FULL SAIL UNIVERSITY

Data Structures and Algorithms

Lab 5: Dictionary

Overview

Dictionaries are an associative container class, meaning that each piece of data in the container has two pieces of data. The first piece of data is commonly referred to as the **key**, and this dictates where the data is stored. The second piece of information is the **value**, and is typically the larger (or more complex) data.

The underlying mechanism this Dictionary uses to store its data is a hash table (or hash map). This allows blazingly fast speeds at adding, removing, and finding data. There is a cost though, and that is quite a bit of additional memory overhead.

This implementation uses a technique called separate chaining, which means that the Dictionary will consist of an array of lists that are sparsely populated. Because of this, any individual pair can be found extremely quickly, and Dictionaries are an excellent choice for very large datasets where lookups are done very often.

Things to Review

- Dynamic arrays of class objects
- std::list methods
- Rule of 3 with arrays

New Topics

- Function pointers
- Hash functions
- auto/typename

Data Members

Pair

key	The key that is used to determine the pair's position in the table
value	The value that is associated with the key

Dictionary

mTable	A dynamic array of lists that store pairs. Each index of this array is a single list, and will be referred to throughout as a **bucket**.
mNumBuckets	The number of elements in the table
mHashFunc	A pointer to the hash function **This function will always return a valid index to the table**

Methods

Pair Constructor

- Assign the members to the values passed in

Dictionary Constructor

- Dynamically allocate the table, based on the supplied number of buckets
    - *Look at the type of pointer the table is if you are having problems*
- Assign your other data members to the passed-in values

Destructor

- Cleans up all dynamically allocated memory
    - You do not need to remove the data from each individual list

Clear

- Empty all of the data from each bucket
- **Do not delete the table or reset any of the data members**

Insert

- Add (or update) a Pair, based on the key value provided
- Call the hash function to find out which bucket to interact with
- **Existing key**
    - If the key is already in the bucket, update the associated value
- **If the key is not in the bucket**
    - Create a **Pair** and add it to the bucket
- *Remember to look through the entire bucket before adding/updating*

Find

- Checks to see if a key is present or not
- Call the hash function to find out which bucket to interact with
- **Key found**
    - Return the address of the pair's value
- **Key not found**
    - Return a null pointer
- *Remember to look through the entire bucket before returning*

Remove

- Removes a pair from the Dictionary if present
- Call the hash function to find out which bucket to interact with
- **Key found**
    - Erase the pair from the bucket and return true to indicate success
- **Key not found**
    - Nothing to remove, so only need to return false
- *Remember to look through the entire bucket before returning*

Assignment Operator

- Assigns all values to match those of the object passed in
- Cleans up existing memory before the deep copy
    - *Clear doesn't work the same as the previous labs*
- Deep copy the table
    - This is an array. Look at previous examples of deep-copying arrays
- Shallow copy the other data members

Copy Constructor

- Creates a copy of the object passed in
- Deep copy the table
    - This is an array. Look at previous examples of deep-copying arrays
- Shallow copy the other data members