ECE250: Lab Project 2

Due Date: Sunday, February 17, 2019 – 11:00PM

1. Project Description

In this project, you design and implement a 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 double hashing. We will use hash functions sufficiently even so as to allow all expected constant-time operations to be **O**(1).

2. C++ Template Class

Similar to the first lab project (P0), you will also implement a *C++ template class*. A template class is a class that allows for variable types. This means that your hash table could store numbers of type *int* or *double*, depending on how the object of this class is declared:

```
DoubleHashTable<int> my_int_ht;
DoubleHashTable<double> my_double_ht;
```

The first declaration declares a hash table that will hold integers while the second declares a hash table that will hold numbers of double type. In Section 5 of this document, and in the provided source code, we refer to this variable type as **T.**

3. How to Test Your Program

We use drivers and tester classes for automated marking, and provide them for you to use while you build your solution. We also provide you with basic test cases, which can serve as a sample for you to create more comprehensive test cases. You can find the testing files on the course website. You will notice that there are separate drivers to test your class template for types *int* and *double*.

4. How to Submit Your Program

Once you have completed your solution, and tested it comprehensively, you need to build a compressed file, in tar.gz format, with should contain the file:

- DoubleHashTable.h

Build your tar file using the UNIX tar command as given below:

- tar –cvzf **xxxxxxxx**_p**n**.tar.gz DoubleHashTable.h

where **xxxxxxxx** is your **UW user id (e.g. jsmith**), and **n** is the project number which is 2 for this project. All characters in the file name must be lower case. Submit your tar.gz file using LEARN, in the drop box corresponding to this project.

5. Class Specifications

The *DoubleHashTable* class implements a hash table using double hashing to resolve collisions. Notice that the expected running time of each member function is specified in parentheses at the end of the function description. It is important that your implementation follows this requirement strictly (submissions that do not satisfy these requirements will not be given full marks). In what follows, the number of elements in the hash table is ***n*** and the size of the hash table is ***M***.

Hash Functions

The double hashing uses a hash function of the form $h(k,i) = (h_1(k) + ih_2(k))\ mod\ M$. The initial probe goes to position $h_1(k)$ (so we have i = 0); successive probe positions are offset from previous positions by amount $h_2(k)$ modulo M. In this project, $h_1(k)$ first statically casts the input to an int (see documentation for static_cast<int>), then takes this integer module M (i.e. $k\%M$) and finally adding M if the value is negative. The hash function $h_2(k)$, (that determines the offset) is derived from the integer divided by M module M, $((k/M)\%M)$. If necessary, this value should be made positive by adding M. Add 1 if the resulting value is even in order to make it odd when necessary.

Member Variables

The *DoubleHashTable* class has at least the following member variables (you may need additional member variables):

- **T** **array* - An array of objects of type **T**. The array contains the values placed in the hash table
- *state *array_state* - An array of objects of type "state" – to store the status of the bin. The state of a bin is one of three possible values: *EMPTY*, *OCCUPIED*, or *DELETED* (Why do we need to differentiate between *EMPTY* and *DELETED* entries?)
- *int count* - The number of elements currently in the hash table
- *int array_size* - The capacity of the hash table
- *int power* - This is associated with the capacity of the hash table ($array\_size = 2^{power}$)

Constructor

DoubleHashTable (int m = 5)

The constructor takes an argument m and creates a hash table with 2^m bins, indexed from 0 to $2^m - 1$. The default value of m is 5. Notice that you need to allocate and initialize two arrays, one for storing the values in the hash table, and one for storing the status of the bins.

Destructor

~DoubleHashTable ()

The destructor deletes the memory allocated for the hash table. Notice that the hash table has been represented using two arrays, and they both need to be deleted.

Accessors

This class has six accessors:

- *int size() const* - Return the number of elements currently stored in the hash table. (**O**(1))
- *int capacity() const* - Return the number of bins in the hash table. (**O**(1))
- *bool empty() const* - Return true if the hash table is empty, false otherwise. (**O**(1))
- *bool member(T const &) const* - Return true if object obj is in the hash table and false otherwise. (**O**(1))
- ***T** bin(int b) const*-Return the entry in bin b. The behaviour of this function is undefined if the bin is not filled. It will only be used to test the class with the expected locations. (**O**(1))
- void print() const - A function you can use to *print* the class for debugging purposes. This function will not be tested.

Mutators

This class has three mutators:

- *void insert(**T** const &)* - Insert the new object into the hash table in the appropriate bin as determined by the two aforementioned hash functions and the rules of double hashing. If the table is full, throw an overflow exception. (**O**(1))
- *bool remove(**T** const &)* - Remove the object from the hash table if it is in the hash table (returning false if it is not) by setting the corresponding flag of the bin to deleted. (**O**(1))
- *void clear()* - Remove all the elements in the hash table. (**O**(*M*))