#ifndef HASHNODE\_H

#define HASHNODE\_H

#include<string>

using namespace std;

class hashNode

{

public:

hashNode();

hashNode(string fullName, string stuId, string stuEmail, string phoneNum);

virtual ~hashNode();

string getFullName(){ return fullName;}

string getStudentID(){ return stuId; }

string getStudentEmail(){ return stuEmail;}

string getPhoneNumber(){ return phoneNum;}

protected:

private:

// key: fullName, value: stuId, stuEmail, phoneNum

string fullName;

string stuId;

string stuEmail;

string phoneNum;

};

#endif // HASHNODE\_H

#include "hashNode.h"

hashNode::hashNode()

{

//ctor

}

hashNode::hashNode(string fullName, string stuId, string stuEmail, string phoneNum)

{

this->fullName = fullName;

this->stuId = stuId;

this->stuEmail = stuEmail;

this->phoneNum = phoneNum;

}

hashNode::~hashNode()

{

//dtor

}

#ifndef HASHTABLE\_H

#define HASHTABLE\_H

#include "hashNode.h"

class hashTable

{

public:

hashTable(int sz);

virtual ~hashTable();

void insertNode(hashNode \*p);

bool deleteNode(string fullName);

hashNode\* searchNode(string fullName);

void display();

protected:

private:

hashNode \*\*table; // dung de quan ly mang con tro hashNode

int TABLE\_SIZE; // size max of table

int curr\_size = 0; // current size of table;

int hashFunc(string fullName);

int quadraticProbe(int hashValue, int i);// hàm xử lý xung đột bậc 2

};

#endif // HASHTABLE\_H

#include "hashTable.h"

#include<iostream>

#include<iomanip>

using namespace std;

hashTable::hashTable(int sz) : TABLE\_SIZE(sz), curr\_size(0)

{

//ctor

table = new hashNode \* [TABLE\_SIZE];

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

{

table[i] = nullptr;

}

}

hashTable::~hashTable()

{

//dtor

}

int hashTable::hashFunc(string fullName)

{

int sum = 0;

for(int i=0; i < fullName.size(); i++)

sum += (int)tolower(fullName[i]);

return sum % TABLE\_SIZE;

}

// hàm xử lý xung đột bậc 2

int hashTable::quadraticProbe(int hashValue, int i)

{

return (hashValue + i \* i) % TABLE\_SIZE;

}

void hashTable::insertNode(hashNode \*p)

{

// check curr size đạt tới size table chưa??

if(curr\_size == TABLE\_SIZE)

return;

int hashIndex = hashFunc(p->getFullName());

if(table[hashIndex] == nullptr || table[hashIndex]->getFullName().empty() || table[hashIndex]->getFullName() == p->getFullName())

table[hashIndex] = p;

else

{

int i = 1;

int curr\_index = quadraticProbe(hashIndex, i);

while(curr\_index != hashIndex)

{

if(table[curr\_index] == nullptr || table[curr\_index]->getFullName().empty() || table[curr\_index]->getFullName() == p->getFullName())

{

table[curr\_index] = p;

break;

}

++i;

curr\_index = quadraticProbe(hashIndex, i);

}

}

++curr\_size;

}

bool hashTable::deleteNode(string fullName)

{

int hashIndex = hashFunc(fullName);

// check phần tử của hashIndex có nullptr ko?; vì nếu nullptr thì ko truy cập đến getFullName() đc

if(table[hashIndex] == nullptr) return false;

// xoá phần tử tại hashIndex

else if(table[hashIndex]->getFullName().compare(fullName) == 0)

{

delete table[hashIndex];

table[hashIndex] = nullptr; // khi delete(giai phong) cần cập nhập lại con trỏ bằng null

// vì ko cập nhập né sẽ trỏ đến 1 địa chỉ vùng nhớ không xác định

--curr\_size;

return true;

}

// xóa phần tử khi bị xung đột tại index

int i = 1;

int curr\_index = quadraticProbe(hashIndex, i);

// curr\_index phải khác index && ptu tại đó ko đc NULL

// khác index dành cho TH cả bảng có chung 1 index ban đầu

while(curr\_index != hashIndex && table[curr\_index] != nullptr)

{

if(table[curr\_index]->getFullName().compare(fullName) == 0)

{

delete table[curr\_index];

table[curr\_index] = nullptr; // khi delete(giai phong) cần cập nhập lại con trỏ bằng null

// vì ko cập nhập nó sẽ trỏ đến 1 địa chỉ vùng nhớ không xác định

--curr\_size;

return true;

}

++i;

curr\_index = quadraticProbe(hashIndex, i);

}

return false;

}

hashNode\* hashTable::searchNode(string fullName)

{

int hashIndex = hashFunc(fullName);

// check phần tử của hashIndex có nullptr ko?; vì nếu nullptr thì ko truy cập đến getFullName() đc

if(table[hashIndex] == nullptr) return nullptr;

else if(table[hashIndex]->getFullName().compare(fullName) == 0)

return table[hashIndex];

// find phần tử khi bị xung đột tại hashIndex

int i = 1;

int curr\_index = quadraticProbe(hashIndex, i);

// curr\_index phải khác index && ptu tại đó ko đc NULL

// khác index dành cho TH cả bảng có chung 1 index ban đầu

while(curr\_index != hashIndex && table[curr\_index] != nullptr)

{

if(table[curr\_index]->getFullName().compare(fullName) == 0)

{

return table[curr\_index];

}

// chia phần trăm đề khi duyệt tới cuối thì chia dư i sẽ về 0, cách này ok hơn là tăng i

++i;

curr\_index = quadraticProbe(hashIndex, i);

}

return nullptr;

}

void hashTable::display()

{

// Traverse the entire hash table

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

// cout << " +--------+--------+" << endl;

cout << i << " |";

// hashNode\* p = table[i];

if (table[i] == nullptr ) {

// NULL record, print empty

cout << "[" << setw(6) << "" << " ]";// | " << setw(6) << "" << " |";

}

if(table[i] != nullptr) {

// Print the record from the table

cout << "[ " << table[i]->getFullName() << " | " << table[i]->getStudentEmail() << ", " << table[i]->getPhoneNumber() << ", " << table[i]->getStudentID() << " ]";

//

// // Traverse and print the chain

// for (p = p->getNext(); p != nullptr ; p = p->getNext()) {

// cout << " --> " << "[ " << p->getFullName() << " | " << p->getStudentEmail() << ", " << p->getPhoneNumber() << ", " << p->getStudentID() << " ]";

// }

}

cout << endl;

}

cout << endl;

}

#include <iostream>

#include"hashNode.h"

#include"hashTable.h"

using namespace std;

int main()

{

hashTable \*st = new hashTable(17);

st->insertNode(new hashNode("Lan", "001", "lan@gmail.com", "0922222222"));

st->insertNode(new hashNode("Phong", "002", "phong@gmail.com", "093333333"));

st->insertNode(new hashNode("Thuy", "003", "thuy@gmail.com", "098273732"));

st->insertNode(new hashNode("Ha", "004", "ha@gmail.com", "0944444444"));

st->insertNode(new hashNode("Anh", "005", "anh@gmail.com", "0955555555"));

st->insertNode(new hashNode("Ban", "006", "ban@gmail.com", "0966666666"));

st->insertNode(new hashNode("Aan", "007", "aan@gmail.com", "0966666666"));

st->insertNode(new hashNode("Cuong", "008", "cuong@gmail.com", "097777777"));

st->insertNode(new hashNode("Khag", "009", "khag@gmail.com", "098888888"));

st->insertNode(new hashNode("Yem", "010", "yem@gmail.com", "09999999999"));

st->display();

st->deleteNode("Yem");

st->display();

cout << ((st->searchNode("Hund")) ? "YES" : "NO ");

return 0;

}