Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа №9

По дисциплине «Основы алгоритмизации и программирования»

На тему «Двусвязные списки»

Выполнила:

Студентка 1 курса 6 группы

Литвинчук Дарья Валерьевна

Преподаватель: асс. Андронова М.В.

2024, Минск

Вариант 10



#include <iostream>

#include <fstream>

#include <Windows.h>

using namespace std;

const unsigned int NAME\_SIZE = 30;

const unsigned int CITY\_SIZE = 20;

//Структура, содержащая адрес людей

struct Address {

char name[NAME\_SIZE]; //имя человека

char city[CITY\_SIZE]; //родной город

Address\* next; //следующий человек в списке

Address\* prev; //предыдущий человек в списке

};

//Меню программы

int menu() {

short choice;

cout << endl;

cout << "1. Ввод имени" << endl;

cout << "2. Удаление по имени" << endl;

cout << "3. Удаление первого встречающегося элемента по городу" << endl;

cout << "4. Вывод на экран" << endl;

cout << "5. Поиск" << endl;

cout << "6. Выход" << endl;

cout << endl;

do {

cout << "Ваш выбор: ";

cin >> choice;

} while (choice < 0 || choice > 6);

return choice;

}

//Добавление элемента в конец списка

void insertElement(Address\* e, Address\*\* phead, Address\*\* plast) {

Address\* p = \*plast; //Временный указатель на конец списка

if (\*plast == NULL) { //Если список пуст

e->next = NULL;

e->prev = NULL;

\*plast = e; //Устанавливаем начало списка на добавленный элемент

\*phead = e; //Устанавливаем конец списка на добавленный элемент

}

else { //Если в списке есть элементы

p->next = e; //Указываем, что следующий элемент в существующем спике - это добавленный элемент

e->next = NULL; //Следюущий нулевой

e->prev = p; //Предыдущий уже находившийся в списке

\*plast = e; //Конец список перенаправляем на добавленный элемент

}

}

// Создание элемента и ввод его значений с клавиатуры

Address\* setElement() {

Address\* temp = new Address(); //

if (!temp) {

cerr << "Ошибка выделения памяти памяти";

return NULL;

}

cout << "Введите имя: ";

cin.ignore(cin.rdbuf()->in\_avail()); //очистка буфера для корректного ввода

cin.clear();

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

cout << "Введите город: ";

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

cin.clear();

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

temp->next = NULL;

temp->prev = NULL;

return temp;

}

//Вывод списка на экран

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

Address\* t = \*phead; //Указатель на начало списка

while (t) { //Пока указатель не NULL, выводим список

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

t = t->next;

}

cout << "" << endl;

}

// Поиск имени в списке

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

Address\* t = \*phead; //Указатель на начало списка

while (t) { //Поиск элемента

if (!strcmp(name, t->name)) { //Если элемент найден, выходим из поиска

break;

}

t = t->next;

}

if (!t) {

cerr << "Имя не найдено" << endl;

}

else {

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

}

}

// Удаление имени

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

struct Address\* t = \*phead; //Указатель на начало списка

while (t) {

if (!strcmp(name, t->name)) { //Если элемент найден, выходим из поиска

break;

}

t = t->next;

}

if (!t) {

cerr << "Имя не найдено" << endl;

}

else {

if (\*phead == t) { //Если элемент находится в начале списке

\*phead = t->next;

if (\*phead) {

(\*phead)->prev = NULL;

}

else {

\*plast = NULL;

}

}

else { //Если элемент где-то не в начале

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

if (t != \*plast) {

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

}

else {

\*plast = t->prev;

}

}

delete t;

cout << "Элемент удален" << endl;

}

}

//Удаление первого встречающегося элемента по городу

void deleteByCity(char city[CITY\_SIZE], Address\*\* phead, Address\*\* plast) {

struct Address\* t = \*phead; //Указатель на начало списка

while (t) {

if (!strcmp(city, t->city)) { //Если элемент найден, выходим из поиска

break;

}

t = t->next;

}

if (!t) {

cerr << "Город не найден" << endl;

}

else {

if (\*phead == t) { //Если элемент находится в начале списке

\*phead = t->next;

if (\*phead) {

(\*phead)->prev = NULL;

}

else {

\*plast = NULL;

}

}

else { //Если элемент где-то не в начале

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

if (t != \*plast) {

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

}

else {

\*plast = t->prev;

}

}

delete t;

cout << "Элемент удален" << endl;

}

}

//главная функция

int main() {

Address\* head = NULL; //создание указателя на начало списка

Address\* last = NULL; //создание указателя на конец списка

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

while (true) {

switch (menu()) {

case 1: {

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

break;

}

case 2: {

char dname[NAME\_SIZE];

cout << "Введите имя: ";

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

cin.sync();

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

deleteByName(dname, &head, &last);

break;

}

case 3: {

char dcity[CITY\_SIZE];

cout << "Введите город: ";

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

cin.sync();

cin.getline(dcity, CITY\_SIZE - 1, '\n');

deleteByCity(dcity, &head, &last);

break;

}

case 4: {

outputList(&head, &last);

break;

}

case 5: {

char fname[NAME\_SIZE];

cout << "Введите имя: ";

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

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

cin.sync();

findName(fname, &head);

break;

}

case 6: {

exit(0);

}

default: {

exit(1);

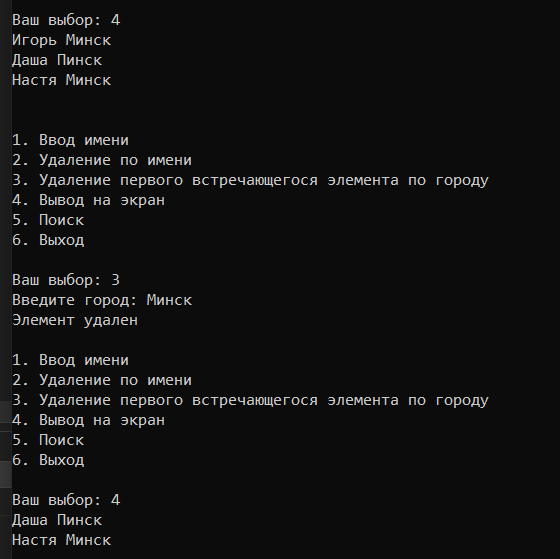
}

}

}

return 0;

}



**Доп. задания**

1



**OAP\_list.cpp**

#include "List.h"

void print(void\* b) {

Country\* a = (Country\*)b;

cout << a->name << " " << a->capital << " " << a->population << " " << a->area << " " << a->head\_lastname << endl;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Object L1 = Create();

unsigned short choice;

bool rc;

do {

cout << "\n0. Выход из приложения\n";

cout << "1. Ввод элементов списка с клавиатуры\n";

cout << "2. Вывод элементов списка в консольное окно\n";

cout << "3. Удаление заданного элемента\n";

cout << "4. Поиск информации\n";

cout << "5. Расчет элементов списка\n";

cout << "6. Удаление списка\n";

cout << "Введите вариант работы приложения: ";

cin >> choice;

switch (choice)

{

case 1: { //Ввод элементов структуры с клавиатуры

Country\* newCountry = new Country;

cout << "Название страны: ";

cin >> newCountry->name;

cout << "Столица страны: ";

cin >> newCountry->capital;

cout << "Численность населения страны: ";

cin >> newCountry->population;

cout << "Площадь страны: ";

cin >> newCountry->area;

cout << "Фамилия главы страны: ";

cin >> newCountry->head\_lastname;

L1.Insert(newCountry);

break;

}

case 2: { //Вывод элементов структуры в консольное окно

cout << "Список:" << endl;

L1.PrintList(print); // Выводим обновленный список

break;

}

case 3: { //Удаление заданного элемента

char deleteName[20];

cout << "Введите необходимую страну: ";

cin >> deleteName;

rc = L1.Delete(deleteName);

if (rc) {

cout << "Элемент удален" << endl;

}

else {

cout << "Элемент не найден" << endl;

}

break;

}

case 4: { //Поиск информации

char searchName[20];

cout << "Введите необходимую страну: ";

cin >> searchName;

Element\* find = L1.Search(searchName);

if (find != NULL) {

Country\* countryInfo = (Country\*)find->Data;

cout << "Информация о стране: " << endl;

print(countryInfo);

}

else {

cout << "Страна не найдена" << endl;

}

break;

}

case 5: { // Расчет элементов списка

cout << "Количество элементов в списке: " << L1.countList() << endl; // Исправлено название метода

break;

}

case 6: { // Удаление списка

rc = L1.deleteList(); // Удаляем элементы списка

if (rc) {

cout << "Список удален" << endl;

}

else {

cout << "Список пуст!" << endl;

}

break;

}

case 0: { //Выход из программы

exit(0);

break;

}

default: {

cout << "Введено некорректное значение. " << endl;

break;

}

}

} while (choice != 0);

return 0;

}

**List.cpp**

#include "List.h"

Object::~Object() {

while (Head != NULL) { // Пока есть элементы в списке

Element\* temp = Head;

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

delete temp; // Удаляем текущий элемент

}

}

Element\* Object::GetLast() {

Element\* e = Head;

Element\* rc = e;

while (e != NULL) { // Пока не достигнем конца списка

rc = e; // Запоминаем текущий элемент

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

}

return rc;

}

Element\* Object::Search(char\* data) {

Element\* rc = Head;

while ((rc != NULL) && (strcmp((char\*)rc->Data, data) != 0)) { // Пока не найдем нужный элемент

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

}

return rc;

}

void Object::Insert(Country \* data) {

if (Head == NULL) { // Если список пуст

Head = new Element(NULL, data, NULL); // Создаем новый элемент

}

else { // Если в списке уже есть элементы

Element\* last = GetLast(); // Получаем последний элемент

last->Next = new Element(last, data, NULL); // Добавляем новый элемент в конец списка

}

}

bool Object::Delete(Element\* e) {

if (e == NULL) return false; // Если переданный элемент NULL, возвращаем false

if (e->Next != NULL)

e->Next->Prev = e->Prev; // Обновляем указатели соседних элементов

if (e->Prev != NULL)

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

else

Head = e->Next; // Если удаляем первый элемент, обновляем указатель на голову списка

delete e; // Удаляем элемент

return true;

}

bool Object::Delete(char\* data) {

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

}

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

Element\* e = Head;

while (e != NULL) { // Пока есть элементы в списке

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

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

}

}

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

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

}

int Object::countList() {

int nElements = 0; //счетчик элементов в списке

Element\* e = Head;

while (e != NULL) { // Пока есть элементы в списке

nElements++; // Вызываем функцию для вывода данных элемента

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

}

return nElements;

}

bool Object::deleteList() {

if (Head == NULL) { //если список пуст

return false;

}

while (Head != NULL) { //если список не пуст

Element\* e = Head;

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

delete e; // Удаляем текущий элемент

}

return true;

}

// Определение функции Create для создания объекта Object

Object Create() {

return Object();

}

**List.h**

#pragma once

#include <iostream>

#include <Windows.h>

using namespace std;

struct Country {

char name[20];

char capital[20];

unsigned int population;

double area;

char head\_lastname[20];

Country\* next;

};

struct Element {

Element\* Prev;

Element\* Next;

void\* Data;

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

Prev = prev;

Data = data;

Next = next;

}

Element\* GetNext() {

return Next;

};

Element\* GetPrev() {

return Prev;

};

};

struct Object {

Element\* Head;

Object() {

Head = NULL;

};

~Object(); // Деструктор для освобождения памяти

Element\* GetFirst() {

return Head;

};

Element\* GetLast();

Element\* Search(char\* data);

void Insert(Country\* data);

bool Delete(char\* data);

bool Delete(Element\* data);

void PrintList(void(\*f)(void\*));

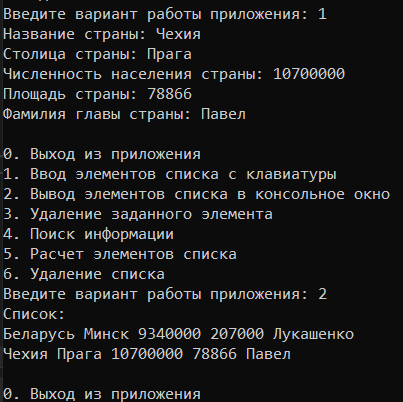
void PrintList(void(\*f)(void\*), Element\*);

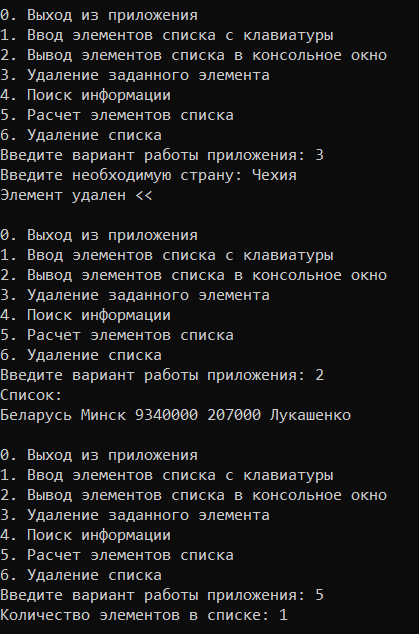
int countList();

bool deleteList();

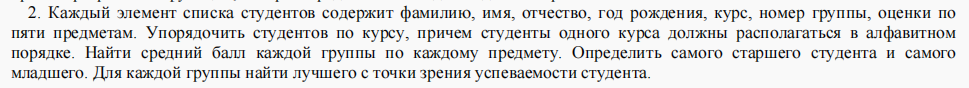
};

Object Create();





2.



#include <iostream>

#include <Windows.h>

#include <iomanip>

using namespace std;

const unsigned int NAME\_SIZE = 30;

const unsigned int CITY\_SIZE = 20;

//Структура, содержащая информацию о студентах

struct Student {

char last\_name[NAME\_SIZE]; //имя

char name[NAME\_SIZE]; //фамилия

char surname[NAME\_SIZE]; //отчество

int birth\_year ; //др

unsigned short course; //курс

unsigned short group; //группа

int grades[5];

Student\* next; //следующий человек в списке

Student\* prev; //предыдущий человек в списке

};

//Структура для хранения среднего балла по группам по каждому предмету

struct GroupStats {

short groupNumber; //Номер группы

short totalGrades[5]; //Сумма баллов по каждому предмету

short amount; // Количество студентов в группе

};

void clearBuf() {

cin.ignore(cin.rdbuf()->in\_avail()); //очистка буфера для корректного ввода

cin.clear();

}

bool isEmpty(Student\*\* phead) {

return \*phead == NULL;

}

//Добавление элемента в конец списка

void insertElement(Student\* e, Student\*\* phead, Student\*\* plast) {

Student\* p = \*plast; //Временный указатель на конец списка

if (\*plast == NULL) { //Если список пуст

e->next = NULL;

e->prev = NULL;

\*plast = e; //Устанавливаем начало списка на добавленный элемент

\*phead = e; //Устанавливаем конец списка на добавленный элемент

}

else { //Если в списке есть элементы

p->next = e; //Указываем, что следующий элемент в существующем спике - это добавленный элемент

e->next = NULL; //Следюущий нулевой

e->prev = p; //Предыдущий уже находившийся в списке

\*plast = e; //Конец список перенаправляем на добавленный элемент

}

}

// Создание элемента и ввод его значений с клавиатуры

Student\* setElement() {

Student\* temp = new Student();

if (!temp) {

cerr << "Ошибка выделения памяти памяти";

return NULL;

}

cout << "Введите фамилию: ";

clearBuf();

cin.getline(temp->last\_name, NAME\_SIZE, '\n');

cout << "Введите имя: ";

clearBuf();

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

cout << "Введите отчество: ";

clearBuf();

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

cout << "Введите год рождения: ";

clearBuf();

cin >> temp->birth\_year;

cout << "Введите курс: ";

clearBuf();

cin >> temp->course;

cout << "Введите группу: ";

clearBuf();

cin >> temp->group;

cout << "Введите оценки (5): ";

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

cin >> temp->grades[i];

}

cout << endl;

temp->next = NULL;

temp->prev = NULL;

return temp;

}

//Вывод списка на экран

void outputList(Student\*\* phead, Student\*\* plast) {

Student\* t = \*phead; //Указатель на начало списка

while (t) { //Пока указатель не NULL, выводим список

cout << t->last\_name << ' ' << t->name << ' ' << t->surname << ' ' << t->birth\_year << ' ' << t->course << ' ' << t->group << endl;

cout << "Оценки: " << t->grades[0] << ' ' << t->grades[1] << ' ' << t->grades[2] << ' ' << t->grades[3] << ' ' << t->grades[4] << endl;

t = t->next;

}

cout << "" << endl;

}

void sortList(Student\*\* phead, Student\*\* plast) {

if ((\*phead)->next == nullptr) {

// Список содержит только один элемент, сортировка не требуется

return;

}

Student\* current = (\*phead)->next; // Начинаем со второго элемента

while (current != nullptr) {

Student\* temp = current;

while (temp->prev != nullptr && (temp->course < temp->prev->course ||

(temp->course == temp->prev->course && strcmp(temp->last\_name, temp->prev->last\_name) < 0))) {

// Сравниваем курсы и фамилии и меняем местами студентов в списке

Student\* prevNode = temp->prev;

Student\* nextNode = temp->next;

temp->prev = prevNode->prev;

temp->next = prevNode;

prevNode->prev = temp;

prevNode->next = nextNode;

if (nextNode != nullptr) {

nextNode->prev = prevNode;

}

if (temp->prev != nullptr) {

temp->prev->next = temp;

}

else {

// Обновляем указатель на начало списка

\*phead = temp;

}

}

current = current->next;

}

\*plast = \*phead;

}

//Нахождение среднего балла в каждой группе по каждому предмету

void getAverage(Student\*\* phead, Student\*\* plast, unsigned int n) {

GroupStats\* groups = new GroupStats[n];

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

groups[i].groupNumber = i + 1;

groups[i].amount = 0;

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

groups[i].totalGrades[j] = 0;

}

}

Student\* temp = \*phead;

while (temp != nullptr) {

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

groups[groupIndex].amount++;

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

groups[groupIndex].totalGrades[i] += temp->grades[i];

}

temp = temp->next;

}

cout << "Средние баллы группы по каждому предмету: " << endl;

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

cout << "Группа " << groups[i].groupNumber << ": ";

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

double average = groups[i].amount > 0 ? static\_cast<double>(groups[i].totalGrades[j]) / groups[i].amount : 0.0;

cout << fixed << setprecision(2) << average << " ";

}

cout << endl;

}

delete[] groups;

}

void getMinAndMax(Student\*\* phead, Student\*\* plast) {

Student\* temp = \*phead;

Student\* youngest = temp;

Student\* oldest = temp;

while (temp != nullptr) {

if (temp->birth\_year > youngest->birth\_year) {

youngest = temp;

}

if (temp->birth\_year < oldest->birth\_year) {

oldest = temp;

}

temp = temp->next;

}

cout << "Самый младший студент: " << youngest->last\_name << " " << youngest->name << " " << youngest->surname << " (" << youngest->birth\_year << ")" << std::endl;

cout << "Самый старший студент: " << oldest->last\_name << " " << oldest->name << " " << oldest->surname << " (" << oldest->birth\_year << ")" << std::endl;

}

void getBestStudent(Student\*\* phead, Student\*\* plast) {

Student\* bestStudent = \*phead;

Student\* temp = \*phead;

while (temp != nullptr) {

int tempSum = 0;

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

tempSum += temp->grades[i];

}

int bestSum = 0;

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

bestSum += bestStudent->grades[i];

}

if (tempSum > bestSum) {

bestStudent = temp;

}

temp = temp->next;

}

cout << "Лучший студент:" << endl;

cout << bestStudent->last\_name << " " << bestStudent->name << " " << bestStudent->surname << endl;

cout << "Год рождения: " << bestStudent->birth\_year << endl;

cout << "Курс: " << bestStudent->course << endl;

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

cout << "Оценки: ";

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

cout << bestStudent->grades[i] << " ";

}

cout << endl;

}

//главная функция

int main() {

Student\* head = NULL; //создание указателя на начало списка

Student\* last = NULL; //создание указателя на конец списка

unsigned int n;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Сколько хотите ввести студентов? - ";

cin >> n;

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

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

}

if (isEmpty(&head)) {

cout << "Список пуст. Невозможно производить дальнейшие действия." << endl;

}

else {

cout << "Полученный список: " << endl;

sortList(&head, &last);

outputList(&head, &last);

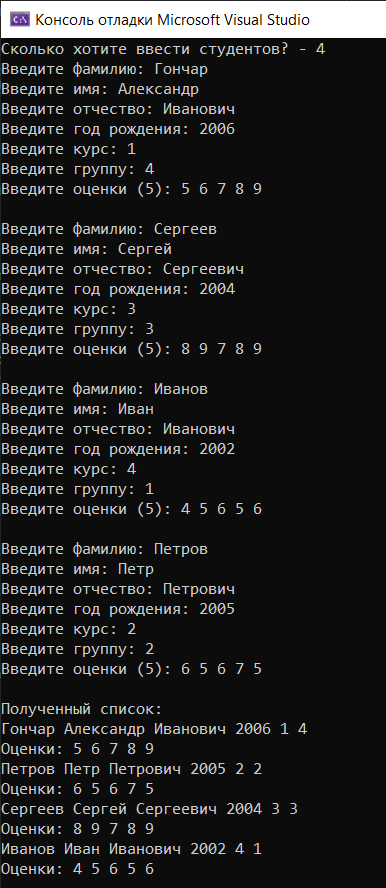
getAverage(&head, &last, n);

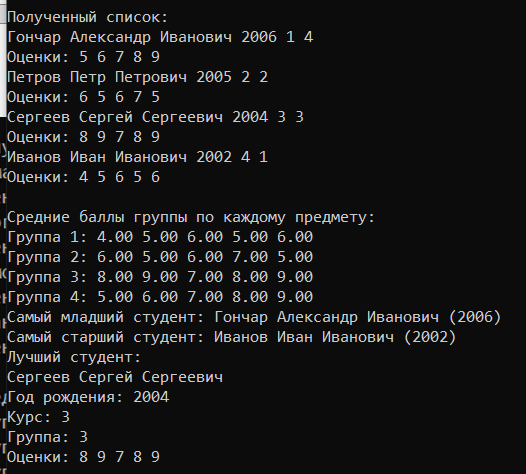
getMinAndMax(&head, &last);

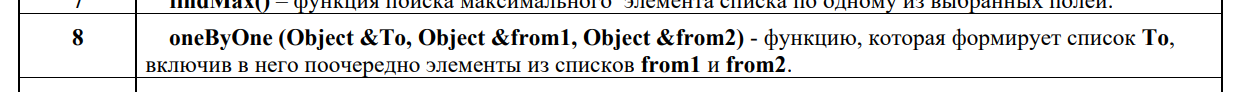
getBestStudent(&head, &last);

}

}







#include <iostream>

#include <fstream>

using namespace std;

const unsigned int NAME\_SIZE = 30;

const unsigned int CITY\_SIZE = 20;

// Структура для элемента списка

struct Address

{

char name[NAME\_SIZE];

char city[CITY\_SIZE];

Address\* next;

Address\* prev;

};

// Функция для добавления элемента в конец списка

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

{

Address\* p = \*plast;

if (\*plast == nullptr)

{

e->next = nullptr;

e->prev = nullptr;

\*plast = e;

\*phead = e;

}

else

{

p->next = e;

e->next = nullptr;

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" << endl;

return nullptr;

}

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 = nullptr;

temp->prev = nullptr;

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 = nullptr;

else

\*plast = nullptr;

}

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 записи в файл

void writeToFile(Address\*\* phead)

{

Address\* t = \*phead;

FILE\* fp;

errno\_t err = fopen\_s(&fp, "mlist", "wb");

if (err)

{

cerr << "Файл не открывается" << endl;

exit(1);

}

cout << "Сохранение в файл" << endl;

while (t)

{

fwrite(t, sizeof(Address), 1, fp);

t = t->next;

}

fclose(fp);

}

// Функция для чтения из файла

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

{

Address\* t;

FILE\* fp;

errno\_t err = fopen\_s(&fp, "mlist", "rb");

if (err)

{

cerr << "Файл не открывается" << endl;

exit(1);

}

while (\*phead)

{

\*plast = (\*phead)->next;

delete\* phead;

\*phead = \*plast;

}

\*phead = \*plast = nullptr;

cout << "Загрузка из файла" << endl;

while (!feof(fp))

{

t = new Address();

if (!t)

{

cerr << "Ошибка выделения памяти" << endl;

return;

}

if (1 != fread(t, sizeof(Address), 1, fp))

break;

insert(t, phead, plast);

}

fclose(fp);

}

// Функция для формирования списка To из списков from1 и from2

void oneByOne(Address\*\* To, Address\* from1, Address\* from2)

{

while (from1 != nullptr && from2 != nullptr)

{

// Создаем новый элемент для списка To

Address\* temp = new Address();

if (temp == nullptr)

{

cerr << "Ошибка выделения памяти" << endl;

exit(1);

}

// Копируем данные из from1

strcpy\_s(temp->name, from1->name);

strcpy\_s(temp->city, from1->city);

// Добавляем элемент в конец списка To

insert(temp, To, To);

// Переходим к следующему элементу from1

from1 = from1->next;

// Если from2 не пустой, продолжаем добавление элементов в To

if (from2 != nullptr)

{

// Создаем новый элемент для списка To

temp = new Address();

if (temp == nullptr)

{

cerr << "Ошибка выделения памяти" << endl;

exit(1);

}

// Копируем данные из from2

strcpy\_s(temp->name, from2->name);

strcpy\_s(temp->city, from2->city);

// Добавляем элемент в конец списка To

insert(temp, To, To);

// Переходим к следующему элементу from2

from2 = from2->next;

}

}

// Если from1 или from2 еще содержат элементы, добавляем оставшиеся элементы в To

while (from1 != nullptr)

{

// Создаем новый элемент для списка To

Address\* temp = new Address();

if (temp == nullptr)

{

cerr << "Ошибка выделения памяти" << endl;

exit(1);

}

// Копируем данные из from1

strcpy\_s(temp->name, from1->name);

strcpy\_s(temp->city, from1->city);

// Добавляем элемент в конец списка To

insert(temp, To, To);

// Переходим к следующему элементу from1

from1 = from1->next;

}

while (from2 != nullptr)

{

// Создаем новый элемент для списка To

Address\* temp = new Address();

if (temp == nullptr)

{

cerr << "Ошибка выделения памяти" << endl;

exit(1);

}

// Копируем данные из from2

strcpy\_s(temp->name, from2->name);

strcpy\_s(temp->city, from2->city);

// Добавляем элемент в конец списка To

insert(temp, To, To);

// Переходим к следующему элементу from2

from2 = from2->next;

}

}

// Функция для вывода меню и выбора опции

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. Запись в файл" << endl;

cout << "6. Чтение из файла" << endl;

cout << "7. Новый список" << endl;

cout << "8. Выход" << endl;

cout << endl;

do

{

cout << "Ваш выбор: ";

cin.sync();

gets\_s(s);

cout << endl;

c = atoi(s);

} while (c < 1 || c > 8);

return c;

}

int main(void)

{

Address\* head = nullptr;

Address\* last = nullptr;

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 << "Запись в файл " << endl;

writeToFile(&head); // Записываем данные в файл перед выходом

exit(0);

case 6:

cout << "Чтение из файла" << endl;

readFromFile(&head, &last); // Считываем данные из файла

break;

case 7:

{

// Создаем новый список для результатов

Address\* newListHead = nullptr;

Address\* newListLast = nullptr;

// Формируем новый список из списков from1 и from2

oneByOne(&newListHead, head, last);

// Выводим новый список

cout << "Новый разделенный список:" << endl;

outputList(&newListHead, &newListLast);

// Очищаем память нового списка

while (newListHead != nullptr)

{

Address\* temp = newListHead;

newListHead = newListHead->next;

delete temp;

}

}

break;

default:

exit(1);

}

}

return 0;

}

