Assignment 5: Generic Hash Table in C++
Due: Saturday April 12th at 11:59 PM

---

1. Introduction
Objective:
The goal of this assignment is to design and implement a **generic hash table in C++** that stores key-value pairs using string keys and templated values. You will also explore **custom hash functions**, **collision resolution**, and use your hash table to solve algorithmic problems.
Learning Outcomes:

- Understand and implement the mechanics of hash tables
- Apply **template programming** in C++
- Explore and design **custom hashing and probing strategies**
- Practice proper **resource management** and implement the **Big Five**
- Use your own data structure in practical **algorithmic problem solving**

---

2. Requirements
Functional:

- Class template: HashTable<T>
- Store key-value pairs: std::string as key, T as value
- Implement:
    - insert(key, value)
    - get(key)
    - remove(key)
    - resize() when load factor exceeds threshold
    - contains(key) (optional but useful)

Non-functional:

- Must use your **own hash function**
- Must implement your **own collision resolution strategy**
    - Preferably an **open addressing scheme**
    - Bonus: invent a novel one or modify an existing one

---

3. Design Components

---

a. HashTable Class (Templated)

- template <typename T>
- Internally uses a **dynamic array** of entries
- Each entry stores:
    - Key (std::string)
    - Value (T)
    - Status (occupied, deleted, empty)

---

b. Entry Struct

```
template <typename T>
struct Entry {
    std::string key;
    T value;
    bool isOccupied;
    bool isDeleted;

    Entry() : key(""), value(), isOccupied(false), isDeleted(false) {}
};
```

---

c. Hash Function

- Implement your own, e.g., modified **djb2**:

```
size_t customHash(const std::string& key) const {
    size_t hash = 5381;
    for (char c : key) {
        hash = ((hash << 5) + hash) + c; // hash * 33 + c
    }
    return hash;
}
```

---

d. Collision Resolution

Pick or create:

- **Quadratic Probing**:

```
index = (hash + i^2) % capacity;
```

- **Custom Scheme** (bonus): e.g., alternating linear + quadratic

---

⚖️ **4. Rule of Five (Big Five in C++)**

Since your class uses **raw dynamic memory**, you must implement:

Method	Purpose
Destructor	Free memory
Copy constructor	Deep copy
Copy assignment operator	Deep copy existing object
Move constructor	Transfer ownership without copy
Move assignment operator	Same as above, but during assignment

---

5. C++ Class Template with Method Stubs

```
#pragma once
#include <iostream>
#include <string>
#include <stdexcept>
#include <vector>

template <typename T>
class HashTable {
private:
    struct Entry {
        std::string key;
        T value;
        bool isOccupied;
        bool isDeleted;

        Entry() : key(""), value(), isOccupied(false), isDeleted(false) {}
    };

    Entry* table;
    size_t capacity;
    size_t size;
    double loadFactorThreshold;

    size_t customHash(const std::string& key) const {
        size_t hash = 5381;
        for (char c : key) {
            hash = ((hash << 5) + hash) + c;
        }
        return hash;
```

```
    }

    size_t probe(const std::string& key, bool forInsert = false) const;

    void resize();

public:
    // Constructor
    HashTable(size_t initialCapacity = 101);

    // Big Five
    ~HashTable();                                       // Destructor
    HashTable(const HashTable& other);                  // Copy constructor
    HashTable& operator=(const HashTable& other);       // Copy assignment
    HashTable(HashTable&& other) noexcept;              // Move constructor
    HashTable& operator=(HashTable&& other) noexcept;   // Move assignment

    // Core methods
    void insert(const std::string& key, const T& value);
    T& get(const std::string& key);
    void remove(const std::string& key);
    bool contains(const std::string& key) const;

    size_t getSize() const { return size; }
    size_t getCapacity() const { return capacity; }
};
```

---

6. Testing and Verification

Test Cases:

- Insert and retrieve values (int, string, custom structs)
- Insert duplicate key (should overwrite or update)
- Delete key and re-insert
- Trigger resize
- Collision resolution (manually induce collisions)

---

7. Application: Leetcode Problem Solving

Problem 1: Two Sum

- Use your HashTable<int> to store value → index mapping
- Lookup complement in O(1) time

Problem 2: Group Anagrams

- Use HashTable<std::vector<std::string>>
- Key = sorted version of each string

---

8. Deliverables

- HashTable.h or .cpp file with complete implementation
- Test program (main.cpp) demonstrating:
    - Unit tests
    - At least 2 Leetcode problems solved using your hash table
- Optional README documenting:
    - Your hash strategy
    - Collision resolution method
    - Performance considerations