C++ STL Masterclass — Part 3

Advanced continuation: Boost extensions, policy-based/data-structure extras, and practical micro-benchmarking techniques for STL code.

---

1) Boost Libraries Worth Knowing (STL complements)

1.1 Boost.Container

- `boost::container::flat_map`, `flat_set` — sorted vector-backed containers: excellent cache locality for mostly-read workloads.
- `boost::container::pmr` aligns with `std::pmr` and gives more utilities for custom allocation.

1.2 Boost.Unordered

- Drop-in replacements with extra tuning knobs and historically faster behavior on some workloads.

1.3 Boost.MultiIndex

- Store objects once but index them multiple ways (by id, by timestamp, by name) with efficient views.
- Great for DB-like in-memory structures.

1.4 Boost.Hana / MPL / Fusion

- Metaprogramming and compile-time data structures.

1.5 Boost.Heap / Priority Queue Variants

- Pairing heap, binomial heap, etc. Useful when you need decrease-key operations or special heap behavior.

---

2) Policy-Based Data Structures (GNU extension)

2.1 Order Statistics Tree (indexed set/map)

Provided by PBDS (Policy Based Data Structures) in GNU C++ (header `<ext/pb_ds/assoc_container.hpp>`): - `tree_order_statistics_node_update` supports: - `find_by_order(k)` — k-th smallest (0-based) - `order_of_key(x)` — number of items strictly smaller than x

Example usage (set with order statistics):

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
```

```
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
```

Caveats: Non-standard (GNU), not portable to MSVC without ports. Use when competitive programming or when you need index-like queries on ordered sets.

---

3) When to Prefer Boost or PBDS

- Need multi-indexing → Boost.MultiIndex.
- Need faster, cache-friendly small maps/sets → `flat_map` / `flat_set`.
- Need order statistics in a set/map → PBDS.
- Need pooling/custom allocators beyond `pmr` → Boost.Pool.

---

4) Micro-Benchmarking STL Code (How to compare choices)

4.1 Principles

- Use `std::chrono::steady_clock` to measure durations.
- Warm up the code (JIT effects irrelevant in C++, but caches/allocations matter).
- Repeat many iterations and take median to avoid outliers.
- Control allocator behavior (use `pmr` or reserve capacity) to test algorithmic differences rather than allocation noise.
- Use realistic data shapes (sorted, reverse, random, many collisions for hashes).

4.2 Tools

- `google/benchmark` — robust microbenchmark library.
- `perf` (Linux), `valgrind` / `callgrind` for hotspots.
- `heaptrack` or `massif` for memory profiling.

4.3 Example: simple timing harness

```
auto timeit = [&](auto&& fn, int iter=5){
  using namespace std::chrono;
  vector<long long> times;
  for (int i=0;i<iter;++i){
    auto t0 = steady_clock::now();
    fn();
    auto t1 = steady_clock::now();
    times.push_back(duration_cast<microseconds>(t1-t0).count());
  }
  sort(times.begin(), times.end());
```

```
  return times[times.size()/2]; // median
};
```

---

5) Benchmark Ideas to Compare Containers

- `vector` vs `list` for many small inserts (measure end-to-end throughput).
- `map` vs `unordered_map` for random keys with varying load factors.
- `flat_map` (Boost) vs `map` for small maps.
- Priority queue implementations (std::priority_queue vs Boost heaps) for decrease-key patterns.

---

6) Practical Tips for Faster STL Code

- Prefer `vector` and `reserve` where possible.
- Use `emplace_back` to avoid temporaries.
- For small fixed-size associative maps, `flat_map` often beats `map`.
- For repeated allocations, use `pmr` or a pool allocator.
- Profile before optimizing — hotspot may be elsewhere.

---

7) Example: Order-statistics usage

```
ordered_set os;
os.insert(5);
os.insert(1);
cout << *os.find_by_order(0) << "\n"; // 1
cout << os.order_of_key(5) << "\n"; // 1
```

---

8) Next steps

- Want me to add runnable benchmarks (google/benchmark) comparing `vector` vs `list` and `map` vs `unordered_map`? I can create a repo-style zip with benchmark code and scripts.
- Or should I implement a demo using PBDS (order statistics) and a Boost.MultiIndex example?

Pick one and I will add it immediately.