5. Hashing Implementation with C++ STL

In C++, instead of creating a Hash Table manually, you could use **Standard Template Library (STL)**. The STL implementation **automatically** handles hash functions, collisions, and **rehashing** when the load factor gets too high.

5.1. `std::unordered_map` (Key-Value)

`std::unordered_map` is a Hash Map implementation for **Key-Value pairs**. The purpose of this template is to map Keys to Values. It acts like a **dictionary**, where you look up a word (key) to get its definition (value).

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

int main() {
    // Key: string (name), Value: int (grade)
    std::unordered_map<std::string, int> studentGrades;

    studentGrades["Budi"] = 90;
    studentGrades["Ani"] = 85;

    // Access value using key
    std::cout << "Budi's Grade: " << studentGrades["Budi"] << std::endl;
}
```

5.2. `std::unordered_set` (Unique Keys)

`std::unordered_set` is a Hash Set implementation that only stores **unique keys**. This template is used to check whether a **key is on the list or not**, like a **guest book**.

```cpp
#include <iostream>
#include <string>
#include <unordered_set>

int main() {
    std::unordered_set<std::string> attendanceList;

    attendanceList.insert("Budi");
    attendanceList.insert("Ani");
    attendanceList.insert("Budi"); // Ignored, because "Budi" already exists

    // Check existence
    if (attendanceList.count("Budi") > 0) {
        std::cout << "Budi is present." << std::endl;
    }
}
```

5.3. Comparison: `map` vs. `set`

Aspect	`map`	`set`
Feature	`std::unordered_map<Key, Value>`	`std::unordered_map<Key>`
What is Stored	Key-Value pairs	Key only
Element Type	`std::pair<const Key, Value>`	`Key`
Main Purpose	Mapping Keys to Values	Storing unique Keys
Data Access	`map[key]` (get/set value)	`set.count(key)` (check existence)

Revision #1
Created 2025-11-04 15:26:59 UTC by RE
Updated 2025-11-04 15:27:39 UTC by RE