**Lab Exercises**

**1. Write a Simple "Hello World" Program in Two Different Programming Languages**

Write a simple 'Hello World' program in Python and C.

**Code Example:**

**Python:**

print("Hello, World!")

**C:**

#include <stdio.h>

int main() {

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

}

**2. Create a Diagram of How Data is Transmitted from a Client to a Server Over the Internet**

****

**Diagram Instructions:**

* **Client** sends a request to the **Server** via the **ISP** (Internet Service Provider).
* Data is routed through **Routers** and **Switches**.
* The **Server** processes the request and sends a response back to the client.

**3. Design a Simple HTTP Client-Server Communication in Java**

**Code Example (Java HTTP Server):**

import java.io.\*;

import java.net.\*;

public class SimpleHTTPServer {

public static void main(String[] args) throws IOException {

ServerSocket serverSocket = new ServerSocket(8080);

System.out.println("Server started at port 8080");

while (true) {

try (Socket clientSocket = serverSocket.accept()) {

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

String requestLine;

// Reading the HTTP request

while ((requestLine = in.readLine()) != null) {

if (requestLine.isEmpty()) break;

System.out.println(requestLine);

}

// Sending a basic HTTP response

out.println("HTTP/1.1 200 OK");

out.println("Content-Type: text/plain");

out.println("");

out.println("Hello, World!");

}

}

}

}

**Code Example (Java HTTP Client):**

import java.io.\*;

import java.net.

public class SimpleHTTPClient {

public static void main(String[] args) throws IOException {

URL url = new URL("http://localhost:8080");

HttpURLConnection connection = (HttpURLConnection) url.openConnection();

connection.setRequestMethod("GET");

BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));

String inputLine;

StringBuffer response = new StringBuffer();

while ((inputLine = in.readLine()) != null) {

response.append(inputLine);

}

in.close();

System.out.println("Response from server: " + response.toString());

}

}

**4. Simulate HTTP and FTP Requests Using Command Line Tools (e.g., curl)**

**Description:**

Use curl to simulate HTTP and FTP requests.

**Examples:**

**HTTP Request using curl:**

curl http://example.com

**FTP Request using curl:**

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

**5. Research Different Types of Internet Connections**

**Description:**

Research and list different types of internet connections, such as broadband, fiber, and satellite. Include their pros and cons.

**Examples:**

1. **Broadband:**
   * **Pros**: Widely available, affordable.
   * **Cons**: Slower than fiber-optic, speeds can vary based on location.
2. **Fiber-Optic:**
   * **Pros**: Extremely fast, high bandwidth, low latency.
   * **Cons**: Expensive installation, not available in all areas.
3. **Satellite:**
   * **Pros**: Available in remote areas.
   * **Cons**: High latency, weather-dependent, more expensive than broadband.

**6. Simulate HTTP and FTP Requests Using Command Line Tools (e.g., curl)**

**Description:**

Simulate HTTP and FTP requests using the curl command-line tool.

**Examples:**

**HTTP Request:**

curl -X GET http://example.com

**FTP Request:**

curl -u username:password <ftp://ftp.example.com/file.txt>

**7. Identify and Explain Three Common Application Security Vulnerabilities**

* **Description**: Identify and explain three common application security vulnerabilities. Suggest possible solutions.

**Vulnerabilities**:

1. **SQL Injection**
   * **Solution**: Use prepared statements or parameterized queries to prevent malicious SQL code injection.
2. **Cross-Site Scripting (XSS)**
   * **Solution**: Sanitize and validate user inputs, use HTTP-only cookies, and implement Content Security Policy (CSP).
3. **Cross-Site Request Forgery (CSRF)**
   * **Solution**: Use anti-CSRF tokens and implement SameSite cookies.

**8. Identify and Classify 5 Applications You Use Daily as Either System Software or Application Software**

* **Description**: Identify 5 applications you use daily and classify them as system software or application software.

**Examples**:

1. **Google Chrome** – Application Software
2. **Microsoft Windows** – System Software
3. **Adobe Photoshop** – Application Software
4. **macOS** – System Software
5. **Spotify** – Application Software

**9. Design a Basic Three-Tier Software Architecture Diagram for a Web Application**

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

**Three-Tier Architecture**:

1. **Presentation Layer**: Frontend UI (HTML, CSS, JavaScript)
2. **Business Logic Layer**: Backend server (e.g., Node.js, Python, Java)
3. **Data Access Layer**: Database (e.g., MySQL, MongoDB)

*You can create the diagram using tools like Lucidchart or draw it on paper.*

**10. Create a Case Study on the Functionality of the Presentation, Business Logic, and Data Access Layers of a Given Software System**

* **Description**: Create a case study on the functionality of the three layers (presentation, business logic, data access) in a software system.

**Example Case Study**:

* **Application**: E-commerce website.
  + **Presentation Layer**: User interface for browsing products, adding items to cart, and checkout.
  + **Business Logic Layer**: Handles cart calculations, user authentication, order processing.
  + **Data Access Layer**: Interacts with a database to store product details, user data, and order history.

**11. Explore Different Types of Software Environments (Development, Testing, Production). Set Up a Basic Environment in a Virtual Machine**

* **Description**: Explore different types of software environments and set up a basic environment in a virtual machine.

**Steps**:

1. **Development Environment**: Install IDE (e.g., Visual Studio Code, IntelliJ IDEA) and configure local databases.
2. **Testing Environment**: Set up a testing server or container (e.g., Docker) for isolated test environments.
3. **Production Environment**: Set up a production server with monitoring tools (e.g., Nagios, Prometheus) and deploy the application.

**12. Write and Upload Your First Source Code File to Github**

* **Description**: Write a simple source code file and upload it to GitHub.

**Steps**:

1. Write a basic program (e.g., Hello World in Java, Python).
2. Create a GitHub repository.
3. Initialize a local Git repository, commit the code, and push it to GitHub.

**Example**:

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

git init

git add HelloWorld.java

git commit -m "Initial commit"

git remote add origin <your-github-repository-url>

git push -u origin master

**13. Create a GitHub Repository and Document How to Commit and Push Code Changes**

* **Description**: Create a GitHub repository, then document how to commit and push code changes.

**Steps**:

1. Go to GitHub and create a new repository.
2. Clone the repository to your local machine.
3. Make changes to the code, commit them, and push back to GitHub.

**Commands**:

git clone <repository-url>

cd <repository-folder>

# Make changes to files

git add .

git commit -m "Your commit message"

git push origin main

**14. Create a Student Account on GitHub and Collaborate on a Small Project with a Classmate**

* **Description**: Create a student account on GitHub and collaborate on a project with a classmate.

**Steps**:

1. Sign up on GitHub.
2. Create a new repository for the project.
3. Collaborate by working on different parts of the project, commit changes, and push them to the shared repository.

**15. Create a List of Software You Use Regularly and Classify Them into the Following Categories: System, Application, and Utility Software**

* **Description**: Create a list of software you use regularly and classify them into system, application, and utility software.

**Examples**:

* **System Software**: Windows, macOS, Linux
* **Application Software**: Microsoft Word, Adobe Photoshop, Google Chrome
* **Utility Software**: Antivirus software, File compression tools (e.g., WinRAR), Backup tools

**16. Follow a Git Tutorial to Practice Cloning, Branching, and Merging Repositories**

* **Description**: Follow a Git tutorial to practice cloning, branching, and merging repositories with C code.

**Steps**:

1. **Clone the repository**:
   * Clone the repository that contains a C project.

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

1. **Create a new branch**:
   * Create a new branch to work on a feature or bug fix:

git checkout -b new-feature

1. **Add a new C file**:
   * Write a simple C program (e.g., hello.c):

#include <stdio.h>

int main() {

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

return 0;

}

* + Commit the changes:

git add hello.c

git commit -m "Add hello.c"

1. **Merge the branch**:
   * Merge the feature branch back into the main branch:

git checkout main

git merge new-feature

**17. Write a Report on the Various Types of Application Software and How They Improve Productivity**

* **Description**: Research various types of application software and write a report. Focus on how productivity can be improved with C-based software.

**Examples**:

1. **Text Editors (e.g., Vim, Emacs)**:
   * Editors that support C programming, with features like syntax highlighting and code completion.
2. **Compilers (e.g., GCC)**:
   * C compilers like GCC enable developers to convert C code into machine-readable instructions.
3. **IDE (e.g., Code::Blocks)**:
   * Integrated development environments (IDEs) for C improve productivity by offering debugging tools, code completion, and error tracking.

**18. Create a Flowchart Representing the Software Development Life Cycle (SDLC)**

* **Description**: Create a flowchart that represents the Software Development Life Cycle, with a focus on C programming projects.

**Flowchart Stages**:

1. **Requirement Gathering**: Understand the problem you want to solve using C.
2. **Design**: Design the system architecture, algorithms, and data structures (e.g., arrays, structures).
3. **Implementation**: Write the C code that implements the design.
4. **Testing**: Test the C program for correctness using test cases.
5. **Deployment**: Deploy the compiled C application.
6. **Maintenance**: Address any bugs or enhancements in the C program.

You can create the flowchart using tools like **Lucidchart**, **Draw.io**, or **Microsoft Visio**.

**19. Lab Exercise: Write a Requirement Specification for a Simple Library Management System**

* **Description**: Write a requirement specification document for a library management system implemented in C.

**Example Requirements**:

1. **Functional Requirements**:
   * The system must allow users to check out and return books.
   * It must keep track of books' due dates and overdue items.
2. **Non-Functional Requirements**:
   * The system should be able to handle up to 1000 book records.
   * The system should have a command-line interface (CLI) written in C.

**20. Lab Exercise: Perform a Functional Analysis for an Online Shopping System**

* **Description**: Perform functional analysis for an online shopping system with a C implementation.

**Functional Analysis**:

1. **User Login**:
   * Implement user authentication using simple file I/O in C (read and write usernames and passwords).
2. **Product Catalog**:
   * Display products in the catalog using arrays or linked lists in C.
3. **Shopping Cart**:
   * Allow users to add products to their cart and calculate totals.
4. **Checkout and Payment**:
   * Simulate checkout with a simple price calculation and payment simulation.

**21. Lab Exercise: Design a Basic System Architecture for a Food Delivery App**

* **Description**: Design the system architecture for a food delivery application with a focus on C programming.

**Architecture Overview**:

1. **Frontend (CLI-based)**:
   * A user interface in C that allows customers to browse a menu and place orders.
2. **Backend (C-based server)**:
   * Handles customer requests, orders, and delivery statuses using C-based networking (e.g., using socket programming).
3. **Database (File-based)**:
   * Use file handling in C (e.g., reading/writing to text files) to store food item details, user information, and order history.

**22. Lab Exercise: Develop Test Cases for a Simple Calculator Program**

* **Description**: Write test cases for a simple calculator program in C that performs addition, subtraction, multiplication, and division.

**Example Calculator Program**:

#include <stdio.h>

int add(int a, int b) { return a + b; }

int subtract(int a, int b) { return a - b; }

int multiply(int a, int b) { return a \* b; }

int divide(int a, int b) {

if (b != 0) {

return a / b;

} else {

printf("Error: Division by zero!\n");

return -1;

}

}

int main() {

printf("Add: %d\n", add(5, 3));

printf("Subtract: %d\n", subtract(5, 3));

printf("Multiply: %d\n", multiply(5, 3));

printf("Divide: %d\n", divide(5, 0)); // Test division by zero

return 0;

}

**Test Cases**:

1. **Test Addition**:
   * **Input**: add(2, 3)
   * **Expected Output**: 5
2. **Test Subtraction**:
   * **Input**: subtract(5, 3)
   * **Expected Output**: 2
3. **Test Multiplication**:
   * **Input**: multiply(2, 3)
   * **Expected Output**: 6
4. **Test Division**:
   * **Input**: divide(6, 3)
   * **Expected Output**: 2
   * **Test Division by Zero**:
   * **Input**: divide(5, 0)
   * **Expected Output**: Error: Division by zero!

**23. Document a Real-World Case Where a Software Application Required Critical Maintenance**

* **Description**: Document a real-world example where a C application required critical maintenance.

**Example Case**:

* **Application**: A file management tool written in C.
  + **Issue**: The application crashed when handling large files due to a memory leak.
  + **Maintenance**: The development team identified the source of the memory leak using debugging tools (e.g., Valgrind). They fixed the issue by improving memory management techniques and optimizing the code.

**24. Create a DFD for a Hospital Management System (HMS)**

Although Data Flow Diagrams (DFDs) are more of a design tool, you can simulate creating the process and data flows in C by modeling the system with structures and functions.

**Steps**:

1. **Define Data Structures** (e.g., Patients, Appointments, Medical Records, etc.):

#include <stdio.h>

#include <string.h>

// Structure for Patient

typedef struct {

int patientID;

char name[100];

int age;

char gender[10];

} Patient;

// Structure for Appointment

typedef struct {

int appointmentID;

int patientID;

char doctor[100];

char date[15];

} Appointment;

// Function to add patient

void addPatient(Patient\* p) {

printf("Enter patient ID: ");

scanf("%d", &p->patientID);

printf("Enter patient name: ");

scanf("%s", p->name);

printf("Enter patient age: ");

scanf("%d", &p->age);

printf("Enter patient gender: ");

scanf("%s", p->gender);

}

// Function to print patient details

void printPatient(Patient p) {

printf("Patient ID: %d\n", p.patientID);

printf("Name: %s\n", p.name);

printf("Age: %d\n", p.age);

printf("Gender: %s\n", p.gender);

}

int main() {

Patient patient1;

addPatient(&patient1);

printPatient(patient1);

return 0;

}

* + **Explanation**: This program models a basic structure for a Patient and a simple addPatient function.
  + **Data Flow Simulation**: You could extend this example by adding other functionalities like appointment scheduling, medical record management, etc.

**25. Build a Simple Desktop Calculator Application Using a GUI Library**

To build a simple desktop calculator in C, you would typically use a GUI library such as **GTK** or **WinAPI** (for Windows). Here’s a simple example using the **GTK** library for a basic calculator.

**Steps**:

1. **Install GTK** (if you don’t have it already):
   * For Ubuntu/Debian:

sudo apt-get install libgtk-3-dev

1. **Write the Calculator Program**:

#include <gtk/gtk.h>

// Function to handle button clicks

void on\_button\_click(GtkButton \*button, gpointer user\_data) {

const char \*button\_label = gtk\_button\_get\_label(button);

GtkEntry \*entry = GTK\_ENTRY(user\_data);

const char \*current\_text = gtk\_entry\_get\_text(entry);

// Append the label of the button to the entry

char new\_text[256];

snprintf(new\_text, sizeof(new\_text), "%s%s", current\_text, button\_label);

gtk\_entry\_set\_text(entry, new\_text);

}

// Function to handle calculation

void on\_calculate(GtkButton \*button, gpointer user\_data) {

GtkEntry \*entry = GTK\_ENTRY(user\_data);

const char \*expression = gtk\_entry\_get\_text(entry);

// This is a simple implementation; you can extend it by adding expression parsing and evaluation logic

// For now, we'll just print the expression entered by the user

printf("Expression: %s\n", expression);

gtk\_entry\_set\_text(entry, ""); // Clear the entry after calculation

}

int main(int argc, char \*argv[]) {

GtkWidget \*window;

GtkWidget \*grid;

GtkWidget \*entry;

GtkWidget \*button;

gtk\_init(&argc, &argv);

// Create a new window

window = gtk\_window\_new(GTK\_WINDOW\_TOPLEVEL);

gtk\_window\_set\_title(GTK\_WINDOW(window), "Simple Calculator");

gtk\_window\_set\_default\_size(GTK\_WINDOW(window), 300, 400);

g\_signal\_connect(window, "destroy", G\_CALLBACK(gtk\_main\_quit), NULL);

// Create a grid to organize the buttons

grid = gtk\_grid\_new();

gtk\_container\_add(GTK\_CONTAINER(window), grid);

// Create an entry for displaying the expression

entry = gtk\_entry\_new();

gtk\_grid\_attach(GTK\_GRID(grid), entry, 0, 0, 4, 1);

// Create buttons for digits and operations

const char \*buttons[] = {

"7", "8", "9", "/",

"4", "5", "6", "\*",

"1", "2", "3", "-",

"0", ".", "=", "+"

};

int row = 1, col = 0;

for (int i = 0; i < 16; i++) {

button = gtk\_button\_new\_with\_label(buttons[i]);

gtk\_grid\_attach(GTK\_GRID(grid), button, col, row, 1, 1);

// Connect button to the click handler

if (buttons[i][0] == '=') {

g\_signal\_connect(button, "clicked", G\_CALLBACK(on\_calculate), entry);

} else {

g\_signal\_connect(button, "clicked", G\_CALLBACK(on\_button\_click), entry);

}

col++;

if (col > 3) {

col = 0;

row++;

}

}

// Show all widgets

gtk\_widget\_show\_all(window);

// Start the GTK main loop

gtk\_main();

return 0;

}

**Explanation**:

* This program creates a simple calculator using GTK. The GUI allows users to enter numbers and operations.
* The program processes button clicks and displays the result of the expression in the text entry widget.

1. **Compiling the Program**:
   * Compile using:

gcc -o calculator calculator.c $(pkg-config --cflags --libs gtk+-3.0)

1. **Running the Program**:
   * After compiling, run the program:

**26. Draw a Flowchart Representing the Logic of a Basic Online Registration System**

In this lab exercise, you can simulate the flowchart logic using C code to represent an online registration system.

**Steps to create a basic C program**:

1. **Define the Structure** for user information:

#include <stdio.h>

#include <string.h>

typedef struct {

char username[50];

char password[50];

char email[100];

} User;

void registerUser(User \*u) {

printf("Enter username: ");

scanf("%s", u->username);

printf("Enter password: ");

scanf("%s", u->password);

printf("Enter email: ");

scanf("%s", u->email);

}

void displayUserInfo(User u) {

printf("Username: %s\n", u.username);

printf("Email: %s\n", u.email);

}

int main() {

User newUser;

registerUser(&newUser);

displayUserInfo(newUser);

return 0;

}

**Explanation**:

* This program collects user details such as username, password, and email from the user.
* The flow of this registration process can be represented as:
  1. **Start**
  2. **Enter username**
  3. **Enter password**
  4. **Enter email**
  5. **Display registration info**
  6. **End**