1. Create a queue with basic operations: enqueue, dequeue, and display using a fixed-size array.
2. Perform the same queue operations but with a dynamic memory allocation approach using a singly linked list.
3. Handle the wrap-around scenario using modulo operation in a circular queue.
4. Insert elements based on priority. Higher priority elements are dequeued before lower ones.
5. Given a queue containing a list of integers (e.g., `{10, 20, 30, 40, 50}`), write a program to reverse the order of its elements using a stack.

You must:

- Use only **queue operations** (`enqueue`, `dequeue`) and **stack operations** (`push`, `pop`).

- **Do not** use any additional arrays or reverse functions.

  Example:

  Original Queue: 10 20 30 40 50
  Reversed Queue: 50 40 30 20 10

6. Given a queue of even length, interleave its two halves.

   Example:
   Original Queue:
   `{10, 20, 30, 40, 50, 60}`

   - First half: `{10, 20, 30}`

   - Second half: `{40, 50, 60}`

   Interleaved Output:
   `{10, 40, 20, 50, 30, 60}`

7. Task Scheduler with Priority and Time-Slice Simulation

Problem Statement:
Design a **task scheduler** where each task has a:

- `taskID`

- `priority` (lower number = higher priority)
- `estimatedTime` (in seconds)

Implement:

- Priority-based scheduling: higher priority tasks are executed first.
- Time-slicing: Each task can run for a maximum of 5 seconds per cycle.
- If a task needs more time, it is re-enqueued with the remaining time.

8. Ticket Counter with Category-based Queues

Problem Statement:
Simulate a **multi-counter ticketing system** with different queues:

- **Normal Queue**
- **Senior Citizen Queue**
- **VIP Queue**

Implement:

- Enqueue customers into their respective category.
- A serving logic where VIPs are served first, then seniors, then normal.
- Every customer has a name, age, and ticket number.
- Each counter processes one customer at a time from the highest priority available queue.