Session 3: Control Statements

1. Title + Big Idea (The Setup)

Topic Name: Control Statements

One-line purpose: To understand how to alter the normal sequential flow of program execution based on conditions or to repeat blocks of code.

2. Why Do We Need It? (The Problem)

Problem it solves: Without control statements, programs would execute linearly from start to finish, unable to make decisions, skip code, or repeat actions. This would make complex problem-solving impossible and programs inflexible.

Real-life analogy: Control statements are like traffic lights and road signs. They direct the flow of vehicles (program execution) based on conditions (red light, turn left sign) to ensure orderly and efficient movement to the destination.

3. Core Concept / Definition (The Truth)

Simple definition: Control statements are programming constructs that enable a program to make decisions, execute specific blocks of code repeatedly, or jump to different parts of the program. They dictate the order in which instructions are executed.

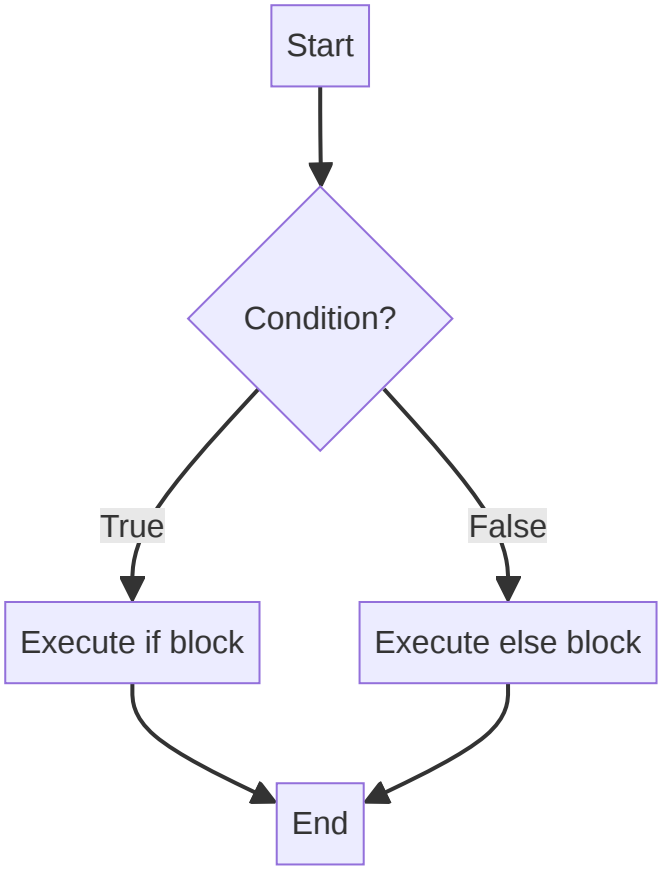
Key rule(s):

- Conditional statements (`if` , `else if` , `else` , `switch`) execute code blocks only if certain conditions are met.
- Looping statements (`while` , `for` , `do-while`) repeat code blocks until a condition is no longer met.

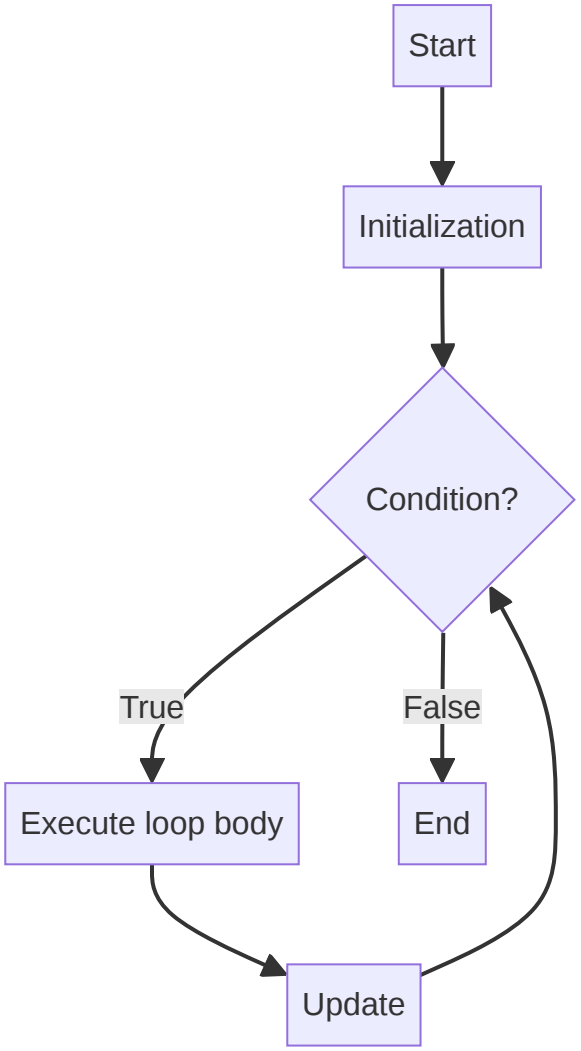
- Jump statements (`break` , `continue` , `goto` , `return`) alter the flow of control unconditionally.

4. Visual / Flow / Diagram (Show, Don’t Tell)

`if-else` Flowchart:



`for` Loop Flowchart:



5. Syntax / Structure (The Tool)

Conditional Statements:

- **`if` statement:**

```
if (condition) {
    // code to be executed if condition is true
}
```

- **`if-else` statement:**

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

- **`if-else if-else` ladder:**

```
if (condition1) {
    // code if condition1 is true
} else if (condition2) {
    // code if condition2 is true
} else {
    // code if no condition is true
}
```

- **`switch` statement:**

```
switch (expression) {
    case value1:
        // code block
        break;
    case value2:
        // code block
        break;
    default:
        // code block
}
```

Looping Statements:

- **`while` loop:**

```
while (condition) {
    // code to be executed as long as condition is true
}
```

- **`for` loop:**

```
for (initialization; condition; update) {
    // code to be executed repeatedly
}
```

- **`do-while` loop:**

```
do {
    // code to be executed at least once
} while (condition);
```

Jump Statements:

- **`break`:** Terminates the loop or `switch` statement.
- **`continue`:** Skips the current iteration of a loop and proceeds to the next.
- **`goto`:** Transfers control to a labeled statement (generally discouraged).

6. Example (The Action)

```c
#include <stdio.h>

int main() {
    int score = 75;

    // Example of if-else if-else
    if (score >= 90) {
        printf("Grade: A\n");
    } else if (score >= 80) {
        printf("Grade: B\n");
    } else if (score >= 70) {
        printf("Grade: C\n");
    } else {
        printf("Grade: F\n");
    }

    // Example of for loop
    printf("\nCounting from 1 to 5:\n");
    for (int i = 1; i <= 5; i++) {
        printf("%d ", i);
    }
    printf("\n");

    // Example of while loop with break
    int count = 0;
    printf("\nEven numbers up to 10 (using while and break):\n");
    while (1) { // Infinite loop
        if (count > 10) {
            break; // Exit loop when count exceeds 10
        }
        if (count % 2 == 0) {
            printf("%d ", count);
        }
        count++;
    }
    printf("\n");

    return 0;
}
```

7. Common Mistakes (The Conflict)

1. **Infinite Loops:** Forgetting to update the loop control variable or having a condition that never becomes false, causing the program to hang.
2. **Off-by-One Errors:** Incorrect loop bounds (e.g., `i < 5` instead of `i <= 5`) leading to one too many or one too few iterations.
3. **Missing `break` in `switch`:** Without `break`, execution falls through to the next `case` block (fall-through behavior), which is often unintended.
4. **Misunderstanding `continue`:** Confusing `continue` with `break`; `continue` skips the rest of the current iteration, while `break` exits the entire loop.

8. Key Points / Rules (The Takeaway)

- Control statements are fundamental for creating dynamic and responsive programs.
- `if`, `else if`, `else`, and `switch` statements handle decision-making based on conditions.
- `for`, `while`, and `do-while` loops enable repetitive execution of code blocks.
- `break` exits a loop or `switch`, `continue` skips to the next iteration of a loop.
- Careful construction of loop conditions and `switch` cases is crucial to avoid logical errors and infinite loops.

9. One-Line Summary (The Ending)

Control statements empower C programs to make decisions and perform repetitive tasks, enabling complex logic and interactive behavior beyond simple sequential execution.

10. (Optional) Practice Trigger

Question: Write a C program that asks the user for a number and then prints whether it is positive, negative, or zero using `if-else if-else`.

Variation Idea: Create a program that prints all even numbers from 1 to 20 using a `for` loop, but skips the number 10 using `continue`.