CS 271 Project 3 – Smushed Universes (Hash Table Implementation)

Instructor: Flannery Currin

Due: October 10th, 2025 by 9:00 AM

1 Learning Goals

Hash tables are often used to implement dictionaries or maps, and hash functions have additional applications in fields like cryptography. In this project, you will implement a templated hash table class and apply it to handle user logins securely. Specifically, after working on this project you should:

- Understand the relationship between the data of an element and its associated key for hashing
- Be able to compute the index in a hash table where an element belongs given its key
- Be able to maintain a hash table as elements are added and removed, handling collisions appropriately
- Be able to analyze and compare different hash functions for specific applications (e.g., handling user login)

2 Project Overview

You should complete this project in a group as assigned on Canvas and in class. You are individually responsible for the learning goals above (dividing and conquering may not be the most effective strategy here). Use the Canvas guide on using Github to help manage version control. You may discuss this assignment with your partner(s), the course TA or instructor, but the work you submit must be your group members' own work. You may use your previous projects in this class as a reference for how to write a templated class.

2.1 Element Class

Implement a templated `Element` class that supports, at minimum, the following operations:

- `get_key()`: `e.get_key()` should return the element's **numeric** key value. For example:

```
Element<string> e(``string data'', 5);
cout << e.get_key() << endl;
```

  should print `5`.

- `get_data()`: `e.get_data()` should return the element's **template** data. For example:

```
Element<string> e(``string data'', 5);
cout << e.get_data() << endl;
```

  should print `string data`.

You may also choose to implement additional methods for your Element class. This choice along with other parts of the class design are left to each group.

2.2 Hash Table

Implement a templated `HashTable` class storing `Element` objects. By default, your hash table will use hash function $h(k) = k\%m$ where m is the size of the hash table, but you should support multiple possible hash functions to complete the analysis for the usecase. Your `HashTable` class should support, at minimum, the following operations:

- `insert(data, key)`: `ht.insert(d, k)` should insert an `Element` with data `d` and key `k` into hash table `ht`. For example:

```
HashTable<string> ht(5);
ht.insert(``example'', 8);
```

  should create a hash table with 5 slots and an element with data "example" in slot $h(8) \in \{0, 1, 2, 3, 4\}$ using your hash function h. Collisions should be handled via chaining using a doubly linked list. New elements to a doubly linked list should be inserted at the head. If you want, you may use the STL's list class for this.

- `remove(data, key)`: `ht.remove(d, k)` should remove the specified `Element` from the hash table `ht`. For example, using the `ht` from the `insert` example:

```
ht.remove(``example'', 8);
```

  should result in `ht` being empty. If there is no element in the hash table at the specified key with matching data, remove should not throw an error – it should simply not modify the hash table.

- `member(data, key)`: `ht.member(d, k)` should indicate whether the hash table `ht` contains an `Element` with data `d` and key `k`. For example, using the table `ht` from the previous examples:

```
ht.insert(``test'', 28);
if(ht.member(``test'', 28)){
    cout << ``(test, 28) is a member of ht'' << endl;
}
if(ht.member(``other'', 28)){
    cout << ``(other, 28) is a member of ht'' << endl;
}
```

  should print **only** `(test, 28) is a member of ht`.

- `to_string()`: `ht.to_string()` returns a string with the elements in each doubly linked list separated by a single space and displayed as `(data, key)`. Each slot in the hash table should be separated by a new line. For example, using the table `ht` from the previous examples and hash function $h(k) = k\%m$ where m is the size of the hash table:

```
cout << ht.to_string() << endl;
```

  should print:

```
0:
1:
2:
3:(test,28)
4:
```

2.3 Unit Testing

Test each `Element` and `HashTable` function thoroughly, using the provided test file `test_hashtable_example.cpp` as a guide and starting point.

2.4 Usecase

Finally, use your `HashTable` class in `usecase.cpp` to solve the following problem. Given a csv file in which each line represents a username, password pair, use a `HashTable` to support the following interaction with the program user:

- Ask the user to enter their username and password.
- If a correct password is entered, notify the user that access has been granted.
- If an inaccurate password is entered, notify the user that access has been denied.

An example csv file has been provided (logins.csv). **N.B., passwords should not be stored in the hash table – use a cryptographic hash function to compute a key based on the password**. Passwords can be any numeric value $\in \mathbb{Z}_{10^{10}}$ while usernames can be any random string.

Your solution should be implemented in `usecase.cpp` using the following two functions:

- `HashTable<T>* create_table(string fname, int m)`
- `bool login(HashTable<T>* ht, T username, string password)`

where `fname` is the name of the csv file containing the username, password pairs and `m` is the intended size of the hash table. Your generated hash table should then be used with the `login` function where `ht` is the table from the `create_table` function, `username` is the entered username, and `password` is the entered password. Your function should return true if and only if the username and password match one of the known pairs (from the csv file). Note that your use case will only be tested when the template is set to string.

In your `usecase.cpp` file, your main function should include at least one example test case demonstrating the accuracy of your solution which allows for user input from the terminal.

2.5 Analysis

Finally, experiment with each of the following hash functions:

1. Most Significant Bits Method: let $h(k)$ be the p most significant bits of k.
2. Cormen's Multiplication Method: let x be the factional part of $k * A$, then $h(k)$ should be the floor of $m * x$. Let A be as suggested by Knuth, $A = (\sqrt{5} - 1)/2$ and m be some power of 2.

Use LaTeXto create the document `analysis.pdf` which compares the performance of these two hash functions. Indicate which you believe is a better hash function *and why*. **Use experimental data to support your position.**

3 Project Submission

On Canvas, before the deadline, you will submit **a zip** containing:

- Your `HashTable.h` and/or `HashTable.cpp` files.
- `hashtable_test.cpp` with your unit tests
- `usecase.cpp` with your usecase implementation and tests
- `analysis.pdf` with your comparison of different hash functions
- A makefile to compile your project
- Any additional files you created that are needed to compile your project using your makefile

4 Grading Scheme

The out-of-class submission will be graded using the following scheme:

Criteria	Description	Points
Completeness	Meets submission requirements (files, documentation, etc.)	2
Correctness	Passes Dr. Currin's extended tests and follows specs	4
Usecase	Contains required components and thorough testing	2
Testing	Tests should cover a range of types and cases, and logic should be sound	1
Analysis	Analysis document provides a sound argument based on data supported by submitted project	1
Total		**10**

The in-class component is worth 20 points. You should not need to study extra or memorize your out-of-class work for the in-class component. Working on the out-of-class component with the learning goals in mind is your best preparation. The in-class project component lets you test how prepared you are to apply the concepts covered by the project to a new scenario.