**CS 700-34 HashTable Strings**

**Student:** Shuhua Song

**Due Date: Submission Data:**

Soft Copy: 02/18/2020 Soft Copy: 02/18/2020

Hard Copy: 02/20/2020 Hard Copy: 02/18/2020

**Algorithm Steps:**

1. **Main()**

Step 1: Open input file ‘inFile’ by using argv[1]

Open output file ‘outFile1’ and ‘outFile2’ by using argv[2] and argv[3];

Step 2: Create a HashTable array and sign every bucket point to the dummy ;listNode, with value (“dummy”, null);

Step 3: storeDataIntoHashTable(inFile, outFile2);

Step 4: Print the Hash Table(outFile1);

Step 5: Close all files;

1. Function **storeDataIntoHashTable(inFile, outFile)** Steps;

Step 1: Read one string from inFile;

Step 2: Create a new node by using the new data, listNode(data, 1, null);

Step 3: Get index by calling function Doit(data);

Step 4: Find the correct bucket in HashTable according to the index

Step 5: Insert the newNode into the linkedlist pointed by the pointer in this bucket.

Step 6: PrintList(index, outFile2) for debugging

Step 7: repeat step1-step6 until the end of inFile;

1. Function **listInsert(newNode**) steps

Step 1: Find the proper position ‘spot’ in the linkedlist for this new node to insert

Step 2: If spot != nullptr, then newNode’s next = spot.next, spot.next = newNode

1. Function **findspot(newNode)** Steps

Step 1: set the spot = listHead

Step 2: if spot’s next != null and spot’s next’s data < newNode’s data

spot = spot’next

Step 3: Repeat step2 until condition failed

Step 4: if spot’s next’s data = newNode’s data

spot’s next count++;

spot = null;

Step 5: return spot;

1. Function **printHashTable**(outFile)

Step 1: set index = 0;

Step 2: printList(index, outFile)

Step 3: index++;

Step 4: repeat step2 to steps while index < bucketSize

1. printList (index,outFile)

Step 0: output the completed to outFile: “hashTable[index]->”

Step 1: printSpot = hashTableAry[index]

Step 2: printNode(printSpot)

printSpot=printSpot’s next

Step3: repeat step 2 while printNode!=null

**Code:**

*// project3.cpp*

*// Created by Shuhua Song on 2/18/20.*

*// Copyright © 2020 Shuhua Song. All rights reserved.*

#include <iostream>

#include <string>

#include <fstream>

**using** **namespace** std;

**class** listNode {

**public**:

string data;

**int** count;

listNode \*next;

listNode() {

data = "dummy";

count = 0;

next = **nullptr**;

}

listNode(string data, **int** count, listNode \*next) {

**this**->data = data;

**this**->count = count;

**this**->next = next;

}

**void** printNode(ofstream &outFile) {

outFile << "->(" << **this**->data << ", " << **this**->count << ", " << **this**->next->data << ")";

}

};

**class** HashTable {

**private**:

**int** bucketSize;

listNode \*\*hashTableAry;

**public**:

HashTable() {

bucketSize = 29;

hashTableAry = **new** listNode \*[bucketSize];

*//hashTableAry = new listNode\*[bucketSize];*

**for** (**int** i = 0; i < bucketSize; i++) {

hashTableAry[i] = **new** listNode();

}

}

**void** storeDataIntoHashTable(ifstream &inFile, ofstream &outFile) {

**if** (inFile.is\_open()) {

**while** (!inFile.eof()) {

string newData;

inFile >> newData;

**if** (inFile.eof()) **break**;

**int** index = Doit(newData);

listNode \*listHead = hashTableAry[index];

listNode \*newNode = **new** listNode(newData, 1, **nullptr**);

listInsert(listHead, newNode);

printList(index, outFile);

}

}

}

**int** Doit(string data) {

**unsigned** **int** val = 1;

**for** (**int** i = 0; i < data.length(); i++) {

val = val \* 32 + (**int**) data[i];

}

**int** index = val % 29;

**return** index;

}

**void** listInsert(listNode \*listHead, listNode \*newNode) {*//this already include the case of listHead->*

listNode \*spot = findSpot(newNode, listHead);

**if** (spot != **nullptr**) {

newNode->next = spot->next;

spot->next = newNode;

}

}

listNode \*findSpot(listNode \*newNode, listNode \*listHead) {

listNode \*Spot = listHead;

**while** (Spot->next != **nullptr** && Spot->next->data.compare(newNode->data) < 0) {*//need to check if Spot->next is nullptr*

Spot = Spot->next;

}

**if** (Spot->next != **nullptr** && Spot->next->data.compare(newNode->data) == 0) {

Spot->next->count++;

**return** **nullptr**;

}

**return** Spot;

}

**void** printHashTable(ofstream &outFile) {

**int** index = 0;

**while** (index < bucketSize) {

printList(index, outFile);

index++;

}

}

**void** printList(**int** index, ofstream &outFile) {

outFile << "hashTable[" << index << "]" ;

listNode \*printSpot = hashTableAry[index];

**while** (printSpot->next != **nullptr**) { *//remember the dummy node is null, so we can't add printSpot!=null*

printSpot->printNode(outFile); *//*

printSpot = printSpot->next;

}

outFile << "->(" << printSpot->data << ", " << printSpot->count << ", " << "NUll" << ")";

outFile << "->NULL\n";

}

~HashTable(){

**delete**[] hashTableAry;

}

};

**int** main(**int** argc, **char** \*argv[]) {

ifstream inFile(argv[1]);

ofstream outFile1(argv[2]);

ofstream outFile2(argv[3]);

HashTable \*myHashTable = **new** HashTable();

outFile2 << "Debug the HashTable: " << endl;

myHashTable->storeDataIntoHashTable(inFile, outFile2);

*// outFile1 << "The Completed HashTable: " << endl;*

myHashTable->printHashTable(outFile1);

inFile.close();

outFile1.close();

outFile2.close();

**return** 0;

}

**Output:**

Completed LinkedList:

hashTable[0]->(dummy, 0, Huihui)->(Huihui, 1, Khoi)->(Khoi, 1, Rashad)->(Rashad, 1, Seth)->(Seth, 1, Shadman)->(Shadman, 1, Wei)->(Wei, 1, Yangfan)->(Yangfan, 1, Yechiel)->(Yechiel, 1, NUll)->NULL

hashTable[1]->(dummy, 0, Colin)->(Colin, 1, Denny)->(Denny, 1, Michael)->(Michael, 1, NUll)->NULL

hashTable[2]->(dummy, 0, Christina)->(Christina, 1, Niraj)->(Niraj, 1, Xihao)->(Xihao, 1, NUll)->NULL

hashTable[3]->(dummy, 0, Conghui)->(Conghui, 1, Evgeniia)->(Evgeniia, 1, Han)->(Han, 1, Laert)->(Laert, 1, NUll)->NULL

hashTable[4]->(dummy, 0, NUll)->NULL

hashTable[5]->(dummy, 0, Heesun)->(Heesun, 1, Kevin)->(Kevin, 1, Lin)->(Lin, 1, Patricio)->(Patricio, 1, NUll)->NULL

hashTable[6]->(dummy, 0, Cesar)->(Cesar, 1, Marco)->(Marco, 1, Matthew)->(Matthew, 1, NUll)->NULL

hashTable[7]->(dummy, 0, Mohebullah)->(Mohebullah, 1, Peter)->(Peter, 1, Rani)->(Rani, 1, NUll)->NULL

hashTable[8]->(dummy, 0, Angelo)->(Angelo, 1, Brian)->(Brian, 2, NUll)->NULL

hashTable[9]->(dummy, 0, Joel)->(Joel, 1, Qi)->(Qi, 1, Sim)->(Sim, 1, Taejoon)->(Taejoon, 1, NUll)->NULL

hashTable[10]->(dummy, 0, Bee)->(Bee, 1, Jason)->(Jason, 1, NUll)->NULL

hashTable[11]->(dummy, 0, Harmandeep)->(Harmandeep, 1, Thurein)->(Thurein, 1, ZhengZhong)->(ZhengZhong, 1, NUll)->NULL

hashTable[12]->(dummy, 0, Jacb)->(Jacb, 1, Yinyu)->(Yinyu, 1, NUll)->NULL

hashTable[13]->(dummy, 0, Yifei)->(Yifei, 1, NUll)->NULL

hashTable[14]->(dummy, 0, Ba)->(Ba, 1, Esteban)->(Esteban, 1, Jiayu)->(Jiayu, 1, Resfred)->(Resfred, 1, NUll)->NULL

hashTable[15]->(dummy, 0, Gregory)->(Gregory, 1, Kelvin)->(Kelvin, 1, Phillip)->(Phillip, 1, Prince)->(Prince, 1, NUll)->NULL

hashTable[16]->(dummy, 0, Zhiheng)->(Zhiheng, 2, NUll)->NULL

hashTable[17]->(dummy, 0, Kenny)->(Kenny, 1, Pinpin)->(Pinpin, 1, Win)->(Win, 1, NUll)->NULL

hashTable[18]->(dummy, 0, Brandon)->(Brandon, 2, Cheng)->(Cheng, 1, Russell)->(Russell, 2, Youyia)->(Youyia, 1, NUll)->NULL

hashTable[19]->(dummy, 0, Ben)->(Ben, 1, Karamvir)->(Karamvir, 1, Yuhuan)->(Yuhuan, 1, NUll)->NULL

hashTable[20]->(dummy, 0, Jasmin)->(Jasmin, 1, Jiaxin)->(Jiaxin, 1, Luis)->(Luis, 1, NUll)->NULL

hashTable[21]->(dummy, 0, Eleftherios)->(Eleftherios, 1, Justin)->(Justin, 1, Qisheng)->(Qisheng, 1, NUll)->NULL

hashTable[22]->(dummy, 0, Emanuel)->(Emanuel, 1, Hishaam)->(Hishaam, 1, Jiade)->(Jiade, 2, Shuhua)->(Shuhua, 1, NUll)->NULL

hashTable[23]->(dummy, 0, Danielle)->(Danielle, 1, Logan)->(Logan, 1, Oscar)->(Oscar, 1, Yuan)->(Yuan, 1, NUll)->NULL

hashTable[24]->(dummy, 0, Chandra)->(Chandra, 1, Christopher)->(Christopher, 2, Joseph)->(Joseph, 1, Juan)->(Juan, 1, Robin)->(Robin, 1, NUll)->NULL

hashTable[25]->(dummy, 0, NUll)->NULL

hashTable[26]->(dummy, 0, Aaron)->(Aaron, 1, Jeffrey)->(Jeffrey, 1, Jianwei)->(Jianwei, 1, NUll)->NULL

hashTable[27]->(dummy, 0, Andres)->(Andres, 2, Taeyong)->(Taeyong, 1, NUll)->NULL

hashTable[28]->(dummy, 0, Akshar)->(Akshar, 1, NUll)->NULL