Assignment Name: Embassy Consulate Line System

DATA STRUCTURE LAB [T]

Semester: Spring 24–25

FAIRUZ FAIYAZ (23-54707-3)

Instructor: FAHIM FAYSAL SAKIB

---

1. Detailed Understanding of the Problem Statement

The objective of this assignment is to design and implement a **token-based queue management system** for an embassy's consulate, which handles the processing of visa applications. Four visa categories are supported:

1. **Tourist Visa (TR)**
2. **Medical Visa (MED)**
3. **Business Visa (BS)**
4. **Government Officials Visa (GO)**

Key functional requirements:

- **Daily Limits:** Each visa type has a strict daily cap of 25 applicants, yielding an overall maximum throughput of 100 applicants per day.
- **Global Token Sequencing:** Tokens carry a prefix for visa type and a global counter (e.g., TR-1, MED-2), ensuring a unified, monotonic sequence regardless of category.
- **Four Service Counters:** Each counter is dedicated primarily to one visa type (Counter 1 → TR, Counter 2 → MED, etc.).
- **Dynamic Reallocation:** If a counter exhausts its primary queue but has not served 25 customers, it automatically pulls the next token from the **largest remaining queue**, maximizing resource utilization and minimizing wait times.
- **Fairness & Ordering:** Within and across visa types, customers are served in **strict arrival order** based on token sequence.
- **End-of-Day Summary (Bonus):** Generate a report summarizing:
    - Number of applicants served per visa type.
    - Breakdown of served tokens per counter, including reused counters.
    - Identify any counters that remained idle (served zero).

This problem combines **fair scheduling**, **resource allocation**, and **data encapsulation**, simulating a real-world service environment under hard capacity constraints.

2. Data Structure Concepts and Justification

Concept	*Usage*	*Why It Fits*
Queue	*Four independent queues (trQueue, medQueue, bsQueue, goQueue) hold incoming Token structs.*	*First-In-First-Out ensures each applicant is served in arrival order, critical for fairness. Queue operations (enqueue/dequeue) run in O(1).*
Struct	*Token packs visaType and number; Queue manages buffer indices; Counter logs served tokens.*	*Encapsulates related fields, improving modularity and readability.*
Array	*Underlying storage for each Queue (fixed-size) and logs in Counter (up to 100 served tokens).*	*Predictable memory usage; direct indexing simplifies implementation of circular queue behavior.*
Global Counter	*Single int globalTokenNumber auto-increments for every request.*	*Guarantees unique, sequential tokens across all categories.*
Linear Search	*When reallocation is needed, scan all queues to find the one with maximum size.*	*With only 4 queues, O(n) search (n=4) is trivial and efficient in practice.*

Logic Behind Structure Choices:

- **Enqueuing**: Upon token requests, check both the visa-specific limit (size < 25) and the overall limit (total < 100) before enqueueing in O(1) time.
- **Dequeuing**: A counter checks its primary queue first; if empty, it iterates through the four queues to identify the longest (by size())—a constant-time scan in this context—and dequeues from it.
- **Logging**: Each Counter appends served tokens and their types to arrays, enabling a straightforward end-of-day summary without additional traversal of queues.

3. Challenges and How They Were Addressed

1. **Global vs. Per-Type Limits**
    - **Challenge:** Simultaneously enforcing a global cap of 100 and per-type caps of 25.
    - **Solution:** Maintain two counters: totalApplicants and visaTypeCounts[4]. Every requestToken call checks both before issuing and enqueuing.
2. **Dynamic Counter Reallocation**
    - **Challenge:** After a counter's primary queue empties, it must serve the "longest" secondary queue without starving smaller queues.
    - **Solution:** A simple loop compares size() across the four queues (ignoring the primary), picking the index with maximum pending tokens. For ties, the first encountered is chosen—acceptable given equal priority.
3. **Fairness Across Categories**
    - **Challenge:** Ensuring no visa category could "jump" ahead out of token order.
    - **Solution:** Tokens are strictly global-sequence-based. Even when counters reallocate, they dequeue the earliest-arrived token from the chosen queue.
4. **Summary Report Accuracy**
    - **Challenge:** Collating served-data across multiple counters and categories for the bonus task.
    - **Solution:** Each Counter instance maintains its own arrays (servedTypes[], servedTokens[]) and a served count. The generateSummaryReport() function iterates these arrays to compile the final statistics.
5. **User Interface Robustness**
    - **Challenge:** Handling invalid inputs (e.g., incorrect menu choices) gracefully without crashing.
    - **Solution:** Input validation with fallback messages and loop re-prompting ensures stability and a friendly user experience.

4. Detailed Implementation Diagram

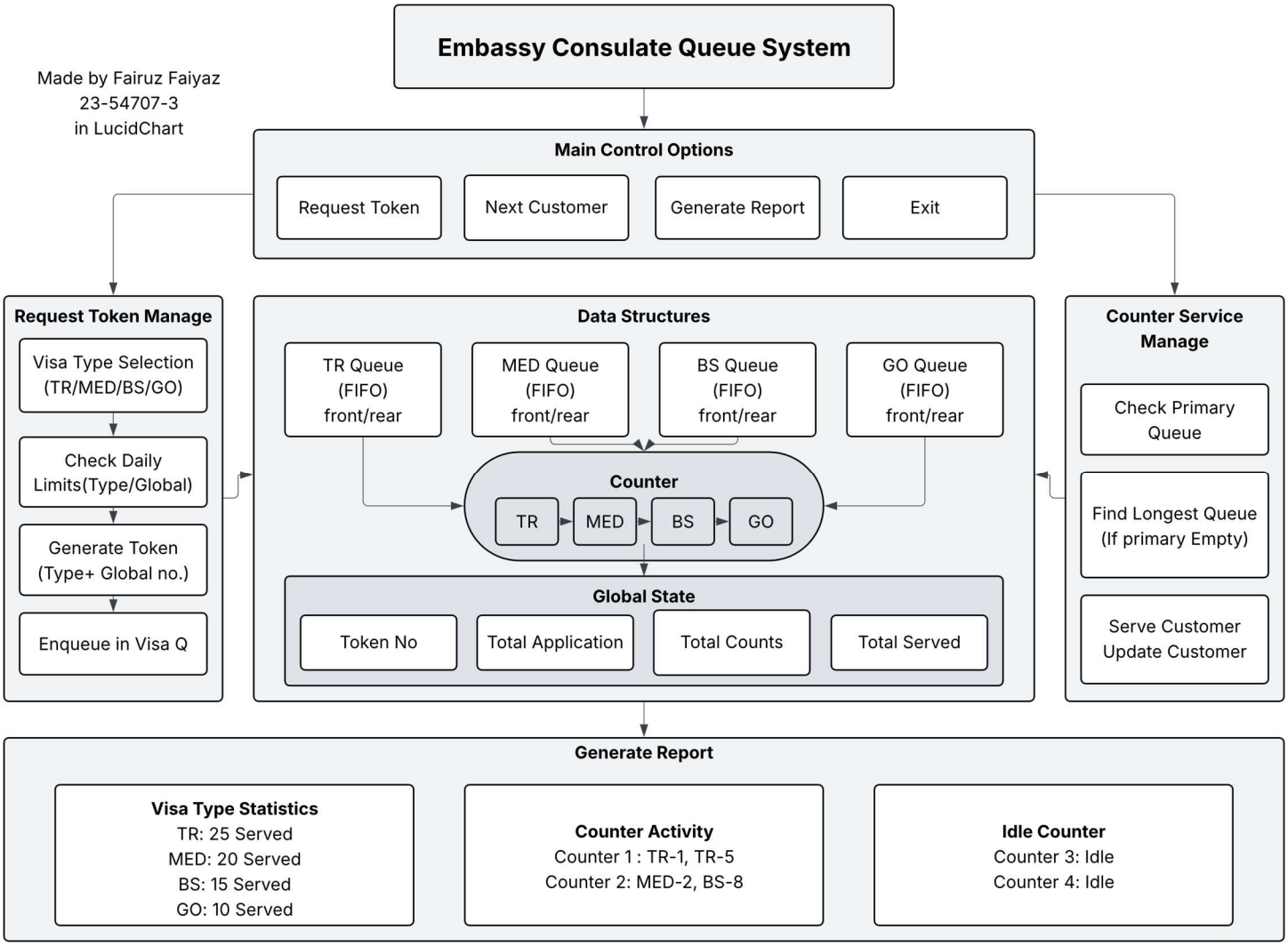

*diagram source file attached in .zip file

Diagram flowchart Explanation

1. System Initialization

- Creates four empty queues:
  - **TR** (Tourist)
  - **MED** (Medical)
  - **BS** (Business)
  - **GO** (Government Officials)
- Initializes four service counters with their **primary visa types**.
- Resets the **global token number**, starting from **1**.

---

2. Token Request Flow

Applicant Action → Visa Type Selection → System Processing

1. User selects a visa type.
2. System checks:
   - If **visa-type limit (25)** is reached.
   - If **overall limit (100)** is reached.
3. If allowed:
   - A **global token** is generated (e.g., TR-1, MED-2).
   - The token is **added to the respective queue**.

---

3. Counter Service Flow

When a staff calls the next customer at a counter:

- **Step A**: Serve from the counter's **primary visa queue** if not empty.
- **Step B**: If empty, find the **longest non-empty** visa queue.
- **Step C**: Serve from that queue (fallback/reallocation).
- **Step D**: Update:
  - **Counter's service history**
  - **Visa type statistics**

---

4. Report Generation

The report is generated based on real-time data tracking:

1. Collect data from:
   - visaTypeCounts[] array (per visa type count)
   - Each counter's servedTokens[] and servedTypes[]
   - Total tokens issued
2. Calculate:
   - Which counters were **idle**
   - **Total applicants served**
3. Format and display the report under:
   - Applicants served by visa type
   - Applicants served by counter
   - Idle counters
   - Total served count

Key Components

- **Queue Management**:
  Array-based **FIFO** queues for each visa type (fixed size = 25).
- **Counter Logic**:
  Priority is given to the **primary visa queue**; fallbacks are handled by selecting the **longest remaining queue**.
- **Token System**:
  Globally incrementing token numbers for fairness and consistency.
- **Limits Enforcement**:
  - 25 tokens per visa type
  - 100 tokens total
- **Data Tracking**:
  Real-time stats collected in memory for summary reporting.

System Constraints

- Maximum **25 tokens** per visa type per day
- Maximum **100 tokens total** across all types
- Counters can serve **other queues** if their primary queue is empty
- All queues follow **First-Come-First-Serve (FCFS)**
- **End-of-day summary report** must be detailed and accurate

---

User Roles & Interactions

- **Applicants**:
  - Request tokens via menu interface
- **Embassy Staff**:
  - Serve customers at their assigned counters
- **Managers/Admins**:
  - Generate and review **daily summary reports**
- **System**:
  - Maintains all operational data **in memory** (no file/database storage)

Conclusion

This project successfully simulates a real-world visa processing system using core data structure concepts. It ensures fair scheduling, optimal resource use, and complete data tracking through queues, structs, and counters. Through this system, I have deepened my understanding of dynamic queue management, condition-based resource allocation, and real-time user interaction handling in C++. The solution is robust, modular, and scalable for future enhancements.