ECE250: Lab Project 2

Due Date: October 11th, 2019, 11:00 pm

1. Project Description

The goal of this project is to write a C++ implementation for hash table data structure. In this data structure, values are mapped to a position in a table using a hash function. For this project, you will implement a hash table in which collisions resolve using two different techniques: (i) open addressing using linear probing and (ii) separate chaining where the chains are unordered. Your hash function uses Division Method. This means *h(k) = k mod m*, where *k* is the key, and *m* is the size of the hash table.

We ask that you write a C++ class for each of these collision techniques. Each class should provide several services: (i) initializing/creating a hash table (ii) inserting a value into a hash table, (iii) searching for a value in a hash table, and (iv) deleting a value from the hash table.

You may use the vector class of the STL C++ library to store the hash table. For the hash table handling collision using chaining technique, you have to implement your own linked list.

2. Program Design

Write a short description of your design. You will submit this document along with your C++ solution files for marking. This document must include your design decisions. Please refer to the course website for "Programming Guidelines" and the expected content for your design document.

3. Project Requirements

Write a test program for each technique handling collision: (1) **openhttest.cpp** for open addressing using linear probing and (2) **unorderedhttest.cpp** for chaining. The test programs will read commands from standard input and write the output to standard output. These programs will respond to the commands described in this section.

Command	**Parameters**	**Description**	**Output**
n	*m*	Defines size of the hash table	**success**
i	*k*	Inserts the key *k*	**success:** if the insertion was successful **failure:** if the insertion was unable to complete since the table was full or the key was already there
s	*k*	Searches for the key *k* in the table	**found in p:** if the desired key was found in the position *p* of the hash table **not found**: if the desired key was not found
d	*k*	Deletes the key *k* from the table	**success:** if the deletion was successful **failure:** if the deletion was unable to complete since the value was not found in the table

Assuming uniform hashing, the expected average runtime for each insert (**i**), delete (**d**), and search (**s**) operation is constant. In your design document, you should also describe how you have achieved this in your implementation.

- **Test Files**

The course website contains example input files for each technique handling collisions with the corresponding output files. The files are named *test01.in*, *test02.in* and so on with the output files named *test01-chain.out*, *test01-open.out*, *test02-chain.out*, *test02-open.out*, and so on.

4. How to Submit Your Program

Once you have completed your solution and tested it comprehensively in your computer or on the lab computers, you have to transfer your files to the *eceUbuntu server* and test there since we perform the automated testing using this environment. Once you finish testing in the *eceUbuntu server*, you will create a compressed file (tar.gz) that should contain:

- A typed document (maximum two pages) describing your design. Submit this document is PDF format (*e.g.,* design.pdf).
- Two tests program, one for each techniques handling collision (**openhttest.cpp** and **unorderedhttest.cpp***)*
- Required header files and classes (ending in *.h .cpp*)
- If your implementation requires more than one file listed in g++ line for compilation, you will need to add to the design document the complete g++ command line, or provide a make file (named Makefile).

The name of your compressed file should be ***xxxxxxxx***_p***n***.tar.gz, where ***xxxxxxxx*** is your UW user id (e.g., jsmith) and ***n*** is the project number that is 2 (two) for this submission.