UNIVERSITY LOGO DEPARTMENT LOGO  
TERM INFORMATION COURSE NAME

**Hash Map Project**

# Overview

This hash map project is intended to provide students with an understanding of how map-based and hash-based data structures function. This project involves the implementation of a templated hash-table-based map with methods that allow for its use as an associative array.

# Structure

A hash map, a type of associative array, is a data structure that associates a value with a key. (Our dictionary could be placed in a hash map, where the keys are the words and the values are their definitions.) The map's hash function generates a hash code from the key and reduces it (usually by modulo) to an index, and the value-key pair is placed in the table at the index (called the bucket).

Suppose our dictionary's table size is 32. If the hash code for a key (word) is the ASCII value of the first character and we wanted to add the word “aardvark”, our hash code would be 97, so our hash function would return an index of 1. The word “aardvark” and its definition would be stored in bucket 1 in the table.

One problem with hash maps is collision between keys. If we also wanted to store the string “amnesia”, the hash function would generate the same hash code 97, so we’d be using bucket 1 again. If there are too many collisions, we might consider changing our hash function or the size of our table. To deal with collisions, we usually make each hash map bucket a linked list that can hold multiple pairs. When we want to store or retrieve a value, we compute the bucket of the key and then step through the bucket's list until we find the correct value.

**A common implementation of a hash map using linked lists as buckets**



# Implementation

You will write the HashMap and HashMap::Iterator classes in the *HashMap.h* file. This HashMap will use the std::hash<T> hashes, along with modulus, to determine the bucket into which the pair is placed. For example, this would compute the bucket for a given key:

int bucket = std::hash<K>()(key) % bucketCount;

HashMap<K,V>::Iterator

*Iterator()*

This is the constructor for **Iterator**.

*unsigned int getCurrentBucket() const*

Returns the number of the bucket this iterator is at currently.

*std::pair<K,V> operator\*() const*

Return the key/value pair at the iterator's current position in the map.

*Iterator& operator++()*

Pre-increment overload; advance the operator one position in the map. Return this iterator. ***NOTE***: if the iterator has reached the end of the map (past the last pair), its data should be equal to **HashMap<K,V>::end()**.

*bool operator==(Iterator const& rhs)*

Returns true it both iterators point to the same location in the map, and false otherwise.

*bool operator!=(Iterator const& rhs)*

Returns false it both iterators point to the same location in the map, and true otherwise.

HashMap<K,V>

*HashMap<K,V>(unsigned int bucketCount)*

This is the constructor for **HashMap**, which allocates the number of buckets specified in bucketCount.

*Iterator begin() const*

Return an Iterator pointing to the beginning of the map (first bucket, first key/value pair).

*Iterator end() const*

Return an Iterator pointing past the end of the map (an invalid, unique state.)

*void clear()*

Remove all key/value pairs from the map.

*void remove(K const& key)*

Remove the key, and its corresponding value, from the map.

*V& operator[](K const& key)*

***CAREFUL – this one is tricky!*** This operator overload allows the use of the indexing (square braces) operator. If the key does not exist in the map, this function default-constructs the value and places the pair in the map. In all cases, it then returns a reference to the value associated with the key. This allows the operator to be used both for assignment and for checking value (like standard arrays.)

# Submissions

Submit the following files at the end of this project:

* HashMap.h

Place them in the ROOT directory of your zip file, not in a subdirectory. Do not submit any other files.