Kickstart

C Programming Fundamentals

Learn C Programming Basics, Pointers, Data Structures, Memory Management, and File Handling with Hands-On Examples and Practice Exercises

Dr. Dipra Mitra

Copyright © 2026 Orange Education Pvt Ltd, AVA®

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author nor **Orange Education Pvt Ltd** or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Orange Education Pvt Ltd has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capital. However, **Orange Education Pvt Ltd** cannot guarantee the accuracy of this information. The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

First Published: May 2026

Published by: Orange Education Pvt Ltd, AVA®

Address: 9, Daryaganj, Delhi, 110002, India

275 New North Road Islington Suite 1314 London,

N1 7AA, United Kingdom

ISBN (PBK): 978-93-49887-63-3

ISBN (E-BOOK): 978-93-49887-23-7

Scan the QR code to explore our entire catalogue

www.orangeava.com

Table of Contents

1. Introduction to C Programming
- Introduction
- Structure
- Why Learn C
  - *Basic Features of C*
- Introduction to Components of a Computer System
- Operating System
- Batch Operating System
- Time-Sharing (Multitasking) Operating System
- Distributed Operating System
- Network Operating System
- Real-Time Operating System
- Mobile Operating System
- Embedded Operating System
- Multi-User and Single-User Operating Systems
- Multiprocessing Operating System
- Compilers
- Interpreter
- Assembler
- Algorithm
- Example of Algorithm: Find the Largest of Three Numbers
- Conclusion
- Exercises

2. Arithmetic Expressions and Precedence
- Introduction
- Structure
- Identifiers
- Types
- Constants
  - *Symbolic Constants*
  - `printf` *Conversion Specifiers*
- Declarations

Arithmetic Operations
Relational and Logical Operations
Bitwise Operators
Type Conversions and Casts
Conclusion
Exercises

3. Branching and Iteration
Introduction
Structure
Branching
Conclusion
Exercises

4. Arrays
Introduction
Structure
Introduction to Arrays
One-Dimensional Array (1-D Array)
Two-Dimensional Array (2-D Array)
Multi-Dimensional Array (M-D Array)
Array Traversal
Array of Pointers
Pointer to an Array
Conclusion
Questions

5. Algorithms
Introduction
Structure
Characteristics of a Good Algorithm
Representations of Algorithms
Classification of Algorithms
Common Algorithms with Examples
Searching Algorithms
Linear Search
Binary Search

Sorting Algorithms
Bubble Sort
Complexity
Asymptotic Notation
Big-O Notation (ο)
Omega Notation (Ω)
Theta notation(Θ)
Conclusion
Questions

6. C Function
Introduction
Structure
Types of Functions
Library Functions
User-Defined Functions
Formal Parameters and Actual Parameters
Passing Parameters to Functions or Types of Argument Passing
Call by Value
Call by Address
Storage Classes
Global Variables
Local Variables (Automatic Variables)
Static Variables
Register Variables
Program 1: A Given Number is an Armstrong Number or Not Using a Function
Program 2: A Given Number is a Palindrome or Not Using a Function
Conclusion
Multiple Choice Questions
Questions

7. C Recursion
Introduction
Structure
Stack Frame Explanation

- Types of Functions
  - *Direct Recursion*
  - *Indirect Recursion*
  - *Tail Recursion*
  - *No Tail/Head Recursion*
  - *Head Recursion*
- Quicksort Using Recursion
  - *Complexity Analysis of QuickSort*
- Merge Sort Using Recursion
- Power of a Number Using Recursion
- Conclusion
- Exercises

8. Structures and Unions

- Introduction
- Structure
- Defining and Declaring Structures
  - *Declaration and Syntax*
  - *Creating Structure Variables*
  - *Initialization of Structures*
- Initialization and Accessing Structure Members
  - *The Dot Operator (.)*
  - *Sample Program: Structure Input and Output*
- Memory Layout and Structure Padding
  - *Contiguous Storage*
  - *Structure Padding*
  - *Sample Program: Verifying Padding*
  - *Structure Assignment (Copying)*
  - *Sample Program: Structure Copy*
- Array of Structures
  - *Definition*
  - *Accessing Array Elements*
  - *Sample Program: Creating a Student Database*
- Pointers to Structures
  - *Definition*
  - *Sample Program: Pointer to Structure*
- Nested Structures

Accessing Nested Members
Sample Program: Nested Structures
Passing Structures to Functions
Pass by Value
Pass by Reference (Using Pointers)
Sample Program: Passing Structures
Unions and their Memory Management
Definition
Memory Allocation in Unions
Accessing Union Members
Sample Program: Union Behavior
Conclusion
Questions
Programming Exercises

9. Pointers
Introduction
Structure
Definition and Concepts
Declaration of Pointers
Core Operators: & and *
Sample Program: Basic Pointer Usage
Why Typed Pointers (int* versus float*)
Pointer Arithmetic
The Rule of Scaling
Valid Operations
Pointers and Arrays
The Array Name as a Pointer
Accessing Array Elements
Sample Program: Traversing an Array Using Pointers
Functions and Pointers: Call by Reference
The Problem with Call by Value
The Solution: Call by Reference
Pointer to Pointer (Double Pointers)
Generic Pointers (void *)
NULL Pointers
Dangling Pointers

- *Returning the Address of a Local Variable*
- *Pointer to Freed Memory*
- Wild (Uninitialized) Pointers
- Const Correctness with Pointers
- `sizeof` Pitfalls: `sizeof(ptr)` vs `sizeof(*ptr)`
- NULL vs 0 in C
- Conclusion
- Questions
- Programming Exercises

10. File Handling and String Handling in C

- Introduction
- Structure
- Definition and Concepts
  - *Types of Files*
  - *The `FILE` Pointer*
  - *Opening a File (`fopen`)*
  - *Closing a File (fclose)*
  - *Sample Program: Creating and Opening a File*
- Reading and Writing Text Files
  - *Character I/O (`fputc` and `fgetc`)*
  - *String I/O (`fputs` and `fgets`)*
  - *Formatted I/O (`fprintf` and `fscanf`)*
  - *Sample Program 1: Writing to a File*
  - *Sample Program 2: Reading from a File*
- Reading and Writing Binary Files
  - *Writing Binary Data: `fwrite()`*
  - *Reading Binary Data: `fread()`*
  - *Sample Program: Writing Structures to a Binary File*
  - *Sample Program: Reading Structures from a BinaryFile*
- Random Access in Files
  - *rewind()*
  - *ftell()*
  - *fseek()*
  - *Sample Program: Random Access*
- Error Handling in File Operations
- String Handling

Declaration and Initialization of Strings
Input and Output of Strings
`scanf()` Function
`gets()` Function (Unsafe / Deprecated)
`puts()` Function
C Program Using Common String Functions
Conclusion
Questions
Programming Exercises

Index

CHAPTER 1

Introduction to C Programming

Introduction

The history of C programming can be traced back to the 1960s, when computer scientists began exploring high-level languages for system development. One such language was **Basic Combined Programming Language (BCPL)**, developed by **Martin Richards** in the year 1966. It was a typeless language designed mainly for writing system software. This inspired the creation of a simpler language called **B**, developed by **Ken Thompson** at Bell Labs in 1969, which was used primarily for early versions of the Unix Operating System. However, B had limitations, especially in terms of data handling and type structures. To overcome the limitations of B, **Dennis Ritchie** at Bell Labs developed the **C programming language** in the year **1972**. C retained the simplicity of B, but introduced powerful features like data types, structured programming constructs (such as loops and conditionals), and pointer-based memory access. This made C a much more flexible and an efficient tool for system-level programming, specifically for writing operating systems.

Today, C continues to play a pivotal role in software development. It is widely used in operating systems, embedded systems, compilers, device drivers, and performance-critical applications. Many modern programming languages such as C++, Java, and Python have been influenced by C. Its efficiency and close-to-hardware capabilities make it an essential language for understanding computer systems and low-level programming.

Structure

In this chapter, we will cover the following topics:

- Basic Concept of C Programming
- Structure of the C Program
- Basic Concept of the Operating System and Its Type

- Introduction to Components of a Computer System
- Operating System

Why Learn C

Foundational Language: Many modern languages including C++, Java, and Python are influenced by C.

- **High Performance**: C allows direct access to memory and hardware, making it suitable for system-level programming.
- **Portability**: C programs can run on many different types of systems with little or no modification.
- **Understanding of Computer Fundamentals**: Learning C helps you understand how computers work at a lower level, such as memory management and data storage.

Basic Features of C

Structured Language: It organizes code using functions and control structures.

- **Rich Set of Operators**: Supports arithmetic, logical, relational, and bitwise operations.
- **Modularity**: Code can be broken into functions for better readability and maintenance.
- **Efficient**: Programs written in C are fast, and uses system resources quite effectively.

Structure of a Simple C Program

```
#include <stdio.h>
 int main()
  {
printf("Hello, World!\n");
    return 0;
  }
```

The preceding code encompasses:

- **Preprocessing:** The directive #include is substituted with the contents of `stdio.h`.
- **Compilation**: The code is converted into assembly instructions for x86 or x86_64 architectures.
- **Assembly:** Assembly is transformed into object code (`.obj` file).
- **Linking:** Establishes connections with the C runtime (for example, `msvcrt.dll`). Generates a Windows PE (Portable Executable) file. greetings.exe.

Key Concepts Introduced:

- **Syntax and Semantics**: Understanding how to write correct C code.
- **Data Types and Variables**: Declaring and using variables of different types.
- **Input and Output**: Using `scanf ()` and `printf ()` for user interaction.
- **Control Structures:** Using conditions and loops to control program flow.
- **Functions:** Breaking code into reusable blocks.

Before learning the C programming language in depth, it is important to understand the basic components of a computer system. Since C interacts closely with hardware and system resources, having a clear understanding of how a computer works will help learners better grasp programming concepts.

Introduction to Components of a Computer System

A computer system is a combination of hardware and software that work together to perform tasks and process data. Understanding the key components of a computer system is essential for anyone beginning their journey in computer science or information technology.

- **Input Unit:** The input unit comprises of devices connected to the computer for data entry. These devices receive input and translate it into a binary code which is comprehensible to the computer. Common

input devices include the keyboard, mouse, joystick, and scanner. The Input Unit consists of one or more input devices connected to a computer. A user provides data and commands using input devices, including a keyboard and mouse. The input unit supplies data to the processor for subsequent processing.

- **Central Processing Unit:** The Central Processing Unit (CPU) serves as the computer's brain, governing all its operations. Upon inputting data via an input device, the CPU processes the information. Initially, it retrieves instructions from memory, subsequently decoding them to comprehend the required actions. It pulls data from the memory or an input device when necessary. Subsequently, the CPU does the work and either retains the result or presents it on an output device. The CPU comprises of three primary components: the Arithmetic Logic Unit (ALU), responsible for computations and logical operations; the Control Unit (CU).
- **Arithmetic and Logic Unit (ALU):** The ALU performs arithmetic and logical operations. Arithmetic operations include addition, subtraction, multiplication, and division. Logical operations involve comparing data values to determine relationships such as greater than, less than, or equal to. The ALU is a key component of the CPU and is responsible for carrying out calculations required by programs.
- **Control Unit:** The Control Unit orchestrates and regulates the data flow to and from the CPU, while also managing the activities of the ALU, memory registers, and input/output units. It is also tasked with executing all the commands stored in the program. The system decodes the retrieved instruction, interprets it, and transmits control signals to input/output devices until the necessary operation is accurately executed by the ALU and memory. The Control Unit is a component of the central processing unit that orchestrates the operations of the processor. It directs the computer's memory, arithmetic and logic unit, and input and output devices on how to execute the processor's commands. The computer components receive signals from the control unit to execute instructions. It is also referred to as the central nervous system, or the brain of the computer.
- **Memory Registers:** A register is a compact, ephemeral memory component within the CPU. The processor utilizes it to retain data that it is presently processing. Registers are available in several sizes,

including 16-bit, 32-bit, and 64-bit, each serving a distinct function. Some store data, some store instructions, while some retain memory addresses. The Accumulator (ACC) is a crucial register within the CPU. It contains one of the values utilized in computations within the Arithmetic and Logic Unit (ALU). In addition to registers, the internal memory, often known as primary or main memory, temporarily stores the data and instructions during program execution. This memory is referred to as the Random Access Memory (RAM). Each data element in RAM is assigned a distinct address, enabling the CPU to access it swiftly without the need to scan the entire memory. RAM is termed Random Access Memory because it allows immediate access to any data location. The Memory Unit serves as the principal storage component of the computer. It retains both data and instructions. Data and instructions are permanently kept in this unit for immediate accessibility when needed.

- **Output Unit:** The output unit comprises devices connected to the computer for output purposes. It translates the binary data from the CPU into a comprehensible format for humans. The prevalent output devices include monitors, printers, and plotters. The output unit presents or reproduces the processed data in an accessible format for users. It is created by connecting the output devices of a computer. The output unit receives data from the CPU, and presents it in a format comprehensible to the user.
- **Motherboard:** The motherboard serves as the central hub of a computer, interlinking essential components such as the CPU, memory, and storage. It facilitates power distribution, information transfer, and device connectivity, including peripherals such as a mouse, keyboard, or monitor. A computer specialist may open the PC to inspect for any loose or damaged connections, such as corrosion, if there is an issue with the motherboard. They may also examine the power supply to verify that the computer is receiving electricity correctly.
- **Random Access Memory (RAM):** RAM serves as the computer's ephemeral memory, temporarily housing data while applications are operational. For instance, upon opening an application, it is loaded into RAM, facilitating rapid access by the computer. A specialist is proficient in identifying the type of RAM in a computer, replacing it if found defective, and resolving data transfer issues within memory.

They comprehend several forms of RAM and prevalent issues that may impact it. Prior to repairing RAM, a technician may back up essential files to avert the loss of critical programs or documents.

- **Power Supply Unit (PSU):** The Power Supply Unit (PSU) provides electrical power to all components of a computer. It typically links the computer to a wall outlet via a power cable. In the event of a malfunction, a technician may diagnose the issue by powering down the computer, disconnecting the power cord, or testing an alternative cord or socket to ascertain whether the fault is with the power supply or not.
- **Attributes of a Computer Velocity:** Computers can do millions of computations per second. The processing speed is quite rapid.

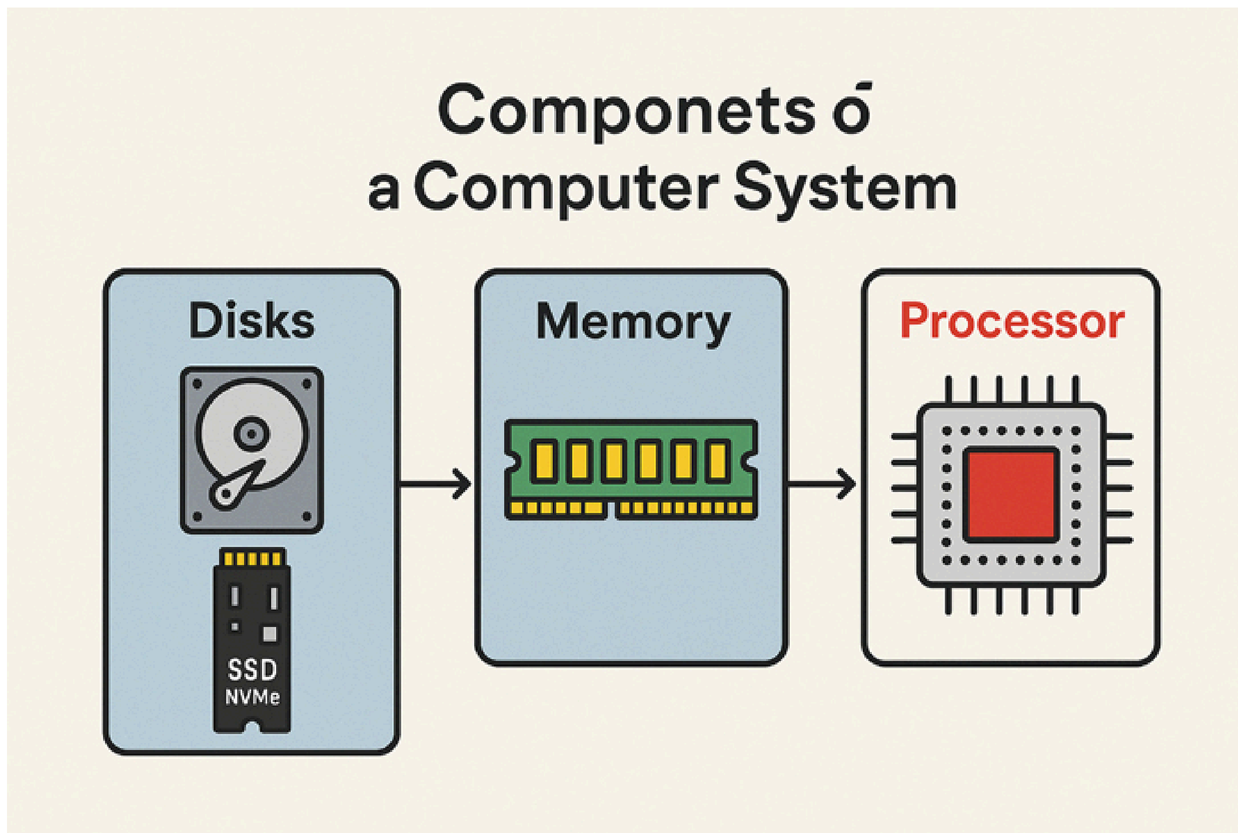

Figure 1.1: Components of a Computer

Main Components of a Computer System:

- **Hardware**

Hardware refers to the physical parts of a computer that you can see and touch. It includes:

  - **Input Devices**: Allow users to enter data into the computer (such as keyboard, mouse, and scanner).
  - **Output Devices**: Display or output information to the user (which includes monitor, printer, or, speakers).
  - **Storage Devices**: Store data permanently or temporarily (namely the hard drive, SSD, and USB flash drive).
  - **Processing Unit (CPU)**: The brain of the computer that processes instructions and performs calculations.
  - **Memory (RAM)**: Temporary storage that holds data and instructions while the computer is running.

- **Software**

  Software is a set of instructions or programs that tell the hardware what to do. It includes:

  - **System Software**: Manages hardware and provides a platform for running application software (includes operating systems such as Windows or Linux).
  - **Application Software**: Programs designed to perform specific tasks for the users (such as word processors, browsers, or games).

- **Peopleware (Users)**

  The users or operators of the computer system. They interact with the system to perform various tasks and make decisions.

- **Data**

  The raw facts and figures that are processed by the computer to produce meaningful information.

- **Networking Components**

  These include devices and technologies that allow computers to communicate with each other, such as routers, modems, and network cards.

A **Hard Disk Drive (HDD)** is a non-volatile data storage device used in computers and other electronic devices to store and retrieve digital information using magnetic storage.

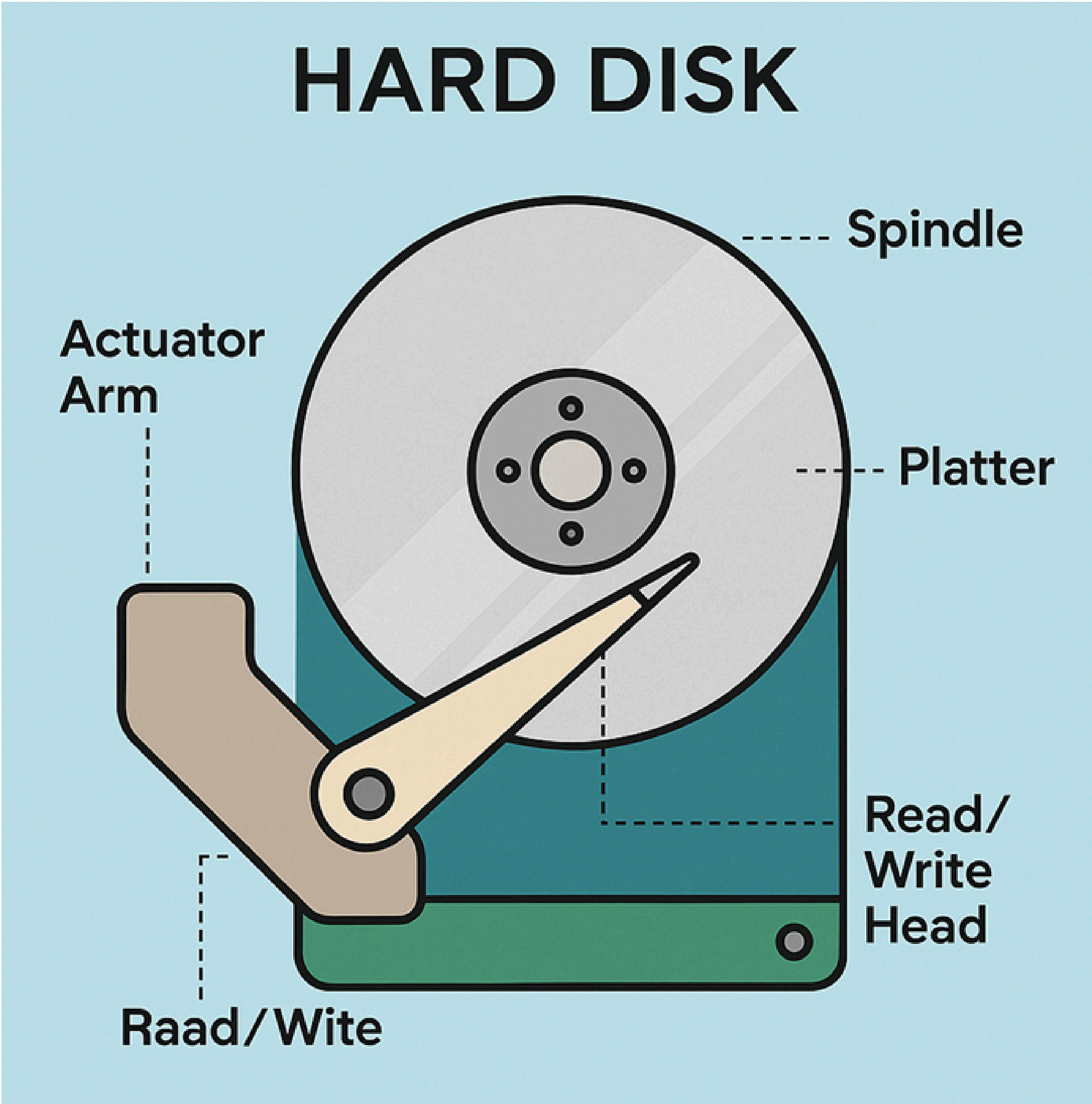

Figure 1.2: Structure of a Hard Disk Drive

Key Features:

- **Structure:** Consists of one or more **spinning disks** (called platters) coated with magnetic material. These platters rotate at high speeds (typically 5400 or 7200 RPM in consumer devices).
- **Read/Write Head:** A small actuator arm with read/write heads moves over the platters to access data. It floats just above the surface on a cushion of air.

- **Magnetic Storage:** Data is stored magnetically in binary format (0s and 1s), making it persistent even when the device is powered off.
- **Interface:** Connects to the computer via interfaces like **Serial ATA (Serial ATA),** or older **IDE** standards.

Advantages:

- **Large Storage Capacity:** Commonly available in sizes from 500 GB to 10+ TB.
- **Cost-Effective:** Lower cost per gigabyte as compared to SSDs.

Disadvantages:

- **Slower Speed:** Slower data access and boot times compared to SSDs.
- **Mechanical Parts:** More prone to failure due to moving components.
- **Heavier and Noisier:** Not ideal for portable or silent computing environments.

Common Uses:

- Desktop and laptop computers (especially budget models)
- External backup drives
- Data centers and servers (for bulk storage)

Operating System

An operating system is a piece of software that handles the administration of the hardware that is found in a computer. It acts as a mediator between the user of the computer and the hardware, creating a basis for the application programs, and also serving as a foundation for them. It is quite remarkable that operating systems can perform a wide variety of functions, which is one of its most amazing characteristics. The primary purpose of mainframe operating systems is to make the most of the hardware. Operating systems for Personal Computers (PCs) make it possible to run complex games, commercial software, and a wide variety of other applications. The purpose of the operating systems that are designed for portable computers is to make it easier for the users to interface with the device in order to execute the programs. Consequently, some operating systems are designed to be convenient, while others are designed to be efficient, and yet others are

designed to be a combination of these two characteristics. In order to properly comprehend the structure of the computer system, it is necessary to first get an understanding of the intricate workings of the system. To get started, we will investigate its essential functions, which include input/output, storage, and initialization function. In addition, we shall provide an outline of the fundamental computer architecture that is necessary for the development of an operating system that is functional. Considering the significant complexity of an operating system, its development must be carried out in steps that are incremental. Every component must be able to accurately represent a distinct section of the system, complete with inputs, outputs, and functions that have been properly established. In this chapter, a comprehensive overview of the primary components that make up an operating system is presented.

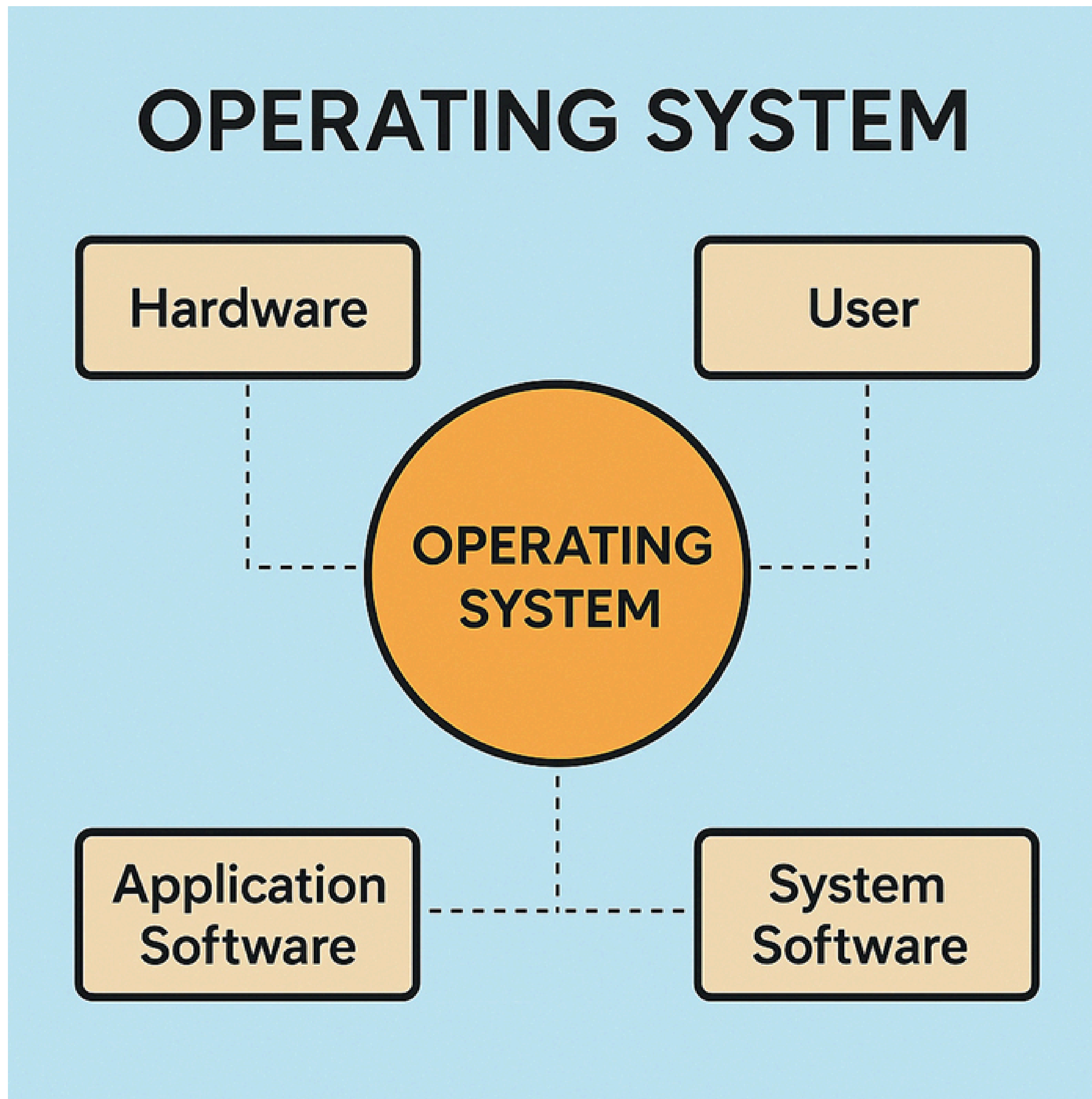

Figure 1.3: *Structure of an Operating System*

Batch Operating System

One of the earliest forms of operating systems—a Batch Operating System is mostly utilized in mainframe computers. It is also one of the oldest types totals. It was designed to carry out a sequence of tasks (programs) in an automated fashion, without requiring any involvement from a human.

- **Example:** IBM OS/360

- **Used in:** Early computing, large mainframes.

Time-Sharing (Multitasking) Operating System

The Time-Sharing Operating System, which is also known as a Multitasking Operating System, is a type of operating system that enables numerous users or processes to use the system at the same time by swiftly switching the CPU between them. There is a little "time slice" of CPU time allocated to each user or job.

- **Example:** UNIX, Linux, Windows (modern versions)
- **Used in:** Multi-user systems, general computing.

Distributed Operating System

A Distributed Operating System, sometimes known as a DOS, is a type of a computer operating system that controls a network of separate computers and displays them to the user as a single, unified system. The purpose is to ensure that the sharing of resources, coordination, and transparency are uninterrupted across a number of different machines.

- **Example:** LOCUS, Amoeba
- **Used in:** Networked environments, cloud computing.

Network Operating System

Computers are able to connect with one another, exchange files, printers, and other devices over a Local Area Network (LAN) or larger networks, thanks to a piece of software known as the Network Operating System (NOS). This software is responsible for managing network resources. On the other hand, with a Network Operating System (NOS), every machine has its own Operating System (OS), and network communication is placed on top of that for the purpose of resource sharing and connectivity.

- **Example:** Novell NetWare, Windows Server
- **Used in:** Servers, enterprise networks.

Real-Time Operating System

As a result of its ability to analyse data and respond to inputs within a predetermined amount of time, a Real-Time Operating System (RTOS) is exceptionally well-suited systems in which timing is of the utmost importance. In embedded systems, when delays or unpredictability can lead to a failure or hazard, it is frequently employed that this technology be used.

- **Example:** VxWorks, FreeRTOS, RTLinux
- **Used in:** Embedded systems, robotics, medical devices.

Mobile Operating System

A Mobile Operating System is a type of operating system that is meant to run on mobile devices such as smartphones, tablets, and other portable devices, including mobile phones. It is possible for the users to interact with the device and run programs as a result of the software platform and the interface that it provides.

- **Example:** Android, iOS
- **Used in:** Mobile and smart devices.

Embedded Operating System

When it comes to embedded systems, which are computing devices that are a component of a larger system or product and typically have dedicated functionality, an embedded operating system, also known as an embedded OS, is a specialized operating system that was built exclusively for embedded systems. It is common for these systems to carry out particular activities, frequently with real-time limitations, and they are designed to function continuously with a high degree of dependability and a minimum amount of resource consumption. An embedded operating system, in contrast to general-purpose operating systems such as Windows or Linux, is tailored for particular hardware, restricted resources (central processing unit, memory, and storage), and unique, specialized use cases.

- **Example:** QNX, Embedded Linux, Windows IoT
- **Used in:** Appliances, cars, industrial machines.

Multi-User and Single-User Operating Systems

As its name suggests, a Single-user Operating System is intended for use by a single individual at a time. Due to the fact that these systems are designed for individual use, it is not possible for several users to access them at the same time. Although the system may be capable of multitasking, which means it may execute numerous programs at the same time, it can only serve a single user at any given time.

- **Multi-user:** Allows multiple users to access the computer simultaneously. *Example: UNIX, Windows Server*
- **Single-user:** Designed for one user at a time. *Example: Windows 10, macOS*

Multiprocessing Operating System

An operating system that can support and manage many Central Processing Units (CPUs) simultaneously within a single computer system is referred to as a Multiprocessing Operating System (MPU OS). There is an increase in the system's performance, efficiency, and reliability because of its capacity to support parallel processing by several processors. It is an usual practice to employ this kind of operating system in settings that call for a significant amount of processing capacity, such as scientific computers, enterprise servers, and contemporary multi-core systems.

- **Example:** Linux, Windows Server with multiple CPUs
- **Used in:** High-performance computing.

Compilers

A Compiler is a software application that converts high-level programming languages, such as C, C++, or Java, into machine language (binary code) that a computer can comprehend and execute.

This translation occurs prior to the program execution, resulting in an executable file. This enables the software to execute swiftly and autonomously from the original source code.

Function of Compiler:

- **Translation:** Transforms source code into machine code.

- **Error Checking:** Identifies syntactic and semantic issues prior to execution.
- **Optimization:** Enhances the code performance and efficiency.
- **Code Generation:** Generates an executable program.

Interpreter

An Interpreter is a program that executes code sequentially, converting high-level programming language into machine code during runtime, rather than compiling the full code simultaneously.

Assembler

An Assembler is a program that converts assembly language, a low-level human-readable format of machine instructions, into a machine code, which consists of binary instructions executable by a CPU.

It is a low-level computer language that does not use binary, but instead, uses mnemonics such as MOV, ADD, and SUB. Computers can only read machine code, which is made up of 0s and 1s. An assembler fills in the blanks.

Algorithm

An Algorithm is a systematic technique, or a collection of clearly specified instructions for addressing a particular problem or executing a task. It receives an input, executes a series of logical operations, and generates an output respectively.

An algorithm is a clearly delineated sequence of instructions intended to execute a given activity or resolve a particular issue. To qualify as an algorithm, a process must demonstrate the following essential characteristics:

Characteristic of an Algorithm:

- **Finiteness:** After a certain number of steps, the algorithm must exit. There must be a termination point; it cannot go forever.
- **Definiteness:** There needs to be a crystal-clear definition of each step. There should be no room for interpretation in the instructions.

- **Input:** An algorithm may possess zero or more inputs. Inputs are the values supplied externally prior to or during the execution of the algorithm.
- **Output:** An algorithm must generate a minimum of one output. The output is the consequence of processing the input via the algorithm**.**
- **Effectiveness:** Each process must be sufficiently fundamental to be executed, in theory, by an individual utilising solely paper and pencil. All processes must be practical and not excessively intricate.
- **Generality:** The algorithm needs to be appropriate to a category of issues rather than a singular special instance. It must yield accurate results for all valid inputs within the given problem domain.

Example of Algorithm: Find the Largest of Three Numbers

Step 1: Start

Step 2: Input three numbers: A, B, and C

Step 3: If A ≥ B and A ≥ C

then

Print A is the largest

Step 4: Else if B ≥ A and B ≥ C,

then

print B is the largest

Step 5: Else

print C is the largest

Step 6: End

Aspect	Algorithm	Flowchart
Definition	A detailed set of instructions for completing a task or resolving an issue.	A visual representation of the stages of a process or algorithm through the use of symbols.
Form	Pseudocode or straightforward language.	Displayed graphically using standardised shapes (for example, diamonds, rectangles).

Understandability	Needs a logical comprehension and interpretation of the instructions.	Visual representation facilitates comprehension.
Representation	A representation of logic that is text-based.	Diagrammatic or graphical representation.
Usage	Used for coding, problem-solving, and logical thinking.	Used for designing and explaining processes visually.
Modification	Easier to edit and update.	More time-consuming to revise, especially in complex diagrams.
Tool Requirement	Does not require any specific tools—can be written on paper or digitally.	Often requires diagramming tools or software for neat presentation.

Table 1.1: *Difference between Flowchart and Algorithm*

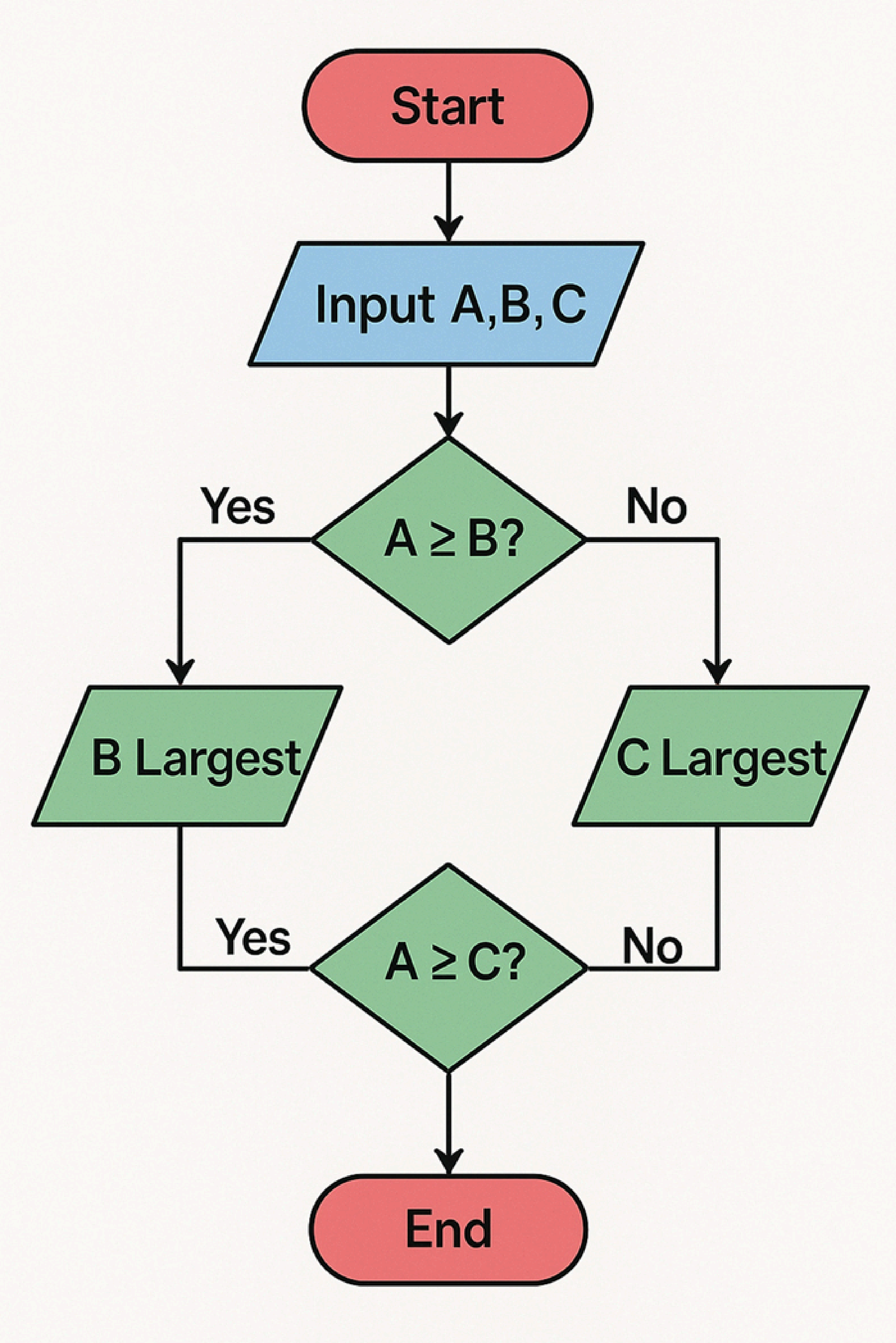
Start
Input A,B, C
A ≥ B?
Yes
No
B Largest
C Largest
A ≥ C?
Yes
No
End

Figure 1.4: *Largest of Three Numbers Flowchart*

Variable:

A memory location that stores data is referred to as a variable. The value of a variable may fluctuate during the execution of a program.

Syntax:

```
data_typevariable_name;
```

Example:

```
int length = 25;
float breadth = 5.9;
char height = 'A';
```

Rules for Creating Variables:

- Must begin with a letter (A-Z or a-z) or underscore _.
- Can contain letters, digits (0–9), and underscores.
- Case sensitive (age and Age are different).
- Cannot be a reserved keyword (such as int, return, and so on).

Data Type:

The type of data that a variable can store is determined by its data types. It instructs the compiler on the amount of space to allocate and the types of operations that can be performed on the data.

Basic Data Types in C:

Data Type	Description	Size (Typical)	Example
int	Integer values	4 bytes	int x = 10;
float	Decimal numbers	4 bytes	float y = 3.5;
double	Double-precision float	8 bytes	double z = 3.14159;
char	Single character	1 byte	char c = 'A';

Table 1.2: *Description of Primitive Data Types*

Format Specifiers:

Used in **`printf ()`** and **`scanf ()`** for input/output:

Data Type	Format Specifier

int	%d
float	%f
double	%lf
char	%c

Table 1.3: *Description of Primitive Data Types and Format Specifiers*

Example

```
1  #include <stdio.h>
2  int main() {
3      int age = 20;
4      float weight = 65.5;
5      char grade = 'B';
6
7      printf("Age: %d\n", age);
8      printf("Weight: %.2f\n", weight);
9      printf("Grade: %c\n", grade);
10
11     return 0;
12 }
```

Figure 1.5: *Code Window*

Output Window:

```
C:\Users\Dipra Mitra\Docume

Age: 20
Weight: 65.50
Grade: B

--------------------------------
Process exited after 0.09616 seconds with return value 0
Press any key to continue . . .
```

Figure 1.6: *Output Window*

Figure 1.5 illustrates the execution of a C program and how to declare variables, data types, and library functions, and *Figure 1.6* illustrates the output window. **`#include`** is a preprocessor directive in C that instructs the compiler to incorporate the Standard Input Output header file before compiling the actual code.

Conclusion

The first chapter of C programming provides a foundational understanding of the language, its structure, and its significance in modern computing. It introduces the history of C, its importance in system-level programming, and the basic structure of a C program.

The chapter has also explained essential programming concepts such as variables, data types, format specifiers, and input/output functions. In addition, the fundamental components of a computer system are discussed to help learners understand how C interacts with hardware and system resources.

Overall, this chapter establishes a strong base in both programming fundamentals, and computer concepts, preparing learners to write simple C programs, and advance toward more complex programming topics in the subsequent chapters.

Exercises

Theory Questions

1. What is the structure of a basic C program?
2. What are the main components of a C program?
3. Explain the purpose of the **`main()`** function in C.
4. What is the use of **`#include<stdio.h>`**?
5. What is the difference between a compiler and an interpreter?
6. Define keywords, identifiers, and constants in C.
7. What are variables in C? How do you declare them?
8. What are the rules for naming identifiers in C?
9. What are data types? List the basic data types in C.
10. What is the difference between int, float, char, and double?

Practical/Programming Questions

1. Write a C program to print "Hello, World!" on the screen.
2. Write a program to add two numbers entered by the user.
3. Write a C program to find the area of a circle given its radius.
4. Write a program to swap two numbers using a third variable.
5. Write a C program to convert temperature from Celsius to Fahrenheit.
6. Write a program to print your name, age, and grade.
7. Write a C program to calculate simple interest.
8. Write a program that takes an integer input from the user and prints whether it is positive, negative, or zero.
9. Write a program to calculate the sum and average of three numbers.
10. Write a program to find the square and cube of a number.

You’ve Just Finished your Free Sample

Enjoyed the preview?

Buy: http://www.ebooks2go.com