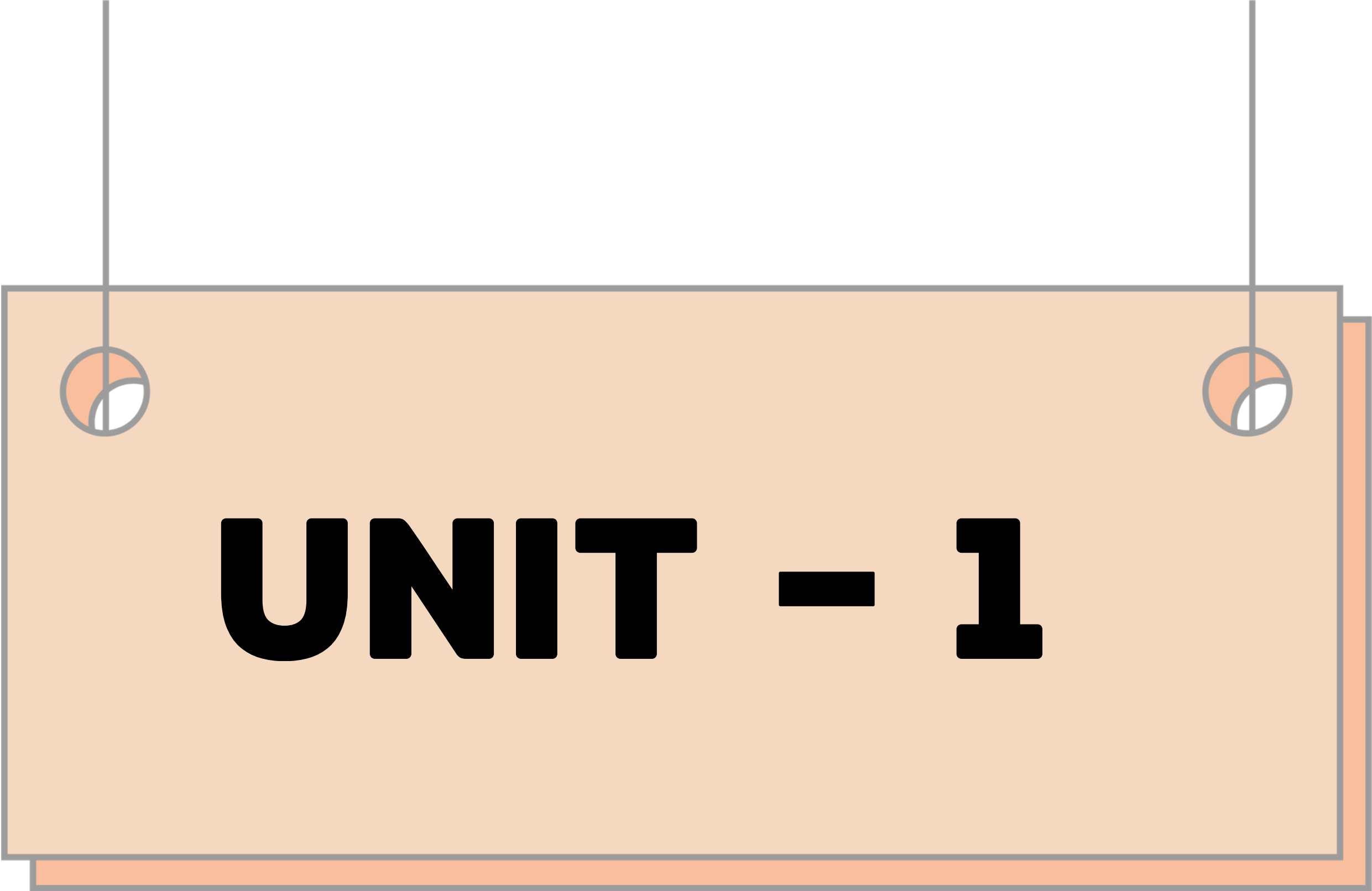


PROGRAMMING FOR PROBLEM SOLVING



Dr.Kavitha N
Associate Professor
Dept. of IT



UNIT - 1

UNIT- I

Topics to be covered :

Overview of C: C Language Elements, Variable Declarations and Data Types, Executable Statements, General Form of a C Program, Arithmetic Expressions, Formatting Numbers in Program Output. Selection Structures: Control Structures, Conditions, if Statement, if Statements with Compound Statements, Decision Steps in Algorithms. Repetition and Loop Statements: Repetition in Programs, Counting Loops and the while Statement, Computing a Sum or Product in a Loop, for Statement, Conditional Loops, Loop Design, Nested Loops, do-while Statement.

Introduction to C Programming



Father of C

Dennis Ritchie



Mother of All Programming Languages

- Simple, powerful, and widely used
- Builds the foundation for learning other languages



Definition

C is a general-purpose, mid-level, procedural, and structured programming language.



Key Feature

Allows direct interaction with computer hardware while supporting clear and structured programming

C Language Elements

Building blocks of every C program

A

Character Set

Letters, digits, symbols, whitespace

ID

Identifiers

Names for variables, functions (e.g., age, total_marks)

K

Keywords

Reserved words (e.g., int, if, return, while)

C

Constants

Fixed values (e.g., 10, 3.14, 'A', "Hello")

↔

Variables

Named storage whose value can change (int num = 10,)

T

Data Types

Define type of data (e.g., int, float, char, double)

+ - ÷ ×

Operators

Symbols for operations (+, -, >, &&, =)

x y

Expressions

Combination of variables, constants, operators (sum = a + b;)

! { }

Statements

Instructions (e.g., if, for, { } blocks)

⚙

Functions

Group of statements; every program starts with main()

HISTORY OF C PROGRAMMING

| | | |
|-------|----------------|--|
| 1960s | B | BCPL (Basic Combined Programming Language) developed by Martin Richards, influencing later languages. |
| 1970 | B | B language created by Ken Thompson at Bell Labs, simplifying BCPL for system programming. |
| 1972 | C | Dennis Ritchie developed C at Bell Labs, adding data types and structures to B. |
| 1978 | UNIX | Unix was rewritten in C, marking the rise of C in system programming. |
| 1989 | K&R | Kernighan & Ritchie published The C Programming Language, standardizing early C (K&R C). |
| 1999 | C99 | C99 introduced features like inline functions and variable-length arrays. |
| 2011 | C11 | C11 brought enhancements such as multithreading support and improved Unicode handling. |

VARIABLES & IDENTIFIERS IN C

VARIABLE

Definition

Named storage location for data

Rules for Variables

- Must begin with a letter or underscore
- Can include letters, digits, and underscores

Valid:

age

Invalid: 1total_marks

IDENTIFIERS

Definition

Names of variables, functions, arrays, etc.

Rules for Identifiers

- Cannot be a C keyword
- Cannot include special symbols
Except Underscore (_)

Valid: total_marks

Invalid: num!

Global and local variables

- **Global Variable:** A variable declared outside any function, accessible from anywhere in the program.
- **Local Variable:** A variable declared inside a function or a block, accessible only within that function or block.

```
#include <stdio.h>

int globalVar = 10; // Global variable

int main() {
    int localVar = 20; // Local variable

    printf("Global: %d\n", globalVar);
    printf("Local: %d\n", localVar);

    return 0;
}
```


DATA TYPES IN C

Data types define the type of data a variable can store.

int

10

float

3.14

char

'A'

double

3.14159

Data Types in C

Primary

- **int**
- **char**
- **float**
- **double**
- **void**

Derived

- **Function**
- **Array**
- **Pointer**

User Defined

- **Structure**
- **Union**
- **Enum**
- **Typedef**

Implicit and Explicit Type Conversion in C

Implicit Type Conversion

Automatic conversion of one data type into another

```
int x = 5;  
float y = x;  
float // int to float
```

Explicit Type Conversion

Conversion of one data type into another using a cast operator

```
int x = 5;  
float y = (float)x;  
// (float) int
```


Understanding ASCII Values

What is ASCII?

ASCII (American Standard Code for Information Interchange) Is a character encoding standard used to represent text in computers and other devices.



48



65



97



121

Executable Statements in C Language



Definition

An executable statement in C is a line of code that specifies an action to be performed when the program is run. It is a command that tells the computer to do something, such as calculate a value, assign data to a variable, call a function, or control the flow of the program.

Types



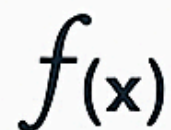
Expression Statements

Evaluate expressions



Assignment Statements

Assign values to variables



Function Call Statements

invoke functions

Example

```
#include <stdio.h>
int main() {
    int score = 85; /* This is an assignment
statement.*/
    /* This is an expression. Its value (score + 5) is
assigned to 'new_score'.*/
    int new_score = score + 5;
    /* This is a function call statement. It calls the
'printf' function.*/
    printf("Your new score is: %d\n", new_score);
    return 0;
}
```


General Form of a C Program

Documentation

Description of the program

Preprocessor

Link directives

Definition

Symbolic constants

Global Declaration

Global variables

main() Function

Starting point of the program

```
// 1. Preprocessor Directives
// Includes the standard input/output library for using printf.
#include <stdio.h>

// 2. The main() Function
// The starting point of program execution.
int main() {

    // 3. Executable Statements
    // Prints the "Hello, World!" message to the console.
    printf("Hello, World!\n");

    // 4. Return Statement
    // Returns 0 to indicate successful program execution.
    return 0;

}
```


DOCUMENTATION SECTION

Single-line Comments

Definition: A comment that starts with `//` and ends at the line's end.

Example: `// This is a single-line comment`

Multi-line Comments

Definition: Comments enclosed between `/*` and `*/` spanning multiple lines.

Example: `/*
This is a
multi-line comment
*/`

Pre-processing Section

The pre-processing section handles directives like `#include` and `#define` before the code is compiled.

Example:

```
#include <stdio.h> /* Includes the standard input/output library.*/  
#define PI 3.14 // Defines a constant named PI.
```

Definition Section

The definition section is where you declare and define variables and functions.

```
#include <stdio.h>  
#define MAX_ITEMS 30  
int main() {  
    int current_items = 15;  
    if (current_items < MAX_ITEMS) {  
        printf("You can add %d more items.\n", MAX_ITEMS - current_items);  
    } else {  
        printf("The inventory is full.\n");  
    }  
    return 0;  
}
```


Global Declaration Section

The global declaration section in C is where you define variables and functions that are accessible throughout the entire program.

Main() Function

- **Mandatory entry point for every C program.**
- **Only one main() function exists in a program; it's where code execution begins.**

```
#include <stdio.h> /* Includes the standard input/output library*/
int main() {
    /* The code inside these curly braces is executed when the
    program runs.*/
    printf("Hello, C Programming!");
    /* Returns 0 to the operating system, indicating successful
    execution.*/
    return 0;
}
```

ARITHMETIC EXPRESSION

Arithmetic Expression is a combination of operands and Arithmetic operators. These combinations of operands and operators should be mathematically meaningful, otherwise, they can not be considered as an Arithmetic expression in C.

Based on the number of operands they require, operators are classified into Unary, Binary, and Ternary operators.

UNARY

A unary operator works with only one operand. It performs operations such as increment, decrement, negation, and logical NOT.

BINARY

A binary operator works with two operands.

TERNARY

- The Ternary operator is the only operator in C that takes three operands.
- It is also called the conditional operator (?:).

Input and Output In C Programming:

- The printf and scanf statements are fundamental functions in C programming used for input and output operations.
 - The printf function is used to display output on the screen. It allows you to print text, variables, and formatted data so that users can see the program's results.
 - Example:
 - `int age = 20;`
 - `printf("My age is %d\n", age);`
 - Output: My age is 20
- The scanf function is used to read input from the user through the keyboard. It allows the user to input values that the program can process.
 - Using scanf, programs can interact dynamically by taking input data, making programs more useful.
 - Example:
 - `int age;`
 - `printf("Enter your age: ");`
 - `scanf("%d", &age);`
 - `printf("You entered: %d\n", age);`
 - The program waits for the user to enter a number and then prints it.

OPERATORS

Operators are special symbols that perform specific actions on variables and values. Think of them as the action words of a program, telling it what to do with the data. They are what allow you to add numbers, compare values, assign a result, or change a variable.

01

Arithmetic Operators

02

Relational Operators

03

Logical Operators

04

Assignment Operators

05

Bitwise Operators

06



Conditional (Ternary) Operator



Arithmetic Operators



Arithmetic operators in C are symbols that perform mathematical calculations like addition, subtraction, multiplication, division, and modulus on numeric data.



- **Addition (+):** Adds two operands. For example, $x + y$.
- **Subtraction (-):** Subtracts the second operand from the first. For example, $x - y$.
- **Multiplication (*):** Multiplies two operands. For example, $x * y$.
- **Division (/):** Divides the first operand by the second. For example, x / y .
- **Modulus (%):** Finds the remainder of the division of one operand by another. This operator only works with integer types. For example, $x \% y$.
- **Increment (++):** Increases the value of an operand by 1. It can be used as a prefix ($++x$) or a postfix ($x++$).

Example of Arithmetic operators in C

Code:

```
#include <stdio.h>
int main() {
    int a, b;
    // Taking input from user
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);
    // Performing arithmetic operations
    printf("Addition: %d + %d = %d\n", a, b, a + b);
    printf("Subtraction: %d - %d = %d\n", a, b, a - b);
    printf("Multiplication: %d * %d = %d\n", a, b, a * b);
    printf("Division: %d / %d = %d\n", a, b, a / b);
    printf("Modulus: %d %% %d = %d\n", a, b, a % b);
    // Increment and Decrement
    printf("Increment: %d++ = %d\n", a, a + 1);
    printf("Decrement: %d-- = %d\n", b, b - 1);
    return 0;
}
```

Sample Input:

Enter two integers: 10 4

Sample Output:

Addition: 10 + 4 = 14

Subtraction: 10 - 4 = 6

Multiplication: 10 * 4 = 40

Division: 10 / 4 = 2

Modulus: 10 % 4 = 2

Increment: 10++ = 11

Decrement: 4-- = 3

Relational Operators

Relational operators are used to compare two values or expressions. They evaluate the relationship between the operands and return 1 (true) if the condition holds, or 0 (false) otherwise. These operators are fundamental for decision-making and include

- Equal to (==): Checks if the values of two operands are equal.
- Not equal to (!=): Checks if the values of two operands are not equal.
- Greater than (>): Checks if the value of the left operand is greater than the value of the right operand.
- Less than (<): Checks if the value of the left operand is less than the value of the right operand.
- Greater than or equal to (>=): Checks if the value of the left operand is greater than or equal to the value of the right operand.
- Less than or equal to (<=): Checks if the value of the left operand is less than or equal to the value of the right operand.

Example of Relational operators in C

Code:

```
#include <stdio.h>
int main() {
    int a = 10, b = 4;
    printf("a = %d, b = %d\n", a, b);
    // Using relational operators
    printf("a == b: %d\n", a == b); // Equal to
    printf("a != b: %d\n", a != b); // Not equal to
    printf("a > b: %d\n", a > b);   // Greater than
    printf("a < b: %d\n", a < b);   // Less than
    printf("a >= b: %d\n", a >= b); // Greater than or equal to
    printf("a <= b: %d\n", a <= b); // Less than or equal to
    return 0;
}
```

Sample Output:

```
a = 10, b = 4
a == b: 0
a != b: 1
a > b: 1
a < b: 0
a >= b: 1
a <= b: 0
```



Logical Operators



Logical operators are symbols used to combine or modify conditions/expressions, resulting in a boolean value (true or false). They are primarily used in decision-making to evaluate multiple conditions.

- **Logical AND (&&):** This operator returns true if both operands are true. Otherwise, it returns false.
- **Logical OR (||):** This operator returns true if at least one of the operands is true. It returns false only if both operands are false.
- **Logical NOT (!):** This is a unary operator that inverts the boolean value of its operand. If the operand is true, it returns false, and if the operand is false, it returns true.



Example of Logical operators in C

Code:

```
#include <stdio.h>
int main() {
    int a = 5, b = 10, c = 5;
    // Logical AND (&&) – true if both conditions are true
    if (a == c && b > a) {
        printf("Logical AND: Both conditions are true\n");
    }
    // Logical OR (||) – true if at least one condition is true
    if (a == c || b < a) {
        printf("Logical OR: At least one condition is true\n");
    }
    // Logical NOT (!) – true if the condition is false
    if (!(a == b)) {
        printf("Logical NOT: Condition is false, thus NOT true\n");
    }
    return 0;
}
```

Sample Output:

Logical AND: Both conditions are true

Logical OR: At least one condition is true

Logical NOT: Condition is false, thus NOT true



Assignment Operators



Assignment operator in C assigns the value of the right operand to the variable on the left operand, facilitating storage and manipulation of data within a program.

- Assignment (=): Assigns the value of the right-hand operand to the left-hand operand.
- Add and Assign (+=): Adds the right operand to the left operand and assigns the result to the left operand.
- Subtract and Assign (-=): Subtracts the right operand from the left operand and assigns the result to the left operand.
- Multiply and Assign (*=): Multiplies the right operand with the left operand and assigns the result to the left operand.
- Divide and Assign (/=): Divides the left operand by the right operand and assigns the result to the left operand.
- Modulus and Assign (%=): Calculates the modulus of the left operand by the right operand and assigns the result to the left operand.



Example of Assignment operators in C

Code:

```
#include <stdio.h>
int main() {
    int a = 10; // Simple assignment
    int b = 5;
    printf("Initial values: a = %d, b = %d\n", a, b);
    a += b; // a = a + b
    printf("After a += b: a = %d\n", a);
    a -= b; // a = a - b
    printf("After a -= b: a = %d\n", a);
    a *= b; // a = a * b
    printf("After a *= b: a = %d\n", a);
    a /= b; // a = a / b
    printf("After a /= b: a = %d\n", a);
    a %= b; // a = a % b
    printf("After a %%= b: a = %d\n", a);
    return 0;
}
```

Sample Output:

Initial values: a = 10, b = 5
After a += b: a = 15
After a -= b: a = 10
After a *= b: a = 50
After a /= b: a = 10
After a %= b: a = 0



Bitwise Operators



Bitwise operators enable performing logical operations on each bit of data separately, making them essential for tasks needing fine-grained bit control in C programming.



- Bitwise AND (&): Sets each bit to 1 if both corresponding bits are 1.
- Bitwise OR (|): Sets each bit to 1 if at least one of the corresponding bits is 1.
- Bitwise XOR (^): Sets each bit to 1 if the corresponding bits are different.
- Bitwise NOT (~): Inverts all the bits of a number.
- Left Shift (<<): Shifts all bits to the left, filling the new positions with zeros.
- Right Shift (>>): Shifts all bits to the right, filling the new positions on the left with the sign bit for signed integers or with zeros for unsigned integers.

Example of Bitwise operators in C

Code:

```
#include <stdio.h>
int main() {
    unsigned int a = 5; // binary: 00000101
    unsigned int b = 9; // binary: 00001001
    unsigned int result;
    result = a & b; // 00000001
    printf("a & b = %u\n", result);
    result = a | b; // 00001101
    printf("a | b = %u\n", result);
    result = a ^ b; // 00001100
    printf("a ^ b = %u\n", result);
    result = ~a; // bits inverted
    printf("~a = %u\n", result);
    result = b << 1; // shifts bits left by 1 place
    printf("b << 1 = %u\n", result);
    result = b >> 1; // shifts bits right by 1 place
    printf("b >> 1 = %u\n", result);
    return 0;
}
```

Sample Output:

```
a & b = 1
a | b = 13
a ^ b = 12
~a = 4294967290
b << 1 = 18
b >> 1 = 4
```


CONDITIONAL (TERNARY) OPERATOR

This operator is also called the ternary operator because it takes three operands. It is used to write concise conditional expressions in a single line.

Example:

```
int num = 10;  
(num % 2 == 0) ? printf("Even\n") : printf("Odd\n");
```

Output:

Even

MISCELLANEOUS OPERATORS

Miscellaneous operators perform various important tasks related to memory addressing, size calculation, conditional evaluation, and member access in structures, making them essential tools in C programming beyond basic arithmetic or logic operations.

- **sizeof**: Returns size of data type or variable.
- **& (address of)**: Returns the address of a variable.
- **, (comma)**: Separates expressions.

Formatting Numbers in Program Output

- These specifiers control how data types are displayed.
- The most common specifiers are:
- %d or %i for integers.
- %f for floating-point numbers. You can control the number of decimal places using %.nf (e.g., %.2f for two decimals).
- %c for single characters.
- %s for strings.
- %u for unsigned integers.
- %ld or %lld for long integers.

SELECTION STRUCTURES (DECISION MAKING):

Decision-making in C allows a program to choose between different paths of execution based on specific conditions. This is essential for creating programs that can respond to different inputs and situations. The main constructs for decision-making are if, if-else, else-if, and switch.

It can be achieved through different forms of if statement.

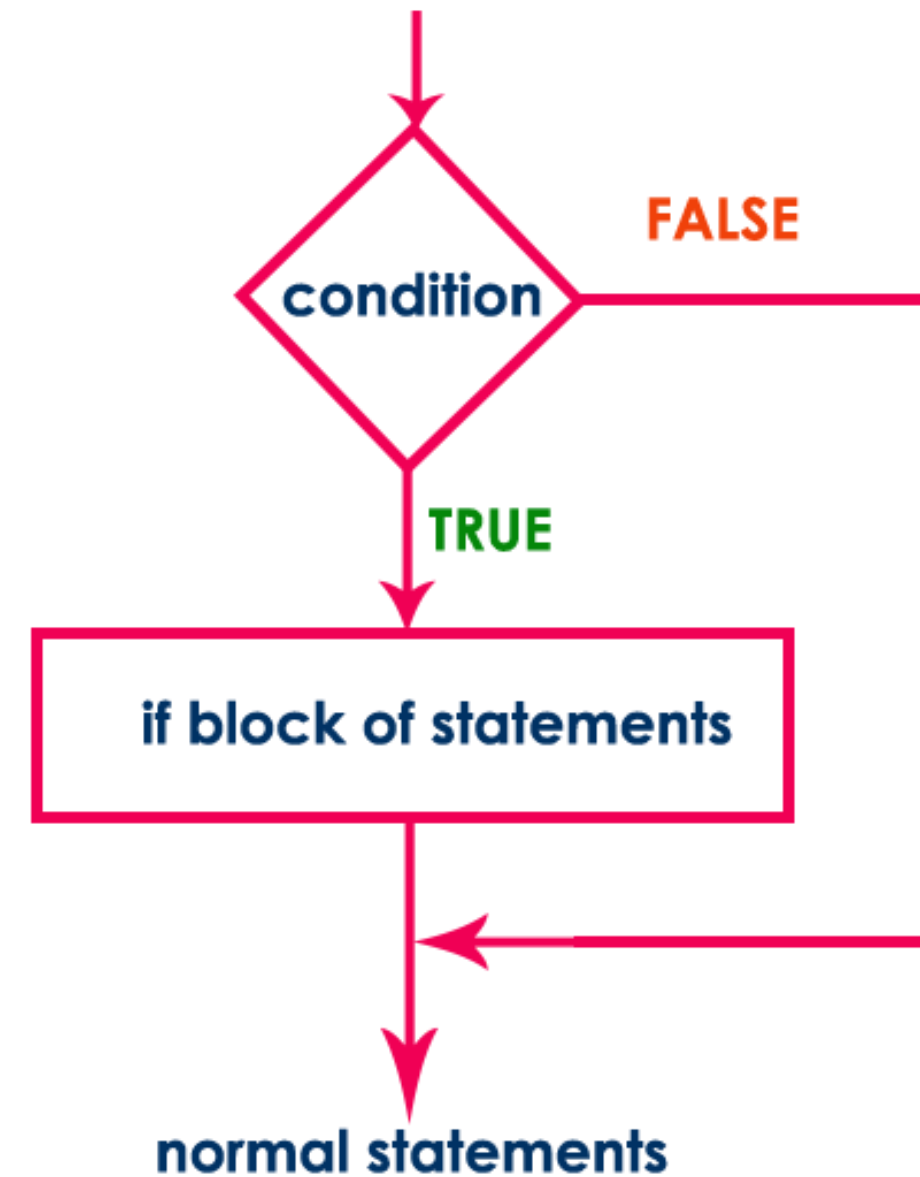
- a) simple if
- b) if – else
- c) nested if-else
- d) else-if ladder and switch statements

a) Simple if: Executes code only if a condition is true.

Syntax

```
if ( condition )  
{  
    ....  
    block of statements;  
    ....  
}
```

Execution flow diagram



Example of simple if in C

Write a C program that asks the user to enter a number. Using a simple if statement, check if the number is positive. If it is, print a message stating that the number is positive.

Code :

```
#include <stdio.h>

int main() {
    int num;

    printf("Enter a number: ");
    scanf("%d", &num);

    if (num > 0) {
        printf("The number is positive.\n");
    }

    return 0;
}
```

Sample Input:

15

Sample Output:

Enter a number: 15

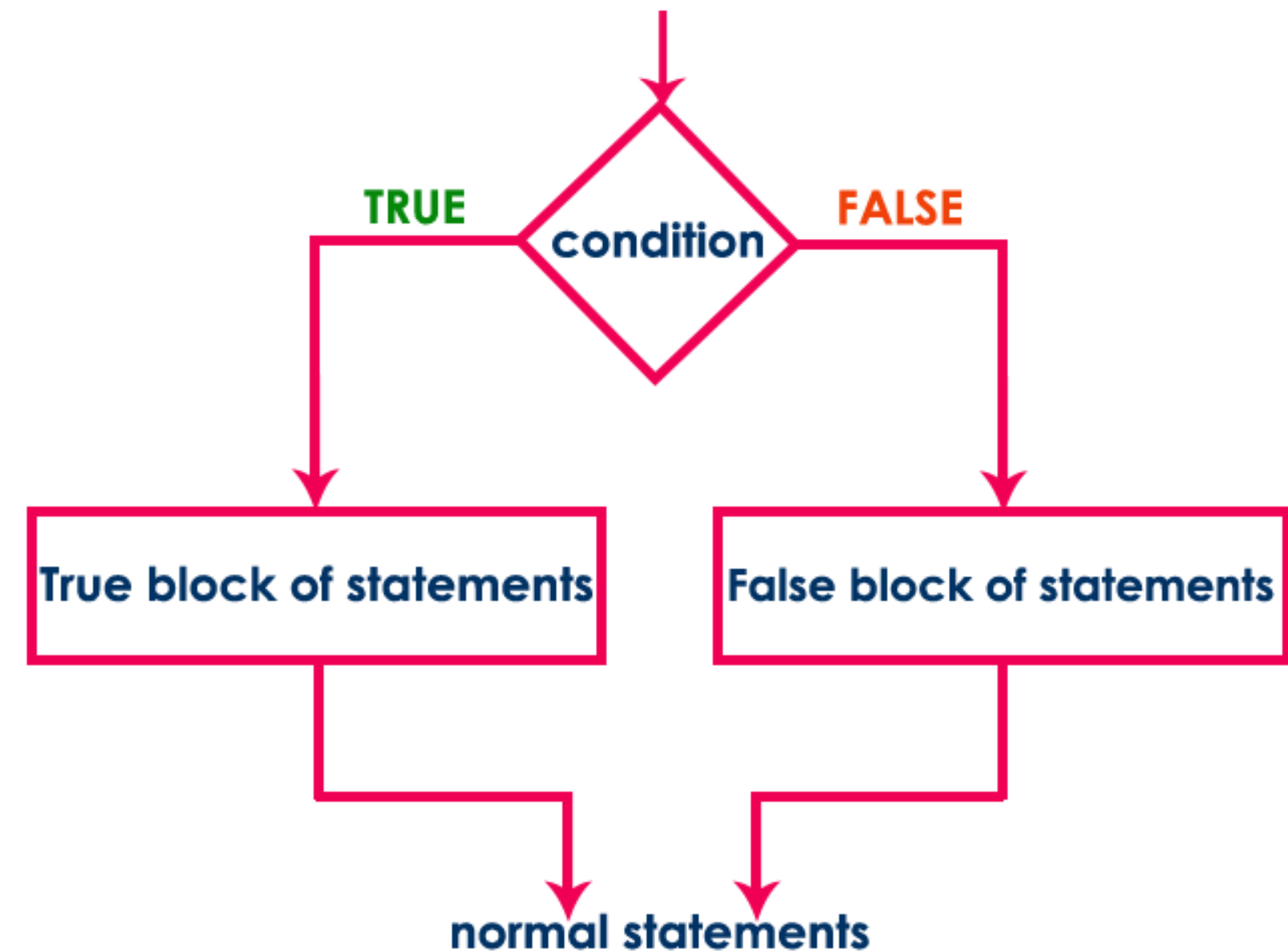
The number is positive.

b) if-else: Executes one block of code if a condition is true, and a different block if it's false.

Syntax

```
if ( condition )  
{  
    ....  
    True block of statements;  
    ....  
}  
else  
{  
    ....  
    False block of statements;  
    ....  
}
```

Execution flow diagram



Example of if - else in C

Write a C program that asks the user to enter their age. The program should use an if statement to check if the user is 18 or older. If they are, it should print "You are old enough to vote." Otherwise, it should print "You are not old enough to vote."

Code :

```
#include <stdio.h>

int main() {
    int age;

    printf("Please enter your age: ");
    scanf("%d", &age);

    if (age >= 18) {
        printf("You are old enough to vote.\n");
    } else {
        printf("You are not old enough to vote.\n");
    }

    return 0;
}
```

Sample Input:

20

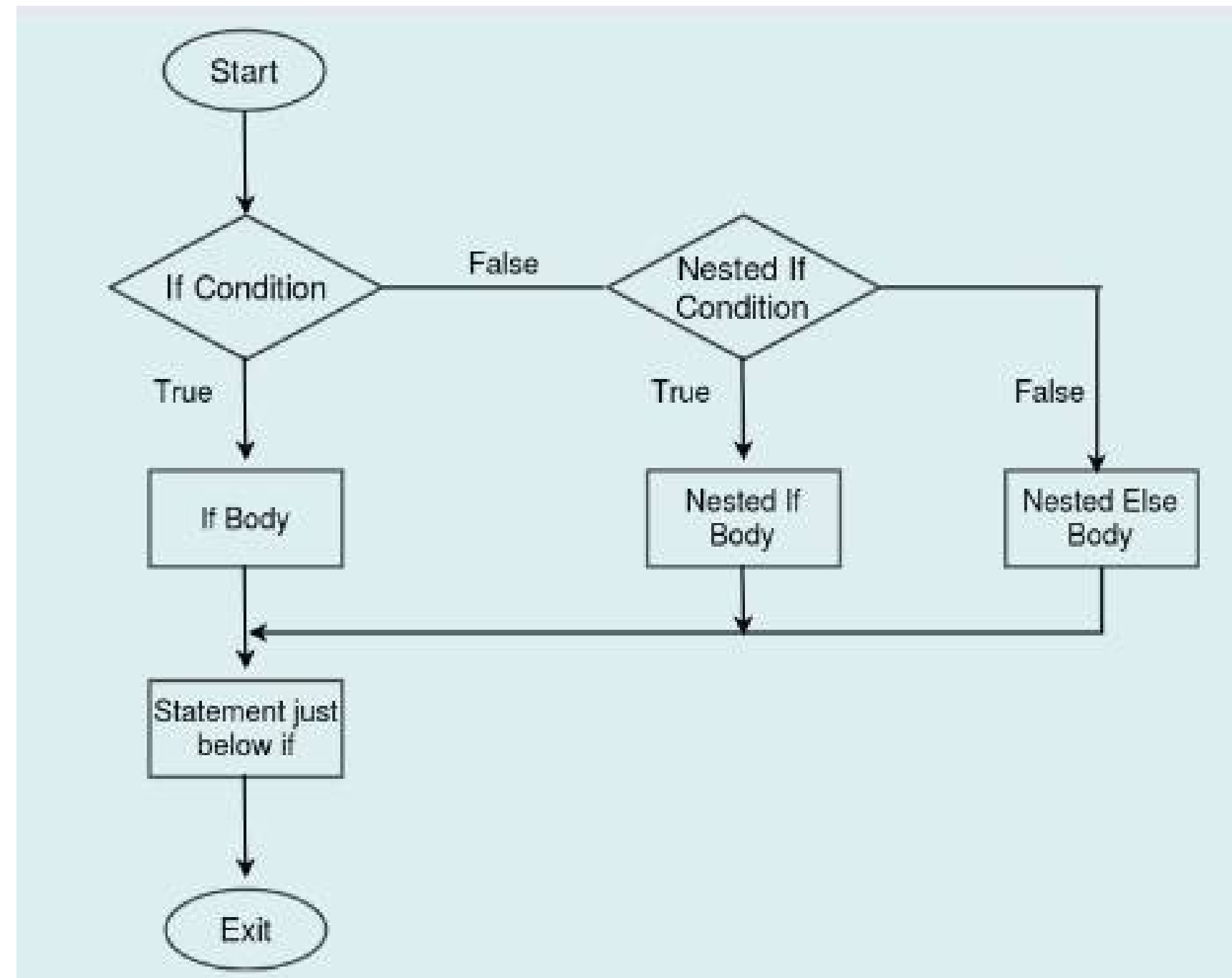
Sample Output:

Please enter your age: 20 You are old enough to vote.

c) Nested if-else: An if-else statement placed inside another if or else block to test multiple conditions.

Syntax

```
if ( condition1 )  
{  
    if ( condition2 )  
    {  
        ....  
        True block of statements 1;  
    }  
    ....  
}  
else  
{  
    False block of condition1;  
}
```



Example of nested if-else in C

Write a C program that asks the user to enter three integers. The program should use nested if-else statements to find and print the largest of the three numbers.

Code :

```
#include <stdio.h>
int main() {
    int num1, num2, num3;
    printf("Enter three integers: ");
    scanf("%d %d %d", &num1, &num2, &num3);
    if (num1 >= num2) {
        if (num1 >= num3) {
            printf("The largest number is: %d\n", num1);
        } else {
            printf("The largest number is: %d\n", num3);
        }
    } else {
        if (num2 >= num3) {
            printf("The largest number is: %d\n", num2);
        } else {
            printf("The largest number is: %d\n", num3);
        }
    }
    return 0;
}
```

Sample Input:

10 5 8

Sample Output:

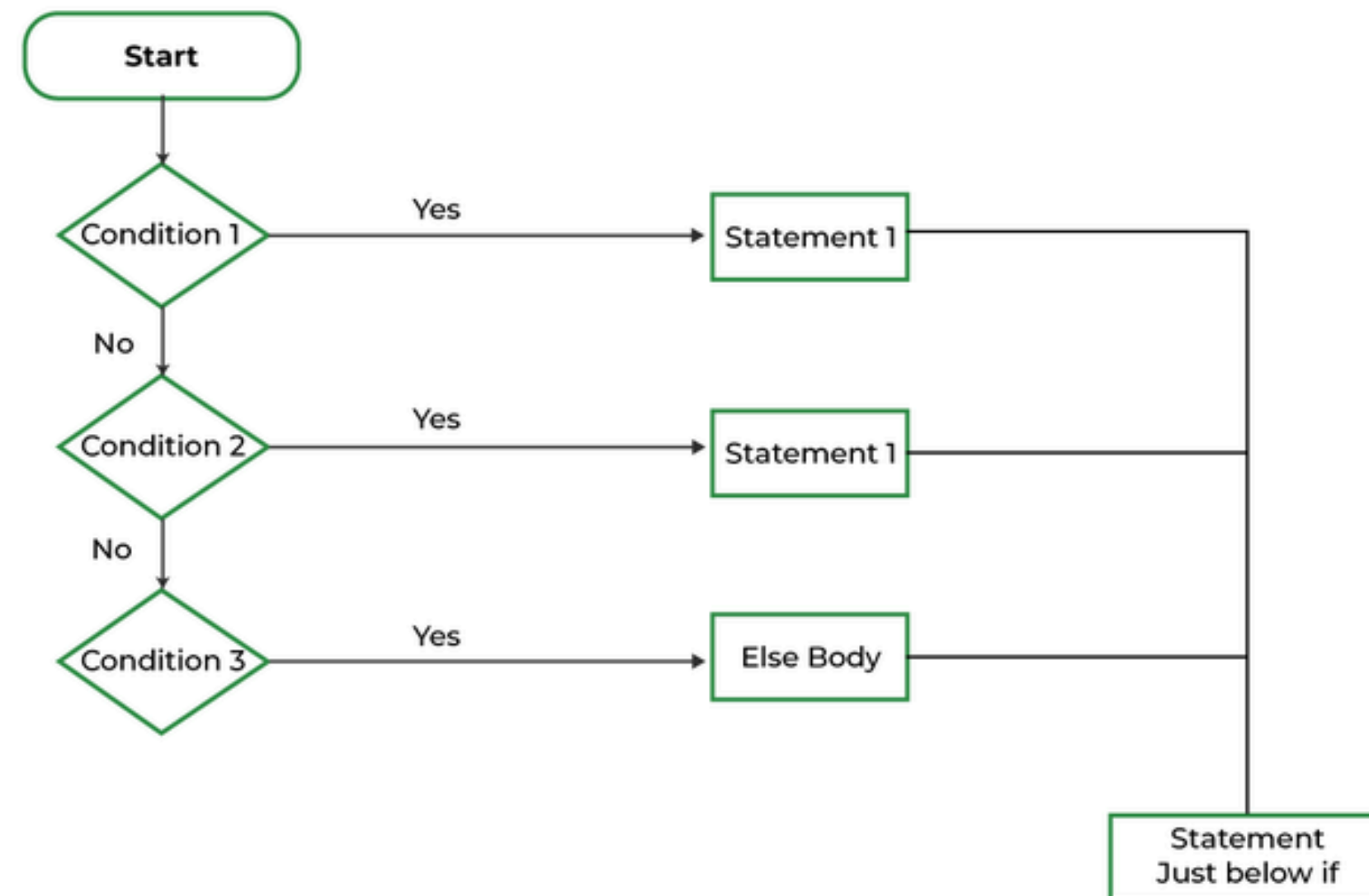
Enter three integers: 10 5 8

The largest number is: 10

d) else-if ladder: Checks multiple conditions in a sequence, executing the code for the first true condition it finds.

Syntax

```
if ( condition1 )  
{  
    ....  
    True block of statements1;  
    ....  
}  
else if ( condition2 )  
{  
    False block of condition1;  
    &  
    True block of condition2  
}
```



Example of else - if ladder in C

Write a C program that asks the user to enter an integer. The program should use an if-else if-else ladder to determine if the number is positive, negative, or zero, and then print the appropriate message.

Code :

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);

    if (num > 0) {
        printf("The number is positive.\n");
    } else if (num < 0) {
        printf("The number is negative.\n");
    } else {
        printf("The number is zero.\n");
    }

    return 0;
}
```

Sample Input:

0

Sample Output:

Enter an integer: 0
The number is zero.

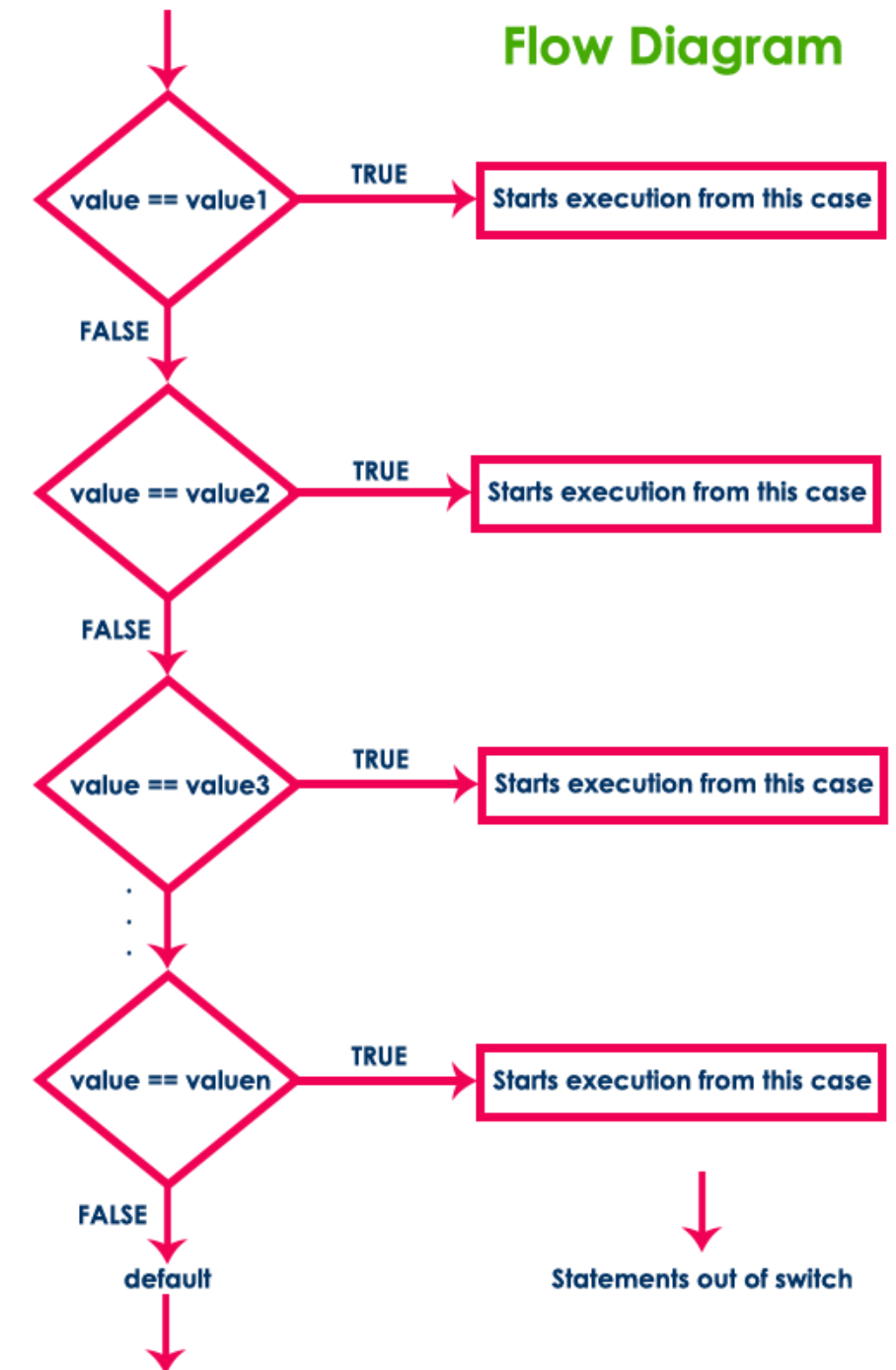
Switch Statement

A switch statement in C selects one of many code blocks to execute based on the value of a single expression.

Syntax

```
switch ( expression or value )  
{  
    case value1: set of statements;  
        ....  
    case value2: set of statements;  
        ....  
    case value3: set of statements;  
        ....  
    case value4: set of statements;  
        ....  
    case value5: set of statements;  
        ....  
    .  
    .  
    default: set of statements;  
}
```

Flow Diagram



Example of switch statement in C

Write a C program that asks the user to enter a single character representing a color initial ('r' for Red, 'g' for Green, 'b' for Blue). Use a switch statement to print the full color name. For any other character, print "Unknown color."

Sample Input:

r

Sample Output:

Enter a color initial (r, g, or b): r

You chose Red.

Code :

```
#include <stdio.h>
int main() {
    char color_initial;
    printf("Enter a color initial (r, g, or b): ");
    scanf(" %c", &color_initial);
    switch (color_initial) {
        case 'r':
        case 'R':
            printf("You chose Red.\n");
            break;
        case 'g':
        case 'G':
            printf("You chose Green.\n");
            break;
        case 'b':
        case 'B':
            printf("You chose Blue.\n");
            break;
        default:
            printf("Unknown color.\n");
            break;
    }
    return 0;
}
```

Loop control statements

The looping statements are used to execute a single statement or block of statements repeatedly until the given condition is FALSE.

C language provides three looping statements...

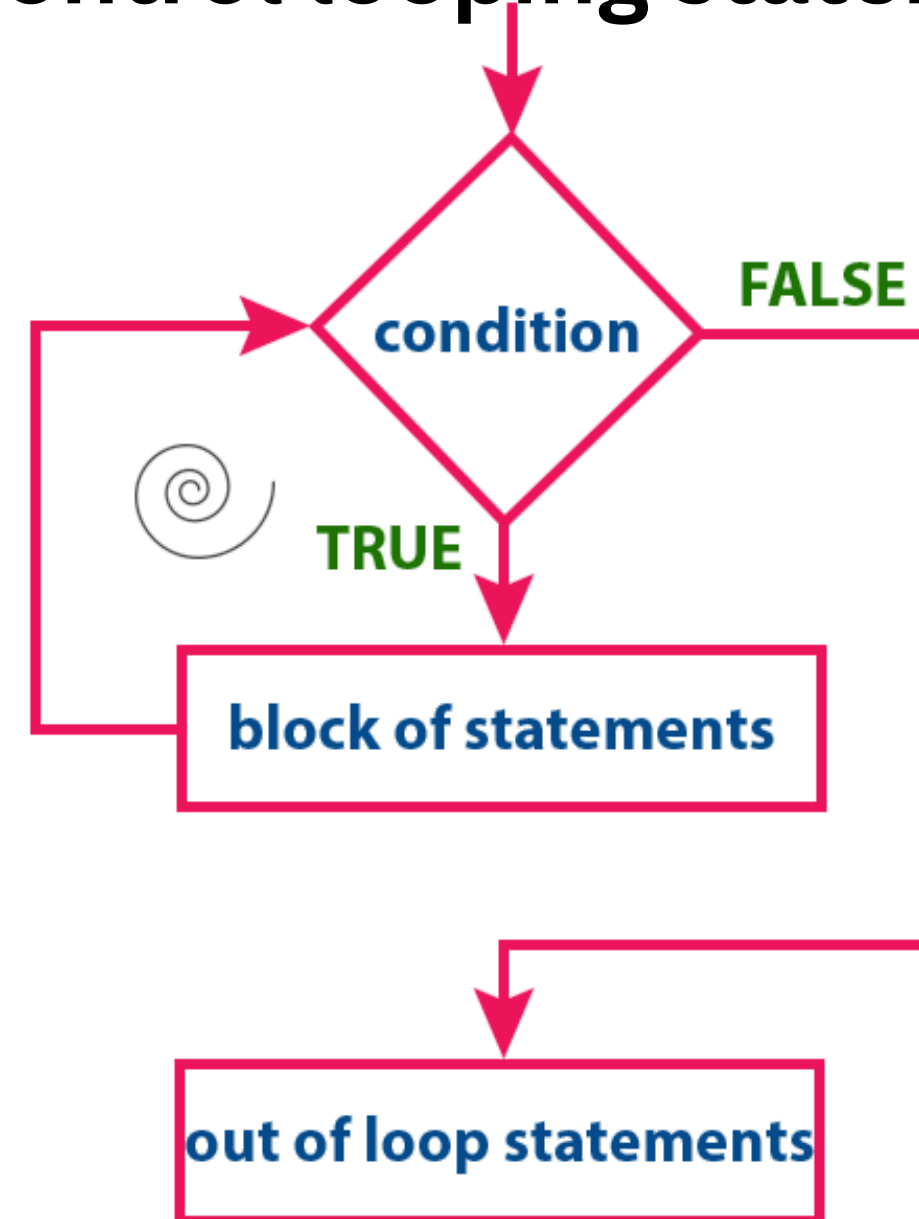
- while statement
- do-while statement
- for statement

while Statement

The while statement is used to execute a single statement or block of statements repeatedly as long as the given condition is TRUE. The while statement is also known as Entry control looping statement.

Syntax:

```
while( condition )  
{  
    ...  
    block of statements;  
    ...  
}
```



Example of while loop in C

Write a C program that repeatedly asks the user to guess a number between 1 and 10 until they guess the correct number, which is 7. The program should print a message saying "Correct!" when the guess is right.

Code :

```
#include <stdio.h>
int main() {
    int guess;
    int correct_number = 7;
    printf("Guess a number between 1 and 10: ");
    scanf("%d", &guess);

    /* The while loop continues as long as the user's guess is not the
    correct number.*/
    while (guess != correct_number) {
        printf("Incorrect guess. Try again: ");
        scanf("%d", &guess);
    }
    printf("Correct!\n");
    return 0;
}
```

Sample Input:

5
2
8
7

Sample Output:

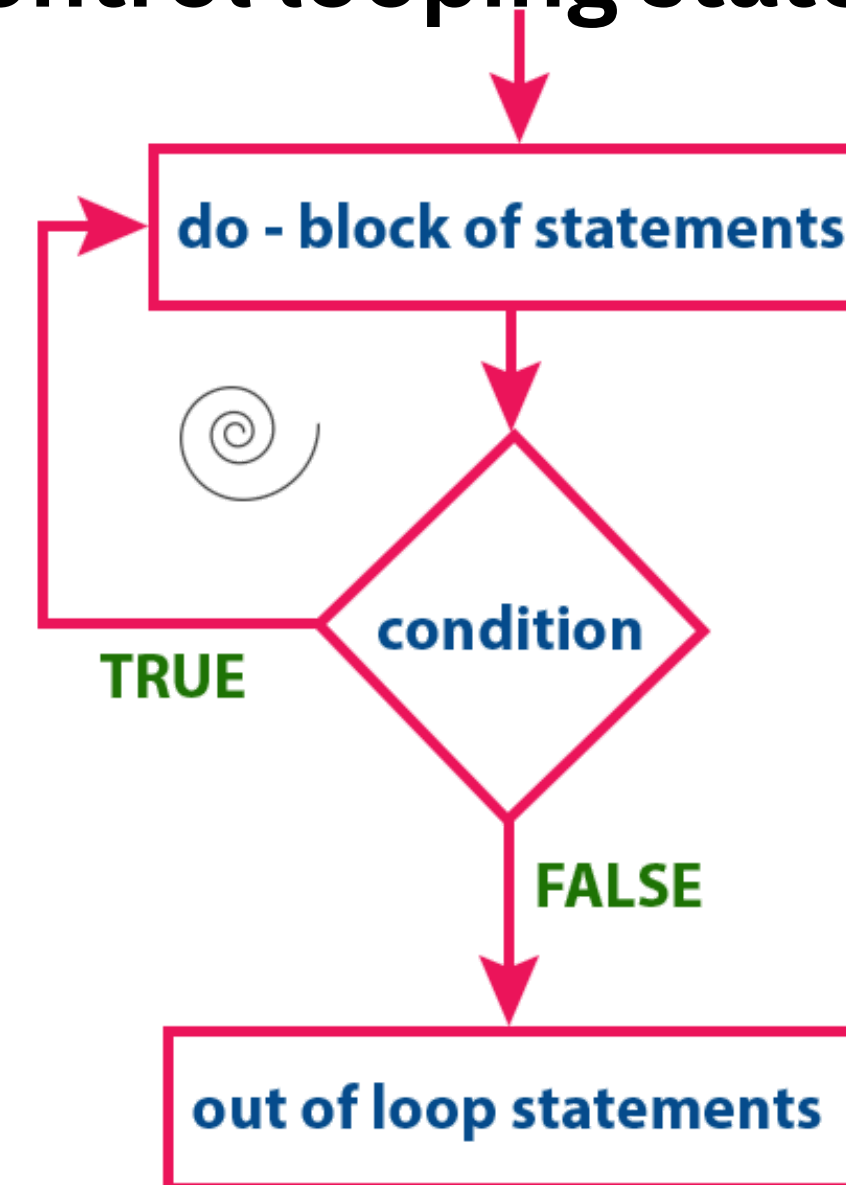
Guess a number between 1 and 10: 5
Incorrect guess. Try again: 2
Incorrect guess. Try again: 8
Incorrect guess. Try again: 7
Correct!

do-while statement in C

The do-while statement is used to execute a single statement or block of statements repeatedly as long as given the condition is TRUE. The do-while statement is also known as the Exit control looping statement.

Syntax:

```
do
{
    ...
    block of statements;
    ...
} while( condition );
```



Example of do-while loop in C

Write a C program that repeatedly asks the user to guess a number between 1 and 10 until they guess the correct number, which is 7. The program should print a message saying "Correct!" when the guess is right.

Code :

```
#include <stdio.h>
int main() {
    int guess;
    int correct_number = 7;
    printf("Guess a number between 1 and 10: ");
    scanf("%d", &guess);

    /* The while loop continues as long as the user's guess is not the
    correct number.*/
    while (guess != correct_number) {
        printf("Incorrect guess. Try again: ");
        scanf("%d", &guess);
    }
    printf("Correct!\n");
    return 0;
}
```

Sample Input:

5
2
8
7

Sample Output:

Guess a number between 1 and 10: 5
Incorrect guess. Try again: 2
Incorrect guess. Try again: 8
Incorrect guess. Try again: 7
Correct!

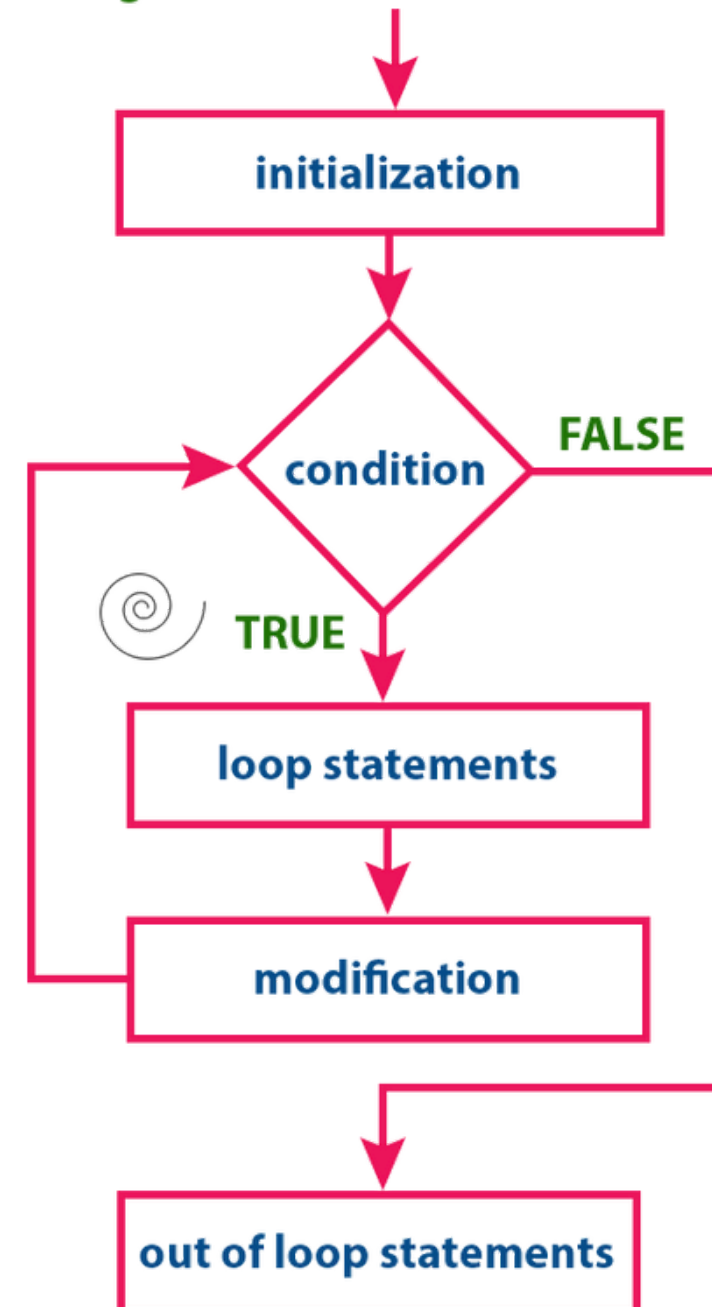
for statement in C

The for statement is used to execute a single statement or a block of statements repeatedly as long as the given condition is TRUE.

Syntax:

```
for( initialization ; condition ; modification )  
{  
    ...  
    block of statements;  
    ...  
}
```

Execution flow diagram:



Example of for loop in C

Write a C program that calculates and prints the sum of all integers from 1 to a number 'n' provided by the user.

Code :

```
#include <stdio.h>
int main() {
    int n, i;
    int sum = 0;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    /*The for loop initializes i to 1, continues as long as i is less than or equal
to n, and increments i by 1 in each iteration.*/
    for (i = 1; i <= n; i++) {
        sum += i;
    }
    printf("The sum of numbers from 1 to %d is: %d\n", n, sum);
    return 0;
}
```

Sample Input:

10

Sample Output:

Enter a positive integer: 10

The sum of numbers from 1 to 10 is: 55

Nested Loops:

- A nested loop is a loop inside another loop.
- The outer loop controls how many times the inner loop executes.
- Commonly used in tables, patterns, and multi-dimensional data (like matrices).

General Form:

```
for (initialization; condition; update) { // Outer loop
    for (initialization; condition; update) { // Inner loop
        // Statements
    }
}
```

Thank You