Module 1: Introduction to C

This first chapter will cover assignment statements, arithmetic statements, loop statements, conditional statements, and other fundamental rules for writing code in C.

Overview of C

INTRODUCTION TO C LANGUAGE

C is a general-purpose programming language that is closely related to how computer machines work. Although often considered difficult to learn, C is actually a simple language with vast capabilities.

Here are some key points to note in C:

- **Case-sensitive**: C distinguishes between uppercase and lowercase letters. For example, `printf` and `Printf` are two different things.
- **Space-insensitive**: Separators such as spaces, tabs, or new lines do not affect the program.
- **Semicolon**: Every statement must end with a semicolon (`;`).
- **Multiple Statements**: Several statements can be written on the same line.

SIMPLE C PROGRAM: PRINTING A LINE OF TEXT

The simplest C program is a program that prints text. Here is an example:

```
#include <stdio.h>


int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Output:

```
Hello, World!
```

Parts of the Program

1. **Comment**:
   - Single-line comments use `//`, while multi-line comments use `/* ... */`.

```
// This is a single-line comment
/*
   This is a
   multi-line comment
*/
```

2. **Header File**:
   - Header files like `stdio.h` are required to use functions such as `printf()` or `scanf()`.

```
#include <stdio.h>
```

3. **Main Function**:
   - The `main()` function is the program's entry point.
   - `int main()` indicates that the function returns an integer (0 for success, 1 or more for failure).

```
int main() {
    // Program code
    return 0; // Indicates successful program execution
}
```

4. **The `printf()` Function**:
   - This function is used to print output to the screen.
   - `\n` is an escape sequence meaning newline (new row).

```
printf("Hello, World!\n");
```

VARIABLES AND DATA TYPES

Variables are "containers" for storing values. The data type determines the kind of value that can be stored in a variable.

Types of Data in C

1. **`int`** - For integer values.

```
int number1;  // Variable without initialization (random value)
int number2 = 20;  // Variable initialized with value 20
```

2. **`float`** - For decimal values.

```
float decimal = 3.14;
```

3. **`char`** - For storing a single character.

```
char letter = 'A';
```

Here is a complete diagram of data types in C:

Alt text

Image not found or type unknown

Naming Variables

- Variable names must start with a letter or an underscore (`_`).
- Spaces or punctuation marks (such as ?, !, etc.) are not allowed.
- Case-sensitive: `name` and `Name` are different variables.

Example:

```
int age = 20;      // Valid
float height = 170; // Valid
char initial = 'A'; // Valid
int ageOfFather = 45; // Valid


int 2age = 20;     // Invalid (cannot start with a number)
```

Complete Example

Here is an example program using variables and data types:

```
#include <stdio.h>
```

```
int main() {
    int age = 25;
    float height = 170.5;
    char initial = 'A';

    printf("Age: %d years\n", age);
    printf("Height: %.2f cm\n", height);
    printf("Initial: %c\n", initial);

    return 0;
}
```

Output:

```
Age: 25 years
Height: 170.50 cm
Initial: A
```

Operator & If-Else Statement

1. Arithmetic Operators

a) **Addition (+)**

Adds two values.

```
int a = 5, b = 3;
int result = a + b; // Result: 8
printf("a + b = %d\n", result); // Output: a + b = 8
```

b) **Subtraction (-)**

Subtracts one value from another.

```
int a = 10, b = 4;
int result = a - b; // Result: 6
printf("a - b = %d\n", result); // Output: a - b = 6
```

c) **Multiplication (*)**

Multiplies two values.

```
int a = 7, b = 3;
int result = a * b; // Result: 21
printf("a * b = %d\n", result); // Output: a * b = 21
```

d) **Division (/)**

Divides two values.

```
int a = 15, b = 5;
int result = a / b; // Result: 3
printf("a / b = %d\n", result); // Output: a / b = 3
```

e) **Modulus (%)**

Returns the remainder of a division.

```
int a = 10, b = 3;
int result = a % b; // Result: 1 (since 10 ÷ 3 = 3 remainder 1)
printf("a %% b = %d\n", result); // Output: a % b = 1
```

f) **Increment (++)**

Increases the value of a variable by 1.

```
int a = 5;
a++; // Result: 6
printf("a = %d\n", a); // Output: a = 6
```

g) **Decrement (--)**

Decreases the value of a variable by 1.

```
int a = 8;
a--; // Result: 7
printf("a = %d\n", a); // Output: a = 7
```

---

2. Logical Operators

a) **Logical AND (&&)**

Returns `true` if both conditions are true.

```
int a = 5, b = 10;
if (a > 3 && b < 15) {
    printf("Both are true!\n"); // Output: Both are true!
}
```

b) **Logical OR (||)**

Returns `true` if at least one condition is true.

```
int a = 7, b = 12;
if (a > 10 || b < 5) {
    printf("At least one is true!\n"); // Will not execute since both conditions are false.
}
```

c) **Logical NOT (!)**

Reverses the condition’s result.

```
int a = 10;
if (!(a < 5)) {
    printf("a is not less than 5!\n"); // Output: a is not less than 5!
}
```

---

3. Comparison Operators

Comparison operators are used to compare two values and return a boolean (`true` or `false`).

a) == (Equal To)

Returns `true` if both operands are equal.

```
int a = 5, b = 5;
if (a == b) {
    printf("a and b are equal\n");
}
```

b) != (Not Equal To)

Returns `true` if the two operands are not equal.

```
int a = 5, b = 10;
if (a != b) {
    printf("a and b are different\n");
}
```

c) > (Greater Than)

Returns `true` if the left operand is greater than the right.

```
int a = 10, b = 5;
if (a > b) {
    printf("a is greater than b\n");
}
```

d) < (Less Than)

Returns `true` if the left operand is less than the right.

```
int a = 5, b = 10;
if (a < b) {
    printf("a is smaller than b\n");
}
```

e) >= (Greater Than or Equal To)

Returns `true` if the left operand is greater than or equal to the right.

```
int a = 10, b = 10;
if (a >= b) {
    printf("a is greater than or equal to b\n");
}
```

f) <= (Less Than or Equal To)

Returns `true` if the left operand is less than or equal to the right.

```
int a = 5, b = 10;
if (a <= b) {
    printf("a is smaller than or equal to b\n");
}
```

---

4. Assignment Operators

a) Simple Assignment (=)

Assigns a value to a variable.

```
int a;
a = 10; // a now holds 10
printf("a = %d\n", a); // Output: a = 10
```

b) Compound Assignment (+=, -=, *=, /=, %=)

Combines arithmetic operations with assignment.

```
int a = 5;
a += 3; // Equivalent to a = a + 3 → Result: 8
a -= 2; // Equivalent to a = a - 2 → Result: 6
a *= 4; // Equivalent to a = a * 4 → Result: 24
a /= 6; // Equivalent to a = a / 6 → Result: 4
a %= 3; // Equivalent to a = a % 3 → Result: 1
printf("Final result of a = %d\n", a); // Output: Final result of a = 1
```

---

5. If-Else Statement Structure

a) **If**

Executes code if the condition is true.

```
int number = 10;
if (number > 5) {
    printf("Number is greater than 5!\n"); // Output: Number is greater than 5!
}
```

b) **If-Else**

Executes an alternative code block if the condition is false.

```
int number = 3;
if (number > 5) {
    printf("Number is greater than 5!\n");
} else {
    printf("Number is not greater than 5!\n"); // Output: Number is not greater than 5!
}
```

c) **Else-If**

Adds additional conditions.

```
int number = 7;
if (number < 5) {
    printf("Number is less than 5!\n");
} else if (number == 5) {
    printf("Number is equal to 5!\n");
} else {
    printf("Number is greater than 5!\n"); // Output: Number is greater than 5!
}
```

Loop & Switch-Case

Loop & Switch-Case

1. While Loop

A `while` loop is a function used to execute the same block of code repeatedly. The loop continues execution as long as the given condition evaluates to `1` (TRUE) or more. When the condition evaluates to `0` (FALSE), the loop stops and the program proceeds to the next lines of code.

Similar to an `if` statement, the `while` loop is built into the C programming language, meaning there is no need to declare or return its value explicitly.

Syntax:

```
while (condition) {
    // Code to be executed repeatedly
}
```

The `condition` is checked before executing the loop body:

- If `condition` is `TRUE`, the code inside the loop is executed.
- If `condition` is `FALSE`, the loop terminates.

Example 1: Counting from 1 to 10

```
#include <stdio.h>

int main() {
    int n = 1;

    while (n <= 10) {  // Loop runs while n is less than or equal to 10
        printf("%d\n", n);
```

```
        n++;  // Increment n by 1 in each iteration
    }

    return 0;
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Infinite Loop

If the condition never changes or is always `TRUE`, the loop will run indefinitely.

Example 2: Infinite Loop (Press Ctrl+C to Stop)

```
#include <stdio.h>

int main() {
    while (1) {  // The condition is always TRUE
        printf("This loop will run forever!\n");
    }

    return 0;
}
```

Output (Repeats Forever):

```
This loop will run forever!
This loop will run forever!
```

```
This loop will run forever!
...
```

Using `break` to Exit a While Loop

The `break` statement can be used to exit a while loop forcefully.

Example 3: Using `break` to Stop the Loop

```c
#include <stdio.h>

int main() {
    int n = 1;

    while (1) {  // Infinite loop
        printf("%d\n", n);
        if (n == 5) {
            break;  // Exit the loop when n reaches 5
        }
        n++;
    }

    return 0;
}
```

Output:

```
1
2
3
4
5
```

Using `continue` to Skip an Iteration

The `continue` statement is used to skip the remaining code in the loop for a specific iteration.

Example 4: Skipping a Number (Skipping 5)

```
#include <stdio.h>

int main() {
    int n = 0;

    while (n < 10) {
        n++;

        if (n == 5) {
            continue;  // Skip printing 5
        }

        printf("%d\n", n);
    }

    return 0;
}
```

Output:

```
1
2
3
4
6
7
8
9
10
```

2. DO-WHILE LOOP

The `do-while` loop is similar to the `while` loop. The difference lies in the execution order:

- In a `while` loop, the condition is checked **before** executing the code.
- In a `do-while` loop, the code is executed **at least once** before checking the condition.

Syntax:

```
do {
    // Code to be executed
} while (condition);
```

Example 5: Difference Between While and Do-While

```
#include <stdio.h>

int main() {
    int n = 0;

    printf("Using while loop:\n");
    while (n > 0) {
        printf("This will NOT be printed.\n");
    }

    printf("\nUsing do-while loop:\n");
    do {
        printf("This WILL be printed at least once.\n");
    } while (n > 0);

    return 0;
}
```

Output:

```
Using while loop:


Using do-while loop:
This WILL be printed at least once.
```

Explanation:

- The `while` loop does **not** execute because `n > 0` is `FALSE`.
- The `do-while` loop runs **once** before checking the condition.

---

3. FOR LOOP

A `for` loop is an advanced version of the `while` loop. It allows for a **specific range** and **controlled iterations**.

The `for` loop consists of **three components**:

1. **Initialization (`init`)** → Sets the starting value. (e.g., `i = 1;`)
2. **Condition (`condition`)** → Determines when the loop stops. (e.g., `i <= 10;`)
3. **Increment (`increment`)** → Updates the loop variable. (e.g., `i++`)

Syntax:

```
for (initialization; condition; increment) {
    // Code to be executed
}
```

Example 6: Printing Numbers 1 to 10

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 10; i++) {
        printf("%d\n", i);
    }

    return 0;
}
```

Output:

```
1
2
3
4
5
6
7
```

```
8
9
10
```

Example 7: Loop with Step Size of 2

```c
#include <stdio.h>

int main() {
    for (int i = 0; i <= 10; i += 2) {
        printf("%d\n", i);
    }

    return 0;
}
```

Output:

```
0
2
4
6
8
10
```

---

4. SWITCH-CASE STATEMENT

A `switch-case` statement is an alternative to `if-else-if` for comparing a variable against multiple **fixed values**.

- If the variable matches a `case`, the corresponding block of code is executed.
- If no cases match, the `default` case is executed (if present).
- The `break` statement **prevents fall-through**, meaning once a match is found, execution stops.

Syntax:

```
switch (variable) {
    case value1:
        // Code to execute
        break;
    case value2:
        // Code to execute
        break;
    default:
        // Code if no cases match
}
```

Example 8: Simple Menu System

```
#include <stdio.h>

int main() {
    int choice;

    printf("Select an option:\n");
    printf("1. Start\n");
    printf("2. Settings\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Game Starting...\n");
            break;
        case 2:
            printf("Opening Settings...\n");
            break;
        case 3:
            printf("Exiting Program...\n");
            break;
        default:
            printf("Invalid Choice!\n");
    }
```

```
    return 0;
}
```

Example Output:

```
Select an option:
1. Start
2. Settings
3. Exit
Enter your choice: 2
Opening Settings...
```

Example 9: Days of the Week

```
#include <stdio.h>

int main() {
    int day;

    printf("Enter a number (1-7) for the day of the week: ");
    scanf("%d", &day);

    switch (day) {
        case 1:
            printf("Sunday\n");
            break;
        case 2:
            printf("Monday\n");
            break;
        case 3:
            printf("Tuesday\n");
            break;
        case 4:
            printf("Wednesday\n");
            break;
        case 5:
            printf("Thursday\n");
            break;
        case 6:
```

```
            printf("Friday\n");
            break;
        case 7:
            printf("Saturday\n");
            break;
        default:
            printf("Invalid input! Enter a number between 1 and 7.\n");
    }

    return 0;
}
```

Example Output:

```
Enter a number (1-7) for the day of the week: 5
Thursday
```

Nested Statements / Loops

1. NESTED IF STATEMENT

A **nested if statement** is an `if` condition inside another `if` block. This allows checking multiple conditions **in a hierarchical manner**.

Syntax:

```
if (condition1) {
    if (condition2) {
        // Code to execute if both conditions are true
    }
}
```

Example 1: Nested If Statement

```
#include <stdio.h>

int main() {
    int num = 10;

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

        if (num % 2 == 0) {  // Inner if
            printf("The number is even.\n");
        }
    }

    return 0;
}
```

Output:

```
The number is positive.
The number is even.
```

---

2. NESTED WHILE LOOP

A **nested while loop** is a `while` loop inside another `while` loop. The **inner loop** executes completely **for each iteration of the outer loop**.

Syntax:

```
while (condition1) {
    while (condition2) {
        // Code to execute
    }
}
```

Example 2: Multiplication Table using Nested While Loop

```
#include <stdio.h>

int main() {
    int i = 1, j;

    while (i <= 5) {
        j = 1;
        while (j <= 5) {
            printf("%d\t", i * j);
            j++;
        }
        printf("\n");
```

```
        i++;
    }

    return 0;
}
```

Output:

```
1  2  3  4  5
2  4  6  8  10
3  6  9  12 15
4  8  12 16 20
5  10 15 20 25
```

---

3. NESTED DO-WHILE LOOP

A **nested do-while loop** is a `do-while` loop inside another `do-while` loop. The inner loop will always execute **at least once** before checking the condition.

Syntax:

```
do {
    do {
        // Code to execute
    } while (condition2);
} while (condition1);
```

Example 3: Number Grid using Nested Do-While

```
#include <stdio.h>

int main() {
```

```
    int i = 1, j;

    do {
        j = 1;
        do {
            printf("%d ", j);
            j++;
        } while (j <= 5);

        printf("\n");
        i++;
    } while (i <= 5);

    return 0;
}
```

Output:

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

---

4. NESTED FOR LOOP

A **nested for loop** is a `for` loop inside another `for` loop. The **inner loop runs completely** for each iteration of the outer loop.

Syntax:

```
for (initialization; condition1; increment) {
    for (initialization; condition2; increment) {
        // Code to execute
    }
}
```

Example 4: Printing a Square Pattern

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        for (int j = 1; j <= 5; j++) {
            printf("* ");
        }
        printf("\n");
    }

    return 0;
}
```

Output:

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

---

5. NESTED SWITCH-CASE

A **nested switch-case** is when a `switch` statement is placed inside another `switch` statement.

Syntax:

```
switch (variable1) {
    case value1:
        switch (variable2) {
            case value2:
                // Code to execute
```

```
            break;
        }
        break;
}
```

Example 5: Nested Switch-Case for User Role and Permission

```
#include <stdio.h>

int main() {
    int role = 1; // 1 = Admin, 2 = User
    int action = 2; // 1 = View, 2 = Edit

    switch (role) {
        case 1:
            printf("Role: Admin\n");
            switch (action) {
                case 1:
                    printf("Action: Viewing data\n");
                    break;
                case 2:
                    printf("Action: Editing data\n");
                    break;
                default:
                    printf("Invalid action!\n");
            }
            break;

        case 2:
            printf("Role: User\n");
            switch (action) {
                case 1:
                    printf("Action: Viewing data\n");
                    break;
                default:
                    printf("Users cannot edit data!\n");
```

```
            }
            break;

        default:
            printf("Invalid role!\n");
    }

    return 0;
}
```

Example Output:

```
Role: Admin
Action: Editing data
```