Write a simple "Hello World" program in two different programming.

Hello World in C:

```
#include <stdio.h>

 void main()
{

   printf("Hello, World!\n");

}
```

Hello World in C++:

```
#include <iostream>

void main()
{

std::cout << "Hello, World!" << std::endl;

}
```

Key Differences in Structure & Syntax:

Aspect	**C**	**C++**
Header File	Uses #include <stdio.h> for standard I/O	Uses #include <iostream> for I/O
Output Statement	Uses printf()	Uses std::cout <<
Namespace	Not applicable	Uses std:: namespace prefix
Standard Library	Procedural, uses functions like printf()	Object-oriented, uses stream objects like cout
Language Style	Procedural programming	Object-oriented programming (OOP support).
Syntax Simplicity	Slightly more straightforward for beginners	More powerful

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

Client (Browser)

|

▼ DNS Query: "example.com → 93.184.216.34"

|

▼ TCP 3-Way Handshake (SYN, SYN-ACK, ACK)

|

▼ HTTPS/TLS Encryption (if secure)

|

▼ HTTP Request: GET / HTTP/1.1

|

Route Through Internet

| (Routers, ISP, BGP)

▼

Server (e.g., Nginx)

|

▼ Process Request → Fetch Data (e.g., HTML)

|

▼ HTTP Response: 200 OK + Data

|

▲

Client Renders Webpage

Design a simple HTTP client-server communication in any language?

<u>Simple HTTP Server in Python python:</u>

```
# simple_http_server.py
from http.server import BaseHTTPRequestHandler, HTTPServer
class SimpleHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        # Send response status code
        self.send_response(200)
        # Send headers
        self.send_header('Content-type', 'text/plain')
        self.end_headers()
       # Send message
        message = "Hello, World!"
        self.wfile.write(message.encode())
      # Server setup
       def run():
    server_address = ('', 8080)
    httpd = HTTPServer(server_address, SimpleHandler)
    print("HTTP server running on port 8080...")
    httpd.serve_forever()

run()
```

<u>Simple HTTP Client in Python:</u>

```
# simple_http_client.py
import http.client
# Connect to localhost server
```

```
conn = http.client.HTTPConnection("localhost", 8080)

# Send GET request

conn.request("GET", "/")

# Get the response

response = conn.getresponse()

print("Status:", response.status)

print("Response:\n", response.read().decode())

conn.close()
```

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

Here's a breakdown of the most common types of internet connections, along with their pros and cons:

<u>1. Definition & Scope:</u>

-Broadband is a general term for high-speed internet access that's always on (as opposed to old dial-up). It includes multiple technologies like DSL, cable, satellite etc.
-Fiber-optic internet is a specific type of broadband that uses thin glass fibers to transmit data as light signals, offering superior performance.

<u>2. Technology Used:</u>

-Broadband (non-fiber types) relies on existing infrastructure:

- DSL uses telephone lines (copper wires)
- Cable uses coaxial TV lines
- Satellite uses radio signals

- Fiber-optic uses hair-thin glass strands that transmit light pulses, allowing much faster data transfer.

<u>3. Speed Comparison:</u>

-Standard broadband (cable/DSL):

- Download: 10-500 Mbps (cable can reach 1 Gbps in some areas)
- Upload: Typically much slower than download (5-50 Mbps)

- Fiber-optic:
    - Download: 100 Mbps to 10 Gbps
    - Upload: Symmetrical speeds (same as download)

<u>4. Reliability & Performance:</u>

- Broadband (cable/DSL):
    - Spe eds fluctuate during peak hours
    - More susceptible to interference and distance limitations
    - Higher latency (20-50ms)
- Fiber-optic:
    - Consistent speeds regardless of time or distance
    - Immune to electromagnetic interference
    - Ultra-low latency (1-10ms)

<u>5. Availability & Cost:</u>

- Broadband is widely available, even in rural areas, and generally more affordable .
- Fiber-optic has limited availability (mostly urban areas) and is more expensive .

<u>6. Best Use Cases:</u>

- Standard broadband works well for:
    - Basic web browsing
    - HD video streaming
    - Small households
- Fiber-optic excels for:
    - 4K/8K streaming
    - Competitive online gaming
    - Large file uploads/downloads
    - Smart homes with multiple devices

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

You can simulate HTTP and FTP requests using command-line tools like curl. Below are examples for each protocol:

1. Simulate HTTP Requests using curl

->GET Request

```
curl http://example.com
```

Fetches the HTML content of the webpage

-> POST Request (send data)

```
curl-X POST-d "username=test&password=123" http://example.com/login
```

Sends form data using HTTP POST.

-> Custom Header

```
curl-H "User-Agent: CustomAgent/1.0" http://example.com
```

Simulates a request with a custom User-Agent.

->Save Response to File

```
curl http://example.com-o output.html
```

Saves the HTML response to a file named output.html.

2. Simulate FTP Requests using curl

-> Anonymous FTP File Download

```
curl ftp://ftp.example.com/file.txt-o file.txt
```

Downloads a file from an open FTP server.

-> FTP with Username and Password

```
curl-u username:password ftp://ftp.example.com/file.txt-o file.txt
```

Authenticated file download from FTP.

-> Upload File to FTP Server

```
curl-T localfile.txt-u username:password ftp://ftp.example.com/upload/
```

Uploads localfile.txt to the specified FTP server directory.

Notes:

-curl comes pre-installed on most Linux/macOS systems. On Windows, use curl from PowerShell or install via Chocolatey or WSL.

-For secure connections, use https (HTTP over SSL/TLS) or ftps for FTP Secure.

Identify and explain three common application security vulnerabilities.

1. SQL Injection (SQLi)

Description:
Occurs when an attacker manipulates SQL queries by injecting malicious SQL code through input fields.

Example:
sql

```
SELECT * FROM users WHERE username = 'admin' AND password = '1234';
```

If the input is not sanitized, an attacker might enter:
bash

```
' OR '1'='1
```

The resulting query:

sql

```
SELECT * FROM users WHERE username = '' OR '1'='1';
```

This could allow unauthorized access to the database.

Prevention:
-Use **prepared statements** or **ORMs**
-Validate and sanitize user inputs

2. Cross-Site Scripting (XSS)

Description:

Allows attackers to inject malicious scripts into web pages viewed by other users. These scripts can steal session cookies, redirect users, or manipulate page content.

Example:

If a comment form allows:

html

```
<script>alert('Hacked!');</script>
```

This will run in every user’s browser when they view that comment.

Prevention:

-Escape HTML output

-Use Content Security Policy (CSP)

-Sanitize user inputs

3. Cross-Site Request Forgery (CSRF)

Description:

Tricks a user into submitting unwanted actions on a web application where they are authenticated (like changing a password or making a purchase).

Example:

A malicious site causes a logged-in user to unknowingly send a request to:

arduino

```
http://bank.com/transfer?amount=1000&to=attacker
```

Prevention:

-Use anti-CSRF tokens

-Validate HTTP referer headers

-Require re-authentication for sensitive actions

Identify and classify 5 applications you use daily as either system software or application software.

APPLICATION	TYPE	CLASSIFICATION	EXPLANATION
WINDOWS OS / MACOS	Operating System	System Software	Manages computer hardware and provides core functions for other applications.
GOOGLE CHROME / SAFARI	Web Browser	Application Software	Lets you access websites and web apps via the internet.
MICROSOFT WORD	Word Processor	Application Software	Used for creating and editing documents.
ANTIVIRUS SOFTWARE (E.G., NORTON)	Security Tool	System Software	Protects and manages system-level operations like virus scanning and firewall.
WHATSAPP / TELEGRAM	Messaging App	Application Software	Used for communication via internet — not required for the system to function.

Summary:

-System Software: Runs in the background and supports core system operations.

-Application Software: Runs on top of system software to perform specific user tasks.

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

Web *Application*

Presentation	Business	Data
Logic	Tier	
(Client-side)	(Server)	(Database)
- HTML/CSS/JS	- Application Logic	- Database (MySQL,
- React/Angular	- API Endpoints	-PostgreSQL, MongoDB)
- Mobile Apps	- Authentication	- Caching (Redis)
- Thick Clients	- Validation Rules	- File Storage (S3)
	- Microservice	

1. Presentation Tier (Frontend):

Role: Handles user interaction and displays data.

Components:

Web UI: HTML/CSS, JavaScript frameworks (React, Vue, Angular).

Mobile Apps: iOS/Android clients.

Thick Clients: Desktop applications (Electron, Qt).

Key Responsibilities:

Renders user interfaces.

Sends HTTP requests to the Business Tier.

Validates user input (e.g., form checks).

2. Business Logic Tier (Backend):

Role: Processes requests, enforces rules, and manages data flow.

Components:

API Servers: RESTful APIs (Node.js, Django, Spring Boot).

Microservices: Auth service, payment service, etc.

Authentication: JWT/OAuth, session management.

Key Responsibilities:

Implements business rules (e.g., "Discounts apply only to premium users").

Handles authentication/authorization.

Communicates with the Data Tier.

How Data Flows:

User submits a request (e.g., "Place Order") → Presentation Tier.

Frontend sends an API call → Business Tier.

Backend validates the request, applies logic → Queries Data Tier.

Database returns results → Backend → Frontend → User.

Create a case study on the functionality of the presentation, business logic, and Data access layers of a given software system.

To understand the functionality and interaction of the three main layers—Presentation, Business Logic, and Data Access—in an online banking application.

Three-Tier Architecture Layers:

1. Presentation Layer (User Interface)

Purpose:
This layer allows customers to interact with the online banking system via web browsers or mobile apps.

Scenario:

User logs into their bank account and clicks "Transfer ₹500 to another account."

2.Business Logic Layer (Application Layer)

Purpose:
This layer contains the core rules and operations of the banking system. It processes input from the user, makes decisions, and ensures compliance with banking rules.

3. Data Access Layer (Persistence Layer)

Purpose:
This layer handles database operations, such as retrieving, inserting, updating, or deleting records in a secure and consistent manner.

Layer Interaction Flow:

Use Case: User transfers ₹10,000 from Account A to Account B

1. **Presentation Layer** collects data (recipient account, amount) and sends it via API.

2.**Business Logic Layer** validates the transfer (sufficient balance, fraud check) and invokes transaction processing.

3.**Data Access Layer** updates both account balances and stores the transaction in the database.

4.Response (Success/Failure) is returned up the chain to the user interface.

Types of Software Environments.

Software environments refer to the different setups used during the software development lifecycle. Each environment serves a distinct purpose.

1.Development Environment:

Purpose:

Used by developers to write, compile, and debug code.

Common Tools:

Code editors (VS Code, IntelliJ)

Local databases (MySQL, SQLite)

Package managers (npm, pip)

Docker (for containerized environments)

2. Testing Environment:

Purpose:

Used to run unit, integration, and system tests to identify bugs before release.

Key Features:

Mimics production setup

Often automated with CI/CD tools (Jenkins, GitHub Actions)

May use mock databases or real test data

3.Production Environment:

Purpose:

Live environment where the application is deployed for end-users.

Characteristics:

Highly secure

Optimized for performance

Monitored 24/7

Scalable infrastructure (AWS, Azure, GCP)

Set up a basic environment in a virtual machine.

Step-by-Step Setup:

1. Install VirtualBox

Download and install VirtualBox from https://www.virtualbox.org.

2. Create a New VM

Open VirtualBox > New VM > Choose "Ubuntu (64-bit)"

Allocate 2 GB RAM and 20 GB disk space

3. Attach Ubuntu ISO and Install OS

Start the VM

Select the downloaded Ubuntu ISO

Follow the on-screen instructions to install Ubuntu

4. Set Up Development Tools

Once Ubuntu is installed:

```
sudo apt update

sudo apt install build-essential git curl python3 python3-pip
```

5. Create a Test Flask App

pip3 install flaskCreate a file app.py:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')

def home():

return "Hello from Development Environment!"

app.run(debug=True)
```

Run the app:

```
python3 app.py
```

Write and upload your first source code file to Github.

Link: https://github.com/sanjay-5355/module-1-.git

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

Here's a sample list of commonly used software, classified into System, Application, and Utility Software categories:

1. System Software:

Software	Purpose
Windows 10 / 11	Operating system
macOS	Operating system
Linux (Ubuntu, Fedora)	Operating system
Android / iOS	Mobile operating systems
BIOS / UEFI	Boot-level hardware initialization
Device Drivers	Hardware communication

2. Application Software:

Software	Purpose
Google Chrome / Firefox	Web browsing
Microsoft Word / Excel	Document and spreadsheet editing
Adobe Photoshop	Image editing
VLC Media Player	Audio/video playback
WhatsApp / Telegram	Messaging and communication
Zoom / Microsoft Teams	Video conferencing
Spotify	

3. Utility Software:

Software	Purpose
WinRAR / 7-Zip	File compression and extraction
Windows Defender / Avast	Antivirus and security
CCleaner	Disk cleanup and optimization
Backup & Restore (Windows	Data backup
Disk Management	Partition and disk setup
Task Manager / Activity Monitor	Monitor system performance

Write a report on the various types of application software and

how they improve

Application software refers to programs designed to help users perform specific tasks such as writing, designing, calculating, communicating, or managing data. These software tools are essential across industries and day-to-day personal use because they improve speed, efficiency, and accuracy in work.

<u>Types of Application Software:</u>

1. Word Processing Software-

Examples: Microsoft Word, Google Docs, LibreOffice Writer

Purpose: Create, format, and edit text documents.

Productivity Benefits:

Speeds up documentation and content creation.

Features like templates, grammar check, and spell-check save time.

Collaboration tools (e.g., Google Docs) support real-time co-authoring.

2. Spreadsheet Software-

Examples: Microsoft Excel, Google Sheets, LibreOffice Calc

Purpose: Organize, analyze, and calculate data using formulas and charts.

Productivity Benefits:

Automates calculations and financial modeling.

Charts and graphs improve data visualization.

Useful for budgeting, project tracking, and data analysis.

3. Presentation Software-

Examples: Microsoft PowerPoint, Google Slides, Prezi

Purpose: Create visual and engaging presentations.

Productivity Benefits:

Enhances communication through visual storytelling.

Saves time with design templates and drag-and-drop features.

Supports team collaboration for meetings and training.

4. Database Management Software-

Examples: Microsoft Access, MySQL, Oracle DB

Purpose: Store, retrieve, and manage large amounts of structured data.

Productivity Benefits:

Organizes business data efficiently.

Reduces manual record-keeping.

Enables quick search, sorting, and reporting.

5. Communication Software-

Examples: Zoom, Microsoft Teams, Slack, Skype

Purpose: Facilitate real-time communication and collaboration.

Productivity Benefits:

Reduces delays with instant messaging and video calls.

Centralizes discussions, file sharing, and meetings.

Essential for remote work and distributed teams.

6. Graphic Design and Multimedia Software-

Examples: Adobe Photoshop, Canva, CorelDRAW

Purpose: Create and edit images, videos, and other media content.

Productivity Benefits:

Enables fast creation of marketing and branding materials.

Templates and automation tools speed up design workflows.

Used widely in advertising, education, and entertainment.

7. Project Management Software-

Examples: Trello, Asana, Microsoft Project, Jira

Purpose: Plan, track, and manage tasks and resources.

Productivity Benefits:

Helps teams stay organized and meet deadlines.

Improves visibility and accountability.

Facilitates collaboration across departments.

8. Web Browsers-

Examples: Google Chrome, Mozilla Firefox, Safari

Purpose: Access online content and web applications.

Productivity Benefits:

Quick access to information and cloud-based tools.

Supports extensions and bookmarks to optimize workflow.

Enables remote work through web apps and platforms.

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

Phases (Use Rectangular Boxes for Each Step):

1. Requirement Gathering & Analysis

Input: Stakeholder needs, business goals.

Output: SRS (Software Requirement Specification).

2.Planning

Input: SRS document.

Output: Project plan, timeline, cost estimation.

3.Design

Input: Requirements.

Output: System architecture, prototypes, UML diagrams.

4.Development (Coding)

Input: Design documents.

Output: Functional software code.

5.Testing

Input: Developed software.
Output: Test reports, bug fixes.

6.Deployment

Input: Tested software.

Output: Live system for users.

7.Maintenance

Input: User feedback, bug reports.

Output: Updates, patches, improvements.

Connectors (Use Arrows)

Flow should be sequential:

Requirement → Planning → Design → Development → Testing → Deployment → Maintenance

Decision Points (Use Diamonds)

After Testing:

"Are all bugs fixed?"

Yes → Proceed to Deployment.

No → Loop back to Development.

After Deployment:

"Are updates needed?"

Yes → Proceed to Maintenance (and possibly revisit earlier phases).

No → End.

Feedback Loops (Dashed Arrows)

From Maintenance back to Requirement Gathering (for iterative improvements).

Visual Flowchart Example (Sketch This on Paper):

[Requirement Gathering] → [Planning] → [Design] → [Development] → [Testing]

↑ ↓

[Maintenance] ← [Deployment] ← (Bug Fixes)

Write a requirement specification for a simple library management system.

Project Title: Simple Library Management System

Version: 1.0

Prepared By: [Your Name/Team]

Date: [Insert Date]

1. Introduction

1.1 Purpose

The purpose of this document is to specify the functional and non-functional requirements for the development of a simple Library Management System (LMS). This system will help in managing books, members, borrowing, and returning activities efficiently.

1.2 Scope

The system will manage:
- Book inventory (add/update/delete)
- Member registration and records
- Book issue and return
- Fine calculation for late returns
- Reports for issued/returned books

It will be used by librarians, library members, and administrators.

1.3 Definitions, Acronyms, Abbreviations

- LMS – Library Management System
- ISBN – International Standard Book Number
- GUI – Graphical User Interface

2. Overall Description

2.1 Product Perspective

This is a standalone web or desktop application with an integrated database. It replaces traditional paper-based record systems.

2.2 User Classes and Characteristics

- Librarian – Manages books and members, handles book issuing/returning
- Member/User – Views catalog, requests books
- Admin – Manages librarian accounts and system settings

2.3 Operating Environment

Windows / Linux OS
Web Browser (for web app)
Localhost or cloud-hosted database (MySQL/PostgreSQL)

2.4 Constraints

The system must be user-friendly and lightweight
The system should support up to 1000 members and 5000 books
Developed using open-source tools

3. Functional Requirements

3.1 Book Management

- Add new books with title, author, ISBN, category, and quantity
- Edit or delete existing books
- Search and filter books by title/author/category

3.2 Member Management

- Register new members with personal details
- Edit or remove member profiles
- Search members by name or ID

3.3 Book Issue/Return

- Issue a book to a valid member (check availability)
- Set due date automatically (e.g., 14 days from issue)
- Return book and mark as available
- Calculate and apply late return fines

3.4 Reports and Logs

- Generate daily/weekly issue-return report
- View currently issued books
- Track member borrowing history

4. Non-Functional Requirements

4.1 Performance Requirements

Should respond to actions within 2 seconds
Support concurrent access by multiple users (if web-based)

4.2 Security Requirements

Login system for librarians and admins
Role-based access (member/librarian/admin)
Passwords stored securely (hashed)

4.3 Usability Requirements

GUI should be intuitive for non-technical users
Use of clear menus and labels

4.4 Portability

Runs on Windows and Linux
Optionally deployable on web server (Apache, Tomcat)

5. Future Enhancements

Email/SMS notifications for due dates
Barcode scanning for books
Online member portal with book reservations

6. Appendix

Sample book entry: Title: The Alchemist, Author: Paulo Coelho, ISBN: 1234567890
Default fine: ₹5/day after the due date.

Perform a functional analysis for an online shopping system.

1. Purpose of the System:

The online shopping system enables users to browse, search, and purchase products over the internet. It handles product management, user accounts, cart operations, payment, and order tracking.

2. Key Actors (Users):

- Customer: Browses, adds items to cart, places orders
- Admin: Manages product listings, inventory, users, and orders
- Delivery Staff: Updates delivery status of orders
- System: Handles authentication, cart, payments, and notifications

3. Core Functional Requirements:

3.1 User Registration and Authentication

- Customers can sign up, log in, and log out
- Users can reset forgotten passwords
- Admins authenticate with role-based access

3.2 Product Catalog Management

- Admins can add, edit, delete product listings
- Products have attributes: name, price, image, description, stock
- Customers can browse, filter, and search products by category, brand, or price

3.3 Shopping Cart

- Customers can add items to their cart
- Items in cart can be updated or removed
- Cart shows total price, quantity, and selected items

3.4 Order Management

- Customers can place orders for items in the cart
- System calculates total cost including taxes and shipping
- Customers receive order confirmation
- Admins can view, update, or cancel orders

3.5 Payment Gateway Integration

- Integration with payment services (e.g., Razorpay, PayPal, Stripe)
- Supports credit/debit cards, UPI, or Cash on Delivery
- Payment status is tracked (Pending, Paid, Failed)

3.6 Order Tracking & Delivery

- Customers can view order history and track delivery status
- Delivery staff updates status: 'Shipped', 'Out for Delivery', 'Delivered'
- Notifications sent via email/SMS

3.7 Customer Feedback and Ratings

- Customers can leave ratings and reviews for products
- Admins can moderate feedback

3.8 Admin Dashboard

- Manage all users, products, orders, and reports
- Generate sales and inventory reports
- View product popularity and user activity analytics

4. Additional Functional Considerations:

- Wishlist: Save products to purchase later
- Promotions/Coupons: Apply discount codes at checkout
- Inventory Management: Automatically update stock after each order
- Email Notifications: Order updates, confirmations, and promotional offers
- Multi-language Support: Interface adapts to user’s language preferences

5. Example Use Case: Place an Order

1. Customer logs in – Authenticates using email/password
2. Browses product catalog – Views electronics category
3. Adds item to cart – Selects quantity, adds to cart
4. Proceeds to checkout – Confirms shipping address and payment
5. Order placed – System stores order, updates inventory

6. Customer gets notification – Email/SMS confirmation sent
7. Admin sees order – Prepares for shipping

6. Conclusion

This functional analysis ensures that all user needs — from browsing to checkout and delivery — are covered in the system. Each function supports the core goal: a smooth, secure, and user-friendly shopping experience**.**

Design a basic system architecture for a food delivery app.

Here is a basic system architecture design for a Food Delivery App — similar to services like Zomato, Swiggy, or Uber Eats.

1. Architecture Overview

The system follows a **multi-tier (layered) architecture**, typically including:

- **Client Layer** (Frontend)
- **Application Layer** (Backend Services)
- **Data Layer** (Database)
- **External Integrations** (Payment, Maps, SMS)

2. Components and Layers

A. Client Layer (User Interfaces)

- Customer App (iOS/Android/Web)
- Restaurant Dashboard (Web App)
- Delivery Partner App (Android/iOS)
- Admin Panel (Web-based)

Responsibilities:

- Browsing restaurants/menus
- Placing orders
- Real-time order status

- Map-based delivery tracking

B. Application Layer (Backend APIs & Business Logic)

Modules and Responsibilities:

- **User Service** – Registration, login, profile, preferences
- **Restaurant Service** – Menu management, order handling
- **Order Service** – Placing, updating, tracking orders
- **Delivery Service** – Assigning, tracking, and updating delivery
- **Notification Service** – Sends push, email, or SMS alerts
- **Payment Service** – Payment gateway integration, refunds
- **Admin Service** – Analytics, reports, moderation

Technologies often used: **Node.js, Django, Spring Boot**
Communication: **RESTful APIs or GraphQL**

C. Data Layer (Databases)

- **Relational DB** (PostgreSQL, MySQL): Users, restaurants, orders, payments
- **NoSQL DB** (MongoDB): Menu items, reviews, logs
- **In-memory Store** (Redis): Session management, caching
- **Blob Storage** (S3 or similar): Restaurant logos, food images

D. Third-Party Integrations

- **Payment Gateway** (Razorpay, Stripe): Handle transactions
- **Maps API** (Google Maps): Geolocation & routing
- **SMS/Email API** (Twilio, SendGrid): Notifications
- **Push Notifications** (Firebase, OneSignal): Real-time alerts

3. High-Level Flow

Customer Places Order

↓

Backend Validates & Stores Order

↓

Restaurant Accepts & Prepares

↓

Delivery Partner Assigned

↓

Order Picked & Delivered

↓

Payment Settled

4. Non-Functional Considerations

- **Scalability** – Use load balancers, microservices
- **Security** – OAuth2, data encryption, secure APIs
- **Performance** – Caching, CDN for assets
- **Monitoring** – Log tracking (ELK), uptime monitoring (Datadog, New Relic)

5. Optional: Diagram Representation

A visual diagram using boxes and arrows can represent this architecture. It typically includes:

-Users (Customer, Restaurant, Delivery)

-APIs/Services

-Databases

-External integrations (Maps, Payment)

Develop test cases for a simple calculator program .

Assume the calculator performs: **Addition, Subtraction, Multiplication, and Division.**

Functional Test Cases:

Test Case ID	Description	Input	Expected Output
TC01	Addition of two positive integers	5 + 3	8
TC02	Subtraction resulting in positive	10- 2	8
TC03	Subtraction resulting in negative	2- 10	-8
TC04	Multiplication of integers	4 × 3	12
TC05	Division with valid input	12 ÷ 3	4
TC06	Division by zero	8 ÷ 0	Error/Exception
TC07	Decimal addition	2.5 + 3.1	5.6
TC08	Negative number multiplication	-5 × 2	-10
TC09	Large number addition	99999 + 1	100000
TC10	Zero multiplication	0 × 50	0

Input Validation Test Cases:

Test Case ID	Description	Input	Expected Output
TC11	Invalid character input	a + b	Invalid Input Error
TC12	Incomplete expression	7 +	Error
TC13	Consecutive operators	8 ++ 2	Error

Test Case ID	Description	Input	Expected Output
TC14	Empty input	(blank)	Error

<u>Boundary and Edge Test Cases:</u>

Test Case ID	Description	Input	Expected Output
TC15	Negative decimal input	-3.5 + 1.2	-2.3
TC16	Leading/trailing spaces	5 + 2	7
TC17	Zero divided by number	0 ÷ 5	0

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

<u>The Ariane 5 Rocket Failure</u>

1. Introduction

Software System: Ariane 5 Rocket Flight Control System (European Space Agency- ESA)

Date of Incident: June 4, 1996

Impact: Rocket self-destructed 37 seconds after launch, causing a $370 million loss.

2. Root Cause Analysis

A. The Software Bug-

Issue: A 64-bit floating-point number (horizontal velocity) was converted into a 16-bit signed integer in the Inertial Reference System (IRS).

Result: Integer overflow → Crash of the primary and backup IRS systems.

Why It Happened:

-The same software from Ariane 4 was reused without proper testing for Ariane 5's higher velocities.
-No exception handling for arithmetic overflow.

B. System Design Flaws-

-No Fail-Safe Mode:
When both IRS systems failed, the rocket's onboard computer triggered self-destruct.

-Inadequate Testing:
The error only occurred under Ariane 5's flight profile (not simulated in Ariane 4 tests).

3. Maintenance & Fixes Implemented

A. Short-Term Fixes (Post-Failure)

1.Code Review & Patch-

Added overflow checks for floating-point conversions.

Modified the IRS to shut down gracefully (instead of crashing).

2.Simulation Testing-

Ran rigorous trajectory simulations with Ariane 5's flight dynamics.

B. Long-Term Improvements

New Software Development Standards

-ESA enforced strict exception handling in safety-critical systems.

Independent Audits

-Third-party verification for flight control software.

Redundancy Enhancements

-Improved backup system isolation to prevent cascading failures.

4. Lessons Learned:

Issue	Lesson
Reusing Legacy Code	Always retest repurposed software in the new environment.
Error Handling	Validate all edge cases (overflow, divide-by-zero, etc.).

Redundancy Failure	Backup systems must fail independently (no common-mode failures).
Testing	Simulate real-world conditions (not just expected scenarios).

5. Outcome:

Ariane 5’s Next Launch (1997): Successful after software fixes.

Industry Impact:

Became a textbook case for software engineering ethics.

Influenced NASA & SpaceX flight software standards.

Create a DFD for a hospital management system.

Level 0 DFD – Hospital Management System:

The Level 0 DFD provides a high-level overview of the Hospital Management System. It shows the interaction between external entities and the system as a single process.

External Entities:

- Patient
- Doctor
- Staff

Main Process:

- Hospital Management System

Data Stores:

- Patient Records
- Appointments
- Medical Records
- Staff Records

Data Flows:

- Patients register, update details, and book appointments
- Doctors access patient data and update medical records
- Staff manage administrative tasks and access staff records

Flow Summary:

[Patient] → [Hospital Management System] → [Patient Records]
[Doctor] → [Hospital Management System] → [Medical Records]
[Staff] → [Hospital Management System] → [Staff Records]
[Patient] → [Hospital Management System] → [Appointments]

Level 1 DFD – Hospital Management System

The Level 1 DFD breaks down the main Hospital Management System process into multiple subprocesses.

Subprocesses:

1. Patient Registration
2. Appointment Scheduling
3. Medical Diagnosis & Treatment
4. Billing & Payment
5. Staff Management
6. Report Generation

Detailed Data Flows:

- Patients register and their information is stored in Patient Records
- Patients schedule appointments and the data is saved in Appointments
- Doctors diagnose and update Medical Records
- Billing system calculates charges and stores payment data
- Admin staff manage Staff Records and assign duties
- Reports are generated for hospital analytics and performance

External Entities in Level 1:

- Patient
- Doctor
- Receptionist/Staff
- Admin

Build a simple desktop calculator application using a GUI library.

This is a basic **desktop calculator application** built using Python's **Tkinter** GUI library. It supports:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Clearing the display (C)

Code:

```
import tkinter as tk


# Define the button click function
def click(event):
    current = str(entry.get())
    button_text = event.widget.cget("text")
    if button_text == "=":
        try:
            result = str(eval(current))
            entry.delete(0, tk.END)
            entry.insert(tk.END, result)
        except Exception:
            entry.delete(0, tk.END)
            entry.insert(tk.END, "Error")
    elif button_text == "C":
        entry.delete(0, tk.END)
    else:
        entry.insert(tk.END, button_text)
```

```
# Create the main application window
window = tk.Tk()
window.title("Simple Calculator")
window.geometry("300x400")

# Input display field
entry = tk.Entry(window, font="Arial 20")
entry.pack(fill=tk.BOTH, ipadx=8, ipady=15, pady=10)

# Create button frame
button_frame = tk.Frame(window)
button_frame.pack()

# Define button layout
buttons = [
    ["7", "8", "9", "/"],
    ["4", "5", "6", "*"],
    ["1", "2", "3", "-"],
    ["C", "0", "=", "+"]
]

# Generate buttons dynamically
for row in buttons:
    frame = tk.Frame(button_frame)
    frame.pack(expand=True, fill='both')
    for button_text in row:
        button = tk.Button(frame, text=button_text, font="Arial 18", height=2)
        button.pack(side=tk.LEFT, expand=True, fill='both')
    button.bind("<Button-1>", click)


# Run the application

window.mainloop()
```

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

```
[Start]
  |
  ▼
[Enter Registration Details]
  |
  ▼
┌───────────────────┐
| Are    Details         |
|    Valid?               |
└─────────┬─────────┘
  |No           |Yes
  ▼             ▼
[Show Error    [Create User
 Message]       Account]
  |                |
  └──────┬──────┘
      ▼
[Display Confirmation]
  |
  ▼
[End]
```