Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа №9

По дисциплине «Основы алгоритмизации и программирования»

На тему « Двусвязные списки »

Выполнил:

Студент 1 курса 8 группы

Лужецкий Владислав Константинович

Преподаватель: асс. Андронова М.В.

2024, Минск

Основной вариант 6:



//Не вижу смысла особо комментировать код из примера. Сделал комментарии конкретно по своей функции findMin

#include <iostream>

#include <fstream>

#include <Windows.h>

using namespace std;

const unsigned int NAME\_SIZE = 30;

const unsigned int CITY\_SIZE = 20;

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. Запись в файла" << endl;

cout << "6. Чтение с файла" << endl;

cout << "7. Поиск минимального(имени)" << 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 > 10);

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;

}

}

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

void writeToFile(Address\*\* phead) //Запись в файл

{

struct 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(struct Address), 1, fp);

t = t->next;

}

fclose(fp);

}

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

void readFromFile(Address\*\* phead, Address\*\* plast) //Считывание из файла

{

struct 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 = NULL;

cout << "Загрузка из файла" << endl;

while (!feof(fp))

{

t = new Address();

if (!t)

{

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

return;

}

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

insert(t, phead, plast);

}

fclose(fp);

}

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

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

{

Address\* t = \*phead;

char min[NAME\_SIZE];

if (t == NULL)

{

cout << "Список пуст\n";

return;

}

strncpy\_s(min, t->name, NAME\_SIZE); // Копируем первое имя в min

min[NAME\_SIZE - 1] = '\0'; // Убеждаемся, что строка заканчивается нулевым символом

t = t->next;

while (t != NULL)

{

if (strlen(t->name) < strlen(min))

{

strncpy\_s(min, t->name, NAME\_SIZE); // Копируем текущее имя в min, если оно короче

min[NAME\_SIZE - 1] = '\0'; // Убеждаемся, что строка заканчивается нулевым символом

}

t = t->next;

}

cout << "Минимальное имя: " << min << '\n';

}

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

int main(void)

{

Address\* head = NULL;

Address\* last = NULL;

setlocale(LC\_CTYPE, "Rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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(); // Игнорируем символы новой строки

cin.clear(); // Сбрасываем флаги ошибки ввода

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(); // Игнорируем символы новой строки

cin.clear(); // Сбрасываем флаги ошибки ввода

find(fname, &head);

}

break;

case 5:

writeToFile(&head);

break;

case 6:

readFromFile(&head, &last);

break;

case 7:

findMin(&head, &last);

break;

case 0:

exit(0);

default:

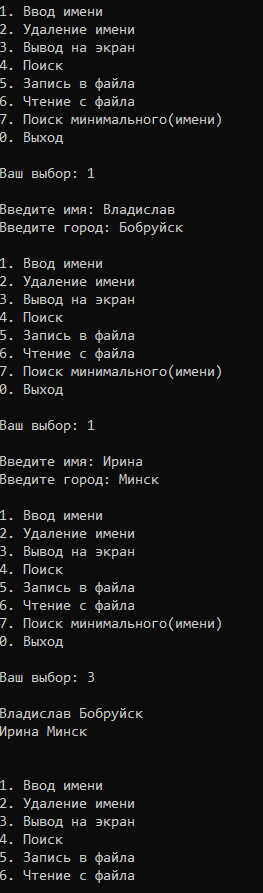
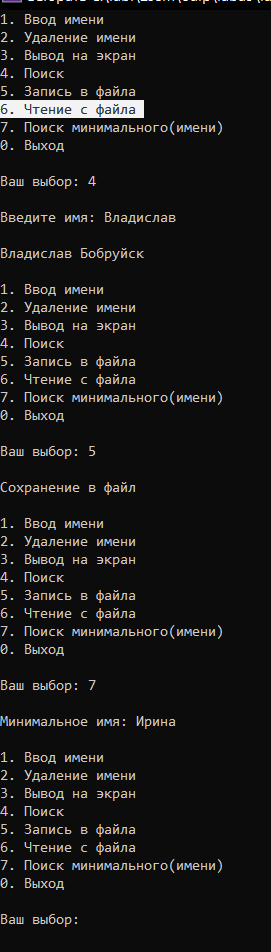
exit(1);

}

}

return 0;

}



**Доп 1:**

//Я просто скину код функции, которая должна быть добавлена, потому что переписывать одно и тоже несколько раз так себе идея



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

{

Address\* t = \*phead;

char max[NAME\_SIZE];

if (t == NULL)

{

cout << "Список пуст\n";

return;

}

strncpy\_s(max, t->name, NAME\_SIZE); // Копируем первое имя в max

max[NAME\_SIZE - 1] = '\0'; // Убеждаемся, что строка заканчивается нулевым символом

t = t->next;

while (t != NULL)

{

if (strlen(t->name) > strlen(max))

{

strncpy\_s(max, t->name, NAME\_SIZE); // Копируем текущее имя в max, если оно длиннее

max[NAME\_SIZE - 1] = '\0'; // Убеждаемся, что строка заканчивается нулевым символом

}

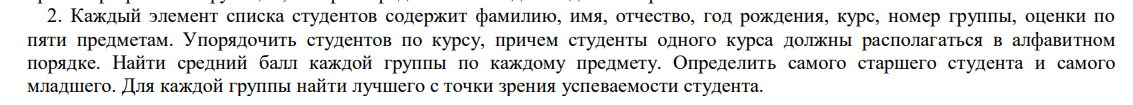
t = t->next;

}

cout << "Максимальное имя: " << max << '\n';

}

**Доп 2:**



#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <Windows.h>

#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 элемента в списке

Student\* setElement(); // Ввод данных о студенте

void outputList(Student\*\* phead, Student\*\* plast); // Вывод списка студентов

void studentOld(Student\*\* phead, Student\*\* plast); // Поиск студента с наибольшим возрастом

void studentYoung(Student\*\* phead, Student\*\* plast); // Поиск студента с наименьшим возрастом

void outputFirstCourse(Student\*\* phead, Student\*\* plast); // Вывод студентов первого курса

void outputSecondCourse(Student\*\* phead, Student\*\* plast); // Вывод студентов второго курса

void outputThirdCourse(Student\*\* phead, Student\*\* plast); // Вывод студентов третьего курса

void outputFourthCourse(Student\*\* phead, Student\*\* plast); // Вывод студентов четвёртого курса

void sortStudentsByCourse(Student\*\* phead); // Сортировка студентов по курсу

void calculateAverageScores(Student\*\* phead); // Расчёт среднего значения оценок студентов

int main() {

setlocale(LC\_ALL, "rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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) //пушим в конец списка

{

if (\*plast == nullptr) {

\*plast = e;

\*phead = e;

}

else {

(\*plast)->next = e;

e->prev = \*plast;

\*plast = e;

}

}

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

{

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 << "Англ.яз: ";

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 << "Англ.яз: " << t->english << endl;

cout << "ОАиП:" << t->OAIP << endl;

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

t = t->next;

}

}

void studentOld(Student\*\* phead, Student\*\* plast) // поиск 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) // поиск 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) // Сортировка 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)

{

// свапаем, если курс идёт не по порядку или фамилия студента должна стоять первее по алфавиту(одинаковый курс)

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) // вывод первого курса

{

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 << "Англ.яз:" << 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) // вывод второго курса

{

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 << "Англ.яз:" << 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) //вывод третьего курса

{

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 << "Англ.яз:" << 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) //вывод четвёртого курса

{

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 << "Англ.яз:" << 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 спискуи суммируем оценки по всем предметам для всех групп

Student\* current = \*phead;

while (current != nullptr)

{

int groupIndex = current->group - 1;

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;

}

for (int i = 0; i < MAX\_GROUPS; i++) // средний балл по каждому предмету для каждой группы

{

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];

// Вывод результата 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;

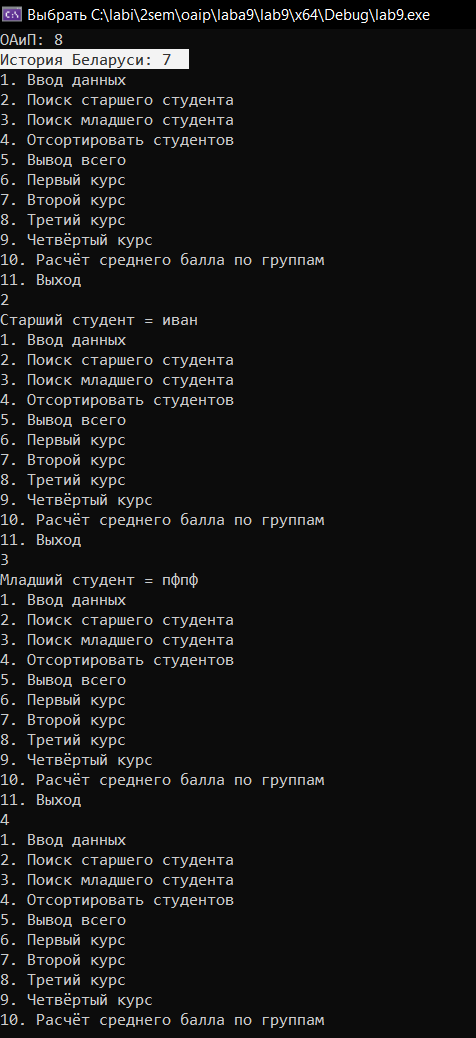
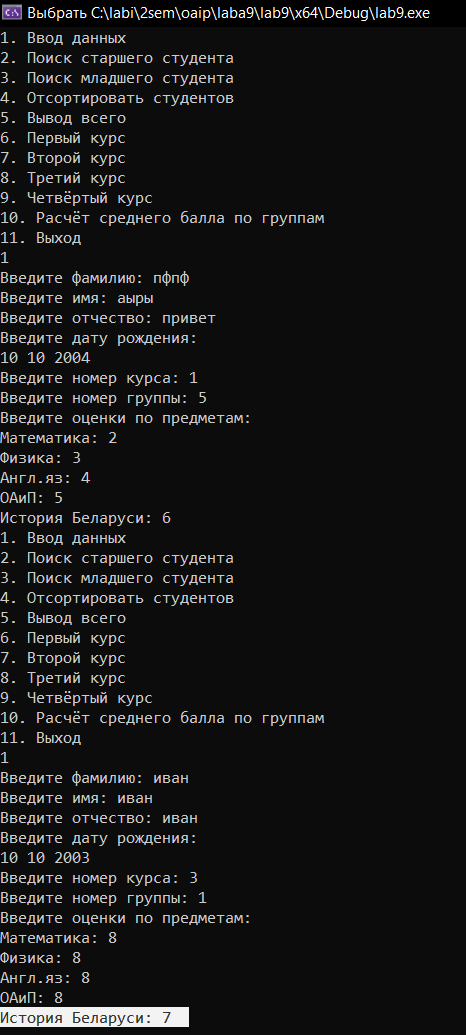
cout << "Средний балл по ОАиПу: " << averageOAIP << endl;

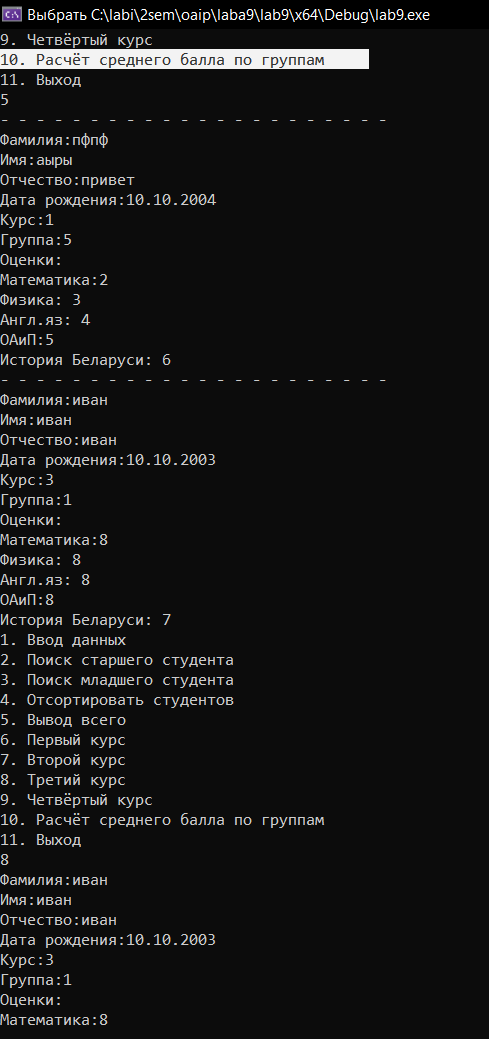
cout << "Средний балл по Истории Беларуси: " << averageHistory << endl;

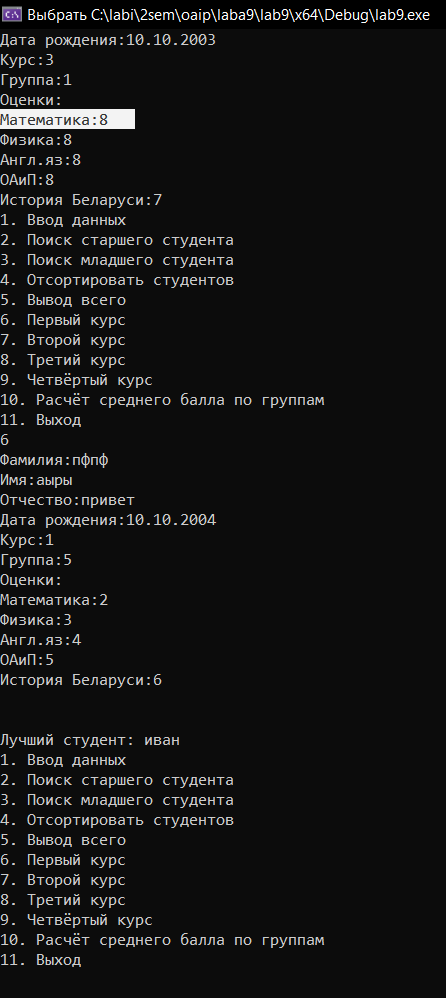
}

}

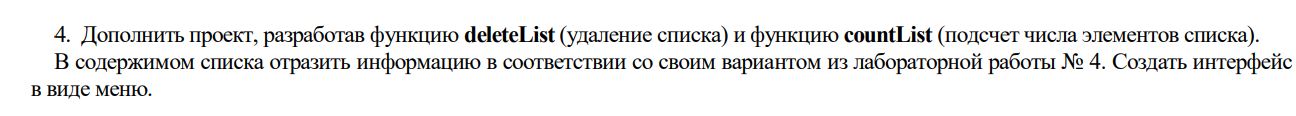
}

****





**Доп 3:**

** **

**Main.cpp**

#include "List.h"

#include <Windows.h>

using namespace std;

void print(char\* nameProduct, char\* price, char\* quantity, int percent)

{

cout << nameProduct << " " << price << " " << quantity << " " << percent << '\n';

}

int main()

{

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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 nameProduct[50];

char price[50];

char quantity[50];

int percent;

cout << "Название продукта: ";

cin >> nameProduct;

cout << "Цена: ";

cin >> price;

cout << "Количество: ";

cin >> quantity;

cout << "Процент надбавки: ";

cin >> percent;

L1.Insert(nameProduct, price, quantity, percent);

break;

}

case 2:

{

L1.PrintList(print);

break;

}

case 3:

{

char price[50];

cout << "Введите Цену: ";

cin >> price;

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

if (e == NULL)

{

cout << "Элемент не найден" << endl;

}

else

{

cout << "Найден элемент= " << e->NameProduct << " " << e->Price << " " << e->Quantity << " " << e->Percent << '\n';

}

break;

}

case 4:

{

char productname[50];

cout << "Введите название товара для удаления: ";

cin >> productname;

bool rc = L1.Delete(productname);

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\* nameproduct, char\* price, char\* quantity, int percent) // Вставка в начало

{

bool rc = 0;

if (Head == NULL) // если список пуст

{

Head = new Element(NULL, nameproduct, price, quantity, percent, Head);

rc = true;

}

else

{

Head = (Head->Prev = new Element(NULL, nameproduct, price, quantity, percent, Head));

rc = true;

}

return rc;

}

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

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

{

Element\* rc = Head;

while ((rc != NULL) && (strcmp(rc->Price, price)))

rc = rc->Next;

return rc;

}

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

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

{

Element\* e = Head;

while (e != NULL)

{

f(e->NameProduct, e->Price, e->Quantity, e->Percent); // 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\*, int), Element\* e)

{

f(e->NameProduct, e->Price, e->Quantity, e->Percent);

}

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

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

#pragma once

#include <iostream>

struct Element // Элемент списка

{

Element\* Prev; // указатель на предыдущий элемент

Element\* Next; // указатель на следующий элемент

char NameProduct[50];

char Price[50];

char Quantity[50];

int Percent;

Element(Element\* prev, char\* nameproduct, char\* price, char\* quantity, int percent, Element\* next) // конструктор для создания элемента

{

Prev = prev;

strcpy\_s(NameProduct, nameproduct);

strcpy\_s(Price, price);

strcpy\_s(Quantity, quantity);

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\*, int); // добавить элемент в начало

bool InsertEnd(char\*, char\*, char\*, int); // 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\*, int));

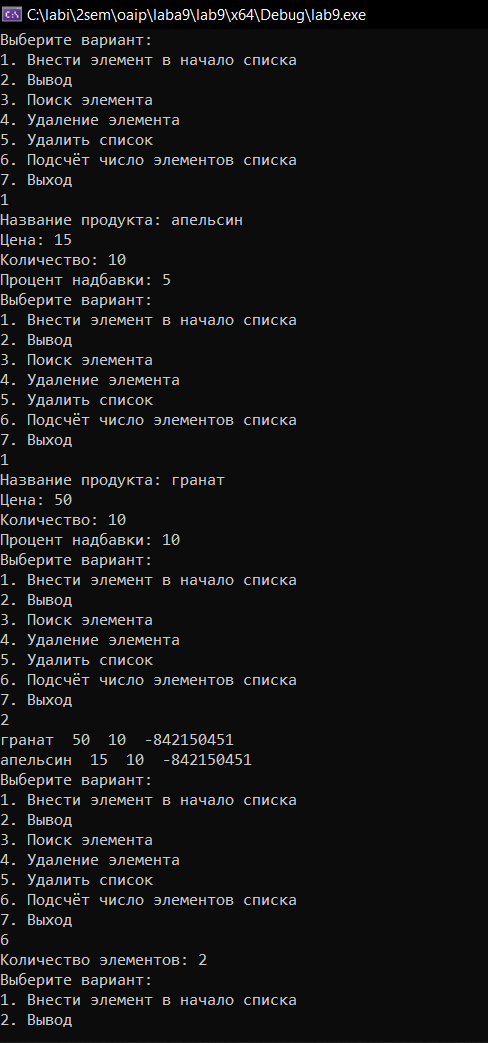
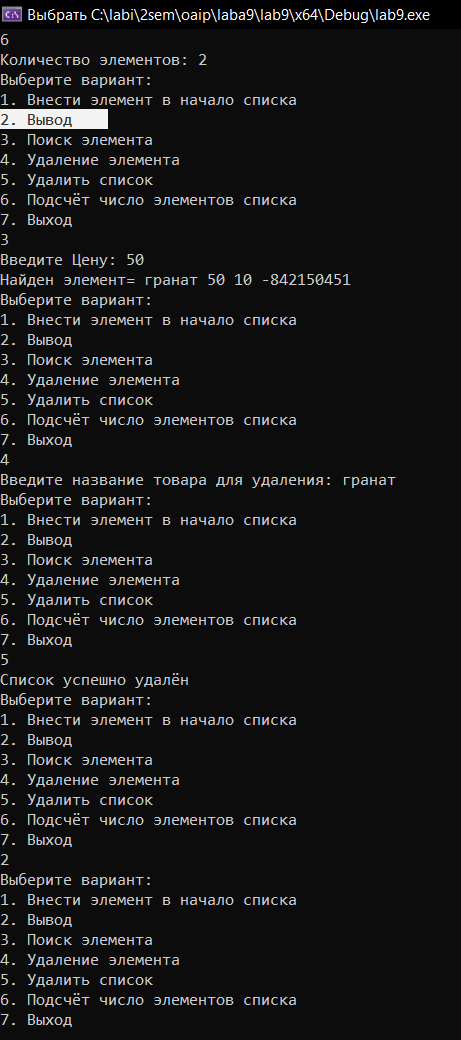
void PrintList(void(\*f)(char\*, char\*, char\*, int), Element\*);

int CountList();

bool DeleteDouble();

};

Object Create();

****