main.cpp

// Ryan Jacoby

#include "StudentHT.h"

#include<iostream>

#include<fstream>

int main() {

std::cout << "================================= Searching =================================\n";

std::ifstream fin;

std::string line;

StudentHT table(100);

table.startStatistics();

fin.open("insert.txt");

while(fin.good() && getline(fin, line))

table.insert(newRecType(stoi(line), "", "")); // Read `insert.txt` into hash table.

fin.close();

fin.open("search.txt");

while(fin.good() && getline(fin, line)) {

int id = table.search(newRecType(stoi(line), "", "")).studentId;

// Search for each item

// in `search.txt`

if(id > 0) std::cout << "The Student Record found is:\nStudent ID: " << id << '\n';

else std::cout << "Record not found for Student ID: " << line << '\n';

}

fin.close();

table.endStatistics();

table.displayStatistics(); // Calculate and show statistics

return 0;

}

StudentHT.h

// Ryan Jacoby

#include<string>

#ifndef StudentHT\_h

#define StudentHT\_h

struct RecType {

int studentId;

std::string firstName;

std::string lastName;

};

struct NodeType {

int studentId;

std::string firstName;

std::string lastName;

NodeType \* nextNode;

};

NodeType newNodeType(int, std::string, std::string, NodeType \*);

RecType newRecType(int, std::string, std::string);

class StudentHT {

private:

int HTSize;

int nodeSearches;

int IDSearched;

bool trackingStats;

NodeType \*\* hashPtr;

public:

StudentHT(int);

~StudentHT();

void insert(RecType);

RecType search(RecType);

void startStatistics();

void endStatistics();

void displayStatistics();

};

#endif

StudentHT.cpp

// Ryan Jacoby

#include "StudentHT.h"

#include <iostream>

/\*

\* Hash function used by hash table. Seperate function to be reimplemented.

\*

\* @param RecType to hash

\* @param Integer size of hash table

\*/

int hash(RecType rt, int n) {

return rt.studentId % n;

}

/\*

\* New node type helper function to create and assign the variables of NodeType struct.

\*

\* @param Integer student ID

\* @param String first name

\* @param String last name

\* @param Pointer to next node

\*/

NodeType newNodeType(int studentId, std::string firstName, std::string lastName, NodeType \* nextNode) {

NodeType ret;

ret.studentId = studentId;

ret.firstName = firstName;

ret.lastName = lastName;

ret.nextNode = nextNode;

return ret;

}

/\*

\* New rec type helper function to create and assign the variables of RecType struct.

\*

\* @param Integer student ID

\* @param String first name

\* @param String last name

\*/

RecType newRecType(int studentId, std::string firstName, std::string lastName) {

RecType ret;

ret.studentId = studentId;

ret.firstName = firstName;

ret.lastName = lastName;

return ret;

}

/\*

\* StudentHT constructor. Creates hash table with size n.

\*

\* @param Integer n size

\*/

StudentHT::StudentHT(int n) {

this->HTSize = n;

this->hashPtr = new NodeType \* [n];

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

this->hashPtr[i] = nullptr;

}

this->trackingStats = false;

}

/\*

\* StudentHT destructor; needs to be implemented as we dynamically

\* allocate memory.

\*/

StudentHT::~StudentHT() {

return;

}

/\*

\* Insert new student into hash table

\*

\* @param RecType entry to add

\*/

void StudentHT::insert(RecType rt) {

int pos = hash(rt, this->HTSize);

if(this->hashPtr[pos] == nullptr) { // If there's nothing in this spot of the hash table, put something there.

this->hashPtr[pos] = new NodeType;

\*this->hashPtr[pos] = newNodeType(rt.studentId, rt.firstName, rt.lastName, nullptr);

return;

}

NodeType \* current = this->hashPtr[pos];

while(current->nextNode != nullptr) current = current→nextNode;

// If there is, link onto the end.

current->nextNode = new NodeType;

\*current->nextNode = newNodeType(rt.studentId, rt.firstName, rt.lastName, nullptr);

}

/\*

\* Search hash table for node.

\*

\* @param RecType entry to search for.

\*

\* @return RecType found. StudentID of RecType returned will be -1 if nothing is found.

\*/

RecType StudentHT::search(RecType rt) {

if(this->trackingStats) this->IDSearched++;

int pos = hash(rt, this->HTSize);

NodeType \* current = this->hashPtr[pos];

while(current != nullptr) { // Search until we find the ID at the known hash position.

if(this->trackingStats) this->nodeSearches++;

if(current->studentId == rt.studentId) return newRecType(current->studentId, current->firstName, current->lastName);

current = current->nextNode;

}

return newRecType(-1, "", ""); // If we don't find, return a RecType with ID of -1

}

/\*

\* Start tracking statistics; reset statistics to zero.

\*/

void StudentHT::startStatistics() {

this->nodeSearches = 0;

this->IDSearched = 0;

this->trackingStats = true;

}

/\*

\* End statistics by setting tracking flag to false;

\*/

void StudentHT::endStatistics() {

this->trackingStats = false;

}

/\*

\* Calculate and print final statistics.

\*/

void StudentHT::displayStatistics() {

std::cout << "=============================================================================\n";

std::cout << "Total number of IDs searched : " << this->IDSearched << '\n';

std::cout << "Total number of nodes searched : " << this->nodeSearches << '\n';

std::cout << "Average number of nodes searched per ID searched : " << ((double)this->nodeSearches) / this->IDSearched << '\n';

std::cout << "=============================================================================\n";

}



