# Introduction

This assignment had us implement three (3) types of hash tables. It is programmed in C++ with Microsoft Visual Studio 2019 Community Edition using the version ISO C++14 Standard. It does NOT utilize generic class templates to have a hash<int, int> or a hash<string, string> as there is an issue with Microsoft Visual Studio 2019 Community Edition implementing templated classes with generic types. We randomly selected 100 numbers with a range from 1-1000 for our usage in this assignment.

## C++ Programming Guide

Download the source code from github. Location is <https://github.com/roddeval/ms549_hash_table_rod_devalcourt>

Build this solution! You might have to modify the include path to console application and unit test project in this solution. You also might have to modify the additional libraries in the linker settings for it to build. There is a hard coded path in the ms549\_hash\_table\_rod\_devalcourt.cpp methods for Chained(int debug), DoubleHash(int debug), and LinearProbing(int debug).

In Microsoft Visual Studio 2019 Community Edition create a new project, select console application.

Add a include directory to DLL\_HASHTABLE project so that Dll\_HashTable.h can be included in your new project.

Add a reference to the console application project by right-clicking, choose properties, then add a reference to the Dll\_HASHTABLE.lib file. This is very important as if you can’t see the Dll\_HASHTABLE.lib to link it to the ms549\_hash\_table\_rod\_devalcourt project the build will fail.

In your console application include “Dll\_HashTable.h”

The project will build ms549\_hash\_table\_rod\_devalcourt.exe which is what you run to see the output.

Run your console application (you should see something similar to this output):

A screenshot of a computer screen

Description automatically generated

This is the display for the first method main calls with is the Chained has method. There are calls after this to DoubleHash and to LinearProbing.

## Class Diagram

Please note the green arrow(s) the HashNode object is returned as a HashNode\* from the Retrieve methods in each of the classes Hash, LinearProbingHash, and DoubleHashedHashTable.

Graphical user interface, diagram

Description automatically generated

ClassDiagram.dc is included in the DLL\_HASHTABLE project.

**class HashNode**

Defined in header Dll\_HashTable.h

// class HashNode was primarily used to store key/value pairs for class LinearProbingHashTable

// initially. However it is used in the other classes primarily in the

// Retrieve method as its return value. All member variables and methods are public.

class HashNode

{

};

HashNode is a class that contains an integer variable named key, an integer variable named value, and a HashNode constructor. This multipurpose class allows us to implement an object to hold a key/value pair.

**Member types**

|  |  |
| --- | --- |
| **Member type** | **Definition** |
| Int key | Holds a key value. |
| int value | Holds the value the key represents |

**Public Member functions**

|  |  |  |
| --- | --- | --- |
| HashNode(int key, int value) |  | Constructor method that allows you to hold a key/value pair. |

**class Hash**

Defined in header Dll\_HashTable.h

// class Hash is a chained hash table.

class Hash

{

};

Hash is a class that performs a chained hash tree implementation. It has private data members, and public methods.

**Private Member types**

|  |  |
| --- | --- |
| **Member type** | **Definition** |
| list<int>\* table; | the hash table is represented as a list<int> \* [mnNumIndexes]  the method Initialize does this |
| int mnNumIndexes; | // represents how many key/value pairs you would like in the hash table  // represented by list<int>\* being the value and list<int> \* [hashed key] as the value for key |
| int mnDebug; | used for a debug flag |

**Public Member functions**

|  |  |
| --- | --- |
| Hash() | Default constructor. Calls Initialize() |
| Initialize(int size, int debug) | Initializes member variables |
| GetHash(int key) | Hash method for key's value. The hashed value of key is returned as an int |
| Insert(int key) | Insert method for key's value |
| Remove(int key) | Remove method for key's value |
| Display() | Displays the contents of the hash table to the screen |
| Retrieve(int key) | retrieve a searched for value as a HashNode (key,value) pair. A HashNode\* is the return value. |

**class LinearProbingHashTable**

Defined in header Dll\_HashTable.h

// class LinearProbingHashTable is a linearly probing hash table implementation.

class LinearProbingHashTable

{

};

**Private Member types**

|  |  |
| --- | --- |
| **Member type** | **Definition** |
| const int MAXCAP | This constant is set to 20000 currently |
| HashNode\*\* table | Represents how many key/value pairs you would like in the hash table represented by list<int>\* being the value and list<int> \* [hashed key] as the value for key |
| int mnSize | Holds the current size of the hash table array. It is set to zero (0) in the member method Initialize |
| int mnDebug; | Used for a debug flag |

**Public Member functions**

|  |  |  |
| --- | --- | --- |
| LinearProbingHashTable(int debug) | Default constructor. Calls Initialize(debug) | |
| Initialize(int debug) | Initializes member variables including the hash table itself | |
| GetHash(int key) | hash method used for the array index of HashNodes. The return value is an int. | |
| Insert(int key, int value) | Lets you insert a key and value pair into the hash table  this method calls GetHash with the value of key which is used as the array index. the key/value pair is stored in the HashNode pointed to by table\*\*. The return value for this method is a HashNode\*. | |
| Retrieve(int key) | With a key we can retrieve the HashNode key/value pair  this method calls the GetHash method to use in the array index. The return value for this method is a HashNode\*. | |
| Remove(int key) | This method removes a key/value from the hash table. Return value is an int. | |
| sizeOfTable() | Returns the current size of the table. Return value is an int. | |
| isEmpty() | Returns whether or not the hashtable is empty. Return value is a bool | |
| Display() | This method displays the contents of the hash table to the console screen | |
| **enum EntryType**  Defined in header Dll\_HashTable.h  The EntryType values of Legi and Emp are used by the Entry struct that is used by the double hashed table | |  | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **struct Entry**  **Defined in header Dll\_HashTable.h**  struct Entry is used to hold the value you insert into the double hashed hash table class. int e is the datapoint, i.e. the value for a key. info is an enum type used in the methods of this class  struct Entry  {  int e;  enum EntryType info;  };  **Public Member types**   |  |  | | --- | --- | | **Member type** | **Definition** | | int e | This holds the element in the array and is the value in a key value pair. | | enum EntryType info | Represents the type Legi or Emp | |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **struct Table**  **Defined in header Dll\_HashTable.h**  struct Table is used by the double hashed hash table to store the hashed entries int s is the size of the table, Entry\* tab is used like an array where the array indexed/key is the double hashed key for a value.  struct Table  {  int s;  Entry\* tab;  };  **Public Member types**   |  |  | | --- | --- | | **Member type** | **Definition** | | int s | Represents the size of the table that Entry\* tab is to hold. | | Entry\* tab | This is the table of Entry\* that is essentially an array of pointers. | |  |

**class DoubleHashedHashTable**

Defined in header Dll\_HashTable.h

DoubleHashedHashTable is a double hashed hash table implementation.

it performs rather well especially with a large number of values (10,000).

class DoubleHashedHashTable

{

};

**Public Member types**

|  |  |
| --- | --- |
| **Member type** | **Definition** |
| Table\* table | public member variable that is double hashed. It uses a struct Table\* |

**Private Member types**

|  |  |
| --- | --- |
| **Member type** | **Definition** |
| int maxSize | This member variable is set, or initialized, in the constructor.  So if you use a rather large number, say 1,000,000 you might end up with 2,000,000 hashed key value pairs. |
| int mnDebug | Used for a debug flag |

**Public Member functions**

|  |  |
| --- | --- |
| DoubleHashedHashTable(int size, int debug) | Default constructor. Calls Initialize(int size, int debug) |
| Initialize(int size, int debug) | Initialize member values |
| int Hash1(int k, int s) | Hash method #1, return an int |
| int Hash2(int k, int s) | Hash method #2, returns an int |
| SearchKey(int k) | Returns a double hash search key given a value k. Returns an int |
| Insert(int k) | Insert method to double hashed hash table. The parameter k is the value you’re inserting to the hash table |
| Display() | Method to display the contents of the double hashed hash table |
| Retrieve(int searchFor) | Retrieve a searched for value as a HashNode (key,value) pair. Returns a HashNode\* representing the key/value pair the parameter searchFor represents the value. |
| Remove(int value) | Remove a value from the double hashed hash table |
| ReHash() | Rehash doubles the size of the hash table it was initialized with |