CPE 102
Introduction to Computer Engineering

SOFTWARE TYPES AND FUNCTIONS
INTRODUCTION TO PROGRAMMING

Adesina Joshua Ayodele, Eng.
Department of Computer Engineering
University of Ilorin

2024/2025 Session

Last lecture recap

Definition and scope of Computer Engineering

Interdisciplinary nature (Electrical Engineering + Computer Science)

Logic gates (AND, OR, NOT, NAND, NOR)

Overview of computer hardware components

Central Processing Unit (CPU): Structure and function

Memory types: RAM, ROM, Cache

Storage devices: HDD, SSD, optical drives

Input and output devices: Keyboard, mouse, monitor, printer

Learning Outcomes: Identify key hardware components and their roles, Differentiate storage and I/O devices and their uses.

Software Types and Functions, Introduction to Programming

By the end of this course, students will be able to:

1. define and differentiate between Application and System software;

2. understand the working of utility software: Antivirus, disk management;

3. classify software types and explain their purpose;

4. understand the basics of programming: Algorithms, flowcharts;

5. understand the overview of programming languages: Highlevel (Python) vs. lowlevel (Assembly);

6. understand programming concepts.

Introduction to Software

Software is a set of instructions that tell a computer how to perform tasks. It is broadly categorized into:

System Software: Manages hardware and provides a platform for applications.

Application Software: Designed for end-users to perform specific tasks.

Operating systems Operating systems (OS) are a core part of system software, acting as an interface between users and hardware. They manage resources like memory, files, and input-output devices, enabling multitasking and security.

Windows, developed by Microsoft, is popular for personal computers, with versions like Windows 10 and 11 offering user-friendly interfaces and game support, though it can be expensive (e.g., Windows 11 costs 8000-12000 INR) and less secure.

Linux, open-source and free, is favored for servers and desktops, known for security and performance, but may feel less user-friendly for beginners. Both handle multitasking and security, with Windows using Kerberos authentication for privacy.

System Software

a. Operating Systems (OS)

Functions: Manages hardware (CPU, memory, storage).

Provides user interface (GUI/CLI).

Controls file systems, security, and multitasking.

1. Windows (Microsoft): User-friendly GUI (Start Menu, Taskbar).

Supports commercial software (e.g., MS Office).

Features: Registry, DirectX, compatibility modes. Versions: Windows 10, 11.

2. Linux (Open-Source): Kernel-based OS with distributions (Ubuntu, Fedora).

- CLI-centric (Bash shell) with optional GUIs (GNOME).

- Secure, lightweight, and customizable.

Windows is ideal for personal and office use, offering tools like calendars and word processors, but its cost and security issues, mitigated by Microsoft Defender in newer versions, are notable drawbacks.

S/N	ASPECT	DETAILS
1	Advantages	Hardware compatibility, pre-loaded software, ease of use, supports many games.
2	Disadvantages	Expensive (e.g., Windows 11), prone to viruses, may lag over time
3	Functions	Manages memory, files, multitasking, security (uses Kerberos authentication).

Linux: Derived from Unix, Linux is free and open-source, built around the Linux kernel, and widely used for desktops and servers. It’s known for high performance, requiring no periodic reboots, and is secure against malware. Top distributions include Ubuntu, Fedora, OpenSUSE, and RedHat, catering to developers for its flexibility.

S/N	ASPECT	DETAILS
1	Advantages	Open-source, free, high performance, secure, multitasking, flexible.
2	Disadvantages	Not user-friendly, not ideal for gamers, many versions can be confusing.
3	Functions	Manages resources efficiently, supports server environments, no reboot needed.

Linux leads on servers and mainframes, being the only OS on TOP500 supercomputers, making it a strong choice for technical users, though its adaptation curve can be steep for beginners.

Both OS types illustrate the diversity in system software, with Windows leaning toward **consumer ease** and Linux toward **technical flexibility**, as detailed in [Commonly Used Operating System](https://www.geeksforgeeks.org/commonly-used-operating-system/).

System Software

b. Device Drivers

Drivers are like translators, enabling hardware like printers or keyboards to communicate with the operating system. Examples include drivers for character devices (mouse, keyboard) and block devices (USB drives, hard drives), ensuring smooth hardware interaction.

Drivers are hardware-dependent and OS-specific, crucial for system operation

Purpose: Act as translators between OS and hardware (e.g., printer, GPU).

Examples: NVIDIA GPU drivers, printer drivers.

Role: Enable hardware functionality (e.g., Wi-Fi, sound cards).

Utility Software

Utility software keeps your computer running smoothly. Antivirus software, like McAfee, protects against viruses by detecting and removing threats, while disk management tools, like MiniTool Partition Wizard, manage disk partitions and optimize storage, introduced in Windows XP for better drive management.

1. Antivirus: Scans for malware using signatures/heuristics. Eg. Norton, McAfee, Windows Defender. Features: Real-time scanning, quarantine, updates.

2. Disk Management: Tools: Disk Cleanup, Disk Defragmentation (Windows Disk Defragmenter), Partition Management- partitioning (GParted), formatting. This is for the purpose of optimizing storage, repair errors (CHKDSK).

Application Software

Productivity tools help you work efficiently, like Microsoft Office’s Word for writing documents, Excel for data analysis, and PowerPoint for presentations, or Google Workspace’s Docs, Sheets, and Slides. These tools simplify office tasks, boosting productivity, and are widely used in professional settings.

a. Productivity Tools

Office Suites: MS Office (Word, Excel, PowerPoint), Google Workspace.

Collaboration: Slack, Trello, Zoom.

Functions: Document creation, data analysis, presentations.

b. Other Application Types

Media: VLC (media player), Adobe Photoshop (graphics).

Educational: MATLAB, Coursera.

Enterprise: SAP (ERP), Salesforce (CRM).

Application Software

S/N	Tool	Examples	Functions
1	Microsoft Office	Word, Excel, PowerPoint	Writing, data analysis, presentations
2	Google Workspace	Docs, Sheets, Slides, Forms	Writing, data management, presentations, surveys
3	Apple iWork	Pages, Numbers, Keynote	Documents, data, slideshows
4	Apache OpenOffice	Writer, Calc, Impress, Draw, Base	Documents, data, presentations, graphics, databases

Other Software Types

1. Middleware: Bridges OS and applications (e.g., database middleware, web servers like Apache).

2. Firmware: Low-level software embedded in hardware. Firmware is specialized software embedded into hardware devices, providing low-level control for the device's specific functions. Unlike regular software, firmware is typically stored in non-volatile memory and remains integral to the hardware's operation, often without user interaction. (e.g., BIOS, router firmware).

3. Embedded Software: Runs on devices (e.g., microwaves, cars).

4. Programming Software: Tools: Compilers (GCC), IDEs (VS Code), debuggers.

Other Software Types

5. Database Software: Manages data (e.g., MySQL, Oracle).

6. Entertainment: Provides leisure activities, enhancing user experience Games (Fortnite), streaming apps (Netflix).

7. Educational Software: Supports learning and training, aiding educational processes. Examples: Khan Academy apps for courses, Duolingo for language learning. Functions: deliver educational content, track progress, enhance learning, suitable for students and educators.

A thorough understanding of the various types of **software and their functions is essential for computer engineers.**

System software lays the groundwork for **hardware operation and resource management,** while application software empowers users to **perform diverse tasks**.

Utility software ensures **system maintenance and security,** and

other specialized software types like middleware, programming tools, and firmware play pivotal roles in the broader computing ecosystem.

As technology evolves, staying informed about these software categories will enable engineers to design and implement efficient, secure, and user-friendly systems.

INTRODUCTION TO PROGRAMMING

Introduction to Programming

Programming is the process of designing and writing instructions that a computer can execute. These instructions, written in a programming language, allow computers to perform specific tasks.

The programming process generally follows these steps:

Problem Definition – Understanding the problem that needs to be solved.

Algorithm Design – Creating a step-by-step procedure to solve the problem.

Flowchart Representation – Visualizing the logic of the solution.

Coding – Writing the actual program using a programming language.

Testing and Debugging – Running the program to check for errors.

Execution and Maintenance – Ensuring that the program continues to work correctly.

What is Algorithms?

An **algorithm** is a step-by-step set of instructions designed to perform a specific task or solve a problem.

It should have the following characteristics:

❖**Definiteness**: Each step is clear and unambiguous.

❖**Finiteness**: It must have a finite number of steps.

❖**Effectiveness**: It should be simple and efficient.

❖**Input and Output**: It must take input(s) and produce an output.

Example of an Algorithm

Algorithm to Find the Sum of Two Numbers

```
1. Start
2. Read two numbers: num1 and num2
3. Compute sum = num1 + num2
4. Display the result
5. Stop
```

For the example, inputs (num1, num2), process (addition), and output (sum).

Flowcharts

A **flowchart** is a **graphical representation** of an algorithm using **standardized symbols** to depict the **flow of processes**. It provides a visual structure of the logical steps involved in a program.

Common Flowchart Symbols:

Oval – Start/End

Parallelogram – Input/Output

Rectangle – Process/Computation

Diamond – Decision (Yes/No or True/False conditions)

Arrows – Flow of execution

Flowcharts

Symbol	Name	Description
● **Oval (Terminator)**	Start/End	Represents the start or end of a flowchart. It is used to indicate the beginning or termination of a process.
▱ **Parallelogram**	Input/Output (I/O)	Represents an input operation (e.g., entering a value) or an output operation (e.g., displaying results).
■ **Rectangle**	Process (Action/Computation)	Represents a process, operation, or instruction (e.g., calculations, assignments).
◇ **Diamond**	Decision (Conditional Check)	Represents a decision point that results in two or more possible outcomes (e.g., "Yes" or "No" conditions in an if-else statement).
→ **Arrow (Flowline)**	Connector (Flow of Execution)	Indicates the direction of flow from one step to another in the process.

Overview on Programming Languages

Programming languages are classified into two broad categories:

Low-Level Languages – Closer to machine instructions and hardware-specific.

High-Level Languages – More human-readable and easier to use.

Low-Level Languages

These are languages that provide minimal abstraction from the hardware.

Machine Language – Written in binary (0s and 1s).

Assembly Language – Uses mnemonics and requires an assembler.

Example of Assembly Language Code

```
MOV AX, 5  ; Load the value 5 into register AX
MOV BX, 10 ; Load the value 10 into register BX
ADD AX, BX ; Add AX and BX, store result in AX
```

Assembly is specific to a particular CPU architecture.

It is fast but difficult to write and debug.

High-Level Languages

High-level languages provide abstraction from the hardware and use English-like syntax.

Examples: Python, Java, C++, JavaScript.

They require compilers or interpreters.

Example of Python Code:

```
num1 = 5

num2 = 10

sum = num1 + num2

print("Sum", sum)
```

Easier to write and understand.

More portable across different systems.

Programming Languages (High-Level vs Low-Level)

S/N	Features	High-Level (Python)	Low-Level (Assembly)
1	Abstraction	Close to human language	Close to machine code
2	Readability	Easy to understand	Hard to understand
3	Portability	Runs on multiple systems	Machine-dependent
4	Speed	Slower (needs interpretation)	Faster (direct execution)
5	Examples	Python, Java, C++	x86 Assembly, ARM Assembly

Code to find the largest of three numbers

```
A = int(input("Enter A: "))

B = int(input("Enter B: "))

C = int(input("Enter C: "))

if A > B and A > C:

   print("A is largest")

elif B > C:

   print("B is largest")

else:

   print("C is largest")
```

Assembly language is more complex and hardware specific

```
section .data
    prompt db 'Enter a number: ', 0
    largest db 'The largest number is: ', 0
section .bss
    num1 resb 2
    num2 resb 2
    num3 resb 2
section .text
    global _start
_start:
    ; Read inputs (simplified for brevity)
    ; Compare and find largest (requires multiple steps)
    ; Output result
    ; Exit
```

Programming Concepts

Regardless of the language, programming follows certain core concepts:

1. Variables and Data Types

A **variable** is a storage location with a name.

Data types define what kind of data a variable can store.

Example in Python:

```
age = 25        # Integer
name = "Alice"   # String
pi = 3.14       # Float
is_student = True # Boolean
```

Programming Concepts

2. Control Structures

Control structures determine the flow of execution in a program.

Conditional Statements (if-else)

Used for decision-making.

```
x = 10
if x > 5:
    print("x is greater than 5")
else:
    print("x is less than or equal to 5")
```

Programming Concepts

Loops (for, while)

Loops help in executing a block of code multiple times.

For Loop:

```
for i in range(1, 6):
    print(i)
```

While Loop:

```
count = 1
while count <= 5:
    print(count)
    count += 1
```

Programming Concepts

3. Functions

Functions are reusable blocks of code that perform specific tasks.

Example of a Function in Python:

```
def add_numbers(a, b):
    return a + b
 result = add_numbers(5, 10)
print("Sum", result)
```

4. Arrays and Lists

Arrays and lists store multiple values in a single variable.

List Example in Python:

```
fruits = ["apple", "banana", "cherry"]
print(fruits[0])  # Output: apple
```

Programming Concepts

5. Object-Oriented Programming (OOP)

OOP is a programming paradigm based on objects.

Example of a Class and Object in Python:

```
class Car:
    def _init_(self, brand, model):
        self.brand = brand
        self.model = model
        def display_info(self):
        print("Car", self.brand, self.model)

my_car = Car("Toyota", "Corolla")
my_car.display_info()
```

Home Task

1. Write an algorithm to check if a number is even or odd.
2. Draw a flowchart for a program that finds the largest of three numbers.
3. Explain the difference between an interpreter and a compiler.