```
#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
```