Module: 1

(1). What is a Program?

A program is a set of instructions written in a specific language that a computer can understand and execute to perform a task or solve a problem.

(2). Write a simple "Hello World" program in two different programming languages of your choice. Compare the structure and syntax.

C lang using:

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

Output: Hello World

Python lang using:

```
print ("Hello, World!")
```

(3). Explain in your own words what a program is and how it functions.

A program is a set of instructions written in a specific language that a computer can understand and execute to perform a task or solve a problem.

Here's how it functions:

1. **Input**: The program may take input from the user, a file, or another source. Input is the data or information that the program uses to perform its task.
2. **Processing**: The program follows the written instructions step by step to manipulate or process the input data. This might include calculations, data sorting, or applying algorithms.
3. **Output**: After processing, the program generates an output. This could be a visual display, a written file, or an action performed by the computer.
4. **Control Flow**: Programs often have a logical structure that includes decisions (e.g., "if this happens, do that") and loops (e.g., "repeat this task until a condition is met"). These determine the sequence and repetition of tasks.
5. **Storage and Memory**: Programs may store data temporarily in memory while running or save data to files for future use.

(4). What is Programming?

Programming is the process of designing, writing, testing, and maintaining instructions (code) that a computer follows to perform specific tasks. It involves using a programming language to create software applications, automate processes, or solve problems. These instructions are written in a way that the computer can understand and execute, enabling it to perform actions, manipulate data, and generate output.

(5). What are the key steps involved in the programming process?

1. **Problem Definition/Planning**:
    - Understand the problem or task you are trying to solve.
    - Break down the problem into smaller, manageable parts.
    - Define the expected outcomes or goals.
2. **Designing the Solution**:
    - Plan how to solve the problem.
    - Choose the right programming language and tools for the task.
    - Design the overall structure of the program, such as algorithms, data structures, and logic flow.
3. **Writing the Code**:
    - Translate the design into actual code using the chosen programming language.
    - Follow the syntax and rules of the programming language.
    - Write the instructions that the computer will execute.
4. **Testing**:
    - Run the program to check if it works as expected.
    - Look for bugs, errors, or unexpected behaviour in the program.
    - Test different scenarios to ensure the program is functioning correctly.
5. **Debugging**:
    - Identify and fix any issues or bugs in the code that are causing the program to malfunction.
    - Use debugging tools or manual techniques to pinpoint and resolve errors.
6. **Optimization**:
    - Improve the program's efficiency by refining the code.
    - This could involve making the program run faster, use less memory, or be more readable.
7. **Documentation**:
    - Write clear comments and documentation to explain how the code works.
    - This helps others (or yourself) understand the program in the future.
8. **Deployment/Release**:
    - Once the program is complete and tested, it can be deployed or released for use.
    - This might involve distributing the software or making it available to users.
9. **Maintenance**:
    - After deployment, continue to monitor the program for any issues.
    - Update and maintain the program by fixing bugs, adding features, or improving performance as needed.

(6). Types of Programming Languages

- Procedural Programming Language C
- Functional Programming Language C++
- Logical Programming Language Prolog
- Object oriented Programming Language Java

(7). What are the main differences between high-level and low-level programming languages?

Feature	High-Level Languages	Low-Level Languages
Abstraction	High (away from hardware)	Low (close to hardware)
Ease of Use	Easier to learn and use	More difficult to learn and use
ControlOver Hardware	Less control over hardware	More control over hardware
Portability	More portable	Less portable
Performance	Slower execution due to abstraction	Faster execution
Error Handling	Built-in error handling	Manual error handling
Examples	Python, Java, C++	Assembly, Machine Code

(8). World Wide Web & How Internet Works

The World Wide Web is a system of interlinked documents and multimedia resources, accessed via the Internet. It allows users to view and interact with text, images, videos, and other content using web browsers (e.g., Chrome, Firefox, Safari).

Key Components of the WWW:

1. **Web Pages**: Documents written in HTML (Hypertext Markup Language) and styled using CSS (Cascading Style Sheets).
2. **Web Browsers**: Software applications used to access web pages (e.g., Google Chrome, Microsoft Edge).
3. **URLs (Uniform Resource Locators)**: Addresses that specify the location of web pages on the Internet.
4. **HTTP/HTTPS (Hypertext Transfer Protocol)**: Protocols used to transfer data between web browsers and servers securely.
5. **Web Servers**: Computers that store, process, and deliver web pages to users.

Internet: The Internet is a vast global network of interconnected computers that communicate with each other. It serves as the infrastructure that supports the World Wide Web and other services like email, file sharing, and online gaming

Key Components of the Internet:

1. **Physical Infrastructure**: Includes routers, cables, satellites, and data centers that connect devices worldwide.
2. **Protocols**: Rules that govern data transfer, such as:
    - **IP (Internet Protocol)**: Assigns unique addresses to devices.
    - **TCP (Transmission Control Protocol)**: Ensures data is sent and received accurately.
3. **ISPs (Internet Service Providers)**: Companies that provide users with access to the Internet.
4. **DNS (Domain Name System)**: Converts human-readable domain names (e.g., www.example.com) into IP addresses.

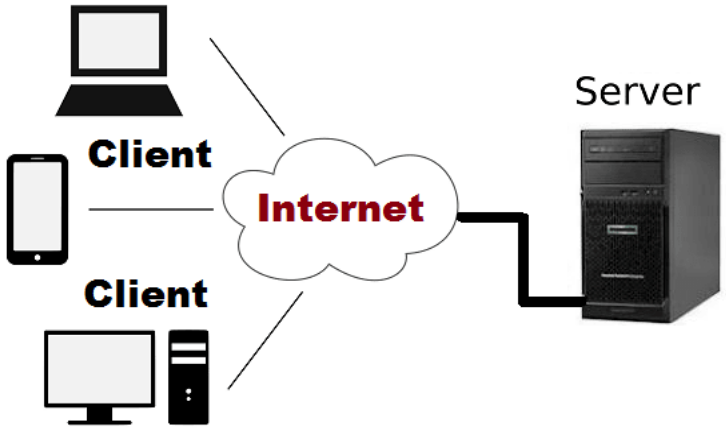
How the Internet Works

1. **Data Transmission**:

- Devices communicate over the Internet using packets, which are small units of data.
- These packets travel through routers and switches to reach their destination.

2. **Protocols**:
   - The **TCP/IP model** ensures reliable transmission and routing of data.
   - DNS translates web addresses into IP addresses to locate servers.
3. **Client-Server Model**:
   - The Internet uses a client-server architecture.
   - Your device (client) requests a service (e.g., viewing a webpage) from a server, which processes the request and sends the desired data back.
4. **Web Access**:
   - When you type a URL into a browser, the browser sends a request to the DNS to find the corresponding IP address.
   - The browser connects to the web server using this address and fetches the requested web page.
   - The page is displayed on your screen, often with multimedia content.

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



(10). Describe the roles of the client and server in web communication.

Role of the Client:

The client is a device or application that initiates communication by requesting resources or services from a server.

Key Functions:

1. **Requesting Data**:
   - The client (e.g., a web browser like Chrome or Safari) sends a request to a server for specific content or services, such as a web page or an API response.
   - Requests are often made using HTTP or HTTPS protocols.
2. **Rendering Content**:
   - Once the client receives data (e.g., HTML, CSS, JavaScript), it processes and displays it to the user.
   - Web browsers interpret and render the data into a readable and interactive format.
3. **User Interaction**:
   - The client provides an interface for users to interact with applications, websites, or services (e.g., filling forms, clicking links).

Examples:

- Web browsers (Chrome, Firefox, Safari)
- Mobile apps that access online resources (e.g., social media apps)
- IoT devices requesting updates or services from servers

Role of the Server:

The server is a system (hardware and software) that responds to client requests by providing the requested resources or performing specified tasks.

Key Functions*:*

1. **Storing Resources**:
    - Servers host files, databases, and application logic that clients request, such as web pages, images, videos, and scripts.
2. **Processing Requests**:
    - Upon receiving a client request, the server processes it, retrieves the appropriate data, and sends a response.
    - For dynamic requests, servers often interact with backend databases or applications.
3. **Ensuring Security and Access Control**:
    - Servers authenticate and authorize users, ensuring data is accessible only to legitimate clients.
    - HTTPS protocols ensure secure data transfer.

Examples*:*

- Web servers (e.g., Apache, Nginx, Microsoft IIS)
- Application servers for backend logic (e.g., Node.js, Spring Boot)
- Database servers (e.g., MySQL, MongoDB)

Client-Server Communication:

1. **Initiation**:
    - A user interacts with a client application (e.g., entering a URL into a browser).
2. **Request**:
    - The client sends a request to the server using an HTTP/HTTPS protocol.
3. **Processing**:
    - The server receives, processes the request, and fetches or generates the required data.
4. **Response**:
    - The server sends the response back to the client.
5. **Rendering**:
    - The client processes the response and displays it to the user.

(11). Network Layers on Client and Server

Client-Side Network Layers: The client-side layers process the user request and send it to the server:

1. **Application Layer**:
    - Interfaces directly with the user.
    - Examples: Web browsers, email clients, and mobile apps.
    - Protocols: HTTP/HTTPS (web), FTP (file transfer), SMTP (email).
2. **Transport Layer**:
    - Ensures reliable data transfer.
    - Breaks the request into smaller packets and assigns port numbers.
    - Protocols: TCP (for reliable communication) or UDP (for faster, connectionless communication).
3. **Network Layer**:
    - Adds source and destination IP addresses to packets for routing.
    - Determines the best path for the data to travel across the internet.
    - Protocol: IP (Internet Protocol).
4. **Data Link Layer**:
    - Prepares packets for physical transmission by encapsulating them into frames.
    - Adds MAC addresses for local delivery within the same network segment.
    - Examples: Ethernet, Wi-Fi.
5. **Physical Layer**:
    - Converts frames into electrical, optical, or radio signals.
    - Physically transmits the data over cables, fibre optics, or wireless signals.

Server-Side Network Layers: The server-side layers reverse the process to receive and process the client's request:

1. **Physical Layer**:
    - The server receives the electrical, optical, or radio signals transmitted over the network.
2. **Data Link Layer**:
    - Extracts frames from the received signals.
    - Verifies the MAC address for local delivery.
3. **Network Layer**:
    - Examines the IP header of the packets.
    - Confirms that the packet is addressed to the server.
    - Handles packet reassembly if data is split across multiple packets.
4. **Transport Layer**:
    - Ensures all packets are received and reassembled correctly.
    - Confirms the correct application is targeted using port numbers.
    - Handles acknowledgments for reliable delivery (if using TCP).
5. **Application Layer**:
    - Processes the request received by the server's application.
    - For example:
        - A web server (e.g., Apache, Nginx) responds to HTTP/HTTPS requests.
        - A database server handles queries and sends results back.

(12). Explain the function of the TCP/IP model and its layers

Functions of the TCP/IP Model

1. **Standardization**: Provides a universal set of protocols for devices to communicate.
2. **Data Encapsulation**: Breaks data into packets and ensures it is transmitted and reassembled correctly.
3. **Interoperability**: Allows diverse hardware and software systems to communicate.
4. **Scalability**: Supports a wide range of devices and networks, from local networks to the global internet

Types Of layers:

1.Presentation Layer:

2.Application Layer

3.Business Layer

4.Persistence Layer

5.Database Layer

(13). Client and Servers

Client**:** A client is a device or application that initiates a request for services or resources from a server. It is typically the "consumer" in this interaction.

Characteristics of a Client*:*

1. **Request Initiator**:
    - Sends requests to servers to fetch data or perform actions.
    - Example: A web browser requesting a webpage from a web server.
2. **User-Focused**:
    - Interacts directly with the end-user through an interface.
    - Examples: Web browsers, email applications, mobile apps.
3. **Temporary Connection**:
    - Establishes connections to the server as needed.
    - Example: A client connects to a web server only while browsing the webpage.
4. **Examples of Client Devices**:
    - Desktop computers, laptops, smartphones, IoT devices.
5. **Examples of Client Applications**:
    - Browsers: Chrome, Firefox.
    - Email apps: Outlook, Gmail.
    - File transfer apps: FileZilla.

Server: A server is a system (hardware or software) that responds to client requests by providing services, resources, or data. It is the "provider" in this interaction.

Characteristics of a Server:

1. **Request Responder**:
    - Always available to process and respond to client requests.
    - Example: A web server responding to a client browser with a webpage.

2. **Centralized Resources**:
    - Stores and manages resources or services that client's access.
    - Examples: Web pages, files, databases.
3. **Persistent Connection**:
    - Operates continuously to serve multiple clients simultaneously.
4. **Examples of Server Types**:
    - **Web Server**: Delivers web pages (e.g., Apache, Nginx).
    - **File Server**: Manages file storage and retrieval.
    - **Database Server**: Manages and queries data (e.g., MySQL, PostgreSQL).
    - **Application Server**: Processes application logic (e.g., Node.js, Django).

(14). Explain Client Server Communication

The process of communication between a client and a server typically involves the following steps:

1. **Client Initiates a Request**:
    - The client sends a request to the server for specific data or services.
    - Example: A browser (client) requests a webpage from a web server.
2. **Server Receives the Request**:
    - The server listens for incoming requests on a specific port.
    - Once it receives the request, it processes it by performing the required operation (e.g., fetching a file, running a query).
3. **Server Sends a Response**:
    - The server sends the processed data or the results of the operation back to the client.
4. **Client Processes the Response**:
    - The client receives the response and uses it to fulfil the user's request (e.g., displaying a webpage, showing search results).

(15). Types of Internet Connections

1. Dial Up connection
2. Broadband connection
3. Digital Subscriber Line
4. Cable
5. Satelite Connection
6. Wireless Connection
7. Cellular
8. Integrated Service Digital Network

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

1. Fibre Optic Internet: Transmits data as light signals through glass or plastic fibres, offering high-speed internet access.

- **Pros**:

    - **High Speeds**: Offers speeds ranging from 200 Mbps to 2 Gbps, suitable for data-intensive activities like streaming and gaming.
    - **Reliability**: Less susceptible to interference and signal degradation over distance.
    - **Symmetrical Speeds**: Provides equal upload and download speeds, beneficial for video conferencing and cloud services.
- **Cons**:
    - **Limited Availability**: Not widely available in rural or remote areas.
    - **Higher Costs**: Installation and service fees can be higher compared to other connection types.

2. Satellite Internet: Provides internet access via satellites, suitable for remote and rural areas.

- **Pros**:
    - **Wide Coverage**: Available in areas where other connection types are not feasible.
- **Cons**:
    - **High Latency**: Signal travel to and from satellites introduces noticeable delays.
    - **Weather Dependence**: Performance can be affected by adverse weather conditions.
    - **Data Caps**: Often comes with data usage limits, leading to additional charges if exceeded.

3. Broadband over Power Lines (BP): Delivers internet access through existing electrical power lines.

- **Pros**:
    - **Infrastructure Utilization**: Leverages existing power lines, reducing the need for new installations.
- **Cons**:
    - **Limited Availability**: Not widely available and often experimental.
    - **Interference Issues**: Can cause electromagnetic interference with other devices.

4. Mobile Broadband (3G, 4G, 5G): Offers internet access through cellular networks, accessible via smartphones and mobile hotspots.

- **Pros**:
    - **Mobility**: Provides internet access on the go, ideal for travellers and remote workers.
    - **No Fixed Infrastructure**: Does not require physical cables or wires.
- **Cons**:
    - **Data Caps**: Many plans have data limits, with overage fees for exceeding them.
    - **Signal Variability**: Performance can vary based on location, network congestion, and signal strength.

(17). How does broadband differ from fibre-optic internet?

Aspect	Broadband	Fibre-Optic Internet
Definition	General term for high-speed internet	A specific type of broadband using fibre cables
Speed	Varies from 10 Mbps to 1 Gbps	100 Mbps to 10 Gbps or higher
Technology	Includes DSL, cable, satellite, fibre	Exclusively uses fibre-optic cables
Reliability	Depends on the technology (less reliable for satellite and DSL)	Very reliable, with low latency and minimal signal loss
Availability	Available in many urban and suburban areas	Limited availability, mainly in urban areas
Cost	Varies based on technology (cheaper for DSL, satellite)	More expensive, but offers high performance and speed

(18). Protocols

A Protocol is a set of rules and conventions that govern the communication between devices over a network. These rules define how data is transmitted, formatted, and processed to ensure that communication happens efficiently and without errors. Protocols are crucial for enabling different systems and devices to communicate with each other, regardless of their hardware or software.

1. **HTTP (Hypertext Transfer Protocol)**:
   - Used for transferring web pages and other resources over the internet.
   - Example: Accessing websites in a browser.
2. **HTTPS (HTTP Secure)**:
   - A secure version of HTTP that encrypts data using SSL/TLS for safe communication over the internet.
3. **FTP (File Transfer Protocol)**:
   - Used for transferring files between computers over a network.
   - Example: Uploading files to a server.
4. **TCP/IP (Transmission Control Protocol / Internet Protocol)**:
   - Fundamental suite of protocols used for communication over the internet.
   - TCP handles reliable transmission of data, and IP handles addressing and routing.
5. **SMTP (Simple Mail Transfer Protocol)**:
   - Used for sending and receiving emails over the internet.
6. **DNS (Domain Name System)**:
   - Resolves domain names (e.g., www.example.com) to IP addresses, enabling browsers to find servers.
7. **UDP (User Datagram Protocol)**:
   - A connectionless protocol used for quick transmissions, but with no guarantee of delivery (e.g., streaming).
8. **SSH (Secure Shell)**:
   - Used for secure remote login and command execution on servers.

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

HTTP Requests with curl

- **GET Request**:

  bash
  curl http://example.com

- **POST Request**:

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

- **GET Request with Headers**:

  bash
  curl -H "Content-Type: application/json" http://example.com/data

- **Download File**:

  bash
  curl -O http://example.com/file.zip

2. FTP Requests with curl

- **Download File from FTP**:

  bash
  curl -u username:password -O ftp://example.com/path/to/file.zip

- **Upload File to FTP**:

  bash
  curl -u username:password -T localfile.txt ftp://example.com/path/to/destination/

- **List Files on FTP Server**:

  bash
  curl -u username:password ftp://example.com/path/to/directory/

(20). Application Security

Application security refers to the measures and practices designed to protect applications from threats and vulnerabilities throughout their lifecycle. It encompasses techniques, tools, and processes to secure software during development, deployment, and operation, ensuring confidentiality, integrity, and availability of the application and its data. Key aspects include secure coding, vulnerability assessments, authentication, authorization, encryption, and regular updates to defend against attacks like SQL injection, cross-site scripting (XSS), and other exploits.

(21). Identify and explain three common application security vulnerabilities. Suggest possible solutions. Shortly

1. SQL Injection (SQLi)

Issue: Attackers inject malicious SQL queries to access or manipulate databases.
Solution: Use parameterized queries, input validation, and restrict database permissions.

2**.** Cross-Site Scripting (XSS)

Issue: Malicious scripts are injected and executed in users' browsers.
Solution: Sanitize inputs, encode outputs, and implement Content Security Policy (CSP).

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

Issue: Tricks users into performing unauthorized actions on authenticated sites.
Solution: Use anti-CSRF tokens, same-site cookies, and re-authenticate for sensitive actions.

(22). What is the role of encryption in securing applications?

Encryption plays a critical role in securing applications by protecting sensitive data from unauthorized access. It transforms data into an unreadable format using algorithms, ensuring only authorized parties with the correct decryption keys can access it. Here's how encryption contributes to application security:

1. **Data Protection**: Ensures sensitive information like passwords, credit card details, and personal data remain confidential, both at rest and in transit.
2. **Secure Communication**: Encrypts data exchanged between clients and servers (e.g., via SSL/TLS) to prevent eavesdropping or interception.
3. **Compliance**: Helps meet regulatory requirements for data protection, such as GDPR or HIPAA.
4. **Data Integrity**: Verifies that data has not been tampered with during storage or transmission.

(23). Software Applications and Its Types

Software applications are programs or sets of programs designed to perform specific tasks or functions for users. They can be broadly categorized into different types based on their functionality and use. Here are the common types:

1. **Desktop Applications**: Software designed to run on a desktop or laptop computer.
   Ex: Microsoft Word, Adobe Photoshop, VLC Media Player.
2. **Web Applications**: Applications that run on web browsers, accessed over the internet.
   Ex: Google Docs, Facebook, Twitter.
3. **Mobile Applications**: Apps developed specifically for mobile devices like smartphones and tablets.
   Ex: WhatsApp, Instagram, Uber.
4. **Enterprise Applications**: Large-scale software used to manage and support business processes, operations, and data.
   Ex: SAP, Oracle ERP, Salesforce.

5. **Cloud Applications**: Software that runs on cloud infrastructure, often offering scalable and remote access.
   Ex: Dropbox, Google Drive, Office 365.
6. **Game Applications**: Software designed for entertainment and interactive play.
   Ex: Fortnite, Candy Crush, Minecraft.
7. **Embedded Software**: Software designed to operate hardware devices with specific functions.
   Ex: Software in washing machines, smart TVs, car infotainment systems.

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

1. **Windows Operating System** (System Software): It manages hardware and provides essential services for other software applications.
2. **Google Chrome** (Application Software): A web browser used for accessing websites and web-based services.
3. **Microsoft Word** (Application Software): A word processor used for creating and editing documents.
4. **Antivirus Software** (System Software): Provides security by detecting and preventing malware and other threats.
5. **Spotify** (Application Software): A music streaming app for listening to music and podcasts.

Classification:

System Software: Windows Operating System, Antivirus Software

Application Software: Google Chrome, Microsoft Word, Spotify

(25). What is the difference between system software and application software?

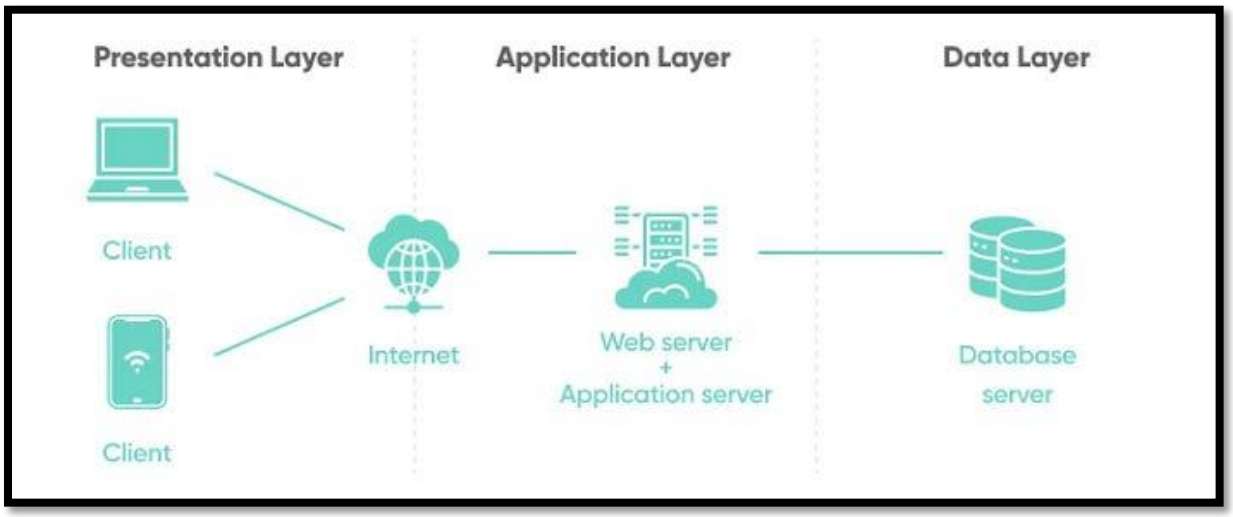
System Software

- **Purpose**: Manages hardware and provides a platform for running application software.
- **Examples**: Operating systems (e.g., Windows, macOS), device drivers, utility programs (e.g., antivirus software).
- **Role**: Acts as an interface between the user and the hardware, ensuring the system operates smoothly. It handles background tasks like memory management, security, and file systems.

Application Software

- **Purpose**: Performs specific tasks or functions for the user.
- **Examples**: Word processors (e.g., Microsoft Word), web browsers (e.g., Google Chrome), media players (e.g., VLC).
- **Role**: Provides tools for the user to carry out tasks like creating documents, browsing the internet, or playing media.

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



(27). What is the significance of modularity in software architecture?

Modularity in software architecture breaks a system into smaller, independent components, each focusing on a specific task. Its significance includes:

1. **Separation of Concerns**: Simplifies development by isolating distinct functionalities.
2. **Reusability**: Promotes the reuse of modules across projects.
3. **Maintainability**: Easier to update and fix modules without affecting others.
4. **Scalability**: New features or modules can be added with minimal impact.
5. **Testing**: Facilitates easier unit testing and debugging.
6. **Collaboration**: Allows multiple developers to work on different modules simultaneously.

(28). Layers in Software Architecture

1.Presentation layer: Manages the user interface and user interaction

2.Application layer: The topmost layer in software architecture that interacts with end-users or other applications. It provides specific services and handles user requests, data processing, and communication between systems

3.Business layer: Handles the core functionality and business rule

4.Persistence layer: The layer manages data storage and retrieval, typically interacting with databases or file systems.

5.Database Layer: that directly interacts with db to manage data storage and retrieval

(29). Create a case study on the functionality of the presentation, business logic, and data access layers of a given software system.

Case Study: Online E-Commerce System

1. Presentation Layer (UI Layer)

Functionality:
The Presentation Layer interacts with the user, displaying the product catalog, managing the shopping cart, and processing user input.

- **Components**: Web and mobile interfaces, client-side JavaScript.
- **Process**: The user browses products, adds items to the cart, and proceeds to checkout. The UI collects inputs (e.g., product selection) and sends requests to the backend.

2. Business Logic Layer (Application Layer)

Functionality:
The Business Logic Layer processes user requests, applies business rules (e.g., calculating totals, handling discounts), and manages the order process.

- **Components**: Shopping cart service, order processing, payment gateway integration.
- **Process**: The layer validates cart data, calculates costs, manages discounts, and processes payments with external services like PayPal.

3. Data Access Layer (Persistence Layer)

Functionality:
The Data Access Layer handles interactions with the database, managing product, user, and order data.

- **Components**: Database (SQL/NoSQL), DAOs, ORM.
- **Process**: Retrieves product details for display, stores order information after checkout, and manages user data such as login credentials.

(30). Why are layers important in software architecture?

Ensure Separation of Concerns: Each layer handles specific responsibilities, reducing complexity.

Improve Maintainability: Changes in one layer don't affect others, making updates easier.

Enhance Scalability: Layers can be independently scaled based on demand.

Promote Reusability: Components in layers can be reused across different projects.

Facilitate Testing: Layers can be tested independently, improving error detection.

Support Flexibility: Layers allow for easier modifications or replacements without disrupting the whole system.

(31). Software Environments

Software Environments are setups that provide the necessary conditions for developing, testing, and running software.

They include:

Development Environment: Used by developers to write and build code (e.g., IDEs, version control).

Testing Environment: For testing the software's functionality and performance before deployment.

Production Environment: The live environment where the software runs for end-users.

Staging Environment: A replica of production, used for final testing before deployment.

Integration Environment: Used to test interactions between different system components.

Local Environment: Developers' personal setup for testing individual features locally.

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

Types of Software Environments:

1. Development Environment:
    - Purpose: Used by developers to write and build code.
    - Components: IDEs, version control (Git), local databases, programming languages (e.g., Python, Java).
    - Example Tools: Visual Studio Code, IntelliJ, Git, MySQL.
2. Testing Environment:
    - Purpose: For testing software to ensure it functions correctly before deployment.
    - Components: Testing frameworks, staging servers, test databases.
    - Example Tools: Selenium, JUnit, Postman.
3. Production Environment:
    - Purpose: The live environment where users interact with the application.
    - Components: Web servers, databases, monitoring tools.
    - Example Tools: Nginx, Apache, AWS, Prometheus.

Steps to Set Up a Basic Environment in a Virtual Machine:

1. Create the VM:
   Use software like VirtualBox or VMware to create a new VM with the desired operating system (e.g., Ubuntu).
2. Install OS:
   Install an OS like Ubuntu from an ISO image on the VM.
3. Set Up Development Environment:
   - Install tools like Git, an IDE (e.g., Visual Studio Code), and a programming language (e.g., Python).
   - Install a local web server (e.g., Apache or Nginx) and a database system (e.g., MySQL).
4. Set Up Testing Environment:
   - Create a new VM or use the same one for testing.
   - Install testing tools (e.g., Selenium, JUnit) and set up a test database.
5. Set Up Production Environment:
   - Install necessary software for production (e.g., Nginx or Apache for web serving, MySQL for databases).
   - Ensure proper security configurations (e.g., firewall, SSL certificates).

(33). Explain the importance of a development environment in software production. Source Code

A development environment is crucial for building, testing, and debugging software efficiently. It provides developers with the necessary tools and resources to write clean, functional code and ensure its quality before deployment.

Key Points*:*

1. **Efficient Code Writing**: Tools like IDEs (e.g., Visual Studio Code) and editors (e.g., Sublime Text) offer features like syntax highlighting and auto-completion to improve productivity.
2. **Debugging and Error Detection**: Built-in debugging tools help identify issues early, making it easier to fix problems before they reach production.
3. **Version Control**: Systems like Git allow for managing code changes, collaboration, and tracking revisions.
4. **Automated Testing**: Integration with testing frameworks (e.g., JUnit, PyTest) ensures that code works as expected through unit and integration tests.
5. **Simulating Production**: Local servers and mock data help simulate production environments to test behaviour before live deployment.
6. **Collaboration**: Developers can work together efficiently with shared tools and version control, ensuring consistency across the project.

A proper development environment streamlines the software creation process, reducing errors and improving overall quality, leading to faster and more reliable deployment.

(34). Write and upload your first source code file to GitHub

1. Set Up Git:

- Install Git.
- Configure your Git settings:

  bash
  git config --global user.name "Your Name"
  git config --global user. Email "your-email@example.com"

2. Create a GitHub Repository:

- Log in to GitHub and create a new repository (e.g., my-first-repo).

3. Write Your First Source Code:

- Create a new file, e.g., hello_world.py:

  python
  print ("Hello, world!")

4. Initialize Git Locally:

- In your project folder, run:

  bash
  git init

5. Add and Commit Your Code:

bash
git add hello_world.py
git commit -m "Initial commit: Add hello world script"

6. Link Repository and Push to GitHub:

bash
git remote add origin https://github.com/your-username/my-first-repo.git
git push -u origin master

7. Verify on GitHub:

- Refresh your repository page to see your uploaded code file.

(35). What is the difference between source code and machine code?

Source Code:

- Written in high-level programming languages (e.g., Python, Java).
- Human-readable and easy to understand.
- Needs to be compiled or interpreted to run on a machine.

Machine Code:

- Written in binary (0s and 1s).
- Directly executed by the computer's CPU.
- Not human-readable and is the lowest-level code.

(36). GitHub and Introductions

GitHub is a platform for hosting and sharing code using **Git**, a distributed version control system. It allows developers to store, track, and collaborate on software projects. GitHub provides both public and private repositories, making it a popular tool for open-source development and team collaboration.

(37). Create a GitHub repository and document how to commit and push code changes.

Step 1: Create a GitHub Repository

1. Log in to GitHub.
2. Click the **+** icon in the top right and select **new repository**.
3. Enter a repository name, choose visibility (public/private), and click **Create repository**.

Step 2: Set Up Git Locally

1. Install Git.
2. Configure Git:

   bash
   git config --global user.name "Your Name"
   git config --global user. Email "your-email@example.com"

Step 3: Clone the Repository

1. Copy the repository URL.
2. Run the following commands in the terminal:

   bash
   git clone https://github.com/your-username/repository-name.git

cd repository-name

Step 4: Make Changes

1. Create or modify a file (e.g., hello_world.py):

   python
   print ("Hello, World!")

Step 5: Commit and Push

1. Stage the changes:

   bash
   Copy code
   git add hello_world.py

2. Commit the changes:

   bash
   Copy code
   git commit -m "Add hello world script"

3. Push to GitHub:

   bash
   Copy code
   git push origin main

Step 6: Verify

- Check your GitHub repository to see the changes

(38). Why is version control important in software development?

Version control is essential in software development because it enables collaboration, tracks code changes, provides a history of revisions, allows for easy rollback, ensures code integrity, facilitates experimentation, and supports automation in CI/CD workflows. It helps teams work efficiently, reduces errors, and ensures reliable software delivery.

(39). Create a student account on GitHub and collaborate on a small project with a classmate.

1. Create a GitHub Account

1. Go to GitHub.
2. Click on **Sign up** in the upper-right corner.
3. Enter your **email**, **username**, and **password**, then follow the instructions to verify your account.

4. GitHub will ask if you're a student. If you're eligible, sign up for the **GitHub Student Developer Pack** to access free tools and resources.
5. Complete your profile setup and confirm your email.

2. Create a New Repository

1. After logging in, click on the **+** icon in the top-right corner and select **new repository**.
2. Give your repository a name (e.g., student-project), choose whether to make it **public** or **private**, and initialize it with a **README** file.
3. Click **Create repository**.

3. Invite Your Classmate to Collaborate

1. Go to your repository page and click on **Settings**.
2. In the left sidebar, click **Collaborators**.
3. Enter your classmate's GitHub username and click **Add collaborator**.
4. Your classmate will receive an invitation to collaborate.

4. Collaborate on the Project

- Your classmate can now clone the repository, make changes, and push updates to it.
- To clone the repo:

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

- Both of you can work on files, create branches, and submit **pull requests** to merge changes.

5. Using Git to Collaborate

- Use **Git** commands to track changes:
  - **Add files**: git add.
  - **Commit changes**: git commit -m "Description of changes"
  - **Push changes**: git push origin main
  - **Pull changes**: git pull origin main

(40). What are the benefits of using GitHub for students?

GitHub benefits students by teaching version control, enabling collaboration, and providing tools for project management. The GitHub Student Developer Pack offers free access to premium resources. Students can build portfolios, host projects, contribute to open source, and gain career-ready skills while networking with developers worldwide.

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

1. System Software

- Operating Systems: Windows, macOS, Linux, Android, iOS
- Device Drivers: Graphics drivers (NVIDIA/AMD), Printer drivers

2. Application Software

- Productivity Tools: Microsoft Office, Google Workspace, Notion, Trello
- Web Browsers: Google Chrome, Mozilla Firefox, Microsoft Edge
- Media Players: VLC Media Player, Spotify
- Design and Development Tools: Adobe Photoshop, Figma, Visual Studio Code, IntelliJ IDEA
- Communication Tools: Zoom, Microsoft Teams, Slack, WhatsApp
- Gaming: Steam, Epic Games Store

3. Utility Software

- Antivirus and Security: Windows Defender, Avast, Norton
- File Management: WinRAR, 7-Zip, File Explorer
- System Maintenance: Cleaner, Disk Clean-up, Task Manager
- Backup Tools: Google Drive, Dropbox, Time Machine
- Network Tools: Wireshark, Ping Plotter

(42). What are the differences between open-source and proprietary software?

Aspect	Open-Source Software	Proprietary Software
Source Code	Source code is freely available for anyone to view, modify, and distribute.	Source code is closed and not accessible to the public.
Cost	Often free, but may involve costs for support or premium features.	Typically requires a purchase or subscription.
Customization	Highly customizable; users can modify the software to suit their needs.	Limited or no ability to modify the software.
Support	Community-driven support, often through forums and documentation.	Official, professional support provided by the company (may require additional cost).
Development	Collaborative, with contributions from a global community of developers.	Controlled solely by the company or organization that owns the software.
Updates	Updates are frequent and driven by the community, but may vary in quality.	Updates are managed by the company, often ensuring consistent quality.
Security	Transparent code allows for public scrutiny, but may rely on the community for patches.	Security is managed by the company, but the closed nature may hide vulnerabilities.

(43). Follow a GIT tutorial to practice cloning, branching, and merging repositories.

1. Clone a Repository

bash
git clone https://github.com/username/repository-name.git
cd repository-name

2. Create a Branch

bash
git checkout -b my-branch

3. Make Changes and Commit

- Edit files and then:

bash
git add.
git commit -m "Your commit message"

4. Push the Branch

bash
git push origin my-branch

5. Create a Pull Request

- Go to GitHub and create a pull request to merge your branch into main.

6. Merge the Branch

- On GitHub, click **Merge pull request** to merge the changes.

7. Update Local Repository

bash
git checkout main
git pull origin main

(44). How does GIT improve collaboration in a software development team?

Git improves collaboration by allowing multiple developers to work on the same project simultaneously without overwriting each other's work. Key benefits include:

1. **Version Control**: Tracks changes and enables reverting to previous versions.
2. **Branching and Merging**: Developers can work on features independently and merge changes seamlessly.
3. **Pull Requests**: Facilitates code review and discussions before merging changes.
4. **Distributed System**: Each developer has a local copy, enabling offline work and later synchronization.
5. **Conflict Resolution**: Helps identify and resolve conflicts when multiple developers modify the same code.

(45). Write a report on the various types of application software and how they improve productivity.

Application software refers to programs designed to perform specific tasks for users. These tools improve efficiency and effectiveness in various fields such as business, education, and personal use.

1. **Productivity Software**

- **How It Improves Productivity**: Provides essential tools for word processing, spreadsheets, and presentations, automating repetitive tasks and improving document formatting, calculations, and data presentation.
- **Ex**: Microsoft Office, Google Workspace

2. Communication Software

- **Examples**: Email clients (Outlook, Gmail), Messaging apps (Slack, Teams), Video conferencing tools (Zoom, Google Meet)
- **How It Improves Productivity**: Facilitates instant communication, collaboration, and remote meetings, reducing time spent on in-person communication and increasing efficiency.

3. Project Management Software

- **Examples**: Trello, Asana, Jira
- **How It Improves Productivity**: Helps teams organize tasks, track progress, and collaborate on projects, ensuring timely completion and better resource management.

4. Accounting and Financial Software

- **Examples**: QuickBooks, Xero, FreshBooks
- **How It Improves Productivity**: Automates invoicing, expense tracking, and financial reporting, saving time on manual calculations and improving financial decision-making.

5. Creative Software

- **Examples**: Adobe Creative Suite, Blender, GarageBand
- **How It Improves Productivity**: Speeds up content creation by providing powerful design, video, and audio editing tools, allowing for high-quality outputs in less time.

6. Database Management Software

- **Examples**: MySQL, Oracle, Microsoft SQL Server
- **How It Improves Productivity**: Organizes and retrieves large volumes of data efficiently, enabling quicker access to accurate information for better decision-making.

7. Development Software

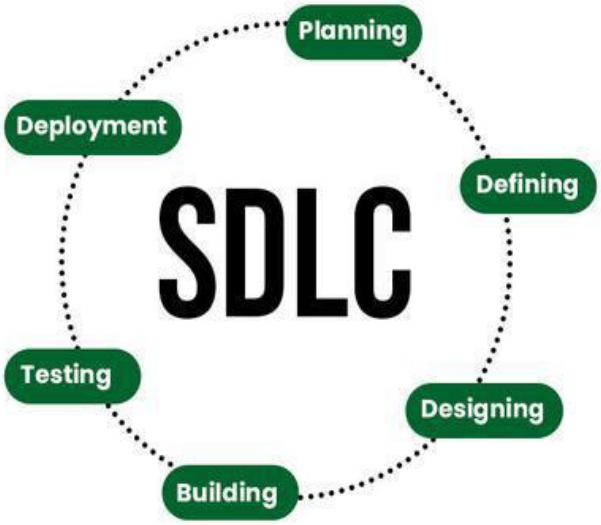
- **Examples**: IDEs (Visual Studio, Eclipse), Git

- **How It Improves Productivity**: Enhances coding, debugging, and version control, streamlining development processes and enabling faster, more collaborative software creation.

(46). What is the role of application software in businesses?

Application software plays a crucial role in businesses by improving efficiency, productivity, and decision-making. It helps automate tasks, streamline operations, manage customer relationships, and facilitate communication and collaboration. Tools like accounting software, project management platforms, CRM systems, and business analytics software allow businesses to manage finances, track projects, analyse data, and improve customer experiences. By enhancing workflow, ensuring accuracy, and enabling data-driven decisions, application software supports business growth, operational efficiency, and competitiveness.

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



(50). What are the main stages of the software development process?

- Planning
- Requirements Analysis
- Design
- Implementation **(**Coding**)**
- Testing
- Deployment
- Maintenance

(51). Write a requirement specification for a simple library management system.

Requirement Specification for Library Management System

1. Overview

The Library Management System (LMS) will help manage books, users (members), and transactions (borrow/return). It will allow staff to add/manage books, and users to search, borrow, and return books.

2. Functional Requirements

- **User Management**:
  - Users can register, log in, and manage profiles.
- **Book Management**:
  - Library staff can add, update, and delete books.
  - Users can search books by title, author, ISBN, and genre.
- **Borrowing and Returning**:
  - Users can borrow and return books. The system will track due dates.
  - Overdue fines should be calculated and displayed.
- **Reports**:
  - Staff can generate reports on book availability, user activity, and overdue books.

3. Non-Functional Requirements

- **Performance**:
  - The system should support 1000+ simultaneous users.
- **Security**:
  - User data should be encrypted and secure.
  - Role-based access control (staff and users).
- **Usability**:
  - The system must have a user-friendly web interface.
- **Availability**:
  - 99.9% uptime with periodic maintenance.

4. System Architecture

- **Frontend**: Web-based interface for users and staff.
- **Backend**: Relational database (e.g., MySQL) for data storage.
- **Security**: Use HTTPS for secure communication.

5. Glossary

- **ISBN**: Unique book identifier.
- **RBAC**: Role-Based Access Control.

(52). Why is the requirement analysis phase critical in software development?

The requirement analysis phase is critical in software development because it helps ensure that the system meets the actual needs of users and stakeholders. By gathering and analysing requirements early, it helps prevent misunderstandings, scope creep, and costly changes later in the project. It sets clear expectations, defines system functionality, and identifies potential

risks, leading to more accurate design, development, and testing phases, ultimately resulting in a successful project.

(53). Perform a functional analysis for an online shopping system.

Functional Analysis for an Online Shopping System

1. User Account Management

- Registration/Login: Users can create an account and log in using credentials.
- Profile Management: Users can update personal details and manage preferences.

2. Product Management

- Product CatLog: Users can browse, search, and filter products by categories, price, and ratings.
- Product Details: Each product has detailed information (description, price, availability).

3. Shopping Cart

- Add/Update Cart: Users can add, modify, or remove products in the cart.
- Cart Summary: Displays total cost, including taxes and shipping fees.

4. Checkout and Payment

- Shipping Information: Users enter/select shipping address.
- Payment Processing: Supports multiple payment methods (credit cards, PayPal).
- Order Confirmation: Users receive confirmation with order details.

5. Order Management

- Order Tracking: Users can track the status of their orders.
- Order History: Users can view past orders and request returns or refunds.

6. Admin Management

- Product/Admin Management: Admins can add, edit, or remove products and manage user accounts.
- Order Management: Admins can process orders and returns.

7. Security

- Data Protection: User data is encrypted and securely stored.
- Authentication: Secure login with optional multi-factor authentication.

(54). What is the role of software analysis in the development process?

Software analysis helps define the system's requirements, ensuring alignment with stakeholder needs. It involves gathering and documenting functional and non-functional requirements, assessing feasibility, identifying risks, and creating models to guide design and development. It ensures the right problem is solved efficiently, providing a clear foundation for the subsequent stages of the development process.

(55). Design a basic system architecture for a food delivery app.

1. **Frontend**:
    - Mobile apps (Customer, Delivery) and web portal (Restaurant, Admin).
2. **Backend**:
    - Microservices: Authentication, Order Management, Payments, Notifications, Delivery Tracking.
3. **Database**:
    - SQL (Users, Orders), NoSQL (Menus, Caching).
4. **Third-Party APIs**:
    - Maps (GPS), Payments, Messaging.
5. **Infrastructure**:
    - Cloud hosting (AWS/GCP), Load balancer, Monitoring tools (Prometheus).

(56). What are the key elements of system design?

The key elements of system design are:

1. **System Requirements**: Define functional and non-functional needs.
2. **Architecture Design**: High-level structure and component interactions.
3. **Data Management**: Database schema, storage, and retrieval optimization.
4. **Scalability**: Handling growth via horizontal/vertical scaling and load balancing.
5. **Security**: Authentication, encryption, and protection against threats.
6. **Fault Tolerance**: Redundancy, error handling, and failover mechanisms.
7. **User Interface**: Focus on usability, responsiveness, and accessibility.
8. **API Design**: Clear, efficient endpoints with versioning and rate limiting.
9. **DevOps and Deployment**: CI/CD pipelines, containerization, and monitoring.
10. **Maintainability**: Modular design, extensibility, and documentation.

(57). Develop test cases for a simple calculator program.

Addition

1. add (2, 3) → Expected: 5
2. add (-2, 3) → Expected: 1
3. add (0, 5) → Expected: 5

Subtraction

1. subtract (5, 3) → Expected: 2
2. subtract (-5, -3) → Expected: -2
3. subtract (0, 5) → Expected: -5

Multiplication

1. multiply (4, 3) → Expected: 12
2. multiply (-4, 3) → Expected: -12
3. multiply (0, 7) → Expected: 0

Division

1. divide (6, 3) → Expected: 2
2. divide (-6, 3) → Expected: -2
3. divide (5, 0) → Expected: Error/Exception

(58). Why is software testing important?

Software testing is important because it ensures the software works as intended, identifies bugs early, improves quality, enhances user experience, prevents costly post-release fixes, ensures security, and meets regulatory standards. It helps deliver a reliable, efficient, and secure product.

- Saves Money
- Security
- Quality of the product
- Satisfaction of the consumer
- Enhancing the development technique
- Easy even as adding new capabilities
- Determining the performance of the software program

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

Case: Healthcare Management System Outage

Problem:

A hospital's Healthcare Management System (HMS) failed during a busy flu season due to a bug in a recent software update. The issue caused database connection problems, making patient records, appointments, and billing inaccessible. This led to missed appointments, delayed claims, and disrupted patient care.

Cause:

The bug was linked to a database query optimization introduced in the update, which caused timeouts and affected system performance.

Solution:

- **Immediate Fix:** A patch was deployed to resolve the query issue.
- **Workaround:** Hospital staff manually handled appointments and records.
- **Post-Incident Review:** The root cause was identified, and improved testing and rollback strategies were implemented.

Outcome:

The system was restored in 48 hours, but the hospital learned the importance of thorough testing and better monitoring to prevent future issues during critical periods.

(60). What types of software maintenance are there?

1. **Corrective Maintenance:** Fixing defects or bugs identified after the software is deployed.
2. **Adaptive Maintenance:** Modifying the software to adapt to changes in the environment, such as updates to operating systems or hardware.
3. **Perfective Maintenance:** Enhancing the software by adding new features or improving performance and usability.
4. **Preventive Maintenance:** Making changes to prevent potential issues in the future, such as improving code quality or refactoring.

(61) What are the key differences between web and desktop applications?

Feature	Web Applications	Desktop Applications
Installation	No installation required; accessed through a web browser.	Requires installation on the local computer.
Platform Dependency	Platform-independent; works on any device with a browser.	Platform-dependent; typically runs on specific OS (Windows, macOS, Linux).
Internet Requirement	Requires an active internet connection.	Can be used offline once installed.
Updates	Updates are automatic and done on the server side.	Updates must be manually downloaded and installed.
Performance	Performance may vary depending on internet speed and server load.	Generally faster as it runs directly on the local machine.
Accessibility	Accessible from any device with an internet connection.	Limited to the device it's installed on.

(62) What are the advantages of using web applications over desktop applications?

1. **Platform Independence:** Web apps can run on any device with a web browser, regardless of the operating system (Windows, macOS, Linux, etc.).
2. **No Installation Required:** Users don't need to install or update web apps, saving time and reducing maintenance complexity.
3. **Automatic Updates:** Web apps are updated automatically, ensuring users always have the latest features and security patches.
4. **Accessibility:** Web apps can be accessed from anywhere with an internet connection, providing greater flexibility and convenience.
5. **Centralized Data Storage:** Data is stored on remote servers, making it easier to manage and back up compared to local desktop storage.
6. **Cross-Device Usage:** Web apps can be accessed on various devices (PCs, tablets, smartphones) with a consistent user experience.
7. **Cost-Effective:** Web apps reduce the need for device-specific software and can be more cost-effective for businesses in terms of maintenance and distribution.
8. **Easier Collaboration:** Web apps facilitate real-time collaboration and sharing, as multiple users can access the app simultaneously from different locations.

(63). What role does UI/UX design play in application development?

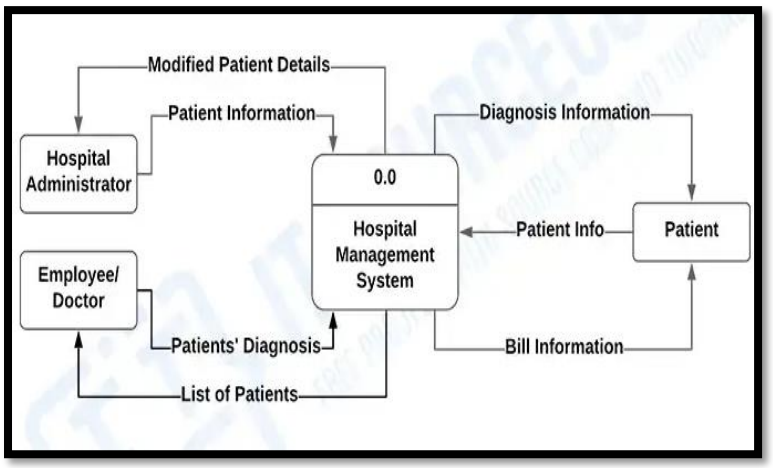
1. **User-Cantered Design:** UI/UX design focuses on understanding the needs, behaviours, and pain points of users, leading to an interface that is intuitive, easy to navigate, and enjoyable to use.
2. **First Impressions:** A well-designed UI creates a positive first impression, encouraging users to engage with the application. An attractive and functional design can improve user retention and satisfaction.
3. **Usability:** UX design ensures that the application is efficient and easy to use, helping user's complete tasks quickly and without frustration. This includes minimizing the number of steps needed for actions and optimizing navigation.
4. **Accessibility:** Good UI/UX design ensures the application is accessible to all users, including those with disabilities, by following accessibility standards and guidelines.
5. **Consistency:** UI/UX design ensures a consistent look and feel across all parts of the application, making it easier for users to learn and use.
6. **Efficiency and Productivity:** By optimizing workflows, UI/UX design can improve user productivity, especially in complex applications, by reducing cognitive load and unnecessary steps.
7. **Brand Identity:** UI/UX design reflects the brand's identity, creating a cohesive experience across platforms that builds trust and familiarity with users.
8. **Reduced Development Costs:** Investing in good UI/UX design early in the development process can reduce the need for costly changes or revisions later by addressing usability issues before they escalate.

(64). What are the differences between native and hybrid mobile apps?

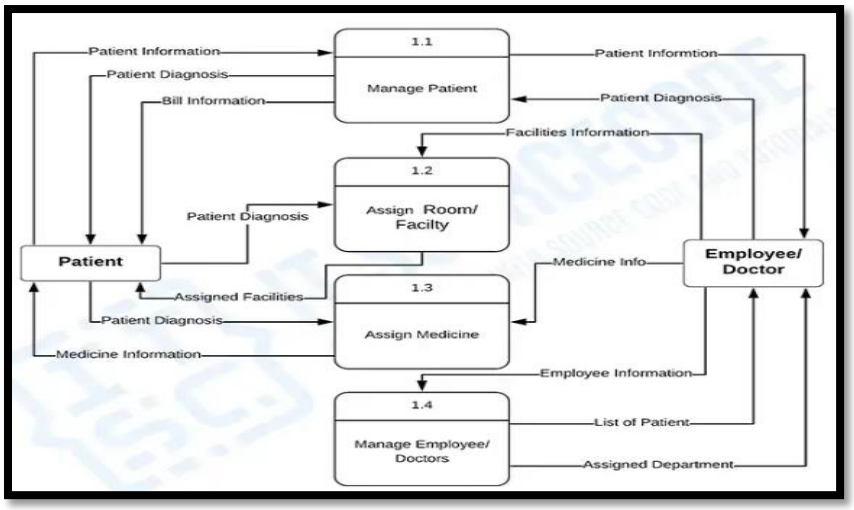
Feature	Native Mobile Apps	Hybrid Mobile Apps
Development Platform	Developed for a specific platform (iOS, Android) using platform-specific languages (Swift, Kotlin).	Developed using web technologies (HTML, CSS, JavaScript) and then wrapped in a native container (e.g., Ionic, React Native).
Performance	Best performance; direct access to device hardware and APIs.	Slightly slower performance due to the web-based nature and wrapper.
User Interface	Custom UI that adheres to the platform's design guidelines.	UI may not be as seamless; can be less responsive than native.
Access to Device Features	Full access to device features like camera, GPS, etc.	Limited access to device features depending on the wrapper and plugins.
Development Time	Longer development time for each platform	Faster development time as one codebase can be used for multiple platforms.
Cost of Development	Higher cost due to separate development for each platform.	Lower cost due to single codebase for both platforms.

(65). Create a DFD for a hospital management system

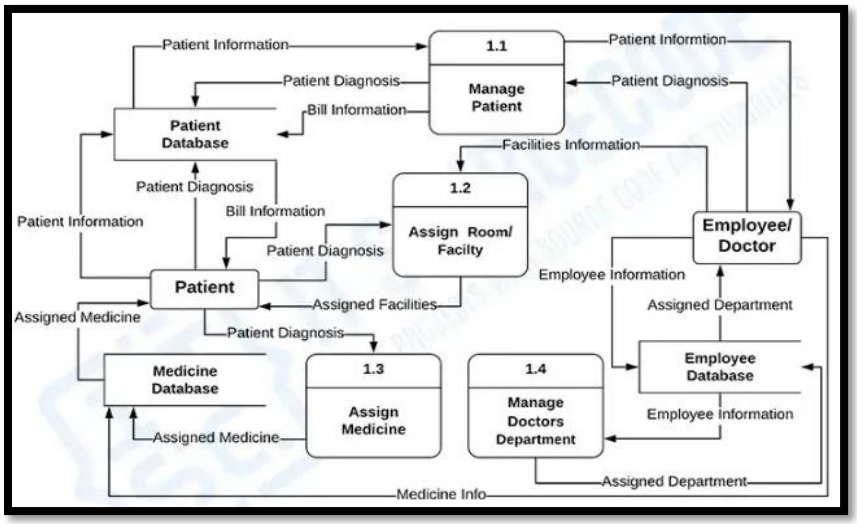
DFD 0:



DFD1:



DFD2:



(66). What is the significance of DFDs in system analysis?

1. **Visualize System Functionality:** DFDs provide a clear and structured way to represent how data moves through a system, helping stakeholders understand the flow of information and system processes.
2. **Simplify Complex Systems:** DFDs break down complex systems into smaller, more manageable components, making it easier for analysts, developers, and non-technical stakeholders to grasp system operations.
3. **Identify Data Sources and Destinations:** DFDs help identify the sources (e.g., users, external systems) and destinations (e.g., databases, output systems) of data, clarifying the inputs and outputs of the system.
4. **Aid in Requirement Gathering:** DFDs help in capturing functional requirements by showing how data is processed and transformed, which assists in identifying necessary system features.

5. **Enhance Communication:** DFDs serve as a communication tool among stakeholders (e.g., developers, project managers, clients), providing a common understanding of the system's structure and data flow.
6. **Detect Redundancies and Bottlenecks:** DFDs highlight inefficiencies, such as redundant data storage or unnecessary data transfers, which can help optimize system design.
7. **Support System Design and Implementation:** By mapping out the flow of data and system processes, DFDs provide the foundation for developing detailed system designs, database structures, and user interfaces.

(67). Build a simple desktop calculator application using a GUI library.

Create a using c program later create using c program

(68). What are the pros and cons of desktop applications compared to web applications?

Pros of Desktop Applications:

- **Performance**: Faster and more responsive as they run locally.
- **Offline Access**: Can be used without an internet connection.
- **System Integration**: Better access to local hardware and system features.
- **Customization**: Greater control over design and functionality.

Cons of Desktop Applications:

- **Platform Dependency**: Requires separate versions for different operating systems.
- **Installation & Updates**: Needs manual installation and updates.
- **Limited Accessibility**: Only available on the device it's installed on.
- **Resource Usage**: Can consume more system resources.

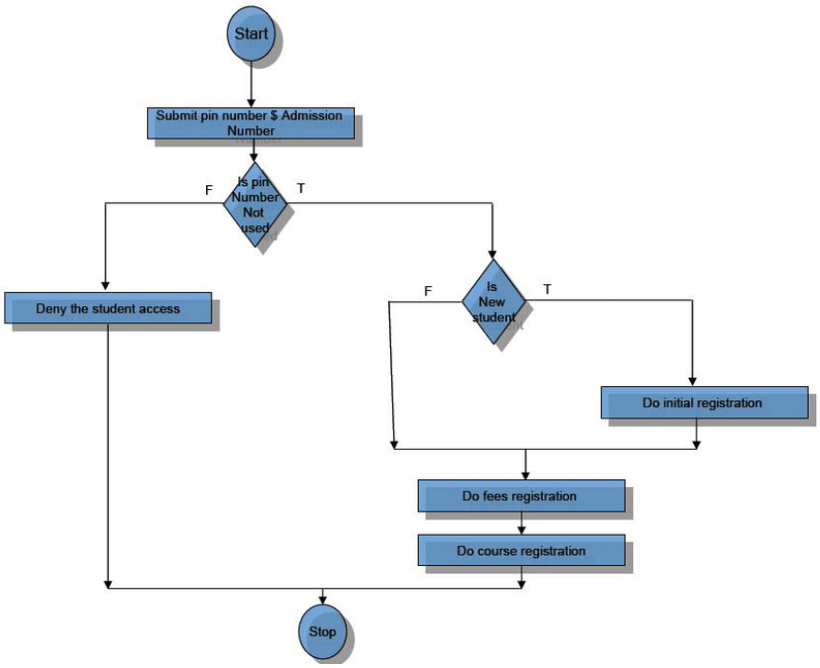
Pros of Web Applications:

- **Accessibility**: Accessible from any device with a browser and internet.
- **Cross-Platform**: Works on multiple operating systems.
- **Automatic Updates**: Always up-to-date without user intervention.
- **Collaboration**: Supports real-time collaboration and sharing.

Cons of Web Applications:

- **Internet Dependency**: Requires an internet connection to function.
- **Performance**: Slower than desktop apps due to network dependency.
- **Limited System Access**: Less integration with local system features.
- **Security Risks**: More vulnerable to online threats.

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



(70). How do flowcharts help in programming and system design?

1. **Visualizing Processes**: They provide a clear, graphical representation of algorithms and processes, making it easier to understand the flow of control and logic.
2. **Simplifying Complex Problems**: Breaking down complex systems into simpler steps, helping in problem-solving and identifying potential issues early.
3. **Enhancing Communication**: Offering a common visual language that developers, designers, and stakeholders can use to communicate ideas.
4. **Improving Debugging and Maintenance**: By outlining the system's structure and logic, flowcharts make it easier to spot errors and optimize or modify the design later.
5. **Documentation**: Serving as useful documentation for future reference and system updates.