Love Gruvberg | IoT25, Data Structures, Algorithms, and Design Patterns

Report: Embedded Event Logger

1. Solution Overview

This project simulates an event-driven logging system in C.

Modules and Abstract Data Types

The solution consists of five main modules organized as ADTs:

Event.h: Defines event types (SENSOR_READING, EVENT_SYSTEM) and the Event struct (timestamp, sensorId, type, value).

EventQueue.h/.c: Fixed-size FIFO queue implemented as a ring buffer. Uses circular indexing (head, tail, count) to avoid shifting elements.

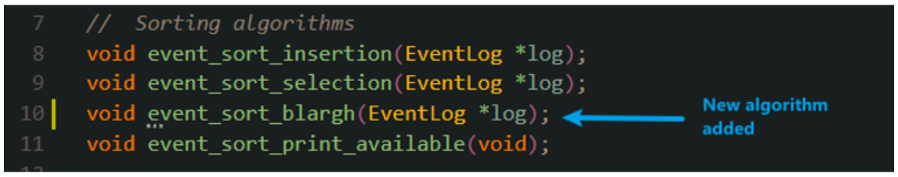
EventLog.h/.c: Dynamically growing array that stores processed events. Doubles capacity when full using realloc.

EventSort.h/.c: Implements the Strategy pattern with a registry system that maps algorithm names to function pointers for runtime selection (see Adding a new sorting algorithm).

EventSearch.h/.c: Linear search to find events by sensor ID.

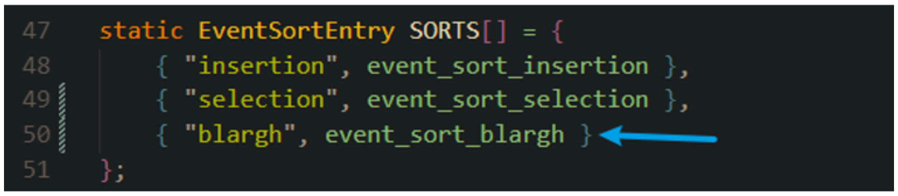
<u>Adding a new sorting algorithm:</u>

EventSort.h

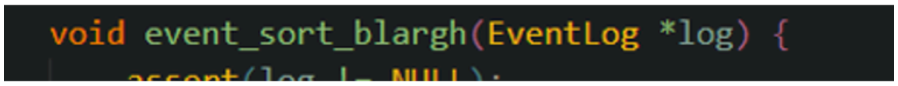


```
//  Sorting algorithms
void event_sort_insertion(EventLog *log);
void event_sort_selection(EventLog *log);
void event_sort_blargh(EventLog *log);
void event_sort_print_available(void);
```

EventSort.c

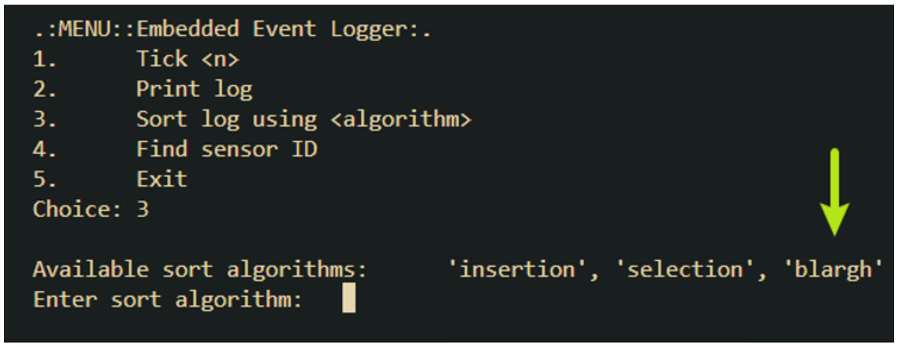


```
static EventSortEntry SORTS[] = {
    { "insertion", event_sort_insertion },
    { "selection", event_sort_selection },
    { "blargh", event_sort_blargh }
};
```



```
void event_sort_blargh(EventLog *log) {
    assert(log != NULL);
```

And here it is on runtime:



```
.:MENU::Embedded Event Logger:.
1.      Tick <n>
2.      Print log
3.      Sort log using <algorithm>
4.      Find sensor ID
5.      Exit
Choice: 3

Available sort algorithms:     'insertion', 'selection', 'blargh'
Enter sort algorithm:   
```

Event Flow

Producer > EventQueue > Consumer > EventLog

tick <n> runs the event loop for *n* iterations. Each iteration generates events, enqueues them to the queue, dequeues them for processing, and appends them to the log.

2. Design Pattern

The **Strategy pattern** is used for sorting. All algorithms share the same function signature and are accessed through function pointers.

I think it fits well as it allows changing sorting behavior without modifying surrounding code, and makes adding new algorithms very straightforward (as seen in the screenshots above).

3. Time Complexity

Sorting Algorithms

- **Insertion Sort**
    - **Best case: O(n)** when already sorted
    - **Worst case: O(n^2)** when in reverse order
- **Selection Sort**
    - **Best case: O(n^2)**
    - **Worst case: O(n^2)**

4. Memory Management

Each ADT manages its own memory, this keeps responsibilities clearly separated and helps avoiding memory leaks. All dynamic allocations are freed on termination.

- **EventQueue:** Fixed-size buffer allocated at init, freed at exit
- **EventLog:** Dynamic array is resized with realloc when capacity is reached