Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа № 9

По дисциплине «Основы алгоритмизации и программирования»

На тему «**Двусвязные списки**»

Выполнил:

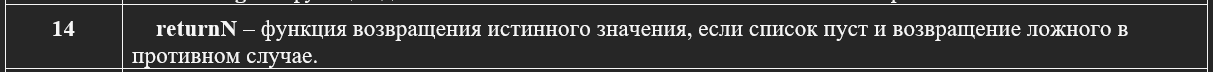
Федорович Вадим

Студент 1 курса 8 группы

Преподаватель: асс. Андронова М.В.

Минск, 2024

Основа:



#include <iostream>

#include <fstream>

using namespace std;

const unsigned int NAME\_SIZE = 30;

const unsigned int CITY\_SIZE = 20;

bool flag;

struct Address

{

char name[NAME\_SIZE];

char city[CITY\_SIZE];

Address\* next;

Address\* prev;

};

//-----------------------------------------------------------

int menu(void)

{

char s[80]; int c;

cout << endl;

cout << "1. Ввод имени" << endl;

cout << "2. Удаление имени" << endl;

cout << "3. Вывод на экран" << endl;

cout << "4. Поиск" << endl;

cout << "5. Проверка за пустоту" << endl;

cout << "0. Выход" << endl;

cout << endl;

do

{

cout << "Ваш выбор: ";

cin.sync();

gets\_s(s);

cout << endl;

c = atoi(s);

} while (c < 0 || c > 5);

return c;

}

//-----------------------------------------------------------

void insert(Address\* e, Address\*\* phead, Address\*\* plast) //Добавление в конец списка

{

Address\* p = \*plast;

if (\*plast == NULL)

{

e->next = NULL;

e->prev = NULL;

\*plast = e;

\*phead = e;

return;

}

else

{

p->next = e;

e->next = NULL;

e->prev = p;

\*plast = e;

}

}

//-----------------------------------------------------------

Address\* setElement() // Создание элемента и ввод его значений с клавиатуры

{

Address\* temp = new Address();

if (!temp)

{

cerr << "Ошибка выделения памяти памяти";

return NULL;

}

cout << "Введите имя: ";

cin.getline(temp->name, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.clear();

cout << "Введите город: ";

cin.getline(temp->city, CITY\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.clear();

temp->next = NULL;

temp->prev = NULL;

return temp;

}

//-----------------------------------------------------------

void outputList(Address\*\* phead, Address\*\* plast) //Вывод списка на экран

{

Address\* t = \*phead;

while (t)

{

cout << t->name << ' ' << t->city << endl;

t = t->next;

}

cout << "" << endl;

}

//-----------------------------------------------------------

void find(char name[NAME\_SIZE], Address\*\* phead) // Поиск имени в списке

{

Address\* t = \*phead;

while (t)

{

if (!strcmp(name, t->name)) break;

t = t->next;

}

if (!t)

cerr << "Имя не найдено" << endl;

else

cout << t->name << ' ' << t->city << endl;

}

//-----------------------------------------------------------

void delet(char name[NAME\_SIZE], Address\*\* phead, Address\*\* plast) // Удаление имени

{

struct Address\* t = \*phead;

while (t)

{

if (!strcmp(name, t->name)) break;

t = t->next;

}

if (!t)

cerr << "Имя не найдено" << endl;

else

{

if (\*phead == t)

{

\*phead = t->next;

if (\*phead)

(\*phead)->prev = NULL;

else

\*plast = NULL;

}

else

{

t->prev->next = t->next;

if (t != \*plast)

t->next->prev = t->prev;

else

\*plast = t->prev;

}

delete t;

cout << "Элемент удален" << endl;

}

}

int returnN(Address\*\*);

//-----------------------------------------------------------

int main(void)

{

Address\* head = NULL;

Address\* last = NULL;

setlocale(LC\_CTYPE, "Rus");

while (true)

{

switch (menu())

{

case 1: insert(setElement(), &head, &last);

break;

case 2: { char dname[NAME\_SIZE];

cout << "Введите имя: ";

cin.getline(dname, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.sync();

delet(dname, &head, &last);

}

break;

case 3: outputList(&head, &last);

break;

case 4: { char fname[NAME\_SIZE];

cout << "Введите имя: ";

cin.getline(fname, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.sync();

find(fname, &head);

}

break;

case 5: flag = returnN(&head);

if (flag) cout << "Список не пуст!";

else cout << "Список пуст...";

case 0: exit(0);

default: exit(1);

}

}

return 0;

}

//-------------------------------------------------------------

int returnN(Address\*\* phead) {

if (\*phead == NULL){

return 0;

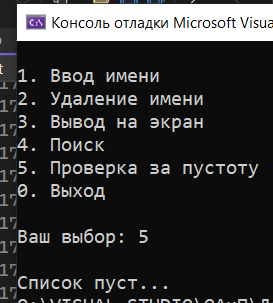
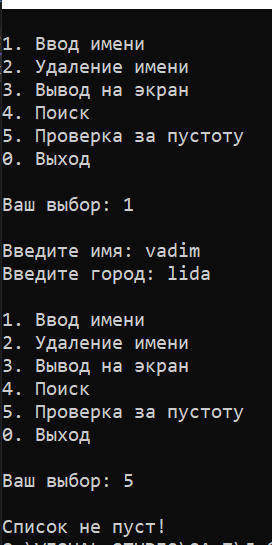
}

if (\*phead != NULL) {

return 1;

}

}

Допы:



#include <iostream>

#include <fstream>

using namespace std;

const unsigned int NAME\_SIZE = 30;

const unsigned int CITY\_SIZE = 20;

bool flag;

struct Address

{

char name[NAME\_SIZE];

char city[CITY\_SIZE];

Address\* next;

Address\* prev;

};

//-----------------------------------------------------------

int menu(void)

{

char s[80]; int c;

cout << endl;

cout << "1. Ввод имени" << endl;

cout << "2. Удаление имени" << endl;

cout << "3. Вывод на экран" << endl;

cout << "4. Поиск" << endl;

cout << "5. функция подсчёта числа элементов списка с заданным значением x." << endl;

cout << "0. Выход" << endl;

cout << endl;

do

{

cout << "Ваш выбор: ";

cin.sync();

gets\_s(s);

cout << endl;

c = atoi(s);

} while (c < 0 || c > 5);

return c;

}

//-----------------------------------------------------------

void insert(Address\* e, Address\*\* phead, Address\*\* plast) //Добавление в конец списка

{

Address\* p = \*plast;

if (\*plast == NULL)

{

e->next = NULL;

e->prev = NULL;

\*plast = e;

\*phead = e;

return;

}

else

{

p->next = e;

e->next = NULL;

e->prev = p;

\*plast = e;

}

}

//-----------------------------------------------------------

Address\* setElement() // Создание элемента и ввод его значений с клавиатуры

{

Address\* temp = new Address();

if (!temp)

{

cerr << "Ошибка выделения памяти памяти";

return NULL;

}

cout << "Введите имя: ";

cin.getline(temp->name, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.clear();

cout << "Введите город: ";

cin.getline(temp->city, CITY\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.clear();

temp->next = NULL;

temp->prev = NULL;

return temp;

}

//-----------------------------------------------------------

void outputList(Address\*\* phead, Address\*\* plast) //Вывод списка на экран

{

Address\* t = \*phead;

while (t)

{

cout << t->name << ' ' << t->city << endl;

t = t->next;

}

cout << "" << endl;

}

//-----------------------------------------------------------

void find(char name[NAME\_SIZE], Address\*\* phead) // Поиск имени в списке

{

Address\* t = \*phead;

while (t)

{

if (!strcmp(name, t->name)) break;

t = t->next;

}

if (!t)

cerr << "Имя не найдено" << endl;

else

cout << t->name << ' ' << t->city << endl;

}

//-----------------------------------------------------------

void delet(char name[NAME\_SIZE], Address\*\* phead, Address\*\* plast) // Удаление имени

{

struct Address\* t = \*phead;

while (t)

{

if (!strcmp(name, t->name)) break;

t = t->next;

}

if (!t)

cerr << "Имя не найдено" << endl;

else

{

if (\*phead == t)

{

\*phead = t->next;

if (\*phead)

(\*phead)->prev = NULL;

else

\*plast = NULL;

}

else

{

t->prev->next = t->next;

if (t != \*plast)

t->next->prev = t->prev;

else

\*plast = t->prev;

}

delete t;

cout << "Элемент удален" << endl;

}

}

int counter;

int countX(Address\*\* phead);

//-----------------------------------------------------------

int main(void)

{

Address\* head = NULL;

Address\* last = NULL;

setlocale(LC\_CTYPE, "Rus");

while (true)

{

switch (menu())

{

case 1: insert(setElement(), &head, &last);

break;

case 2: { char dname[NAME\_SIZE];

cout << "Введите имя: ";

cin.getline(dname, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.sync();

delet(dname, &head, &last);

}

break;

case 3: outputList(&head, &last);

break;

case 4: { char fname[NAME\_SIZE];

cout << "Введите имя: ";

cin.getline(fname, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.sync();

find(fname, &head);

}

break;

case 5: counter = countX(&head); cout <<"Элементов в списке с таким именем: " << counter;

case 0: exit(0);

default: exit(1);

}

}

return 0;

}

//-------------------------------------------------------------

int countX(Address\*\* phead) {// функция подсчёта числа элементов списка с заданным значением x

int sum = 0; char name[25];

struct Address\* t = \*phead;

if (\*phead == NULL){

cout << "Список пуст...";

return 0;

}

if (t != NULL) {

cout << "Введите имя, поиском которого функиця будет заниматься: ";

gets\_s(name);

}

while (t) {

if (!strcmp(name, t->name)){

sum++;

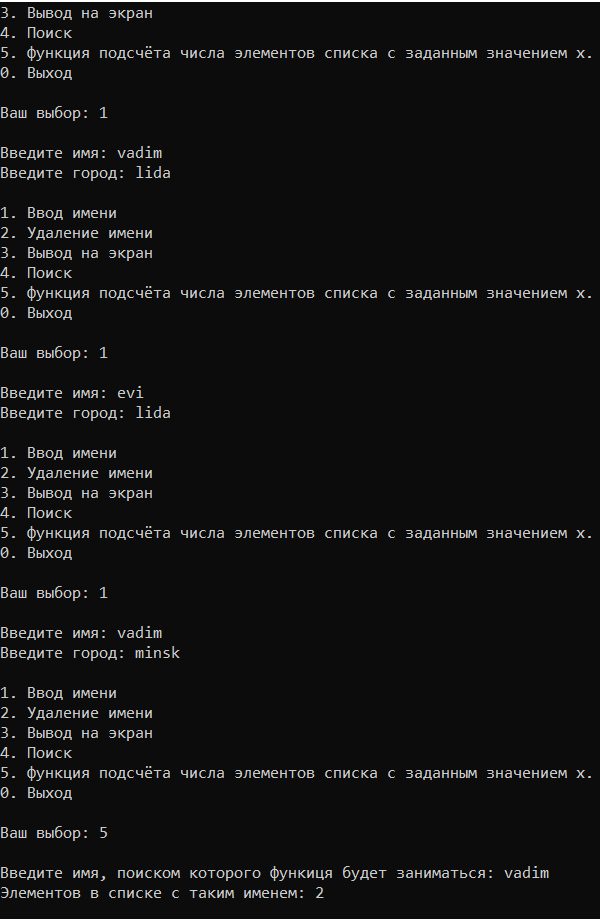
}

t = t->next;

}

return sum;

}





#include <iostream>

#include <fstream>

using namespace std;

const unsigned int NAME\_SIZE = 30;

const unsigned int CITY\_SIZE = 20;

bool flag;

struct Address

{

int num;

char name[NAME\_SIZE];

char city[CITY\_SIZE];

Address\* next;

Address\* prev;

};

//-----------------------------------------------------------

int menu(void)

{

char s[80]; int c;

cout << endl;

cout << "1. Ввод имени" << endl;

cout << "2. Удаление имени" << endl;

cout << "3. Вывод на экран" << endl;

cout << "4. Поиск" << endl;

cout << "5. функция подсчёта минимального элемента списка" << endl;

cout << "0. Выход" << endl;

cout << endl;

do

{

cout << "Ваш выбор: ";

cin.sync();

gets\_s(s);

cout << endl;

c = atoi(s);

} while (c < 0 || c > 5);

return c;

}

//-----------------------------------------------------------

void insert(Address\* e, Address\*\* phead, Address\*\* plast) //Добавление в конец списка

{

Address\* p = \*plast;

if (\*plast == NULL)

{

e->next = NULL;

e->prev = NULL;

\*plast = e;

\*phead = e;

return;

}

else

{

p->next = e;

e->next = NULL;

e->prev = p;

\*plast = e;

}

}

//-----------------------------------------------------------

Address\* setElement() // Создание элемента и ввод его значений с клавиатуры

{

Address\* temp = new Address();

int x;

if (!temp)

{

cerr << "Ошибка выделения памяти памяти";

return NULL;

}

cout << "Введите имя: ";

cin.getline(temp->name, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.clear();

cout << "Введите город: ";

cin.getline(temp->city, CITY\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.clear();

cout << "Введите число: ";

cin >> x;

temp->num = x;

cin.ignore(cin.rdbuf()->in\_avail());

cin.clear();

temp->next = NULL;

temp->prev = NULL;

return temp;

}

//-----------------------------------------------------------

void outputList(Address\*\* phead, Address\*\* plast) //Вывод списка на экран

{

Address\* t = \*phead;

while (t)

{

cout << t->name << ' ' << t->city << endl;

t = t->next;

}

cout << "" << endl;

}

//-----------------------------------------------------------

void find(char name[NAME\_SIZE], Address\*\* phead) // Поиск имени в списке

{

Address\* t = \*phead;

while (t)

{

if (!strcmp(name, t->name)) break;

t = t->next;

}

if (!t)

cerr << "Имя не найдено" << endl;

else

cout << t->name << ' ' << t->city << endl;

}

//-----------------------------------------------------------

void delet(char name[NAME\_SIZE], Address\*\* phead, Address\*\* plast) // Удаление имени

{

struct Address\* t = \*phead;

while (t)

{

if (!strcmp(name, t->name)) break;

t = t->next;

}

if (!t)

cerr << "Имя не найдено" << endl;

else

{

if (\*phead == t)

{

\*phead = t->next;

if (\*phead)

(\*phead)->prev = NULL;

else

\*plast = NULL;

}

else

{

t->prev->next = t->next;

if (t != \*plast)

t->next->prev = t->prev;

else

\*plast = t->prev;

}

delete t;

cout << "Элемент удален" << endl;

}

}

int counter;

int findMin(Address\*\* phead);

//-----------------------------------------------------------

int main(void)

{

Address\* head = NULL;

Address\* last = NULL;

setlocale(LC\_CTYPE, "Rus");

while (true)

{

switch (menu())

{

case 1: insert(setElement(), &head, &last);

break;

case 2: { char dname[NAME\_SIZE];

cout << "Введите имя: ";

cin.getline(dname, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.sync();

delet(dname, &head, &last);

}

break;

case 3: outputList(&head, &last);

break;

case 4: { char fname[NAME\_SIZE];

cout << "Введите имя: ";

cin.getline(fname, NAME\_SIZE - 1, '\n');

cin.ignore(cin.rdbuf()->in\_avail());

cin.sync();

find(fname, &head);

}

break;

case 5: counter = findMin(&head); cout <<"Минимальный элемент: " << counter;

case 0: exit(0);

default: exit(1);

}

}

return 0;

}

//-------------------------------------------------------------

int findMin(Address\*\* phead) {// функция подсчёта числа элементов списка с заданным значением x

int min;

struct Address\* t = \*phead;

if (\*phead == NULL){

cout << "Список пуст...";

return 0;

}

if (t != NULL) {

min = t->num;

while (t) {

if (t->num < min){

min = t->num;

}

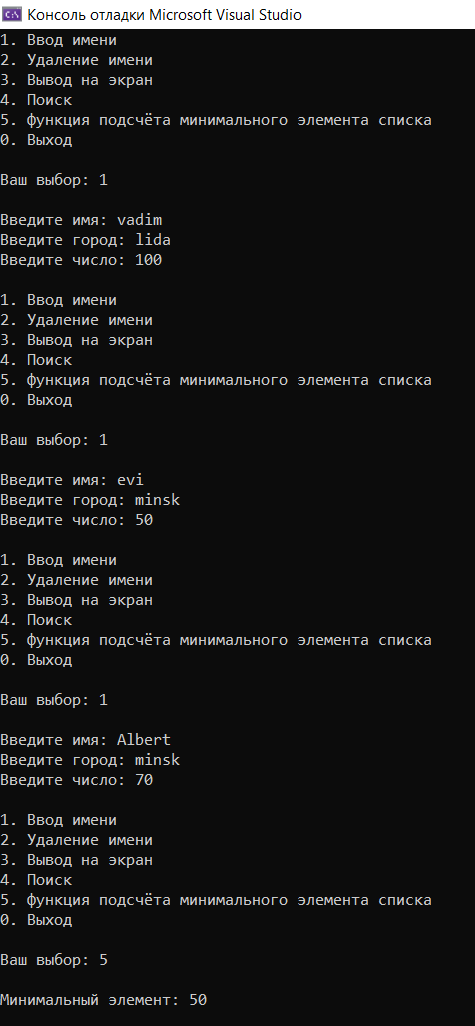
t = t->next;

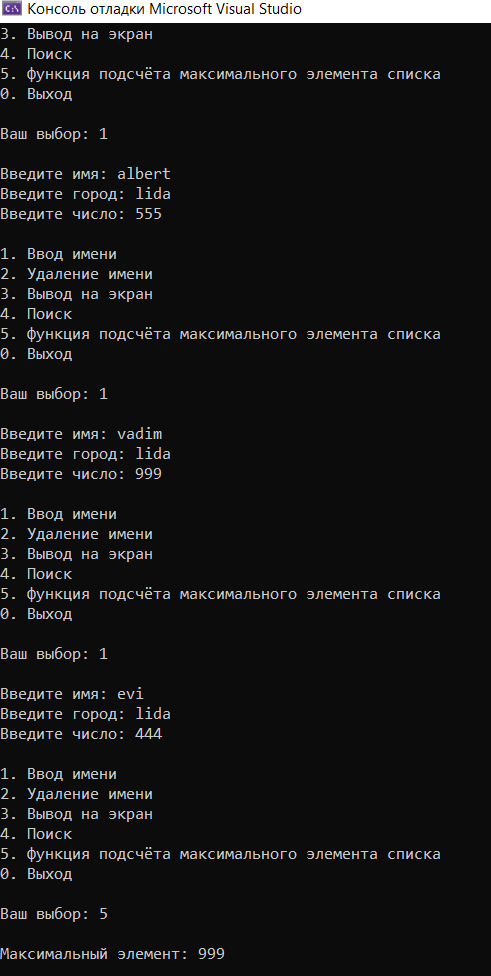
}

return min;

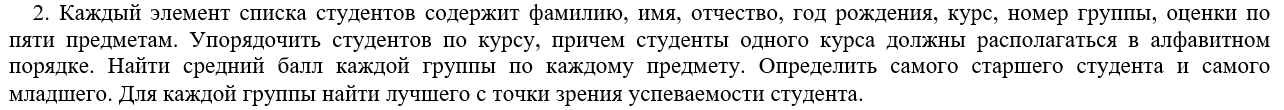
}

}





2



#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#define SIZE 50

using namespace std;

struct Student {

char surname[SIZE];

char name[SIZE];

char patronymic[SIZE];

int day;

int month;

int year;

int course;

int group;

float math;

float physics;

float english;

float OAIP;

float history;

Student\* next;

Student\* prev;

};

struct AverageScore {

float math;

float physics;

float english;

float OAIP;

float history;

};

void insert(Student\* e, Student\*\* phead, Student\*\* plast); //размещение

Student\* setElement(); //ввод

void outputList(Student\*\* phead, Student\*\* plast); //вывод

void studentOld(Student\*\* phead, Student\*\* plast); // Поиск старшего

void studentYoung(Student\*\* phead, Student\*\* plast); // Поиск младшего

void outputFirstCourse(Student\*\* phead, Student\*\* plast); //Первый курс

void outputSecondCourse(Student\*\* phead, Student\*\* plast); // Второй курс

void outputThirdCourse(Student\*\* phead, Student\*\* plast); //Третий курс

void outputFourthCourse(Student\*\* phead, Student\*\* plast); //Четвёртый курс

void sortStudentsByCourse(Student\*\* phead); // сортировка

void calculateAverageScores(Student\*\* phead); // расчёт среднего значения

int main() {

setlocale(LC\_ALL, "rus");

Student\* head = nullptr;

Student\* last = nullptr;

int choice;

do {

cout << "1. Ввод данных" << endl;

cout << "2. Поиск старшего студента" << endl;

cout << "3. Поиск младшего студента" << endl;

cout << "4. Отсортировать студентов" << endl;

cout << "5. Вывод всего" << endl;

cout << "6. Первый курс" << endl;

cout << "7. Второй курс" << endl;

cout << "8. Третий курс" << endl;

cout << "9. Четвёртый курс" << endl;

cout << "10. Расчёт среднего балла по группам" << endl;

cout << "11. Выход" << endl;

cin >> choice;

switch (choice)

{

case 1:

insert(setElement(), &head, &last);

break;

case 2:

studentOld(&head, &last);

break;

case 3:

studentYoung(&head, &last);

break;

case 4:

sortStudentsByCourse(&head);

break;

case 5:

outputList(&head, &last);

break;

case 6:

outputFirstCourse(&head, &last);

break;

case 7:

outputSecondCourse(&head, &last);

break;

case 8:

outputThirdCourse(&head, &last);

break;

case 9:

outputFourthCourse(&head, &last);

break;

case 10:

calculateAverageScores(&head);

break;

}

} while (choice != 11);

return 0;

}

void insert(Student\* e, Student\*\* phead, Student\*\* plast) //Добавление в конец списка

{

if (\*plast == nullptr) {

\*plast = e;

\*phead = e;

}

else {

(\*plast)->next = e;

e->prev = \*plast;

\*plast = e;

}

}

Student\* setElement() // Создание элемента и ввод его значений с клавиатуры

{

Student\* temp = new Student();

if (!temp) {

cerr << "Ошибка выделения памяти";

return nullptr;

}

cout << "Введите фамилию: ";

cin >> temp->surname;

cout << "Введите имя: ";

cin >> temp->name;

cout << "Введите отчество: ";

cin >> temp->patronymic;

cout << "Введите дату рождения: " << endl;

cin >> temp->day >> temp->month >> temp->year;

cout << "Введите номер курса: ";

cin >> temp->course;

cout << "Введите номер группы: ";

cin >> temp->group;

cout << "Введите оценки по предметам: " << endl;

cout << "Математика: ";

cin >> temp->math;

cout << "Физика: ";

cin >> temp->physics;

cout << "Английский язык: ";

cin >> temp->english;

cout << "ОАиП: ";

cin >> temp->OAIP;

cout << "История Беларуси: ";

cin >> temp->history;

temp->next = nullptr;

temp->prev = nullptr;

return temp;

}

void outputList(Student\*\* phead, Student\*\* plast) // выводим всех студентов

{

Student\* t = \*phead;

if (t == nullptr) {

cout << "Список пуст!" << endl;

return;

}

while (t != nullptr)

{

cout << "--------------------------------------------------" << endl;

cout << "Фамилия:" << t->surname << endl;

cout << "Имя:" << t->name << endl;

cout << "Отчество:" << t->patronymic << endl;

cout << "Дата рождения:" << t->day << "." << t->month << "." << t->year << endl;

cout << "Курс:" << t->course << endl;

cout << "Группа:" << t->group << endl;

cout << "Оценки:" << endl;

cout << "Математика:" << t->math << endl;

cout << "Физика: " << t->physics << endl;

cout << "Английский язык: " << t->english << endl;

cout << "ОАиП:" << t->OAIP << endl;

cout << "История Беларуси: " << t->history << endl;

t = t->next;

}

}

void studentOld(Student\*\* phead, Student\*\* plast) // ищем старшего студента

{

int old = 0;

Student\* tmp = \*phead;

old = tmp->year;

while (tmp)

{

if (tmp->year < old)

{

old = tmp->year;

}

tmp = tmp->next;

}

Student\* t = \*phead;

while (t)

{

if (t->year == old)

{

cout << "Старший студент = " << t->surname << endl;

}

t = t->next;

}

}

void studentYoung(Student\*\* phead, Student\*\* plast) // ищем младшего студента

{

int young = 0;

Student\* tmp = \*phead;

young = tmp->year;

while (tmp)

{

if (tmp->year > young)

{

young = tmp->year;

}

tmp = tmp->next;

}

Student\* t = \*phead;

while (t)

{

if (t->year == young)

{

cout << "Младший студент = " << t->surname << endl;

}

t = t->next;

}

}

void sortStudentsByCourse(Student\*\* phead) // Сротировка студентов по курсу и фамилии

{

Student\* current = \*phead;

int swapped;

Student\* temp;

if (current == NULL || current->next == NULL)

{

return;

}

do {

swapped = 0;

current = \*phead;

while (current->next != nullptr)

{

// Если курс следующего студента меньше текущего или фамилия следующего при однаковом курсе раньше в алфавитном порядке то меняем их местами

if (current->course > current->next->course || (current->course == current->next->course && strcmp(current->surname, current->next->surname) > 0))

{

temp = current->next;

current->next = temp->next;

temp->next = current;

if (current->prev != NULL)

{

current->prev->next = temp;

}

temp->prev = current->prev;

current->prev = temp;

if (current == \*phead)

{

\*phead = temp;

}

if (current->next != NULL)

{

current->next->prev = current;

}

swapped = 1;

}

else

{

current = current->next;

}

}

} while (swapped);

}

void outputFirstCourse(Student\*\* phead, Student\*\* plast) // первый курс

{

Student \*t = \*phead;

float av = 0, temp;

char bestud[SIZE];

// среднее значение

if (t == NULL)

{

cout << "Список пуст!" << endl;

}

while (t)

{

if (t->course == 1)

{

temp = (t->math + t->physics + t->history + t->OAIP + t->english)/5;

cout << "Фамилия:" << t->surname << endl;

cout << "Имя:" << t->name << endl;

cout << "Отчество:" << t->patronymic << endl;

cout << "Дата рождения:" << t->day << "." << t->month << "." << t->year << endl;

cout << "Курс:" << t->course << endl;

cout << "Группа:" << t->group << endl;

cout << "Оценки:" << endl;

cout << "Математика:" << t->math << endl;

cout << "Физика:" << t->physics << endl;

cout << "Английский язык:" << t->english << endl;

cout << "ОАиП:" << t->OAIP << endl;

cout << "История Беларуси:" << t->history << endl<<endl;

t = t->next;

if (t != nullptr && temp > av)

{

av = temp;

strcpy(bestud, t->surname); // Копируем фамилию лучшего студента в массив

}

}

else

{

t = t->next;

}

}

cout << endl << "Лучший студент: " << bestud<<endl;

}

void outputSecondCourse(Student\*\* phead, Student\*\* plast) // второй курс

{

Student\* t = \*phead;

float av = 0, temp;

char bestud[SIZE];

if (t == NULL)

{

cout << "Список пуст!" << endl;

}

while (t)

{

if (t->course == 2)

{

temp = (t->math + t->physics + t->history + t->OAIP + t->english) / 5;

cout << "Фамилия:" << t->surname << endl;

cout << "Имя:" << t->name << endl;

cout << "Отчество:" << t->patronymic << endl;

cout << "Дата рождения:" << t->day << "." << t->month << "." << t->year << endl;

cout << "Курс:" << t->course << endl;

cout << "Группа:" << t->group << endl;

cout << "Оценки:" << endl;

cout << "Математика:" << t->math << endl;

cout << "Физика:" << t->physics << endl;

cout << "Английский язык:" << t->english << endl;

cout << "ОАиП:" << t->OAIP << endl;

cout << "История Беларуси:" << t->history << endl;

t = t->next;

if (t != nullptr && temp > av)

{

av = temp;

strcpy(bestud, t->surname); // Копируем фамилию лучшего студента в массив

}

}

else

t = t->next;

}

}

void outputThirdCourse(Student\*\* phead, Student\*\* plast) //третий курс

{

Student\* t = \*phead;

float av = 0, temp;

char bestud[SIZE];

if (t == NULL)

{

cout << "Список пуст!" << endl;

}

while (t)

{

if (t->course == 3)

{

temp = (t->math + t->physics + t->history + t->OAIP + t->english) / 5;

cout << "Фамилия:" << t->surname << endl;

cout << "Имя:" << t->name << endl;

cout << "Отчество:" << t->patronymic << endl;

cout << "Дата рождения:" << t->day << "." << t->month << "." << t->year << endl;

cout << "Курс:" << t->course << endl;

cout << "Группа:" << t->group << endl;

cout << "Оценки:" << endl;

cout << "Математика:" << t->math << endl;

cout << "Физика:" << t->physics << endl;

cout << "Английский язык:" << t->english << endl;

cout << "ОАиП:" << t->OAIP << endl;

cout << "История Беларуси:" << t->history << endl;

t = t->next;

if (t != nullptr && temp > av)

{

av = temp;

strcpy(bestud, t->surname); // Копируем фамилию лучшего студента в массив

}

}

else

t = t->next;

}

}

void outputFourthCourse(Student\*\* phead, Student\*\* plast) //четвёртый курс

{

Student\* t = \*phead;

float av = 0, temp;

char bestud[SIZE];

if (t == NULL)

{

cout << "Список пуст!" << endl;

}

while (t)

{

if (t->course == 4)

{

temp = (t->math + t->physics + t->history + t->OAIP + t->english) / 5;

cout << "Фамилия:" << t->surname << endl;

cout << "Имя:" << t->name << endl;

cout << "Отчество:" << t->patronymic << endl;

cout << "Дата рождения:" << t->day << "." << t->month << "." << t->year << endl;

cout << "Курс:" << t->course << endl;

cout << "Группа:" << t->group << endl;

cout << "Оценки:" << endl;

cout << "Математика:" << t->math << endl;

cout << "Физика:" << t->physics << endl;

cout << "Английский язык:" << t->english << endl;

cout << "ОАиП:" << t->OAIP << endl;

cout << "История Беларуси:" << t->history << endl;

t = t->next;

if (t != nullptr && temp > av)

{

av = temp;

strcpy(bestud, t->surname); // Копируем фамилию лучшего студента в массив

}

}

else

t = t->next;

}

}

void calculateAverageScores(Student\*\* phead) // Расчет средних баллов по каждому предмету для каждой группы

{

const int MAX\_GROUPS = 10;

int numStudents[MAX\_GROUPS] = { 0 }; // Массив для хранения количества студентов в каждой группе

float math[MAX\_GROUPS] = { 0 }, physics[MAX\_GROUPS] = { 0 }, totalEnglish[MAX\_GROUPS] = { 0 }, totalOAIP[MAX\_GROUPS] = { 0 }, totalHistory[MAX\_GROUPS] = { 0 };

// Проходим по списку студентов и суммируем оценки по каждому предмету для каждой группы

Student\* current = \*phead;

while (current != nullptr)

{

int groupIndex = current->group - 1; // Индекс группы в массиве

numStudents[groupIndex]++;

math[groupIndex] += current->math;

physics[groupIndex] += current->physics;

totalEnglish[groupIndex] += current->english;

totalOAIP[groupIndex] += current->OAIP;

totalHistory[groupIndex] += current->history;

current = current->next;

}

// Вычисляем средний балл по каждому предмету для каждой группы

for (int i = 0; i < MAX\_GROUPS; i++)

{

if (numStudents[i] != 0) {

float averageMath = math[i] / numStudents[i];

float averagePhysics = physics[i] / numStudents[i];

float averageEnglish = totalEnglish[i] / numStudents[i];

float averageOAIP = totalOAIP[i] / numStudents[i];

float averageHistory = totalHistory[i] / numStudents[i];

// Выводим результаты для каждой группы

cout << "Группа " << i + 1 << ":" << endl;

cout << "Средний балл по Математике: " << averageMath << endl;

cout << "Средний балл по Физике: " << averagePhysics << endl;

cout << "Средний балл по Английскому языку: " << averageEnglish << endl;

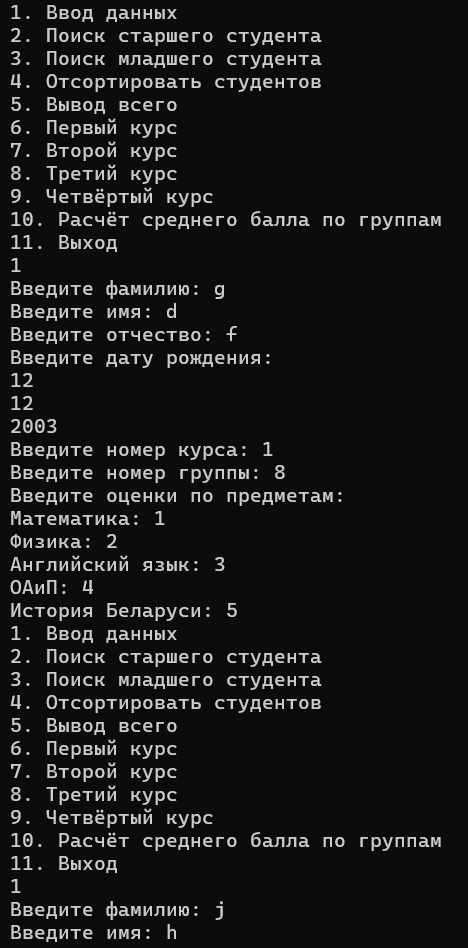
cout << "Средний балл по ОАиПу: " << averageOAIP << endl;

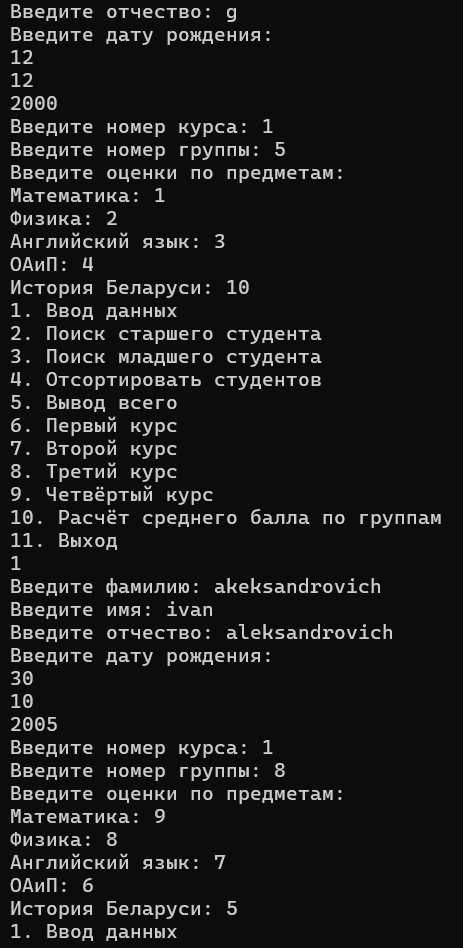
cout << "Средний балл по Истории Беларуси: " << averageHistory << endl;

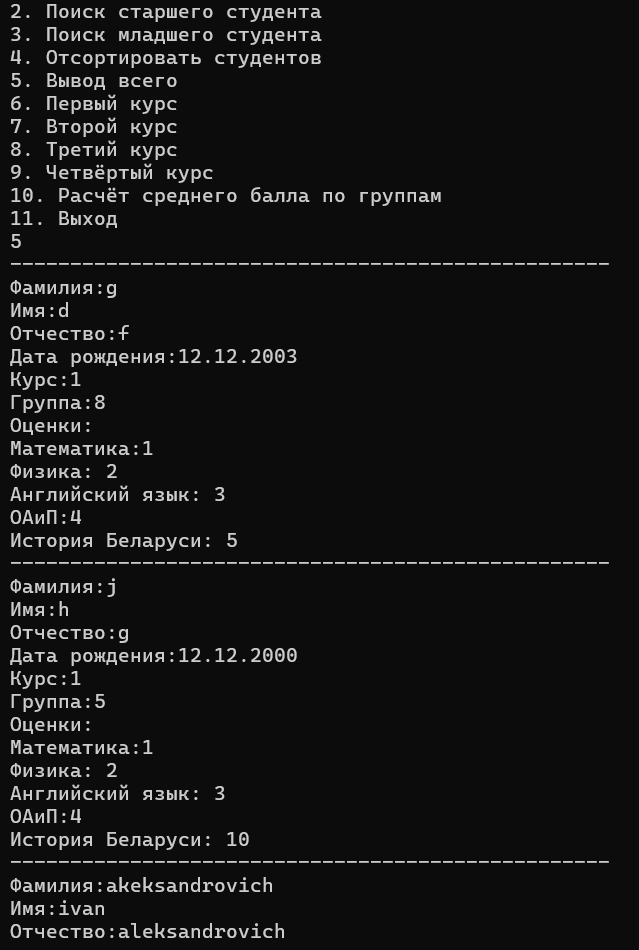
}

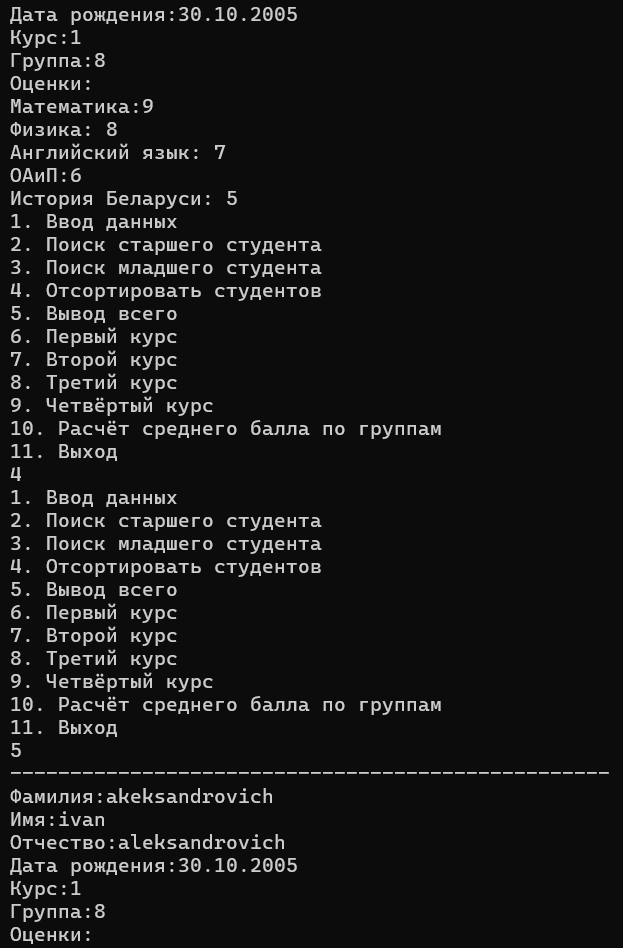
}

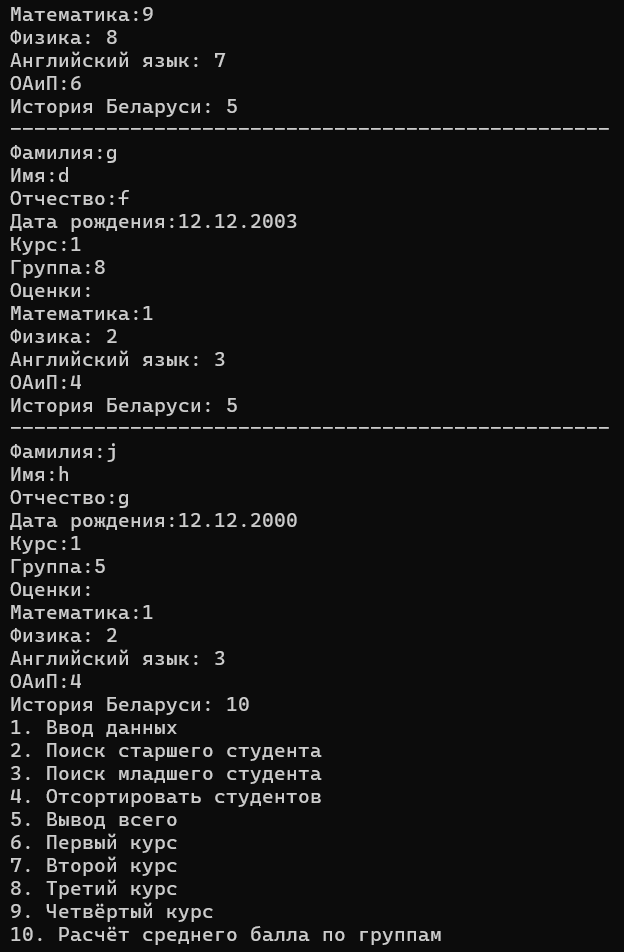
}

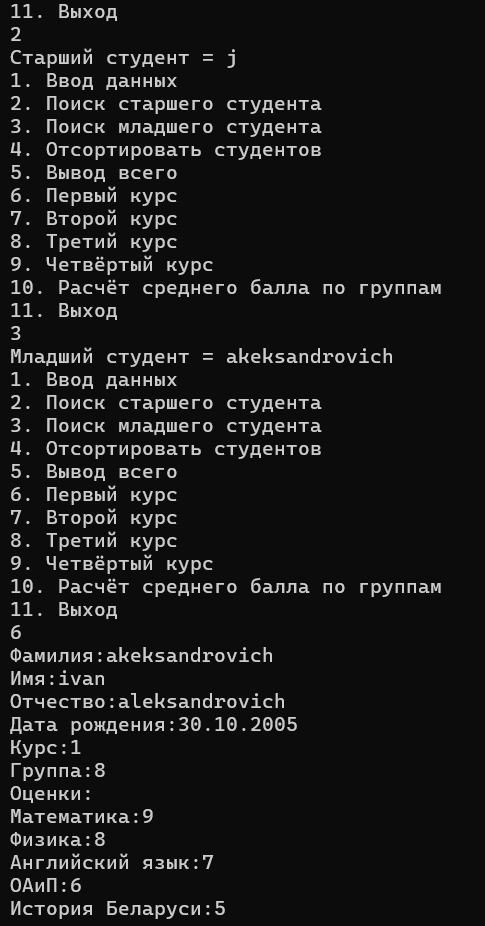


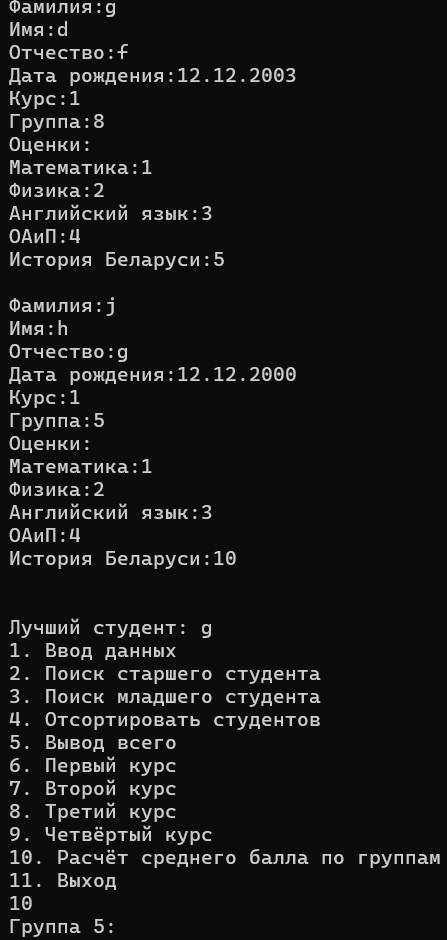


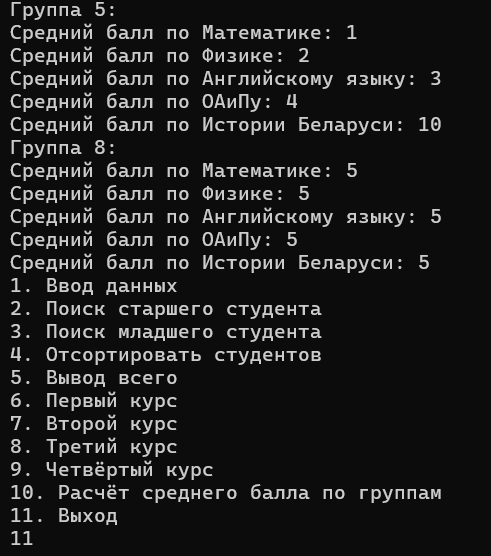












1



Main.cpp

#include "List.h"

using namespace std;

void print(char\* carmodel, char\* color, char\* carnumb, char\* daterealese, char\* typecar, int datetech, char\* owner)

{

cout << carmodel << " " << color << " " << carnumb<< " " << daterealese<< " " << typecar <<" " << datetech<< " " << owner << endl;

}

int main()

{

setlocale(LC\_ALL, "Russian");

int choice;

Object L1 = Create();

do{

cout << "Выберите вариант: " << endl;

cout << "1. Внести элемент в начало списка" << endl;

cout << "2. Вывод" << endl;

cout << "3. Поиск элемента" << endl;

cout << "4. Удаление элемента" << endl;

cout << "5. Удалить список" << endl;

cout << "6. Подсчёт число элементов списка" << endl;

cout << "7. Выход" << endl;

cin >> choice;

switch (choice)

{

case 1:

{

char Carmodel[50];

char Color[50];

char Carnumb[50];

char Daterelease[50];

char Typecar[50];

int Datetech;

char Owner[50];

cout << "Марка автомобиля: ";

cin >> Carmodel;

cout << "Цвет: ";

cin >> Color;

cout << "Заводской номер: ";

cin >> Carnumb;

cout << "Дата выпуска: ";

cin >> Daterelease;

cout << "Тип кузова: ";

cin >> Typecar;

cout << "Дата техосмотра: ";

cin >> Datetech;

cout << "Владелец: ";

cin >> Owner;

L1.Insert(Carmodel, Color, Carnumb, Daterelease, Typecar, Datetech, Owner);

break;

}

case 2:

{

L1.PrintList(print);

break;

}

case 3:

{

char owner[50];

cout << "Введите Владельца: ";

cin >> owner;

Element\* e = L1.Search(owner);

if (e == NULL)

{

cout << "Элемент не найден" << endl;

}

else

{

cout << "Найден элемент= " << e->Carmodel <<" " << e->Color << " " << e->Carnumb << " "<< e->Daterelease << " " << e->Typecar<< " "<< e->Datetech << " " << e->Owner << endl;

}

break;

}

case 4:

{

char carnumb[50];

cout << "Введите владельца для удаления: ";

cin >> carnumb;

bool rc = L1.Delete(carnumb);

if (rc == true) cout << "Удален элемент" << endl;

break;

}

case 5:

{

if (L1.DeleteList())

{

cout << "Список успешно удалён" << endl;

}

else {

cout << "Ошибка удаления" << endl;

}

break;

}

case 6: {

cout << "Количество элементов: " << L1.CountList() << endl;

break;

}

}

} while (choice != 7);

return 0;

}

List.cpp

#include "List.h"

bool Object::Insert(char\* carmodel, char\* color, char\* carnumb, char\* daterealese, char\* typecar, int datetech, char\* owner) // Вставка в начало

{

bool rc = 0;

if (Head == NULL) // если список пуст

{

Head = new Element(NULL, carmodel, color, carnumb, daterealese, typecar, datetech, owner, Head);

rc = true;

}

else

{

Head = (Head->Prev = new Element(NULL, carmodel, color, carnumb, daterealese, typecar, datetech, owner, Head));

rc = true;

}

return rc;

}

//-----------------------------------------------------------

Element\* Object::Search(char\* owner) // Найти заданный элемент

{

Element\* rc = Head;

while ((rc != NULL) && (strcmp(rc->Owner, owner)))

rc = rc->Next;

return rc;

}

//-----------------------------------------------------------

void Object::PrintList(void(\*f)(char\*, char\*, char\*, char\*, char\*, int, char\*)) // Вывод // передача функции как аргумента

{

Element\* e = Head;

while (e != NULL)

{

f(e->Carmodel, e->Color, e->Carnumb, e->Daterelease, e->Typecar, e->Datetech, e->Owner); // использование формального аргумента функции для вывода элемента

e = e->GetNext(); // получение указателя на следующий элемент с помощью метода

};

}

//-----------------------------------------------------------

void Object::PrintList(void(\*f)(char\*, char\*, char\*, char\*, char\*, int, char\*), Element\* e)

{

f(e->Carmodel, e->Color,e->Carnumb, e->Daterelease, e->Typecar,e->Datetech,e->Owner);

}

//-----------------------------------------------------------

bool Object::Delete(Element\* e) // Удалить по ссылке

{

bool rc = 0;

if (rc = (e != NULL))

{

if (e->Next != NULL)

e->Next->Prev = e->Prev;

if (e->Prev != NULL)

e->Prev->Next = e->Next;

else

Head = e->Next;

delete e;

}

return rc;

}

//-----------------------------------------------------------

bool Object::Delete(char\* owner) // Удалить по значению

{

return Delete(Search(owner));

};

//-----------------------------------------------------------

Element\* Object::GetLast()

{

Element\* e = Head, \* rc = e;

while (e != NULL)

{

rc = e;

e = e->GetNext();

}

return rc;

}

//-----------------------------------------------------------

Object Create()

{

return \*(new Object()); // разыменование имени конструктора с выделением памяти под объект, а также выполнение операций внутри конструктора

}

int Object::CountList()

{

Element\* p = Head;

int counter = 0;

while (p) {

p = p->Next;

counter++;

}

return counter;

}

bool Object::DeleteList()

{

while (Head) {

Element\* p = Head;

Head = Head->Next;

delete[]p;

}

return Head == NULL ? 1 : 0;

}

List.h

#pragma once

#include <iostream>

struct Element // Элемент списка

{

Element\* Prev; // указатель на предыдущий элемент

Element\* Next; // указатель на следующий элемент

char Carmodel[50];

char Color[50];

char Carnumb[50];

char Daterelease[50];

char Typecar[50];

int Datetech;

char Owner[50]; // данные

Element(Element\* prev, char\* carmodel, char \*color, char\* carnumb, char\* daterealese, char\* typecar,int datetech, char \* owner, Element\* next) // конструктор для создания элемента

{

Prev = prev;

strcpy\_s(Carmodel,carmodel);

strcpy\_s(Color, color);

strcpy\_s(Carnumb, carnumb);

strcpy\_s(Daterelease, daterealese);

strcpy\_s(Typecar, typecar);

Datetech = datetech;

strcpy\_s(Owner, owner);

Next = next;

}

Element\* GetNext() // получить следующий

{

return Next;

};

Element\* GetPrev() // получить предыдущий

{

return Prev;

};

};

//-----------------------------------------------------------

struct Object // Блок управления списком

{

Element\* Head; // указатель на начало списка

Object()

{

Head = NULL;

};

Element\* GetFirst() // получить первый элемент списка

{

return Head;

};

Element\* GetLast(); // получить последний элемент списка

Element\* Search(char\*); // найти первый элемент по данным

bool Insert(char\*, char\*, char\*, char\*, char\*, int, char\*); // добавить элемент в начало

bool InsertEnd(char \*, char\*, char\*, char\*, char\*, int, char\*); // добавить в конец

bool Delete(Element\* e); // удалить по адресу элемента

bool Delete(char\*); // удалить первый по данным

bool DeleteList(); // очистить список

void PrintList(void(\*f)(char\*, char\*, char\*, char\*, char\*, int, char\*));

void PrintList(void(\*f)(char\*, char\*, char\*, char\*, char\*, int, char\*), Element\*);

int CountList();

bool DeleteDouble();

};

Object Create();

