

**LAB2.h**

#pragma once

#include <iostream>

#include <string>

using namespace std;

//Населённый пункт университета

extern string university\_location;

//Перечисление критериев поиска

const int category\_amount = 5;

enum SearchCategory {

CAT\_ALL,

CAT\_PERFECT\_EXAM,

CAT\_ATTEST5,

CAT\_OUTSIDE\_LOCATION,

CAT\_NEED\_DORMITORY

};

//Абитуриент

struct Abitur {

string surname;

unsigned short mark[3];

bool attest5;

string location;

bool need\_dormitory;

operator string();

};

struct Node;

struct Descriptor {

Node\* first = nullptr;

Node\* last = nullptr;

int size = 0;

};

class List {

private:

Descriptor desc[category\_amount];

public:

string read\_all(SearchCategory cat);

void clear();

bool is\_empty();

int get\_size(SearchCategory cat);

bool insert(Abitur abitur);

bool remove(int pos);

void get\_all(SearchCategory cat, Abitur\* result\_array);

~List();

};

bool match(SearchCategory cat, Abitur& Abitur);

**module.cpp**

#include "LAB2.h"

struct Node {

Abitur value;

Node\* next[category\_amount];

Node() {

for (int i = 0; i < 5; i++) next[i] = nullptr;

}

};

Abitur::operator string() {

string result = surname + ": ";

result += "\n Экзамены: " + to\_string(mark[0]) + " " + to\_string(mark[1]) + " " + to\_string(mark[2]);

result += "\n Аттестат с отличием: ";

result += attest5 ? "да" : "нет";

if (university\_location == location)

result += "\n В пределах населённого пункта университета (" + location + ")";

else

result += "\n За пределами населённого пункта университета (" + location + ")";

result += "\n Нуждается в общежитии: ";

result += need\_dormitory ? "да" : "нет";

return result + "\n";

}

string List::read\_all(SearchCategory cat) {

int sz = desc[(int)cat].size;

string result = "Список (size = " + to\_string(sz) + "):\n";

Abitur\* Abiturs = new Abitur[sz];

get\_all(cat, Abiturs);

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

result += to\_string(i + 1) + ". " + string(Abiturs[i]);

}

delete[] Abiturs;

return result;

}

void List::clear() {

Node\* current = desc[0].first;

while (current) {

Node\* next = current->next[0];

delete current;

current = next;

}

for (int i = 0; i < category\_amount; i++) {

desc[i].first = nullptr;

desc[i].last = nullptr;

desc[i].size = 0;

}

}

bool List::is\_empty() {

return desc[0].first == nullptr;

}

int List::get\_size(SearchCategory cat) {

return desc[(int)cat].size;

}

bool List::insert(Abitur abitur) {

//Создаём узел с добавляемым содержимым

Node\* node = new Node();

node->value = abitur;

//Берём предшествующий узел из каждой категории и привязываем созданный узел

for (int i = 0; i < category\_amount; i++) {

if (!match((SearchCategory)i, abitur))

continue;

//Нет предшествующего узла для данной категории. Узел - начало списка.

if (desc[i].last == nullptr) {

desc[i].first = node;

desc[i].last = node;

}

//Есть предшествующий узел

else {

desc[i].last->next[i] = node;

desc[i].last = node;

}

desc[i].size++;

}

return true;

}

bool List::remove(int pos) {

if (pos < 0 || get\_size(CAT\_ALL) <= pos) return false;

if (is\_empty()) return false;

//Проход по главному списку

Node\* prev0 = nullptr;

Node\* node = desc[0].first;

if (pos > 0) {

for (int i = 0; i < pos - 1; i++)

node = node->next[0];

prev0 = node;

node = node->next[0];

}

//Попадает ли в список

bool matches[category\_amount];

for (int i = 0; i < category\_amount; i++) {

matches[i] = match((SearchCategory)i, node->value);

}

//Удаляем узел повторным перебором

for (int i = 0; i < category\_amount; i++) {

if (matches[i] == false) continue;

//Узел - начало списка

if (node == desc[i].first) {

desc[i].first = node->next[i];

//Узел - и начало и конец

if (node == desc[i].last) {

desc[i].last = nullptr;

}

desc[i].size--;

continue;

}

//Берём предыдущий узел

Node\* prev\_node\_in\_cat = desc[i].first;

while (prev\_node\_in\_cat->next[i] != node) {

prev\_node\_in\_cat = prev\_node\_in\_cat->next[i];

}

//Узел - конец списка.

if (node == desc[i].last) {

desc[i].last = prev\_node\_in\_cat;

}

prev\_node\_in\_cat->next[i] = node->next[i];

desc[i].size--;

}

delete node;

return true;

}

void List::get\_all(SearchCategory cat, Abitur\* result\_array) {

int cat\_number = (int)cat;

Node\* current = desc[cat\_number].first;

int i = 0;

while (current) {

result\_array[i++] = current->value;

current = current->next[cat\_number];

}

}

List::~List() { clear(); }

bool match(SearchCategory cat, Abitur& Abitur) {

switch (cat) {

case CAT\_ALL:

return true;

case CAT\_PERFECT\_EXAM:

return Abitur.mark[0] == 5 && Abitur.mark[1] == 5 && Abitur.mark[2] == 5;

case CAT\_ATTEST5:

return Abitur.attest5;

case CAT\_OUTSIDE\_LOCATION:

return Abitur.location != university\_location;

case CAT\_NEED\_DORMITORY:

return Abitur.need\_dormitory;

}

return false;

}

**AISD\_LAB2.cpp**

#include "LAB2.h"

#include "Windows.h"

#include <chrono>

#include <random>

typedef chrono::steady\_clock my\_clock;

my\_clock::time\_point start;

my\_clock::time\_point finish;

string university\_location;

long long timer = 0;

//unsigned long long test\_count = 10000;

//float p = 100;

//int to = 100;

Abitur input() {

Abitur elem;

cout << "Фамилия >> ";

cin >> elem.surname;

cout << "Оценки за три вступительных экзамена >> ";

cin >> elem.mark[0] >> elem.mark[1] >> elem.mark[2];

while (true) {

string answer;

cout << "Имеется аттестат с отличием? (да/нет) >> ";

cin >> answer;

if (answer == "да") elem.attest5 = true;

else if (answer == "нет") elem.attest5 = false;

else {

cout << "Ответ не распознан." << endl;

continue;

}

break;

}

cout << "Населённый пункт >> ";

cin >> elem.location;

while (true) {

string answer;

cout << "Нуждается в общежитии? (да/нет) >> ";

cin >> answer;

if (answer == "да") elem.need\_dormitory = true;

else if (answer == "нет") elem.need\_dormitory = false;

else {

cout << "Ответ не распознан." << endl;

continue;

}

break;

}

return elem;

}

//int get\_rand(int from, int to) {

// random\_device dev;

// mt19937 rng(dev());

// uniform\_int\_distribution<mt19937::result\_type> r(from, to);

// return r(rng);

//}

int main() {

setlocale(LC\_ALL, "ru");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

//university\_location = "УНИВЕРСИТЕТ";

cout << "Населённый пункт университета >> ";

cin >> university\_location;

List list;

//cout << "Вероятность попадания для подсписка: " << (float)p / to << endl;

//ВСТАВКА

//for (int i = 0; i < 20; i++) cout << ".";

//cout << endl;

//for (int i = 1; i <= test\_count; i++) {

// if(test\_count / 20 != 0)

// if (i % (test\_count / 20) == 0) cout << "|";

// Abitur abit;

// abit.surname = to\_string(i) + "-ФАМИЛИЯ";

// abit.attest5 = get\_rand(1, to) < p;

// abit.location = (get\_rand(1, to) < p) ? "где-то не здесь " + to\_string(i) : university\_location;

// abit.mark[0] = 5; abit.mark[1] = 4 + (get\_rand(1, to) < p); abit.mark[2] = 5;

// abit.need\_dormitory = get\_rand(1, to) < p;

// start = my\_clock::now();

// list.insert(abit);

// finish = my\_clock::now();

// timer += chrono::duration\_cast<chrono::nanoseconds>(finish - start).count();

//}

//cout << timer \* 0.000001 << "мс" << endl;

//cout << (timer \* 0.000001) / test\_count << "мс - среднее время на одну вставку" << endl;

//

////УДАЛЕНИЕ

//timer = 0;

//for (int i = 0; i < 20; i++) cout << ".";

//cout << endl;

//for (int i = 1; i <= test\_count; i++) {

// if (test\_count / 20 != 0)

// if (i % (test\_count / 20) == 0) cout << "|";

// int pos = get\_rand(0, list.get\_size(CAT\_ALL)-1);

// start = my\_clock::now();

// list.remove(pos);

// finish = my\_clock::now();

// timer += chrono::duration\_cast<chrono::nanoseconds>(finish - start).count();

//}

//cout << timer \* 0.000001 << "мс" << endl;

//cout << (timer \* 0.000001) / test\_count << "мс - среднее время на одно удаление" << endl;

//system("pause");

int cmd;

while (true) {

cout << "\033[0m";//Сделать текст стандартным

cout <<

"\

Список команд:\n\

1. Ввод записи об абитуриенте.\n\

2. Формирование и вывод списков абитуриентов по критериям:\n\

1) Все экзамены сданы на «отлично»;\n\

2) Имеется аттестат с отличием;\n\

3) Проживает за пределами населенного пункта, в котором расположен университет;\n\

4) Нуждается в общежитии.\n\

3. Формирование и вывод полного списка абитуриентов.\n\

4. Удаление записи об абитуриенте.\n\

5. Удаление всех записей многосвязного списка.\n\

0. Выход из программы.\n\

";

cout << "Введите номер команды >> ";

cin >> cmd;

cout << "\033[1m";//Сделать текст жирным

cout << "\n";

string answer;

if (cmd == 1) {

if (list.insert(input()))

answer = "Студент добавлен.";

else

answer = "Студент не добавлен. Такой позиции ";

}

else if (cmd == 2) {

int cat;

cout << "Критерий >> ";

cin >> cat;

if (1 <= cat && cat <= 4)

answer = list.read\_all((SearchCategory)cat);

else

answer = "Нет такого номера.";

}

else if (cmd == 3) {

answer = list.read\_all(CAT\_ALL);

}

else if (cmd == 4) {

if (list.is\_empty()) {

answer = "Список пуст.";

}

else {

int total\_size = list.get\_size(CAT\_ALL);

Abitur\* Abiturs = new Abitur[total\_size];

list.get\_all(CAT\_ALL, Abiturs);

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

cout << to\_string(i + 1) + ". " + Abiturs[i].surname + ".\n";

}

delete[] Abiturs;

int num;

cout << "Номер студента >> ";

cin >> num;

if (num < 0 || num > total\_size)

answer = "Указан неверный номер.";

else {

list.remove(num - 1);

answer = "Студент удалён.";

}

}

}

else if (cmd == 5) {

list.clear();

answer = "Список очищен.";

}

else if (cmd == 0) {

list.clear();

cout << "Программа завершена." << endl;

exit(0);

}

else

answer = "Нет команды с таким номером";

cout << answer << endl;

}

}