ECE 250 Project 1: Design Document

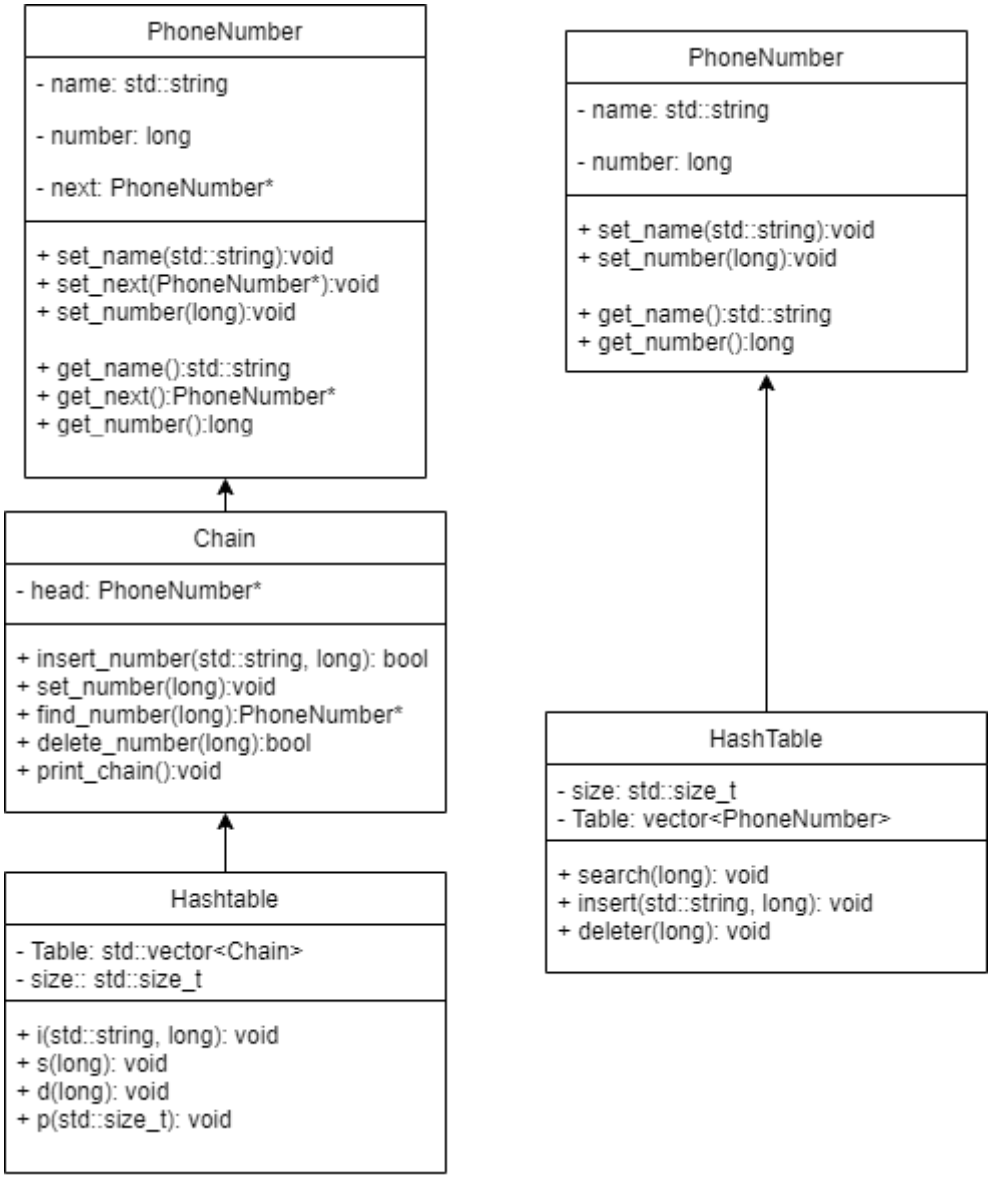
Akbar Zafar: 20832368

Overview of Classes

The classes used were a Node class called PhoneNumber. A linked list class called Chain which is a linked list of the node class. The Hashtable class implements a vector to represent a hashtable.

Based on the kind of hashing we want to implement, the hashtable class holds different objects. For chaining, the hashtable holds a linked list object(Chain). For double hashing, the hashtable holds a node object(PhoneNumber).

UML Class Diagram



Details on Design Decisions

The size of my index for my hashtable is of type std::size_t because we don't know the size of the hashtable. The constructor takes a size to initialize the vector of type Chain

for Chaining, and vector of type PhoneNumber for double hashing. For chaining, we only initialize the head pointer to nullptr in the constructor for the linked list class, further, for the node class we use number(long) and name(std::string) to initialize the object. For double hashing, we have a dummy constructor which sets the number value to -1 indicating an empty entry once the hashtable vector is declared.

The only class requiring a destructor was the Chain class since this is the only class which allocates memory dynamically. Due to this, the destructor frees the memory flagged by iterating through every node in the linked list.

Test Cases

Aside from using the provided test cases, I tested my code against specific cases. Also, I tested my code against test cases not covered by the inputs given. Some notable tests for chaining include:

Performance Evaluation

Chaining: insert(): since the insert function uses insertion sort, the worst case runtime for this function would be O(n). Also, the average time for this function is O(n) since we call a search function to ensure that the key being inserted is not already present..

Chaining: delete(): as the delete function first checks if the key exists, the worst possible time for this would be O(n), which occurs when the node to be deleted is at the end of the list. Assuming uniform hashing, the average runtime for this function is O(1).

Chaining: search(): to iterate through the linked list, the worst case runtime for this function is O(n) and that occurs when the node is at the end of the list. Assuming uniform hashing, the average runtime for this function is O(1).

Double Hashing: insert(): As the insert function first checks for duplicate keys and if the table is full, the worst case scenario is O(n) and it occurs when the table is full. Assuming uniform hashing, the runtime for this function is O(1).

Double Hashing: delete(): Assuming uniform hashing, the average runtime for the function is O(1). The worst case scenario occurs when there are n-1 collisions providing O(n).

Double Hashing: search(): Assuming uniform hashing, the average runtime for this function is O(1), the worst case scenario is O(n) when there are n-1 collisions providing O(n).