Objective

To design a terminal-based telecom billing system that handles customer registration, plan assignment, usage tracking, bill generation, and payment processing, including edge cases like reconnection and duplicate entries.

◇ System Initialization

```
- Initialize empty lists:
  plans[], customers[], usages[], bills[], payments[]
- Set counters:
  customer_id = 1, bill_id = 1, payment_id = 1
```

◇ Main Menu Loop

```
WHILE True:
    DISPLAY menu options:
        1. Add Customer
        2. Assign Plan
        3. Add Usage
        4. Generate Bill
        5. Make Payment
        6. Exit

    INPUT choice

    IF choice == 1:
        CALL add_customer()
    ELSE IF choice == 2:
        CALL assign_plan()
    ELSE IF choice == 3:
        CALL add_usage()
    ELSE IF choice == 4:
        CALL generate_bill()
    ELSE IF choice == 5:
        CALL make_payment()
    ELSE IF choice == 6:
        BREAK
    ELSE:
        DISPLAY "Invalid choice"
```

◇ Function: Add Customer

```
INPUT: name, phone, email, address, plan_id

FOR each customer IN customers:
    IF customer.phone == phone OR customer.email == email:
        IF customer.is_active == False:
            customer.is_active = True
            customer.plan_id = plan_id
            DISPLAY "Customer reactivated"
            RETURN
        ELSE:
            DISPLAY "Customer already exists"
            RETURN

CREATE new Customer with customer_id++
ADD to customers[]
DISPLAY "Customer added"
```

◇ Function: Assign Plan

```
INPUT: customer_id, new_plan_id

FIND customer BY customer_id

IF customer.is_active == False:
    DISPLAY "Customer is inactive"
    RETURN

customer.plan_id = new_plan_id
DISPLAY "Plan updated"
```

◇ Function: Add Usage

```
INPUT: customer_id, month, call_minutes, data_gb

FIND customer BY customer_id

IF customer.is_active == False:
    DISPLAY "Customer is inactive"
    RETURN

FOR each usage IN usages:
    IF usage.customer_id == customer_id AND usage.month == month:
        DISPLAY "Usage already recorded"
        RETURN

CREATE new Usage
ADD to usages[]
DISPLAY "Usage recorded"
```

◇ Function: Generate Bill

```
INPUT: customer_id, month
FIND customer BY customer_id
IF customer.is_active == False:
    DISPLAY "Customer is inactive"
    RETURN

FIND plan BY customer.plan_id
FIND usage BY customer_id AND month

IF usage NOT FOUND:
    total = plan.monthly_fee
ELSE:
    extra_call = MAX(0, usage.call_minutes - plan.call_limit) * 0.5
    extra_data = MAX(0, usage.data_gb - plan.data_limit) * 10
    total = plan.monthly_fee + extra_call + extra_data

CREATE new Bill with bill_id++, total, status = "Unpaid"
ADD to bills[]
DISPLAY "Bill generated"
```

◇ Function: Make Payment

```
INPUT: bill_id, date, amount, mode
FIND bill BY bill_id
IF bill.status == "Paid":
    DISPLAY "Bill already paid"
    RETURN

IF amount < bill.total_amount:
    DISPLAY "Partial payment recorded" (optional)
    bill.status = "Partially Paid"
ELSE:
    bill.status = "Paid"

CREATE new Payment with payment_id++
ADD to payments[]
DISPLAY "Payment successful"
```

◇ Edge Case Handling

Scenario	Handling
Reconnecting customer	Reactivate and update plan
Duplicate customer	Prevent addition
Usage for inactive customer	Block entry
Duplicate usage	Prevent entry
Bill without usage	Charge base fee only
Exceeding limits	Add extra charges
Payment for paid bill	Block payment
Partial payment	Optional support

Conclusion

- This assignment helped me understand how to build a simple telecom billing system using basic programming logic.
- I designed the system to work in the terminal, allowing users to manage customers, assign plans, track usage, generate bills, and process payments.
- I included real-world scenarios like:
    - Reconnecting old customers without creating duplicate records
    - Preventing duplicate usage entries
    - Calculating extra charges when usage exceeds plan limits
- I wrote a clear pseudocode for each function, making the logic easy to follow and implement.
- The system is modular, beginner-friendly, and can be expanded in the future with features like service deactivation or file storage.
- Overall, this project gave me hands-on experience in designing a complete, real-world application using structured logic.

T D Yasaswini
Employee code-81692