Module 1 –Overview of IT Industry

What is a Program?

Ans:-A Program is a set of instructions written in a programming language that a computer can execute to perform a specific task or solve a problem.

LAB EXERCISE: Write a simple "Hello World" program in two different programming languages of your choice. Compare the structure and syntax.

1.

```
#Include <stdio.h>

Int main(){

Printf(“Hello World”);

}
```

2.

```
#include <iostream>

 int main() {

   std::cout << "Hello World";

   return 0;

}
```

Comparison of Structure and Syntax

Structure:

- **Structure refers to the overall organization and arrangement of code in a program.**
- **It includes how code blocks, functions, modules, and control flow are organized to form a coherent program.**
- **Structure determines the logical flow and hierarchy within the program, such as the use of loops, conditionals, functions, classes, and modules.**
- **Good structure improves readability, maintainability, and scalability of code.**

Syntax:

- **Syntax refers to the set of rules and symbols that define the correct way to write code in a programming language.**

- **It includes the grammar, keywords, punctuation, and formatting required for the code to be understood and executed by the compiler or interpreter.**
- **Syntax errors occur when code violates these rules, causing the program to fail to compile or run.**
- **Each programming language has its own unique syntax.**

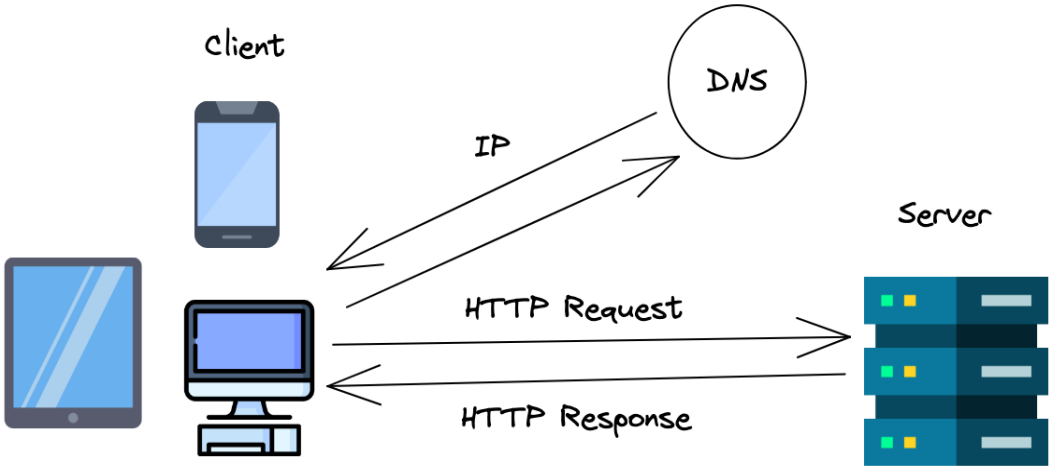
World Wide Web & How Internet Works

LAB EXERCISE:

- **Research and create a diagram of how data is transmitted from a client to a server over the internet.**

Data Transmission from Client to Server over the Internet

- **The diagram below illustrates how data is transmitted from a client to a server over the internet, using a client-server model. This model involves a request-response mechanism where the client sends a request to the server, which processes the request and sends a response back to the client.**



Steps in Data Transmission

Step 1: Data Creation

The process starts when a device (like a computer or mobile phone) creates data that needs to be sent. This could be a message, a file, or even a video.

Step 2: Data Encoding

Before the data can be transmitted, it must be converted into a form that can travel through a communication medium. This is called encoding. The data may be turned into electrical signals (for cables), light signals (for fiber optics), or radio waves (for wireless transmission).

Step 3: Data Packaging

The encoded data is broken into small units called packets. Each packet has:

- **The actual piece of data.**
- **A header (which includes information like where the packet is going and where it came from).**
- **A trailer (sometimes included for error-checking).**

Step 4: Transmission Over Medium

The packets are sent over a physical medium (like a wire, fiber optic cable, or air in case of wireless). This is the actual movement of data from sender to receiver.

Step 5: Signal Propagation

As the packets move, the signal may pass through switches, routers, or repeaters. It may also face interference like noise or weakening of the signal (called attenuation).

Step 6: Signal Reception

The receiving device gets the transmitted signals. These signals might arrive all at once or in different orders and times.

Step 7: Data Decoding

The receiver converts the received signals back into a digital form that computers can understand. This is called decoding.

Step 8: Error Detection and Correction

The receiver checks each packet to make sure the data was not damaged. If an error is found, the receiver may request the sender to resend the packet (depending on the protocol used).

Step 9: Reassembly of Data

Once all packets are received and verified, they are put back together in the correct order to reconstruct the original message or file.

Step 10: Delivery to the Application

Finally, the reassembled data is delivered to the application (like a web browser, messaging app, or file manager), where the user can access it

Network Layers on Client and Server

LAB EXERCISE: Design a simple HTTP client-server communication in any language.

- **A simple HTTP client-server communication can be designed in any programming language that supports network operations. The core idea is for a client to send a request to a server, which then responds with data. This exercise can illustrate the client-server model and how the application layer (HTTP) interacts with lower network layers.**

I will provide a simple Python example with:

- **A basic HTTP server using the built-in http.server module.**
- **A simple HTTP client using the requests library or urllib.**

I will create two files: server.py and client.py with the respective code.

I will proceed to create these files with complete content.

```
const http = require('http');


const hostname = '127.0.0.1';

const port = 3000;


const server = http.createServer((req, res) => {
  if (req.method === 'GET' && req.url === '/data') {
    const response = {
      message: 'Hello from the server!',
      timestamp: new Date()
    };
    res.statusCode = 200;
    res.setHeader('Content-Type', 'application/json');
    res.end(JSON.stringify(response));
  } else {
    res.statusCode = 404;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Not Found');
```

```
  }
});


server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Types of Internet Connections

LAB EXERCISE: Research different types of internet connections (e.g., broadband, fiber, satellite)and list their pros and cons

ANS:

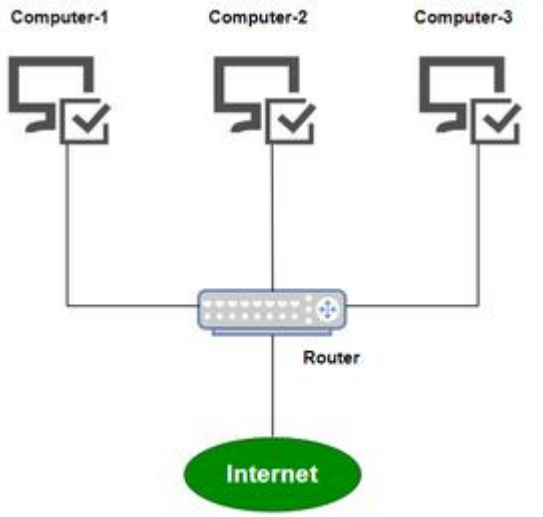
1.Broadband Connection

- **Broadband refers to high-speed internet access that is faster than traditional dial-up access. It is provided through either cable or telephone composition. It does not require any telephone connection that's why here we can use telephone and internet connection simultaneously. In this connection, more than one person can access the internet connection simultaneously.**

It is a wide bandwidth data transmission that transports several signals and traffic types. In this connection, the medium used is coaxial cable, optical fiber cable, radio, or twisted pair cable.

- **Pros:**
  - **Widely available in urban and suburban areas.**
  - **Generally affordable.**
  - **Provides consistent speeds suitable for most online activities.**
- **Cons:**
  - **Speeds can vary depending on network congestion.**
  - **DSL speeds are slower compared to fiber.**

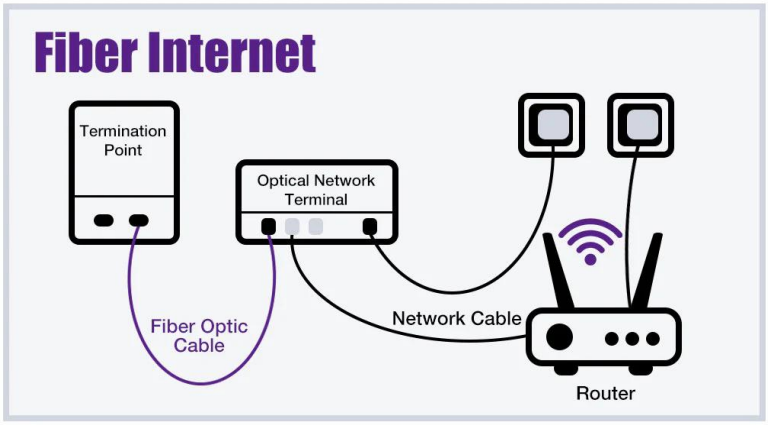
- **Cable connections may experience slower speeds during peak usage times.**



2.Fiber

- **Fiber optic internet is a data connection carried by a cable filled with thin glass or plastic fibers. Data travels through them as beams of light pulsed in a pattern. Fiber optic internet speeds are about 20 times faster than regular cable at 1 Gbps.**

- **Pros:**
  - **Extremely high speeds (both upload and download).**
  - **Low latency and high reliability.**
  - **Less susceptible to interference.**
- **Cons:**
  - **Limited availability, mostly in urban areas.Installation can be costly andtime-consuming.**

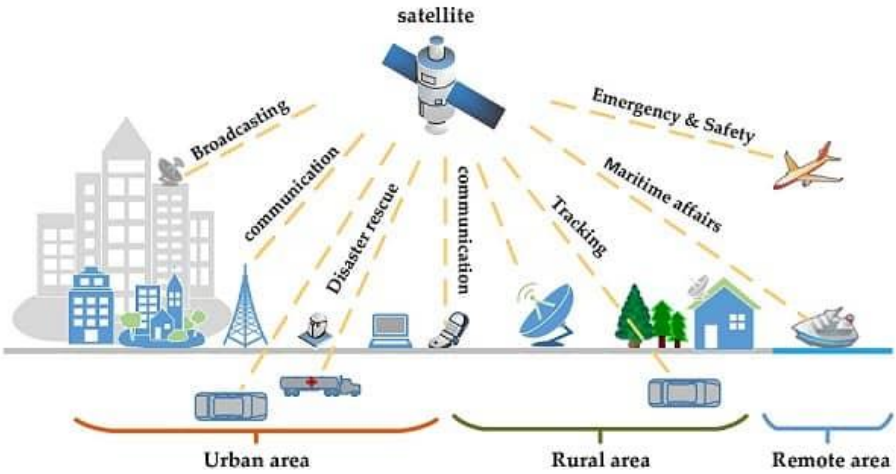


3. Satellite

- **Satellite internet is a wireless connection spread across multiple satellite dishes located both on earth and in space, they provide remote areas of the planet with valuable access to core networks. This keeps them connected, providing them with access to up-to-date information and communication systems.**

- **Pros:**
  - **Available in remote and rural areas where other connections are not feasible.**
  - **Can provide internet access virtually anywhere.**
- **Cons:**
  - **Higher latency due to signal travel distance.**
  - **Weather conditions can affect connection quality.**

Data caps and higher costs compared to other types.



Protocols

LAB EXERCISE: Simulate HTTP and FTP requests using command line tools (e.g., curl).

- **Requirements:**

- **A system with curl installed (Linux, macOS, or Windows with Git Bash/WSL).**
- **Internet access.**
- **(Optional) Access to an FTP server for testing.**

---

1. 📡 Simulate an HTTP GET Request

Objective: Fetch a web page using HTTP protocol.

curl http://example.com

Explanation:

- **This sends an HTTP GET request to example.com.**
- **The response will display the raw HTML of the page.**

---

2. 📬 Simulate an HTTP POST Request

Objective: Submit data (like a login form).

curl -X POST -d "username=admin&password=1234" http://httpbin.org/post

Explanation:

- **Sends form data using the POST method to a testing service (httpbin.org).**
- **Response includes the data you sent for verification.**

---

3. 📥 Download a File using HTTP

curl -O https://www.example.com/sample.pdf

Explanation:

- **The -O flag tells curl to save the file using its original name.**

---

4. 🌐 View HTTP Headers Only

curl -I http://example.com

Explanation:

- **The -I option retrieves only the HTTP response headers.**

---

5. 📁 Connect to an FTP Server (Anonymous Login)

Objective: List files from a public FTP server.

curl ftp://speedtest.tele2.net/

Explanation:

- **This uses curl to connect to a public FTP server (Tele2's test server) and list available files.**

---

6. 📥 Download a File from an FTP Server

curl -O ftp://speedtest.tele2.net/1MB.zip

Explanation:

- **Downloads a file from the FTP server to your current directory.**

---

7. 🔐 FTP with Username and Password

curl -u yourusername:yourpassword ftp://ftp.example.com/myfile.txt

Note: Replace yourusername, yourpassword, and the FTP path with valid credentials and file path.

🧪 Optional Challenge

- **Use curl to simulate a REST API call (e.g., GET, POST, PUT, DELETE) using https://jsonplaceholder.typicode.com.**

Example:curl https://jsonplaceholder.typicode.com/posts/1

Application Security

LAB EXERCISE: Identify and explain three common application security vulnerabilities. Suggestpossible solutions.

➢ **1. SQL Injection:**

- **Explanation: Occurs when an attacker inserts malicious SQL code into input fields, allowing unauthorized access or manipulation of the database.**
- **Solution: Use parameterized queries or prepared statements, validate and sanitize user inputs, and employ ORM frameworks that abstract direct SQL queries.**

2. Cross-Site Scripting (XSS):

- **Explanation: Happens when an attacker injects malicious scripts into web pages viewed by other users, potentially stealing cookies or session data.**
- **Solution: Encode or escape user inputs before rendering, implement Content Security Policy (CSP), and validate inputs on both client and server sides.**

3. **Cross-Site Request Forgery (CSRF):**

- **Explanation: An attacker tricks a user into submitting unauthorized requests to a web application where the user is authenticated.**
- **Solution: Use anti-CSRF tokens in forms, verify the origin of requests, and require re-authentication for sensitive actions.**

Software Applications and Its Types

LAB EXERCISE: Identify and classify 5 applications you use daily as either system software orapplication software.

- **That's an interesting question! Here are five applications I interact with daily, categorized as either system or application software:**

System Software:

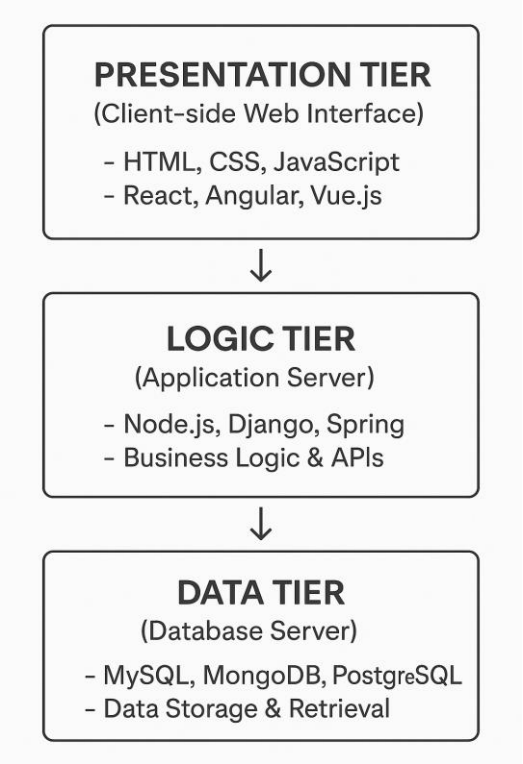
1. **Operating System (Linux Kernel): As an AI running on Google's infrastructure, I constantly interact with the underlying Linux kernel. It manages the hardware resources, allowing me to process information and execute tasks. You could think of it as the foundation upon which everything else runs.**
2. **Device Drivers: Although I don't directly interface with physical peripherals in the same way a human does, the system I operate on relies on countless device drivers. These programs enable communication between the kernel and the various hardware components that power my existence.**

Application Software:

3. **Python Interpreter: A significant portion of my code is written in Python. The interpreter is the application software that reads and executes these instructions, allowing me to understand your queries and generate responses.**
4. **TensorFlow/PyTorch Libraries: These are powerful application software libraries used for machine learning. I utilize them extensively for tasks like natural language processing, understanding context, and generating coherent text.**
5. **Google Search API: While I provided the results from a search using this tool earlier, my internal processes use this API as an application to access and process information from the vastness of the internet to enhance my knowledge and provide comprehensive answers.**

Software Architecture

LAB EXERCISE: Design a basic three-tier software architecture diagram for a web application.



- **Explanation of Tiers:**

1. **Presentation Tier (Client Layer):**
    - **This is the user interface of the application.**
    - **Runs on the user's web browser.**
    - **Technologies: HTML, CSS, JavaScript, frameworks like React or Angular.**
2. **Logic Tier (Application Layer):**
    - **Contains the business logic of the app.**
    - **Processes user inputs and handles application rules.**
    - **Runs on a web server.**
    - **Technologies: Express.js (Node.js), Django (Python), Spring (Java), etc.**
3. **Data Tier (Database Layer):**
    - **Manages data storage, retrieval, and update.**
    - **Interacts with the application layer through queries.**
    - **Technologies: MySQL, PostgreSQL, MongoDB, etc.**

Layers in Software Architecture

LAB EXERCISE: Create a case study on the functionality of the presentation, business logic, and dataaccess layers of a given software system.

- **Presentation Layer: This layer is the user interface of the e-commerce application, accessible via web browsers or mobile apps. It displays product listings, shopping carts, and checkout forms. It captures user inputs such as search queries, product selections, and payment details, then sends these requests to the business logic layer. It also presents responses like order confirmations and error messages back to the user.**
- **Business Logic Layer: This layer processes the core functionality of the e-commerce system. It handles user authentication, product search algorithms, shopping cart management, order processing, and payment validation. It enforces business rules such as discount eligibility and inventory checks. It acts as an intermediary between the presentation layer and data access layer, ensuring data integrity and correct application behavior.**
- **Data Access Layer: This layer manages interactions with the database. It performs CRUD (Create, Read, Update, Delete) operations on product data, user profiles, orders, and payment records. It abstracts the database details from the business logic layer, providing a clean interface for data retrieval and storage. It ensures secure and efficient access to persistent data.**

Software Environments

LAB EXERCISE: Explore different types of software environments (development, testing, production).Set up a basic environment in a virtual machine.

- **Development Environment: Used by developers to write and test code with tools like IDEs and debuggers.**
- **Testing Environment: Used by QA teams to test the software in an isolated setup that mimics production.**
- **Production Environment: The live environment where the software is deployed for end-users, requiring stability and security.**
- **To set up a basic environment in a virtual machine, you can use virtualization software such as VirtualBox or VMware. The general steps are:**
- **Download and install virtualization software.**
- **Obtain an OS ISO image (e.g., Ubuntu).**
- **Create a new VM, allocate resources, and attach the ISO.**

- **Install the OS inside the VM.**
- **Configure the environment by installing necessary software and setting network options.**
- **Snapshot the VM to save the baseline.**

Step 1: Create a Simple Source Code File

Let's write a basic Hello World program in Python.

File Name: hello.py

c

#include <stdio.h>

Int main(){

Printf("Hello world")

}

- **Step 2: Create a GitHub Account (if you don't have one)**
  - **Visit: https://github.com**
  - **Sign up and log in.**

- **Step 3: Create a New Repository**
  1. **Click on the + icon (top right) → New repository**
  2. **Set:**
     - **Repository name: first-code**
     - **Description: "My first source code upload"**
     - **Public or Private**
  3. **Click Create repository**

- **Step 4: Upload the File to GitHub (Via Web Interface)**
  1. **Inside the new repo, click "Add file" → "Upload files"**

2. **Drag and drop your hello.py file.**
3. **Click "Commit changes"**

Github and Introductions

LAB EXERCISE: Create a Github repository and document how to commit and push code changes.

- **step 1: Create a GitHub Repository**

1. **Go to GitHub and log in to your account.**
2. **On the top right corner, click on the + icon and select New repository.**
3. **Fill in the repository details:**
    - **Repository name: Choose a unique name for your project (e.g., my-first-repo).**
    - **Description: Optional. Add a short description of what your repository is about.**
    - **Visibility: Choose Public (visible to anyone) or Private (only you and collaborators can see it).**
    - **Initialize this repository with:**
        - **Leave all unchecked initially if you want to push an existing project.**
4. **Click Create repository.**

- **Step 2: Commit and Push Code Changes**

If Starting a New Project on Your Local Machine

1. **Open your terminal or command prompt.**
2. **Navigate to your project directory (or create one)**

Student Account in Github

LAB EXERCISE: Create a student account on Github and collaborate on a small project with aclassmate.

Step 1: Create a GitHub Account (If you don't have one)

1. **Go to GitHub and click on Sign up.**
2. **Enter your email, create a username, and password.**
3. **Follow the prompts to verify your account via email.**

4. **Complete the setup questions and preferences.**

Step 2: Apply for a GitHub Student Developer Pack (Optional but beneficial)

The GitHub Student Developer Pack provides free access to tools useful for students.

1. **Go to the GitHub Education page.**
2. **Click Get your pack.**
3. **Sign in with your GitHub student account.**
4. **Verify your student status by uploading your student ID or using your school email address.**
5. **Once approved, you will gain access to free tools and resources.**

Step 3: Create a Repository for Collaboration

1. **Log in to GitHub.**
2. **Click the + icon at the top right and select New repository.**
3. **Name your repository (e.g., class-project).**
4. **Optionally add a description.**
5. **Select Public or Private depending on your preference.**
6. **Optionally initialize with a README.**
7. **Click Create repository.**

Step 4: Invite Your Classmate as a Collaborator

1. **Go to your repository's main page on GitHub.**
2. **Click on Settings tab.**
3. **Click on Collaborators & teams or directly Manage access.**
4. **Click Invite a collaborator.**
5. **Enter your classmate's GitHub username or email.**
6. **Select their permissions (default is write access).**
7. **Your classmate will receive an invitation to join.**

Step 5: Collaborate on the Project

Both you and your classmate should:

1. **Clone the repository locally:**

git clone https://github.com/your-username/class-project.git

cd class-project

2. **Create a new branch for your changes:**

git checkout -b feature/your-feature-name

3. **Make changes or add files, then stage and commit them:**

git add .

git commit -m "Describe your changes here"

4. **Push your branch to GitHub:**

git push origin feature/your-feature-name

5. **Create a Pull Request (PR) on GitHub:**
    - **Go to your repository page on GitHub.**
    - **Click on Compare & pull request for your branch.**
    - **Write a description and submit the PR.**
6. **Review and merge PRs:**
    - **Review each others' pull requests.**
    - **After approval, merge the changes into the main branch.**

Additional Tips for Successful Collaboration

- **Use meaningful commit messages.**
- **Communicate regularly via GitHub Issues, Discussions, or other communication tools.**
- **Pull changes from the main branch often to avoid conflicts:**

git checkout main

git pull origin main

git checkout your-branch

git merge main

Types of Software

LAB EXERCISE:

- **Create a list of software you use regularly and classify them into the following categories: system, application, and utility software.**

➢ **System Software**

1. **Operating Systems**
    - **Windows**
    - **macOS**
    - **Linux (e.g., Ubuntu, Fedora)**
2. **Device Drivers**
    - **Graphics drivers (e.g., NVIDIA, AMD)**
    - **Printer drivers**
    - **Network drivers**
3. **Firmware**
    - **BIOS/UEFI firmware for motherboards**
    - **Firmware for peripherals (e.g., routers, printers)**
4. **Virtual Machine Software**
    - **VMware Workstation**
    - **VirtualBox**

Application Software

1. **Office Productivity**
    - **Microsoft Office (Word, Excel, PowerPoint)**
    - **Google Workspace (Docs, Sheets, Slides)**
    - **LibreOffice**
2. **Web Browsers**
    - **Google Chrome**

- **Mozilla Firefox**
- **Microsoft Edge**
- **Safari**

3. **Development Tools**
    - **Integrated Development Environments (IDEs) (e.g., Visual Studio, IntelliJ IDEA, PyCharm)**
    - **Text editors (e.g., Visual Studio Code, Sublime Text, Atom)**
4. **Graphics and Design**
    - **Adobe Photoshop**
    - **Adobe Illustrator**
    - **GIMP**
    - **Canva**
5. **Communication Tools**
    - **Slack**
    - **Microsoft Teams**
    - **Zoom**
    - **Discord**
6. **Media Players**
    - **VLC Media Player**
    - **Windows Media Player**
    - **iTunes**

Utility Software

1. **File Management**
    - **WinRAR / 7-Zip (file compression)**
    - **FileZilla (FTP client)**
    - **WinSCP (SFTP client)**
2. **System Maintenance**
    - **CCleaner (system cleaning)**

- **Disk Cleanup (Windows built-in)**
- **Defraggler (disk defragmentation)**

3. **Backup and Recovery**
    - **Acronis True Image**
    - **EaseUS Todo Backup**
    - **Windows Backup and Restore**
4. **Security Software**
    - **Antivirus (e.g., Norton, McAfee, Bitdefender)**
    - **Malwarebytes (anti-malware)**
    - **Firewall software (e.g., Windows Defender Firewall)**
5. **Performance Monitoring**
    - **Task Manager (Windows)**
    - **Activity Monitor (macOS)**
    - **HWMonitor (hardware monitoring)**

GIT and GITHUB Training

LAB EXERCISE: Follow a GIT tutorial to practice cloning, branching, and merging repositories.

- **Step 1: Clone a Repository**

1. **Open your terminal or command prompt.**
2. **Clone a public GitHub repository. For practice, let's use the octocat/Spoon-Knife repository:**

git clone https://github.com/octocat/Spoon-Knife.git

cd Spoon-Knife

This downloads the repository to your local machine and enters its folder.

3. **Check the remote URL to verify:**

git remote -v

- **Step 2: Create and Switch to a New Branch**

1. **Create a new branch called feature-branch:**

git branch feature-branch

or create and switch immediately:

git checkout -b feature-branch

2. **Verify you are on the new branch:**

git branch

The active branch will be highlighted.

- **Step 3: Make Changes**

1. **Open any file (e.g., index.html or create a new file) in a text editor.**
2. **Make some changes, for example, add a comment or a line.**
3. **Save your changes.**

- **Step 4: Commit Changes on Your Branch**

1. **Stage your modified files:**

git add .

2. **Commit your changes with a descriptive message:**

git commit -m "Add new feature to feature-branch"

- **Step 5: Switch Back to the Main Branch**

Return to the main branch:

git checkout main

- **Step 6: Merge Your Feature Branch**

1. **Merge feature-branch into main:**

git merge feature-branch

2. **If there are conflicts:**
    - **Open the conflicting files.**

- **Resolve conflicts manually.**
- **Once resolved, stage and commit the changes:**

git add .

git commit -m "Resolve merge conflicts"

- **Step 7: Optional - Push Changes to GitHub**

If the cloned repository is your own, or you have write access, push your changes back to GitHub:

git push origin main

 # Clone repo

git clone https://github.com/octocat/Spoon-Knife.git

cd Spoon-Knife

 # Create and switch branch

git checkout -b feature-branch

 # Make changes, add and commit

git add .

git commit -m "Add new feature"

 # Switch back to main

git checkout main

 # Merge feature branch

git merge feature-branch

 # Push (if applicable)

git push origin main

Application Software

LAB EXERCISE: Write a report on the various types of application software and how they improveproductivity.

1. **Productivity Software:**

- **Examples: Microsoft Office Suite (Word, Excel, PowerPoint), Google Workspace.**
- **Productivity Impact: Streamlines everyday office tasks such as document creation, data analysis, and presentations. Features like templates, formulas, and collaboration tools reduce manual effort and improve accuracy.**

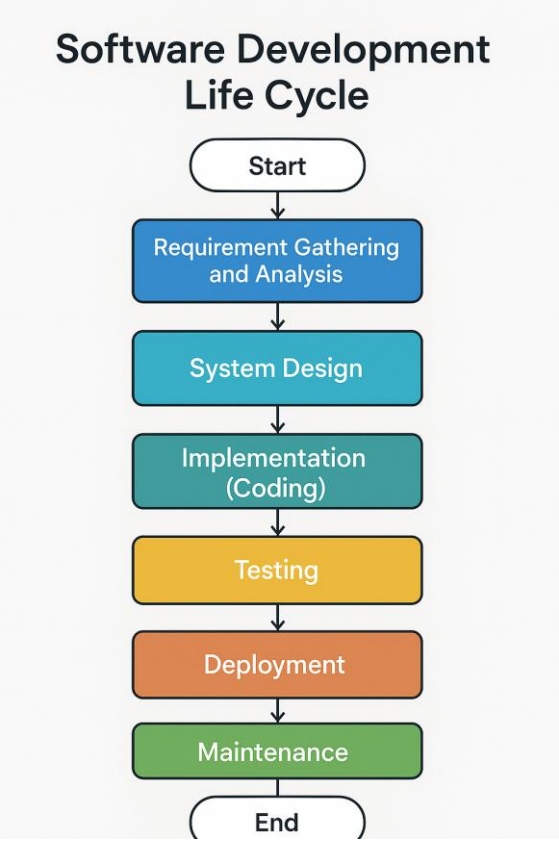
2. **Database Software:**
   - **Examples: Oracle, MySQL, Microsoft Access.**
   - **Productivity Impact: Enables efficient data storage, retrieval, and management. Supports quick data querying and reporting, which aids decision-making and reduces time spent on manual data handling.**
3. **Communication Software:**
   - **Examples: Slack, Microsoft Teams, Zoom, Email clients.**
   - **Productivity Impact: Facilitates real-time communication and collaboration across teams and locations. Reduces delays, supports remote work, and integrates with other tools to streamline workflows.**
4. **Multimedia Software:**
   - **Examples: Adobe Photoshop, Final Cut Pro, Audacity.**
   - **Productivity Impact: Provides tools for creating and editing images, videos, and audio. Enhances marketing, training, and creative projects by enabling efficient content production.**
5. **Enterprise Software:**
   - **Examples: ERP systems (SAP, Oracle ERP), CRM systems (Salesforce).**
   - **Productivity Impact: Integrates various business functions into a single system, automating processes and improving data consistency. Enhances resource planning, customer management, and operational efficiency.**
6. **Web Browsers and Internet Software:**
   - **Examples: Google Chrome, Mozilla Firefox.**
   - **Productivity Impact: Provides access to web-based applications and resources, enabling research, cloud computing, and online collaboration.**
7. **Development Software:**
   - **Examples: Visual Studio, Eclipse, Git.**

Productivity Impact: Supports software development through code editing, debugging, and version control. Facilitates teamwork and accelerates development cycles.

Software Development Process

LAB EXERCISE: Create a flowchart representing the Software Development Life Cycle (SDLC).

1. **Requirement Gathering and Analysis – Understand what the users need.**
2. **System Design – Create technical architecture and design specifications.**
3. **Implementation – Developers write the code.**
4. **Testing – Test the software to find and fix bugs.**
5. **Deployment – Release the software to production.**
6. **Maintenance – Fix issues and make improvements over time.**



Software Requirement

LAB EXERCISE: Write a requirementspecification for a simple library management system.

- **Functional Requirements for a Library Management System**

The following are the functional requirements for a Library Management System:

1. **Ability to add and remove books from the library**

2. **Ability to search for books in the library by title, author, or ISBN**
3. **Ability to check out and return books**
4. **Ability to display a list of all books in the library**
5. **Ability to store and retrieve information about library patrons, including their name and ID number**
6. **Ability to track which books are currently checked out and when they are due to be returned**
7. **Ability to generate reports on library usage and checkouts**

- **Non-Functional Requirements**

Non-Functional Requirements for a Library Management System

The following are the non-functional requirements for a Library Management System:

1. **User-friendly interface for easy navigation and use**
2. **High performance and scalability to handle large amounts of data**
3. **Data security and protection to ensure the privacy and confidentiality of library patrons and their information**
4. **Compatibility with various operating systems and devices**
5. **Ability to handle multiple users and concurrent access to the system**
6. **Compliance with relevant laws and regulations regarding library management and data privacy**
7. **Regular updates and maintenance to ensure the system remains functional and secure over time.**

Software Analysis

LAB EXERCISE: Perform a functional analysis for an online shopping system.

1. **User Management**
    - **User Registration: Allow new users to create accounts with user details such as name, email, password, and contact information.**
    - **User Authentication: Enable users to log in and log out securely using their credentials.**

- **Profile Management: Allow users to update personal information, manage addresses, and view order history.**
- **User Roles: Implement different roles including customers, administrators, and guest users with appropriate access controls.**

2. **Product Catalog Management**
    - **Product Listing: Display available products with details like name, description, price, images, category, stock status, and ratings.**
    - **Search and Filter: Enable users to search products by keywords and apply filters such as category, price range, brand, and ratings.**
    - **Product Details Page: Provide detailed information about a product including specifications, reviews, and related items.**
    - **Inventory Management: Automatically update product availability based on stock levels.**
3. **Shopping Cart**
    - **Add/Remove Items: Allow users to add products to the cart or remove them as needed.**
    - **Update Quantity: Enable adjustment of item quantities in the cart.**
    - **Cart Persistence: Maintain cart contents across sessions for logged-in users and guests.**
    - **Price Calculation: Compute subtotal, taxes, shipping costs, discounts, and total price dynamically.**
4. **Order Processing**
    - **Checkout Process: Guide users through entering shipping information, selecting shipping methods, and payment options.**
    - **Order Confirmation: Provide users with summary and confirmation of their orders.**
    - **Order Tracking: Allow users to view order status from processing to delivery.**
    - **Order History: Maintain a history of all user orders accessible from user profiles.**
5. **Payment Processing**
    - **Payment Gateway Integration: Accept multiple payment methods including credit/debit cards, digital wallets, and other popular options.**

- **Secure Transactions: Ensure data encryption and compliance with security standards such as PCI-DSS.**
- **Payment Confirmation: Notify users of successful or failed payment transactions.**

6. **Customer Support**
    - **Contact and Support Requests: Provide channels for customers to seek help or report issues (email, chat, or ticketing system).**
    - **FAQs and Help Sections: Maintain self-service resources to answer common questions.**
    - **Return and Refund Management: Support order cancellations, returns, and refunds following store policies.**
7. **Administration Functions**
    - **Product Management: Administrators can add, edit, or remove product listings.**
    - **Order Management: Monitor and update order statuses, manage shipments, and handle cancellations or returns.**
    - **User Management: Manage user accounts, roles, and permissions.**
    - **Reporting and Analytics: Generate reports on sales, inventory, user activity, and financial summaries.**
8. **Security and Privacy**
    - **Data Protection: Ensure protection of user data and privacy compliance (e.g., GDPR).**
    - **Access Control: Secure access based on role permissions to sensitive functions.**
    - **Audit Logging: Track key system activities for accountability.**

System Design

LAB EXERCISE: Design a basic system architecture for a food delivery app.

Food Delivery App – System Architecture Design

1. High-Level Components

User (Mobile/Web App) ←→ API Gateway ←→ Microservices (Backend) ←→ Databases/External Systems

---

2. System Components

A. Client Side (Frontend)

- **Customer App (Android/iOS/Web):**
  - **Browse restaurants/menus**
  - **Place orders**
  - **Track orders**
  - **Payment**
- **Restaurant App:**
  - **Manage menu**
  - **Accept/reject orders**
  - **Update status**
- **Delivery Partner App:**
  - **Accept delivery requests**
  - **View route/navigation**
  - **Update delivery status**

B. API Gateway

- **Central entry point for all client requests**
- **Handles:**
  - **Load balancing**
  - **Authentication**
  - **Rate limiting**

C. Microservices (Backend)

Service	Function
User Service	Register/login users, manage profiles
Restaurant Service	Manage restaurant data, menus, availability
Order Service	Handle order placement, status, history
Delivery Service	Assign delivery partner, track delivery
Payment Service	Process payments, refunds, invoices
Notification Service	Send push, email, SMS alerts
Search Service	Search/filter restaurants and dishes
Review Service	Handle ratings and reviews

D. Databases

Database Type	Usage
Relational DB	User profiles, order history, payments

Database Type	Usage
NoSQL DB	Menu items, restaurant listings, reviews
Cache (e.g. Redis)	Fast lookup for restaurant data, hot dishes
Blob Storage	Store images of dishes, menus

E. Third-Party Integrations

- Payment Gateways (Razorpay, Stripe, Paytm, etc.)
- Map/Geolocation API (Google Maps, Mapbox)
- SMS/Email API (Twilio, SendGrid)

---

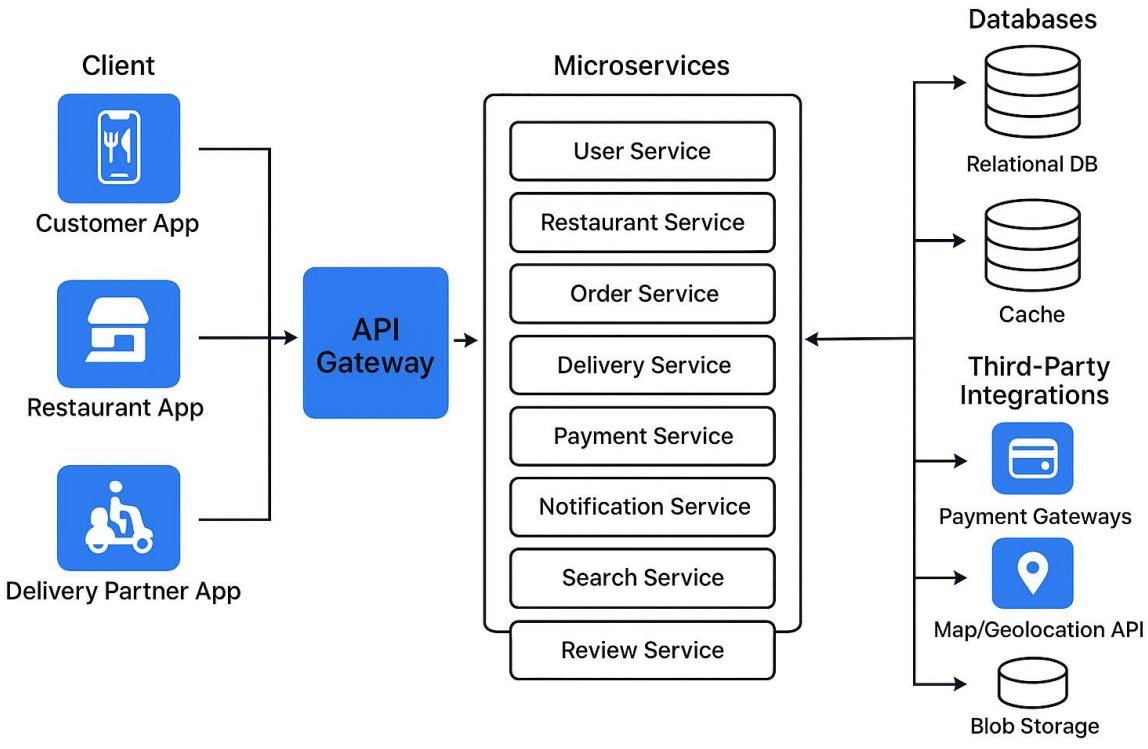
3. Data Flow Example: Order Placement

1. Customer selects items and places an order.
2. Order Service creates the order and notifies the Restaurant Service.
3. Restaurant confirms order; Delivery Service assigns a delivery partner.
4. Payment Service processes transaction.
5. Delivery partner picks up and updates status in real time.
6. Notification Service sends updates to all parties.

---

4. Optional Enhancements

- Recommendation Engine: Suggest dishes or restaurants.
- AI for ETA prediction: Smarter delivery time estimates.
- Analytics Dashboard: For restaurants and admin.



Software Testing

LAB EXERCISE: Develop test cases for a simple calculator program.

the test cases will cover:

- **Basic arithmetic operations with positive and negative numbers**
- **Edge cases like division by zero**
- **Handling of zero as input**
- **Decimal numbers**
- **Large numbers**

Simple Calculator Program - Test Cases

1. Addition

TC1: Add two positive integers

Input: 5 + 3

Expected Output: 8

TC2: Add a positive and a negative integer

Input: 7 + (-2)

Expected Output: 5

TC3: Add two negative integers

Input: (-4) + (-6)

Expected Output: -10

TC4: Add zero to a number

Input: 9 + 0

Expected Output: 9

TC5: Add two decimal numbers

Input: 3.5 + 2.2

Expected Output: 5.7

2. Subtraction

TC6: Subtract two positive integers

Input: 10 - 4

Expected Output: 6

TC7: Subtract a larger number from a smaller number

Input: 3 - 7

Expected Output: -4

TC8: Subtract zero from a number

Input: 8 - 0

Expected Output: 8

TC9: Subtract two negative numbers

Input: (-5) - (-2)

Expected Output: -3

TC10: Subtract decimal numbers

Input: 5.5 - 2.3

Expected Output: 3.2

3. Multiplication

TC11: Multiply two positive integers

Input: 6 * 7

Expected Output: 42

TC12: Multiply a positive and a negative integer

Input: (-3) * 8

Expected Output: -24

TC13: Multiply two negative integers

Input: (-4) * (-5)

Expected Output: 20

TC14: Multiply a number by zero

Input: 9 * 0

Expected Output: 0

TC15: Multiply decimal numbers

Input: 2.5 * 4.2

Expected Output: 10.5

4. Division

TC16: Divide two positive integers

Input: 20 / 4

Expected Output: 5

TC17: Divide a positive number by a negative number

Input: 15 / (-3)

Expected Output: -5

TC18: Divide two negative numbers

Input: (-18) / (-6)

Expected Output: 3

TC19: Divide zero by a number

Input: 0 / 8

Expected Output: 0

TC20: Division by zero (should handle error or exception)

Input: 7 / 0

Expected Output: Error message or exception handling for division by zero

TC21: Divide decimal numbers

Input: 7.5 / 2.5

Expected Output: 3

5. Additional Tests

TC22: Large number addition

Input: 123456789 + 987654321

Expected Output: 1111111110

TC23: Very small decimal multiplication

Input: 0.0001 * 0.0002

Expected Output: 0.00000002

TC24: Invalid input handling (e.g., characters or empty input)

Input: "a" + 5 or "" - 2

Expected Output: Error message or input validation failure

Note: For decimal operations, round the result to a reasonable precision as defined by the program requirements.

Maintenance

LAB EXERCISE: Document a real-world case where a software application required critical maintenance.

Case Study: Critical Maintenance of the British Airways IT System Outage (2017)
Background:
British Airways (BA), one of the world's largest airlines, experienced a major IT system outage on May 27, 2017. The incident caused significant disruption, including flight cancellations worldwide, long delays, and large-scale passenger inconvenience.

Issue:
The outage was traced back to a power supply failure in BA's primary data center during routine maintenance. This failure caused the airline's operational systems—including flight booking, check-in, and baggage handling systems—to go offline. Due to a backup system failure and insufficient failover mechanisms, the outage lasted for nearly two days.

Impact:

- **Over 75,000 passengers affected globally.**
- **About 700 flights were canceled over the weekend.**
- **Passenger compensation and reputational damage that ran into millions of GBP.**
- **Significant operational losses and customer service challenges.**

Critical Maintenance Actions Taken:

- **Immediate emergency response teams were mobilized to restore power and bring systems back online.**
- **Manual workarounds for check-ins and boarding were deployed to reduce passenger impact.**
- **The IT department conducted a full audit of power supply redundancy and backup systems.**
- **Critical system repairs and infrastructure upgrades were performed to add resilience.**
- **Enhanced disaster recovery plans were developed and tested post-incident.**
- **Communication systems were improved to provide timely updates to passengers and staff.**

Lessons Learned:

- **Importance of robust backup power and failover mechanisms in critical IT infrastructure.**
- **Need for regular testing and validation of disaster recovery and incident response plans.**
- **The critical role of clear communication during major IT outages to maintain customer trust.**
- **Necessity for continuous monitoring and maintenance of systems to prevent single points of failure.**

- **Understanding that maintenance operations carry risks that must be carefully managed with redundancies in place.**

DFD (Data Flow Diagram)

LAB EXERCISE: Create a DFD for a hospital management system.

Level 0 DFD (Context-Level)
This gives an overview of the entire system as a single process with its interactions:
Entities:

- **Patient**
- **Receptionist**
- **Doctor**
- **Admin**

Process:

- **Hospital Management System**

Data Flows:

- **Patient submits appointment request, receives confirmation.**
- **Receptionist registers patients and manages appointments.**
- **Doctor views patient history and updates treatment records.**
- **Admin manages user roles and hospital records.**

Data Stores (Shown in Level 1):

- **Patient Records**
- **Appointment Schedule**
- **Medical Records**
- **Billing Info**

Level 1 DFD – Major Subsystems
Breaks down the main system into key processes:

Processes:

1. **Patient Registration**
2. **Appointment Scheduling**
3. **Medical Consultation**
4. **Billing and Payment**
5. **Reports & Administration**

Data Stores:

- **D1: Patient Database**
- **D2: Appointment DB**
- **D3: Treatment/Diagnosis Records**
- **D4: Billing Records**

Data Flow Overview:

1. Patient Registration

- **Patient → [1] → Registration → D1**

2. Appointment Scheduling

- **Receptionist selects patient → [2] → Schedule Appointment → D2**
- **Patient receives schedule confirmation**

3. Medical Consultation

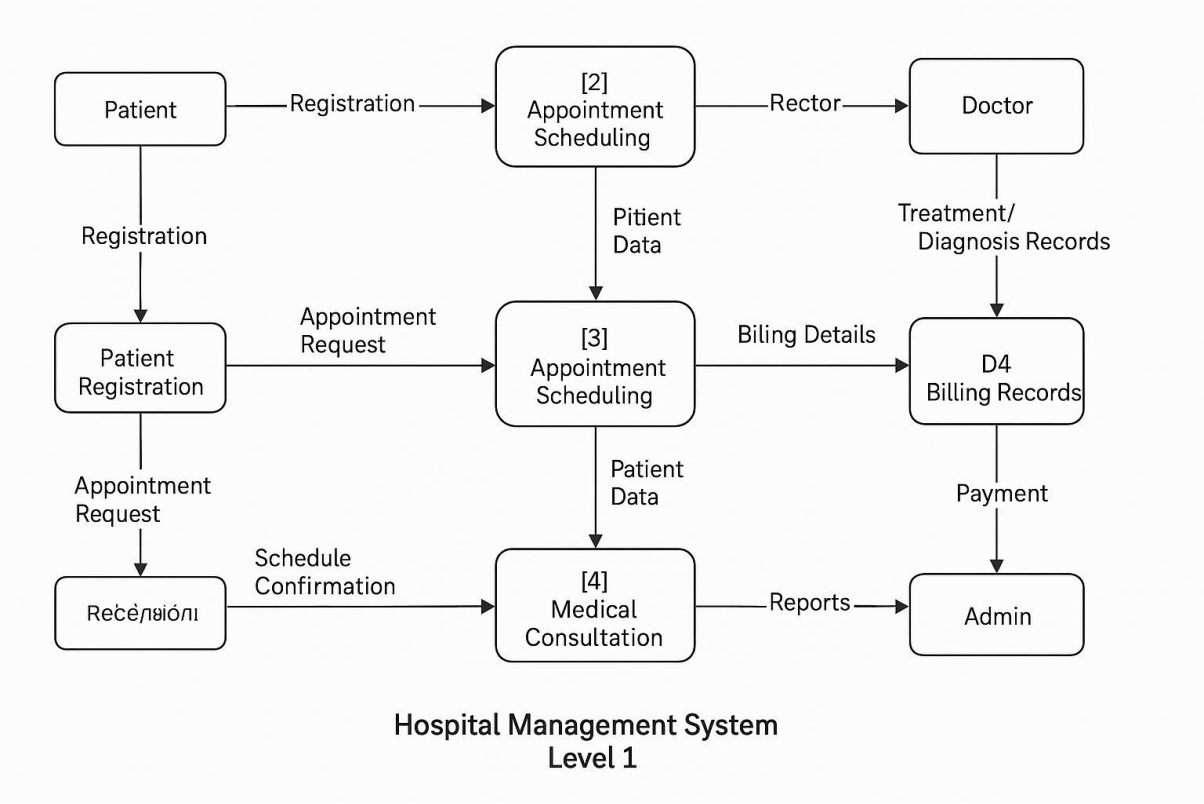
- **Doctor → [3] → Retrieve patient data from D1 and D3**
- **Doctor updates treatment info → D3**

4. Billing and Payment

- **Billing staff → [4] → Generate bill from D4**
- **Patient makes payment → Receipt issued**

5. Administration

- **Admin → [5] → Access reports from all modules**



Hospital Management System
Level 1

Desktop Application

LAB EXERCISE:

- **Build a simple desktop calculator application using a GUI library.**

Ans:

```
#include <gtk/gtk.h>

#include <stdlib.h>
```

```
#include <string.h>
#include <ctype.h>
typedef struct {
    GtkWidget *entry;
    char expression[256];
} CalculatorData;
void clear_expression(CalculatorData *data) {
    data->expression[0] = '\0';
    gtk_entry_set_text(GTK_ENTRY(data->entry), "");
}
void append_to_expression(CalculatorData *data, const char *text) {
    if (strlen(data->expression) + strlen(text) < sizeof(data->expression) - 1) {
        strcat(data->expression, text);
        gtk_entry_set_text(GTK_ENTRY(data->entry), data->expression);
    }
}
// Simple evaluation function for +, -, *, / with no operator precedence and no error checking
// This is to keep the example simple; complex parsing would require a proper expression parser
double evaluate_expression(const char *expr, int *error) {
    double result = 0;
    char current_op = '+';
    const char *p = expr;
    double current_num = 0;
    int has_num = 0;
    *error = 0;
    while (*p) {
        // Skip spaces
```

```
while (*p && isspace(*p)) p++;
// Parse number
char *endptr;
current_num = strtod(p, &endptr);
if (p == endptr) {
   // No valid number found
   *error = 1;
   return 0;
}
p = endptr;
// Apply operation
switch (current_op) {
   case '+': result += current_num; break;
   case '-': result -= current_num; break;
   case '*': result *= current_num; break;
   case '/':
      if (current_num == 0) {
         *error = 2; // division by zero
         return 0;
      }
      result /= current_num;
      break;
   default:
      *error = 1;
      return 0;
}
// Skip spaces
while (*p && isspace(*p)) p++;
```

```
        // Check for operator
        if (*p) {
            if (*p == '+' || *p == '-' || *p == '*' || *p == '/') {
                current_op = *p;
                p++;
            } else {
                *error = 1;
                return 0;
            }
        }
    }
    return result;
}
void button_clicked(GtkWidget *widget, gpointer user_data) {
    CalculatorData *data = (CalculatorData*)user_data;
    const char *button_text = gtk_button_get_label(GTK_BUTTON(widget));
    if (strcmp(button_text, "C") == 0) {
        clear_expression(data);
    } else if (strcmp(button_text, "=") == 0) {
        int error = 0;
        double result = evaluate_expression(data->expression, &error);
        if (error == 1) {
            gtk_entry_set_text(GTK_ENTRY(data->entry), "Error");
        } else if (error == 2) {
            gtk_entry_set_text(GTK_ENTRY(data->entry), "Div/0 Error");
        } else {
            char result_string[256];
            snprintf(result_string, sizeof(result_string), "%g", result);
```

```
            gtk_entry_set_text(GTK_ENTRY(data->entry), result_string);
            strncpy(data->expression, result_string, sizeof(data->expression));
        }
    } else {
        append_to_expression(data, button_text);
    }
}
int main(int argc, char *argv[]) {
    gtk_init(&argc, &argv);
    CalculatorData data;
    data.expression[0] = '\0';
    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Simple Calculator");
    gtk_window_set_default_size(GTK_WINDOW(window), 300, 400);
    gtk_container_set_border_width(GTK_CONTAINER(window), 10);
    g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);
    GtkWidget *vbox = gtk_box_new(GTK_ORIENTATION_VERTICAL, 5);
    gtk_container_add(GTK_CONTAINER(window), vbox);
    data.entry = gtk_entry_new();
    gtk_entry_set_alignment(GTK_ENTRY(data.entry), 1);  // Align right
    gtk_entry_set_editable(GTK_ENTRY(data.entry), FALSE);
    gtk_entry_set_text(GTK_ENTRY(data.entry), "");
    gtk_box_pack_start(GTK_BOX(vbox), data.entry, FALSE, FALSE, 0);
    GtkWidget *grid = gtk_grid_new();
    gtk_grid_set_row_spacing(GTK_GRID(grid), 5);
    gtk_grid_set_column_spacing(GTK_GRID(grid), 5);
    gtk_box_pack_start(GTK_BOX(vbox), grid, TRUE, TRUE, 0);
```

```
    const char *buttons[5][4] = {
        { "7", "8", "9", "/" },
        { "4", "5", "6", "*" },
        { "1", "2", "3", "-" },
        { "0", ".", "C", "+" },
        { "=", "", "", "" }
    };
    for (int i=0; i<5; i++) {
        for (int j=0; j<4; j++) {
            if (buttons[i][j][0] == '\0')
                continue;
            GtkWidget *button = gtk_button_new_with_label(buttons[i][j]);
            gtk_widget_set_hexpand(button, TRUE);
            gtk_widget_set_vexpand(button, TRUE);
            g_signal_connect(button, "clicked", G_CALLBACK(button_clicked), &data);
            gtk_grid_attach(GTK_GRID(grid), button, j, i, 1, 1);
        }
    }
    gtk_widget_show_all(window);
    gtk_main();
    return 0;
}
```

Flow Chart

LAB EXERCISE: Draw a flowchart representing the logic of a basic online registration system.

Steps of the Flowchart:

1. **Start – The process begins.**

2. **Display Registration Form – Show the user the form to enter details (name, email, password, etc.).**
3. **Input Details – User fills out the form.**
4. **Are Details Complete? – Decision:**
    - **Yes → proceed to next step.**
    - **No → prompt user to complete all fields and return to Step 3.**
5. **Submit Registration – User submits the form.**
6. **Validate Details (e.g., Email & Password format) – Decision:**
    - **Valid → go to next step.**
    - **Invalid → display error and return to Step 3.**
7. **Registration Successful? – Decision:**
    - **Yes → confirmation message.**
    - **No → display error and return to Step 3.**
8. **End – Process end**