Checklista Inlämningsuppgift

Algoritmer, datastrukturer och design patterns

OBS: Det finns visst utrymme för tolkning och egna tillvägagångssätt för att lösa uppgiften, ***om detta kan motiveras*.**

0) Kompilerar mitt program?

1. **Synka headers och .c-filer**
- Funktionsnamn måste matcha *exakt* (annars link-fel).
- Returtyper/parametrar måste matcha.
2. **Alla `int`/`bool`-funktioner måste returnera på alla kodvägar**
- Inga "faller ur" utan `return`.
3. **Kompilera gärna med varningar på**
- Ex: `-Wall -Wextra` och fixa varningarna (de är ofta riktiga buggar).

---

1) Event-modell (krav)

4. **Event-typ måste minst ha**
- `timestamp (int)`, `sensorId (int)`, `type (enum)`, `value (int)`.
5. **Timestamp ska vara strikt stigande vid skapande**
- T.ex. en global räknare som ++ varje nytt event. Använd gärna faktisk tid.

---

2) ADT: EventLog som dynamisk lista/array (krav)

6. **Implementera EventLog som dynamisk array/lista**
    - Uppgiften vill att resten av programmet använder loggen via funktioner (ADT), inte peta i intern data.
7. **Minsta gränssnitt (enligt exempel)**
    - `EventLog* log_create(int capacity)`
    - `void log_destroy(EventLog* log)`
    - `int log_size(const EventLog* log)`
    - `void log_append(EventLog* log, Event e)`
    - `Event log_get(const EventLog* log, int index)`
    - `void log_set(EventLog* log, int index, Event e)`
8. **Krav: specialfall + minne**
    - Hantera tom logg, 1 element, rimlig indexkontroll.
    - Allt frigörs i `log_destroy`.

---

3) ADT: EventQueue som ringbuffer, fast kapacitet (krav)

9. **Bygg Queue exakt som ringbuffer**
    - Fast kapacitet, ingen flytt av element vid dequeue.
10. **Funktioner som ska finnas**
    - `Queue* queue_create(int capacity)`
    - `void queue_destroy(Queue* q)`
    - `bool queue_isEmpty(const Queue* q)`
    - `bool queue_isFull(const Queue* q)`
    - `bool queue_enqueue(Queue* q, Event e)`
    - `bool queue_dequeue(Queue* q, Event* out)` (false om tom)
11. **Interna fält (handhållning från uppgiften)**
    - `buffer[capacity]`, `head`, `tail`, `count`.

---

4) Event loop (producer/consumer) + tick (krav)

Producer = funktionen/loopen som gör:

```
Event e = make_random(...); queue_enqueue(q, e);
```

Consumer = funktionen/loopen som gör:

```
queue_dequeue(q, &e); log_append(log, e);
```

12. **Event ska komma in via Queue, inte direkt till Log**
    - Producer skapar event → enqueue.
    - Consumer dequeue'ar → log_append.
13. **Implementera kommandot `tick <n>`**
    - Kör n iterationer där programmet försöker skapa event + lägga i kön, och försöker konsumera ett event från kön till loggen.

---

5) Sortering av loggen (krav)

14. **Minst 1 sort för G**
    - Insertion sort (rekommenderas) *eller* Selection sort.
15. **Sorten måste funka för logg med 0–1 element**
    - Och sortera korrekt.
16. **Lägg gärna till hjälpfunktion**
    - `bool isSortedByTimestamp(const EventLog* log)` (tips i uppgiften).

---

6) Sökning (linjär) (krav)

17. **Implementera `find <sensorId>`**
    - Ska gå linjärt genom loggen och skriva ut alla events med den sensorId.

---

7) Kommandomeny i terminal (krav)

18. **Minimikrav på kommandon**
    - `tick <n>`
    - `print`
    - `sort <name>`
    - `find <sensorId>`
    - `help`
    - `exit`
19. **Gör formatet lätt att testa**
    - En loop som läser en rad och tolkar kommandot.

---

8) Design pattern: Strategy-light för sort (krav)

20. **Sortval måste ske via strategi även om du bara har 1 sort**
    - Dvs: funktionspekare (C) / funktionsobjekt (C++), mappa text → sortfunktion.

Exempelidé (som uppgiften ger):

- `typedef void (*SortFn)(EventLog* log);`
- funktion som mappar sträng → `SortFn`.

---

9) För G krävs även minst 1 valfri modul (A/B/C/D)

21. **Välj minst en modul**
    - A: AlarmSet (rekommenderas)

- B: Andra sortalgoritm (kopplad via Strategy)
- C: Hash/array för “last <sensorId>”
- D: Observer-light (bonus)

Och kom ihåg: G-kriterierna listar uttryckligen “minst 1 valfri modul”.

---

10) Inlämning (gör programmet så enkelt som möjligt att testa för andra)

22. **README som visar**
    - hur man bygger/kör
    - ett par exempelkommandon (en “körning” som visar att det fungerar).

23. **Rapport (kort)**

    - översikt, Strategy-förklaring, grov komplexitet (sort + linjär sökning), minneshantering.