

# Module 1 – Overview of IT Industry

## (Practical Exercises)

### 1. Write a simple 'Hello World' program in two different programming languages of your choice. Compare the structure and syntax.

#### i. Python

Python is known for its readability and simplicity.

- Program :-

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

#### Java

Java is a more structured, object-oriented language.

- Program :-

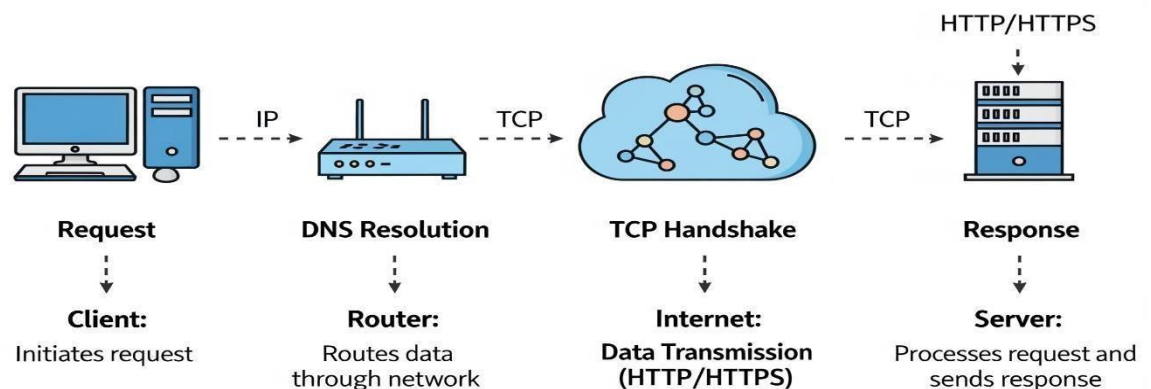
```
class HelloWorld {    public static void        main(String[] args) {        System.out.println("Hello, World!");    }    }
```

#### ✚ Comparison of Structure and Syntax

Feature	Python	Java
<b>Verbosity</b>	<b>Minimal.</b> A single, direct command is all that's needed. This makes it very easy for beginners to start with.	<b>Verbose.</b> The "Hello, World!" logic is wrapped inside a main method, which itself is inside a HelloWorld class. This is called "boilerplate" code
Feature	Python	Java

<b>Structure</b>	<b>Flexible.</b> This example is a simple script. Code doesn't need to be inside a class.	<b>Strictly Object-Oriented.</b> All code in Java must reside within a class. The program's execution starts from the public static void main method.
------------------	---	---

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

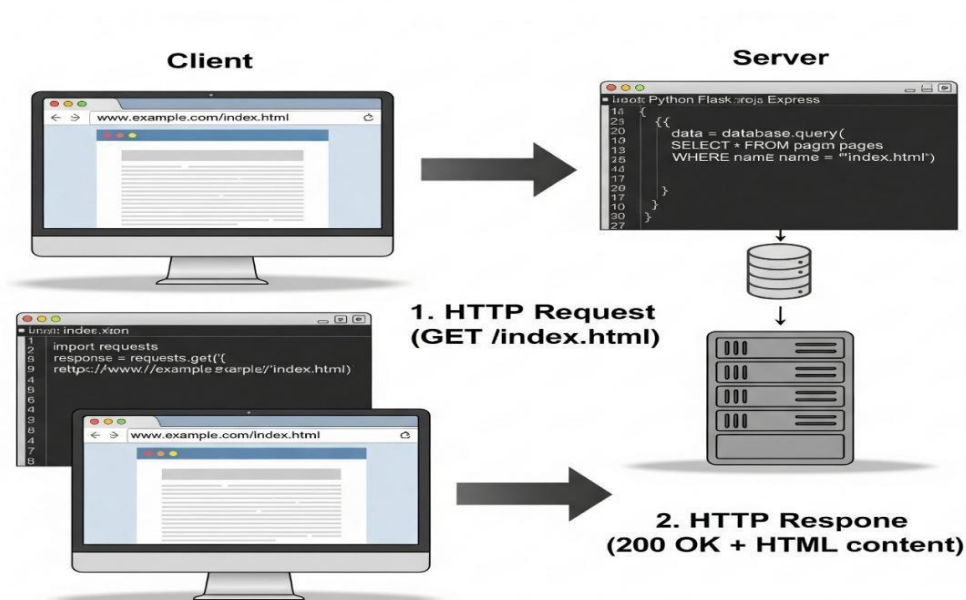


### ✚ How Data Travels Across the Internet :-

- The Request: It starts on your device, the client. When you want to go to a website, your device sends a request for that site's data.

- ii. Finding the Destination (DNS): Your request goes to a Domain Name System (DNS) server. The DNS is like the internet's phonebook; it looks up the website's name (like <https://www.google.com/search?q=google.com>) and finds its numerical IP address.
- iii. The Journey:
  - Once it has the IP address, your request is broken into small pieces called packets.
  - These packets travel from your local router onto the internet.
  - IP (Internet Protocol) addresses the packets to the correct server.
  - TCP (Transmission Control Protocol) makes sure all packets arrive in the right order.
- iv. The Server's Response:
  - The packets arrive at the destination server, a powerful computer holding the website's data.
  - The server processes your request and sends the data back to you in a new set of packets as a response. This whole process is usually handled by HTTP or the secure HTTPS protocol.

### 3. Design a simple HTTP client-server communication in any language.



- Client Sends Request: The client (a web browser or code) sends an HTTP Request to the server to ask for a specific webpage (e.g., `index.html`).

- **Server Processes Request:** The server receives the request, finds the necessary information (sometimes from a database), and prepares to send it back.
- **Server Sends Response:** The server sends back an HTTP Response, which includes a success code (200 OK) and the webpage's HTML content.
- **Client Displays Content:** The client's browser receives the HTML and renders it into the visual webpage you see on the screen.

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

##### **i. Dial-up (PSTN) Pros:**

- Very low cost.
- Uses existing telephone lines.
- Easy to set up (historically).
- Cons:
- Very slow (max ~56 kbps).
- Blocks the phone line during use.

- High latency and outdated.
- ii. DSL (Digital Subscriber Line) Pros:
- Always-on connection.
- Uses existing telephone wiring.
- Affordable for basic browsing.

##### **Cons:**

- Speed drops with distance from provider.
- Lower upload speeds than download.
- Less future-proof than fiber.

##### **iii. Cable Broadband Pros:**

- High download speeds.
- Widely available in cities and towns.
- Reliable for home and office use.

##### **Cons:**

- Shared connection — speed may drop during peak hours.
- Higher latency than fiber.
- Upload speed is usually slower than download iv. Fiber to the Home

(FTTH) Pros:

- Very high speeds (up to multiple Gbps).
- Equal upload and download speeds possible.
- Low latency and highly reliable.
- Future-proof for growing internet needs.

Cons:

- Not available everywhere.
- Installation can be costly.
- Slower rollout in rural areas.

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

```

rudray@lab-pc:~$ curl https://example.com
<!doctype html>
<html>
<head>
  <title>Example Domain</title>
  <meta charset="utf-8" />
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
</head>
<body>
<div>
  <h1>Example Domain</h1>
  <p>This domain is for use in illustrative examples in documents.</p>
</div>
</body>
</html>

rudray@lab-pc:~$ curl -I https://example.com
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 648
Connection: keep-alive
Date: Tue, 12 Aug 2025 07:50:00 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
ETag: "3147526947"
Accept-Ranges: bytes

rudray@lab-pc:~$ curl ftp://speedtest.tele2.net/1MB.zip -o 1MB.zip
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             0      0  512k      0  0:00:02  0:00:02 --:--:--  600k
rudray@lab-pc:~$

```

## 6. Identify and explain three common application security vulnerabilities. Suggest possible solutions.

### SQL Injection (SQLi)

This vulnerability allows attackers to inject malicious SQL code into input fields. This can bypass authentication and expose, modify, or delete data from the database.

**Solution:** Use prepared statements or parameterized queries to separate SQL code from user input. Always validate and sanitize all user-provided data.

### Cross-Site Scripting (XSS)

XSS involves injecting malicious scripts (e.g., JavaScript) into web pages. When other users view the page, their browsers execute the script, which can lead to session hijacking, data theft, or website defacement.

**Solution:** Output encode all user data before displaying it. Implement a Content Security Policy (CSP) to restrict which scripts are allowed to run on your site.

## **Broken Authentication**

This refers to weak or improperly configured authentication and session management systems. Flaws can allow attackers to compromise passwords, impersonate users, or gain unauthorized access.

**Solution:** Enforce strong password policies and use multi-factor authentication (MFA). Implement secure session management by generating random, unpredictable session IDs and invalidating them after logout. Protect against brute-force attacks by limiting failed login attempts.

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

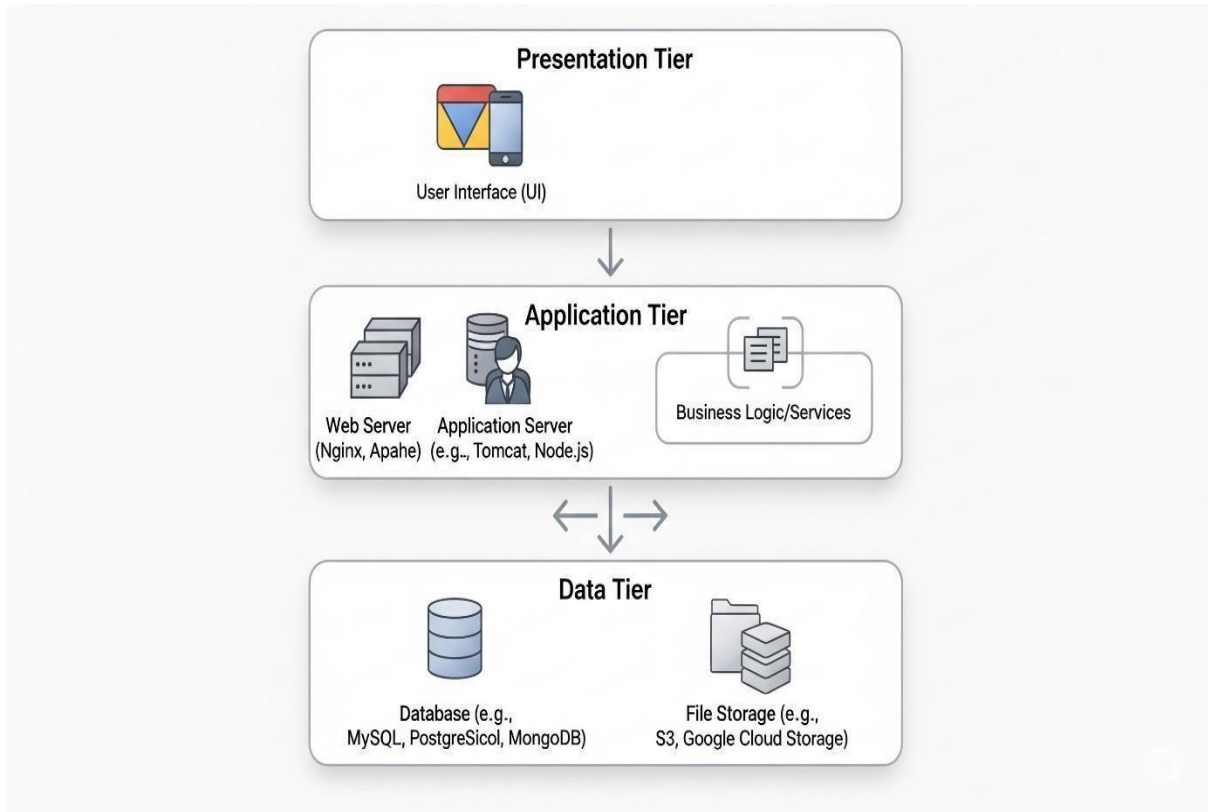
### **System Software**

- i. Android OS: This is an operating system that runs on many smartphones and tablets. It manages all the device's resources, including the screen, processor, and memory, and allows you to install and run applications.
- ii. Microsoft Windows: A popular operating system for desktop and laptop computers. It provides the core environment for everything from managing files to running games and productivity software.

### **Application Software**

- i. Google Chrome: A web browser used for navigating the internet. Its sole purpose is to allow a user to view websites and interact with web-based content.
- ii. Microsoft Word: A word processor used for creating, editing, and formatting documents. It's a prime example of a productivity application that helps a user complete a specific task.
- iii. Spotify: A media player application for streaming music and podcasts. It's designed to provide a specific form of entertainment to the end-user.

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



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

**✚ Case Study: Functionality of Presentation, Business Logic, and Data Access Layers 1. Introduction**

Three-tier architecture divides software into:

1. Presentation Layer (PL) – User interface and interaction.
2. Business Logic Layer (BLL) – Core rules and processing.
3. Data Access Layer (DAL) – Database operations.

This case study uses an Online Bookstore System to explain each layer.



## **2. Layer Functions A. Presentation Layer (PL)**

- Role: Displays information and collects user inputs.
- Functions: Show book lists, search/filter books, display cart, validate forms.
- Example: User searches "Science Fiction," PL sends request to BLL and displays results.

## **B. Business Logic Layer (BLL)**

- Role: Processes requests, enforces rules, coordinates data flow.
- Functions: Validate stock, apply discounts, handle order logic.
- Example: On "Place Order," BLL checks stock, applies discounts, and sends order to DAL.

## **C. Data Access Layer (DAL)**

- Role: Manages database operations securely.
- Functions: CRUD operations, parameterized queries, data mapping.
- Example: Fetch all books in a category using SQL and return to BLL.

## **3. Data Flow Example: Order Placement**

1. PL: Sends order request to BLL.
2. BLL: Validates and processes order → Sends to DAL.
3. DAL: Saves to database → Confirms to BLL → PL displays success message.

## **4. Benefits**

- Scalability
- Easier maintenance
- Improved security

## **5. Conclusion**

Separating PL, BLL, and DAL ensures a structured, maintainable, and scalable system. In the Online Bookstore, each layer handles specific tasks, creating a robust and efficient application.

## **10. Explore different types of software environments**

(development, testing, production). Set up a basic environment in a virtual machine.

### ✚ Types of Software Environment :-

#### i. Development Environment

- Used by programmers to write and debug code.
- Contains IDEs, compilers, version control tools, and dependencies.

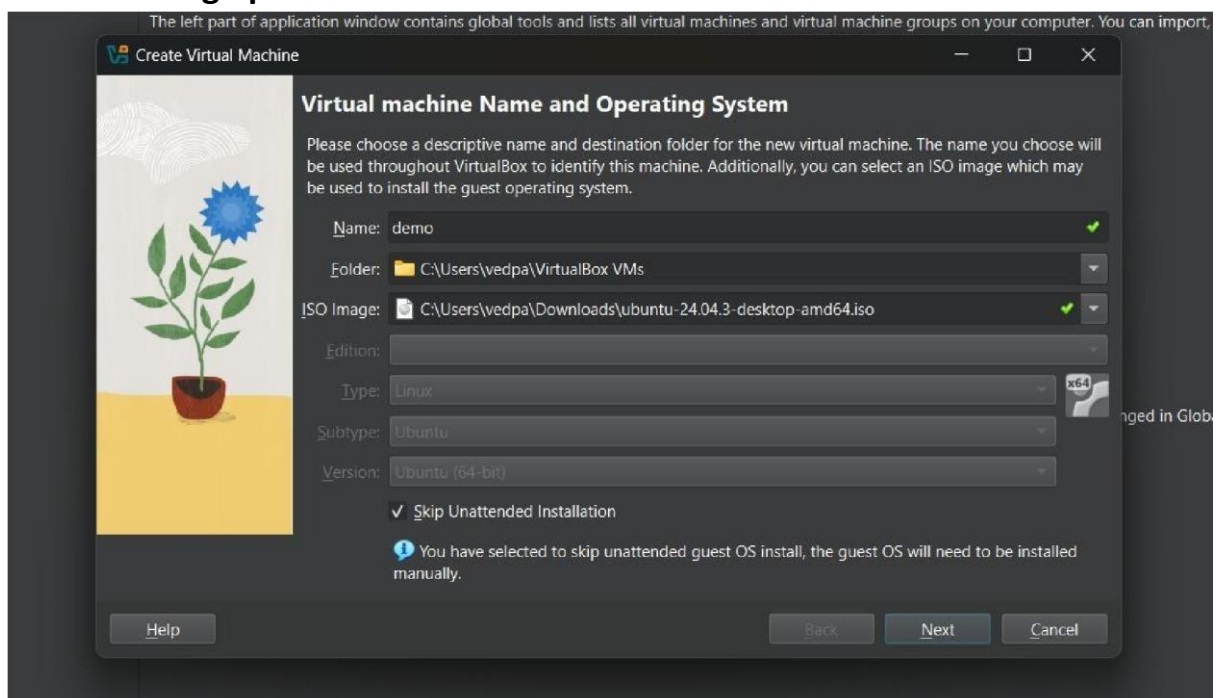
#### ii. Testing Environment

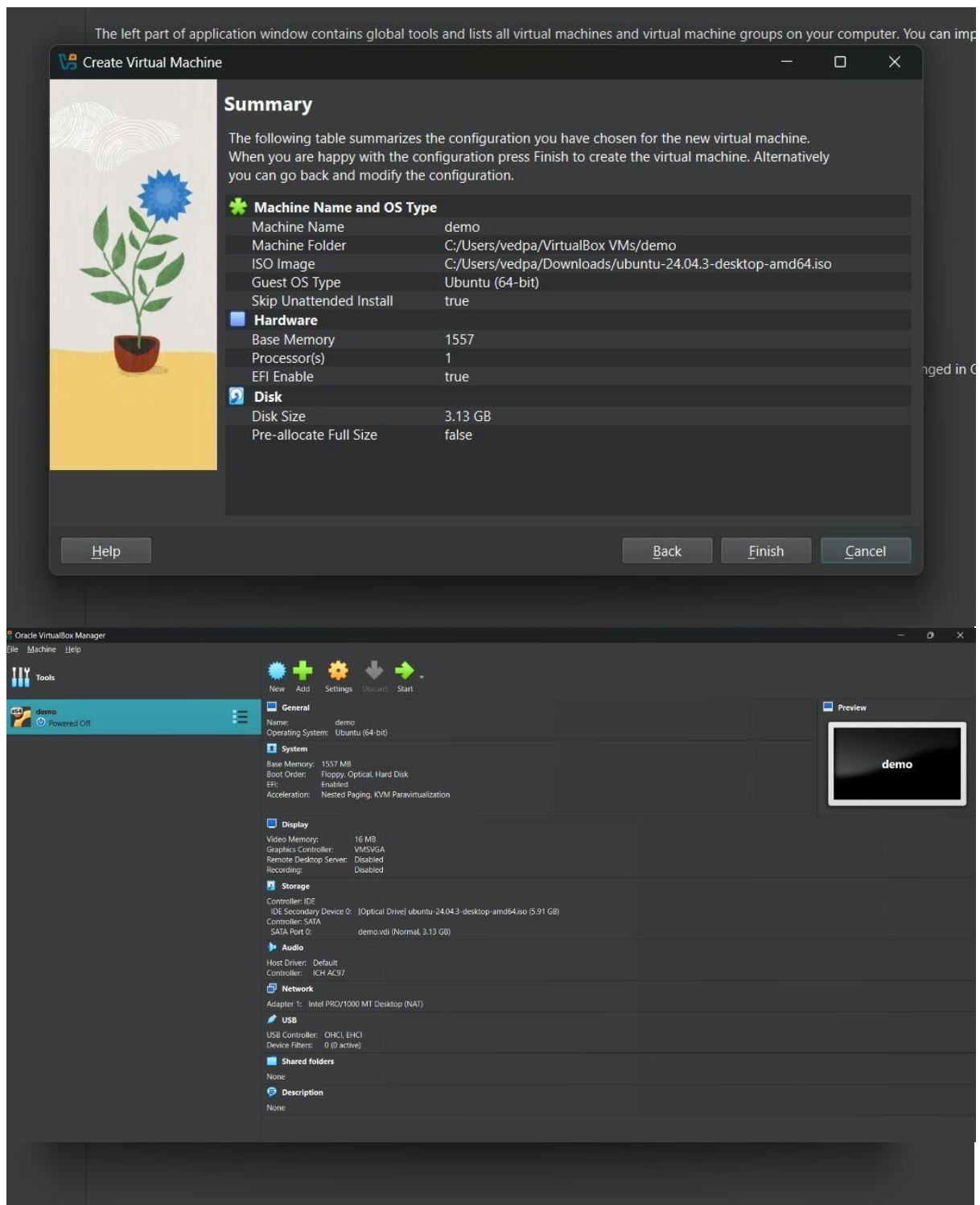
- Used for quality assurance and bug checking.
- Simulates production with test data.

#### iii. Production Environment

- The live system where real users interact.
- Requires high stability, performance, and security.

### ✚ Setting up a Basic Environment in a Virtual Machine :-






**11. Write and upload your first source code file to Github.**

**○ Creating a New Repository :-**

1

General


Owner \*

 nitesh5034

 / 

Repository name \*

my\_first\_code

 my\_first\_code is available.

Great repository names are short and memorable. How about **curly-funicular**?

Description


0 / 350 characters

2

Configuration

Choose visibility \*

Choose who can see and commit to this repository

 Public

Add README

READMEs can be used as longer descriptions. [About READMEs](#)

On ☒

Add .gitignore

.gitignore tells git which files not to track. [About ignoring files](#)

No .gitignore


Add license

Licenses explain how others can use your code. [About licenses](#)

No license

Create repository

## ○ Adding Source Code File :-

 nitesh5034 / my\_first\_code

code

Issues

Pull requests

1

Actions

Projects

Wiki

Security

Insights

Settings

Files


292c00

to file

README.md

hello.c


my\_first\_code / hello.c

 nitesh5034 Add files via upload

Code

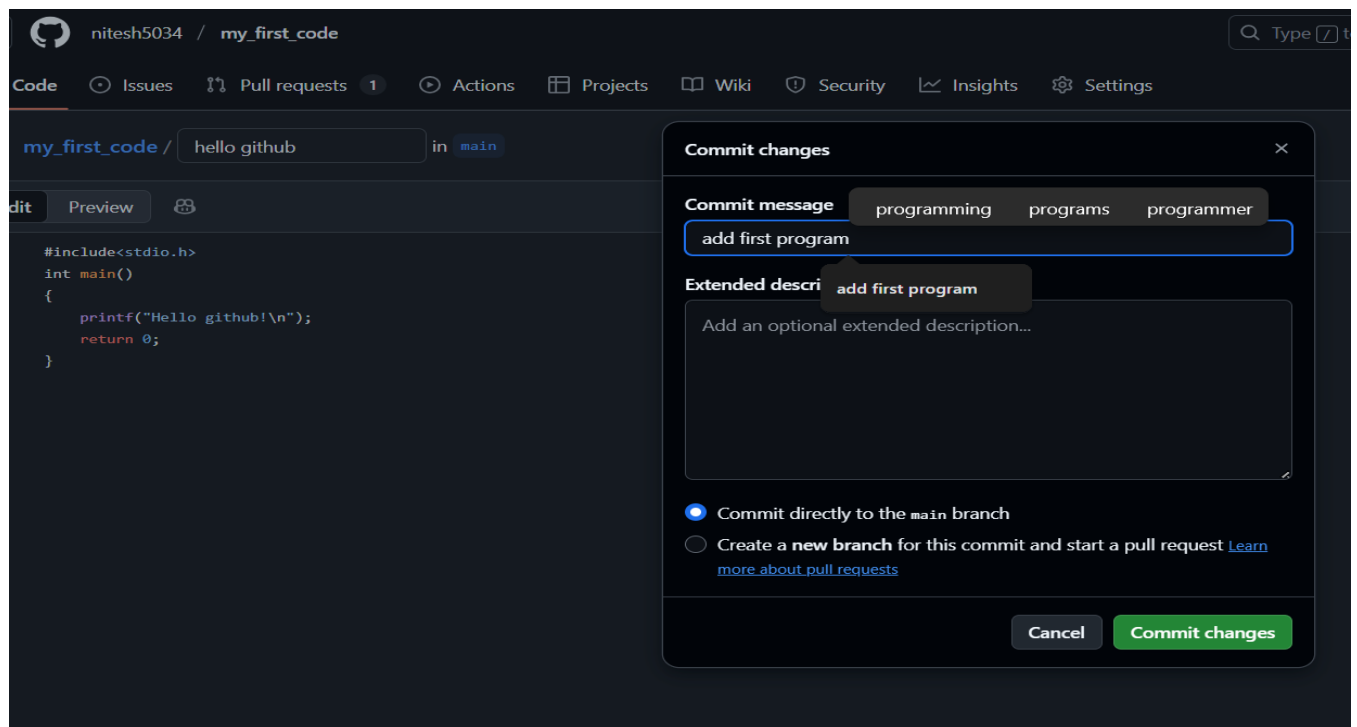
Blame

6 lines (6 loc) · 82 Bytes



```
1  #include<stdio.h>
2  int main()
3  {
4      printf("Hello github!\n");
5      return 0;
6  }
```

## ○ Committing new file :-




## 12. Create a Github repository and document how to commit and push code changes.

### ○ Creating a New Repository :-

1

General

Owner \*

 nitesh5034

Repository name \*

my\_first\_code

✔ my\_first\_code is available.

Great repository names are short and memorable. How about curly-funicular?

Description

0 / 350 characters

2

Configuration

Choose visibility \*

Choose who can see and commit to this repository

Public

Add README

READMEs can be used as longer descriptions. [About READMEs](#)

On ☒

Add .gitignore

.gitignore tells git which files not to track. [About ignoring files](#)

No .gitignore

Add license

Licenses explain how others can use your code. [About licenses](#)

No license

Create repository

## ○ Adding Source Code File :-

nitesh5034 / my\_first\_code

code Issues Pull requests 1 Actions Projects Wiki Security Insights Settings

Files

292c00

to file

README.md

hello.c

my\_first\_code / hello.c

nitesh5034

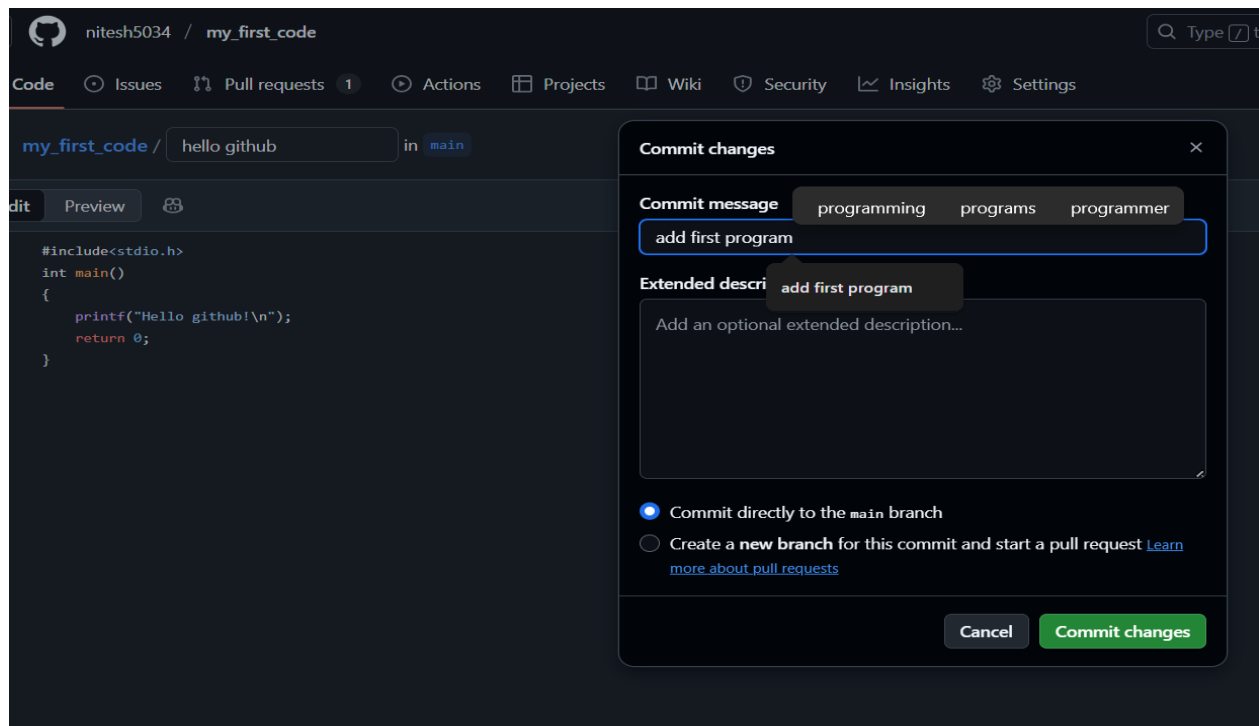
Add files via upload

Code

Blame6 lines (6 loc) · 82 Bytes

```
1  #include<stdio.h>
2  int main()
3  {
4      printf("Hello github!\n");
5      return 0;
6  }
```

## ○ Committing new file :-




**13. Create a student account on Github and collaborate on a small project with a classmate.**

✚ **Github Repository (Project File) :-**

1

General


Owner \*

 nitesh5034

 / 

Repository name \*

my\_first\_code

 my\_first\_code is available.

Great repository names are short and memorable. How about **curly-funicular**?

Description


0 / 350 characters

2

Configuration

Choose visibility \*

Choose who can see and commit to this repository

 Public

Add README

READMEs can be used as longer descriptions. [About READMEs](#)

On ☒

Add .gitignore

.gitignore tells git which files not to track. [About ignoring files](#)


No .gitignore

Add license

Licenses explain how others can use your code. [About licenses](#)

No license

Create repository

 nitesh5034 / my\_first\_code

code

Issues

Pull requests

1

Actions

Projects

Wiki

Security

Insights

Settings

Files


292c00

to file

README.md

hello.c

my\_first\_code / hello.c


 nitesh5034

 Add files via upload

Code

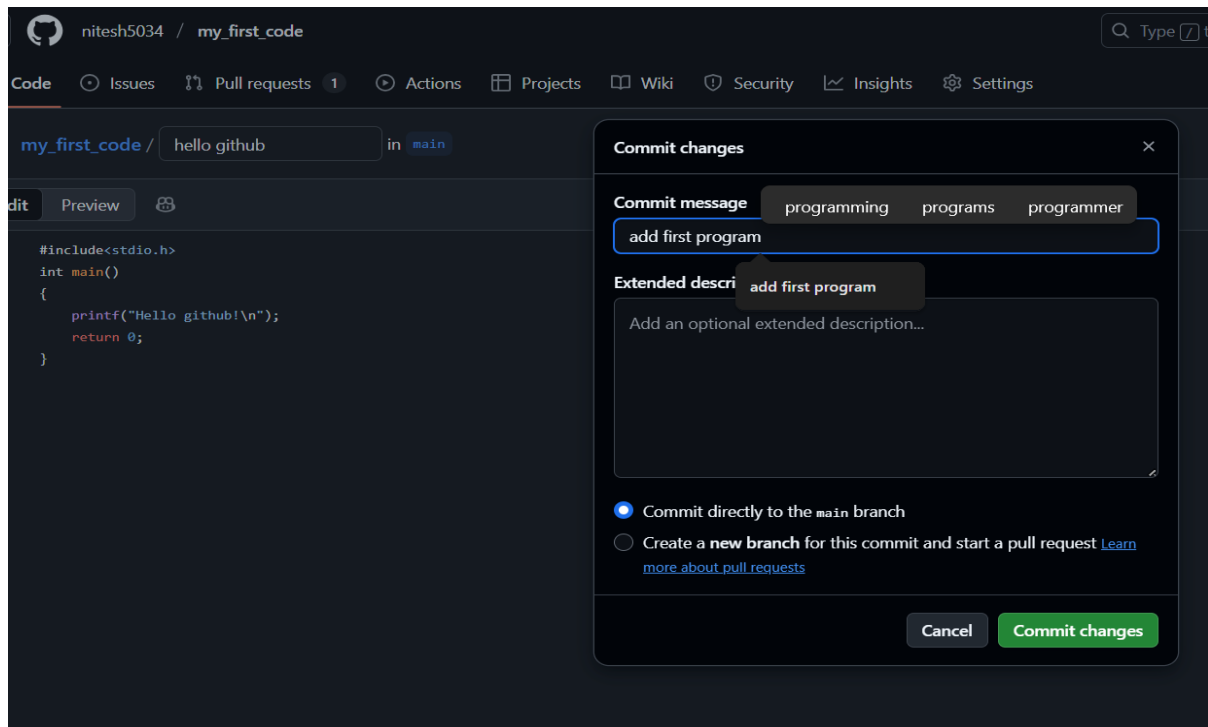
Blame

6 lines (6 loc) · 82 Bytes

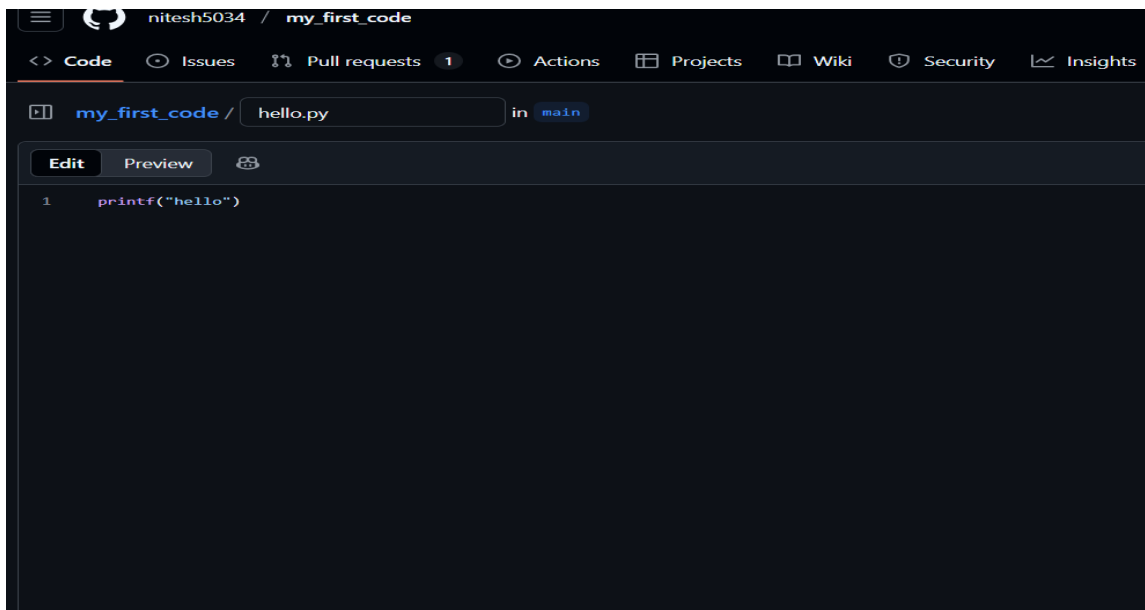


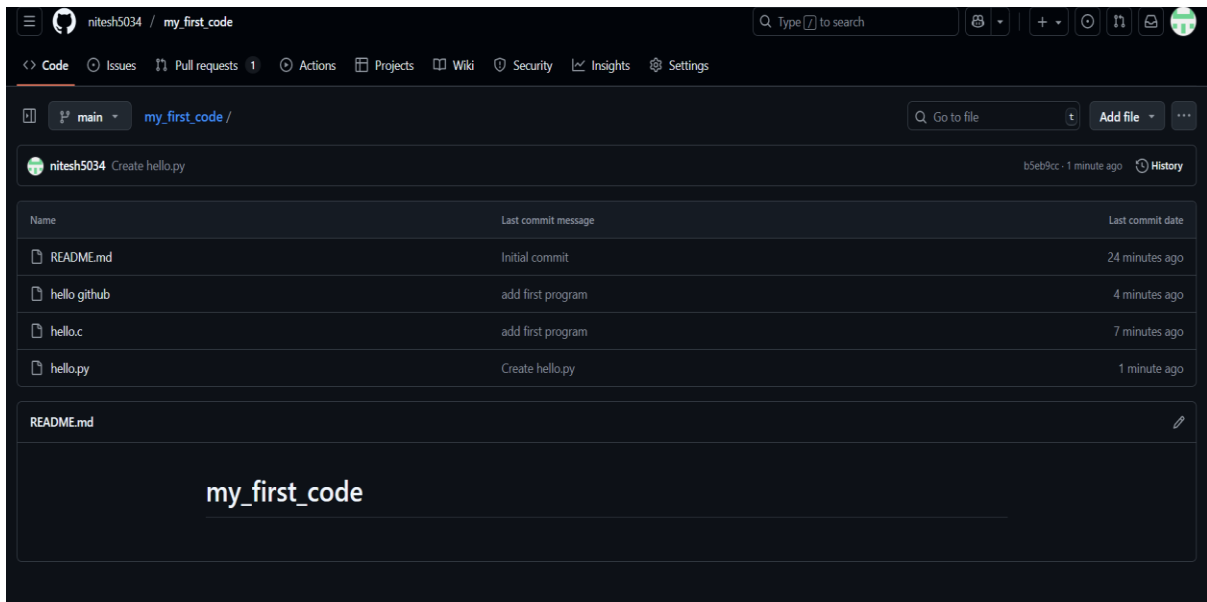
```
1  #include<stdio.h>
2  int main()
3  {
4      printf("Hello github!\n");
5      return 0;
6  }
```





## ✚ Collaborator's work (Classmate) :-





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

### System Software

- i. Windows 10 Operating System ii. Device Drivers iii. BIOS/UEFI Firmware iv. iOS – Apple’s mobile operating system for iPhones and iPads.

### Application Software

- i. Google Chrome (Web Browser) ii. Microsoft Word iii. YouTube App iv. WhatsApp Desktop

### Utility Software

- i. WinRAR
- ii. Microsoft Defender Antivirus iii. CCleaner iv. 7-Zip

**15. Follow a GIT tutorial to practice cloning, branching, and merging repositories.**

**○ Cloning a Repository :-**

```

C:\Users\ADMIN\My_First> git branch
* main

C:\Users\ADMIN\My_First> git branch My_First_Code

C:\Users\ADMIN\My_First> git checkout -b My_First_Code
fatal: a branch named 'My_First_Code' already exists

C:\Users\ADMIN\My_First> git checkout -b My_First_Code1
Switched to a new branch 'My_First_Code1'

```

### ○ Branching a Repository :-

```

C:\Users\ADMIN\My_First>git branch
My_First_Code
* My_First_Code1
main

```

### ○ Merging a Repository :-

```

C:\Users\ADMIN\My_First> git checkout main
M README.md
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

C:\Users\ADMIN\My_First> git merge feature-branch
merge: feature-branch - not something we can merge

C:\Users\ADMIN\My_First> git merge My_First_Code
Already up to date.

```

## 16. Write a report on the various types of application software and how they improve productivity.

### ✚ Report on the Various Types of Application Software and How They Improve Productivity :-

#### I. Introduction

Application software refers to computer programs designed to help users perform specific tasks or solve particular problems. Unlike system software, which manages and controls

computer hardware, application software is user-focused. It enables individuals, businesses, and organizations to complete work efficiently, accurately, and creatively. This report discusses three key types of application software and explains how each type contributes to improving productivity.

## **II. Types of Application Software**

### **II.I Word Processing Software**

- **Description:** Used to create, edit, format, and print text documents.
- **Examples:** Microsoft Word, Google Docs.
- **Productivity Benefits:**
  - Speeds up document creation with templates and formatting tools.
  - Enables quick editing and error correction.
  - Facilitates collaboration through track changes and comments.

### **II.II Spreadsheet Software**

- **Description:** Designed to organize, calculate, and analyze data using tables, formulas, and charts.
- **Examples:** Microsoft Excel, Google Sheets.
- **Productivity Benefits:**
  - Automates complex calculations.
  - Provides data visualization through graphs and charts.
  - Simplifies financial and statistical analysis.

### **II.III Presentation Software**

- **Description:** Creates visually appealing slideshows for communication and teaching.
- **Examples:** Microsoft PowerPoint, Google Slides.
- **Productivity Benefits:**
  - Enhances communication of ideas with visuals.
  - Offers templates that save design time.
  - Supports integration of multimedia for engaging presentations.

### **III. How Application Software Improves Productivity**

- I. Automation of Tasks** – Reduces manual effort and saves time.
- II. Accuracy and Consistency** – Minimizes errors in data entry and formatting.
- III. Collaboration** – Enables multiple users to work together in real time.
- IV. Flexibility and Mobility** – Cloud-based apps allow work from anywhere.
- V. Better Decision-Making** – Data and visuals improve clarity and understanding.

### **IV. Conclusion**

The three types of application software discussed—word processing, spreadsheet, and presentation tools—play a significant role in improving productivity across education, business, and personal tasks. They streamline work processes, enhance communication, and support better decision-making. As technology advances, these tools continue to evolve, offering users more features and efficiency in their work.

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

# Software Development Life Cycle (SDLC) Flowchart

```
graph TD; RG[Requirements Gathering] --> D[Design]; D -- "in" --> IC[Implementation / Coding]; IC --> M[Maintenance]; M --> T[Testing]; T --> D; T --> DP{Passes Testing?}; DP -- Yes --> M; DP -- No --> T;
```

The flowchart illustrates the Software Development Life Cycle (SDLC) process, showing the sequence of stages and their associated activities.

**Stages and Activities:**

- Requirements Gathering** (Blue Box)
  - User Stories
  - Use Cases
  - Functional Specs
- Design** (Green Box)
  - Testing Implementation
- Implementation / Coding** (Yellow Box)
  - Implementation
- Maintenance** (Red Box)
  - Repulemation
- Testing** (Orange Box)
  - Unit, Integration
  - System
  - Acceptance
- Deployment** (Purple Box)

**Flow and Decision Points:**

- The process flows sequentially through Requirements Gathering, Design, Implementation / Coding, and Maintenance.
- A feedback loop exists from Testing back to Design.
- From Testing, the process moves to Deployment.
- A decision point, **Passes Testing?**, follows Deployment.
- If the answer is **Yes**, the process continues to Maintenance.
- If the answer is **No**, the process loops back to Testing.

**18. Write a requirement specification for a simple library management system.**

## † Requirement Specification – Simple Library Management System

### Purpose:

A system to manage books, members, and borrowing/returning activities in a library, replacing manual record-keeping. **Scope:**

- Maintain book and member records.
- Track book issue/return with due dates.
- Provide search and reporting features.

### Functional Requirements:

1. User Management: Add/update/delete members, assign IDs.
2. Book Management: Add/update/delete books, track availability.

3. Issue & Return: Record loans, returns, and late fees.
4. Search: Find books by title/author/ISBN; find members by name/ID.
5. Reports: Borrowed books list, overdue list, member history.

#### **Non-Functional Requirements:**

- Fast search ( $\leq 2$  sec).
- Secure admin login.
- Daily backups.
- Simple, user-friendly interface.

#### **System Features:**

- Easy management of members and books.
- Quick issuing/returning process.
- Accurate reporting and secure access.

## **19. Perform a functional analysis for an online shopping system.**

### **✚ Functional Analysis – Online Shopping System**

#### **Objective:**

To enable customers to browse, purchase, and receive products online efficiently. **Main**

#### **Functional Areas:**

##### **i. User Management**

- a. User registration and login.
- b. Profile update and password management.
- c. Address and payment details storage.

##### **ii. Product Management**

- a. Add, update, and delete product listings (admin).
- b. Categorize products (e.g., electronics, clothing).
- c. Display product details with images and descriptions.

**iii. Product Search & Browsing**

- a. Search by name, category, price, or brand.
- b. Apply filters (price range, ratings, discounts).

**iv. Shopping Cart & Wishlist**

- a. Add/remove products to cart.
- b. Save products in wishlist for later purchase.
- c. Update product quantity in cart.

**v. Order Processing**

- a. Checkout with selected payment method (card, UPI, COD).
- b. Apply discount codes or offers.
- c. Generate order ID and confirmation.

**vi. Payment Gateway Integration**

- a. Secure payment processing.
- b. Transaction success/failure notifications.

**vii. Order Tracking & Delivery**

- a. View order status (processing, shipped, delivered).
- b. Track shipment with tracking ID.

**viii. Customer Support**

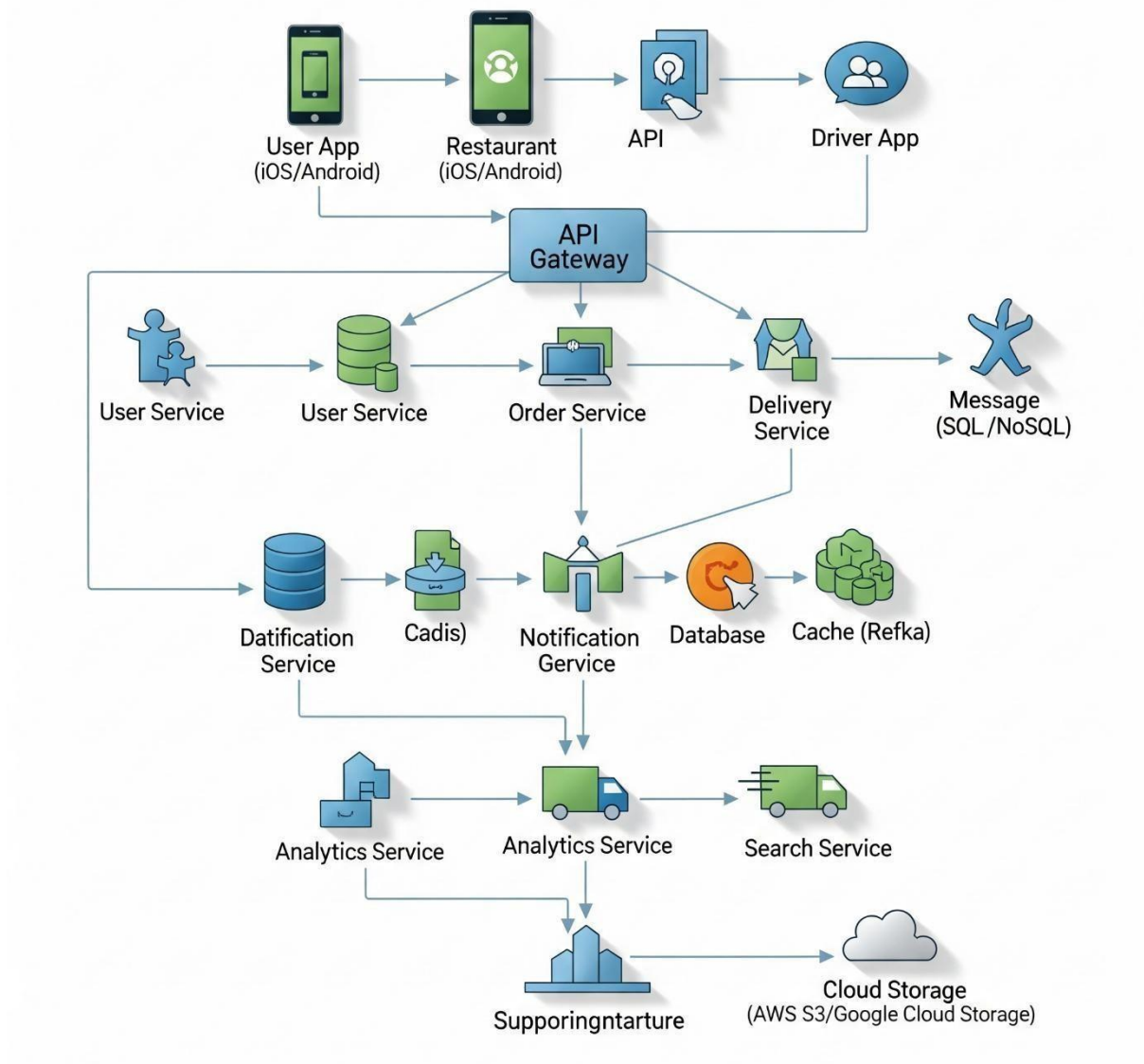
- a. Contact form, live chat, or helpdesk.
- b. Order cancellation or return requests.

**ix. Admin Functions**

- a. Manage products, categories, and stock.
- b. View sales reports and user activity.

**20. Design a basic system architecture for a food delivery app.**





This architecture is designed to be scalable, reliable, and efficient, ensuring a smooth experience for all users. It is broken down into three main components:

### Client-Side (User-Facing Applications)

- **Customer App:** This is the primary interface for users to browse restaurants, place orders, and make payments. It is available on both iOS and Android platforms to ensure a wide reach.
- **Restaurant App:** This application is for restaurant owners and staff to manage their menus, receive and process orders, and track their earnings.
- **Rider App:** This is used by delivery personnel to view and accept delivery requests, navigate to the restaurant and customer locations, and manage their delivery schedule.

## Server-Side (Backend Infrastructure)

- **API Gateway:** This acts as a single entry point for all client requests. It routes requests to the appropriate microservice, ensuring that the system is secure and easy to manage.
- **Microservices:** The backend is built using a microservices architecture, where each service is responsible for a specific function. This makes the system more resilient and easier to scale.
  - **User Service:** Manages user authentication, profiles, and account settings.
  - **Restaurant Service:** Handles restaurant information, menus, and ratings.
  - **Order Service:** Manages the entire order lifecycle, from placement to delivery.
  - **Payment Service:** Integrates with payment gateways to process transactions securely.
  - **Delivery Service:** Assigns delivery requests to riders and tracks their location in real-time.
  - **Notification Service:** Sends real-time notifications to users, restaurants, and riders via push notifications, SMS, or email.

## Data Storage

- **Databases:** The system uses a combination of SQL and NoSQL databases to store different types of data. For example, user and order data might be stored in a SQL database, while restaurant menus and reviews could be stored in a NoSQL database.
- **Cache:** A caching layer (e.g., Redis) is used to store frequently accessed data, reducing latency and improving performance.
- **Cloud Storage:** Media files, such as images of food items and restaurant logos, are stored in a cloud storage service like AWS S3 or Google Cloud Storage.

## 21. Develop test cases for a simple calculator program.

### ✚ Test Cases – Simple Calculator Program

Assumption: Calculator supports addition (+), subtraction (-), multiplication (×), and division (÷) for two numbers.

Test Case ID	Description	Input	Expected Output	Remarks
--------------	-------------	-------	-----------------	---------

TC01	Addition of two positive numbers	$5 + 3$	8	Pass if correct
TC02	Addition with zero	$7 + 0$	7	Pass if correct
TC03	Addition with negative number	$-4 + 6$	2	Pass if correct
TC04	Subtraction of two numbers	$9 - 5$	4	Pass if correct
TC05	Subtraction resulting in negative	$3 - 8$	-5	Pass if correct
TC06	Multiplication of two numbers	$4 \times 7$	28	Pass if correct
TC07	Multiplication with zero	$9 \times 0$	0	Pass if correct
TC08	Division of two numbers	$12 \div 4$	3	Pass if correct
TC09	Division by one	$9 \div 1$	9	Pass if correct
TC10	Division by zero	$5 \div 0$	Error message / Infinity	Must handle safely
TC11	Floating-point addition	$2.5 + 1.2$	3.7	Pass if correct
TC12	Large number multiplication	$100000 \times 1000$	100000000	Pass if correct

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

### ✚ Case Study – Critical Maintenance in WhatsApp (2020 Global Outage)

Background:

On January 19, 2020, WhatsApp experienced a major outage that prevented millions of users worldwide from sending or receiving messages for approximately 2 hours.

**Issue:**

A configuration error during a routine server update caused a mismatch between the application layer and database synchronization service. This led to message delivery failures and connection drops.

**Impact:**

- Affected over 2 billion users globally.
- Disrupted personal and business communications.
- Social media platforms saw a spike in complaints and outage reports.

**Maintenance Action Taken:**

- The engineering team rolled back the faulty configuration to the previous stable version.
- Performed emergency database synchronization.
- Increased monitoring alerts to detect similar issues earlier.

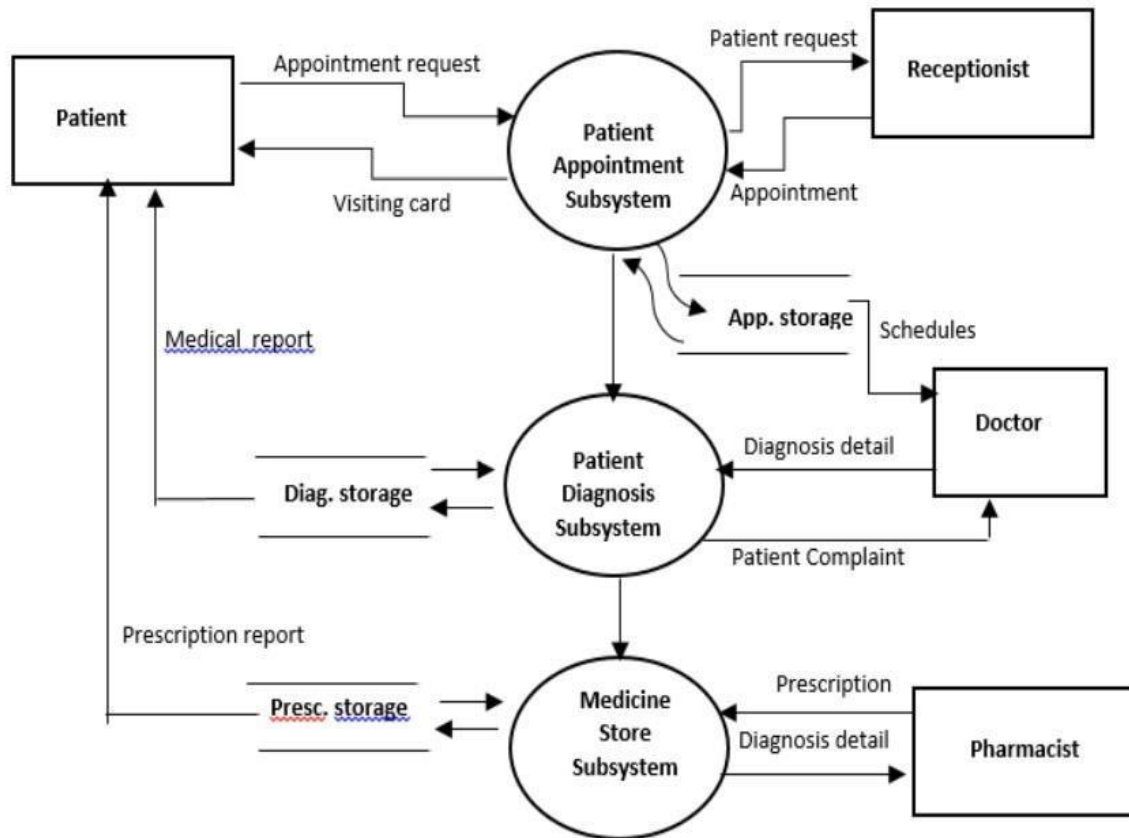
**Outcome:**

- Service was restored in under 2 hours.
- A post-mortem analysis identified a need for better pre-deployment testing in a staging environment.
- WhatsApp introduced automated rollback mechanisms to minimize downtime in future incidents.

**Lesson Learned:**

Critical maintenance requires quick fault detection, immediate rollback plans, and improved testing procedures to prevent repeat failures.

## **23. Create a DFD for a hospital management system.**

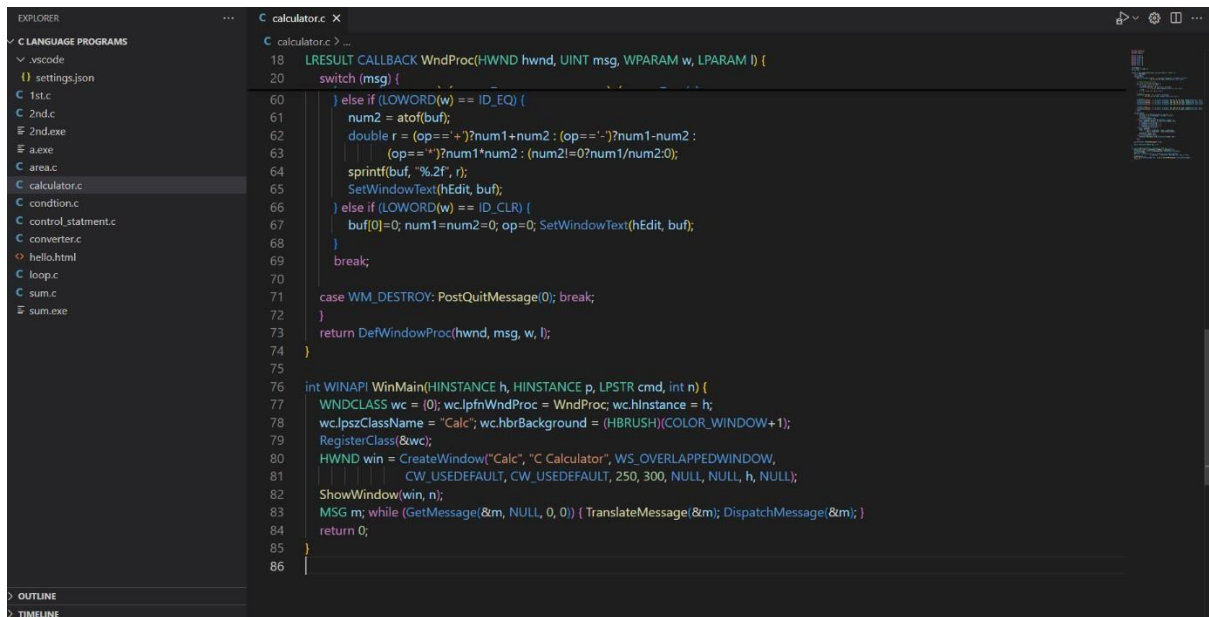


**24. Build a simple desktop calculator application using a GUI library.**

```
EXPLORER
C LANGUAGE PROGRAMS
  .vscode
  settings.json
  1st.c
  2nd.c
  2nd.exe
  a.exe
  area.c
  calculator.c
  condition.c
  control_statement.c
  converter.c
  hello.html
  loop.c
  sum.c
  sum.exe

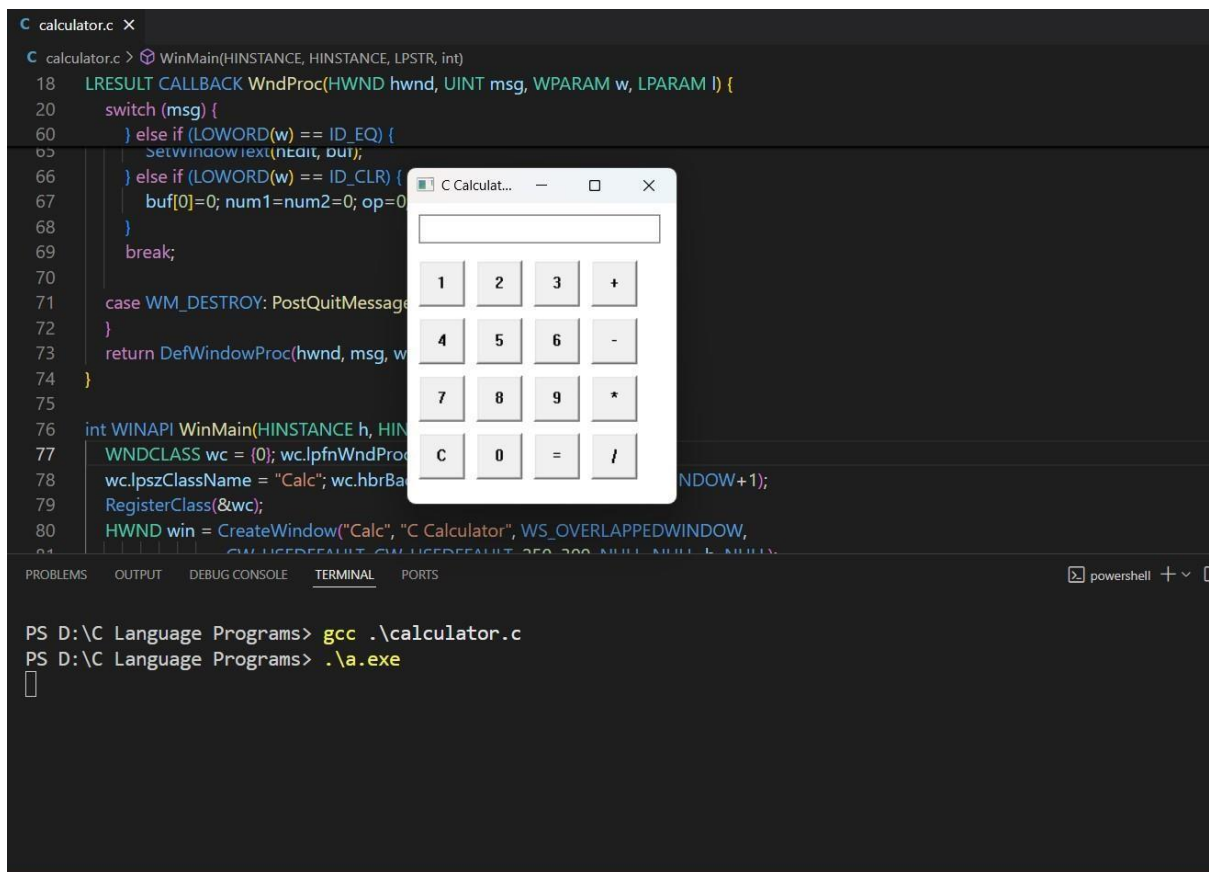
calculator.c
1 #include <windows.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4
5 #define ID_EDIT 1
6 #define ID_NUM0 10
7 #define ID_ADD 20
8 #define ID_SUB 21
9 #define ID_MUL 22
10 #define ID_DIV 23
11 #define ID_EQ 24
12 #define ID_CLR 25
13
14 char buf[64];
15 double num1 = 0, num2 = 0;
16 char op = 0;
17
18 LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM w, LPARAM l) {
19     static HWND hEdit;
20     switch (msg) {
21     case WM_CREATE:
22         hEdit = CreateWindow("EDIT", "", WS_CHILD | WS_VISIBLE | WS_BORDER | ES_RIGHT,
23             10, 10, 210, 25, hwnd, (HMENU)ID_EDIT, NULL, NULL);
24
25         // Create number buttons 1-9
26         int x = 10, y = 50, id = ID_NUM0 + 1;
27         for (int i = 1; i <= 9; i++) {
28             char txt[2] = { i + '0', 0 };
29             CreateWindow("BUTTON", txt, WS_CHILD | WS_VISIBLE,
30                 x, y, 40, 40, hwnd, (HMENU)ID_NUM0 + i, NULL, NULL);
31             x += 50;
32             if (i % 3 == 0) { x = 10; y += 50; }
```

```
calculator.c
18 LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM w, LPARAM l) {
20     switch (msg) {
27         for (int i = 1; i <= 9; i++) {
34             // Button 0
35             CreateWindow("BUTTON", "0", WS_CHILD | WS_VISIBLE,
36                 60, 200, 40, 40, hwnd, (HMENU)ID_NUM0, NULL, NULL);
37
38             // Operators
39             CreateWindow("BUTTON", "+", WS_CHILD | WS_VISIBLE, 160, 50, 40, 40, hwnd, (HMENU)ID_ADD, NULL, NULL);
40             CreateWindow("BUTTON", "-", WS_CHILD | WS_VISIBLE, 160, 100, 40, 40, hwnd, (HMENU)ID_SUB, NULL, NULL);
41             CreateWindow("BUTTON", "*", WS_CHILD | WS_VISIBLE, 160, 150, 40, 40, hwnd, (HMENU)ID_MUL, NULL, NULL);
42             CreateWindow("BUTTON", "/", WS_CHILD | WS_VISIBLE, 160, 200, 40, 40, hwnd, (HMENU)ID_DIV, NULL, NULL);
43
44             // Equal & Clear
45             CreateWindow("BUTTON", "=", WS_CHILD | WS_VISIBLE, 110, 200, 40, 40, hwnd, (HMENU)ID_EQ, NULL, NULL);
46             CreateWindow("BUTTON", "C", WS_CHILD | WS_VISIBLE, 10, 200, 40, 40, hwnd, (HMENU)ID_CLR, NULL, NULL);
47             break;
48
49     case WM_COMMAND:
50         if (LOWORD(w) >= ID_NUM0 && LOWORD(w) <= ID_NUM0 + 9) {
51             char d[2] = { (char)(LOWORD(w) - ID_NUM0 + '0'), 0 };
52             strcat(buf, d);
53             SetWindowText(hEdit, buf);
54         } else if (LOWORD(w) >= ID_ADD && LOWORD(w) <= ID_DIV) {
55             num1 = atof(buf); buf[0] = 0; SetWindowText(hEdit, buf);
56             if (LOWORD(w) == ID_ADD) op = '+';
57             if (LOWORD(w) == ID_SUB) op = '-';
58             if (LOWORD(w) == ID_MUL) op = '*';
59             if (LOWORD(w) == ID_DIV) op = '/';
60         } else if (LOWORD(w) == ID_EQ) {
61             num2 = atof(buf);
```



This screenshot shows the Visual Studio Code editor with the file explorer on the left displaying a project named 'C LANGUAGE PROGRAMS'. The file 'calculator.c' is selected. The main editor window shows the source code of 'calculator.c'. The code includes a Windows API header, defines a window class 'Calc', registers it, creates a window, and implements a message loop. The 'WndProc' function handles messages for adding, subtracting, multiplying, and dividing two numbers, as well as clearing the display and quitting the application.

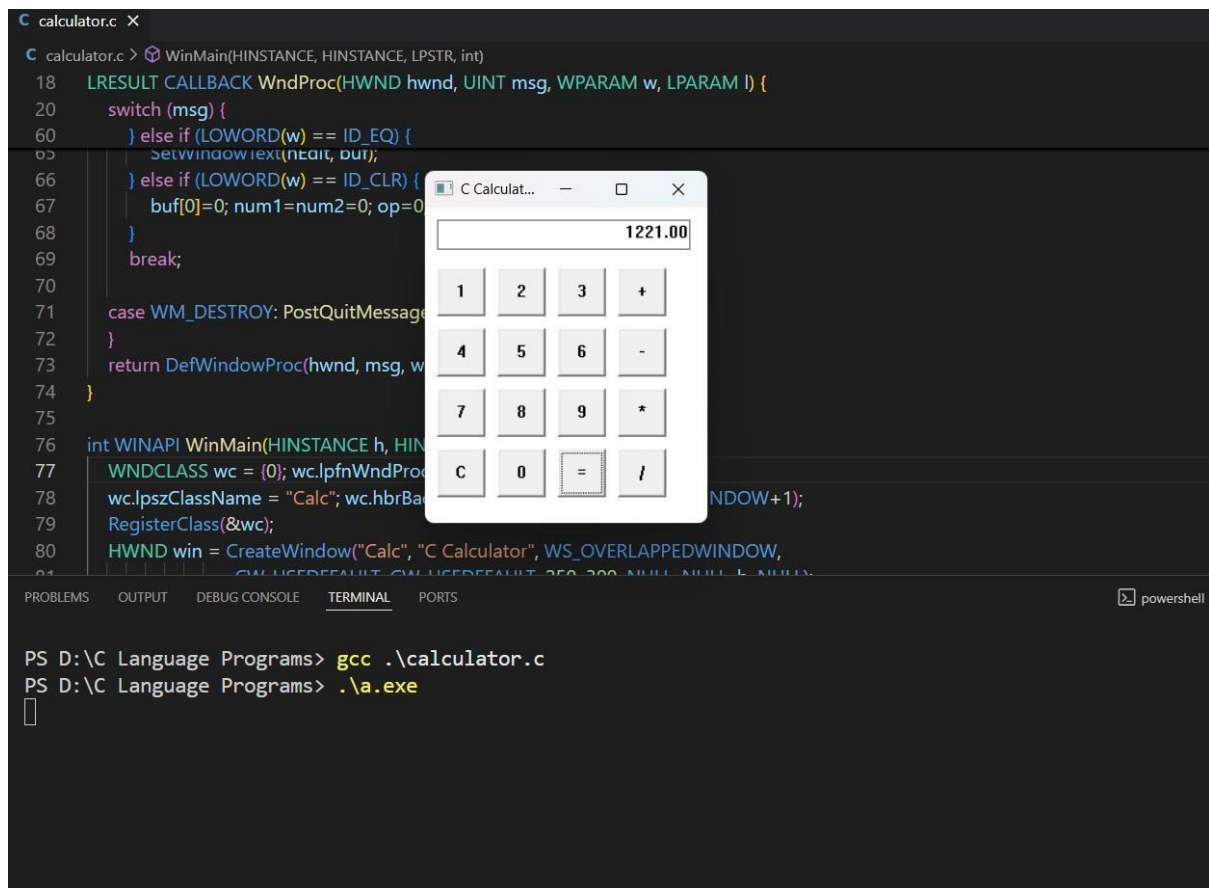
```
18 LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM w, LPARAM l) {
20     switch (msg) {
60     } else if (LOWORD(w) == ID_EQ) {
61         num2 = atof(buf);
62         double r = (op == '+') ? num1 + num2 : (op == '-') ? num1 - num2 :
63             (op == '*') ? num1 * num2 : (num2 != 0 ? num1 / num2 : 0);
64         sprintf(buf, "%2f", r);
65         SetWindowText(hEdit, buf);
66     } else if (LOWORD(w) == ID_CLR) {
67         buf[0] = 0; num1 = num2 = 0; op = 0; SetWindowText(hEdit, buf);
68     }
69     break;
70
71 case WM_DESTROY: PostQuitMessage(0); break;
72 }
73 return DefWindowProc(hwnd, msg, w, l);
74 }
75
76 int WINAPI WinMain(HINSTANCE h, HINSTANCE p, LPSTR cmd, int n) {
77     WNDCLASS wc = {0}; wc.lpfnWndProc = WndProc; wc.hInstance = h;
78     wc.lpszClassName = "Calc"; wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
79     RegisterClass(&wc);
80     HWND win = CreateWindow("Calc", "C Calculator", WS_OVERLAPPEDWINDOW,
81         CW_USEDEFAULT, CW_USEDEFAULT, 250, 300, NULL, NULL, h, NULL);
82     ShowWindow(win, n);
83     MSG m; while (GetMessage(&m, NULL, 0, 0)) { TranslateMessage(&m); DispatchMessage(&m); }
84     return 0;
85 }
86 }
```



This screenshot shows the same Visual Studio Code editor with 'calculator.c' open. A small, functional calculator window titled 'C Calculat...' is overlaid on the code. The calculator has a display field and buttons for digits 0-9, '+', '-', '\*', '/', and a 'C' button. Below the code editor, the terminal window shows the command prompt output for compiling and running the program.

```
C calculator.c > WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
18 LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM w, LPARAM l) {
20     switch (msg) {
60     } else if (LOWORD(w) == ID_EQ) {
61         SetWindowText(hEdit, buf);
66     } else if (LOWORD(w) == ID_CLR) {
67         buf[0]=0; num1=num2=0; op=0;
68     }
69     break;
70
71 case WM_DESTROY: PostQuitMessage(0);
72 }
73 return DefWindowProc(hwnd, msg, w, l);
74 }
75
76 int WINAPI WinMain(HINSTANCE h, HINSTANCE p, LPSTR cmd, int n) {
77     WNDCLASS wc = {0}; wc.lpfnWndProc = WndProc; wc.hInstance = h;
78     wc.lpszClassName = "Calc"; wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
79     RegisterClass(&wc);
80     HWND win = CreateWindow("Calc", "C Calculator", WS_OVERLAPPEDWINDOW,
81         CW_USEDEFAULT, CW_USEDEFAULT, 250, 300, NULL, NULL, h, NULL);
82     ShowWindow(win, n);
83     MSG m; while (GetMessage(&m, NULL, 0, 0)) { TranslateMessage(&m); DispatchMessage(&m); }
84     return 0;
85 }
86 }
```

PS D:\C Language Programs> gcc .\calculator.c  
PS D:\C Language Programs> .\a.exe



The image shows a Windows terminal window with a C program named `calculator.c` running. The program is a simple calculator that takes two numbers and an operator as input and outputs the result. The code is as follows:

```
18 LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM w, LPARAM l) {
20     switch (msg) {
60         } else if (LOWORD(w) == ID_EQ) {
65             SetWindowText(hwnd, buf);
66         } else if (LOWORD(w) == ID_CLR) {
67             buf[0]=0; num1=num2=0; op=0;
68         }
69     }
70     break;
71     case WM_DESTROY: PostQuitMessage(0);
72     }
73     return DefWindowProc(hwnd, msg, w, l);
74 }
75
76 int WINAPI WinMain(HINSTANCE h, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
77     WNDCLASS wc = {0}; wc.lpfnWndProc = WndProc; wc.hInstance = h;
78     wc.lpszClassName = "Calc"; wc.hbrBackground = (HBRUSH)COLOR_WINDOW+1;
79     RegisterClass(&wc);
80     HWND win = CreateWindow("Calc", "C Calculator", WS_OVERLAPPEDWINDOW,
81                             CW_USEDEFAULT, CW_USEDEFAULT, 300, 300, NULL, NULL, h, NULL);
```

The terminal output shows the program being compiled and executed:

```
PS D:\C Language Programs> gcc .\calculator.c
PS D:\C Language Programs> .\a.exe
```

A calculator window titled "C Calculat..." is overlaid on the terminal, showing the result "1221.00". The calculator has a standard interface with buttons for digits 0-9, operators (+, -, \*, /), and a clear (C) button.

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



# Basic Online Registration System

