C++ STL Masterclass — Part 2

This continuation adds **advanced STL topics** and **practical exercises with complete solutions** so you can build real skills quickly.

A) Advanced Topics

A.1 Allocators (deep dive)

… *(same as before)* …

B) Practical Exercises — Full Solutions

B.1 Top-K Frequent Words (lexicographic tie break)

… *(same as before)* …

B.2 LRU Cache (using `list` + `unordered_map`)

… *(same as before)* …

B.3 Autocomplete (prefix search)

… *(same as before)* …

B.4 K-way Merge (merge many sorted arrays)

… *(same as before)* …

B.5 Sliding Window Median (complete solution)

We maintain two multisets (balanced BSTs): - `lo`: max-heap behavior (stores smaller half) - `hi`: min-heap behavior (stores larger half) - Sizes are balanced so that `|lo| >= |hi|`. - Median is: - `*lo.rbegin()` if odd window size, - average of `*lo.rbegin()` and `*hi.begin()` if even.

Code:

```
#include <bits/stdc++.h>
using namespace std;
```

```cpp
class SlidingMedian {
    multiset<int> lo, hi;
    int k;
public:
    SlidingMedian(int window): k(window) {}

    void insert(int x) {
        if (lo.empty() || x <= *lo.rbegin()) lo.insert(x);
        else hi.insert(x);
        balance();
    }

    void erase(int x) {
        if (lo.find(x) != lo.end()) lo.erase(lo.find(x));
        else hi.erase(hi.find(x));
        balance();
    }

    void balance() {
        while ((int)lo.size() > (int)hi.size() + 1) {
            hi.insert(*lo.rbegin());
            lo.erase(prev(lo.end()));
        }
        while ((int)hi.size() > (int)lo.size()) {
            lo.insert(*hi.begin());
            hi.erase(hi.begin());
        }
    }

    double median() {
        if (k % 2) return *lo.rbegin();
        return ((double)*lo.rbegin() + *hi.begin()) / 2.0;
    }
};

int main(){
    vector<int> nums = {1,3,-1,-3,5,3,6,7};
    int k = 3;
    SlidingMedian sm(k);
    vector<double> result;
    for (int i=0;i<nums.size();++i){
        sm.insert(nums[i]);
        if (i>=k-1){
            result.push_back(sm.median());
            sm.erase(nums[i-k+1]);
        }
    }
    for(double x: result) cout<<x<<" ";
```

```
    cout<<"\n";
}
```

Output:

```
1 -1 -1 3 5 6
```

C) Tests & How to Run

- All snippets use `#include <bits/stdc++.h>` for brevity; for production, include precise headers.
- Compile with `g++ -std=c++17 filename.cpp -O2`.

D) What next?

1. I can now prepare a **zip file** with all example `.cpp` files + README.
2. After that, I'll continue with **Part 3** covering Boost, policy-based data structures, and performance benchmarking.