CS2810 OOAIA

STL Part 2

Associative Containers (Ordered & Unordered)

Example Code Snippets

1. `set` Example - Ordered Unique Elements

```
#include <iostream>
#include <set>

int main() {
    std::set<int> s = {5, 3, 1, 4, 2};

    s.insert(5); // Duplicate ignored
    s.erase(1);  // Remove element 1

    for (int val : s)
        std::cout << val << " "; // Output: 2 3 4 5

    auto it = s.find(3);
    std::cout << "\nElement 3 " << (it != s.end() ? "found" :
"not found");
    return 0;
}
```

2. `multiset` Example - Ordered Elements with Duplicates

```cpp
#include <iostream>
#include <set>

int main() {
    std::multiset<int> ms = {5, 2, 5, 1};

    ms.insert(3);
    ms.erase(ms.find(5)); // Remove one occurrence of 5

    std::cout << "Multiset elements: ";
    for (int val : ms)
        std::cout << val << " "; // Output: 1 2 3 5

    std::cout << "\nCount of 5: " << ms.count(5); // Output: 1
    return 0;
}
```

3. `map` Example - Key-Value Pairs (Ordered)

```cpp
#include <iostream>
#include <map>

int main() {
    std::map<std::string, int> m = {{"Karan", 23}, {"Parthiv",
22}};

    m["Prateek"] = 28; // Insert/update
    m.erase("Parthiv");

    std::cout << "Map contents:\n";
    for (auto& pair : m)
        std::cout << pair.first << ": " << pair.second << "\n";
    // Output: Karan: 23, Prateek: 28
```

```
    std::cout << "Age of Karan: " << m["Karan"]; // Output: 25
    return 0;
}
```

4. `multimap` Example - Multiple Keys

```
#include <iostream>
#include <map>

int main() {
    std::multimap<std::string, int> mm = {{"Karan", 23},
{"Karan", 13}};

    mm.insert({"Prateek", 28});
    auto range = mm.equal_range("Karan");

    std::cout << "Karan's ages: ";
    for (auto it = range.first; it != range.second; ++it)
        std::cout << it->second << " "; // Output: 23 13
    return 0;
}
```

5. `unordered_set` Example - Hash-Based Unique Elements

```
#include <iostream>
#include <unordered_set>

int main() {
    std::unordered_set<int> us = {5, 3, 1, 4, 2};

    us.insert(3); // Duplicate ignored
    std::cout << "Unordered Set elements: ";
    for (int val : us)
        std::cout << val << " "; // Order varies (e.g., 1 2 3 4
5)
```

```
    std::cout << "\nBucket count: " << us.bucket_count(); //
Depends on implementation
    return 0;
}
```

6. `unordered_multiset` Example - Hash-Based with Duplicates

```
#include <iostream>
#include <unordered_set>

int main() {
    std::unordered_multiset<int> ums = {5, 3, 5, 2, 3}; //
Allows duplicates

    ums.insert(3); // Add another 3
    ums.erase(ums.find(5)); // Remove one occurrence of 5

    std::cout << "Elements (order varies): ";
    for (int val : ums)
        std::cout << val << " "; // Example output: 2 3 3 3 5
    std::cout << "\nCount of 3: " << ums.count(3); // Output: 3
    return 0;
}
```

7. `unordered_map` Example - Fast Key-Value Access

```
#include <iostream>
#include <unordered_map>

int main() {
    std::unordered_map<std::string, int> um = {{"Apple", 5},
{"Banana", 3}};

    um["Cherry"] = 10;
```

```
    std::cout << "Cherry count: " << um.at("Cherry") << "\n"; //
10
    std::cout << "Hash table contents:\n";
    for (auto& pair : um)
        std::cout << pair.first << ": " << pair.second << "\n";
// Order varies
    return 0;
}
```

8. `unordered_multimap` Example - Key-Value Pairs with Duplicate Keys

```
#include <iostream>
#include <unordered_map>
#include <string>

int main() {
    std::unordered_multimap<std::string, int> umm = {
        {"Karan", 25}, {"Karan", 30}, {"Parthiv", 28}
    };

    umm.insert({"Karan", 35});

    auto range = umm.equal_range("Karan");
    std::cout << "Ages for Karan: ";
    for (auto it = range.first; it != range.second; ++it)
        std::cout << it->second << " "; // Output: 25 30 35
(order varies)

    // Iterate all entries (unordered)
    std::cout << "\nAll entries:\n";
    for (auto& pair : umm)
        std::cout << pair.first << ": " << pair.second << "\n";
    return 0;
}
```

Summary of Key Operations

Container	Insert	Remove	Access	Iteration	Notes
`set`	`insert(val)`	`erase(val)`	`find(val)`	Ordered (ascending)	Unique elements, Red-Black Tree
`multiset`	`insert(val)`	`erase(val)`	`find(val)`	Ordered (ascending)	Allows duplicates
`map`	`insert({key, val})`	`erase(key)`	`operator[], find`	Ordered by key	Unique keys, sorted
`multimap`	`insert({key, val})`	`erase(key)`	`equal_range(key)`	Ordered by key	Multiple keys allowed
`unordered_set`	`insert(val)`	`erase(val)`	`find(val)`	Unordered	Hash table, O(1) average access
`unordered_multiset`	`insert(val)`	`erase(val)`	`find(val)`	Unordered	Allows duplicates, hash-based
`unordered_map`	`insert({key, val})`	`erase(key)`	`operator[], find`	Unordered	Hash table, no sorting
`unordered_multimap`	`insert({key, val})`	`erase(key)`	`equal_range(key)`	Unordered	Multiple keys, hash table

Practice Problems

1. Problem: Merge two `std::map<int, string>` into one, resolving key conflicts by keeping the value from the second map.

Example:

```
Input: map1 = {1: "A", 2: "B"}, map2 = {2: "C", 3: "D"}
Output: {1: "A", 2: "C", 3: "D"}
```

2. Problem: Given a string, find the first non-repeating character.

Example:

```
Input: "haldiram"
Output: 'h'
```

Hint: Use `unordered_map<char, int>` to count frequencies.

3. Problem: Remove duplicates from a vector of integers while preserving order.

Example:

```
Input: [4, 2, 4, 3, 2]
Output: [4, 2, 3]
```

Hint: Use `unordered_set` to track seen elements.

4. Problem: Group anagrams from a list of strings.

Example:

```
Input: ["eat", "tea", "tan", "ate", "nat", "bat"]
Output: [["eat","tea","ate"], ["tan","nat"], ["bat"]]
```

Hint: Use `unordered_map<string, vector<string>>` with sorted strings as keys.

5. Problem: Given a vector of integers, use a multiset to find the mode (most frequent element).

Example:

```
Input: [3, 7, 3, 5, 3, 5]
Output: 3 (appears 3 times)
```

6. Problem: Given a list of words and their page numbers in a book (e.g., [("apple", 5), ("banana", 3), ("apple", 7)]), generate an index where each word points to all pages it appears on.

Example:

```
Input: [("apple", 5), ("apple", 7), ("banana", 3)]
Output: apple: [5,7], banana: [3]
```

Hint: Use `multimap<string, int>` and `equal_range` to aggregate pages.