#include <fstream> //file input / output

#include <iostream> //console in/output

#include <string> //use string datatype

using namespace std;

//struct for each student that connects their first/last names and ID

struct Record

{

int studentID;

string firstName;

string lastName;

};

//class definition for the hashTable containing hashTable data variables

//and also the functions that create/manipulate the data in the hashTable

class hashTable

{

private:

int getSize(int);

public:

//holds data for each student in the table

struct NodeType

{

int studentID;

string firstName;

string lastName;

NodeType \* next;

};

//variables/data

int idSearches;

int nodeSearches;

double avgSearches;

int TABLE\_SIZE;

NodeType\*\* hashPtr;

//constructor/destructor

hashTable(int);

~hashTable();

//functions

void insert(Record);

Record search(Record);

void getAverage();

void displayStatistics();

}; //class hashTable

//return the table size

int hashTable::getSize(int idNum) { return idNum%TABLE\_SIZE; }

//constructor for the hashTable

hashTable::hashTable(int SIZE)

{

//statistics

idSearches = 0;

nodeSearches = 0;

avgSearches = 0;

TABLE\_SIZE = SIZE; //set size

hashPtr = new NodeType\*[SIZE]; //create table

for (int i = 0; i < SIZE; i++) hashPtr[i] = nullptr; //initialize

}

//destructor to release memory used by the hashTable

hashTable::~hashTable()

{

for (int i = 0; i < TABLE\_SIZE; ++i)

{

NodeType\* entry = hashPtr[i];

while (entry != NULL)

{

NodeType\* prev = entry;

entry = entry->next;

delete prev;

}

}

delete[] hashPtr;

}

//insert a new node (student) into the table

void hashTable::insert(Record item)

{

//creat the key used for inserting the new entry

int bucket = getSize(item.studentID);

//put the necessary data into the new node

struct NodeType \* newItem = new struct NodeType;

newItem->studentID = item.studentID;

newItem->firstName = item.firstName;

newItem->lastName = item.lastName;

newItem->next = nullptr;

//enter data if first location in the hashtable is empty

if (hashPtr[bucket] == nullptr) hashPtr[bucket] = newItem;

//seek until index is at the data location

else

{

struct NodeType \*temp = hashPtr[bucket];

while (temp->next != NULL) temp = temp->next;

temp->next = newItem;

}

}

//go through the hashTable in a certain order

//once the item of type record is found,

//return that item (return type record).

//if not found, return a value of -1

Record hashTable::search(Record item)

{

//increment the number of id's searched

idSearches++;

//create the key for searching

int bucket = getSize(item.studentID);

//create a new record to be returned

Record \*target = new Record;

//check if the location is empty

if (hashPtr[bucket] != nullptr)

{

struct NodeType \*temp = hashPtr[bucket];

while (temp !=nullptr)

{

nodeSearches++;

//if the node is found, return it (target)

if (temp->studentID == item.studentID)

{

target->studentID = temp->studentID;

target->firstName = temp->firstName;

target->lastName = temp->lastName;

return \*target;

}

temp =temp->next;

}//while loop

}//if

target->studentID =-1;

return \*target;

}

//calculate the average number of nodes searched per ID

void hashTable::getAverage() { avgSearches = (double)nodeSearches/idSearches; }

//display the end results

void hashTable::displayStatistics()

{

cout << "Total Number Of ID'S Searched: " << idSearches << endl;

cout << "Total Number Of Nodes Searched: " << nodeSearches << endl;

cout << "Average Of Nodes Searched Per ID: " << avgSearches << endl;

}

int main()

{

string addFile = "add.txt";

string searchFile = "search.txt";

int i;

//create the hash table

hashTable hTable(100);

fstream inputFile;

inputFile.open(addFile.c\_str(), ios::in);

if (inputFile.is\_open())

{

while(!inputFile.eof())

{

Record alpha;

inputFile >> alpha.studentID;

alpha.firstName = "Mark";

alpha.lastName = "Samples";

hTable.insert(alpha);

}

inputFile.close();

}

else cout << "File 'Add' cannot be opened.";

inputFile.open(searchFile.c\_str(), ios::in);

if (inputFile.is\_open())

{

while(!inputFile.eof())

{

Record beta;

inputFile >> beta.studentID;

beta.firstName = "Mark";

beta.lastName = "Samples";

hTable.search(beta);

}

inputFile.close();

}

else cout << "File 'Search' cannot be opened.";

hTable.getAverage();

hTable.displayStatistics();

}

