📚 C Decision & Loop Guide

Brief Overview

This note covering Decision and Loop Control Structures was created from a PDF document with 58 pages. It covers **Decision Statements**, **Loop Constructs**, break/continue, goto, and illustrative examples.

Key Points

- Decision Statements: if, if-else, nested, ladder, and switch.
- Loop Constructs: for, while, do-while with practical code snippets.
- Control flow modifiers: break, continue, and goto.
- Detailed flowcharts and examples for each construct.

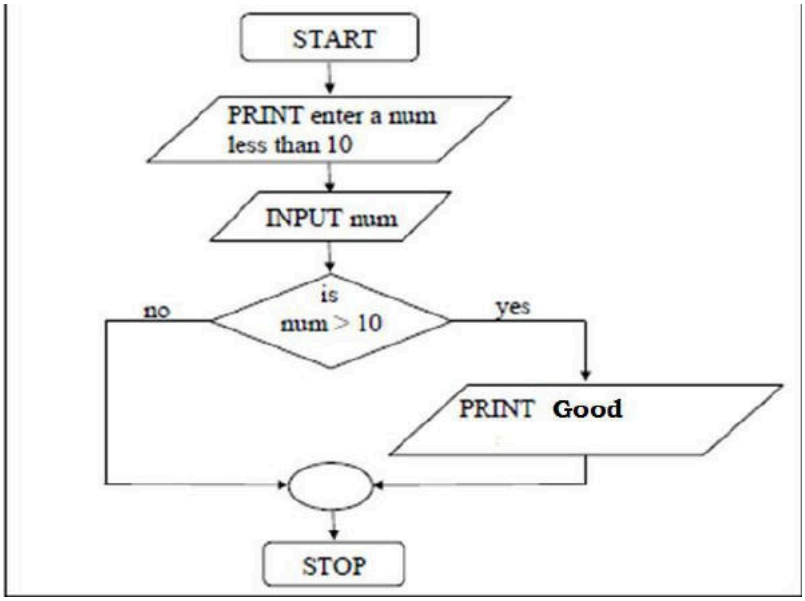
📊 Decision Control Structures

🔷 if Statement

> **Definition:** Executes a single statement (or block) only when the given condition evaluates to true.

- Condition must be inside parentheses: if (condition).
- Relational operators (==, !=, <, >, <=, >=) are used to form conditions.
- Any non-zero value is **true**; zero is **false**.

```
/* Demonstration of if statement */
void main() {
    int num;
    printf("Enter a number greater than 10: ");
    scanf("%d", &num);
    if (num > 10)
        printf("Good!!\n");
}
```



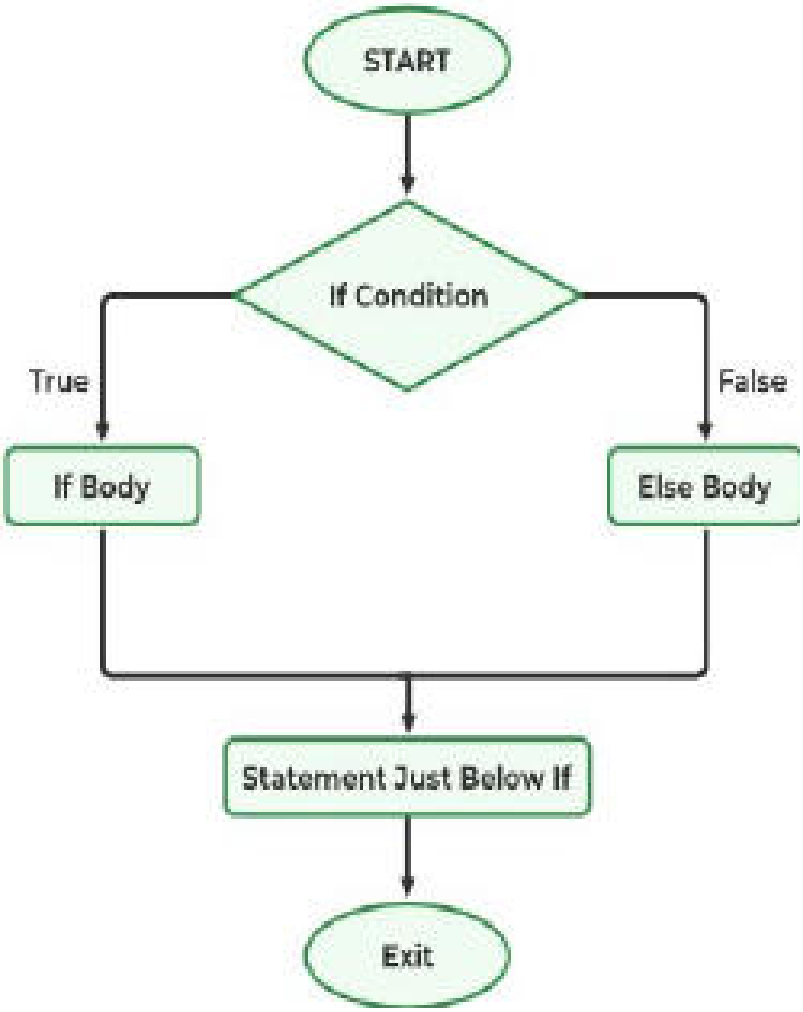
The flowchart shows the program start, input of a number, a decision diamond checking num > 10, printing "Good" when true, and looping back when false.

🔷 if-else Statement

> **Definition:** Provides two alternative blocks – one executed when the condition is true, the other when it is false.

Syntax

```
if (condition) {
    /* if-block */
} else {
    /* else-block */
}
```



The diagram illustrates the decision point "If Condition" leading to either the "If Body" or the "Else Body".

Key points

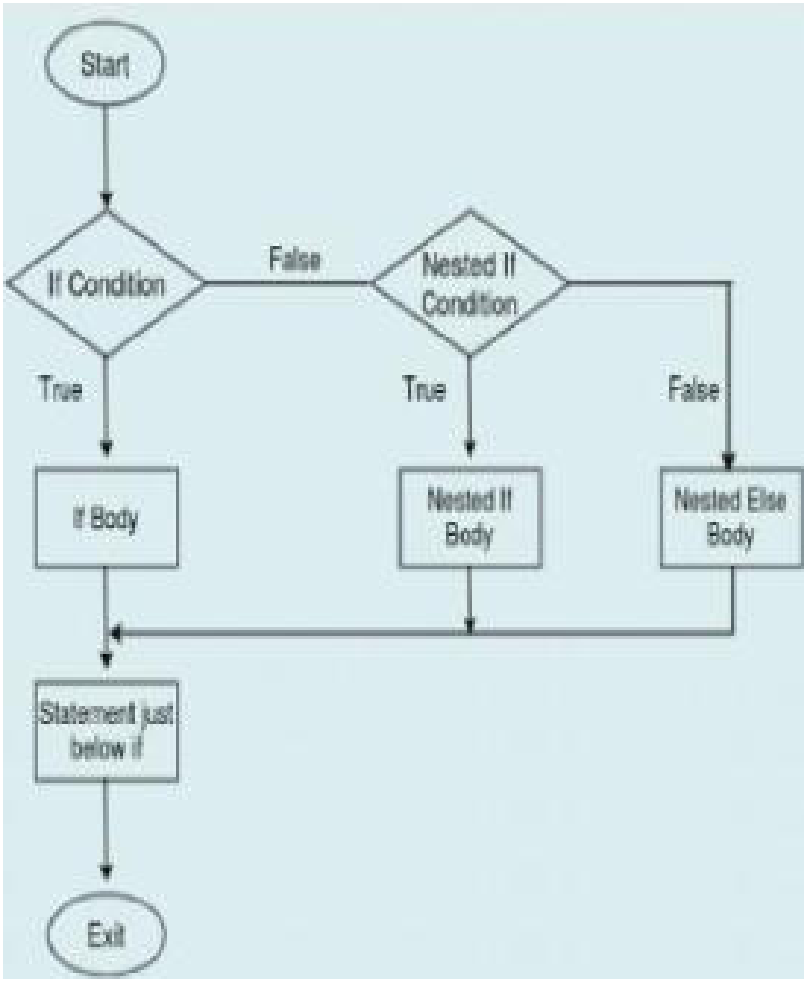
- The statements between if and else form the **if-block**; those after else form the **else-block**.
- Braces may be omitted when each block contains only one statement.

🔷 Nested if-else

> **Definition:** An if or else block that itself contains another complete if-else construct.

Syntax

```
if (condition1) {
    if (condition2) {
        /* code when both true */
    } else {
        /* code when condition1 true, condition2 false */
    }
} else {
    /* code when condition1 false */
}
```



Shows the primary decision followed by a secondary decision inside the else branch.

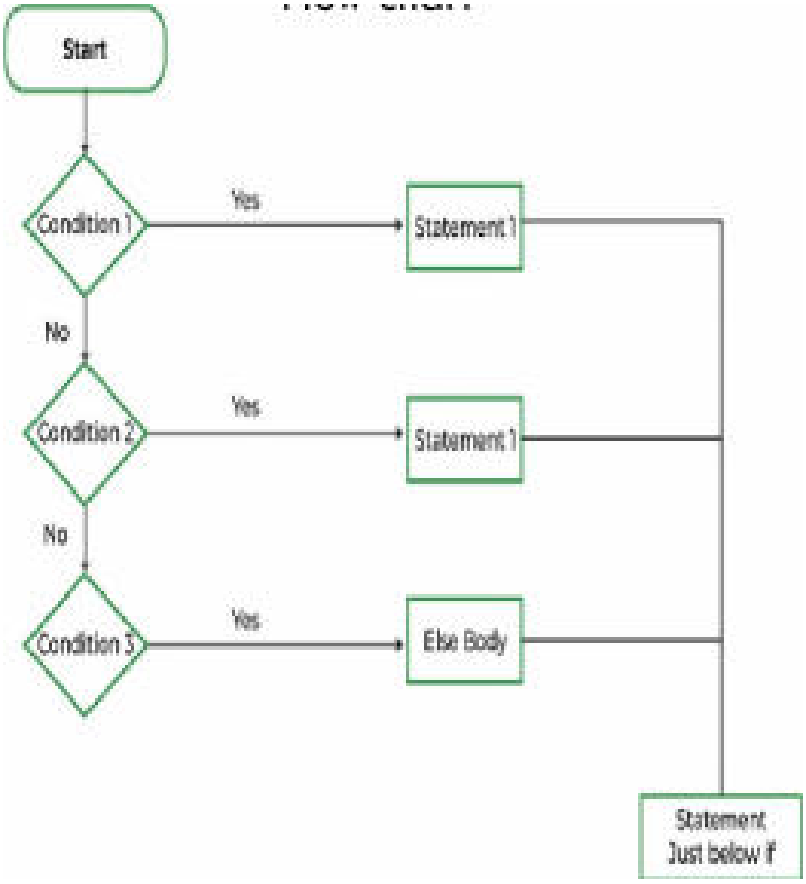
```c
/* Quick demo of nested if-else */
void main() {
    int i;
    printf("Enter either 1 or 2: ");
    scanf("%d", &i);
    if (i == 1)
        printf("Entered One\n");
    else {
        if (i == 2)
            printf("Entered Two\n");
        else
            printf("Entered Other than One and Two\n");
    }
}
```

🔷 Cascaded if-else (if-else ladder)

> **Definition:** A series of else if clauses that test multiple mutually exclusive conditions.

Syntax

```c
if (cond1) {
    /* statement1 */
} else if (cond2) {
    /* statement2 */
} else if (cond3) {
    /* statement3 */
} else {
    /* default statement */
}
```



Illustrates sequential evaluation of conditions until one matches.

Example – Grade Calculation

```
/* Method I – nested if-else */
void main() {
    int m1, m2, m3, m4, m5, per;
    printf("Enter marks in five subjects: ");
    scanf("%d %d %d %d %d", &m1, &m2, &m3, &m4, &m5);
    per = (m1 + m2 + m3 + m4 + m5) / 5;

    if (per >= 60)
        printf("First division\n");
    else if (per >= 50)
        printf("Second division\n");
    else if (per >= 40)
        printf("Third division\n");
    else
```

```
        printf("Fail\n");
}
```

🔷 switch Statement

> **Definition:** Selects one of many code blocks to execute based on the value of an integer or character expression.

Syntax

```
switch (expression) {
    case constant1:
        /* statements */
        break;
    case constant2:
        /* statements */
        break;
    /* … */
    default:
        /* statements */
}
```

- expression is evaluated once; its value is compared with each case constant.
- Execution continues from the matching case until a break (or end of switch).
- If no case matches, the default block runs.

Simple Calculator using switch

```
#include

int main() {
    float a, b, res;
    char op;
    printf("Enter two operands: ");
    scanf("%f %f", &a, &b);
    printf("Enter an operator (+, -, *, /): ");
    scanf(" %c", &op);
```

```
    switch (op) {
        case '+': res = a + b; break;
        case '-': res = a - b; break;
        case '*': res = a * b; break;
        case '/': res = a / b; break;
        default:  printf("Incorrect Operator Value\n"); return 1;
    }
    printf("Result = %.2f\n", res);
    return 0;
}
```

🔁 Loop Control Structures

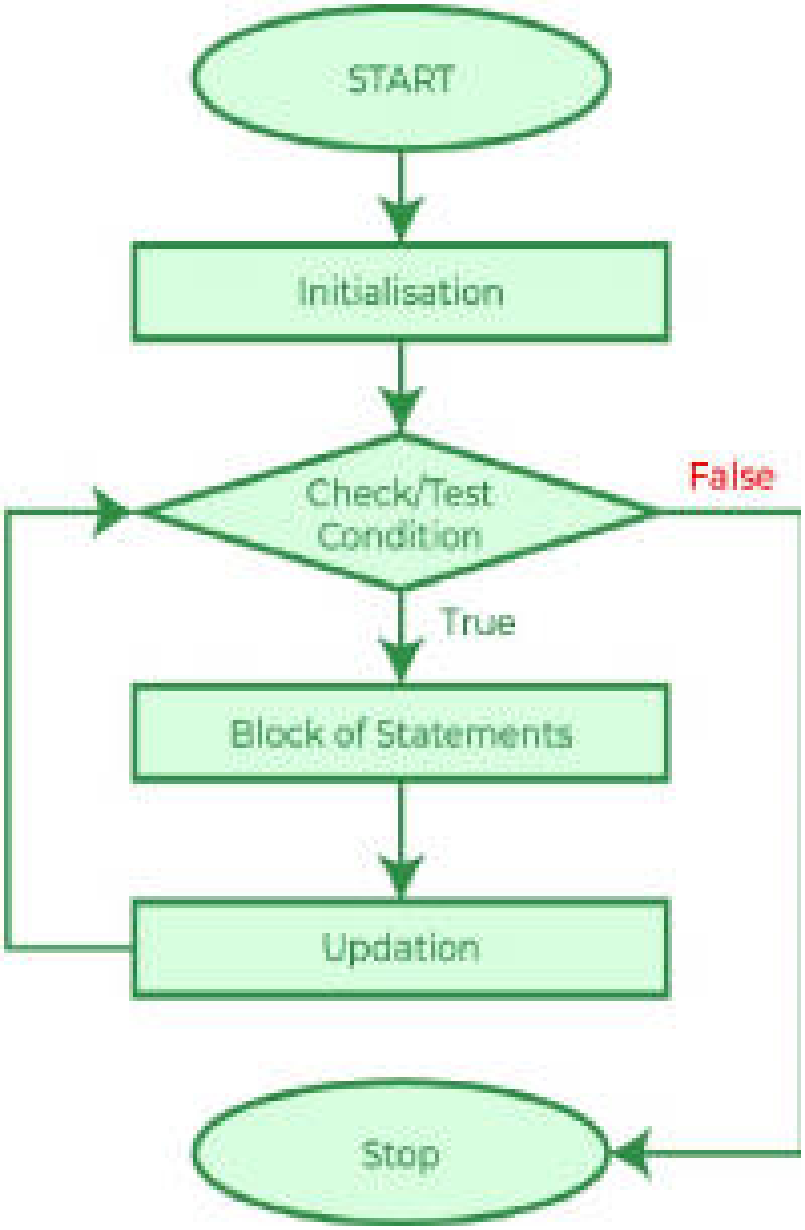
🔁 for Loop

> **Definition:** Entry-controlled loop; the test condition is evaluated before each iteration.

Syntax

```
for (initialization; test_expression; update) {
    /* loop body */
}
```

- initialization runs once.
- If test_expression is false, the loop terminates.
- After each iteration, update executes.



Shows initialization, test, body execution, update, and termination.

Common examples

```
/* Print numbers 1 to 10 */
for (int i = 1; i <= 10; i++)
    printf("%d ", i);

/* Table of a given number */
int num, i, table;
```

```
printf("Enter the number whose table you want: ");
scanf("%d", &num);
for (i = 1; i <= 10; i++) {
    table = num * i;
    printf("%d x %d = %d\n", num, i, table);
}

/* Sum of 1 to 10 */
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
printf("Sum = %d\n", sum);

/* Factorial */
int n, factorial = 1;
printf("Enter a number: ");
scanf("%d", &n);
for (int i = 1; i <= n; i++)
    factorial *= i;
printf("Factorial of %d is %d\n", n, factorial);
```

Pattern printing

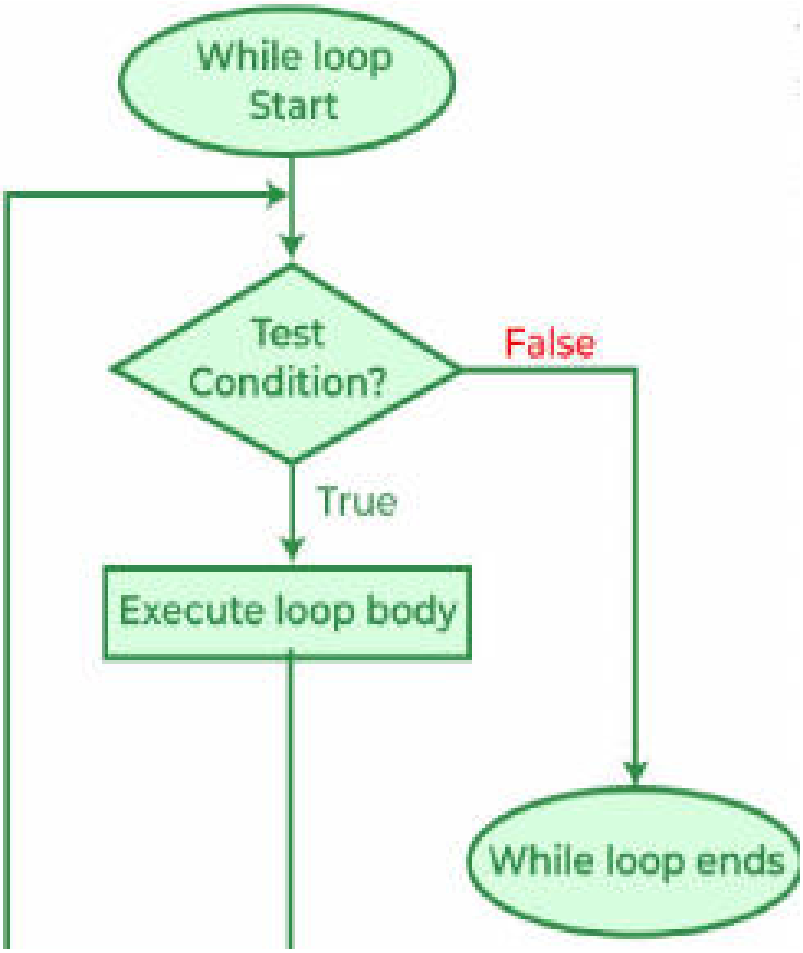
```
/* Pattern 1 */
for (int i = 1; i <= 4; i++) {
    for (int j = 1; j <= i; j++)
        printf("%d", j);
    printf("\n");
}

/* Inverted star pattern */
for (int i = 4; i >= 1; i--) {
    for (int j = 0; j < i; j++)
        printf("*");
    printf("\n");
}
```

🔁 while Loop

> **Definition:** Entry-controlled loop; the condition is evaluated before each iteration, suitable when the number of repetitions is not known beforehand.

Syntax

```
while (test_expression) {
    /* loop body */
}
```



Depicts the test condition, body execution, and loop back.

Example – Print 0 to 5

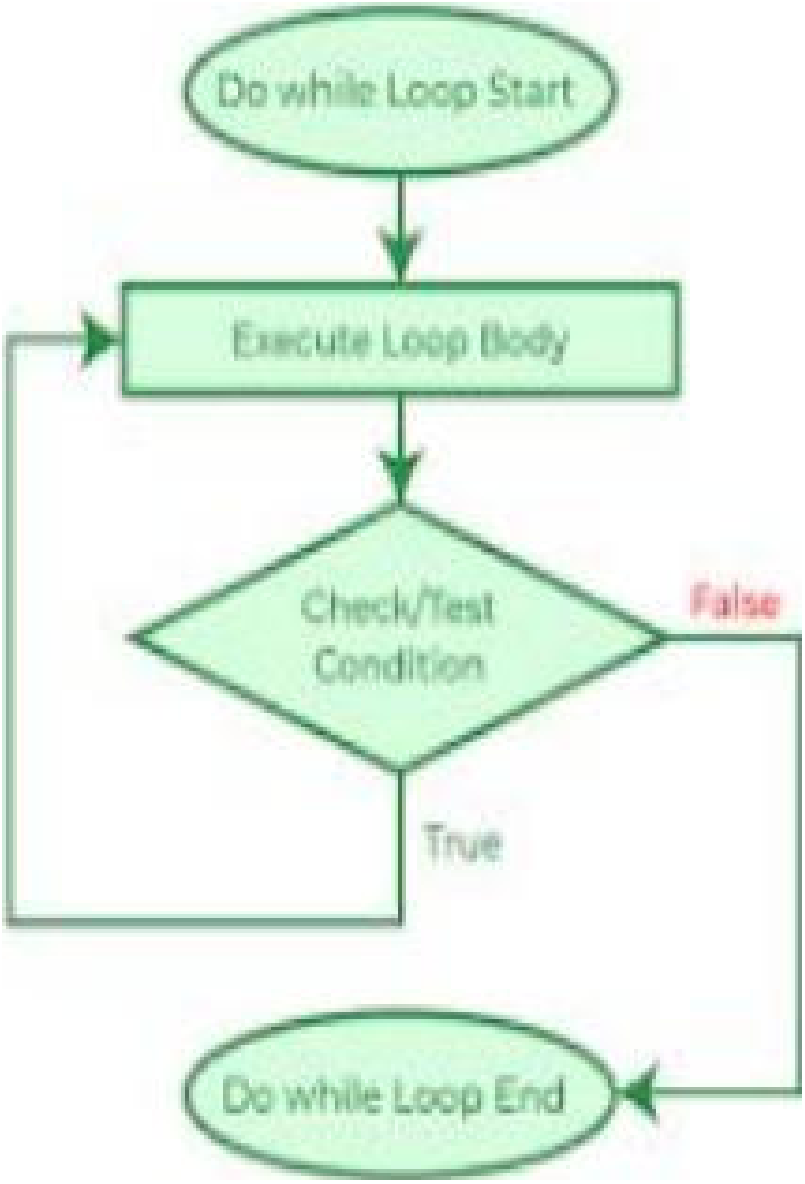
```c
int i = 0;
while (i < 5) {
    printf("%d\n", i);
    i++;
}
```

🔁 do-while Loop

> **Definition:** Exit-controlled loop; the body executes at least once, then the condition is tested.

Syntax

```c
do {
    /* loop body */
} while (test_expression);
```



Shows mandatory execution of the body before the condition check.

Examples

```
/* Simple repeat */
int i = 0;
do {
    printf("Sanjivani\n");
    i++;
} while (i < 3);
```

```c
/* Table using do-while */
int i = 1, num;
printf("Enter the number: ");
scanf("%d", &num);
do {
    printf("%d x %d = %d\n", num, i, num * i);
    i++;
} while (i <= 10);
```

🔲 break and continue Statements

Feature	break	continue
Effect	Terminates the **nearest** enclosing loop immediately.	Skips the remaining statements in the current iteration and jumps to the loop's update/condition test.
Typical use	Exit loop early when a condition is met.	Skip particular values (e.g., ignore multiples of 3).

Break example (for loop)

```c
for (int i = 0; i < 10; i++) {
    if (i == 5) break;        // exits loop when i == 5
    printf("%d ", i);
}
printf("\nExited with i = %d\n", i);
```

Continue vs. break demonstration

```c
printf("Loop with break:\n");
for (int i = 1; i <= 7; i++) {
    if (i == 3) break;
    printf("%d ", i);
}
printf("\nLoop with continue:\n");
for (int i = 1; i <= 7; i++) {
    if (i == 3) continue;
```

```
    printf("%d ", i);
}
```

Output:

- Break: 1 2 (stops at 3)
- Continue: 1 2 4 5 6 7 (skips 3)

↪️ goto Statement

> **Definition:** Transfers control to a labeled statement elsewhere in the program.

Syntax

```
label:   /* statement */
goto label;
```

Drawbacks (shown in a table)

Issue	Description
Readability	Code flow becomes hard to follow.
Maintainability	Increases difficulty of debugging and verification.
Structured alternatives	Same effect can be achieved with break, continue, or loop constructs.

Example using goto to print a multiplication table

```
void main() {
    int num, i = 1;
    printf("Enter the number whose table you want: ");
    scanf("%d", &num);
    table:
        printf("%dx%d = %d\n", num, i, num * i);
        i++;
```

```
        if (i <= 10) goto table;
}
```

📚 Summary Table of Decision & Loop Constructs

Construct	Type	Condition Evaluation	Typical Use
if	Decision	Single condition (true/false)	Execute one block conditionally
if-else	Decision	Single condition with two alternatives	Choose between two paths
Nested if-else	Decision	Hierarchical conditions	Complex multi-level decisions
Cascaded if-else	Decision	Multiple mutually exclusive conditions	Ladder of choices
switch	Decision	Integer/character expression matched to cases	Multi-way branching
for	Loop (entry-controlled)	Before each iteration	Known iteration count
while	Loop (entry-controlled)	Before each iteration	Unknown count, condition-driven
do-while	Loop (exit-controlled)	After each iteration	Must run at least once
break	Control	–	Exit loop early
continue	Control	–	Skip to next iteration
goto	Control	–	Jump to label (generally discouraged)
