Laboratory Work: Thread-Safe Data Structures

1. Student Information

Name: Oleksandr Zahorodnyi
Group: K-27

2. Task Condition (Variant 11)

Develop a thread-safe data structure with `m=3` integer fields.

- **Requirements:** Independent `read` / `write` for fields, and `operator string` for the whole structure.
- **Constraint:** No `std::atomic`, only mutexes.
    - **Workloads:**
      Scenario A (Var 11): Weighted distribution.
      Scenario B: Equal distribution.
      Scenario C (Worst Case): 90% `string` operations (global lock).

3. Data Protection Scheme

The solution uses **Fine-Grained Locking** optimized for Apple Silicon.

1. **Per-Field Mutexes:** Each field has its own `std::mutex`.
    - *Justification:* Allows threads to access different indices simultaneously without blocking.
2. **Global Snapshot (`operator string`):** Uses `std::scoped_lock` to acquire all mutexes at once.
    - *Justification:* Ensures a consistent state (atomic snapshot) and prevents deadlocks.
3. **Cache Line Alignment (`alignas(128)`):**
    - *Justification:* Prevents "False Sharing" on the M1 processor by keeping mutexes on separate cache lines.

4. Experimental Results (Time in ms)

Scenario	1 Thread	2 Threads	3 Threads
A (Variant 11)	14.4673 ms	3.91858 ms	127.287 ms
B (Equal)	5.691 ms	17.303 ms	55.9817 ms
C (Worst Case)	26.8083 ms	28.729 ms	309.019 ms

5. Conclusions

- **Scenarios A & B:** Performance improves with more threads as operations target different fields (low contention).
- **Scenario C:** Performance degrades or plateaus because the `string` operation locks the entire structure, effectively serializing execution.
- **Optimization:** The architecture-specific alignment was critical to minimize hardware-level cache contention.

6. Individual Contribution

1. Designed `ConcurrentStructure` using fine-grained locking strategy.
2. Implemented memory alignment to optimize for CPU cache lines.
3. Developed the `TestUtils` module for generating weighted test files.
4. Created the benchmarking logic to measure execution time across multiple threads.