Goal: You will **choose** a realistic FIFO scenario and implement a working queue in C. You also choose the data-structure variant.

1) Choose your own scenario

Think first, then pick a scenario where **fair turn-taking (FIFO)** matters.

Examples (you may NOT reuse verbatim; customize or invent your own): print jobs, helpdesk tickets, call waiting, packet buffer, producer consumer, cafeteria service, etc. If unsure, propose your idea to the TA for a quick approval.

2) Pick one implementation path (your choice)

- **Array (non-circular, lazy move + compaction)**
  - Keep front, rear; live window is a[front..rear-1].
  - Define and implement a **compaction trigger**.
- **Linked List (head/tail)**
  - enqueue at tail, dequeue at head; free nodes; maintain length.

3) Define your own methods (your API)

- **Name & signature** (parameters, return types)
- **Behavior on empty/full** (what is returned, what error code/flag)
- **Effects on state** (front/rear or head/tail, size/length updates)
- **Any invariants** you rely on (e.g., one-slot-empty rule, or size used to distinguish states)

4) Report checklist (PDF, 2–4 pages)

- **Design**: which variant (array or list) and **why it fits** your scenario; include a small diagram marking **front/rear** or **head/tail**.
- **Your methods**: list the API you defined (names, brief behavior) and justify any special choices (e.g., compaction threshold).
- **Edge cases handled**: enumerate and point to code locations handling them (empty dequeue/peek; full on array; last-item removal in list; memory safety).
- **Complexity**: time for your main methods; space cost; when compaction happens (if array) and why it's acceptable.
- **Evidence of correctness**: provide concise proof that FIFO order is preserved in your runs (e.g., short trace, screenshot, or table). Keep it lightweight; format is your choice.

5) Deliverables

- **One PDF (the report) per group.**
- **Screenshot the code in the report.**