STUDY MATERIAL

Fundamentals of C Programming

Contents

1. Introduction to Computers and Programming

1.1. What is a Computer?

A **computer** is a machine that helps us do many things faster and more easily. It takes **instructions**, processes them, and gives us the **results**.

You can think of a computer like a *smart helper*. Just like a cook follows a recipe to make food, a computer follows instructions to do tasks like:

- Adding two numbers
- Showing a movie
- Playing a song
- Sending a message

Computers are used **everywhere** — in homes, schools, offices, banks, hospitals, and even inside cars and mobile phones.

1.2. What is Software?

Software is a set of **instructions** that tells the computer what to do. Without software, a computer is just a *box of metal and wires*. It cannot do anything on its own.

For example, when you use a mobile phone and open apps like YouTube or WhatsApp — these are all **software applications (apps)**. They tell the phone what to show and how to work.

There are two main types of software:

- **System software** – like *Windows* or *Linux*, which help the computer start and run.
- **Application software** – like *MS Word*, *Google Chrome*, or *games*, which help you do specific tasks.

1.3. What is a Programming Language?

A **programming language** is a way for **humans to talk to computers**. Since computers do not understand normal languages like English or Hindi, we use special languages to give them instructions.

Popular programming languages include:

- C
- C++
- Python
- Java

Just like we use Hindi or English to talk to people, we use **C language** to talk to computers and tell them what to do.

Example: If you want the computer to add two numbers, you can write a program in C to do that. Below is a simple C program that adds two numbers and shows the result on the screen:

```c
#include <stdio.h>

int main() {
    int num1 = 5;
    int num2 = 3;
    int sum = num1 + num2;

    printf("The sum is: %d", sum);
    return 0;
}
```

Code 1. Simple C program to add two numbers

1.4. Why Learn C?

C is one of the **oldest and most powerful programming languages**. It is simple, fast, and very important in the world of computers.

Learning C is like learning the *alphabet* before writing words. It teaches you the basics of programming in a very clear way.

Here's why learning C is a smart choice for beginners:

- **C helps you understand the basics of programming.** Just like you learn alphabets before reading or writing, C teaches you how programming works from the ground up.
- **C is used in real-world systems.** Big systems like *ATMs*, *medical machines*, *mobile phones*, and even *car software* are built using C. If you ever dream of building such things, C is a good starting point.
- **C is fast and powerful.** Programs written in C run very quickly. That's why it's used where speed matters — like in games, robots, or space programs.
- **C builds a strong foundation.** If you want to learn other languages like C++, Java, or Python in the future, learning C first will make it easier to understand them.
- **C is used in college and job interviews.** Many universities teach C in the first semester. Also, many companies ask C programming questions in placement tests and interviews.
- **C is everywhere, even if you don't see it.** Many systems use C behind the scenes. Learning it helps you understand how computers really work.

Example: Suppose you want to build a calculator, a small game, or an app that controls lights or sensors. You can do all these things by writing programs in C.

Learning C is like learning to work with the engine of a machine. Once you understand it, you can build anything!

1.5. Real-world Applications of C

You may think C is an old language, but it is still used in many places around us — even today!

Here are some real-world uses of C:

- **Operating Systems:** C is used to build parts of operating systems like *Windows*, *Linux*, and *Android*. These systems run on laptops, mobile phones, and servers.
- **Mobile Phones and Smart Devices:** Inside your mobile phone, smartwatch, or smart TV, there is a small software called *firmware* — and many times, it is written in C.
- **Home Appliances (Embedded Systems):** Machines like washing machines, microwave ovens, and remote controls use tiny computers. The software inside them is often written in C.
- **Games and Graphics:** Some parts of popular games and their engines are still written in C to make them fast. Even famous old games like **Super Mario** were built using C.
- **Robotics and IoT Devices:** If you want to build a robot or a sensor-based smart system (like automatic lights), C is the main language used in Arduino boards and microcontrollers.
- **Compilers and Other Languages:** The tools that convert code to machine language (called compilers) — including C++ and Python compilers — are themselves written in C!

Example: The software in your school's projector, digital clock, or even your Bluetooth speaker might be written in C.

Example: If you join a robotics club or want to make an IoT device (like an automatic water tank sensor), you will probably use C.

In short, C is everywhere — even if you don't see it!

2. Keywords and Identifiers

When we write a program in C, we use many words. Some of these words have special meaning — they are called **keywords**. Others are names we create — they are called **identifiers**.

Keywords

Keywords are words that are already defined in the C language. These words have special purposes and we cannot use them for anything else.

There are 32 keywords in C. Some common ones are:

- `int` – used to declare an integer variable
- `return` – used to return a value from a function
- `if`, `else` – used for decision making
- `for`, `while` – used for loops
- `void` – used when no value is returned

List of All C Keywords

`auto`	`break`	`case`	`char`
`const`	`continue`	`default`	`do`
`double`	`else`	`enum`	`extern`
`float`	`for`	`goto`	`if`
`int`	`long`	`register`	`return`
`short`	`signed`	`sizeof`	`static`
`struct`	`switch`	`typedef`	`union`
`unsigned`	`void`	`volatile`	`while`

Note: Keywords are always written in `lowercase` and cannot be used as variable names. For example, You cannot name your variable as `int` or `return` because they are already used by C.

Identifiers

Identifiers are the names we give to things in a program — like variables, functions, arrays, etc.

You can think of an identifier like a label or a tag. It helps the computer identify something in your code. Examples of identifiers are:

- `marks`
- `total`
- `calculateSum`
- `student_name`

Rules for Naming Identifiers:

- An identifier can only have **letters (A-Z, a-z)**, **digits (0-9)**, and **underscore (_)**.
- It **must not start with a digit**.
- **No spaces or special characters** (like @, #, $) are allowed.
- It **must not be a keyword**.
- C is **case-sensitive** — so `Total` and `total` are different.

Good Identifier Examples: `age`, `student_id`, `totalMarks`
Bad Identifier Examples: `2number` (starts with a digit), `int` (keyword), `full name` (has a space)

Tip: Try to give meaningful names to your identifiers so that your program is easy to read and understand.

3. Constants in C

A **constant** is a value that **does not change** while the program is running. For example:

- The number `10` is a constant.
- The letter `'A'` is a constant.
- The word `"Hello"` is also a constant.

We use constants in a program when we want to use a fixed value. These values are not given by the user and do not change during the program. In C, constants are mainly of two types:

- **Number Constants (Numeric Constants)**
- **Character Constants**

3.1. Number Constants (Numeric Constants)

These constants are numbers. They can be:

- **Integer Constants** — Whole numbers(that can be positive, negative, or zero) without decimal point. *Examples:* `5`, `-20`, `1000`
- **Floating-point (Decimal) Constants** — Numbers with decimal point. *Examples:* `3.14`, `-0.5`, `100.00`

3.2. Character Constants

These constants are characters or text. They can be:

- **Single Character Constants** — A single character, written inside single quotes. *Examples:* `'A'`, `'x'`, `'1'`
- **String Constants** — A group of characters (text), written inside double quotes. *Examples:* `"Hello"`, `"123"`, `"C Programming"`

  Note 1: Remember — single characters use **single quotes** (`' '`), and strings use **double quotes** (`" "`).

  Note 2: In C programming, the way you write the value tells the compiler what type it is.

  - Anything written in `' '` (single quotes) is treated as a **character constant**.
    Example: `'A'`, `'5'`, `'@'`
  - Anything written in `" "` (double quotes) is treated as a **string constant**.
    Example: `"A"`, `"5"`, `"@"`, `"Hello"`

  Even though `'A'` and `"A"` look similar, they are **not the same**. `'A'` is a single character, and `"A"` is a string with one character.

Similarly, `5` is a **number constant** (integer). *It is used for mathematical calculations.* `'5'` is a **character constant**. *It represents the character '5', not the number.* For example, `'5'` is different from `5`. One is a character; the other is a number. `"5"` is a **string constant**. *It is a string that contains one character: the digit 5.*

Constant	Type
`5`	Integer constant
`5.0`	Floating-point (float) constant
`'5'`	Character constant
`"5"`	String constant

4. Variables in C

A **variable** is like a container that stores a value in your computer's memory. The value stored in a variable can change while the program runs — that's why it's called a *variable*.

Think of a variable as a box with a label. You can store something inside the box, check what's inside it, or change what's in the box later.

4.1. Why Do We Need Variables?

We use variables to:

- Store data like numbers, characters, or text
- Use and update values while the program runs
- Make the program reusable and flexible

4.2. Declaring a Variable

To create (or declare) a variable, we must tell the computer:

- The **data type** of the variable (what kind of value it will store)
- The **name** of the variable

Syntax:

```
data_type variable_name;
```

Example:

```
int age;
float height;
char grade;
```

You can also assign a value at the time of declaration:

```
int age = 18;
float height = 5.7;
char grade = 'A';
```

4.3. Multiple Variable Declaration

You can declare multiple variables of the same type in one line:

```
int a, b, c;
float x = 1.1, y = 2.2;
```

4.4. Rules for Naming Variables

When you create a variable name in C:

- It can contain letters, digits, and underscores (_)
- It **must start with a letter** or underscore
- It **cannot start with a number**
- It **cannot use spaces**
- It **cannot be a keyword** (like `int`, `return`, `for`, etc.)

Examples of valid variable names:

- `age`
- `total_marks`
- `number1`
- `_temp`

Examples of invalid variable names:

- `1value` (starts with number)
- `float` (keyword)
- `my value` (contains space)

4.5. Best Practices

- Use meaningful variable names (like `marks`, `price`, `total`, etc.)
- Avoid using single-letter names unless in loops or very short programs
- Initialize variables before using them

4.6. C is Case Sensitive

In C programming, **case matters**. That means `age`, `AGE`, and `Age` are all treated as **different** variable names. This is called **case sensitivity**.

What does it mean? Uppercase and lowercase letters are not the same in C.

```
int age = 20;
printf("%d", AGE); // This will cause an error!
```

In the above example:

- `age` is a variable
- But we tried to use `AGE`, which the program does not recognize

So, the program will give an **error** like: `'AGE' undeclared (first use in this function)`

Helpful Analogy:

Imagine you have three different people named:

- `ROHIT`
- `Rohit`
- `rohit`

Even though they look similar, they are treated as different names. C treats variable names the same way.

Try It Yourself

```
#include <stdio.h>
int main() {
    int value = 100;
    int Value = 200;
    int VALUE = 300;

    printf("%d %d %d", value, Value, VALUE);
    return 0;
}
```

Expected Output:

```
100 200 300
```

So remember: **Be careful with capital and small letters in C!**

Examples

Question: Identify which of the following are valid or invalid variable names and explain why.

Variable Name	Valid?	Reason
`BASICSALARY`	Yes	Uses only uppercase letters, valid identifier
`#MEAN`	No	Cannot begin with # (special characters not allowed)
`population in 2006`	No	Contains spaces and starts with a number
`FLOAT`	Yes	Valid (case-sensitive, not the same as keyword `float`)
`team'svictory`	No	Contains a special apostrophe (not allowed)
`_basic`	Yes	Valid: starts with an underscore
`group.`	No	Ends with a dot (invalid character)
`over time`	No	Contains a space
`hELLO`	Yes	Valid, uses letters only
`Plot # 3`	No	Contains # and spaces
`basic-hra`	No	Hyphen - not allowed in identifiers
`422`	No	Cannot start with a digit
`mindovermatter`	Yes	All lowercase letters, valid
`queue.`	No	Ends with a dot (invalid)
`2015_DDay`	No	Starts with a number

True or False: Variable Naming in C

Mark each statement as **True** or **False** based on C variable naming rules.

1. A variable name can start with a digit. ..... **False**
2. Variable names are case-sensitive. ..... **True**
3. You can use spaces in variable names. ..... **False**
4. Underscore (`_`) is allowed in variable names. ..... **True**
5. `float`, `int`, and `return` can be used as variable names. **False**
6. Variable names can contain special characters like `@`, `#`, `$`. **False**
7. A variable name can begin with an underscore (`_`). ..... **True**
8. C treats `Value`, `value`, and `VALUE` as three different variables. **True**
9. You can use dots (`.`) in variable names. ..... **False**
10. `char1` is a valid variable name. ..... **True**
11. `my-variable` is a valid variable name. ..... **False**
12. `while` is a valid variable name. ..... **False**
13. Variable names can be as long as needed. ..... **True**
14. You can declare a variable with the name `main`. ..... **True**
15. The name `_count` is a valid variable name. ..... **True**
16. Variable names can include capital and small letters. ... **True**
17. `first_name` is a valid and commonly used style. ..... **True**
18. You cannot use numbers anywhere in a variable name. . **False**
19. `class` can be used as a variable name in C. ..... **True**
20. Variable names must begin with a letter or underscore only. **True**

5. Basic Data Types in C

1. Integer Type – `int`

`int` is used to store whole numbers (without decimals).

- Example values: `10`, `-25`, `1000`
- Memory size: Usually 4 bytes (can vary by system)
- Format Specifier: %d

```c
int age = 18;
```

2. Floating Point Type – `float`

`float` is used to store decimal (real) numbers with single precision.

- Example values: `3.14`, `-0.5`, `1.0`
- Memory size: Usually 4 bytes
- Format Specifier: %f

Example:

```c
float pi = 3.14;
```

3. Character Type – `char`

`char` is used to store a single character like 'A', '1', or '

- Example values: `'A'`, `'x'`, `'5'`
- Memory size: 1 byte
- Format Specifier: %c

Note: Characters must be written inside single quotes.
Example:

```c
char grade = 'A';
```

Data Type	Size (Bytes)	Format Specifier
`int`	4	%d
`float`	4	%f
`char`	1	%c

6. Input and Output in C

In any C program, we often need to:

- Show information to the user (Output)
- Take data from the user (Input)

C provides some built-in functions to do this. The most common ones are:

- `printf()` – used to display output
- `scanf()` – used to take input from the user
- `getchar()` and `putchar()` – used to read or display characters

Let's understand each one with examples.

6.1. Printing Output using `printf()`

`printf()` is used to display text, numbers, or values on the screen.

1. Printing Constants

We can directly print constants of any type using `printf()`.

```c
#include <stdio.h>

int main() {
    printf("%d", 10);         // Integer constant
    printf("%f", 3.14);       // Float constant
    printf("%c", 'A');        // Character
    constant
    return 0;
}
```

2. Printing a Single Variable

We can store values in variables and print them using `printf()` with format specifiers.

```c
#include <stdio.h>

int main() {
    int age = 20;
    printf("%d", age);  // %d is used for
    integers
    return 0;
}
```

3. Printing Multiple Variables

We can print more than one variable in the same line.

```c
#include <stdio.h>

int main() {
    int a = 5, b = 10;
    printf("%d%d", a, b);
    return 0;
}
```

4. Printing Float with Decimal Points

You can control how many decimal places you want to display using `%.2f`, `%.3f`, etc.

```c
#include <stdio.h>

int main() {
    float pi = 3.14159;
    printf("Value of pi = %.2f", pi);    //
    prints 3.14
    return 0;
}
```

Common Format Specifiers

- `%d` – Integer
- `%f` – Float
- `%c` – Character

6.2. Taking Input using `scanf()`

`scanf()` is used to take values from the user and store them in variables.

1. Taking One Input

```c
#include <stdio.h>

int main() {
    int age;
    printf("Enter your age: ");
    scanf("%d", &age);     // & means address of
    variable
    printf("You entered: %d", age);
    return 0;
}
```

```c
#include <stdio.h>

int main() {
    float percentage;
    printf("Enter your age: ");
    scanf("%f", &percentage);     // & means
    address of variable
    printf("You entered: %f", percentage);
    return 0;
}
```

```c
#include <stdio.h>

int main() {
    char grade;
    printf("Enter your age: ");
    scanf("%c", &grade);     // & means address
    of variable
    printf("You entered: %c", grade);
    return 0;
}
```

2. Taking Multiple Inputs

```c
#include <stdio.h>

int main() {
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    printf("%d", a + b);
    return 0;
}
```

Taking Float Input

```c
#include <stdio.h>

int main() {
    float height;
    printf("Enter your height in meters: ");
    scanf("%f", &height);
    printf("Height = %f meters", height);
    return 0;
}
```

6.3. Reading and Displaying Characters: `getchar()` and `putchar()`

These functions are used when we want to read or display single characters.

Using `getchar()`

`getchar()` reads a single character typed by the user.

```c
#include <stdio.h>

int main() {
    char ch;
    printf("Enter a character: ");
    ch = getchar();
    printf("You entered: ");
    putchar(ch);
    return 0;
}
```

Using `putchar()`

`putchar()` prints one character on the screen.

You can also use it like this:

```c
#include <stdio.h>

int main() {
    putchar('A');  // prints A
    return 0;
}
```

7. Input and Output Formatting in C

Formatting means **controlling how input and output looks** in a C program.

When you use `printf()` to display values or `scanf()` to take input, you can use **format specifiers** and other formatting rules to:

- Print values neatly
- Align output
- Control decimal places
- Set width and spacing

7.1. Controlling Output Formatting with `printf()`

1. Printing with Decimal Precision

You can control how many digits you want after the decimal point using `.nf`, where `n` is the number of digits.

```c
#include <stdio.h>

int main() {
    float pi = 3.14159;
    printf("%.2f\n", pi);  // prints: 3.14
    printf("%.4f\n", pi);  // prints: 3.1416
    return 0;
}
```

```
3.14
3.1416
```

2. Width Formatting (Right-Aligned)

You can print numbers in a fixed space (width) using `%w.df`, where `w` is width and `d` is decimal places.

```c
#include <stdio.h>

int main() {
    float num = 12.5;
    printf("%8.2f\n", num);  // prints in 8
    spaces, right-aligned
    return 0;
}
```

```
   12.50
```

3. Left-Aligned Output

To left-align the output, use a minus sign `-` inside the format.

```c
#include <stdio.h>

int main() {
    float num = 12.5;
    printf("%-8.2fEnd\n", num);  // left-aligned
     in 8 spaces
    return 0;
}
```

```
12.50   End
```

4. Printing Multiple Values with Format

You can format and label multiple values neatly in a single line.

```c
#include <stdio.h>

int main() {
    int roll = 101;
    float marks = 88.55;
```

```
    printf("Roll: %d\tMarks: %.1f\n", roll,
    marks);
    return 0;
}
```

```
Roll: 101    Marks: 88.6
```

7.2. Formatting Input with `scanf()`

`scanf()` also uses format specifiers to read input correctly. Each variable must match the correct format.

1. Reading Multiple Values with Proper Format

```
#include <stdio.h>

int main() {
    int age;
    float salary;

    printf("Enter your age and salary: ");
    scanf("%d %f", &age, &salary);
    printf("Age: %d, Salary: %.2f", age, salary)
    ;
    return 0;
}
```

```
23 55000.75
```

```
Age: 23, Salary: 55000.75
```

7.3. Using `\n` and `\t` in Output

In C programming, special characters called **escape sequences** are used to control how the output appears on the screen.

Two commonly used escape sequences are:

- `\n` – New Line
- `\t` – Tab Space

1. `\n` – New Line

The `\n` character is used to move the output to the next line, just like pressing "Enter" on the keyboard.

```
#include <stdio.h>

int main() {
    printf("Hello\nWorld");
    return 0;
}
```

```
Hello
World
```

2. `\t` – Tab Space

The `\t` character adds a horizontal tab (a few spaces), just like pressing the "Tab" key.

```
#include <stdio.h>

int main() {
    printf("Name:\tRahul\n");
    printf("Age:\t20");
    return 0;
}
```

```
Name:    Rahul
Age:     20
```

Why Use Them?

- `\n` makes output more readable by separating lines.
- `\t` helps align text in columns.

Example: Complete Use of Input and Output Formatting

```
#include <stdio.h>

int main() {
    char name[20];
    int age;
    float marks;

    printf("Enter your name, age, and marks: ");
    scanf("%s %d %f", name, &age, &marks);

    printf("\n--- Student Info ---\n");
    printf("Name : %s\n", name);
    printf("Age  : %d years\n", age);
    printf("Marks: %.2f\n", marks);

    return 0;
}
```

```
Rahul 20 87.45
```

```
--- Student Info ---
Name : Rahul
Age  : 20 years
Marks: 87.45
```

8. Programming Exercises

Problem 1: Write a C program to print the following information using `printf()`:

- Your Name
- Your Date of Birth
- Your Mobile Number

```
Name   : Rahul Kumar
DOB    : 15/08/2000
Mobile : 9876543210
```

Problem 2: Write a C program to print the letter **F** using the **#** symbol, with a height of 6 characters. The letter F should have a full top and middle horizontal line, and a single vertical line on the left.

```
#####
#
#
####
#
#
```

Solution

```c
#include <stdio.h>

int main() {
    printf("#####\n");
    printf("#\n");
    printf("#\n");
    printf("####\n");
    printf("#\n");
    printf("#\n");
    return 0;
}
```

Problem 3: Write a C program to print a large letter **C** using the **#** symbol. The letter C should be open on the right side and curved from the top, side, and bottom.

```
  #####
 #
 #
 #
 #
  #####
```

Problem 4: Write a C program to read an integer from the user and print it.

Expected Input

```
Enter an integer: 45
```

Expected Output

```
You entered: 45
```

Solution:

```c
#include <stdio.h>

int main() {
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);
    printf("You entered: %d", number);
    return 0;
}
```

Problem 5: Write a C program to read a floating-point number from the user and print it with two decimal places.

Expected Input:

```
Enter a number: 23.456
```

Expected Output:

```
Value = 23.46
```

Solution:

```c
#include <stdio.h>

int main() {
    float value;
    printf("Enter a number: ");
    scanf("%f", &value);
    printf("Value = %.2f", value);
    return 0;
}
```

Problem 6: Write a C program to take a single character input (e.g., grade) and display it.

Expected Input:

```
Enter your grade: A
```

Expected Output:

```
Your grade is: A
```

Solution:

```c
#include <stdio.h>

int main() {
    char grade;
    printf("Enter your grade: ");
    scanf(" %c", &grade);  // Space before %c to
     consume leftover newline
    printf("Your grade is: %c", grade);
    return 0;
}
```