Parul® University

Subject: Fundamental of Computer Science

Unit-4 INTRODUCTION TO SYSTEM SOFTWARES AND PROGRAMMING LANGUAGES

Parul® University

1. Classification of Computer Languages

1. Machine Language (First Generation Language)
Written in **binary (0s and 1s)**.
Directly understood by the computer (no translation needed).
Very difficult for humans to read/write.
Example: 10110000 01100001

2. Assembly Language (Second Generation Language)
Uses **mnemonics (short codes)** instead of binary.
Needs an **Assembler** to convert into machine code.
Easier than machine language but still hardware-specific.
Example:
MOV A, 5
ADD B, A

Parul® University

1. Classification of Computer Languages

3. High-Level Languages (Third Generation Languages)
Closer to human language (English-like).
Needs a **compiler or interpreter**.
Machine-independent (can run on different systems with little modification).
Examples: C, C++, Java, Python, FORTRAN, COBOL.

4. Very High-Level Languages (Fourth Generation Languages – 4GL)
More abstract, closer to natural language.
Focuses on **problem-solving rather than hardware details**.
Often used in **databases, report generation, scripting**.
Examples: SQL, MATLAB, Perl.

5. Fifth Generation Languages (5GL)
Based on **artificial intelligence (AI) and logic programming**.
Users state **what they want**, not how to do it.
Used in **AI, Expert Systems, Neural Networks**.
Examples: Prolog, Lisp.

Parul® University

2. Introduction of Operating System

An **Operating System (OS)** is **system software** that acts as an **interface between the user and the computer hardware**.

It manages hardware, software, memory, and processes, making the computer usable.

Without an OS, a user cannot interact easily with the computer.

Parul® University

2. Introduction of Operating System

Evolution of Operating System

Operating Systems evolved over time to meet increasing needs:

Early Computers (No OS – 1940s)

Programs were entered manually (using switches, punch cards).

No automation, very slow.

Batch Processing Systems (1950s–1960s)

Jobs were collected in batches and executed sequentially.

Example: IBM Mainframes.

Multiprogramming & Time-Sharing (1960s–1970s)

Multiple programs loaded into memory at once.

Time-sharing allowed multiple users to use the system simultaneously.

Examples: UNIX (1969).

Personal Computer OS (1980s–1990s)

OS became user-friendly with **Graphical User Interface (GUI)**.

Examples: MS-DOS, Windows 95, macOS.

Modern OS (2000s–Present)

Multitasking, multiprocessing, cloud-based, mobile OS.

Examples: Windows 11, Linux, Android, iOS.

Parul® University

2. Introduction of Operating System

Functions of an Operating System

Process Management – Manages execution of programs (CPU scheduling, multitasking).
Memory Management – Allocates and deallocates memory to processes.
File Management – Handles storage, retrieval, and organization of files.
Device Management – Controls input/output devices (keyboard, printer, disk).
Security & Access Control – Protects data and system resources.
User Interface – Provides GUI (Graphical User Interface) or CLI (Command Line Interface).

Parul® University

2. Introduction of Operating System

Types of Operating Systems

Batch OS – Executes jobs in batches (old systems).
Time-sharing OS – Multiple users share system at the same time.
Distributed OS – Manages group of computers as a single system.
Real-time OS – Used where quick response is needed (medical systems, robotics).
Mobile OS – Designed for smartphones (Android, iOS).

Parul®
University

2. Introduction of Operating System

1. Batch Operating System
Jobs are collected in batches and executed one after another.
No user interaction during execution.
Real-time Example: Payroll system in old banks, billing in early electricity boards, IBM Mainframe systems.

2. Time-Sharing Operating System
Multiple users share the CPU at the same time.
Each user gets a small time slice (CPU scheduling).
Real-time Example: University mainframes, UNIX systems in labs.

3. Distributed Operating System
Controls multiple computers and makes them appear as a single system.
Resources are shared across connected machines.
Real-time Example: LOCUS, Amoeba OS, Google's Android-based cloud services.

2. Introduction of Operating System

4. Real-Time Operating System (RTOS)
Used where **instant response** is needed.
Divided into:
Hard RTOS → Missing a deadline = system failure. (e.g., Flight control systems)
Soft RTOS → Small delays acceptable. (e.g., multimedia streaming).
Real-time Example:
Hard RTOS → VxWorks in spacecraft, QNX in medical equipment.
Soft RTOS → Windows CE, Mobile Robots.

5. Mobile Operating System
Designed for smartphones and tablets.
Handles touch input, sensors, and wireless communication.
Real-time Example: Android, iOS, Windows Mobile.

Parul® University

2. Introduction of Operating System

Examples of Operating Systems

Desktop/Laptop OS: Windows, Linux, macOS, Unix
Mobile OS: Android, iOS
Server OS: Windows Server, Red Hat Enterprise Linux

An **Operating System** is like a **manager** that coordinates all parts of a computer so that users and applications can work smoothly with hardware.

Parul® University

UNIX commands

Helps in managing files, processes, permissions, and system operations directly from the command line.

Command	Purpose / Use	Example
`pwd`	Show present working directory	`pwd`
`ls`	List files & directories	`ls -l`
`cd`	Change directory	`cd Documents`
`mkdir`	Create new directory	`mkdir myfolder`
`rmdir`	Remove empty directory	`rmdir oldfolder`
`rm`	Remove file	`rm file.txt`
`touch`	Create empty file	`touch new.txt`
`cp`	Copy file	`cp file1.txt copy.txt`
`mv`	Move/rename file	`mv old.txt new.txt`

Parul®
University

UNIX commands

`cat`	Display file contents	`cat notes.txt`
`more` / `less`	View file page by page	`less bigfile.txt`
`nano` / `vi`	Open text editor	`nano file.txt`
`chmod`	Change permissions	`chmod 755 script.sh`
`grep`	Search text in file	`grep "main" file.c`
`find`	Search for files	`find /home -name "*.txt"`
`ps`	Show running processes	`ps`
`kill`	Kill a process	`kill 1234`
`tar`	Archive files	`tar -cvf files.tar file1 file2`
`gzip`	Compress file	`gzip file.txt`
`ping`	Check network connectivity	`ping google.com`

Parul®
University

DIGITAL LEARNING CONTENT

UNIX commands

`man`	Show manual/help	`man ls`
`whoami`	Show current user	`whoami`
`date`	Show system date/time	`date`
`cal`	Show calendar	`cal`

Parul®
University

Features of a Good Programming Language

Simplicity – Easy to learn, read, and write (clear syntax).
Readability – Code should be understandable like plain English.
Efficiency – Should produce optimized programs (fast execution, low memory usage).
Portability – Programs should run on different hardware/OS with little or no modification.
Reliability – Must handle errors properly and give consistent results.
Flexibility – Should support multiple programming paradigms (procedural, OOP, functional).
Extensibility – Should allow new features/libraries to be added easily.
Security – Should provide features for safe coding (memory management, type checking).
Support for Abstraction – Should allow breaking problems into smaller modules/functions.
Community & Tool Support – Good documentation, libraries, IDEs, and large user community.

Parul®
University

Selection of a Good Programming Language

When choosing a language for a project, consider:

1.Application Domain –

1. Scientific apps → Fortran, MATLAB
2. Business apps → COBOL, SQL
3. Web apps → JavaScript, Python, PHP
4. System software → C, C++

2.Efficiency Requirements – If speed and memory are critical → C/C++.

3.Ease of Use – For beginners or rapid development → Python.

4.Portability – Need to run on multiple platforms → Java, Python.

5.Availability of Libraries/Frameworks – For AI/ML → Python; for web → JavaScript/Java.

6.Security Needs – Languages with strict type-checking and safety (e.g., Java, Rust).

7.Team Skills & Community Support – Choose a language your team knows and has good global support.

8.Maintainability – Language should support modularity and readability for long-term projects.

Parul®
University

DIGITAL LEARNING CONTENT

Development of Program

The **development of a program** (or program development) means creating computer software step by step to solve a problem. It is also called the **Program Development Life Cycle**

Problem Definition
Understand what needs to be solved.
Example: You want to make a program to add two numbers.
Analysis
Study the requirements in detail.
Example: Input → two numbers, Process → add them, Output → sum.
Design
Create the solution plan using **algorithms, pseudocode, or flowcharts**.
Example: Start → Input numbers → Add → Print result → End.

Parul® University

Development of Program

Coding (Implementation)
Write the program in a programming language (like C, Python, Java).

Testing and Debugging
Run the program, check results, and fix errors (bugs).

Documentation
Write details about how the program works for future reference.

Maintenance
Update and improve the program when requirements change or errors are found.

Parul® University

DIGITAL LEARNING CONTENT

Algorithm

An **algorithm** is a **step-by-step procedure** or **set of instructions** written in simple language to solve a specific problem.

Key Points about an Algorithm

It is **finite** (must end after a certain number of steps).
It is **clear and unambiguous** (each step should be understandable).
It is **effective** (gives the correct solution).
It is **language independent** (written in plain English, not in programming language).

Parul® University

DIGITAL LEARNING CONTENT

Algorithm

Algorithm for Addition of Two Numbers

Start
Input first number (A)
Input second number (B)
Add A and B → Store in SUM
Display SUM
Stop

Find the Largest of Two Numbers

Start
Input two numbers A and B
If A > B, then Largest = A
Else Largest = B
Display Largest
Stop

Algorithm

Check Whether a Number is Even or Odd

Start
Input Number (N)
If N mod 2 = 0, then print "Even"
Else print "Odd"
Stop

Calculate Area of a Rectangle

Start
Input Length (L) and Breadth (B)
Area = L × B
Display Area
Stop

Parul®
University

Flowchart

A **flowchart** is a **diagrammatic representation** of an algorithm.
It uses **symbols and arrows** to show the step-by-step flow of a process or program.

Key Features of a Flowchart
Uses **shapes** (symbols) to represent different actions.
Shows the **sequence of steps** using arrows.
Makes it easier to **visualize** how a program will work.
Simple to **understand** for both programmers and non-programmers.

Parul® University

Flowchart

Symbol	Name	Meaning / Use
● **Oval / Ellipse**	**Start / Stop**	Shows the beginning or end of a process
□ **Parallelogram**	**Input / Output**	Used for taking input (e.g., enter number) or showing output (e.g., print result)
□ **Rectangle**	**Process**	Represents a calculation or action (e.g., SUM = A + B)
◆ **Diamond**	**Decision**	Used for condition checking (Yes/No, True/False)
→ **Arrow (Connector line)**	**Flow line**	Shows the direction of steps

Example: To calculate the area of a circle

Algorithm:

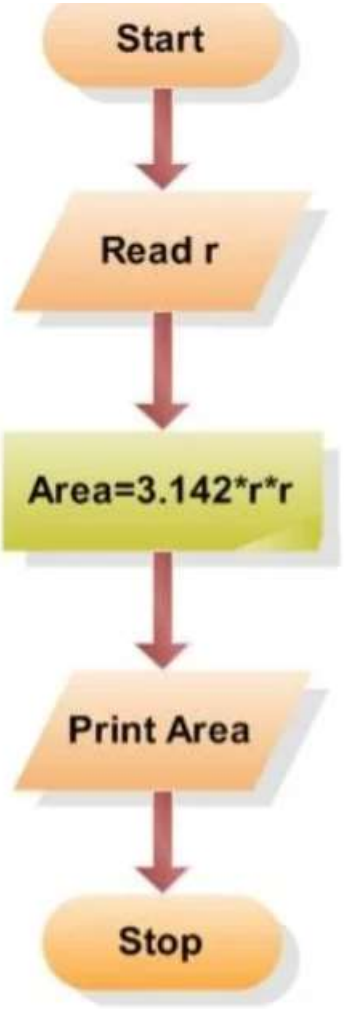
Step1: Start

Step2: Input radius of the circle say r

Step3: Use the formula πr2 and store result in a variable AREA

Step4: Print AREA

Step5: Stop



Example: To calculate the area of a circle

Algorithm:

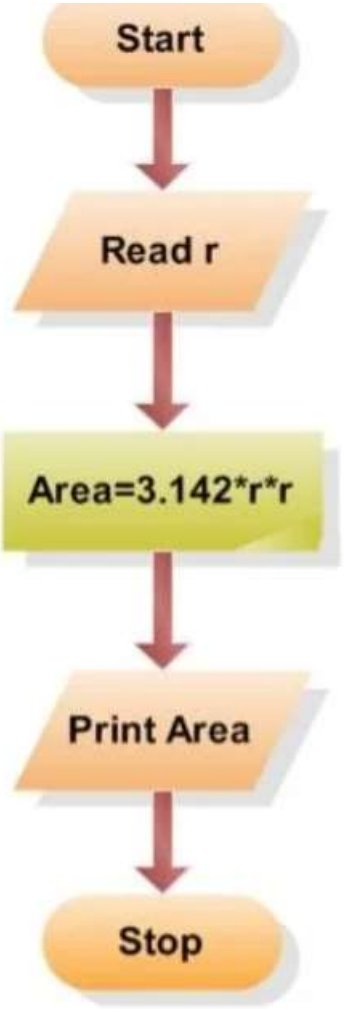
Step1: Start

Step2: Input radius of the circle say r

Step3: Use the formula πr2 and store result in a variable AREA

Step4: Print AREA

Step5: Stop



Parul® University

Program Testing and Debugging

Testing

When we develop a program, after coding we must **test it** before using it.
Purpose: To check if the program gives **correct results** for all possible inputs.

In Program Development: Testing ensures the program meets the requirements defined in the problem analysis phase.

Example:
If we write a program to add two numbers, we must test it with different inputs (positive numbers, negative numbers, zero) to confirm it always works.

Parul® University

Program Testing and Debugging

Debugging

During testing, if we find errors (bugs), we do **debugging**.

Purpose: To **find, analyze, and correct errors** in the program.

In Program Development: Debugging makes the program error-free and reliable before final delivery.

Example:
If the addition program is giving wrong output because we used - instead of +, debugging means finding that mistake and correcting it.

Parul® University

Program Documentation and Paradigms

Documentation = Record of program details.

Paradigms = Style/approach used to design and implement the program.

Parul® University

Program Documentation and Paradigms

Program documentation is the process of **writing detailed information** about a program so that others (or the programmer later) can easily understand, use, and maintain it.

Internal Documentation

Comments written inside the source code.

Example:

```
// This program adds two numbers
int sum = a + b;   // sum stores the result
```

External Documentation

Written outside the code (manuals, guides, reports).

Includes:

Problem definition

Algorithm & Flowchart

User manuals

Installation guides

Maintenance notes

Parul® University

Program Documentation and Paradigms

Importance of Documentation:

Makes program **easy to understand**.
Helps in **maintenance and debugging**.
Useful when **different programmers** work on the same project.
Provides **guidelines for users**.

Parul® University

Program Documentation and Paradigms

A **programming paradigm** is a style or way of programming (a method to structure and write code).

Paradigm	Focus	Example Languages
Procedural	Step-by-step procedures	C, Fortran
OOP	Objects (data + functions)	Java, C++
Functional	Functions, immutability	Haskell, Python
Logic	Rules and facts	Prolog

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in