CMPSC 122 Lab 15 Report

Code

Code for hashtable.h:

// Programmer: Yizhou Wang

// Section: 2

// Lab: 15

// Date: April 13, 2014

// Description: Chained Hash Table Implementation

#include <iostream>

#include "iomanip"

using namespace std;

int const NUMOFSLOTS = 13; //size of this hashtable table

struct NodeType; //linked list structure used for chaining

typedef NodeType \* NodePtr; //make the the pointer of a NodeType node point to the

// next NodeType node

struct NodeType

{

int key; //key stored in the NodeType node

NodePtr next; //the pointer of a NodeType node pointing to the

// next NodeType node

};

class hashtable

{

public:

hashtable();

//POST: a default hashtable object is constructed

int hashFunction(int k);

//PRE: k is initialized, and k >= 0

//POST: FCTVAL = k mod NUMOFSLOTS

void insertion(int new\_key);

//PRE: new\_key initialized and is not in some list in this hashtable

//POST: new\_key is inserted at the head of the correct list of this hashtable

int search(int k);

//PRE: k is initialized

//POST: FCTVAL = the distance from head to k in the linked list pointed to by

// slot[hashFunction(k)], if k exists in this hashtable;

// FCTVAL = -1 if k doesn't exist in any list of this hashtable

double loadcalculate();

//POST: FCTVAL = the load factor of this hashtable

void drawing();

//POST: this hashtable is drawn to the console in a neat fashion

private:

NodePtr slot[NUMOFSLOTS]; //a static array of pointers holding this hashtable

int numofElements; //the logical size of this hashtable

};

Code for hashtable.cpp:

// Programmer: Yizhou Wang

// Section: 2

// Lab: 15

// Date: April 13, 2014

// Description: Chained Hash Table Implementation

#include "hashtable.h"

hashtable::hashtable()

//POST: a default hashtable object is constructed

{

for (int i=0; i < NUMOFSLOTS; i++) //all the slots of this hashtable are initialized

// to null

{

slot[i] = NULL;

}

numofElements = 0; //initialize the load factor of hash table

};

int hashtable::hashFunction(int k)

//PRE: k is initialized, and k >= 0

//POST: FCTVAL = k mod NUMOFSLOTS

{

return k % NUMOFSLOTS; //return the slot number of the slot k is going to

}

void hashtable::insertion(int new\_key)

//PRE: new\_key initialized and is not in some list in this hashtable

//POST: new\_key is inserted at the head of the correct list of this hashtable

{

//DATA DICTIONARY

NodePtr temp; //a temporary NodeType pointer

// holding new\_key

if (slot[hashFunction(new\_key)] == NULL) //if the list of the slot is empty

{

slot[hashFunction(new\_key)] = new NodeType; //allocate a new NodeType node

slot[hashFunction(new\_key)] -> key = new\_key; //store new\_key as the key of the

// NodeType node

slot[hashFunction(new\_key)] -> next = NULL; //point the pointer to NULL

}

else

{

temp = new NodeType; //allocate a NodeType node holding

// new\_key

temp -> key = new\_key;

//make temp the new head node of the list

temp -> next = slot[hashFunction(new\_key)];

slot[hashFunction(new\_key)] = temp;

}

numofElements ++; //increase logical size of this

// hashtable by 1

};

int hashtable::search(int k)

//PRE: k is initialized

//POST: FCTVAL = the distance from head to k in the linked list pointed to by slot[hashFunction(k)],

// if k exists in this hashtable;

// FCTVAL = -1 if k doesn't exist in any list of this hashtable

{

//DATA DICTIONARY

NodePtr cur; //a pointer pointing to the node we are intestigating

int count; //a counter counting the position of k in the list

bool found; //an indicator whether k is in the list of this hashtable

cur = slot[hashFunction(k)];

count = 0;

found = false;

while (cur != NULL && found == false) //traverse the link list to its end or intil finding k

{

if (cur -> key == k)

{

found = true;

}

cur = cur -> next;

count ++;

}

if(found == true) //if k is found in the list of this hashtable

{

return count;

}

else //if k is not found in the list of this hashtable

{

return -1;

}

};

double hashtable::loadcalculate()

//POST: FCTVAL = the load factor of this hashtable with decimal digits

{

return (numofElements \* 1.0 / NUMOFSLOTS);

}

void hashtable::drawing()

//POST: this hashtable is drawn to the console in a neat fashion

{

//DATA DICTIONARY

NodePtr cur; //a pointer pointing to the node we are

// intestigating

cout << "Here below is the hash table"

<< " you have implementated:" << endl;

for (int i=0; i < NUMOFSLOTS; i++) //print slot[NUMOFSLOTS] and their lists

// line by line

{

cout << "Slot " << setw(2) << i << ":";

cur = slot[i];

if (cur == NULL) //if there is no list pointed to by the

// slot, draw "/"

{

cout << "/";

}

else //if there is a list pointed to by the

// slot

{

while (cur != NULL) //traverse the list and print every node

// connected by arrows

{

cout << " --> " << setw(3) << cur -> key;

cur = cur -> next;

}

}

cout << endl;

}

};

Code for main.cpp:

// Programmer: Yizhou Wang

// Section: 2

// Lab: 15

// Date: April 13, 2014

// Description: Chained Hash Table Implementation

#include <iostream>

#include "hashtable.h"

using namespace std;

int main()

{

//test1

cout << "\*\*\*Test1: initialize a new hash table" << endl;

hashtable table1;

table1.drawing();

cout << "The load factor of the current hash table is: " << setprecision(4)

<< table1.loadcalculate() << endl;

cout << endl;

//test2

cout << "\*\*\*Test2: insert integers from 1 to 8 (inclusively) "

<< "into the hash table and " << "try to find 50" << endl;

for (int i=1; i <= 8; i++)

{

table1.insertion(i);

}

table1.drawing();

cout << "The load factor of the current hash table is: "

<< table1.loadcalculate() << "." << endl;

//look for 50

cout << "Does 50 exist in this hash table?" << endl;

if (table1.search(50) == -1) //if 50 doesn't exist in this hashtable

{

cout << "No, there is no 50 in the hash table.";

}

else //if 50 exists in this hashtable

{

cout << "Yes, its position in the list pointed to by slot "

<< 50 % NUMOFSLOTS << " is " << table1.search(50) << ".";

}

cout << endl << endl;

//test3

cout << "\*\*\*Test3: insert 50, 60, 70, 80 into the hash table,"

<< " and try to find 50, 129, 2" << endl;

table1.insertion(50);

table1.insertion(60);

table1.insertion(70);

table1.insertion(80);

table1.drawing();

cout << "The load factor of the current hash table is: "

<< table1.loadcalculate() << "." <<endl;

//look for 50

cout << "Does 50 exist in this hash table?" << endl;

if (table1.search(50) == -1) //if 50 doesn't exist in this hashtable

{

cout << "No, there is no 50 in the hash table.";

}

else //if 50 exists in this hashtable

{

cout << "Yes, its position in the list pointed to by slot "

<< 50 % NUMOFSLOTS << " is " << table1.search(50) << ".";

}

cout << endl << endl;

//look for 129

cout << "Does 129 exist in this hash table?" << endl;

if (table1.search(129) == -1) //if 129 doesn't exist in this hashtable

{

cout << "No, there is no 129 in the hash table.";

}

else //if 129 exists in this hashtable

{

cout << "Yes, its position in the list pointed to by slot "

<< 129 % NUMOFSLOTS << " is " << table1.search(129) << ".";

}

cout << endl << endl;

//look for 2

cout << "Does 2 exist in this hash table?" << endl;

if (table1.search(2) == -1) //if 2 doesn't exist in this hashtable

{

cout << "No, there is no 2 in the hash table.";

}

else //if 2 exists in this hashtable

{

cout << "Yes, its position in the list pointed to by slot "

<< 2 % NUMOFSLOTS << " is " << table1.search(2) << ".";

}

cout << endl << endl;

return 0;

}

Sample Runs

\*\*\*Test1: initialize a new hash table

Here below is the hash table you have implementated:

Slot 0:/

Slot 1:/

Slot 2:/

Slot 3:/

Slot 4:/

Slot 5:/

Slot 6:/

Slot 7:/

Slot 8:/

Slot 9:/

Slot 10:/

Slot 11:/

Slot 12:/

The load factor of the current hash table is: 0

\*\*\*Test2: insert integers from 1 to 8 (inclusively) into the hash table and try to find 50

Here below is the hash table you have implementated:

Slot 0:/

Slot 1: --> 1

Slot 2: --> 2

Slot 3: --> 3

Slot 4: --> 4

Slot 5: --> 5

Slot 6: --> 6

Slot 7: --> 7

Slot 8: --> 8

Slot 9:/

Slot 10:/

Slot 11:/

Slot 12:/

The load factor of the current hash table is: 0.6154.

Does 50 exist in this hash table?

No, there is no 50 in the hash table.

\*\*\*Test3: insert 50, 60, 70, 80 into the hash table, and try to find 50, 129, 2

Here below is the hash table you have implementated:

Slot 0:/

Slot 1: --> 1

Slot 2: --> 80 --> 2

Slot 3: --> 3

Slot 4: --> 4

Slot 5: --> 70 --> 5

Slot 6: --> 6

Slot 7: --> 7

Slot 8: --> 60 --> 8

Slot 9:/

Slot 10:/

Slot 11: --> 50

Slot 12:/

The load factor of the current hash table is: 0.9231.

Does 50 exist in this hash table?

Yes, its position in the list pointed to by slot 11 is 1.

Does 129 exist in this hash table?

No, there is no 129 in the hash table.

Does 2 exist in this hash table?

Yes, its position in the list pointed to by slot 2 is 2.

Program ended with exit code: 0