run (Mind Clear Ho Jaye)

Row = 1

`row % 2 = 1 → start = 1`

`Output: 1`

Row = 2

`row % 2 = 0 → start = 0`

`Output: 01`

Row = 3

`row % 2 = 1 → start = 1`

`Output: 101`

👉 Ab pattern ka behaviour crystal clear.

🧱 Algorithm (Logic Flow)

1. Outer loop → rows control kare
2. `bool num = row % 2`
3. Inner loop → row times print
4. Har print ke baad → `num = !num`
5. Row end → `endl`

💻 Clean Code (Interview-Ready)

```
for(int row = 1; row <= 5; row++){


    bool num = row % 2;

    // Odd row → 1

    // Even row → 0


    for(int col = 0; col < row; col++){

        cout << num;

        num = !num;   // 1 → 0, 0 → 1

    }


    cout << endl;

}
```

⏱️ Complexity Analysis

- **Time Complexity:** $O(n^2)$
- **Space Complexity:** $O(1)$

🧠 Interview Lock 🔒

> "I used boolean flipping to avoid condition-heavy logic
> and determined the starting value using row parity."

✅ What You Actually Learned

✓ Pattern = observation first
✓ Boolean + NOT operator = elegant solution
✓ Clean logic > long if-else chains

✨ Character Pattern (A, AB, ABC…)

🎯 Problem Statement

Hume ek aisa pattern print karna hai jisme
har row me characters A se start ho kar row ke length tak print hon.

```
A

AB

ABC

ABCD

ABCDE
```

🧠 First Principle Thinking

Sabse pehle ye samjho:

• Rows = 5
• Har row me characters = row number
• Starting character hamesha **'A'**

👉 Pattern **numbers ka nahi**, **characters ka hai**
lekin logic numbers jaisa hi chalega.

🔑 Core Concept — ASCII

C++ me **char bhi internally number hota hai.**

```
'A' → 65

'B' → 66

'C' → 67
```

👉 Iska matlab:

`'A' + 1 = 'B'`

`'A' + 2 = 'C'`

🧩 Observation

Row decide karti hai **kitne characters print honge.**

Row	Characters
1	A
2	A B
3	A B C
4	A B C D

👉 Har row me:
• Start = `'A'`
• End = `'A' + row - 1`

🧪 Dry Run (Concept Lock)

Row = 3

`'A' → 'B' → 'C'`

`Output: ABC`

Row = 5

`'A' → 'B' → 'C' → 'D' → 'E'`

`Output: ABCDE`

👉 Ab logic bilkul clear hai.

📦 Algorithm (Simple & Clean)

1️⃣ Outer loop → rows
2️⃣ Inner loop → `'A'` se current row tak
3️⃣ Har character print
4️⃣ Row complete → `endl`

💻 Clean Code

```
for(char row = 'B'; row <= 'F'; row++){

    for(char ch = 'A'; ch < row; ch++){

        cout << ch;

    }

    cout << endl;

}
```

🔍 **Why 'B' to 'F'?**
Because:

```
Row 'B' → prints A

Row 'C' → prints AB
```

⏱️ Complexity

- **Time:** $O(n^2)$
- **Space:** $O(1)$

🧠 Interview Lock 🔒

> “Characters in C++ follow ASCII values,
> so pattern problems can be solved using arithmetic on chars.”

✅ What You Learned

✓ Char ≠ magic, char = number
✓ Patterns = loops + observation
✓ Clean bounds > complex logic

🔁 Same Letter Repeat Pattern

🎯 Problem Statement

Hume ek aisa pattern banana hai jisme
har row me ek hi character repeat ho,
aur **repeat count = row number.**

```
A

BB

CCC

DDDD

EEEEE
```

🧠 First Principle Thinking

Pattern ko tod ke samjho:

• Rows = 5
• Har row ka **character alag**
• Har row ka **repeat count alag**

👉 Ek row = **ek character × row number**

🔑 Key Observation

Row	Character	Times
1	A	1
2	B	2
3	C	3

4	D	4
5	E	5

👉 Character change ho raha hai
👉 Repeat count bhi change ho raha hai

🔤 Char Calculation (ASCII Lock 🔒)

C++ me:

```
'A' + 0 = A

'A' + 1 = B

'A' + 2 = C
```

👉 **Row number se hi character mil sakta hai**:

```
char ch = 'A' + row - 1;
```

🧪 Dry Run (Concept Clear)

Row = 3

```
ch = 'A' + (3 - 1) = 'C'

Print 3 times → CCC
```

Row = 5

```
ch = 'A' + (5 - 1) = 'E'

Print 5 times → EEEEE
```

👉 Ab koi confusion nahi bachta.

🧱 Algorithm (Step-by-Step)

1️⃣ Outer loop → rows (1 to 5)
2️⃣ Character calculate karo (`'A' + row - 1`)
3️⃣ Inner loop → row times print
4️⃣ New line

💻 Clean Code

```
for(int row = 1; row <= 5; row++){


    char ch = 'A' + row - 1;


    for(int col = 0; col < row; col++){

        cout << ch;

    }


    cout << endl;

}
```

⏱️ Complexity

- **Time:** $O(n^2)$
- **Space:** $O(1)$

🧠 Interview Lock 🔒

> “Character patterns can be derived using ASCII arithmetic,
> where row index decides both character and frequency.”

✅ What You Learned

✓ Row se character nikala
✓ Row se repeat count decide ki
✓ Logic simple, output powerful

🔢 Number Pyramid Pattern ⭐

🎯 Problem Statement

Hume ek **centered number pyramid** banana hai:

```
   1
  121
 12321
1234321
123454321
```

👉 Ye pattern **spacing + increasing + decreasing** ka perfect mix hai.

🧠 First Principle Thinking

Har row ko tod ke dekho 👇

👉 **Ek row me 3 parts hote hain**:

1️⃣ Spaces (left side)
2️⃣ Increasing numbers (1 → n)
3️⃣ Decreasing numbers (n-1 → 1)

🔍 Row-Wise Breakdown

Row 1

```
Spaces: 4
```

```
Numbers: 1
```

Row 3

```
Spaces: 2
```

```
Increasing: 1 2 3

Decreasing: 2 1
```

Row 5

```
Spaces: 0

Increasing: 1 2 3 4 5

Decreasing: 4 3 2 1
```

👉 Clear pattern dikh raha hai:

- Row badhti hai → spaces ghatte hain
- Numbers pehle badhte, phir ghatte

📐 Math Behind Pattern

Agar total rows = **5**, aur loop row = **r (0-based)**

• Spaces = `row`
• Max number = `5 - row`

👉 Isliye outer loop **ulta** chalaya gaya hai.

🧱 Algorithm (Step-by-Step)

1️⃣ Outer loop → rows (4 → 0)
2️⃣ Print spaces
3️⃣ Print increasing numbers
4️⃣ Print decreasing numbers
5️⃣ New line

💻 Clean Code (Lecture-Level)

```
for(int row = 4; row >= 0; row--){


    // Step 1: spaces

    for(int s = 0; s < row; s++){

        cout << " ";
```

```
    }

    // Step 2: increasing numbers
    for(int num = 1; num <= 5 - row; num++){
        cout << num;
    }

    // Step 3: decreasing numbers
    for(int num = 5 - row - 1; num >= 1; num--){
        cout << num;
    }

    cout << endl;
}
```

🧪 Dry Run (Concept Clear)

Row = 2

`Spaces → 2`

`Inc → 1 2 3`

`Dec → 2 1`

`Output →  12321`

👉 Agar ye dry run samajh aaya
to **koi bhi pyramid pattern easy** 💯

⏱️ Complexity

- **Time:** $O(n^2)$
- **Space:** $O(1)$

🧠 Interview Lock 🔒

> “Complex patterns are just a combination of
> spaces + increasing loop + decreasing loop.”

✅ What You Learned

✓ Pattern ko parts me todna
✓ Spaces ka control
✓ Mirror logic (increase + decrease)
✓ Pyramid = confidence booster 🔥

🧠 Logic Building Part

🟡 Prime Number Check 🤔

🎯 What is a Prime Number?

Prime number wo hota hai jo:

✓ 2 se bada ho
✓ sirf 1 aur khud se divisible ho

📌 Examples:

- 2, 3, 5, 7 → Prime
- 4 (2×2), 6 (2×3), 9 (3×3) → Not Prime

🧠 First Principle Thinking

Agar kisi number **n** ko prime check karna hai, to socho:

👉 Kya **2 se leke (n-1)** ke beech
koi aisa number hai jo **n ko divide** kar de?

Agar **ek bhi mil gaya** → ❌ Not Prime
Agar **koi nahi mila** → ✅ Prime

📦 Logic Steps

1️⃣ Agar `n < 2` → Not Prime
2️⃣ 2 se `(n-1)` tak loop chalao
3️⃣ Agar `n % i == 0` → Not Prime
4️⃣ Loop poora ho gaya → Prime

💻 Clean Code

```
int n;

cin >> n;


if(n < 2){

    cout << "Not Prime";

    return 0;

}


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

    if(n % i == 0){

        cout << "Not Prime";

        return 0;

    }

}


cout << "Prime Number";
```

🧪 Dry Run (n = 7)

- 7 % 2 ❌
- 7 % 3 ❌
- 7 % 4 ❌

- 7 % 5 ❌
- 7 % 6 ❌
  👉 Koi divisor nahi mila → **Prime**

🎯 Interview Lock 🔒

> “Prime check means searching for a divisor.
> If none exists, the number is prime.”

🟢 SUM OF DIGITS 🔢

🎯 Problem

Given number:

`345128`

Output:

`3 + 4 + 5 + 1 + 2 + 8 = 23`

🧠 First Principle Thinking (Real Life)

Socho number ek **stack of coins** jaisa hai:

- Sabse upar wali coin → **last digit**
- Use nikalne ke liye → `% 10`
- Use hataane ke liye → `/ 10`

📦 Logic

1️⃣ Last digit nikalo → `num % 10`
2️⃣ Sum me add karo
3️⃣ Digit hatao → `num / 10`
4️⃣ Jab tak number 0 na ho jaaye

💻 Clean Code

```
int num = 345128;

int sum = 0;


while(num > 0){

    int lastDigit = num % 10;

    sum = sum + lastDigit;

    num = num / 10;

}


cout << sum;
```

🧪 Dry Run

num	lastDigit	sum
345128	8	8
34512	2	10
3451	1	11
345	5	16
34	4	20

3 3 23

👉 num = 0 → loop stop

🔁 WHILE LOOP — First Principle + Real Life

❓ Why While When For Exists?

Key Rule:

🟦 **FOR loop** → jab iterations pehle se pata ho
🟩 **WHILE loop** → jab iterations pata hi na ho

🧠 Real Life Example

🚰 Water Tank Example

```
Jab tak tank empty na ho:

    paani nikaalte raho
```

👉 Tumhe nahi pata:

- Kitni baar tap kholna padega
- Kab last paani niklega

➡️ **WHILE loop perfect fit**

💡 Why Sum Of Digits Can't Use FOR?

❌ FOR loop ko chahiye:

```
kitni baar loop chalega
```

❌ Digits ki count pehle nahi pata

✓ WHILE loop bolta hai:

```
jab tak number > 0 hai, kaam karo
```

🧠 Comparison

Situation	Loop
Fixed count (1–10)	for
Unknown count	while
Condition based	while
Digit processing	while
User input based	while

🤌 Interview Gold Lines

> “When the number of iterations is unknown,
> while loop models the problem naturally.”

✅ Final Lock — What You Learned

✓ Prime logic = divisor search
✓ Digit problems = while loop
✓ %10 and /10 are digit tools
✓ for ≠ while (different purpose)
✓ Logic > syntax