University of Waterloo
CS240 Fall 2020
Programming Question 3

Due Date: Wednesday, Nov 11 at 5:00pm

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every asessment in this course you must read and sign an Academic Integrity Declaration before you start working on the assessment and submit it **before the deadline of November 11th** along with your program; i.e. **read, sign and submit PQ03-AID.txt now or as soon as possible**. The agreement will indicate what you must do to ensure the integrity of your grade. If you are having difficulties with the assignment, course staff are there to help (provided it isn't last minute).

The Academic Integrity Declaration must be signed and submitted on time or the assessment will not be marked.

Please read `http://www.student.cs.uwaterloo.ca/~cs240/f20/guidelines.pdf` for guidelines on submission. Submit the file `hash.cpp` to MarkUs.

Problem 1 [10 marks]

In this programming question you are asked to implement the hashing methods described in class (listed below) to compare the number of probes required to insert and search for keys within the dictionary.

For simplicity, we will only insert unique keys (no duplicates) and also, only the key will be inserted, not their corresponding value. Keys will be non-negative integer values so you may use a sentinel value of -1 to mark a location as deleted. Use the hash functions described in class and the golden ratio constant of 0.618 where needed.

To facilitate resizing of the hash table, you may use a vector or your own dynamically allocated array. Start with a hash table of size 11.

- **Separate Chaining** - Create an array of pointers to unsorted linked lists. Newly inserted items must be inserted to the front of the linked lists.
- **Linear Probing** - Allow only a single item per slot in the hash table. Implemented with lazy deletion.
- **Double Hashing** - Similar to linear probing above except with a second hash function to compute the offset for the probe sequence.
- **Cuckoo Hashing** - Implemented with two hash functions and two tables.

You may not use any pre-existing code that would trivialize the implementation (e.g. built in data structures from STL, smart pointers, etc). You may, however, use the C++ vector data structure. If in doubt, make a private Piazza post and ask.

Implement your program in C++ and provide a main function that accepts the following commands from stdin (you may assume that all inputs are valid):

- `i key` - inserts an item with the given *key* into all 4 hash tables and prints the number of probes each method requires to find the location where *key* is inserted. Print out the number of probes each method requires on a single line, in the order listed above, followed by a newline.
  If you are not able to insert the key into all 4 hash tables, insert the key into the ones that you are able to, then perform a rehash event (specified below) on the table(s) where insert failed and then re-attempt to insert the key. In this case the number of probes should be the sum of all attempts to insert the key.
  Note: for separate chaining, the number of probes to insert the key should always simply be 1.

- `d key` - deletes the item with the given *key* from all 4 hash tables and prints the number of probes each method requires to either find the key or determine that key is not in the table.

- `s key` - searches for the item with the given *key* in all 4 hash tables and prints the number of probes each method requires to either find the key or determine that key is not in the table.

- `stats` - prints 9 int values followed by a newline. The first 4 ints correspond to the total number of probes used by each hashing method (sum of probes used by insert, delete and search). The next int is the number of items inserted into the hash tables. The last 4 ints are the size of each hash table, in the order listed above.

- `r h` - rehashes one of the tables depending on the value of h where 1 is separate chaining, 2 is linear probing, 3 is double hashing, and 4 is cuckoo hashing. To determine the new size of the table, choose the smallest prime number that is larger than double the size of the current table size. Finally print "REHASH" followed the new table size and a newline.

Your program terminates when you run out of input from stdin.
Place your entire program in the file `hash.cpp`