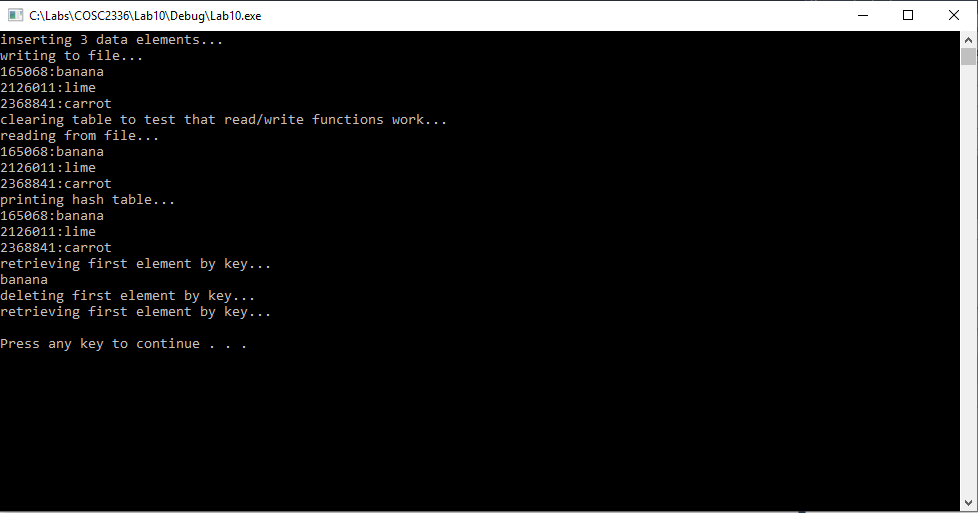
Name: William A. Brannon

Assignment: Lab Assignment Report #10

Date Due: Monday, April, 08, 2019

Class: Fundamentals of Programming III Section #1

**Program Output**



**Lab10.cpp**

// Lab 10: Hashing

// By: William Brannon on 04/08/2019

#include "stdafx.h"

#include <iostream>

#include <vector>

#include <algorithm>

#include <fstream>

#include <string>

using namespace std;

class hashCell {

public:

hashCell(int h, string v) {

hash = h;

value = v;

};

int hash;

string value;

bool operator <(const hashCell& r) const {

if (hash < r.hash) {

return true;

}

else {

return false;

}

}

};

class hashTable {

public:

vector<hashCell> nodes;

vector<int> EratosthenesSieve;

void setSieve(int n) {

EratosthenesSieve.clear();

for (int i = 2; i <= n; i++) {

EratosthenesSieve.push\_back(i);

}

vector<int> toClear;

for (int p: EratosthenesSieve) {

bool found = false;

for (int i : toClear) {

if (i == p) {

found = true;

}

}

if (!found) {

double compareVal = pow(p, 2);

for (int j : EratosthenesSieve) {

if (j >= compareVal) {

if (j % p == 0) {

toClear.push\_back(j);

}

}

}

}

}

for (int k : toClear) {

for (unsigned int i = 0; i < EratosthenesSieve.size(); i++) {

if (k == EratosthenesSieve.at(i)) {

EratosthenesSieve.erase(EratosthenesSieve.begin() + i);

}

}

}

};

int getSieve(int i) {

return EratosthenesSieve.at(i);

};

int getHash(string key, int j) {

unsigned long hash = 0;

for (unsigned int i = 0; i < key.length(); i++)

{

hash = (hash \* getSieve(i+j)) + key.at(i);

}

return hash;

}

void insert(string key, string value) {

int hash = getHash(key,0);

hashCell temp(hash, value);

int i = 1;

for (hashCell node : nodes) {

if (node.hash == temp.hash) {

temp.hash = getHash(key, i); //rehash with offset if hash key already exists

i++; //increment offset for if needing multiple rehashing

}

}

nodes.push\_back(temp);

std::sort(nodes.begin(), nodes.end());

}

string retrieve(string key) {

string temp = "";

int hash = 0;

for (unsigned int i = 0; i < nodes.size(); i++) {

for (unsigned int j = 0; j < nodes.size(); j++) {

hash = getHash(key, j);

hashCell tempCell = nodes.at(i);

if (tempCell.hash == hash) {

temp = tempCell.value;

}

}

}

return temp;

}

void deleteCell(string key) {

int hash = 0;

for (unsigned int i = 0; i < nodes.size(); i++) {

for (unsigned int j = 0; j < nodes.size(); j++) {

hash = getHash(key, j);

hashCell tempCell = nodes.at(i);

if (tempCell.hash == hash) {

nodes.erase(nodes.begin() + i);

}

}

}

}

void readFile() {

ifstream fin;

string line;

nodes.clear();

fin.open("hashtable.txt", ifstream::in);

// Execute a loop until EOF (End of File)

while (true) {

// Read a Line from File

getline(fin, line);

string delimiter = ":";

size\_t pos = line.find(delimiter);

string s1 = line.substr(0, pos);

line.erase(0, pos + delimiter.length());

string s2 = line;

// Print values into runtime hash table

cout << s1 << ":" << s2 << endl;

hashCell temp(stoi(s1), s2);

nodes.push\_back(temp);

if (fin.eof()) {

break;

}

}

// Close the file

fin.close();

}

void writeFile() {

ofstream fout;

string line;

fout.open("hashtable.txt", ofstream::out);

// Execute a loop until EOF (End of File)

if (fout.is\_open()) {

for (unsigned int i = 0; i < nodes.size() - 1; i++) {

//cout << line << endl; debugging

line = to\_string(nodes.at(i).hash) + ":" + nodes.at(i).value;

// Print line in file

fout << line << endl;

cout << line << endl;

}

line = to\_string(nodes.at(nodes.size() - 1).hash) + ":" + nodes.at(nodes.size() - 1).value;

fout << line;

cout << line << endl;

}

// Close the file

fout.close();

}

void clearTable() {

nodes.clear();

}

void printTable() {

int j = nodes.size();

for (int i = 0; i < j; i++) {

string line = "";

//cout << line << endl; debugging

//cout << nodes.at(i).hash << endl; debugging

//cout << nodes.at(i).value << endl; debugging

line = to\_string(nodes.at(i).hash) + ":" + nodes.at(i).value;

cout << line << endl;

}

}

};

int main()

{

hashTable table;

table.setSieve(128); //initialize the prime number table used for hashing function

cout << "inserting 3 data elements..." << endl;

table.insert("apple", "banana");

table.insert("cherry", "lime");

table.insert("potato", "carrot");

cout << "writing to file..." << endl;

table.writeFile();

cout << "clearing table to test that read/write functions work..." << endl;

table.clearTable();

cout << "reading from file..." << endl;

table.readFile();

cout << "printing hash table..." << endl;

table.printTable();

cout << "retrieving first element by key..." << endl;

cout << table.retrieve("apple") << endl;

cout << "deleting first element by key..." << endl;

table.deleteCell("apple");

cout << "retrieving first element by key..." << endl;

cout << table.retrieve("apple") << endl;

system("pause");

return 0;

}