ECE 250 - PROJECT 1 - DESIGN DOCUMENT

ARJUN MEHTA – A47MEHTA – 20839709

OVERVIEW OF CLASSES

class hash (common class)

Represents the class that stores phone number and the name of the caller.

Member Variables :	phonenumber (long integer)	**stores phone number of the caller**
	callername (string)	**stores name of the caller**
Member Functions :	None	

class node (common class)

Represents the class which represents a node in a linked list. Inherits class hash.

Member Variables :	data (object)	**data in every node is an object of class hash**
	*next (pointer)	**pointer to next node**
Member Functions :	None	

class openaddressing (class for open addressing implementation)

Provides options to insert, search and delete entries. Implements hashtable with open addressing and double hashing. Inherits class hash.

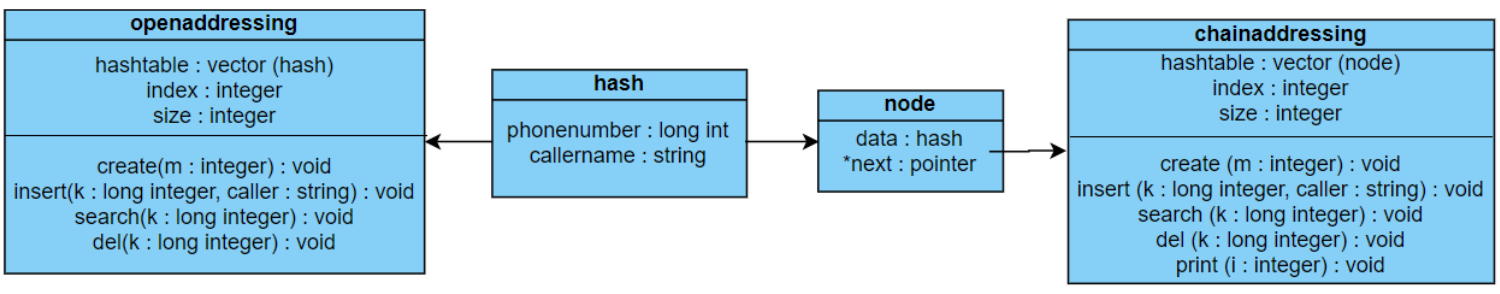
Member Variables :	hashtable (vector<hash>)	**vector of objects of class Hash**
	index (integer)	**index of the hash object in vector 'hashtable'**
	size (integer)	**size of vector**
Member Functions :	create()	**defines size of the hash table**
	insert()	**inserts the key k and the associated caller**
	search()	**searches for the key k in the table**
	del()	**deletes the key k from the table**

class chainaddressing (class for chain addressing implementation)

Provides options to insert, search, delete and print entries. Implements hashtable with chain addressing. Inherits class node (which in turn inherits hash).

Member Variables :	hashtable (vector<*node>)	**vector of pointer to a node, each node's data is an object of class hash**
	index (integer)	**index of the hash object in vector 'hashtable'**
	size (integer)	**size of vector**
Member Functions :	create()	**defines size of the hash table**
	insert()	**inserts the key k and the associated caller**
	search()	**searches for the key k in the table**
	del()	**deletes the key k from the table**
	print()	**prints the chain of keys that starts at position i**

UML CLASS DIAGRAM



DETAILS ON DESIGN DECISIONS

Constructors
class hash : set default value of phonenumber to be zero.
class node : set next = nullptr so that a newly created node always points towards a null pointer.
class openaddressing : set default size and index to zero.
class chainaddressing : set default size and index to zero.
Destructor : class node : delete next to avoid memory leaks

TEST CASES

Open Addressing with Double Hashing
Create table, insert data, search for data, delete data and then search for it again. Try to insert data into a deleted index. Test with calling functions without proper parameters to check if constructors work properly. Check if the data is stored as per double hashing implementation.
Chain addressing
Follow same testing strategy like open addressing. In addition, print the data at an index and check if correct values are displayed in ascending order.

```
n 35
i 5143332345;Mr Patrick
i 4356667876;Carlos James
s 5143332345
s 4356667876
d 5143332345
d 4356667876
i 5143332345;Rohan Mathew
n 45
i 6128789898;Mr SpongeBob
s 6128789898
exit
```

Creates and inserts data. Searches for inserted data. Tries to insert a deleted phone number. Resizes table again.

```
n 45
n 65
i 8765437777;Mrs Smith
i 2453456768;Panda Man
i 3456547789;Jack
i 6253647399;Ron
i 7165243589;Kevin
p 9
d 6253647399
p 9
d 3456547789
p 9
p 7
s 3456547789
exit
```

Chain addressing – prints all phone numbers at index 9. Keeps deleting them and prints again. Checks if they got deleted and if all numbers are in ascending order.

```
n 35
n 25
i 64527679999;Jacob
i 81728908875;Kat
s 64527679999
d 64527679999
s 64527679999
i 64527679999;Peter
s 64527679999
exit
i 2763334567;Jeremy
```

Uses similar methods like other test cases. Exits before the last statement to make sure that program gets exited and last insertion is not executed.

PERFORMANCE CONSIDERATIONS

Time complexity and time taken by each function
Create function : O(1) for open addressing and O(n) for chain addressing
Insert function : O(1)
Search funtion : O(1)
Delete function : O(1)
Print function : O(1)
Achieved constant time for insert, search, delete by not using for/while loops to implement functions. Insert is done by using double hashing and keeping track of status of position in open addressing. In chain addressing, insert data in new node and link it. Search, deletion are done using the index that we found during insertion using hash functions.