**Лабораторная работа № 9. Двусвязные списки**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. В правой части записана программа, которая формирует *двусвязный список* для хранения информации, содержащей адреса людей (имя и город).  Написать комментарии к программе. | **#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];**  **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. 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 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: exit(0);**  **default: exit(1);**  **}**  **}**  **return 0;**  **}** |  | |
| 2. В правой части приведены функции, осуществляющие запись информации в файл и считывание из файла.  Добавить эти функции в программу п.1 и внести соответствующие изменения в меню. | **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);**  **}** |  | |
| Вариант 10  **addXBegin (int x)** – функция добавления элемента **x** в начало списка  Вариант 15  addLEnd – функция добавления в конец списка всех элементов некоторого списка L. | #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];  char city[CITY\_SIZE];  Address\* next;  Address\* prev;  };  int menu(void)  {  char s[80]; int c;  cout << endl;  cout << "0. Выход" << endl;  cout << "1. Ввод в конец" << endl;  cout << "2. Удаление имени" << endl;  cout << "3. Вывод на экран" << endl;  cout << "4. Поиск" << endl;  cout << "5. Ввод в начало" << endl;  cout << "6. Добавление другого списка в конец" << endl;  cout << "7. Запись в файл" << endl;  cout << "8. Считывание из файла" << endl;  cout << endl;  do  {  cout << "Ваш выбор: ";  cin.sync();  gets\_s(s);  cout << endl;  c = atoi(s);  } while (c < 0 || c > 6);  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;  }  }  //-----------------------------------------------------------  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 addXBegin(Address\* e, Address\*\* phead, Address\*\* plast) {//Добавление в начало списка  Address\* p = \*phead;  if (p == NULL) {  e->next = NULL;  e->prev = NULL;  \*plast = e;  \*phead = e;  return;  }  else {  p->prev = e;  e->prev = NULL;  e->next = p;  \*phead = e;  return;  }  }  //-----------------------------------------------------------  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 addLEnd(Address\*\* phead, Address\*\* plast) {  int n;  cout << "Введите количество элементов списка L: ";  cin >> n;  cin.ignore(cin.rdbuf()->in\_avail());  Address\* headL = NULL, \* lastL = NULL;  for (int i = 0; i < n; i++) {    insert(setElement(), &headL, &lastL);  }  Address\* curr = headL;  while (curr!= NULL) {    Address\* copy = new Address();  strcpy\_s(copy->name, curr->name);  strcpy\_s(copy->city, curr->city);  copy->next = NULL;  copy->prev = NULL;  insert(copy, phead, plast);  curr= curr->next;  }    curr = headL;  while (curr != NULL) {  Address\* temp = curr;  curr = curr->next;  delete 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 main(void)  {  Address\* head = NULL;  Address\* last = NULL;  setlocale(LC\_CTYPE, "Rus");  while (true)  {  switch (menu())  {  case 0: { exit(0); break; }  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: {  addXBegin(setElement(), &head, &last);  break;  }  case 6: {  addLEnd(&head, &last);  break;  }  case 7: {  writeToFile(&head);  break;  }  case 8: {  readFromFile(&head, &last);  break;  }  default: exit(1);  }  }  return 0;  }  //-----------------------------------------------------------  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);  } | |  |
| 4. Дополнить проект, разработав функцию deleteList (удаление списка) и функцию countList (подсчет числа элементов списка).  В содержимом списка отразить информацию в соответствии со своим вариантом из лабораторной работы № 4. Создать интерфейс в виде меню. | List.h  #pragma once  #include <iostream>  #include <cstring>  using namespace std;  struct Element {  Element\* Prev;  Element\* Next;  void\* Data;  Element(Element\* prev, void\* data, Element\* next)  : Prev(prev), Data(data), Next(next) {  }  };  struct Car {  char brand[20];  char color[15];  char vin[17];  char prod\_date[11];  char body\_type[10];  char last\_check[11];  char owner[30];  };  struct Object {  Element\* Head;  Object() : Head(nullptr) {}  Element\* GetFirst() { return Head; }  Element\* GetLast();  Element\* Search(void\* data);  bool Insert(void\* data);  bool InsertEnd(void\* data);  bool Delete(Element\* e);  bool Delete(void\* data);  bool DeleteList();  void PrintList(void(\*f)(void\*));  void PrintList(void(\*f)(void\*), Element\*);  int CountList();  Element\* SearchByOwner(const char\* owner);  Element\* SearchByVIN(const char\* vin);  };  Object Create();  void printCar(void\* b);  List.cpp  #include "List.h"  Object Create() {  return Object();  }  void printCar(void\* b) {  Car\* a = (Car\*)b;  cout << "Марка: " << a->brand << "\nЦвет: " << a->color  << "\nVIN: " << a->vin << "\nДата выпуска: " << a->prod\_date  << "\nТип кузова: " << a->body\_type << "\nТехосмотр: "  << a->last\_check << "\nВладелец: " << a->owner << "\n\n";  }  bool Object::Insert(void\* data) {  Head = new Element(nullptr, data, Head);  if (Head->Next) Head->Next->Prev = Head;  return true;  }  bool Object::InsertEnd(void\* data) {  Element\* last = GetLast();  Element\* newElem = new Element(last, data, nullptr);  if (last) last->Next = newElem;  else Head = newElem;  return true;  }  Element\* Object::Search(void\* data) {  Element\* e = Head;  while (e && e->Data != data) e = e->Next;  return e;  }  bool Object::Delete(Element\* e) {  if (!e) return false;  if (e->Prev) e->Prev->Next = e->Next;  else Head = e->Next;  if (e->Next) e->Next->Prev = e->Prev;  delete e;  return true;  }  bool Object::Delete(void\* data) {  return Delete(Search(data));  }  bool Object::DeleteList() {  while (Head) Delete(Head);  return true;  }  Element\* Object::GetLast() {  Element\* e = Head;  while (e && e->Next) e = e->Next;  return e;  }  void Object::PrintList(void(\*f)(void\*)) {  Element\* e = Head;  while (e) {  f(e->Data);  e = e->Next;  }  }  void Object::PrintList(void(\*f)(void\*), Element\* start) {  Element\* e = start;  while (e) {  f(e->Data);  e = e->Next;  }  }  int Object::CountList() {  int count = 0;  Element\* e = Head;  while (e) {  count++;  e = e->Next;  }  return count;  }  Element\* Object::SearchByOwner(const char\* owner) {  Element\* e = Head;  while (e) {  Car\* car = (Car\*)e->Data;  if (strcmp(car->owner, owner) == 0) return e;  e = e->Next;  }  return nullptr;  }  Element\* Object::SearchByVIN(const char\* vin) {  Element\* e = Head;  while (e) {  Car\* car = (Car\*)e->Data;  if (strcmp(car->vin, vin) == 0) return e;  e = e->Next;  }  return nullptr;  }  Main.cpp  #include "List.h"  void showMenu(Object& list) {  int choice;  char input[30];  Element\* startElem = nullptr;  do {  cout << "\n=== Меню ===\n"  << "1. Добавить в начало\n"  << "2. Добавить в конец\n"  << "3. Удалить по VIN\n"  << "4. Показать все\n"  << "5. Показать с элемента\n"  << "6. Найти по владельцу\n"  << "7. Количество автомобилей\n"  << "8. Очистить список\n"  << "9. Выход\n"  << "Выберите действие: ";  cin >> choice;  cin.ignore();  switch (choice) {  case 1:  case 2: {  Car\* newCar = new Car();  cout << "Марка: "; cin.getline(newCar->brand, 20);  cout << "Цвет: "; cin.getline(newCar->color, 15);  cout << "VIN: "; cin.getline(newCar->vin, 17);  cout << "Дата выпуска: "; cin.getline(newCar->prod\_date, 11);  cout << "Тип кузова: "; cin.getline(newCar->body\_type, 10);  cout << "Техосмотр: "; cin.getline(newCar->last\_check, 11);  cout << "Владелец: "; cin.getline(newCar->owner, 30);  (choice == 1) ? list.Insert(newCar) : list.InsertEnd(newCar);  break;  }  case 3: {  cout << "Введите VIN: ";  cin.getline(input, 17);  Element\* e = list.SearchByVIN(input);  if (e) {  list.Delete(e);  cout << "Удалено.\n";  }  else cout << "Не найдено!\n";  break;  }  case 4:  list.PrintList(printCar);  break;  case 5: {  cout << "Введите VIN начального элемента: ";  cin.getline(input, 17);  startElem = list.SearchByVIN(input);  if (startElem) list.PrintList(printCar, startElem);  else cout << "Элемент не найден!\n";  break;  }  case 6: {  cout << "Введите владельца: ";  cin.getline(input, 30);  Element\* e = list.SearchByOwner(input);  if (e) printCar(e->Data);  else cout << "Не найдено!\n";  break;  }  case 7:  cout << "Количество: " << list.CountList() << "\n";  break;  case 8:  list.DeleteList();  cout << "Список очищен.\n";  break;  case 9:  list.DeleteList();  cout << "Выход.\n";  break;  default:  cout << "Ошибка выбора!\n";  }  } while (choice != 9);  }  int main() {  setlocale(LC\_ALL, "Russian");  Object carList = Create();  showMenu(carList);  return 0;  } | | |
| 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. | #include <iostream>  #include <iomanip>  #include <limits>  #include <cstring>  using namespace std;  const int MAX = 30;  const int SUBJ = 5;  struct Student {  char lastName[MAX];  char firstName[MAX];  char patronymic[MAX];  int birth;  int course;  int group;  int grades[SUBJ] = { 0 };  Student\* next;  Student\* prev;  };  void addStudent(Student\*& head, Student\*& tail, Student\* student) {  if (!head) {  head = tail = student;  }  else {  tail->next = student;  student->prev = tail;  tail = student;  }  student->next = nullptr;  }  int inputInt() {  int value;  while (!(cin >> value)) {  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  cout << "Ошибка ввода. Пожалуйста, введите целое число: ";  }  return value;  }  void inputStudent(Student\*& head, Student\*& tail) {  Student\* newStudent = new Student();  cout << "Введите фамилию: ";  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  cin.getline(newStudent->lastName, MAX);  cout << "Введите имя: ";  cin.getline(newStudent->firstName, MAX);  cout << "Введите отчество: ";  cin.getline(newStudent->patronymic, MAX);  cout << "Введите год рождения: ";  newStudent->birth = inputInt();  cout << "Введите курс: ";  newStudent->course = inputInt();  cout << "Введите группу: ";  newStudent->group = inputInt();  cout << "Введите оценки по пяти предметам:\n";  for (int i = 0; i < SUBJ; i++) {  cout << "Предмет " << i + 1 << ": ";  newStudent->grades[i] = inputInt();  }  addStudent(head, tail, newStudent);  }  void sortStudents(Student\*& head, Student\*& tail) {  if (!head || !head->next) return;  bool swapped;  do {  swapped = false;  Student\* current = head;  Student\* prev = nullptr;  Student\* nextNode = nullptr;  while (current && current->next) {  nextNode = current->next;  bool needSwap = false;  if (current->course > nextNode->course) {  needSwap = true;  }  else if (current->course == nextNode->course) {  if (strcmp(current->lastName, nextNode->lastName) > 0) {  needSwap = true;  }  }  if (needSwap) {  if (current->prev) current->prev->next = nextNode;  nextNode->prev = current->prev;  current->prev = nextNode;  current->next = nextNode->next;  if (nextNode->next) nextNode->next->prev = current;  nextNode->next = current;  if (current == head) head = nextNode;  if (nextNode == tail) tail = current;  swapped = true;  prev = nextNode;  }  else {  prev = current;  current = current->next;  }  }  } while (swapped);  }  void calculateGroupAverages(Student\* head) {  struct GroupData {  int course;  int group;  double sums[SUBJ] = { 0 };  int count = 0;  };  GroupData groups[100];  int groupCount = 0;  for (Student\* curr = head; curr; curr = curr->next) {  bool found = false;  for (int i = 0; i < groupCount; i++) {  if (groups[i].course == curr->course && groups[i].group == curr->group) {  for (int j = 0; j < SUBJ; j++) {  groups[i].sums[j] += curr->grades[j];  }  groups[i].count++;  found = true;  break;  }  }  if (!found) {  groups[groupCount].course = curr->course;  groups[groupCount].group = curr->group;  for (int j = 0; j < SUBJ; j++) {  groups[groupCount].sums[j] = curr->grades[j];  }  groups[groupCount].count = 1;  groupCount++;  }  }  cout << "\nСредние баллы по группам:\n";  for (int i = 0; i < groupCount; i++) {  cout << "Курс " << groups[i].course  << ", Группа " << groups[i].group << ": ";  for (int j = 0; j < SUBJ; j++) {  cout << fixed << setprecision(2)  << groups[i].sums[j] / groups[i].count << " ";  }  cout << endl;  }  }  void findAgeExtremes(Student\* head) {  if (!head) return;  Student\* oldest = head;  Student\* youngest = head;  for (Student\* curr = head; curr; curr = curr->next) {  if (curr->birth < oldest->birth) oldest = curr;  if (curr->birth > youngest->birth) youngest = curr;  }  cout << "\nСамый старший: " << oldest->lastName  << " " << oldest->firstName << " (" << oldest->birth << ")\n";  cout << "Самый младший: " << youngest->lastName  << " " << youngest->firstName << " (" << youngest->birth << ")\n";  }  void findTopStudents(Student\* head) {  struct GroupBest {  int group;  Student\* best;  double avg;  };  GroupBest bests[100];  int groupCount = 0;  for (Student\* curr = head; curr; curr = curr->next) {  double sum = 0;  for (int grade : curr->grades) sum += grade;  double avg = sum / SUBJ;  bool found = false;  for (int i = 0; i < groupCount; i++) {  if (bests[i].group == curr->group) {  if (avg > bests[i].avg) {  bests[i].best = curr;  bests[i].avg = avg;  }  found = true;  break;  }  }  if (!found) {  bests[groupCount] = { curr->group, curr, avg };  groupCount++;  }  }  cout << "\nЛучшие студенты по группам:\n";  for (int i = 0; i < groupCount; i++) {  cout << "Группа " << bests[i].group << ": "  << bests[i].best->lastName << " "  << bests[i].best->firstName << " (Ср. балл: "  << fixed << setprecision(2) << bests[i].avg << ")\n";  }  }  void printStudents(Student\* head) {  cout << "\nСписок студентов:\n";  for (Student\* curr = head; curr; curr = curr->next) {  cout << curr->lastName << " " << curr->firstName  << " (Курс: " << curr->course  << ", Группа: " << curr->group << ")\n";  }  }  void deleteStudents(Student\*& head) {  while (head) {  Student\* temp = head;  head = head->next;  delete temp;  }  }  int main() {  setlocale(0, "rus");  Student\* head = nullptr;  Student\* tail = nullptr;  int count;  cout << "Введите количество студентов: ";  count = inputInt();  for (int i = 0; i < count; i++) {  cout << "\nСтудент #" << i + 1 << ":\n";  inputStudent(head, tail);  }  sortStudents(head, tail);  printStudents(head);  calculateGroupAverages(head);  findAgeExtremes(head);  findTopStudents(head);  deleteStudents(head);  return 0;  } | | |