Basic Algorithmic Structures

The three fundamental building blocks of all software logic.

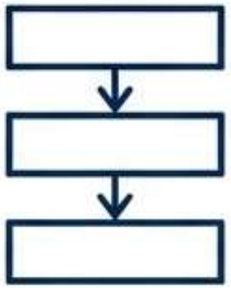
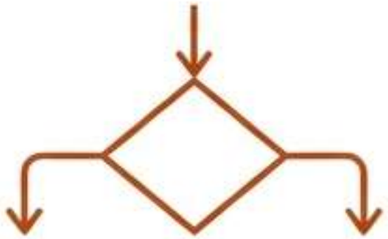
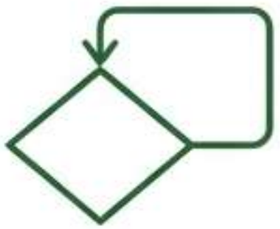
Learning Objectives

- **11.5.3.9** Describe and apply the sequential structure
- **11.5.3.10** Describe and apply the branching structure
- **11.5.3.11** Describe and apply the loop structure

Session Roadmap

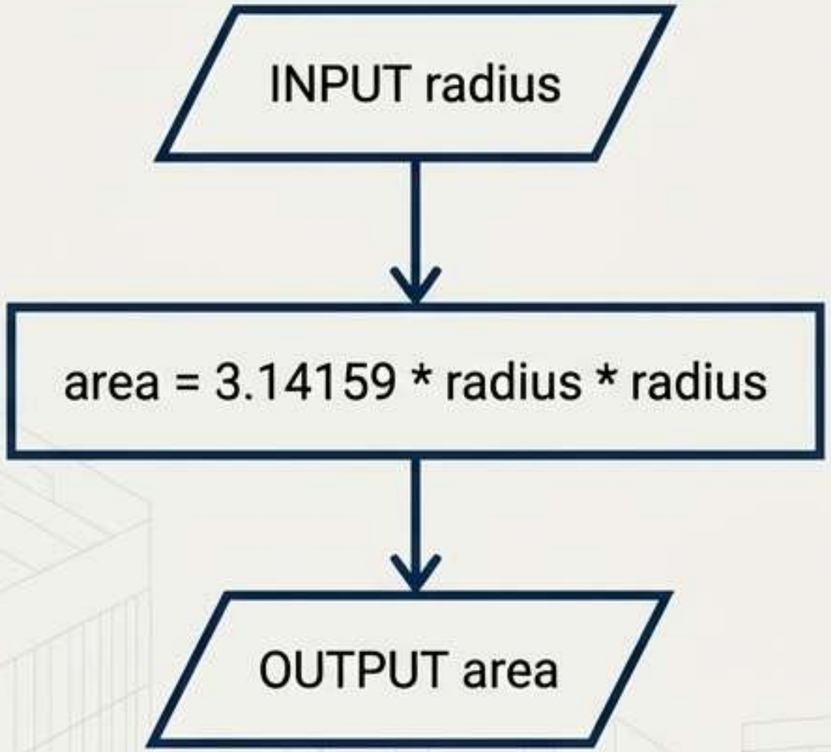
- [x] 1. Theory & Syntax
- [x] 2. Architectural Examples
- [x] 3. Multi-Level Tasks & Diagnostic Questions (Bloom's Taxonomy)

The Concept Matrix: Logic as a Recipe

Real World	Flowchart	Code Logic
 Doing steps in order: Chop, mix, cook.		**Sequence** Straight-line execution. Top to bottom.
 Decision: IF sauce is thick, THEN add water.		**Branching (Selection)** Alternative paths based on a condition.
 Repetition: Stir every 2 minutes UNTIL golden.		**Iteration (Loops)** Repeating statements while a condition is met.

Brick 1: Sequential Structure

Statements are executed one after another, top to bottom, left to right. No decisions, no repetitions.



Pseudocode

```
INPUT radius
area ← 3.14159 * radius * radius
OUTPUT "Area = ", area
```

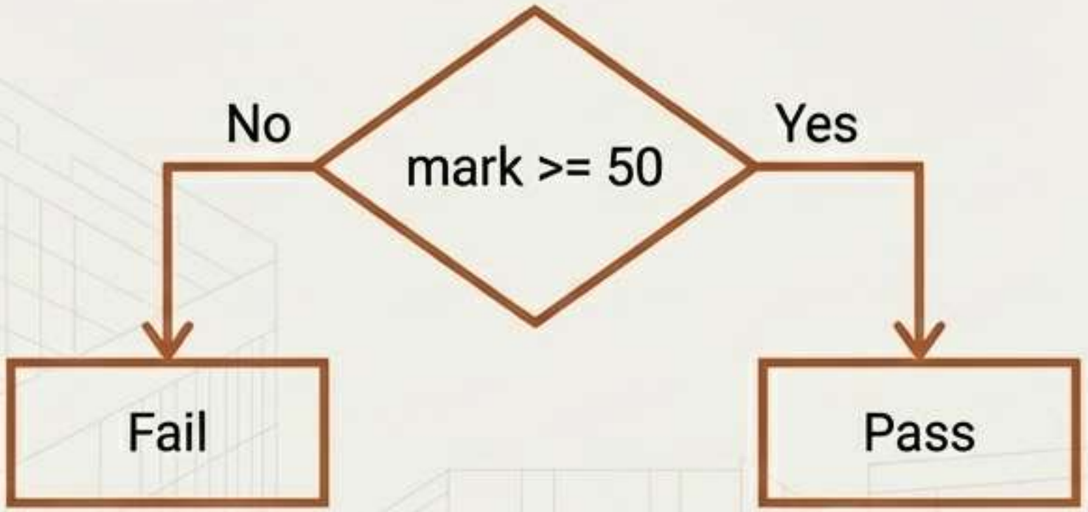
C++

```
double radius;
cin >> radius;
double area = 3.14159 * radius * radius;
cout << "Area = " << area << endl;
```

Brick 2: Branching Structure

The program makes a decision based on a condition. If TRUE, one path is taken; otherwise, an alternative path is followed.

- Simple (IF...THEN)
- Two-way (IF...ELSE)
- Multi-way (CASE)
- Multi-way (CASE)
- Nested (IF inside IF)



Pseudocode

```
INPUT mark
IF mark >= 50 THEN
   OUTPUT "Pass"
ELSE
   OUTPUT "Fail"
ENDIF
```

C++

```
int mark;
cin >> mark;
if (mark >= 50) { cout << "Pass" << endl; }
else { cout << "Fail" << endl; }
```

Multi-way Branching

Handling several alternatives cleanly without deep nesting

Pseudocode (CASE)

```
INPUT grade
CASE OF grade
  "A": OUTPUT "Excellent"
  "B": OUTPUT "Good"
  "C": OUTPUT "Satisfactory"
  OTHERWISE: OUTPUT "Below standard"
ENDCASE
```

C++ (switch)

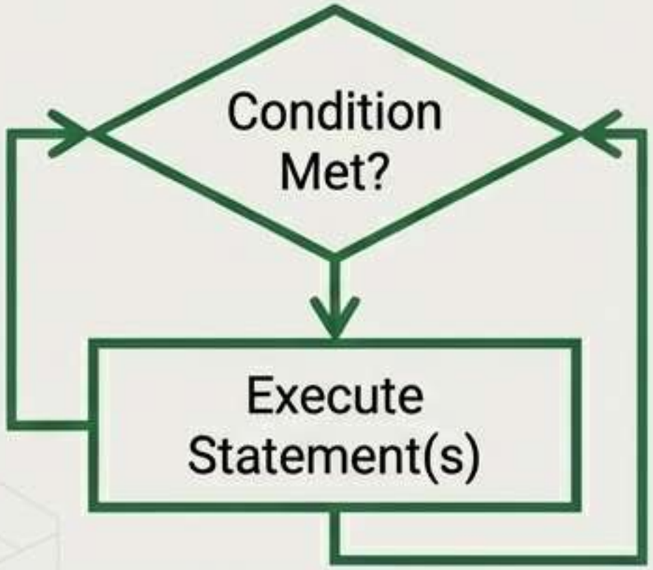
```
char grade; cin >> grade;
switch (grade) {
  case 'A': cout << "Excellent"; break;
  case 'B': cout << "Good"; break;
  case 'C': cout << "Satisfactory"; break;
  default: cout << "Below standard";
}
```

Mandatory in C++ to prevent falling through cases.

Brick 3: Loop Structure (Iteration)

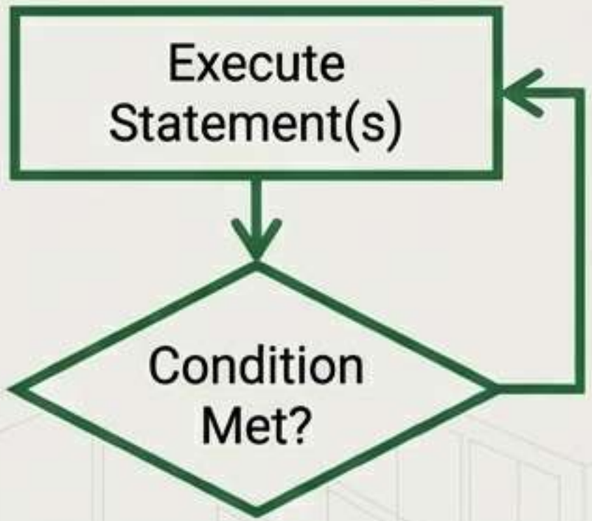
Repeating a group of statements while a condition is met.

Pre-condition Loop (WHILE)



Key Insight:
Condition is checked first.
May execute 0 times.

Post-condition Loop (REPEAT/UNTIL)



Key Insight:
Process happens first.
Always executes at least once.

The Loop Selection Matrix

Loop Type	Execution Guarantee	Pseudocode	C++ Keyword	When to Use This
Count-controlled	Exact number specified	`FOR…NEXT`	`for`	Known number of iterations (e.g., read exactly 10 inputs)
Pre-condition	0 or more times	`WHILE…ENDWHILE`	`while`	Condition checked before execution (e.g., read until Sentinel value)
Post-condition	1 or more times	`REPEAT…UNTIL`	`do…while`	Guaranteed execution needed (e.g., showing a Menu to a user)

Pro-Tip: Draw a flowchart before you code. Visualizing the logic prevents infinite loops.

The Algorithmic Architecture Glossary

Every algorithm, no matter how complex, can be mapped using only these five symbols connected in Sequences, Branches, and Loops.

Architect's Legend

Start / End

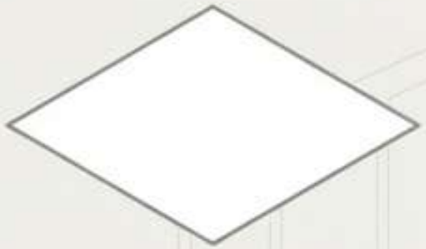
Identifies program boundaries.

Process

Used for calculations, variable assignments (Sequence).

Input / Output

Reading data or displaying results.



Decision

Two exits (Yes/No). The core of Branching and Loops.

Flow Direction

The tracks that connect the logic.

Synthesis in Action: Temperature Classifier

Iterates exactly 5 times (Loop).

Executes in straight line (Sequence).

Classifies each individual input (Branching).

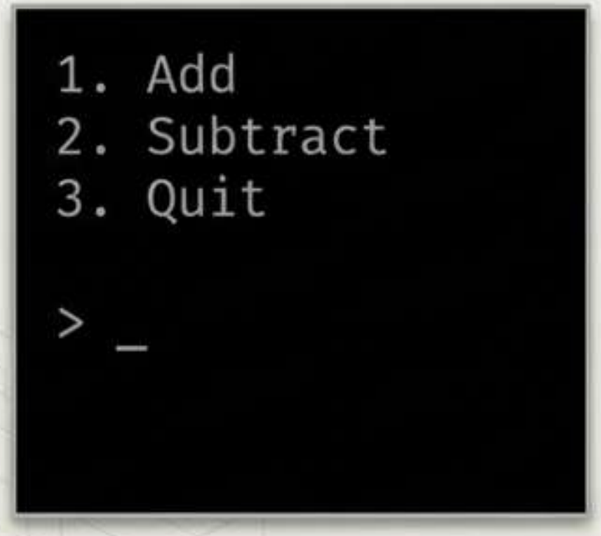
```
FOR i ← 1 TO 5
    OUTPUT "Enter temperature: "
    INPUT temp
    IF temp < 10 THEN
        OUTPUT "Cold"
    ELSE IF temp < 25 THEN
        OUTPUT "Warm"
    ELSE
        OUTPUT "Hot"
    ENDIF
NEXT i
```

C++ Equivalent

```
for (int i = 1; i <= 5; i++) { … }
```

Pattern: Menu-Driven Interface

Synergizing REPEAT/UNTIL with CASE logic.



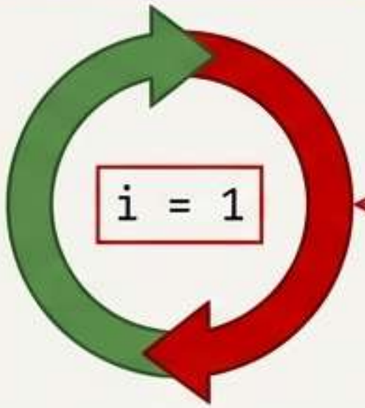

```
REPEAT
  OUTPUT "1. Add  2. Subtract  3. Quit"
  INPUT choice
  CASE OF choice
    1: OUTPUT "Result = ", a + b
    2: OUTPUT "Result = ", a - b
    3: OUTPUT "Goodbye!"
    OTHERWISE: OUTPUT "Invalid choice"
  ENDCASE
UNTIL choice = 3
```

Guarantees the menu shows at least once.

Catches bad user input safely.

Debugging Dashboard: Structural Failures

Failure A: The Infinite Loop

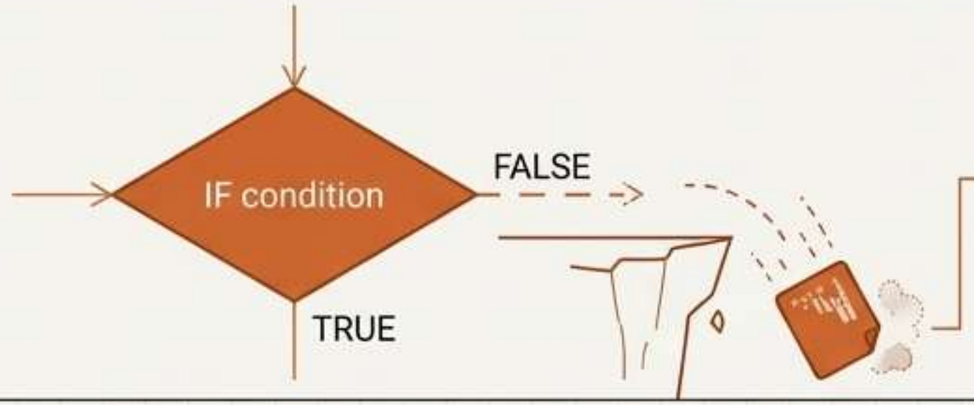


Code snippet: WHILE `i < 5` without any `i ← i + 1` inside the loop body.

Concept: Forgetting to update the loop variable.

Fix: Always ensure the condition will eventually evaluate to FALSE.

Failure B: Missing Alternatives

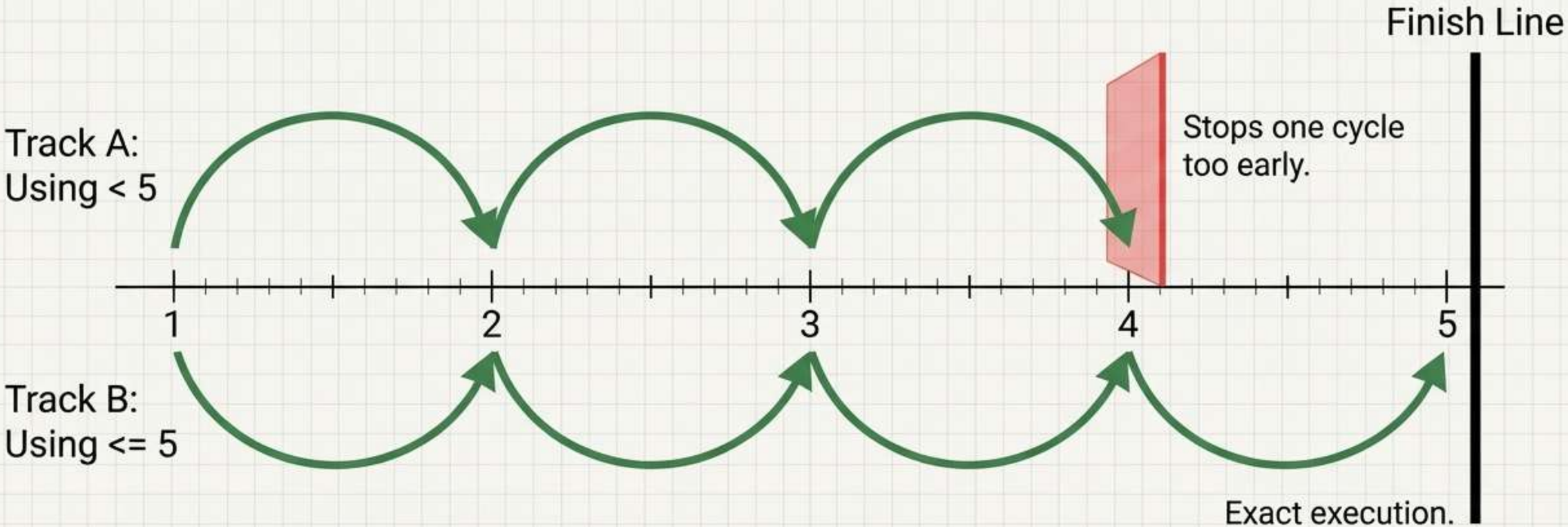


Concept: Forgetting the default or otherwise case.

Fix: Always include an ELSE or OTHERWISE statement in branching to handle unexpected inputs securely.

Debugging Dashboard: **Off-by-One Errors**

Visualizing the mathematical boundary of a loop.



Analytical Takeaway: Always mentally trace the very first and the very last iteration of your loop to verify boundary conditions.

The Proving Ground: **Foundation**

Multi-Level Tasks & Questions

Remember

Name the three basic algorithmic structures and give one real-life example of each.

(Hint: Think back to the kitchen recipe analogy.)

Understand

Explain the exact difference between a pre-condition loop (WHILE) and a post-condition loop (REPEAT/UNTIL).

(Prompt: When would you explicitly choose to use one over the other?)

Apply

Write pseudocode that reads 10 numbers from a user, then outputs exactly how many of those numbers were even, and how many were odd.

(Constraint: Must successfully combine sequence, a loop, and branching logic.)

The Proving Ground: **Mastery**

Multi-Level Tasks & Questions

Analyze

Scenario: A student wrote the following code:

```
FOR i ← 1 TO 10
  i ← i + 1
  OUTPUT i
NEXT i
```

The Task: Trace the execution manually. Explain mathematically what goes wrong when you manually increment the loop variable inside a count-controlled FOR loop.

Create

Scenario: The Higher/Lower Number Guessing Game.

Requirements: Design a program where the computer picks a random number (1-100). The user inputs guesses. The program replies ‘Higher’ or ‘Lower’ until the correct guess is made.

The Task: Write the complete algorithm using Pseudocode or C++. You must successfully implement all three structures: Sequence, Selection, and Iteration.

Diagnostic Self-Check

Rapid fundamental retention verification

Q1: Which structure executes statements one after another with strictly no decisions?

Answer: Sequential structure.

Q2: In flowchart architecture, what specific shape represents a decision point?

Answer: Diamond (rhombus).

Q3: Which specific loop type inherently guarantees it will execute at least once?

Answer: Post-condition loop (`REPEAT...UNTIL` / `do...while`).

Q4: What is the direct C++ equivalent of the pseudocode `CASE OF` multi-way branch?

Answer: The `switch` statement.