Practical Slip 1

Q.1) Database Management System (PostgreSQL)

Consider the following database: Room (room_no, room_name, room_type, charges) Guest (Guest_code, Gname, city, no_of persons) The relationship is as follows: Room-Guest: one-to-one. The room_type can have values as either 'AC' or 'NonAC'.

A) Create above database in PostgreSQL and insert sufficient records.

```
CREATE TABLE Room (
    room_no INT PRIMARY KEY,
    room_name VARCHAR(50),
    room_type VARCHAR(10) CHECK (room_type IN ('AC', 'NonAC')),
    charges DECIMAL(10, 2)
);

CREATE TABLE Guest (
    Guest_code INT PRIMARY KEY,
    Gname VARCHAR(50),
    city VARCHAR(50),
    no_of_persons INT,
    room_no INT UNIQUE REFERENCES Room(room_no)
);

INSERT INTO Room VALUES
(101, 'Ocean View', 'AC', 5000),
(102, 'Garden View', 'NonAC', 2000),
(103, 'Suite', 'AC', 8000);

INSERT INTO Guest VALUES
(1, 'Sameer', 'Pune', 2, 101),
(2, 'Rohan', 'Mumbai', 3, 102),
(3, 'Sneha', 'Delhi', 2, 103);
```

i) List all guests whose name starts with "S".

```
SELECT * FROM Guest WHERE Gname LIKE 'S%';
```

ii) Increase the charges of all AC rooms by 15%.

```
UPDATE Room SET charges = charges + (charges * 0.15) WHERE room_type = 'AC';
```

iii) List the minimum charges of a room.

```
SELECT MIN(charges) FROM Room;
```

iv) List the names of the guests in the sorted order by city name.

```
SELECT Gname, city FROM Guest ORDER BY city ASC;
```

B) Write a procedure to find sum and product of two numbers.

```
CREATE OR REPLACE PROCEDURE calculate_sum_prod(n1 INT, n2 INT)
LANGUAGE plpgsql
AS $$
DECLARE
    sum_val INT;
    prod_val INT;
BEGIN
```

```
    sum_val := n1 + n2;
    prod_val := n1 * n2;
    RAISE NOTICE 'Sum: %, Product: %', sum_val, prod_val;
END;
$$;

CALL calculate_sum_prod(10, 5);
```

Q.2) Operating Systems

Write a program to implement FCFS CPU scheduling algorithm. Take arrival time, burst time for n number of processes from the user. Calculate average waiting time.

C

```c
#include <stdio.h>

struct Process {
    int id;
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
};

void sortProcs(struct Process p[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].at > p[j + 1].at) {
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n;
    float total_wt = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
    }

    sortProcs(p, n);

    int current_time = 0;
    for (int i = 0; i < n; i++) {
        if (current_time < p[i].at) {
            current_time = p[i].at;
        }
        p[i].ct = current_time + p[i].bt;
        p[i].tat = p[i].ct - p[i].at;
        p[i].wt = p[i].tat - p[i].bt;
        total_wt += p[i].wt;
        current_time = p[i].ct;
    }

    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
    }

    printf("\nAverage Waiting Time: %.2f\n", total_wt / n);

    return 0;
}
```

Practical Slip 2

Q.1) Database Management System (PostgreSQL)

Consider the following database: College (cno, cname, street name, ccity) Principal (pno, pname, experience, Salary) The relationship is as follows: College-Principal: one-to-one. Experience must greater than 10 years.

A) Create above database in PostgreSQL and insert sufficient records.

SQL

```
CREATE TABLE College (
    cno INT PRIMARY KEY,
    cname VARCHAR(100),
    street_name VARCHAR(100),
    ccity VARCHAR(50)
);

CREATE TABLE Principal (
    pno INT PRIMARY KEY,
    pname VARCHAR(50),
    experience INT CHECK (experience > 10),
    salary DECIMAL(10, 2),
    cno INT UNIQUE REFERENCES College(cno)
);

INSERT INTO College VALUES
(1, 'Fergusson College', 'FC Road', 'Pune'),
(2, 'Modern College', 'MG Road', 'Pune');

INSERT INTO Principal VALUES
(101, 'Dr. Sharma', 15, 120000, 1),
(102, 'Dr. Anand', 12, 110000, 2);
```

i) Display all colleges whose name contains 'and'.

```
SELECT * FROM College WHERE cname ILIKE '%and%';
```

ii) List the average salary of a Principal.

```
SELECT AVG(salary) FROM Principal;
```

iii) List the names of all Principals having experience between 10 to 20 years.

```
SELECT pname FROM Principal WHERE experience BETWEEN 10 AND 20;
```

iv) Change the street name of college from MG Road to Nehru road.

```
UPDATE College SET street_name = 'Nehru road' WHERE street_name = 'MG Road';
```

B) Write a stored procedure to insert a record in table College.

```
CREATE OR REPLACE PROCEDURE insert_college(
    p_cno INT,
    p_cname VARCHAR,
    p_street VARCHAR,
    p_city VARCHAR
)
```

```
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO College (cno, cname, street_name, ccity)
    VALUES (p_cno, p_cname, p_street, p_city);
END;
$$;

CALL insert_college(3, 'Wadia College', 'Bund Garden', 'Pune');
```

Q.2) Operating Systems

Write a program to implement FCFS CPU scheduling algorithm. Take arrival time, burst time for n number of processes from the user. Calculate average turnaround time.

```c
#include <stdio.h>

struct Process {
    int id;
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
};

void sortProcs(struct Process p[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].at > p[j + 1].at) {
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n;
    float total_tat = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
    }

    sortProcs(p, n);

    int current_time = 0;
    for (int i = 0; i < n; i++) {
        if (current_time < p[i].at) {
            current_time = p[i].at;
        }
        p[i].ct = current_time + p[i].bt;
        p[i].tat = p[i].ct - p[i].at;
        p[i].wt = p[i].tat - p[i].bt;
        total_tat += p[i].tat;
        current_time = p[i].ct;
    }

    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
    }

    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);

    return 0;
}
```

Practical Slip 3

Q.1) Database Management System (PostgreSQL)

Consider the following database:Employee(eno, ename, designation, salary)Department(dno, dname, location)The relationship is as follows: Employee-Department: many-to-one. Location should not be null.

A) Create above database in PostgreSQL and insert sufficient records.

```
CREATE TABLE Department (
    dno INT PRIMARY KEY,
    dname VARCHAR(50),
    location VARCHAR(50) NOT NULL
);

CREATE TABLE Employee (
    eno INT PRIMARY KEY,
    ename VARCHAR(50),
    designation VARCHAR(50),
    salary DECIMAL(10, 2),
    dno INT REFERENCES Department(dno)
);

INSERT INTO Department VALUES
(1, 'Sales', 'Pune'),
(2, 'Marketing', 'Mumbai'),
(3, 'HR', 'Pune');

INSERT INTO Employee VALUES
(101, 'Omkar', 'Manager', 85000.00, 1),
(102, 'Sagar', 'Clerk', 45000.00, 1),
(103, 'Pooja', 'Analyst', 60000.00, 2),
(104, 'Sameer', 'Manager', 82000.00, 3);
```

i) Give a 5% raise in salary to all the employees.

```
UPDATE Employee SET salary = salary + (salary * 0.05);
```

ii) Display average salary of an employee.

```
SELECT AVG(salary) FROM Employee;
```

iii) List the details of all the departments located at city "Pune".

```
SELECT * FROM Department WHERE location = 'Pune';
```

iv) Display the details of employees whose names ends with an alphabet "r".

```
SELECT * FROM Employee WHERE ename LIKE '%r';
```

B) Write a stored function using cursors to display all the details of Employee whose salary is more than 80,000.

```
CREATE OR REPLACE FUNCTION display_high_salary_emp()
RETURNS VOID AS $$
DECLARE
```

```
    emp_cursor CURSOR FOR SELECT eno, ename, designation, salary FROM Employee WHERE salary > 80000;
    rec RECORD;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO rec;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'ID: %, Name: %, Desg: %, Salary: %', rec.eno, rec.ename, rec.designation, rec.salary;
    END LOOP;
    CLOSE emp_cursor;
END;
$$ LANGUAGE plpgsql;

SELECT display_high_salary_emp();
```

Q.2) Operating Systems

Write a program to simulate Pre-emptive Shortest Job First (SJF) CPU scheduling algorithm. Accept no. of processes, arrival time and burst time from user. Calculate and display the average waiting time.

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

struct Process {
    int id;
    int at;
    int bt;
    int rt;
    int wt;
    int tat;
    int ct;
};

int main() {
    int n, completed = 0, current_time = 0, min_rt_idx = -1, min_rt = INT_MAX;
    float total_wt = 0;
    bool found = false;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].rt = p[i].bt;
    }

    while (completed != n) {
        min_rt = INT_MAX;
        min_rt_idx = -1;
        found = false;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= current_time && p[i].rt > 0) {
                if (p[i].rt < min_rt) {
                    min_rt = p[i].rt;
                    min_rt_idx = i;
                    found = true;
                }
            }
        }

        if (!found) {
            current_time++;
            continue;
        }

        p[min_rt_idx].rt--;
        current_time++;

        if (p[min_rt_idx].rt == 0) {
            completed++;
            p[min_rt_idx].ct = current_time;
            p[min_rt_idx].tat = p[min_rt_idx].ct - p[min_rt_idx].at;
            p[min_rt_idx].wt = p[min_rt_idx].tat - p[min_rt_idx].bt;
            total_wt += p[min_rt_idx].wt;
        }
    }

    printf("\nPID\tAT\tBT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].tat, p[i].wt);
    }

    printf("\nAverage Waiting Time: %.2f\n", total_wt / n);

    return 0;
}
```

Practical Slip 4

Q.1) Database Management System (PostgreSQL)

Consider the following database: Person (pnumber, pname, birthdate, income) Area (area_code, aname, area_type, pincode) The relationship is as follows: Person-Area: many-to-one. The area_type can have values as either "urban" or "rural".

A) Create above database in PostgreSQL and insert sufficient records.

```
CREATE TABLE Area (
    area_code INT PRIMARY KEY,
    aname VARCHAR(50),
    area_type VARCHAR(10) CHECK (area_type IN ('urban', 'rural')),
    pincode INT
);

CREATE TABLE Person (
    pnumber INT PRIMARY KEY,
    pname VARCHAR(50),
    birthdate DATE,
    income DECIMAL(10, 2),
    area_code INT REFERENCES Area(area_code)
);

INSERT INTO Area VALUES
(1, 'Kalyaninagar', 'urban', 411006),
(2, 'Manchar', 'rural', 410503),
(3, 'Kothrud', 'urban', 411038);

INSERT INTO Person VALUES
(101, 'Ramesh', '1990-05-15', 55000.00, 1),
(102, 'Suresh', '1992-08-20', 45000.00, 2),
(103, 'Rita', '1995-12-10', 75000.00, 3),
(104, 'Mahesh', '1988-01-25', 60000.00, 1);
```

i) List the details of all people whose name starts with the alphabet "R".

```
SELECT * FROM Person WHERE pname LIKE 'R%';
```

ii) Display the details of people in the sorted order of their income.

```
SELECT * FROM Person ORDER BY income ASC;
```

iii) Display the count of areas of "urban" type.

```
SELECT COUNT(*) FROM Area WHERE area_type = 'urban';
```

iv) Change the pincode of "kalyaninagar" to 411036.

```
UPDATE Area SET pincode = 411036 WHERE aname = 'Kalyaninagar';
```

B) Create a stored procedure named as "addrecords" for adding person records.

```
CREATE OR REPLACE PROCEDURE addrecords(
    p_no INT,
    p_name VARCHAR,
    p_dob DATE,
    p_inc DECIMAL,
    p_acode INT
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO Person (pnumber, pname, birthdate, income, area_code)
    VALUES (p_no, p_name, p_dob, p_inc, p_acode);
END;
$$;

CALL addrecords(105, 'Ganesh', '1999-07-07', 30000.00, 2);
```

Q.2) Operating Systems

Write a program to simulate Pre-emptive Shortest Job First (SJF) CPU scheduling algorithm. Accept no. of processes, arrival time and burst time from user. Calculate and Display the average turnaround time.

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

struct Process {
    int id;
    int at;
    int bt;
    int rt;
    int wt;
    int tat;
    int ct;
};

int main() {
    int n, completed = 0, current_time = 0, min_rt_idx = -1, min_rt = INT_MAX;
    float total_tat = 0;
    bool found = false;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].rt = p[i].bt;
    }

    while (completed != n) {
        min_rt = INT_MAX;
        min_rt_idx = -1;
        found = false;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= current_time && p[i].rt > 0) {
                if (p[i].rt < min_rt) {
                    min_rt = p[i].rt;
                    min_rt_idx = i;
                    found = true;
                }
            }
        }

        if (!found) {
            current_time++;
            continue;
        }

        p[min_rt_idx].rt--;
        current_time++;

        if (p[min_rt_idx].rt == 0) {
            completed++;
            p[min_rt_idx].ct = current_time;
            p[min_rt_idx].tat = p[min_rt_idx].ct - p[min_rt_idx].at;
            p[min_rt_idx].wt = p[min_rt_idx].tat - p[min_rt_idx].bt;
            total_tat += p[min_rt_idx].tat;
        }
    }

    printf("\nPID\tAT\tBT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].tat, p[i].wt);
    }

    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);

    return 0;
}
```

Practical Slip 5

Q.1) Database Management System (PostgreSQL)

Consider the following database:Doctor (dno, dname, addr, phone_no, specialization)Patient (pno, pat_name, city, disease)The relationship is as follows: Doctor-Patient: many-to-many.

A) Create above database in PostgreSQL and insert sufficient records.

```
CREATE TABLE Doctor (
    dno INT PRIMARY KEY,
    dname VARCHAR(50),
    addr VARCHAR(50),
    phone_no VARCHAR(15),
    specialization VARCHAR(50)
);

CREATE TABLE Patient (
    pno INT PRIMARY KEY,
    pat_name VARCHAR(50),
    city VARCHAR(50),
    disease VARCHAR(50)
);

CREATE TABLE Doctor_Patient (
    dno INT REFERENCES Doctor(dno),
    pno INT REFERENCES Patient(pno),
    PRIMARY KEY (dno, pno)
);

INSERT INTO Doctor VALUES
(1, 'Dr. Mahajan', 'Alandi', '9876543210', 'Neurologist'),
(2, 'Dr. Sathe', 'Pune', '9123456789', 'Cardiologist'),
(3, 'Dr. Mane', 'Alandi', '9988776655', 'Dentist');

INSERT INTO Patient VALUES
(101, 'Ravi', 'Nashik', 'Fever'),
(102, 'Sita', 'Pune', 'Flu'),
(103, 'Gita', 'Nashik', 'Fever');

INSERT INTO Doctor_Patient VALUES (1, 101), (2, 102), (1, 103);
```

i) Find the names of all doctors which start with "M".

```
SELECT dname FROM Doctor WHERE dname LIKE 'M%';
```

ii) Count the number of doctors who are Neurologists.

```
SELECT COUNT(*) FROM Doctor WHERE specialization = 'Neurologist';
```

iii) Give the list of all patients who are suffering from "Fever".

```
SELECT * FROM Patient WHERE disease = 'Fever';
```

iv) Find the specialization and phone numbers of all doctors from Alandi.

```
SELECT specialization, phone_no FROM Doctor WHERE addr = 'Alandi';
```

B) Write a stored function using cursors to display all the details of all Patients from Nashik city.

```
CREATE OR REPLACE FUNCTION display_nashik_patients()
RETURNS VOID AS $$
DECLARE
    pat_cursor CURSOR FOR SELECT * FROM Patient WHERE city = 'Nashik';
    rec RECORD;
BEGIN
    OPEN pat_cursor;
    LOOP
        FETCH pat_cursor INTO rec;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'ID: %, Name: %, Disease: %', rec.pno, rec.pat_name, rec.disease;
    END LOOP;
    CLOSE pat_cursor;
END;
$$ LANGUAGE plpgsql;

SELECT display_nashik_patients();
```

Q.2) Operating Systems

Write a program to simulate Non-Pre-emptive Shortest Job First (SJF) scheduling. Accept no. of processes, arrival time and burst time. Calculate and display the average waiting time.

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

struct Process {
    int id;
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
    bool is_completed;
};

int main() {
    int n, completed = 0, current_time = 0;
    float total_wt = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].is_completed = false;
    }

    while (completed != n) {
        int idx = -1;
        int min_bt = INT_MAX;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= current_time && !p[i].is_completed) {
                if (p[i].bt < min_bt) {
                    min_bt = p[i].bt;
                    idx = i;
                }
            }
        }

        if (idx != -1) {
            p[idx].ct = current_time + p[idx].bt;
            p[idx].tat = p[idx].ct - p[idx].at;
            p[idx].wt = p[idx].tat - p[idx].bt;
            total_wt += p[idx].wt;
            p[idx].is_completed = true;
            completed++;
            current_time = p[idx].ct;
        } else {
            current_time++;
        }
    }

    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
    }

    printf("\nAverage Waiting Time: %.2f\n", total_wt / n);

    return 0;
}
```

Practical Slip 6

Q.1) Database Management System (PostgreSQL)

Consider the following database: Student (rno, name, city) Teacher(tno, tname, phone_no, salary)

The relationship is as follows: Student-Teacher: many-to-many with subject as a descriptive attribute.

A) Create above database in PostgreSQL and insert sufficient records.

```
CREATE TABLE Student (
    rno INT PRIMARY KEY,
    name VARCHAR(50),
    city VARCHAR(50)
);

CREATE TABLE Teacher (
    tno INT PRIMARY KEY,
    tname VARCHAR(50),
    phone_no VARCHAR(15),
    salary DECIMAL(10, 2)
);

CREATE TABLE Student_Teacher (
    rno INT REFERENCES Student(rno),
    tno INT REFERENCES Teacher(tno),
    subject VARCHAR(50),
    PRIMARY KEY (rno, tno)
);

INSERT INTO Student VALUES
(1, 'Shubham', 'Pune'),
(2, 'Shivani', 'Mumbai'),
(3, 'Rahul', 'Pune');

INSERT INTO Teacher VALUES
(101, 'Prof. Satkar', '9999999999', 75000.00),
(102, 'Prof. Joshi', '8888888888', 82000.00);

INSERT INTO Student_Teacher VALUES (1, 101, 'OS'), (2, 102, 'DB'), (3, 101, 'OS');
```

i) List all students whose name start from 'Sh'

```
SELECT * FROM Student WHERE name LIKE 'Sh%';
```

ii) Display the count of students from city

```
SELECT COUNT(*) FROM Student WHERE city = 'Pune';
```

iii) Find the maximum salary of teachers.

```
SELECT MAX(salary) FROM Teacher;
```

iv) Change the phone number of "Prof. Satkar" to "9822131226"

```
UPDATE Teacher SET phone_no = '9822131226' WHERE tname = 'Prof. Satkar';
```

B) Create a stored procedure named as "updaterecords" to give 5% rise in salary of teacher.

```
CREATE OR REPLACE PROCEDURE updaterecords()
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE Teacher SET salary = salary + (salary * 0.05);
    RAISE NOTICE 'Salaries updated successfully.';
END;
$$;

CALL updaterecords();
```

Q.2) Operating Systems

Write a program to simulate Non-Pre-emptive Shortest Job First (SJF) scheduling. Accept no. of processes, arrival time and burst time. Calculate and display the average turnaround time.

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

struct Process {
    int id;
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
    bool is_completed;
};

int main() {
    int n, completed = 0, current_time = 0;
    float total_tat = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].is_completed = false;
    }

    while (completed != n) {
        int idx = -1;
        int min_bt = INT_MAX;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= current_time && !p[i].is_completed) {
                if (p[i].bt < min_bt) {
                    min_bt = p[i].bt;
                    idx = i;
                }
            }
        }

        if (idx != -1) {
            p[idx].ct = current_time + p[idx].bt;
            p[idx].tat = p[idx].ct - p[idx].at;
            p[idx].wt = p[idx].tat - p[idx].bt;
            total_tat += p[idx].tat;
            p[idx].is_completed = true;
            completed++;
            current_time = p[idx].ct;
        } else {
            current_time++;
        }
    }

    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
    }

    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);

    return 0;
}
```

Practical Slip 7

Q.1) Database Management System (PostgreSQL)

Consider the following database: Policy (pno, pname, premium_amt, policy_type) Customer (cno, cname, city, agent_name) The relationship is as follows: Policy-Customer: many-to-one. The "policy_type" can have values as "Yearly", "Half-yearly" or "Monthly"

A) Create above database in PostgreSQL and insert sufficient records.

```
CREATE TABLE Customer (
    cno INT PRIMARY KEY,
    cname VARCHAR(50),
    city VARCHAR(50),
    agent_name VARCHAR(50)
);

CREATE TABLE Policy (
    pno INT PRIMARY KEY,
    pname VARCHAR(50),
    premium_amt DECIMAL(10, 2),
    policy_type VARCHAR(20) CHECK (policy_type IN ('Yearly', 'Half-yearly', 'Monthly')),
    cno INT REFERENCES Customer(cno)
);

INSERT INTO Customer VALUES
(1, 'Amit', 'Pune', 'Ramesh'),
(2, 'Sumit', 'Mumbai', 'Suresh');

INSERT INTO Policy VALUES
(101, 'Jeevan Anand', 12000, 'Yearly', 1),
(102, 'Health Plus', 5000, 'Monthly', 1),
(103, 'Term Life', 6000, 'Half-yearly', 2);
```

i) List the details of all customers who live in "Pune" city.

```
SELECT * FROM Customer WHERE city = 'Pune';
```

ii) Display the average premium amount.

```
SELECT AVG(premium_amt) FROM Policy;
```

iii) Increases the premium amount for Monthly policies by 10%.

```
UPDATE Policy SET premium_amt = premium_amt + (premium_amt * 0.10)
WHERE policy_type = 'Monthly';
```

iv) Display the policy type wise count of policies.

```
SELECT policy_type, COUNT(*) FROM Policy GROUP BY policy_type;
```

B) Create a stored function named as "max_premium" which will find max premium amount.

```
CREATE OR REPLACE FUNCTION max_premium()
RETURNS DECIMAL AS $$
DECLARE
    max_val DECIMAL;
BEGIN
    SELECT MAX(premium_amt) INTO max_val FROM Policy;
    RETURN max_val;
```

```
END;
$$ LANGUAGE plpgsql;

SELECT max_premium();
```

Q.2) Operating Systems

Write a program for Round Robin (RR) scheduling for a given time quantum. Accept no. of processes, arrival time and burst time for every process and time quantum. Calculate the waiting time of every process and Display the average waiting time.

```
#include <stdio.h>

int main() {
    int i, n, time = 0, remain, flag = 0, tq;
    int wait_time = 0, turn_around_time = 0, at[10], bt[10], rt[10];

    printf("Enter Number of Process: ");
    scanf("%d", &n);

    remain = n;

    for(i = 0; i < n; i++) {
        printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);
        scanf("%d %d", &at[i], &bt[i]);
        rt[i] = bt[i];
    }

    printf("Enter Time Quantum: ");
    scanf("%d", &tq);

    printf("\nProcess\t| Turnaround Time | Waiting Time\n");

    for(time = 0, i = 0; remain != 0;) {
        if(rt[i] <= tq && rt[i] > 0) {
            time += rt[i];
            rt[i] = 0;
            flag = 1;
        } else if(rt[i] > 0) {
            rt[i] -= tq;
            time += tq;
        }

        if(rt[i] == 0 && flag == 1) {
            remain--;
            printf("P[%d]\t|\t%d\t|\t%d\n", i + 1, time - at[i], time - at[i] - bt[i]);
            wait_time += time - at[i] - bt[i];
            turn_around_time += time - at[i];
            flag = 0;
        }

        if(i == n - 1)
            i = 0;
        else if(at[i + 1] <= time)
            i++;
        else
            i = 0;
    }

    printf("\nAverage Waiting Time= %.2f\n", wait_time * 1.0 / n);

    return 0;
}
```

Practical Slip 8

Q.1) Database Management System (PostgreSQL)

Consider the following database: Item (item no, name, quantity) Supplier (sno, name, city)

The relationship is as follows: Item-Supplier: many-to-many.

A) Create above database in PostgreSQL and insert sufficient records.

```
CREATE TABLE Item (
    item_no INT PRIMARY KEY,
    name VARCHAR(50),
    quantity INT
);

CREATE TABLE Supplier (
    sno INT PRIMARY KEY,
    name VARCHAR(50),
    city VARCHAR(50)
);
CREATE TABLE Item_Supplier (
    item_no INT REFERENCES Item(item_no),
    sno INT REFERENCES Supplier(sno),
    PRIMARY KEY (item_no, sno)
);

INSERT INTO Item VALUES
(1, 'Mouse', 500),
(2, 'Keyboard', 300),
(3, 'Monitor', 100);

INSERT INTO Supplier VALUES
(101, 'Manoj', 'Pune'),
(102, 'Rajesh', 'Mumbai'),
(103, 'Mohan', 'Delhi');

INSERT INTO Item_Supplier VALUES (1, 101), (2, 102), (1, 103);
```

i) Change the quantity for item "Mouse" to 800.

```
UPDATE Item SET quantity = 800 WHERE name = 'Mouse';
```

ii) List the details of the suppliers whose name begins with the alphabet "M".

```
SELECT * FROM Supplier WHERE name LIKE 'M%';
```

iii) Display the count of items.

```
SELECT COUNT(*) FROM Item;
```

iv) List the names of suppliers who do not live in "Pune".

```
SELECT name FROM Supplier WHERE city != 'Pune';
```

B) Write a stored function to find the minimum quantity of item.

```
CREATE OR REPLACE FUNCTION min_item_quantity()
RETURNS INT AS $$
```

```
DECLARE
    min_qty INT;
BEGIN
    SELECT MIN(quantity) INTO min_qty FROM Item;
    RETURN min_qty;
END;
$$ LANGUAGE plpgsql;

SELECT min_item_quantity();
```

Q.2) Operating Systems

Write a program to implement Bankers algorithm. Mention no. of processes and available resources. Calculate need matrix based on max and allocation matrix.

```
#include <stdio.h>
int main() {
    int n, m, i, j;
    int alloc[10][10], max[10][10], need[10][10];

    printf("Enter number of processes: ");
    scanf("%d", &n);

    printf("Enter number of resources: ");
    scanf("%d", &m);

    printf("Enter Allocation Matrix:\n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }
    printf("Enter Max Matrix:\n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }
    printf("\nNeed Matrix is:\n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++) {
            printf("%d\t", need[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Practical Slip 9

Q.1) Database Management System (PostgreSQL)

Consider the following database:Student (sno, s_name, s_class) s_class can be either "FY", "SY" or "TY" **Teacher (tno, t_name, yrs_experience)The relationship is as follows: Student-Teacher: M-M with descriptive attribute subject.**

A) Create above database in PostgreSQL and insert sufficient records.

```
CREATE TABLE Student (
    sno INT PRIMARY KEY,
    s_name VARCHAR(50),
    s_class VARCHAR(10) CHECK (s_class IN ('FY', 'SY', 'TY'))
);

CREATE TABLE Teacher (
    tno INT PRIMARY KEY,
    t_name VARCHAR(50),
    yrs_experience INT
);

CREATE TABLE Student_Teacher (
    sno INT REFERENCES Student(sno) ON DELETE CASCADE,
    tno INT REFERENCES Teacher(tno),
    subject VARCHAR(50),
    PRIMARY KEY (sno, tno)
);

INSERT INTO Student VALUES
(101, 'Rahul', 'TY'),
(102, 'Priya', 'SY'),
(103, 'Amit', 'FY');

INSERT INTO Teacher VALUES
(1, 'Prof. Deshmukh', 15),
(2, 'Prof. Kulkarni', 8);

INSERT INTO Student_Teacher VALUES (101, 1, 'Java'), (102, 2, 'C++'), (103, 1, 'Java');
```

i) Give class-wise number of students.

```
SELECT s_class, COUNT(*) FROM Student GROUP BY s_class;
```

ii) List all students studying in class "TY".

```
SELECT * FROM Student WHERE s_class = 'TY';
```

iii) Count the number of students who have taken subject "Java".*(Note: The specific subject name was missing in the slip, "Java" is used as an example).*

```
SELECT COUNT(*) FROM Student_Teacher WHERE subject = 'Java';
```

iv) Delete record of student whose sno = 101.

```
DELETE FROM Student WHERE sno = 101;
```

B) Write a stored function to take teacher name as input and returns the years of experience of that teacher.

```
CREATE OR REPLACE FUNCTION get_teacher_experience(t_name_input VARCHAR)
RETURNS INT AS $$
DECLARE
    exp INT;
BEGIN
    SELECT yrs_experience INTO exp FROM Teacher WHERE t_name = t_name_input;
    RETURN exp;
END;
$$ LANGUAGE plpgsql;

-- Example usage:
SELECT get_teacher_experience('Prof. Deshmukh');
```

Q.2) Operating Systems

Consider a system with 'm' processes and 'n' resource types. Accept number of instances for each resource type. For each process accept the allocation and maximum requirement matrices. Write a program to display the contents of need matrix.

```
#include <stdio.h>

int main() {
    int n, m, i, j;
    int alloc[10][10], max[10][10], need[10][10];

    printf("Enter number of processes (m): ");
    scanf("%d", &m);

    printf("Enter number of resource types (n): ");
    scanf("%d", &n);

    // Note: The problem asks to accept available resources ("instances"),
    // but only asks to *calculate* the Need matrix, which depends only on Max and Allocation.
    // We will accept Max and Allocation as requested.

    printf("Enter Allocation Matrix (m x n):\n");
    for(i = 0; i < m; i++) {
        printf("Process P%d: ", i);
        for(j = 0; j < n; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }

    printf("Enter Max Matrix (m x n):\n");
    for(i = 0; i < m; i++) {
        printf("Process P%d: ", i);
        for(j = 0; j < n; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    // Calculate Need = Max - Allocation
    for(i = 0; i < m; i++) {
        for(j = 0; j < n; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    printf("\nNeed Matrix:\n");
    for(i = 0; i < m; i++) {
        printf("P%d: ", i);
        for(j = 0; j < n; j++) {
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Practical Slip 10

Q.1) Database Management System (PostgreSQL)

Consider the following database: Account (acct_no, acct_type, balance, branch_name) Customer (cust_no, cust_name, cust_city)

Relationships: Customer-Account: 1-M. acct_type can be "saving" or "current"

A) Create above database in PostgreSQL and insert sufficient records.

```
CREATE TABLE Customer (
    cust_no INT PRIMARY KEY,
    cust_name VARCHAR(50),
    cust_city VARCHAR(50)
);

CREATE TABLE Account (
    acct_no INT PRIMARY KEY,
    acct_type VARCHAR(20) CHECK (acct_type IN ('saving', 'current')),
    balance DECIMAL(12, 2),
    branch_name VARCHAR(50),
    cust_no INT REFERENCES Customer(cust_no)
);

INSERT INTO Customer VALUES
(1, 'Rahul', 'Pune'),
(2, 'Riya', 'Mumbai'),
(3, 'Raj', 'Pune');

INSERT INTO Account VALUES
(101, 'saving', 600000.00, 'M.G.Road', 1),
(102, 'current', 40000.00, 'FC Road', 2),
(103, 'saving', 550000.00, 'M.G.Road', 3);
```

i) Display information of all saving accounts having balance > 500,000

```
SELECT * FROM Account WHERE acct_type = 'saving' AND balance > 500000;
```

ii) Count customers whose name starts with 'r'.

```
SELECT COUNT(*) FROM Customer WHERE cust_name ILIKE 'r%';
```

iii) Find the total balance at branch "M.G.Road".

```
SELECT SUM(balance) FROM Account WHERE branch_name = 'M.G.Road';
```

iv) Delete the record whose cust name is "Rahul".

```
DELETE FROM Customer WHERE cust_name = 'Rahul';
```

B) Write a stored function using cursors to print names of all customers from city "Pune".

```
CREATE OR REPLACE FUNCTION display_pune_customers()
RETURNS VOID AS $$
DECLARE
    cust_cursor CURSOR FOR SELECT cust_name FROM Customer WHERE cust_city = 'Pune';
    c_name VARCHAR;
```

```
BEGIN
    OPEN cust_cursor;
    LOOP
        FETCH cust_cursor INTO c_name;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'Customer Name: %', c_name;
    END LOOP;
    CLOSE cust_cursor;
END;
$$ LANGUAGE plpgsql;

SELECT display_pune_customers();
```

Q.2) Operating Systems

Write a Program to implement following functionality: Accept Available. Display Allocation, Max. Display the contents of need matrix.

(Note: The slip provides specific data for P1-P5, but the code below is general enough to solve it. You can input the table values when running the program).

```
#include <stdio.h>

int main() {
    int n, m, i, j;
    int alloc[10][10], max[10][10], need[10][10], avail[10];

    // Based on the slip table, m=5 (Processes P1-P5) and n=3 (Resources A, B, C)
    printf("Enter number of processes (e.g., 5): ");
    scanf("%d", &m);

    printf("Enter number of resources (e.g., 3): ");
    scanf("%d", &n);

    printf("Enter Available Resources (A B C): ");
    for(j = 0; j < n; j++) {
        scanf("%d", &avail[j]);
    }

    printf("Enter Allocation Matrix (P1 to P%d):\n", m);
    for(i = 0; i < m; i++) {
        printf("For P%d: ", i + 1);
        for(j = 0; j < n; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }

    printf("Enter Max Matrix (P1 to P%d):\n", m);
    for(i = 0; i < m; i++) {
        printf("For P%d: ", i + 1);
        for(j = 0; j < n; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    // Output Section
    printf("\n--- Allocation Matrix ---\n");
    for(i = 0; i < m; i++) {
        for(j = 0; j < n; j++) printf("%d ", alloc[i][j]);
        printf("\n");
    }

    printf("\n--- Max Matrix ---\n");
    for(i = 0; i < m; i++) {
        for(j = 0; j < n; j++) printf("%d ", max[i][j]);
        printf("\n");
    }

    printf("\n--- Need Matrix (Max - Allocation) ---\n");
    for(i = 0; i < m; i++) {
        for(j = 0; j < n; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
            printf("%d ", need[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Practical Slip 11

Q.1) Database Management System (PostgreSQL)

Consider the following database: Bus (Bus_no, capacity, depot_name) Route (Route_no, source, destination, no_of_stations) Relationship: Bus-Route: M-1. Bus capacity is not null.

A) Create above database in PostgreSQL and insert sufficient records.

```
CREATE TABLE Route (
   Route_no INT PRIMARY KEY,
   source VARCHAR(50),
   destination VARCHAR(50),
   no_of_stations INT
);

CREATE TABLE Bus (
   Bus_no INT PRIMARY KEY,
   capacity INT NOT NULL,
   depot_name VARCHAR(50),
   Route_no INT REFERENCES Route(Route_no)
);

INSERT INTO Route VALUES
(101, 'Swargate', 'Katraj', 12),
(102, 'Shivajinagar', 'Kothrud', 8),
(103, 'Pune Station', 'Hadapsar', 15);

INSERT INTO Bus VALUES
(1, 50, 'Swargate Depot', 101),
(2, 40, 'Kothrud Depot', 102),
(3, 55, 'Hadapsar Depot', 103),
(4, 50, 'Swargate Depot', 101);
```

i) List all buses which belongs to depot "Swargate Depot".

```
SELECT * FROM Bus WHERE depot_name = 'Swargate Depot';
```

ii) Delete Bus details whose Bus number is 2.

```
DELETE FROM Bus WHERE Bus_no = 2;
```

iii) List the route details having number of stations > 10.

```
SELECT * FROM Route WHERE no_of_stations > 10;
```

iv) List all routes starting from Station "Swargate".

```
SELECT * FROM Route WHERE source = 'Swargate';
```

B) Write a stored function using cursors to accept route_no from the user and display number of stations of that route.

```
CREATE OR REPLACE FUNCTION get_stations_count(r_no INT)
RETURNS VOID AS $$
DECLARE
   r_cursor CURSOR FOR SELECT no_of_stations FROM Route WHERE Route_no = r_no;
   stations INT;
BEGIN
```

```
    OPEN r_cursor;
    FETCH r_cursor INTO stations;
    IF FOUND THEN
        RAISE NOTICE 'Number of Stations: %', stations;
    ELSE
        RAISE NOTICE 'Route not found';
    END IF;
    CLOSE r_cursor;
END;
$$ LANGUAGE plpgsql;

SELECT get_stations_count(101);
```

Q.2) Operating Systems

Write a program to simulate Non-pre-emptive Shortest Job First (SJF) - scheduling. Accept no. of processes, arrival time and burst time from user. The output should be the waiting time for each process. Also find the average waiting time.

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

struct Process {
    int id;
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
    bool is_completed;
};

int main() {
    int n, completed = 0, current_time = 0;
    float total_wt = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].is_completed = false;
    }

    while (completed != n) {
        int idx = -1;
        int min_bt = INT_MAX;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= current_time && !p[i].is_completed) {
                if (p[i].bt < min_bt) {
                    min_bt = p[i].bt;
                    idx = i;
                }
            }
        }

        if (idx != -1) {
            p[idx].ct = current_time + p[idx].bt;
            p[idx].tat = p[idx].ct - p[idx].at;
            p[idx].wt = p[idx].tat - p[idx].bt;
            total_wt += p[idx].wt;
            p[idx].is_completed = true;
            completed++;
            current_time = p[idx].ct;
        } else {
            current_time++;
        }
    }

    printf("\nPID\tAT\tBT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].wt);
    }

    printf("\nAverage Waiting Time: %.2f\n", total_wt / n);

    return 0;
}
```

Practical Slip 12

Q.1) Database Management System (PostgreSQL)

Consider the following database: Game (gcode, gname, noofplayers, coachname, captain_name) Player (pno, pname)

There exists a one-to-many relationship between Game and Player.

A) Create above database in PostgreSQL and insert sufficient records.

```
CREATE TABLE Game (
    gcode INT PRIMARY KEY,
    gname VARCHAR(50),
    noofplayers INT,
    coachname VARCHAR(50),
    captain_name VARCHAR(50)
);

CREATE TABLE Player (
    pno INT PRIMARY KEY,
    pname VARCHAR(50),
    gcode INT REFERENCES Game(gcode)
);

INSERT INTO Game VALUES
(1, 'Football', 11, 'Zidane', 'Messi'),
(2, 'Hockey', 11, 'Ravi Shastri', 'Manpreet'),
(3, 'Kho Kho', 9, 'Pawar', 'Rahul');

INSERT INTO Player VALUES
(101, 'Rohan', 1),
(102, 'Sohan', 2),
(103, 'Mohan', 3),
(104, 'Amit', 1);
```

i) Display all game names that ends with "ball".

```
SELECT gname FROM Game WHERE gname LIKE '%ball';
```

ii) Give the average number of players.

```
SELECT AVG(noofplayers) FROM Game;
```

iii) Display all details of game "Kho Kho".

```
SELECT * FROM Game WHERE gname = 'Kho Kho';
```

iv) Update the coach name from "Ravi Shastri" to "Dravid" for game "Hockey".

```
UPDATE Game SET coachname = 'Dravid' WHERE gname = 'Hockey';
```

B) Create a stored procedure named as "deleterecords" for deleting the Game record having coach name "Pawar".

```
CREATE OR REPLACE PROCEDURE deleterecords(c_name VARCHAR)
```

```
LANGUAGE plpgsql
AS $$
BEGIN
    -- First delete players associated with the game to satisfy FK constraint
    DELETE FROM Player WHERE gcode IN (SELECT gcode FROM Game WHERE coachname = c_name);
    -- Then delete the game
    DELETE FROM Game WHERE coachname = c_name;
END;
$$;

CALL deleterecords('Pawar');
```

Q.2) Operating Systems

Write a program to simulate Non-pre-emptive Shortest Job First (SJF) CPU- scheduling. Accept no. of processes, arrival time and burst time from user. The output should be the turnaround time for each process. Also find the average turnaround time.

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

struct Process {
    int id;
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
    bool is_completed;
};

int main() {
    int n, completed = 0, current_time = 0;
    float total_tat = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].is_completed = false;
    }

    while (completed != n) {
        int idx = -1;
        int min_bt = INT_MAX;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= current_time && !p[i].is_completed) {
                if (p[i].bt < min_bt) {
                    min_bt = p[i].bt;
                    idx = i;
                }
            }
        }

        if (idx != -1) {
            p[idx].ct = current_time + p[idx].bt;
            p[idx].tat = p[idx].ct - p[idx].at;
            p[idx].wt = p[idx].tat - p[idx].bt;
            total_tat += p[idx].tat;
            p[idx].is_completed = true;
            completed++;
            current_time = p[idx].ct;
        } else {
            current_time++;
        }
    }

    printf("\nPID\tAT\tBT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].tat);
    }

    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);

    return 0;
}
```

Practical Slip 13

Q.1) Database Management System (PostgreSQL)

Consider the following database: Item (item_no, name, quantity, rate) Supplier (s_no, name, city, contact)

The relationship is as follows: Item-Supplier: many-to-many.

A) Create above database in PostgreSQL and insert sufficient records.

SQL
```
CREATE TABLE Item (
    item_no INT PRIMARY KEY,
    name VARCHAR(50),
    quantity INT,
    rate DECIMAL(10, 2)
);

CREATE TABLE Supplier (
    s_no INT PRIMARY KEY,
    name VARCHAR(50),
    city VARCHAR(50),
    contact VARCHAR(15)
);

CREATE TABLE Item_Supplier (
    item_no INT REFERENCES Item(item_no),
    s_no INT REFERENCES Supplier(s_no),
    PRIMARY KEY (item_no, s_no)
);

INSERT INTO Item VALUES
(1, 'Mouse', 40, 150.00),
(2, 'Keyboard', 25, 400.00),
(3, 'Pen Drive', 50, 450.00);

INSERT INTO Supplier VALUES
(101, 'Prasad', 'Pune', '9876543210'),
(102, 'Raj', 'Mumbai', '9123456789'),
(103, 'Prakash', 'Pune', '8888888888');

INSERT INTO Item_Supplier VALUES (1, 101), (2, 102), (3, 103);
```

i) List the details of the suppliers whose name begins with the alphabet "P".

SQL
```
SELECT * FROM Supplier WHERE name LIKE 'P%';
```

ii) Delete record of item_no 2.

SQL
```
DELETE FROM Item WHERE item_no = 2;
```

iii) Display the count of items with rate > 50Rs.

SQL
```
SELECT COUNT(*) FROM Item WHERE rate > 50;
```

iv) List the names of suppliers live in "Pune" city.

SQL
```sql
SELECT name FROM Supplier WHERE city = 'Pune';
```

B) Write a function to find the details of items whose quantity is greater than 30.

SQL
```sql
CREATE OR REPLACE FUNCTION get_items_qty_gt_30()
RETURNS SETOF Item AS $$
BEGIN
    RETURN QUERY SELECT * FROM Item WHERE quantity > 30;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM get_items_qty_gt_30();
```

Q.2) Operating Systems

Write a program to simulate FCFS CPU-scheduling. Accept no. of Processes, arrival time and burst time from user. The output should give Gantt chart, and waiting time for each process. Also find the average waiting time.

C
```c
#include <stdio.h>

struct Process {
    int id;
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
};

void sortProcs(struct Process p[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].at > p[j + 1].at) {
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n;
    float total_wt = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
    }
    sortProcs(p, n);

    int current_time = 0;
    printf("\n--- Gantt Chart ---\n");
    for (int i = 0; i < n; i++) {
        if (current_time < p[i].at) {
            current_time = p[i].at;
        }
        printf("| P%d ", p[i].id);

        p[i].ct = current_time + p[i].bt;
        p[i].tat = p[i].ct - p[i].at;
        p[i].wt = p[i].tat - p[i].bt;
        total_wt += p[i].wt;

        current_time = p[i].ct;
    }
    printf("|\n");

    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
    }

    printf("\nAverage Waiting Time: %.2f\n", total_wt / n);

    return 0;
}
```

Practical Slip 14

Q.1) Database Management System (PostgreSQL)

Consider the following database: Book (Book_no, title, author, price, year_published) Customer (cid, cname, addr)

Relation between Book and Customer is Many to Many with quantity as descriptive attribute.

A) Create above database in PostgreSQL and insert sufficient records.

SQL
```
CREATE TABLE Book (
    Book_no INT PRIMARY KEY,
    title VARCHAR(100),
    author VARCHAR(50),
    price DECIMAL(10, 2),
    year_published INT
);

CREATE TABLE Customer (
    cid INT PRIMARY KEY,
    cname VARCHAR(50),
    addr VARCHAR(50)
);

CREATE TABLE Book_Customer (
    Book_no INT REFERENCES Book(Book_no),
    cid INT REFERENCES Customer(cid),
    quantity INT,
    PRIMARY KEY (Book_no, cid)
);

INSERT INTO Book VALUES
(101, 'The Alchemist', 'Paulo Coelho', 300.00, 2024),
(102, 'Harry Potter', 'J.K. Rowling', 500.00, 2001),
(103, 'Wings of Fire', 'A.P.J. Abdul Kalam', 250.00, 2024);

INSERT INTO Customer VALUES
(1, 'Rohan', 'Pune'),
(2, 'Sohan', 'Mumbai');

INSERT INTO Book_Customer VALUES (101, 1, 2), (102, 2, 1);
```

i) Display customer details staying at "Pune".

SQL
```
SELECT * FROM Customer WHERE addr = 'Pune';
```

ii) Display author wise details of book.

SQL
```
SELECT * FROM Book ORDER BY author;
```

iii) Display the average price of a book.

SQL

```
SELECT AVG(price) FROM Book;
```

iv) Delete the record from book table with Book_no 102.

SQL
```
DELETE FROM Book_Customer WHERE Book_no = 102; -- Delete dependency first
DELETE FROM Book WHERE Book_no = 102;
```

B) Write a function, to define a cursor to print the details of the Books published in year 2024.

SQL
```
CREATE OR REPLACE FUNCTION list_books_2024()
RETURNS VOID AS $$
DECLARE
    book_cursor CURSOR FOR SELECT title, author, price FROM Book WHERE year_published = 2024;
    rec RECORD;
BEGIN
    OPEN book_cursor;
    LOOP
        FETCH book_cursor INTO rec;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'Title: %, Author: %, Price: %', rec.title, rec.author, rec.price;
    END LOOP;
    CLOSE book_cursor;
END;
$$ LANGUAGE plpgsql;

SELECT list_books_2024();
```

Q.2) Operating Systems

Write a program to simulate FCFS CPU-scheduling. Accept no. of Processes, arrival time and burst time from user. The output should give Gantt chart, and turn around time for each process. Also find the average turn around time.

C
```
#include <stdio.h>

struct Process {
    int id;
    int at;
    int bt;
    int ct;
    int tat;
    int wt;
};

void sortProcs(struct Process p[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].at > p[j + 1].at) {
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n;
    float total_tat = 0;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
    }

    sortProcs(p, n);

    int current_time = 0;
    printf("\n--- Gantt Chart ---\n");
    for (int i = 0; i < n; i++) {
        if (current_time < p[i].at) {
            current_time = p[i].at;
        }
        printf("| P%d ", p[i].id);

        p[i].ct = current_time + p[i].bt;
        p[i].tat = p[i].ct - p[i].at;
        p[i].wt = p[i].tat - p[i].bt;
        total_tat += p[i].tat;

        current_time = p[i].ct;
    }
    printf("|\n");

    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);
    }

    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);

    return 0;

}
```

Practical Slip 15

Q.1) Database Management System (PostgreSQL)

Consider the following database: Sales_order(s_orderno, s_order_date, order_amt) Client(client_no, name, address)

The relationship is as follows: Client and Sales_order: one-many. order_amt should be > 0

A) Create above database in PostgreSQL and insert sufficient records.

SQL
```
CREATE TABLE Client (
    client_no INT PRIMARY KEY,
    name VARCHAR(50),
    address VARCHAR(50)
);

CREATE TABLE Sales_order (
    s_orderno INT PRIMARY KEY,
    s_order_date DATE,
    order_amt DECIMAL(10, 2) CHECK (order_amt > 0),
    client_no INT REFERENCES Client(client_no)
);

INSERT INTO Client VALUES
(1, 'Amit', 'Nasik'),
(2, 'Sumit', 'Pune'),
(3, 'Raj', 'Nasik');

INSERT INTO Sales_order VALUES
(101, '2023-01-15', 5000.00, 1),
(102, '2024-02-20', 12000.00, 2),
(103, '2023-05-10', 8000.00, 1);
```

i) Display all sale records having order date before "2024-01-01". *(Note: Date assumed as it was blank in the slip)*

SQL
```
SELECT * FROM Sales_order WHERE s_order_date < '2024-01-01';
```

ii) Find maximum sales order amount.

SQL
```
SELECT MAX(order_amt) FROM Sales_order;
```

iii) Update the client address of all clients from "Nasik" to "Ahilyanagar".

SQL
```
UPDATE Client SET address = 'Ahilyanagar' WHERE address = 'Nasik';
```

iv) Add column order_status to the Sales_order table.

SQL
```
ALTER TABLE Sales_order ADD COLUMN order_status VARCHAR(20);
```

B) Create a stored procedure named as "addrecords" for adding new sales order records.

SQL

```sql
CREATE OR REPLACE PROCEDURE addrecords(
    p_no INT,
    p_date DATE,
    p_amt DECIMAL,
    p_cno INT
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO Sales_order (s_orderno, s_order_date, order_amt, client_no)
    VALUES (p_no, p_date, p_amt, p_cno);
END;
$$;

CALL addrecords(104, '2024-03-15', 7500.00, 2);
```

Q.2) Operating Systems

Write a program to simulate Pre-emptive Shortest Job First (SJF) - scheduling. Accept no. of processes, arrival time and burst time from user. The output should be the waiting time for each process. Also find the average waiting time.

C

```c
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

struct Process {
    int id;
    int at;
    int bt;
    int rt;
    int wt;
    int tat;
    int ct;
};

int main() {
    int n, completed = 0, current_time = 0, min_rt_idx = -1, min_rt = INT_MAX;
    float total_wt = 0;
    bool found = false;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].rt = p[i].bt;
    }

    while (completed != n) {
        min_rt = INT_MAX;
        min_rt_idx = -1;
        found = false;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= current_time && p[i].rt > 0) {
                if (p[i].rt < min_rt) {
                    min_rt = p[i].rt;
                    min_rt_idx = i;
                    found = true;
                }
            }
        }

        if (!found) {
            current_time++;
            continue;
        }

        p[min_rt_idx].rt--;
        current_time++;

        if (p[min_rt_idx].rt == 0) {
            completed++;
            p[min_rt_idx].ct = current_time;
            p[min_rt_idx].tat = p[min_rt_idx].ct - p[min_rt_idx].at;
            p[min_rt_idx].wt = p[min_rt_idx].tat - p[min_rt_idx].bt;
            total_wt += p[min_rt_idx].wt;
        }
    }

    printf("\nPID\tAT\tBT\tWT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].wt);
    }

    printf("\nAverage Waiting Time: %.2f\n", total_wt / n);

    return 0;
}
```

Practical Slip 16

Q.1) Database Management System (PostgreSQL)

Consider the following database: Car (car_code, c_name, c_price, color_type) color_type can be "metallic" or "solid" **Customer (cust_code, cust_name, cust_address)**

The relationship is as follows: Customer and car: one-to-many.

A) Create above database in PostgreSQL and insert sufficient records.

SQL
```
CREATE TABLE Customer (
    cust_code INT PRIMARY KEY,
    cust_name VARCHAR(50),
    cust_address VARCHAR(50)
);

CREATE TABLE Car (
    car_code INT PRIMARY KEY,
    c_name VARCHAR(50),
    c_price DECIMAL(12, 2),
    color_type VARCHAR(10) CHECK (color_type IN ('metallic', 'solid')),
    cust_code INT REFERENCES Customer(cust_code)
);

INSERT INTO Customer VALUES
(1, 'Bharat', 'ShivajiNagar'),
(2, 'Rohan', 'Kothrud'),
(3, 'Bipasha', 'Pune');

INSERT INTO Car VALUES
(101, 'Ferrari', 4500000.00, 'metallic', 1),
(102, 'Honda City', 1200000.00, 'solid', 2),
(103, 'Swift', 800000.00, 'metallic', 1),
(104, 'Ferrari', 4000000.00, 'solid', 3);
```

i) Find the names of all Customers whose name start with "B".

SQL
```
SELECT cust_name FROM Customer WHERE cust_name LIKE 'B%';
```

ii) Count the number of "metallic" cars.

SQL
```
SELECT COUNT(*) FROM Car WHERE color_type = 'metallic';
```

iii) Give the list of all customers staying in ShivajiNagar.

SQL
```
SELECT * FROM Customer WHERE cust_address = 'ShivajiNagar';
```

iv) Increase the price of all "Ferrari" cars by 15%.

SQL
```
UPDATE Car SET c_price = c_price + (c_price * 0.15) WHERE c_name = 'Ferrari';
```

B) write a stored function to display details of all metallic coloured cars having price in the range 100000 to 500000.

SQL
```
CREATE OR REPLACE FUNCTION display_metallic_cars()
RETURNS VOID AS $$
DECLARE
    car_cursor CURSOR FOR SELECT * FROM Car WHERE color_type = 'metallic' AND c_price BETWEEN 100000 AND
500000;
    rec RECORD;
BEGIN
    OPEN car_cursor;
    LOOP
        FETCH car_cursor INTO rec;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'Car: %, Price: %', rec.c_name, rec.c_price;
    END LOOP;
    CLOSE car_cursor;
END;
$$ LANGUAGE plpgsql;

SELECT display_metallic_cars();
```

Q.2) Operating Systems

Write a program to simulate Pre-emptive Shortest Job First (SJF) - scheduling. Accept no. of processes, arrival time and burst time from user. The output should be the turn around time for each process. Also find the average turn around time.

C
```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

struct Process {
    int id;
    int at;
    int bt;
    int rt;
    int wt;
    int tat;
    int ct;
};

int main() {
    int n, completed = 0, current_time = 0, min_rt_idx = -1, min_rt = INT_MAX;
    float total_tat = 0;
    bool found = false;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].rt = p[i].bt;
    }

    while (completed != n) {
        min_rt = INT_MAX;
        min_rt_idx = -1;
        found = false;

        for (int i = 0; i < n; i++) {
            if (p[i].at <= current_time && p[i].rt > 0) {
                if (p[i].rt < min_rt) {
                    min_rt = p[i].rt;
                    min_rt_idx = i;
                    found = true;
                }
            }
        }

        if (!found) {
            current_time++;
            continue;
        }

        p[min_rt_idx].rt--;
        current_time++;

        if (p[min_rt_idx].rt == 0) {
            completed++;
            p[min_rt_idx].ct = current_time;
            p[min_rt_idx].tat = p[min_rt_idx].ct - p[min_rt_idx].at;
            p[min_rt_idx].wt = p[min_rt_idx].tat - p[min_rt_idx].bt;
            total_tat += p[min_rt_idx].tat;
        }
    }

    printf("\nPID\tAT\tBT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].tat);
    }

    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);

    return 0;
}
```

Practical Slip 17

Q.1) Database Management System (PostgreSQL)

Consider the following database: Property (pno, description, area, rate) Owner (owner_name, city, phno) The relationship is as follows: Owner and Property: One to Many. Rate should be > 0.

A) Create above database in PostgreSQL and insert sufficient records.

SQL
```
CREATE TABLE Owner (
    owner_id INT PRIMARY KEY,
    owner_name VARCHAR(50),
    city VARCHAR(50),
    phno VARCHAR(15)
);

CREATE TABLE Property (
    pno INT PRIMARY KEY,
    description VARCHAR(100),
    area VARCHAR(50),
    rate DECIMAL(10, 2) CHECK (rate > 0),
    owner_id INT REFERENCES Owner(owner_id)
);

INSERT INTO Owner VALUES
(1, 'Dr. Vikas', 'Pune', '9876543210'),
(2, 'Anita', 'Mumbai', '9988776655'),
(3, 'Rahul', 'Pune', '8888888888');

INSERT INTO Property VALUES
(101, '2BHK Flat', 'Kothrud', 5000000.00, 1),
(102, 'Shop', 'Deccan', 8000000.00, 1),
(103, 'Plot', 'Baner', 6000000.00, 2);
```

i) List the name of owners that ends with letter 'a'.

```
SELECT owner_name FROM Owner WHERE owner_name LIKE '%a';
```

ii) Display the average rate of a property.

```
SELECT AVG(rate) FROM Property;
```

iii) Update the phone Number of "Dr. Vikas" to 8856916175.

```
UPDATE Owner SET phno = '8856916175' WHERE owner_name = 'Dr. Vikas';
```

iv) Display area wise property details.

```
SELECT * FROM Property ORDER BY area;
```

B) Create a stored function named as "min_price" which will find minimum rate of property.

SQL
```
CREATE OR REPLACE FUNCTION min_price()
RETURNS DECIMAL AS $$
DECLARE
    min_val DECIMAL;
```

```
BEGIN
    SELECT MIN(rate) INTO min_val FROM Property;
    RETURN min_val;
END;
$$ LANGUAGE plpgsql;

SELECT min_price();
```

Q.2) Operating Systems

Write a program for Round Robin scheduling for given time quantum. Accept no. of processes, arrival time and burst time for each process and time quantum. The output should give the waiting time for each process. Also display the average waiting time.

C

```
#include <stdio.h>

int main() {
    int i, n, time = 0, remain, flag = 0, tq;
    int wait_time = 0, at[10], bt[10], rt[10];

    printf("Enter Number of Process: ");
    scanf("%d", &n);

    remain = n;

    for(i = 0; i < n; i++) {
        printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);
        scanf("%d %d", &at[i], &bt[i]);
        rt[i] = bt[i];
    }

    printf("Enter Time Quantum: ");
    scanf("%d", &tq);

    printf("\nProcess\t| Waiting Time\n");
    for(time = 0, i = 0; remain != 0;) {
        if(rt[i] <= tq && rt[i] > 0) {
            time += rt[i];
            rt[i] = 0;
            flag = 1;
        } else if(rt[i] > 0) {
            rt[i] -= tq;
            time += tq;
        }

        if(rt[i] == 0 && flag == 1) {
            remain--;
            int wt = time - at[i] - bt[i];
            printf("P[%d]\t|\t%d\n", i + 1, wt);
            wait_time += wt;
            flag = 0;
        }

        if(i == n - 1)
            i = 0;
        else if(at[i + 1] <= time)
            i++;
        else
            i = 0;
    }
    printf("\nAverage Waiting Time= %.2f\n", wait_time * 1.0 / n);

    return 0;
}
```

Practical Slip 18

Q.1) Database Management System (PostgreSQL)

Consider the following database: Employee (emp_no, emp_name, city, designation, salary) Project (project_no, project_name, status, start_date)

The relationship is as follows: Employee and Project: many-to-one.

A) Create above database in PostgreSQL and insert sufficient records.

SQL

```
CREATE TABLE Project (
   project_no INT PRIMARY KEY,
   project_name VARCHAR(50),
   status VARCHAR(20),
   start_date DATE
);

CREATE TABLE Employee (
   emp_no INT PRIMARY KEY,
   emp_name VARCHAR(50),
   city VARCHAR(50),
   designation VARCHAR(50),
   salary DECIMAL(10, 2),
   project_no INT REFERENCES Project(project_no)
);

INSERT INTO Project VALUES
(10, 'Banking App', 'In progress', '2023-01-01'),
(20, 'E-Commerce', 'Complete', '2022-05-01');

INSERT INTO Employee VALUES
(1, 'Amit', 'Pune', 'Developer', 50000.00, 10),
(2, 'Sumit', 'Mumbai', 'Tester', 40000.00, 10),
(3, 'Raj', 'Pune', 'Manager', 80000.00, 20);
```

i) Add constraint status. The value of status should be "Complete", "In progress".

```
ALTER TABLE Project ADD CONSTRAINT chk_status CHECK (status IN ('Complete', 'In progress'));
```

ii) Count the number of Projects which are "in progress".

```
SELECT COUNT(*) FROM Project WHERE status = 'In progress';
```

iii) Increase the salaries of all employees working on project 10 by 5%.

```
UPDATE Employee SET salary = salary + (salary * 0.05) WHERE project_no = 10;
```

iv) Display names of all completed projects.

```
SELECT project_name FROM Project WHERE status = 'Complete';
```

B) Create a stored function named as "max_salary" which will find maximum salary of an employee.

```
CREATE OR REPLACE FUNCTION max_salary()
RETURNS DECIMAL AS $$
DECLARE
   max_val DECIMAL;
BEGIN
```

```
    SELECT MAX(salary) INTO max_val FROM Employee;
    RETURN max_val;
END;
$$ LANGUAGE plpgsql;

SELECT max_salary();
```

Q.2) Operating Systems

Write a program for Round Robin scheduling for given time quantum. Accept no. of processes, arrival time and burst time for each process and time quantum. Calculate the turn around time for each process. Display the average turn around time.

```
#include <stdio.h>
int main() {

    int i, n, time = 0, remain, flag = 0, tq;
    int turn_around_time = 0, at[10], bt[10], rt[10];

    printf("Enter Number of Process: ");
    scanf("%d", &n);

    remain = n;

    for(i = 0; i < n; i++) {
        printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);
        scanf("%d %d", &at[i], &bt[i]);
        rt[i] = bt[i];
    }

    printf("Enter Time Quantum: ");
    scanf("%d", &tq);

    printf("\nProcess\t| Turnaround Time\n");

    for(time = 0, i = 0; remain != 0;) {
        if(rt[i] <= tq && rt[i] > 0) {
            time += rt[i];
            rt[i] = 0;
            flag = 1;
        } else if(rt[i] > 0) {
            rt[i] -= tq;
            time += tq;
        }

        if(rt[i] == 0 && flag == 1) {
            remain--;
            int tat = time - at[i];
            printf("P[%d]\t|\t%d\n", i + 1, tat);
            turn_around_time += tat;
            flag = 0;
        }

        if(i == n - 1)
            i = 0;
        else if(at[i + 1] <= time)
            i++;
        else
            i = 0;
    }

    printf("\nAverage Turnaround Time= %.2f\n", turn_around_time * 1.0 / n);

    return 0;
}
```

Practical Slip 19

Q.1) Database Management System (PostgreSQL)

Consider the following database: Project (pno, pname, start_date, budget, status) Project Status Constraints: C-completed, P-Progressive, I-Incomplete **Department (dno, dname, HOD, no_of_staff)**

The relationship is as follows: Project-Department Many to One.

A) Create above database in PostgreSQL and insert sufficient records.

SQL
```
CREATE TABLE Department (
    dno INT PRIMARY KEY,
    dname VARCHAR(50),
    HOD VARCHAR(50),
    no_of_staff INT
);

CREATE TABLE Project (
    pno INT PRIMARY KEY,
    pname VARCHAR(50),
    start_date DATE,
    budget DECIMAL(10, 2),
    status CHAR(1) CHECK (status IN ('C', 'P', 'I')),
    dno INT REFERENCES Department(dno)
);

INSERT INTO Department VALUES
(1, 'Computer', 'Dr. Patil', 15),
(2, 'Electronics', 'Dr. Joshi', 10);

INSERT INTO Project VALUES
(101, 'Library Mgmt', '2019-06-12', 40000.00, 'C', 1),
(102, 'AI System', '2020-01-01', 25000.00, 'P', 1),
(103, 'IoT Sensor', '2021-03-15', 50000.00, 'P', 2);
```

i) Display the project names that have start date as 12/6/2019.

SQL
```
SELECT pname FROM Project WHERE start_date = '2019-06-12';
```

ii) Display the total budget of projects.

SQL
```
SELECT SUM(budget) FROM Project;
```

iii) Display the HOD name of Computer department.

SQL
```
SELECT HOD FROM Department WHERE dname = 'Computer';
```

iv) Display all project names having budget more than 30000.

SQL
```
SELECT pname FROM Project WHERE budget > 30000;
```

B) Write a stored function using cursors to display names of all projects which are "in progress".

SQL
```
CREATE OR REPLACE FUNCTION list_progressive_projects()
RETURNS VOID AS $$
DECLARE
    proj_cursor CURSOR FOR SELECT pname FROM Project WHERE status = 'P';
    p_name VARCHAR;
BEGIN
    OPEN proj_cursor;
    LOOP
        FETCH proj_cursor INTO p_name;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'Project In Progress: %', p_name;
    END LOOP;
    CLOSE proj_cursor;
END;
$$ LANGUAGE plpgsql;

SELECT list_progressive_projects();
```

Q.2) Operating Systems

Write a program to simulate FCFS CPU-scheduling. Accept no. of Processes, arrival time and burst time from user. The output should give Gantt chart.

C
```
#include <stdio.h>

struct Process {
    int id;
    int at;
    int bt;
    int start_time;
    int ct;
};

void sortProcs(struct Process p[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (p[j].at > p[j + 1].at) {
                temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n, current_time = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for Process %d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
    }

    sortProcs(p, n);

    printf("\n--- Gantt Chart ---\n");
    printf("Time: %d", current_time);

    for (int i = 0; i < n; i++) {
        if (current_time < p[i].at) {
            printf(" -> [Idle] -> %d", p[i].at);
            current_time = p[i].at;
        }

        p[i].start_time = current_time;
        current_time += p[i].bt;
        p[i].ct = current_time;

        printf(" -> [P%d] -> %d", p[i].id, current_time);
    }
    printf("\n");

    return 0;
}
```

Practical Slip 20

Q.1) Database Management System (PostgreSQL)

Consider the following database: Bus (bus_no, capacity, depot_name) Driver (driver_no, driver_name, license_no, address, age)

The relationship is as follows: Bus and Driver: M-M with the descriptive attribute Date_of_duty.

A) Create above database in PostgreSQL and insert sufficient records.

SQL
```
CREATE TABLE Bus (
    bus_no INT PRIMARY KEY,
    capacity INT,
    depot_name VARCHAR(50)
);

CREATE TABLE Driver (
    driver_no INT PRIMARY KEY,
    driver_name VARCHAR(50),
    license_no VARCHAR(20),
    address VARCHAR(50),
    age INT
);

CREATE TABLE Bus_Driver (
    bus_no INT REFERENCES Bus(bus_no),
    driver_no INT REFERENCES Driver(driver_no),
    date_of_duty DATE,
    PRIMARY KEY (bus_no, driver_no, date_of_duty)
);

INSERT INTO Bus VALUES
(101, 30, 'Swargate'),
(102, 50, 'Shivajinagar');

INSERT INTO Driver VALUES
(1, 'Suresh', 'MH122020', 'Pune', 45),
(2, 'Ramesh', 'MH142021', 'Mumbai', 35),
(3, 'Sunil', 'MH122022', 'Pune', 42);

INSERT INTO Bus_Driver VALUES (101, 1, '2024-01-01'), (102, 2, '2024-01-02');
```

i) Find the number of buses having capacity more than 20.

SQL
```
SELECT COUNT(*) FROM Bus WHERE capacity > 20;
```

ii) Count number of drivers having age > 40.

SQL
```
SELECT COUNT(*) FROM Driver WHERE age > 40;
```

iii) Give the names of all drivers starting with 'S'.

SQL

```
SELECT driver_name FROM Driver WHERE driver_name LIKE 'S%';
```

iv) Display all bus details of "Swargate" depot.

SQL
```
SELECT * FROM Bus WHERE depot_name = 'Swargate';
```

B) Write a stored procedure to find maximum of two numbers.

SQL
```
CREATE OR REPLACE PROCEDURE find_max(n1 INT, n2 INT)
LANGUAGE plpgsql
AS $$
BEGIN
    IF n1 > n2 THEN
        RAISE NOTICE 'Maximum is: %', n1;
    ELSE
        RAISE NOTICE 'Maximum is: %', n2;
    END IF;
END;
$$;

CALL find_max(10, 20);
```

Q.2) Operating Systems

Write a program for Round Robin scheduling for given time quantum. Accept no. of processes, arrival time and burst time for each process and time quantum from user. Display the content of Gantt Chart.

C
```
#include <stdio.h>

struct Process {
    int id;
    int at;
    int bt;
    int rt;
};

int main() {
    int n, tq, i, time = 0, remain;

    printf("Enter number of processes: ");
    scanf("%d", &n);
    remain = n;

    struct Process p[n];

    for(i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for P%d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].rt = p[i].bt;
    }

    printf("Enter Time Quantum: ");
    scanf("%d", &tq);

    printf("\n--- Gantt Chart Content ---\n");
    printf("Time %d", time);

    int flag = 0;
    i = 0;
    // Simple logic to simulate the Gantt Chart flow
    while(remain != 0) {
        if(p[i].rt <= tq && p[i].rt > 0) {
            time += p[i].rt;
            p[i].rt = 0;
            flag = 1;
            printf(" -> [P%d] -> %d", p[i].id, time);
        } else if(p[i].rt > 0) {
            p[i].rt -= tq;
            time += tq;
            printf(" -> [P%d] -> %d", p[i].id, time);
        }

        if(p[i].rt == 0 && flag == 1) {
            remain--;
            flag = 0;
        }

        if(i == n - 1)
            i = 0;
        else if(p[i+1].at <= time)
            i++;
        else
            i = 0;
    }
    printf("\n");

    return 0;
}
```

Practical Slip 21

Q.1) Database Management System (PostgreSQL)

Consider the following database: Customer (cust_no, cust_name, city) Loan (loan_no, loan_amt)
loan_amt should be > 0

Relation between Customer and Loan is Many to Many.

A) Create above database in PostgreSQL and insert sufficient records.

SQL
```
CREATE TABLE Customer (
    cust_no INT PRIMARY KEY,
    cust_name VARCHAR(50),
    city VARCHAR(50)
);

CREATE TABLE Loan (
    loan_no INT PRIMARY KEY,
    loan_amt DECIMAL(12, 2) CHECK (loan_amt > 0)
);

CREATE TABLE Customer_Loan (
    cust_no INT REFERENCES Customer(cust_no),
    loan_no INT REFERENCES Loan(loan_no),
    PRIMARY KEY (cust_no, loan_no)
);

INSERT INTO Customer VALUES
(1, 'Amit', 'Pune'),
(2, 'Ajay', 'Mumbai'),
(3, 'Rahul', 'Pune');

INSERT INTO Loan VALUES
(101, 500000.00),
(102, 150000.00),
(103, 300000.00);

INSERT INTO Customer_Loan VALUES (1, 101), (2, 102), (3, 103);
```

i) List all customers whose name starts with 'A'.

SQL
```
SELECT * FROM Customer WHERE cust_name LIKE 'A%';
```

ii) Display city-wise customer names.

SQL
```
SELECT city, cust_name FROM Customer ORDER BY city;
```

iii) Display all loan numbers whose amount is more than 2 lakhs.

SQL
```
SELECT loan_no FROM Loan WHERE loan_amt > 200000;
```

iv) Change city 'Pune' to 'Mumbai' for customer.

SQL
```
UPDATE Customer SET city = 'Mumbai' WHERE city = 'Pune';
```

B) Write a stored function using cursors to display details of all customers sorted by city names.

SQL
```
CREATE OR REPLACE FUNCTION display_customers_by_city()
RETURNS VOID AS $$
DECLARE
    cust_cursor CURSOR FOR SELECT cust_name, city FROM Customer ORDER BY city;
    rec RECORD;
BEGIN
    OPEN cust_cursor;
    LOOP
        FETCH cust_cursor INTO rec;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'City: %, Name: %', rec.city, rec.cust_name;
    END LOOP;
    CLOSE cust_cursor;
END;
$$ LANGUAGE plpgsql;

SELECT display_customers_by_city();
```

Q.2) Operating Systems

Write a program to simulate Pre-emptive Shortest Job First (SJF) - scheduling. Accept no. of processes, arrival time and burst time from user. Display the content of Gantt Chart.

C
```
#include <stdio.h>
#include <limits.h>

struct Process {
    int id;
    int at;
    int bt;
    int rt;
};

int main() {
    int n, time = 0, completed = 0, min_rt, shortest = -1, finish_time;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for(int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for P%d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].rt = p[i].bt;
    }

    printf("\n--- Gantt Chart Content ---\n");
    printf("Time 0");

    int prev_proc = -1;

    while(completed != n) {
        min_rt = INT_MAX;
        shortest = -1;
        for(int i = 0; i < n; i++) {
            if(p[i].at <= time && p[i].rt < min_rt && p[i].rt > 0) {
                min_rt = p[i].rt;
                shortest = i;
            }
        }

        if(shortest == -1) {
            time++;
            continue;
        }

        // Print only when context switches or process completes (for readable Gantt)
        if(prev_proc != p[shortest].id && prev_proc != -1) {
            printf(" -> [P%d] -> %d", prev_proc, time);
        }
        prev_proc = p[shortest].id;

        p[shortest].rt--;
        time++;

        if(p[shortest].rt == 0) {
            completed++;
            printf(" -> [P%d] -> %d", p[shortest].id, time);
            prev_proc = -1; // Reset so next process prints start
        }
    }
    printf("\n");

    return 0;
}
```

Practical Slip 22

Q.1) Database Management System (PostgreSQL)

Consider the following database: Customer (cust_no, cust_name, city) product (product_no, pname, price) price should be > 0

Relation between Customer and product is Many to Many.

A) Create above database in PostgreSQL and insert sufficient records.

SQL

```
CREATE TABLE Customer (
    cust_no INT PRIMARY KEY,
    cust_name VARCHAR(50),
    city VARCHAR(50)
);

CREATE TABLE Product (
    product_no INT PRIMARY KEY,
    pname VARCHAR(50),
    price DECIMAL(10, 2) CHECK (price > 0)
);

CREATE TABLE Customer_Product (
    cust_no INT REFERENCES Customer(cust_no),
    product_no INT REFERENCES Product(product_no),
    PRIMARY KEY (cust_no, product_no)
);

INSERT INTO Customer VALUES
(1, 'Riya', 'Pune'),
(2, 'Sonia', 'Mumbai'),
(3, 'Pooja', 'Pune');

INSERT INTO Product VALUES
(101, 'Laptop', 50000.00),
(102, 'Mouse', 500.00),
(103, 'Mobile', 15000.00);

INSERT INTO Customer_Product VALUES (1, 101), (2, 103), (1, 102);
```

i) List all customers whose name ends with 'A'.

SQL

```
SELECT * FROM Customer WHERE cust_name LIKE '%a'; -- 'LIKE' is case insensitive in some configs, otherwise use ILIKE
'%a'
```

ii) Count number of products whose price is more than 1000.

SQL

```
SELECT COUNT(*) FROM Product WHERE price > 1000;
```

iii) Increase price of all products by 5%.

SQL

```
UPDATE Product SET price = price + (price * 0.05);
```

iv) Display details of customer who are from "Pune" city.

SQL
```
SELECT * FROM Customer WHERE city = 'Pune';
```

B) Create a stored procedure named as "addrecords" to add customer record.

SQL
```
CREATE OR REPLACE PROCEDURE addrecords(
    c_no INT,
    c_name VARCHAR,
    c_city VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO Customer (cust_no, cust_name, city)
    VALUES (c_no, c_name, c_city);
END;
$$;

CALL addrecords(4, 'Neha', 'Nashik');
```

Q.2) Operating Systems

Write a program to simulate Non-pre-emptive Shortest Job First (SJF) CPU-scheduling. Accept no. of processes, arrival time and burst time from user. Display the content of Gantt Chart.

C
```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

struct Process {
    int id;
    int at;
    int bt;
    bool is_completed;
};

int main() {
    int n, time = 0, completed = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    for(int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for P%d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].is_completed = false;
    }

    printf("\n--- Gantt Chart Content ---\n");
    printf("Time 0");

    while(completed != n) {
        int idx = -1;
        int min_bt = INT_MAX;

        for(int i = 0; i < n; i++) {
            if(p[i].at <= time && !p[i].is_completed) {
                if(p[i].bt < min_bt) {
                    min_bt = p[i].bt;
                    idx = i;
                }
            }
        }

        if(idx != -1) {
            time += p[idx].bt;
            p[idx].is_completed = true;
            completed++;
            printf(" -> [P%d] -> %d", p[idx].id, time);
        } else {
            time++;
        }
    }
    printf("\n");

    return 0;
}
```

Practical Slip 23

Q.1) Database Management System (PostgreSQL)

Consider the following database: Student (rno, name, city) Subject (subno, subname, teachername)

Relation between Student and Subject is Many to Many with descriptive attribute mark.

A) Create above database in PostgreSQL and insert sufficient records.

SQL
```
CREATE TABLE Student (
   rno INT PRIMARY KEY,
   name VARCHAR(50),
   city VARCHAR(50)
);

CREATE TABLE Subject (
   subno INT PRIMARY KEY,
   subname VARCHAR(50),
   teachername VARCHAR(50)
);

CREATE TABLE Student_Subject (
   rno INT REFERENCES Student(rno),
   subno INT REFERENCES Subject(subno),
   mark INT,
   PRIMARY KEY (rno, subno)
);

INSERT INTO Student VALUES
(1, 'Rohan', 'Pune'),
(2, 'Sohan', 'Mumbai'),
(3, 'Mohan', 'Pune');

INSERT INTO Subject VALUES
(101, 'OS', 'Prof. Kale'),
(102, 'DB', 'Prof. More'),
(103, 'Java', 'Prof. Kale');

INSERT INTO Student_Subject VALUES (1, 101, 80), (2, 102, 75), (3, 101, 85);
```

i) List all students from city "Pune".

SQL
```
SELECT * FROM Student WHERE city = 'Pune';
```

ii) Count number of subjects taught by "Prof. Kale".

SQL
```
SELECT COUNT(*) FROM Subject WHERE teachername = 'Prof. Kale';
```

iii) Display name of all teachers who teaches subject "OS".

SQL
```
SELECT teachername FROM Subject WHERE subname = 'OS';
```

iv) Delete record of a student named "Sohan".

SQL
```sql
DELETE FROM Student_Subject WHERE rno = (SELECT rno FROM Student WHERE name = 'Sohan');
DELETE FROM Student WHERE name = 'Sohan';
```

B) Create a stored procedure named as "addrecords" to add student record.

SQL
```sql
CREATE OR REPLACE PROCEDURE addrecords(
    r_no INT,
    s_name VARCHAR,
    s_city VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO Student (rno, name, city)
    VALUES (r_no, s_name, s_city);
END;
$$;

CALL addrecords(4, 'Gita', 'Nashik');
```

Q.2) Operating Systems

Write a program for Round Robin scheduling for given time quantum. Accept no. of processes, arrival time and burst time for each process and time quantum. Display the content of Gantt Chart.

C
```c
#include <stdio.h>

struct Process {
    int id;
    int at;
    int bt;
    int rt;
};

int main() {
    int n, tq, i, time = 0, remain;

    printf("Enter number of processes: ");
    scanf("%d", &n);
    remain = n;

    struct Process p[n];

    for(i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter AT and BT for P%d: ", i + 1);
        scanf("%d %d", &p[i].at, &p[i].bt);
        p[i].rt = p[i].bt;
    }

    printf("Enter Time Quantum: ");
    scanf("%d", &tq);

    printf("\n--- Gantt Chart Content ---\n");
    printf("Time %d", time);

    int flag = 0;
    i = 0;

    while(remain != 0) {
        if(p[i].rt <= tq && p[i].rt > 0) {
            time += p[i].rt;
            p[i].rt = 0;
            flag = 1;
            printf(" -> [P%d] -> %d", p[i].id, time);
        } else if(p[i].rt > 0) {
            p[i].rt -= tq;
            time += tq;
            printf(" -> [P%d] -> %d", p[i].id, time);
        }

        if(p[i].rt == 0 && flag == 1) {
            remain--;
            flag = 0;
        }

        if(i == n - 1)
            i = 0;
        else if(p[i+1].at <= time)
            i++;
        else
            i = 0;
    }
    printf("\n");

    return 0;
}
```

Practical Slip 24

Q.1) Database Management System (PostgreSQL)

Consider the following database: Book (bid, btitle, price, publication) Author (aid, aname, mobile number, city)

Relation between Author and Book is one to Many.

A) Create above database in PostgreSQL and insert sufficient records.

SQL
```
CREATE TABLE Author (
    aid INT PRIMARY KEY,
    aname VARCHAR(50),
    mobile_number VARCHAR(15),
    city VARCHAR(50)
);

CREATE TABLE Book (
    bid INT PRIMARY KEY,
    btitle VARCHAR(100),
    price DECIMAL(10, 2),
    publication VARCHAR(50),
    aid INT REFERENCES Author(aid)
);

INSERT INTO Author VALUES
(1, 'Sane Guruji', '9999888877', 'Pune'),
(2, 'Chetan Bhagat', '8888777766', 'Mumbai');

INSERT INTO Book VALUES
(101, 'Shyamchi Aai', 150.00, 'Prentice Hall', 1),
(102, 'Five Point Someone', 200.00, 'Rupa', 2),
(103, '2 States', 250.00, 'Prentice Hall', 2);
```

i) Display author names that starts with S.

SQL
```
SELECT aname FROM Author WHERE aname LIKE 'S%';
```

ii) Display the total price of book published by "Prentice hall".

SQL
```
SELECT SUM(price) FROM Book WHERE publication = 'Prentice Hall';
```

iii) Update mobile number of author named "Sane Guruji" to 9844567822.

SQL
```
UPDATE Author SET mobile_number = '9844567822' WHERE aname = 'Sane Guruji';
```

iv) Display details of books written by author "Chetan Bhagat".

SQL
```
SELECT b.* FROM Book b JOIN Author a ON b.aid = a.aid WHERE a.aname = 'Chetan Bhagat';
```

B) Create a stored function named as "max_price" which will find maximum book price.

SQL

```sql
CREATE OR REPLACE FUNCTION max_price()
RETURNS DECIMAL AS $$
DECLARE
    max_val DECIMAL;
BEGIN
    SELECT MAX(price) INTO max_val FROM Book;
    RETURN max_val;
END;
$$ LANGUAGE plpgsql;

SELECT max_price();
```

Q.2) Operating Systems

Write a program to implement Bankers algorithm. Mention no. of processes and available resources. Calculate need matrix based on max and allocation matrix.

C

```c
#include <stdio.h>

int main() {
    int n, m, i, j;
    int alloc[10][10], max[10][10], need[10][10];

    printf("Enter number of processes: ");
    scanf("%d", &n);

    printf("Enter number of resources: ");
    scanf("%d", &m);

    printf("Enter Allocation Matrix (n x m):\n");
    for(i = 0; i < n; i++) {
        printf("Process P%d: ", i);
        for(j = 0; j < m; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }

    printf("Enter Max Matrix (n x m):\n");
    for(i = 0; i < n; i++) {
        printf("Process P%d: ", i);
        for(j = 0; j < m; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    // Calculate Need Matrix
    for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    printf("\nNeed Matrix (Max - Allocation):\n");
    for(i = 0; i < n; i++) {
        printf("P%d: ", i);
        for(j = 0; j < m; j++) {
            printf("%d\t", need[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Practical Slip 25

Q.1) Database Management System (PostgreSQL)

Consider the following database: Professor (prof_no, prof_name, designation, salary) Department (dno, dname, location)

The relationship is as follows: Department-Professor: one to many.

A) Create above database in PostgreSQL and insert sufficient records.

SQL

```
CREATE TABLE Department (
   dno INT PRIMARY KEY,
   dname VARCHAR(50),
   location VARCHAR(50)
);

CREATE TABLE Professor (
   prof_no INT PRIMARY KEY,
   prof_name VARCHAR(50),
   designation VARCHAR(50),
   salary DECIMAL(10, 2),
   dno INT REFERENCES Department(dno)
);

INSERT INTO Department VALUES
(1, 'Computer', 'Pune'),
(2, 'Electronics', 'Mumbai'),
(3, 'Mathematics', 'Pune');

INSERT INTO Professor VALUES
(101, 'Prof. Sameer', 'HOD', 90000.00, 1),
(102, 'Prof. Mayur', 'Lecturer', 60000.00, 1),
(103, 'Prof. Sagar', 'Asst. Prof', 55000.00, 2),
(104, 'Prof. Sudhir', 'Lecturer', 50000.00, 1);
```

i) Display average salary of professor.

SQL

```
SELECT AVG(salary) FROM Professor;
```

ii) List the details of all the departments located at "Pune".

```
SELECT * FROM Department WHERE location = 'Pune';
```

iii) Display the details of professors whose names ends with an alphabet "r".

```
SELECT * FROM Professor WHERE prof_name LIKE '%r';
```

iv) Display details of all professors working in "Computer" department.

```
SELECT p.* FROM Professor p

JOIN Department d ON p.dno = d.dno
WHERE d.dname = 'Computer';
```

B) Create a stored procedure named as "display_message" which will display the message "Welcome to RDBMS world!!!!."

```
CREATE OR REPLACE PROCEDURE display_message()

LANGUAGE plpgsql
AS $$
BEGIN
    RAISE NOTICE 'Welcome to RDBMS world!!!!';
END;
$$;

CALL display_message();
```

Q.2) Operating Systems

Write a Program to implement following functionality:

1. **Accept Available**
2. **Display Allocation, Max**
3. **Display the contents of need matrix**

Use the provided table data (Process P1-P5, Resources A, B, C).

C

```
#include <stdio.h>

int main() {
    int n, m, i, j;
    int alloc[10][10], max[10][10], need[10][10], avail[10];

    // Standard inputs based on the slip example (5 Processes, 3 Resources)
    printf("Enter number of processes (e.g., 5): ");
    scanf("%d", &n);

    printf("Enter number of resources (e.g., 3): ");
    scanf("%d", &m);

    printf("Enter Available Resources (A B C): ");
    for(j = 0; j < m; j++) {
        scanf("%d", &avail[j]);
    }

    printf("Enter Allocation Matrix (P1 to P%d):\n", n);
    for(i = 0; i < n; i++) {
        printf("For P%d: ", i + 1);
        for(j = 0; j < m; j++) {
            scanf("%d", &alloc[i][j]);
        }
    }

    printf("Enter Max Matrix (P1 to P%d):\n", n);
    for(i = 0; i < n; i++) {
        printf("For P%d: ", i + 1);
        for(j = 0; j < m; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    // Calculate Need Matrix
    for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++) {
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }

    // Display Allocation and Max as requested
    printf("\nProcess\tAllocation\tMax\t\tNeed\n");
    printf("\tA B C\t\tA B C\t\tA B C\n");

    for(i = 0; i < n; i++) {
        printf("P%d\t", i + 1);

        // Display Allocation
        for(j = 0; j < m; j++) printf("%d ", alloc[i][j]);
        printf("\t\t");

        // Display Max
        for(j = 0; j < m; j++) printf("%d ", max[i][j]);
        printf("\t\t");

        // Display Need
        for(j = 0; j < m; j++) printf("%d ", need[i][j]);
        printf("\n");
    }

    return 0;
}
```