Topic: Conditional Statements & Multiple Conditions

---

1. Warm-up (2 mins)

Objective:
Help students connect real-life decision-making with programming logic.

Activity Example:
Ask:

- “If it’s raining, what do you do?”
- “If you wake up late, what happens next?”

Use this to explain that **computers make decisions** based on given conditions — this is the essence of conditional statements.

---

2. Introduction to Conditionals

Concept:

Conditional statements allow the program to decide which block of code to execute based on a condition’s truth value (true/false).

Syntax:

```
if (condition) {
    // code runs if true
}
else {
    // code runs if false
}
```

Example:

```
int age = 20;

if (age >= 18) {
```

```
    cout << "You are an adult.";
}
else {
    cout << "You are a minor.";
}
```

Dry Run:

Step	Variable	Condition Checked	Result	Output
1	age = 20	age >= 18	true	“You are an adult.”

---

3. if, else if, and else

Concept:

Use `else if` when multiple outcomes are possible.
Only one block executes — the first one whose condition is true.

Example:

```
int marks = 76;

if (marks >= 90)
    cout << "A Grade";
else if (marks >= 80)
    cout << "B Grade";
else if (marks >= 70)
    cout << "C Grade";
else
    cout << "Fail";
```

Dry Run:

Step	marks	Condition	True/False	Output
1	76	marks >= 90	false	—
2	76	marks >= 80	false	—

3	76	marks >= 70	true	“C Grade”

---

4. Nested if-else (2 mins)

Concept:

A nested if-else means having one if-statement inside another.
Used when one decision depends on another.

Example:

```
int marks = 82;
int attendance = 74;

if (marks >= 50) {
    if (attendance >= 75) {
        cout << "You passed!";
    }
    else {
        cout << "You failed due to low attendance.";
    }
}
else {
    cout << "You failed due to low marks.";
}
```

Dry Run:

Step	marks	attendance	Outer If	Inner If	Output
1	82	74	true	false	“You failed due to low attendance.”

---

5. Multiple Conditions (5 mins)

Concept:

When more than one condition must be checked, use logical operators.

Operator	Meaning	Example	True When
`&&`	AND	`age > 18 && marks > 50`	Both are true
`	`	`	`	OR
`!`	NOT	`!(x > 10)`	Condition is false

Example:

```
int age = 20;
string nationality = "Pakistani";

if (age >= 18 && nationality == "Pakistani") {
    cout << "Eligible to vote.";
}
else {
    cout << "Not eligible.";
}
```

Dry Run:

Step	age	nationality	Condition	Result	Output
1	20	Pakistani	20 >= 18 && Pakistani == Pakistani	true && true	“Eligible to vote.”

---

6. Operator Precedence (2 mins)

Concept:

When multiple operators are used, precedence defines which executes first.

Order (High → Low):

1. `!` (NOT)
2. `&&` (AND)

3. `||` (OR)

Example:

```
int a = 5, b = 10, c = 15;

if (a > 2 || b > 15 && c < 20)
   cout << "Condition True";
else
   cout << "Condition False";
```

Dry Run:

1. Evaluate `b > 15 && c < 20` → `false && true` → `false`
2. Then `a > 2 || false` → `true || false` → `true`
3. Output: `Condition True`

---

7. Common Logical Errors and Debugging (5 mins)

Frequent Errors:

Using `=` instead of `==`

```
 if (x = 5) // Wrong
if (x == 5) // Correct
```

1.
2. Missing or incorrect parentheses.
3. Overlapping or unreachable conditions.

Debugging Tips:

- Use print statements to trace execution.

- Test with boundary values (49, 50, 51 for a pass mark).
- Keep indentation clear for readability.

What to Teach:

- Conditions behave differently when **boundary values** are used.
- Teach students to test both **sides of the condition**.

Examples:

Condition	Input	Why It's an Edge Case
`if (marks >= 90)`	marks = 90	Boundary value — equals threshold
Nested ifs	multiple true conditions	Check which executes first
Logical operators	`(a > b	
Boolean logic	`!(true && false)`	Mix NOT, AND, OR for confusion

Tricky Example:

```
if (x > 5 || y++ > 2) cout << y;
```

Edge case: when `x > 5` is true — `y++` never runs!

4. Arithmetic / Logical Operations Edge Cases

Operation	Edge Case	Explanation
Division	divide by 0	Program crash
Modulus	negative numbers	Language-dependent behavior

Overflow	very large ints	Exceeds data type limit
Floating comparison	`if (a == b)`	Floating point rounding issues

1. The Short-Circuit Trap

```
int a = 5, b = 2, c = 10;

if (a > 10 && ++b > 2 || c == 10)
    cout << "YES " << b;
else
    cout << "NO " << b;
```

Dry Run:

Step	Expression	Result	Notes
1	`a > 10`	false	So `&&` short-circuits → `++b` not executed
2	`false && ++b > 2` → false	-	
3	`false	`c == 10` → true
Final:	true	Executes first branch	

Output: `YES 2`

> ⚠️ **Concepts Tested:**
> Short-circuiting, logical precedence (`&&` before `||`), side effects (`++b` not executed).

---

2. The Nested NOT Confusion

```
bool x = true, y = false;

if (!x && !y || !(x || y))
    cout << "A";
else
    cout << "B";
```

Step-by-step Evaluation:

1. `!x && !y` → `false && true` → `false`
2. `x || y` → `true || false` → `true`
3. `!(x || y)` → `false`
4. `false || false` → `false`

Output: `B`

> ⚠️ **Concepts Tested:**
> Nested negations, mixed precedence, and the importance of parentheses.
> Many students mentally misread `!x && !y || !(x || y)` as all "nots" → think it's true when it's not.

---

3. The Ambiguous Else

```
int x = 5, y = 10;

if (x > 0)
   if (y < 5)
      cout << "Case 1";
   else
      cout << "Case 2";
```

Dry Run 1:

x	y	Evaluation	Output
5	10	Outer if true → inner if false → else executes	Case 2
-5	10	Outer if false → skips all	(no output)

Output: `Case 2`

⚠️ **Concepts Tested:**
Dangling `else` — it always binds to the **nearest unmatched if**.
Adding `{}` changes behavior dramatically.

Modified Version (for demonstration):

```
if (x > 0) {
   if (y < 5)
      cout << "Case 1";
} else
   cout << "Case 2";
```

Output now: *(no output)* because the `else` now pairs with the **outer if**.

Great live demo: run both versions and show how one brace changes logic completely.

---

4. The Parentheses Puzzle

```
int a = 3, b = 4, c = 5;

if (a + b > c && b - c < a || !(c - a <= b))
   cout << "YES";
else
   cout << "NO";
```

Step-by-step:

1. `a + b > c` → `7 > 5` → `true`
2. `b - c < a` → `-1 < 3` → `true`
3. `(a + b > c && b - c < a)` → `true && true` → `true`
4. `c - a <= b` → `2 <= 4` → `true`
5. `!(c - a <= b)` → `false`
6. Final: `true || false` → `true`

Output: `YES`

Edge Variation (for tricking students):

Change `a = 3, b = 1, c = 5`

1. `a + b > c` → `4 > 5` → false
2. `b - c < a` → `-4 < 3` → true
3. `(false && true)` → false
4. `c - a <= b` → `2 <= 1` → false
5. `!(false)` → true
6. `false || true` → true

Output: still `YES`

> ⚠️ **Concepts Tested:**
> Operator precedence (`&&` before `||`), negation, boundary cases, and relational reasoning.

How to Use These in Your Workshop

1. **Show the code first**, ask students to **predict the output** — no running.
2. Make them **dry-run step by step**, writing each Boolean result.
3. Then reveal output — and discuss *why*.
4. Finally, ask: *"What would you change to flip the output?"*
   (This teaches debugging & logical thinking.)

8. Practice Problems

Easy

1. **Even or Odd:**
   Input a number; print whether it's even or odd.
   (Use modulus operator `%`)

2. **Positive, Negative, or Zero:**
   Check whether an input number is positive, negative, or zero.

3. **Pass or Fail:**
   Input marks; print "Pass" if ≥ 50, otherwise "Fail."

Medium

4. **Grading System:**
   Input marks and display grade (A, B, C, D, or F) using else-if chain.

5. **Age Category:**
   Input age → print "Child" (0–12), "Teen" (13–19), "Adult" (20–59), "Senior" (60+).

6. **Login Simulation:**
   Check if entered username and password match preset values.

Hard

7. **Scholarship Eligibility:**
   Input marks and attendance:

   - Marks ≥ 85 and attendance ≥ 80 → Eligible
   - Marks ≥ 70 and attendance ≥ 90 → Partial Scholarship
   - Else → Not eligible

8. **Electricity Bill Calculator:**
   Based on units:

- <100 units: Rs. 10/unit
- 100–200: Rs. 15/unit
- 200–500: Rs. 20/unit
- 500: Rs. 25/unit
  Add 10% tax if total > 3000.

9. **Triangle Type Finder:**
   Input three sides.
   - If triangle is valid, check if it's Equilateral, Isosceles, or Scalene.
     (Hint: For a valid triangle, sum of any two sides > third side)

10. **Student Result Evaluation:**
    Input marks of 3 subjects.

- Calculate average.
- If average ≥ 80 and no subject < 50 → "Excellent"
- Else if average ≥ 60 → "Good"
- Else → "Needs improvement."

---

9. Wrap-up (5 mins)

Recap:

- Purpose of conditional statements.
- Use of `if`, `else if`, `else`, and nested structures.
- Combining conditions using `&&`, `||`, and `!`.
- Avoiding logical and syntax errors.

```
#include <iostream>
using namespace std;

int main() {
    int a = 1, b = 2, c = 3;

    if (a++ > --b)
        if (b++ > c--)
            cout << a + b + c;
        else if ((a == 2 && c == 2) || (b = a + c) && b > 3)
            cout << b - a;
        else
            cout << c;
    else if (a == 2 || b == 1 && c == 3)
        cout << a * b * c;
    else
        cout << "X";
}
```