1. I did not find any bugs or other problems
2. I believe my code satisfies the Big-O requirement
3. bool HashTable::set(key, value, permanent)
   1. if (no capacity or no buckets)
      1. return immediately
   2. otherwise determine what bucket number the association should go in
   3. if the bucket is empty
      1. create a new Node
      2. if permanent is not true
         1. call add a new DLL to the Doubly Linked List
         2. and set its address equal to Node’s m\_DLL
      3. store the new Node in the bucket
      4. increase usedCap and usedBuckets
      5. return true
   4. set curr to the first Node in the Bucket
   5. set prev to nullptr
   6. else the bucket is not empty
   7. search through the linked list
      1. if the current Node’s key equal user’s key and not permanent
         1. update Node’s value
         2. and Node’s DLL’s value
         3. move Node’s DLL to the front of the Doubly Linked List
         4. return true
      2. if the current Node’s key equal user’s key and permanent
         1. return false
      3. else
         1. update curr and prev pointers
   8. search through Linked List and did not find key
   9. create a new Node and link it to the last Node
   10. if permanent is not true
       * 1. call add a new DLL to the Doubly Linked List
         2. and set its address equal to Node’s m\_DLL
   11. update usedCap and usedBuckets
   12. return true
4. bool HashTable::touch(key)
   1. if (no capacity or no buckets)
      1. return false
   2. determine what bucket association should go in
   3. search through the linked list in the Bucket
      1. If found key and not permanent
         1. Move Node’s DLL to front of Doubly Linked List
         2. Return true
      2. If found key and permanent >>> Return false
      3. If did not find key >>> Return false
5. bool HashTable::discard(key, value)
   1. if Doubly Linked List or capacity or number of Buckets equal 0
      1. return false
   2. otherwise something to discard
   3. set key and value equal to proper values from least recently added association
   4. delete the tail of the Doubly Linked List
   5. determine bucket for key
   6. set p equal to whatever is in the bucket
   7. if first bucket is empty >>> return false
   8. if the first Node has the value
      1. set del pointer to Node
      2. set p equal to temp Node to delete’s next
      3. set bucket to p
      4. delete del
      5. update usedCapand return true
   9. otherwise not in front
      1. search through linked list and delete Node if found
      2. return true
   10. otherwise it was not found return false;