#pragma once

#ifndef HASH\_TABLE\_H

#define HASH\_TABLE\_H

#include <iostream>

#include <fstream>

#include <list>

#include <vector>

using namespace std;

struct patient {

int card\_number;

int illness\_code;

string doctor\_surname;

int index = 0;

~patient() {

card\_number = 0;

illness\_code = 0;

doctor\_surname = "";

index = 0;

}

};

class hash\_table {

private:

int current\_size = 0;

int real\_size = 8;

int number\_of\_line = 0;

vector<list<patient>> table;

string file\_path = "";

int hash\_func(int card\_number) {

return card\_number % real\_size;

}

public:

hash\_table(int n)

{

if (n > 8) real\_size = n;

table.resize(real\_size);

}

int get\_current\_size() {

return this->current\_size;

}

bool add(patient tmp, int index)

{

int hash = this->hash\_func(tmp.card\_number);

for (auto iter = table[hash].begin(); iter != table[hash].end(); iter++) {

if (iter->card\_number == tmp.card\_number) {

return false;

}

}

tmp.index = index;

this->table[hash].push\_front(tmp);

current\_size++;

if (float(current\_size) / float(real\_size) >= 0.75) resize();

return true;

}

patient\* get\_patient(int card\_number)

{

int n = hash\_func(card\_number);

patient\* tmp = new patient();

tmp->card\_number = -1;

for (auto i = table[n].begin(); i != table[n].end(); i++)

{

tmp->card\_number = (\*i).card\_number;

tmp->illness\_code = (\*i).illness\_code;

tmp->doctor\_surname = (\*i).doctor\_surname;

tmp->index = (\*i).index;

if (tmp->card\_number == card\_number) return tmp;

else {

tmp->card\_number = -1;

}

}

return tmp;

}

void display\_field(int card\_number)

{

patient\* tmp = get\_patient(card\_number);

if (tmp->card\_number == -1) return;

cout << card\_number << " " << tmp->illness\_code << " " << tmp->doctor\_surname << "\n";

}

void display\_all\_file(string path\_out)

{

ofstream fout(path\_out, ios::binary | ios::out);

bool is\_empty = true;

for (int i = 0; i < real\_size; i++)

{

if (table[i].empty()) continue;

is\_empty = false;

for (auto iter = table[i].begin(); iter != table[i].end(); iter++)

{

patient tmp = \*(iter);

fout.write((char\*)&tmp, sizeof(patient));

}

}

fout.close();

if (is\_empty)

cout << "Table is empty\n";

}

void display\_all()

{

bool is\_empty = true;

for (int i = 0; i < real\_size; i++)

{

if (table[i].empty()) continue;

is\_empty = false;

for (auto iter = table[i].begin(); iter != table[i].end(); iter++)

{

patient tmp = \*(iter);

cout<<tmp.index << " " << tmp.card\_number << " " << tmp.illness\_code << " " << tmp.doctor\_surname << "\n";

}

}

if (is\_empty)

cout << "Table is empty\n";

}

void resize()

{

vector<list<patient>> tmp(table);

real\_size \*= 2;

table.clear();

table.resize(real\_size);

for (int i = 0; i < this->real\_size/2; i++) {

while (!tmp[i].empty()) {

table[hash\_func(tmp[i].front().card\_number)].push\_front(tmp[i].front());

tmp[i].pop\_front();

}

}

tmp.clear();

}

void delete\_field(int card\_number)

{

int n = hash\_func(card\_number);

for (auto i = table[n].begin(); i != table[n].end(); i++)

{

if (i->card\_number == card\_number)

{

table[n].erase(i);

return;

}

}

}

void clear()

{

table.clear();

real\_size = 8;

current\_size = 0;

table.resize(real\_size);

}

bool isEmpty() {

return !current\_size;

}

string get\_file\_path() {

return file\_path;

}

void set\_file\_path(string path) {

this->file\_path = path;

}

~hash\_table()

{

delete[] &table;

}

};

#endif