CS 341 – Spring 2023
Assignment #6 – The Last Can Of Who Hash
Due: 4/24/2023

We have explored some new data structures as part of your journey in this course and we want to wrap things up (as Lebowski said, "this assignment really tied the whole course together"….or something like that) with an exploration of data structures and their performance. Each data structure has a unique and special purpose. For instance, we want to use a Binary Search Tree (or Red-Black Tree) if we want to quickly search for an element – it can be done in `log(n)` time; while insertions are much easier in a Linked List than in an Array. In this assignment, we are going to explore a Hash Table and see how it can provide the ability to add or remove an element in constant time!

Now our faithful CEO (Blue IV) has observed all of our hard work this semester and wants to reward us with a can of Who Hash – only one small problem he doesn't understand how Hash Tables work. Now he hears that we have been learning all about Hash Tables from Dr. R. He also recently learned that collisions can happen in Hash Tables and they create big headaches. What to do!? It just so happens that a certainly data structure, a Linked List, can be used to avoid collisions in Hash Tables. Now this is GREAT news because it just so happens that in Assignment #4 we created a Doubly Linked List!

Blue IV wants to see how different hashing approaches, that we have learned about in lecture, compare. Specifically he wants us to compare four techniques for solving collisions in Hash Tables: Linear Probing, Quadratic Probing, Separate Chaining, and Cuckoo Hashing (we will discuss each of these in lecture). For our data set he wants us to use the duplicates list that we created in Assignment #3 – as it just so happens that our Dictionary was essentially a Hash Table (key, value)!

For this assignment you are tasked with creating a Hash Table using our Doubly Linked List from Assignment #4 along with providing the implementation necessary to handle both Open and Closed Addressing.

A few notes about the specific requirements of the program:

- The program should load SPACE delimited (key, value) numerical data from a text file (`dictionary.txt`) generated in Assignment #3.
- The program should allow you to choose which Collision Technique to use (Linear Probing, Quadratic Probing, Separate Chaining, and Cuckoo Hashing).
    - You should then insert each (key, value) into the Hash Table using the selected technique.
    - For our Hash function we will always use modulus to perform the calculation – except in the case of Cuckoo Hashing (two hash functions).
        - `hashValue = key % size`
- You will be using an Array and a List (Doubly Linked List) to represent the Hash Table.
    - Each implementation should provide the ability to Insert, Search, Print, and Remove from the Hash Table.
- The program then should allow the user to Search, Remove, Print the Hash Table.
- The program should handle invalid cases (e.g., invalid text entry, file I/O, etc.).
- The Hash Table should be placed on the Heap (memory management).
    - The program should contain no memory leaks – make sure to use Valgrind!
        - `valgrind --log-file=valgrind.txt A6.exe`

Development Process:

For the development of your code: you should create four (4) new classes: `HashTable`, `HashEntry` `HashTableArray`, `HashTableCuckoo`, and `HashTableChaining`. For the `HashEntry` class, you can modify/use your existing `Dictionary` class from Assignment #3 if you wish. You will also be using your `DoublyLinkedList` (and associated files) from Assignment #4.

The `HashTable` Class should be a Base Class for `HashTableArray`, `HashTableCuckoo`, and `HashTableChaining` should inherit from.

The `HashTable` Class should have the following attributes and operations:

- Public:
  - Contructor
  - Destructor
  - `void insert(int key, int value) = 0;`
  - `int search(int key) = 0;`
    - This function should return the value.
  - `void remove(int key) = 0;`
  - `void print() = 0;`

The `HashEntry` Class (you can modify your `Dictionary` class from Assignment #3 if you wish here) should have the following attributes and operations:

- Private:
  - `int key_;`
  - `int value_;`
  - `Status status_;`
- Public:
  - Constructor(s)
  - Destructor
  - Accessor Method(s)
- `enum Status {EMPTY, OCCUPIED, REMOVED};`

The `HashTableArray` Class should inherit publically from our `HashTable` Class. The `HashTableArray` Class should have the following attributes and operations:

- Private:
  - `HashEntry * entry_;`
  - `int size_;`
- Public:
  - Constructor(s)
  - Destructor
  - Accessor Methods
  - `void insert(int key, int value);`
  - `int search(int key);`
  - `void remove(int key);`
  - `void print();`

*You can add a convenience method to keep track of which type of probing we are using.

The `HashTableCuckoo` Class should inherit publically from our `HashTable` Class. The `HashTableCuckoo` Class should have the following attributes and operations:

- Private:
  - `HashEntry * entry_; // Table I`
  - `HashEntry * entry2_; // Table II`
  - `int size_;`
- Public:
  - Constructor(s)
  - Destructor
  - Accessor Methods
  - `void insert(int key, int value);`
  - `int search(int key);`
  - `void remove(int key);`
  - `void print();`

*You can add convenience method(s) to calculate the two (2) hashing functions.

```
hash1 = key % size
hash2 = (key / size) % size
```

The `HashTableChaining` Class should inherit publically from our `HashTable` Class. The `HashTableChaining` Class should have the following attributes and operations:

- Private:
  - `DoublyLinkedList * entry_;`
  - `int size_;`
- Public:
  - Constructor(s)
  - Destructor
  - Accessor Methods
  - `void insert(int key, int value);`
  - `int search(int key);`
  - `void remove(int key);`
  - `void print();`

You will need to modify your `DoublyLinkedList` class as follows:

- Public:
  - `int find(int key);`
    - This will return the value of the element stored in the `LinkedNode`.
  - `void removeLinkedNode(int key);`
    - This will add the ability to remove a `LinkedNode` from our `DoublyLinkedList`.

You will need to modify all the files from Assignment #4 to store a `HashEntry` rather than just an `int` in your files. We will cover this in lecture to ensure this is done properly.

Tying this all together you will need to write a `driver` that will test your Hash Table and provide the necessary functionality as described earlier as outlined by Blue IV. Your driver program should allow for the user to input a space delimited text file (`duplicates.txt`) to be loaded and inserted into our Hash Table. Finally, you will need to create a `makefile` that properly links all of this code together and creates an executable named **A6.exe**

An example of sample output is as follows - given the following sample input file (user input in **RED**):

```
76 1
40 1
48 1
5 1
55 1
```

Your output would contain the following:

```
Welcome to Blue IV's Can of Who Hash!

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Cuckoo Hashing
5) Quit Program

Please enter your choice: 1

Please enter the size of the Hash Table you wish to create: 6

**********************************
[0]: 48
[1]: 5
[2]: 55
[3]:
[4]: 76
[5]: 40
**********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Cuckoo Hashing
5) Quit Program

Please enter your choice: 2

Please enter the size of the Hash Table you wish to create: 6

**********************************
[0]: 48
[1]: 55
[2]:
[3]: 5
[4]: 76
[5]: 40
**********************************
```

```
1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Cuckoo Hashing
5) Quit Program

Please enter your choice: 3

Please enter the size of the Hash Table you wish to create: 6

**********************************
[0]: 48
[1]: 55
[2]:
[3]:
[4]: 76<-->40
[5]: 5
**********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Cuckoo Hashing
5) Quit Program

Please enter your choice: 4

Please enter the size of the Hash Table you wish to create: 6

**********************************
[0]: 48
[1]: 55
[2]:
[3]:
[4]: 40
[5]: 5
**********************************

**********************************
[0]: 76
[1]:
[2]:
[3]:
[4]:
[5]:
**********************************
```

```
1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Cuckoo Hashing
5) Quit Program

Please enter your choice: 3

Please enter the size of the Hash Table you wish to create: 6

**********************************
[0]: 48
[1]: 55
[2]:
[3]:
[4]: 76<-->40
[5]: 5
**********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 1

Search (Please enter a Key): 15

Invalid key! Key 15 not found in table!

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 1

Search (Please enter a Key): 5

Key: 5  Value: 1

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 2

Remove (Please enter a Key): 76

Key 76 removed.
```

```
1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 3

**********************************
[0]: 48
[1]: 55
[2]:
[3]:
[4]: 40
[5]: 5
**********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Cuckoo Hashing
5) Quit Program

Please enter your choice: 4

Please enter the size of the Hash Table you wish to create: 5

<<<--- Insufficient Hash Table Size! Re-hash! --->>>

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Cuckoo Hashing
5) Quit Program

Please enter your choice: 4
```

```
     <<<<<<<<<NOTE: The below sample uses a different input file>>>>>>>>>
Please enter the size of the Hash Table you wish to create: 12

**********************************
[0]: 36
[1]:
[2]: 50
[3]: 39
[4]: 100
[5]: 53
[6]: 6
[7]: 67
[8]: 20
[9]: 105
[10]:
[11]:
**********************************

**********************************
[0]: 3
[1]:
[2]:
[3]:
[4]:
[5]:
[6]: 75
[7]:
[8]:
[9]:
[10]:
[11]:
**********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Cuckoo Hashing
5) Quit Program

Please enter your choice: 4

Please enter the size of the Hash Table you wish to create: 11

Cycle Present - Rehash!
Key Unpositioned: 105

<<<--- Insufficient Hash Table Size! Re-hash! --->>>

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Cuckoo Hashing
5) Quit Program

Please enter your choice: 5

Thank you for using Blue IV's program - Goodbye!
```

I will be grading what is located in the **master branch** of your GitHub repository. It is strongly recommended that you commit and push often! Please make use of the Git feature of leaving descriptive messages along with your commits. Be sure to add me to any repository as a collaborator so I can view and grade your submissions. Failure to do so will result in a 0 on the assignment as I will have nothing to grade!

Submission:

All assignments must be submitted on Butler GitHub (github.butler.edu). I will allow you to work on this assignment with up to one (1) other person. If you choose to work on this project with a partner you need to email me letting me know of your "group." Be sure to make note of specific contributions of each team member so I can assign grades accordingly. Both team members must submit the collaborated code into their own respective repositories. **Note:** You are NOT required to work with a partner. The name of your Butler GitHub repository must be as follows: **cs341_spring2023**

The directory structure of the repository must contain the following files:

- **driver.cpp**
- **HashTable.h** ← **Provided to you (Canvas)**
- **HashTableChaining.h**
- **HashTableChaining.cpp**
- **HashTableArray.h**
- **HashTableArray.cpp**
- **HashTableCuckoo.h**
- **HashTableCuckoo.cpp**
- **HashEntry.h**
- **HashEntry.cpp** ← **From Assignment #3**
- **LinkedList.h**
- **LinkedList.cpp**
- **DoubleLinkedList.h**
- **DoubleLinkedList.cpp**
- **LinkedNode.h**
- **LinkedNode.cpp** ← **From Assignment #4**
- **Node.h**
- **Node.cpp**
- **data.txt** ← **From Assignment #3**
- **makefile**

Each source file (`.cpp/.h`) **must** include the Honor Pledge and digital signature – if working in pairs, please ensure both students' digital signatures are present on the files.