Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа №9

По дисциплине «Основы алгоритмизации и программирования»

На тему «**Полустатические структуры данных: очереди**»

Выполнил:

Студент 1 курса 10 группы

Мандрик Алексей Иванович

Преподаватель: асс. Андронова М.В.

Минск, 2024

Дополнить программу функцией в соответствии со своим вариантом из таблицы, представленной ниже.

**Вариант 8**



**Код программы:**

#include <iostream>

using namespace std;

// Структура узла односвязного списка

struct Node {

double data; // Данные узла

Node\* prev, \* pnext; // Указатели на предыдущий и следующий узлы

public:

// Конструктор узла

Node(double data) {

this->data = data;

this->prev = this->pnext = NULL;

}

};

// Односвязный список

struct LinkedL {

public:

Node\* head, \* tail; // Указатели на голову и хвост списка

public:

// Конструктор списка

LinkedL() {

head = tail = NULL;

}

// Метод добавления элемента в список

Node\* push(double data) {

Node\* ptr = new Node(data);

ptr->pnext = head;

if (head != NULL) {

head->prev = ptr;

}

if (tail == NULL) {

tail = ptr;

}

head = ptr;

return ptr;

}

};

// Структура узла второго списка

struct Node2 {

double data2; // Данные узла

Node2\* prev2, \* pnext2; // Указатели на предыдущий и следующий узлы

public:

// Конструктор узла

Node2(double data2) {

this->data2 = data2;

this->prev2 = this->pnext2 = NULL;

}

};

// Структура узла третьего списка

struct Node3 {

double data3; // Данные узла

Node3\* prev3, \* pnext3; // Указатели на предыдущий и следующий узлы

public:

// Конструктор узла

Node3(double data3) {

this->data3 = data3;

this->prev3 = this->pnext3 = NULL;

}

};

// Односвязный список второго типа

struct LinkedL2 {

public:

Node2\* head2, \* tail2; // Указатели на голову и хвост списка

public:

// Конструктор списка

LinkedL2() {

head2 = tail2 = NULL;

}

// Метод добавления элемента в список

Node2\* push(double data2) {

Node2\* ptr2 = new Node2(data2);

ptr2->pnext2 = head2;

if (head2 != NULL) {

head2->prev2 = ptr2;

}

if (tail2 == NULL) {

tail2 = ptr2;

}

head2 = ptr2;

return ptr2;

}

};

// Односвязный список третьего типа

struct LinkedL3 {

public:

Node3\* head3, \* tail3; // Указатели на голову и хвост списка

public:

// Конструктор списка

LinkedL3() {

head3 = tail3 = NULL;

}

// Метод добавления элемента в список

Node3\* push(double data3) {

Node3\* ptr3 = new Node3(data3);

ptr3->pnext3 = head3;

if (head3 != NULL) {

head3->prev3 = ptr3;

}

if (tail3 == NULL) {

tail3 = ptr3;

}

head3 = ptr3;

return ptr3;

}

};

int main() {

setlocale(LC\_ALL, "Rus");

int choice; // Переменная для выбора пользователя

LinkedL from1; // Первый список

LinkedL2 from2; // Второй список

LinkedL3 To; // Третий список

int ar1[100], ar2[100], s = 0, i = 0, j = 0; // Вспомогательные массивы и переменные

// Меню

do {

cout << "Меню:\n"

<< "1. Добавить числа в списки\n"

<< "2. Показать списки\n"

<< "3. Объединить списки\n"

<< "4. Выйти\n";

cin >> choice; // Ввод выбора пользователя

// Обработка выбора пользователя

switch (choice) {

case 1: {

cout << "Введите число для первого списка: ";

double n;

cin >> n;

from1.push(n);

cout << "Введите число для второго списка: ";

double n2;

cin >> n2;

from2.push(n2);

s++;

break;

}

case 2: {

cout << "Первый список:";

for (Node\* ptr = from1.head; ptr != NULL; ptr = ptr->pnext) {

cout << "-> " << ptr->data << " -> ";

}

cout << "NULL\n";

cout << "Второй список:";

for (Node2\* ptr2 = from2.head2; ptr2 != NULL; ptr2 = ptr2->pnext2) {

cout << "-> " << ptr2->data2 << " -> ";

}

cout << "NULL\n";

cout << "Третий список:";

for (Node3\* ptr3 = To.head3; ptr3 != NULL; ptr3 = ptr3->pnext3) {

cout << "-> " << ptr3->data3 << " -> ";

}

cout << "NULL\n";

break;

}

case 3: {

// Перенос данных из списков в массивы

for (Node\* ptr = from1.tail; ptr != NULL; ptr = ptr->prev) {

ar1[i] = ptr->data;

i++;

}

for (Node2\* ptr2 = from2.tail2; ptr2 != NULL; ptr2 = ptr2->prev2) {

ar2[j] = ptr2->data2;

j++;

}

// Объединение массивов в третий список

for (int t = 0; t < s; t++) {

To.push(ar2[t]);

To.push(ar1[t]);

}

break;

}

case 4:

cout << "Выход из программы...\n";

break;

default:

cout << "Неверный выбор. Пожалуйста, попробуйте снова.\n";

break;

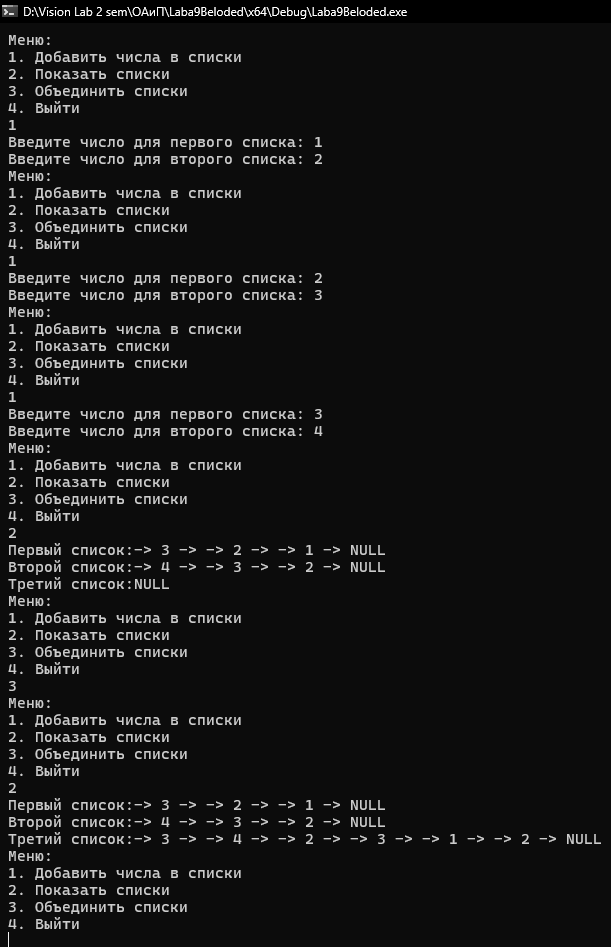
}

} while (choice != 4);

return 0;

}

}**Результат выполнения:**



**Дополнительные задания:**

**Доп 1:**

. В [приложении 2](file:///C:\Users\User\Downloads\ПрактикумОАП-2часть2021.doc#prilogenie2) приведен проект, в котором реализован проект с использованием *двусвязного* *списка*. На основе данного проекта разработать функции, которые предлагается создать в данном приложении.

**Код программы:**

**Main**

#include "List.h" // Подключение заголовочного файла с объявлением списка

#include <iostream> // Подключение стандартной библиотеки ввода-вывода

using namespace std; // Использование пространства имен std

// Структура для представления человека в списке

struct Person

{

char name[20]; // Имя человека

int age; // Возраст человека

Person\* next; // Указатель на следующего человека в списке

};

// Функция для печати информации о человеке

void print(void\* b) // Функция используется при выводе

{

Person\* a = (Person\*)b; // Приведение указателя к типу Person\*

cout << a->name << " " << a->age << endl; // Вывод имени и возраста человека

}

// Функция для отображения меню и ввода выбора пользователя

int menu() {

setlocale(LC\_ALL, "Rus"); // Установка локали для отображения русского текста

cout << "Меню:" << endl;

cout << "1) Вывод + кол-во элементов" << endl;

cout << "2) Удалить весь список" << endl;

int temp;

cin >> temp; // Ввод выбора пользователя

return temp;

}

// Главная функция программы

int main()

{

setlocale(LC\_ALL, "Russian"); // Установка локали для отображения русского текста

Person d = { "delete", 0 }; // Создание человека для удаления в списке

// Создание нескольких человек

Person a1 = { "Петров", 20 };

Person a2 = { "Сидоров", 25 };

Person a3 = { "Гончаров", 47 };

Object L1 = Create(); // Создание списка с именем L1

L1.Insert(&a1); // Вставка человека a1 в список

L1.Insert(&a2); // Вставка человека a2 в список

L1.Insert(&a3); // Вставка человека a3 в список

// Бесконечный цикл работы с меню

while (true) {

switch (menu()) { // Выбор пользователем действия из меню

case 1:

L1.PrintList(print); // Вывод списка с вызовом функции print для каждого элемента

cout << "\nКоличество элементов списка: " << L1.CountList(&(L1.Head)) << endl; // Вывод количества элементов в списке

break;

case 2:

L1.DeleteList(&(L1.Head), (L1.GetLast()), L1.CountList(&(L1.Head))); // Удаление всего списка

break;

default:

exit(1); // Выход из программы

}

}

return 0;

}

**List.cpp**

#include "List.h" // Подключение заголовочного файла с объявлением списка

// Метод вставки элемента в список

bool Object::Insert(void\* data)

{

bool rc = 0;

if (Head == NULL) // Если список пуст

{

Head = new Element(NULL, data, Head); // Создаем новый элемент и делаем его головой списка

rc = true;

}

else

{

Head = (Head->Prev = new Element(NULL, data, Head)); // Иначе создаем новый элемент и вставляем его перед текущей головой

rc = true;

}

return rc;

}

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

// Метод поиска элемента в списке

Element\* Object::Search(void\* data)

{

Element\* rc = Head; // Начинаем поиск с головы списка

while ((rc != NULL) && (rc->Data != data)) // Пока не достигнут конец списка или не найден элемент

rc = rc->Next; // Переходим к следующему элементу

return rc; // Возвращаем найденный элемент или NULL, если элемент не найден

}

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

// Метод печати списка с использованием заданной функции

void Object::PrintList(void(\*f)(void\*))

{

Element\* e = Head; // Начинаем с головы списка

while (e != NULL) // Пока не достигнут конец списка

{

if (e->Data != smert) { // Если элемент не помечен для удаления

f(e->Data); // Вызываем функцию для печати элемента

}

e = e->GetNext(); // Переходим к следующему элементу

};

}

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

// Метод печати одного элемента списка с использованием заданной функции

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

{

f(e->Data); // Вызываем функцию для печати элемента

}

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

// Метод удаления элемента из списка

bool Object::Delete(Element\* e)

{

bool rc = 0;

if (rc = (e != NULL)) // Если элемент существует

{

if (e->Next != NULL)

e->Next->Prev = e->Prev; // Переносим указатель на следующий элемент

if (e->Prev != NULL)

e->Prev->Next = e->Next; // Переносим указатель на предыдущий элемент

else

Head = e->Next; // Если элемент является головой списка, то меняем указатель на голову

delete e; // Удаляем элемент

}

return rc; // Возвращаем результат операции удаления

}

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

// Метод удаления элемента по его данным

bool Object::Delete(void\* data)

{

return Delete(Search(data)); // Удаляем элемент, найденный по данным

};

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

// Метод получения указателя на последний элемент списка

Element\* Object::GetLast()

{

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

while (e != NULL) // Пока не достигнут конец списка

{

rc = e; // Запоминаем текущий элемент как последний найденный

e = e->GetNext(); // Переходим к следующему элементу

}

return rc; // Возвращаем указатель на последний элемент

}

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

// Функция создания объекта списка

Object Create()

{

return \*(new Object()); // Создаем новый объект списка и возвращаем его

}

// Метод для подсчета количества элементов в списке, исключая помеченные для удаления

int Object::CountList(Element\*\* Head) {

Element\* e = \*Head; // Начинаем с головы списка

int i = 0;

while (e != NULL) // Пока не достигнут конец списка

{

if (e->Data != smert) // Если элемент не помечен для удаления

i++; // Увеличиваем счетчик

e = e->GetNext(); // Переходим к следующему элементу

}

return i; // Возвращаем количество элементов

}

// Метод для пометки всех элементов списка как удаленные

void Object::DeleteList(Element\*\* phead, Element\* plast, int k) {

Element\* t = \*phead; // Начинаем с головы списка

while (t != NULL) { // Пока не достигнут конец списка

t->Data = smert; // Помечаем элемент для удаления

t = t->Next; // Переходим к следующему элементу

}

}

**List.h**

#pragma once

#define NULL 0

#include "fstream" // Подключение заголовочного файла для работы с файлами

// Массив символов для пометки удаляемых элементов

\_\_declspec(selectany) char smr[7] = "delete";

\_\_declspec(selectany) void\* smert = smr; // Указатель на массив для пометки

// Структура элемента списка

struct Element

{

Element\* Prev; // Указатель на предыдущий элемент списка

Element\* Next; // Указатель на следующий элемент списка

void\* Data; // Указатель на данные элемента

Element(Element\* prev, void\* data, Element\* next)

{

Prev = prev;

Data = data;

Next = next;

}

Element\* GetNext() // Функция получения следующего элемента списка

{

return Next;

};

Element\* GetPrev() // Функция получения предыдущего элемента списка

{

return Prev;

};

};

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

// Структура списка

struct Object

{

Element\* Head; // Указатель на голову списка

Object()

{

Head = NULL; // Инициализация головы списка как NULL

};

Element\* GetFirst() // Функция получения первого элемента списка

{

return Head;

};

Element\* GetLast(); // Функция получения последнего элемента списка

Element\* Search(void\* data); // Функция поиска элемента по данным

bool Insert(void\* data); // Функция вставки элемента в список

bool InsertEnd(void\* data); // Функция вставки элемента в конец списка

bool Delete(Element\* e); // Функция удаления элемента из списка

bool Delete(void\* data); // Функция удаления элемента по данным

int CountList(Element\*\* Head); // Функция подсчета количества элементов в списке

void DeleteList(Element\*\* phead, Element\* plast, int k); // Функция удаления всех элементов списка

void PrintList(void(\*f)(void\*)); // Функция печати списка с использованием заданной функции

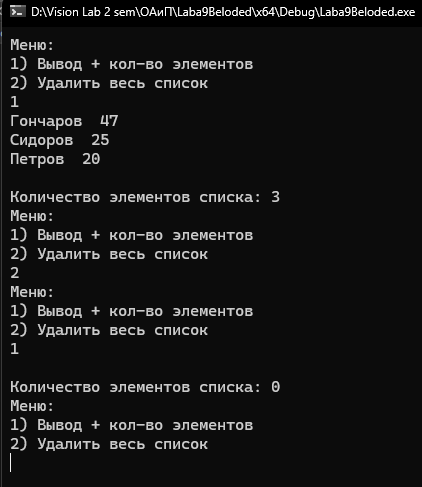
void PrintList(void(\*f)(void\*), Element\*); // Функция печати одного элемента списка с использованием заданной функции

bool DeleteDouble(); // Функция удаления повторяющихся элементов

};

Object Create(); // Функция создания объекта списка

**Результат выполнения:**

****

**Доп 2:**

Каждый элемент списка студентов содержит фамилию, имя, отчество, год рождения, курс, номер группы, оценки по пяти предметам. Упорядочить студентов по курсу, причем студенты одного курса должны располагаться в алфавитном порядке. Найти средний балл каждой группы по каждому предмету. Определить самого старшего студента и самого младшего. Для каждой группы найти лучшего с точки зрения успеваемости студента.

**Код программы:**

#include <iostream>

#include <string>

#include <list>

#include <algorithm>

using namespace std;

struct Student {

string lastName; // Фамилия

string firstName; // Имя

string middleName; // Отчество

int birthYear; // Год рождения

int course; // Курс

int groupNumber; // Номер группы

int marks[5]; // Оценки по пяти предметам

bool operator<(const Student& other) const {

if (course != other.course)

return course < other.course;

return lastName < other.lastName;

}

};

struct GroupAverage {

int groupNumber; // Номер группы

double averages[5]; // Средние баллы по каждому предмету

};

struct BestStudent {

int groupNumber; // Номер группы

Student student; // Студент

};

// Функция для вычисления среднего балла каждой группы по каждому предмету

void calculateGroupAverages(const list<Student>& students, list<GroupAverage>& groupAverages) {

for (const auto& student : students) {

for (auto& groupAverage : groupAverages) {

if (groupAverage.groupNumber == student.groupNumber) {

for (int i = 0; i < 5; ++i) {

groupAverage.averages[i] += student.marks[i];

}

break;

}

}

}

for (auto& groupAverage : groupAverages) {

int count = 0;

for (const auto& student : students) {

if (student.groupNumber == groupAverage.groupNumber) {

++count;

}

}

if (count != 0) {

for (int i = 0; i < 5; ++i) {

groupAverage.averages[i] /= count;

}

}

}

}

// Функция для нахождения старшего студента

Student findOldest(const list<Student>& students) {

auto oldest = students.front();

for (const auto& student : students) {

if (student.birthYear < oldest.birthYear) {

oldest = student;

}

}

return oldest;

}

// Функция для нахождения младшего студента

Student findYoungest(const list<Student>& students) {

auto youngest = students.front();

for (const auto& student : students) {

if (student.birthYear > youngest.birthYear) {

youngest = student;

}

}

return youngest;

}

// Функция для нахождения лучшего студента в группе

BestStudent findBestStudent(const list<Student>& students, int groupNumber) {

BestStudent best;

best.groupNumber = groupNumber;

double maxAverage = 0;

for (const auto& student : students) {

if (student.groupNumber == groupNumber) {

double totalMarks = 0;

for (int i = 0; i < 5; ++i) {

totalMarks += student.marks[i];

}

double average = totalMarks / 5;

if (average > maxAverage) {

maxAverage = average;

best.student = student;

}

}

}

return best;

}

int main() {

setlocale(LC\_ALL, "Rus");

list<Student> students = {

{"Иванов", "Иван", "Иванович", 2005, 1, 10, {6, 6, 6, 6, 6}},

{"Петров", "Петр", "Петрович", 2004, 2, 10, {5, 5, 5, 5, 5}},

{"Сидоров", "Сидор", "Сидорович", 2004, 2, 9, {6, 8, 4, 6, 6}},

// добавьте еще студентов по вашему усмотрению

};

students.sort(); // Сортировка студентов по курсу и фамилии

list<GroupAverage> groupAverages;

for (const auto& student : students) {

bool exists = false;

for (const auto& groupAverage : groupAverages) {

if (groupAverage.groupNumber == student.groupNumber) {

exists = true;

break;

}

}

if (!exists) {

groupAverages.push\_back({ student.groupNumber, {0} });

}

}

calculateGroupAverages(students, groupAverages); // Вычисление средних баллов для каждой группы

cout << "Средние оценки для каждой группы:\n";

for (const auto& groupAverage : groupAverages) {

cout << "Группа " << groupAverage.groupNumber << ": ";

for (double average : groupAverage.averages) {

cout << average << " ";

}

cout << endl;

}

cout << "\nСтарший студент: " << findOldest(students).firstName << " " << findOldest(students).lastName << endl;

cout << "Младший студент: " << findYoungest(students).firstName << " " << findYoungest(students).lastName << endl;

cout << "\nЛучшие студенты в каждой группе:\n";

for (const auto& groupAverage : groupAverages) {

auto bestStudent = findBestStudent(students, groupAverage.groupNumber);

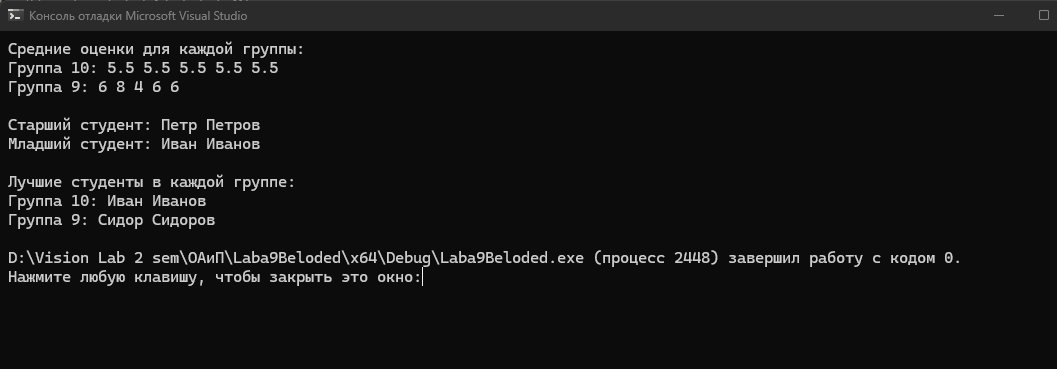
cout << "Группа " << bestStudent.groupNumber << ": " << bestStudent.student.firstName << " " << bestStudent.student.lastName << endl;

}

return 0;

}

**Результат выполнения:**



**Доп 3: (Вариант 2)**



**Код программы:**

#include <iostream>

#include <fstream>

using namespace std;

const unsigned int NAME\_SIZE = 30;

const unsigned int CITY\_SIZE = 20;

struct Address

{

char name[NAME\_SIZE]; // Имя

char city[CITY\_SIZE]; // Город

Address\* next; // Указатель на следующий элемент списка

Address\* prev; // Указатель на предыдущий элемент списка

};

// Функция для отображения меню и возврата выбранной опции

int menu()

{

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. Удалить первые k" << endl;

cout << endl;

do

{

cout << "Ваш выбор: ";

cin.sync();

gets\_s(s);

cout << endl;

c = atoi(s);

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

return c;

}

// Функция для добавления элемента в конец списка

void insert(Address\* e, Address\*\* phead, Address\*\* plast)

{

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;

}

}

// Функция для создания элемента и ввода его значений с клавиатуры

Address\* setElement()

{

Address\* temp = new Address();

if (!temp)

{

cerr << "Ошибка выделения 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;

}

// Функция для вывода списка на экран

void outputList(Address\*\* phead, Address\*\* plast)

{

Address\* t = \*phead;

while (t)

{

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

t = t->next;

}

cout << "" << endl;

}

// Функция для поиска имени в списке

void find(char name[NAME\_SIZE], Address\*\* phead)

{

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;

}

// Функция для удаления имени из списка

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

{

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;

}

}

// Функция для удаления первых k элементов из списка

void deleteKfirst(int k, Address\*\* phead, Address\*\* plast)

{

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

Address\* t = \*phead;

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;

}

}

// Главная функция

int main()

{

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:

cout << "Введите k = "; int k; cin >> k;

deleteKfirst(k, &head, &last);

menu();

break;

default: exit(1);

}

}

return 0;

}

**Результат выполнения:**

