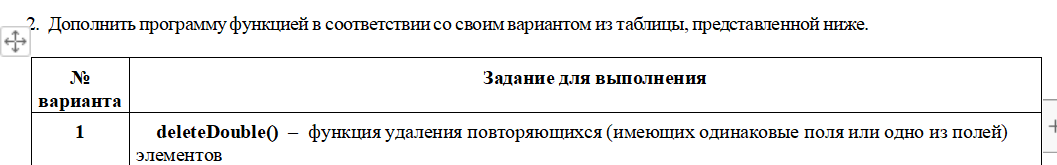
Банкузов Михаил 7 группа Лабораторная работа №9 Вариант 1



#include <iostream>

#include <fstream>

#pragma warning (disable : 4996)

using namespace std;

// Определяем размеры

const unsigned int NAME\_SIZE = 30;

const unsigned int CITY\_SIZE = 20;

// Структура

struct Address {

char name[NAME\_SIZE];//размер имени

char city[CITY\_SIZE];//размер города

Address\* next = nullptr;

Address\* prev = nullptr;

};

//меню

int menu(void) {

// Объявляем массив символов для хранения ввода пользователя

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. Считывание из файла" << endl;

cout << "7. Удалить дубликаты" << endl;

cout << endl;

// Повторяем цикл, пока пользователь не введет корректный выбор

do {

cout << "Ваш выбор: ";

// Очищаем буфер ввода

cin.sync();

// Считываем строку из стандартного потока ввода

cin.getline(s, 80);

cout << endl;

// Преобразуем строку в целое число

c = atoi(s);

// Проверяем, что выбор находится в диапазоне от 1 до 7

} while (c < 1 || c > 7);

// Возвращаем выбор пользователя

return c;

}

// Функция для вставки элемента в двусвязный список

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

// Если список пуст, то делаем элемент первым и последним

if (\*plast == nullptr) {

\*plast = e;

\*phead = e;

}

// Иначе, добавляем элемент в конец списка и обновляем указатели

else {

e->prev = \*plast;

(\*plast)->next = e;

\*plast = e;

}

}

// Функция для создания и заполнения элемента списка

Address\* setElement() {

// Выделяем память под элемент

Address\* temp = new Address();

// Проверяем, что память выделена успешно

if (!temp) {

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

return nullptr;

}

// Запрашиваем имя и город от пользователя и записываем их в элемент

cout << "Введите имя: ";

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

cout << "Введите город: ";

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

// Инициализируем указатели на следующий и предыдущий элементы как nullptr

temp->next = nullptr;

temp->prev = nullptr;

// Возвращаем указатель на элемент

return temp;

}

// Функция для вывода списка на экран

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

// Создаем временный указатель на начало списка

Address\* t = \*phead;

// Перебираем список до конца и выводим имя и город каждого элемента

while (t) {

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

t = t->next;

}

cout << endl;

}

// Функция для поиска элемента по имени

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

// Создаем временный указатель на начало списка

Address\* t = \*phead;

// Перебираем список до конца и сравниваем имя каждого элемента с заданным

while (t) {

if (strcmp(name, t->name) == 0) {

// Если нашли совпадение, выводим имя и город элемента и выходим из функции

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

return;

}

t = t->next;

}

// Если не нашли совпадение, выводим сообщение об ошибке

cerr << "Имя не найдено" << endl;

}

// Функция для удаления элемента по имени

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

// Создаем временный указатель на начало списка

Address\* t = \*phead;

// Перебираем список до конца и сравниваем имя каждого элемента с заданным

while (t) {

if (strcmp(name, t->name) == 0) {

// Если нашли совпадение, удаляем элемент из списка и обновляем указатели

if (\*phead == t) {

\*phead = t->next;

}

if (\*plast == t) {

\*plast = t->prev;

}

if (t->prev != nullptr) {

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

}

if (t->next != nullptr) {

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

}

delete t;

cout << "Элемент удален" << endl;

return;

}

t = t->next;

}

// Если не нашли совпадение, выводим сообщение об ошибке

cerr << "Имя не найдено" << endl;

}

// Функция для записи списка в файл

void writeToFile(Address\*\* phead) {

Address\* t = \*phead;

ofstream file;

file.open("addresses.txt"); // открываем файл для записи

while (t) {

file << t->name << ' ' << t->city << endl; // записываем имя и город в файл

t = t->next;

}

file.close(); // закрываем файл

cout << "Данные записаны в файл" << endl;

}

// Функция для чтения списка из файла

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

ifstream file;

file.open("addresses.txt"); // открываем файл для чтения

if (!file.is\_open()) { // если не удалось открыть файл

cerr << "Не удалось открыть файл" << endl;

return;

}

char name[NAME\_SIZE];

char city[CITY\_SIZE];

while (file >> name >> city) { // читаем имя и город из файла

Address\* temp = new Address();

if (!temp) { // если не удалось выделить память

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

return;

}

strcpy(temp->name, name); // копируем имя в список

strcpy(temp->city, city); // копируем город в список

insert(temp, phead, plast); // вставляем элемент в список

}

file.close(); // закрываем файл

cout << "Данные считаны из файла" << endl;

}

// Функция для удаления повторяющихся элементов в списке

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

Address\* current = \*phead;

while (current != nullptr) {

Address\* runner = current->next;

while (runner != nullptr) {

if (strcmp(current->name, runner->name) == 0 || strcmp(current->city, runner->city) == 0) {

if (runner == \*plast) {

\*plast = runner->prev;

}

if (runner->prev != nullptr) {

runner->prev->next = runner->next;

}

if (runner->next != nullptr) {

runner->next->prev = runner->prev;

}

Address\* temp = runner;

runner = runner->next;

delete temp;// удаляем повторяющийся элемент

}

else {

runner = runner->next;

}

}

current = current->next;

}

cout << "Повторяющиеся элементы удалены" << endl;

}

int main() {

setlocale(LC\_ALL, "Russian");

Address\* head = nullptr;

Address\* last = nullptr;

while (true) {

switch (menu()) { // выводим меню и обрабатываем выбор пользователя

case 1:

insert(setElement(), &head, &last); // добавляем элемент в список

break;

case 2: {

char name[NAME\_SIZE];

cout << "Введите имя для удаления: ";

cin.getline(name, NAME\_SIZE);

delet(name, &head, &last); // удаляем элемент из списка

break;

}

case 3:

outputList(&head, &last); // выводим список на экран

break;

case 4: {

char name[NAME\_SIZE];

cout << "Введите имя для поиска: ";

cin.getline(name, NAME\_SIZE);

find(name, &head);

break;

}

case 5:

writeToFile(&head);

break;

case 6:

readFromFile(&head, &last);

break;

case 7:

removeDuplicates(&head, &last);

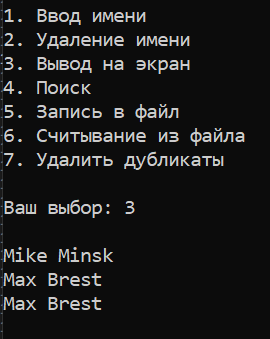
break;

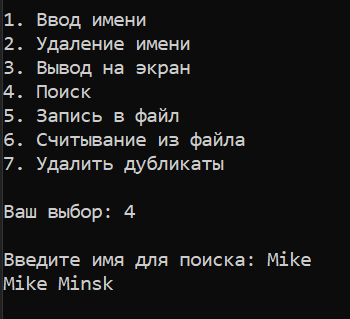
}

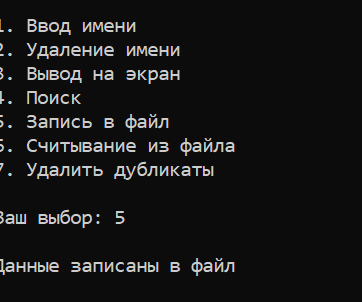
}

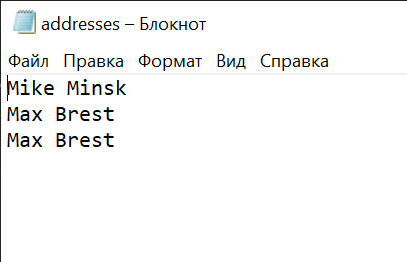
return 0;

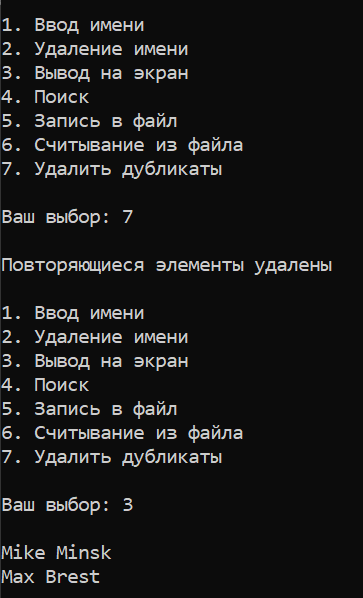
}

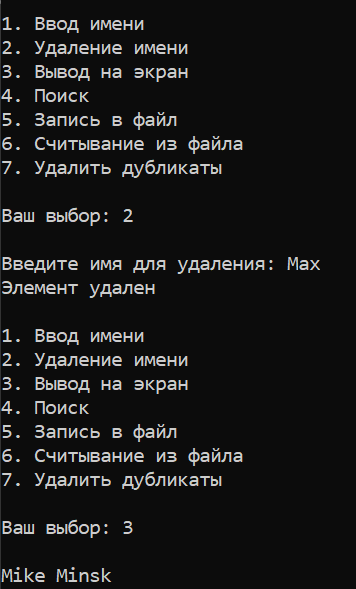




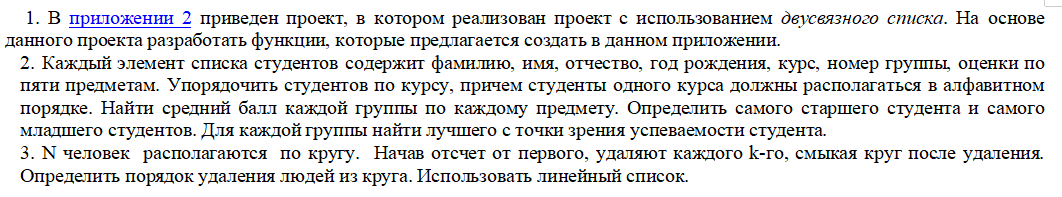








Дополнительные задания



Задание 1

#include <iostream>

#include <conio.h>

using namespace std;

struct Node

{

int x;

Node\* Next;

};

class List

{

public:

Node\* Head, \* Tail;

int size;

List() : Head(NULL), Tail(NULL), size(0) {};

~List();

void Add(int x);

void Show(int size);

int Count();

void Clear();

};

List::~List()

{

while (size != 0)

{

Node\* temp = Head->Next;

delete Head;

Head = temp;

size--;

}

}

int List::Count()

{

return size;

}

void List::Add(int x)

{

size++;

Node\* temp = new Node;

temp->Next = Head;

temp->x = x;

if (Head != NULL)

{

Tail->Next = temp;

Tail = temp;

}

else Head = Tail = temp;

}

void List::Show(int temp)

{

if (size != 0)

{

Node\* tempHead = Head;

temp = size;

while (temp != 0)

{

cout << tempHead->x << " ";

tempHead = tempHead->Next;

temp--;

}

}

else cout << "В списке нет элементов: " << size;

}

void List::Clear()

{

while (size)

{

Node\* temp = Head->Next;

delete Head;

Head = temp;

size--;

}

}

void main()

{

setlocale(LC\_ALL, "Russian");

int n;

List lst;

cout << "Введите кол-во элементов в списке" << endl;

cin >> n;

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

{

lst.Add(i);

}

cout << "Список:" << endl;

lst.Show(lst.Count());

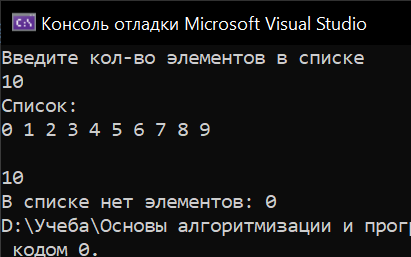
cout << endl;

cout << endl << lst.Count() << endl;

lst.Clear();

lst.Show(lst.Count());

}



Задание 2

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <limits>

#include <map>

#include <numeric>

#define total 0

struct Student { //инициализируем структуру

std::string surname; //фамилия

std::string firstName; //имя

std::string patronymic; //отчество

int birthYear; //год рождения

int course; //курс

int groupNumber; //номер группы

std::vector<int> grades; //оценки

};

bool compareStudents(const Student& s1, const Student& s2) { //сравниваем студентов

if (s1.course != s2.course) { //если курсы не равны

return s1.course < s2.course; //возвращаем курс

}

return s1.surname < s2.surname; //иначе возвращаем фамилию

}

void sortByCourseAndName(std::vector<Student>& students) { //сортируем по курсу и имени

std::sort(students.begin(), students.end(), compareStudents);

}

//. Найти средний балл каждой группы по каждому предмету (всего 5 предметов).

void findAverageGrade(const std::vector<Student>& students) { //находим средний балл

std::map<int, std::vector<int>> gradesByGroup; //инициализируем карту

for (const auto& student : students) { //для каждого студента

gradesByGroup[student.groupNumber].push\_back(student.grades[0]); //добавляем оценки

gradesByGroup[student.groupNumber].push\_back(student.grades[1]);

gradesByGroup[student.groupNumber].push\_back(student.grades[2]);

gradesByGroup[student.groupNumber].push\_back(student.grades[3]);

gradesByGroup[student.groupNumber].push\_back(student.grades[4]);

}

for (const auto& group : gradesByGroup) { //для каждой группы

double averageGrade = std::accumulate(group.second.begin(), group.second.end(), 0.0) / group.second.size(); //считаем средний балл

std::cout << "Group " << group.first << ": " << averageGrade << std::endl; //выводим в консоль

}

}

void findOldestAndYoungest(const std::vector<Student>& students) { //находим самого старшего и самого младшего студента

int oldestYear = std::numeric\_limits<int>::max(); //инициализируем переменные

std::string oldestName; //инициализируем переменные

int youngestYear = std::numeric\_limits<int>::min(); //инициализируем переменные

std::string youngestName; //инициализируем переменные

for (const auto& student : students) { //для каждого студента

if (student.birthYear < oldestYear) { //если год рождения меньше

oldestYear = student.birthYear; //то присваиваем переменной год рождения

oldestName = student.surname + " " + student.firstName + " " + student.patronymic; //иначе присваиваем переменной фамилию, имя и отчество

}

if (student.birthYear > youngestYear) { //если год рождения больше

youngestYear = student.birthYear; //то присваиваем переменной год рождения

youngestName = student.surname + " " + student.firstName + " " + student.patronymic; //иначе присваиваем переменной фамилию, имя и отчество

}

}

std::cout << "Oldest student: " << oldestName << " (" << oldestYear << ")" << std::endl;

std::cout << "Youngest student: " << youngestName << " (" << youngestYear << ")" << std::endl;

}

//найти лучшего студента и вывести его в консоль

void findBestStudent(const std::vector<Student>& students) { //находим лучшего студента

int bestStudentIndex = -1; //инициализируем переменные

int bestStudentGrade = std::numeric\_limits<int>::min(); //инициализируем переменные

for (size\_t i = 0; i < students.size(); i++) { //для каждого студента

int grade = students[i].grades[0] + students[i].grades[1] + students[i].grades[2] + students[i].grades[3] + students[i].grades[4]; //считаем сумму оценок

if (grade > bestStudentGrade) { //если сумма оценок больше

bestStudentGrade = grade; //то присваиваем переменной сумму оценок

bestStudentIndex = i; //иначе присваиваем переменной индекс

}

}

const Student& bestStudent = students[bestStudentIndex]; //присваиваем переменной студента

std::cout << "Best student: " << bestStudent.surname << " " << bestStudent.firstName << " " << bestStudent.patronymic << std::endl;

}

void printStudents(const std::vector<Student>& students) { //выводим студентов в консоль

for (size\_t i = 0; i < students.size(); i++) { //для каждого студента

const Student& student = students[i]; //присваиваем переменной студента

std::cout << student.surname << " " << student.firstName << " " << student.patronymic << " "

<< student.birthYear << " " << student.course << " " << student.groupNumber << " "; //выводим в консоль

for (size\_t j = 0; j < student.grades.size(); j++) { //для каждой оценки

std::cout << student.grades[j] << " "; //выводим в консоль

}

std::cout << std::endl;

}

std::cout << std::endl << std::endl;

}

int main() {

std::string surname;

std::string firstName;

std::string patronymic;

int birthYear;

int course;

int groupNumber;

std::vector<int> grades(5);

std::vector<Student> students = {

{"Bankuzov", "Mikhail", "Olegovich", 2005, 1, 7, {5, 4, 3, 4, 5}},

{"Plesko", "Max", "Petrovich", 2005, 2, 8, {4, 5, 4, 5, 4}},

{"Dmuhovskiy", "Denis", "Viktorovich", 2005, 1, 9, {3, 4, 5, 4, 3}},

}; //вектор студентов

//функция для вывода всех значений вектора Students

printStudents(students);

sortByCourseAndName(students);

printStudents(students);

findOldestAndYoungest(students);

findBestStudent(students);

findAverageGrade(students);

return 0;

}



Задание 3

#include <iostream>

#include <conio.h>

using namespace std;

struct Node

{

int x;

Node\* Next;

};

class List

{

public:

Node\* Head, \* Tail;

int size;

List() : Head(NULL), Tail(NULL), size(0) {};

~List();

void Add(int x);

void Show(int size);

int Count();

void Clear();

void Rans(List lst);

};

List::~List()

{

while (size != 0)

{

Node\* temp = Head->Next;

delete Head;

Head = temp;

size--;

}

}

int List::Count()

{

return size;

}

void List::Add(int x)

{

size++;

Node\* temp = new Node;

temp->Next = Head;

temp->x = x;

if (Head != NULL)

{

Tail->Next = temp;

Tail = temp;

}

else Head = Tail = temp;

}

void List::Show(int temp)

{

if (size != 0)

{

Node\* tempHead = Head;

temp = size;

while (temp != 0)

{

cout << tempHead->x << " ";

tempHead = tempHead->Next;

temp--;

}

}

else cout << "В списке нет элементов: " << size;

}

void List::Rans(List lst)

{

int k;

cout << endl;

cout << "Введите k:" << endl;

cin >> k;

Node\* temp = lst.Head;

while (temp->Next != nullptr)

{

for (int i = 1; i < k; i++)

{

temp = temp->Next;

}

Node\* buf = temp->Next;

cout << buf->x << " ";

if (buf == lst.Head) lst.Head = buf->Next;

if (buf == lst.Tail) lst.Tail = temp;

temp->Next = buf->Next;

delete buf;

}

}

void main()

{

setlocale(LC\_ALL, "Russian");

int n;

List lst;

cout << "Введите кол-во элементов в списке" << endl;

cin >> n;

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

{

lst.Add(i);

}

cout << "Список:" << endl;

lst.Show(lst.Count());

cout << endl;

lst.Rans(lst);

}

