УО «Белорусский государственный технологический университет»

Факультет информационных технологий

Кафедра «Информационных систем и технологий»

Специальность 1-40 05 01 «Информационные Системы и Технологии»

**Методическое пособие по дисциплине**

**Структуры данных**

|  |  |  |  |
| --- | --- | --- | --- |
| **Выполнил** |  |  |  |
| Студент 1 курса группы 1 |  |  | Д.И. Велютич |
|  | подпись, дата |  | инициалы и фамилия |
| **Принял(а)** |  |  | Н.И. Белодед |
|  | подпись, дата |  | инициалы и фамилия |

Минск 2023

**Содержание**

[Лабораторная работа 1 2](#_Toc132484758)

[Лабораторная работа 2 8](#_Toc132484759)

[Лабораторная работа 3 14](#_Toc132484760)

[Лабораторная работа 4 19](#_Toc132484761)

[Лабораторная работа 5 24](#_Toc132484762)

[Лабораторная работа 6 28](#_Toc132484763)

[Лабораторная работа 7 33](#_Toc132484764)

[Лабораторная работа 8 42](#_Toc132484765)

[Лабораторная работа 9 46](#_Toc132484766)

# **Лабораторная работа 1**

**Тема: "****Перечисление, Объединение, STL контейнеры,**

**Массив, Дек, Одно/дву-связный список, Вектор"**

**Цель: Научится работать с коллекциями элементов различных типов данных.**

1. Введение

В данной лабораторной работе будут рассмотрены различные контейнеры в C++, которые позволяют хранить и управлять коллекциями элементов различных типов данных.

2. Теоретическая основа

Первый тип контейнера, который будет рассмотрен, это перечисление. Оно представляет собой набор констант, каждой из которых соответствует уникальное целочисленное значение. Перечисления удобны для использования в коде, где нужно задать конечный набор значений, например, при работе с состояниями или типами объектов.

Второй тип контейнера, который будет изучен, это объединение. Оно позволяет объединить несколько переменных различных типов в один объект, при этом все переменные используют одну область памяти. Объединения могут быть полезны при работе с данными, которые имеют разные форматы в зависимости от каких-то условий.

Далее будут изучены контейнеры из стандартной библиотеки C++, включающие массив, дек, односвязный и двусвязный список, а также вектор. Массив представляет собой набор элементов одного типа, расположенных в памяти последовательно.

Дек и списки являются контейнерами, позволяющими быстро добавлять и удалять элементы в начале и конце коллекции. Односвязный список хранит каждый элемент в узле, содержащем указатель на следующий элемент, а двусвязный список - ещё и на предыдущий. Вектор является контейнером, предоставляющим динамический массив, способный изменять свой размер по мере необходимости.

Использование STL контейнеров значительно упрощает работу с коллекциями в C++, так как позволяет не беспокоиться о выделении и освобождении памяти, обходить коллекцию в циклах и быстро выполнять различные операции с элементами, такие как поиск, сортировка и т.д.

2.1 Примеры

Конкретные примеры кода помогут лучше понять теорию о контейнерах в C++.

Вот несколько примеров:

Пример использования перечислений:

enum PCState { ON, OFF, SLEEP };

PCState currentState = ON;

if (currentState == ON) {

cout << "PC is ON" << endl;

}

Пример использования объединений:

union myUnion {

int i;

float f;

};

myUnion u;

u.i = 42;

cout << "Union value: " << u.f << endl;

Пример использования массива:

int myArray[5] = { 1, 2, 3, 4, 5 };

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

cout << myArray[i] << " ";

}

Пример использования дека:

deque<int> myDeque;

myDeque.push\_back(1);

myDeque.push\_front(2);

myDeque.push\_back(3);

cout << "Deque front element: " << myDeque.front() << endl;

Пример использования односвязного списка:

struct Node {

int data;

Node\* next;

};

Node\* head = NULL;

Node\* newNode = new Node();

newNode->data = 42;

newNode->next = head;

head = newNode;

cout << "List head element: " << head->data << endl;

Пример использования двусвязного списка:

struct Node {

int data;

Node\* next;

Node\* prev;

};

Node\* head = NULL;

Node\* newNode = new Node();

newNode->data = 42;

newNode->next = head;

newNode->prev = NULL;

if (head != NULL) {

head->prev = newNode;

}

head = newNode;

cout << "List head element: " << head->data << endl;

Пример использования вектора:

vector<int> myVector;

myVector.push\_back(1);

myVector.push\_back(2);

myVector.push\_back(3);

cout << "Vector element: " << myVector[1] << endl;

3. Заключение

В результате выполнения данной лабораторной работы студент должен овладеть основными типами контейнеров в C++ и понимать, как правильно выбирать подходящий контейнер для решения конкретных задач.

Задание к лабораторной работе:

1. Напишите программу на C++, которая выводит состояния ПК в зависимости от выбора пользователя (включен, выключен, спящий режим), используя перечисления.
2. Напишите программу на C++, которая выводит цвета радуги в зависимости от выбора пользователя, используя перечисления.
3. Напишите программу на C++, которая выводит название планеты Солнечной системы в зависимости от выбора пользователя, используя перечисления.
4. Напишите программу на C++, которая выводит дни недели в зависимости от выбора пользователя, используя перечисления.
5. Напишите программу на C++, которая добавляет n элементов в конец односвязного списка и выводит список на экран.
6. Напишите программу на C++, которая добавляет n элементов в начало двусвязного списка и выводит список на экран.
7. Напишите программу на C++, которая находит максимальный элемент вектора и выводит его на экран.
8. Напишите программу на C++, которая находит минимальный элемент дека, добавляет его в начало и выводит дек на экран.
9. Напишите программу на C++, которая находит сумму минимального и максимального элементов массива и выводит ее на экран.
10. Напишите программу на C++, которая находит сумму всех элементов вектора и выводит ее на экран.
11. Напишите программу на C++, которая удаляет n элемент из двусвязного списка, добавляет его в начало и выводит список на экран.
12. Напишите программу на C++, которая находит произведение всех элементов дека и выводит его на экран.
13. Напишите программу на C++, которая удаляет все элементы односвязного списка, добавляет n новых и выводит список на экран.

Программный код:

#include <iostream>

using namespace std;

// Перечисление для состояний ПК

enum PCState {

OFF,

ON,

SLEEP

};

// Перечисление для цветов радуги

enum RainbowColor {

RED,

ORANGE,

YELLOW,

GREEN,

BLUE,

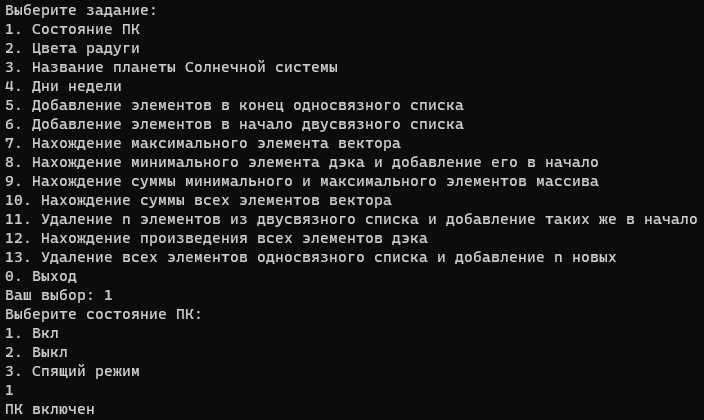
INDIGO,

VIOLET

};

Скриншоты выполнения программы:

Функция «Состояние ПК»



Дополнительные задания:  
  
1. Напишите программу на C++, которая находит максимальный элемент массива и меняет его местами с первым элементом, затем выводит массив на экран.

2. Напишите программу на C++, которая запрашивает у пользователя два целых числа, затем выводит на экран все числа от меньшего до большего включительно. Например, если пользователь вводит числа 4 и 9, программа должна вывести на экран числа 4, 5, 6, 7, 8, 9.

3. Напишите программу на C++, которая запрашивает у пользователя целое число N и выводит на экран первые N чисел последовательности Фибоначчи. Последовательность Фибоначчи начинается с чисел 0 и 1, а каждое следующее число равно сумме двух предыдущих чисел. Например, первые 10 чисел последовательности Фибоначчи: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34.

Контрольные вопросы:

1. Какие преимущества предоставляют перечисления при работе с конечным набором значений, и в каких ситуациях их использование наиболее уместно?

2. В чем основная разница между односвязным и двусвязным списками, и какие преимущества они предоставляют при выполнении операций добавления и удаления элементов?

3. Какие особенности и преимущества использования векторов как контейнеров в сравнении с другими типами контейнеров, такими как массивы, деки и списки?

# **Лабораторная работа 2**

**Тема: "Схема БД. Меню программы"**

**Цель: Создание программы для управления Б/Д**

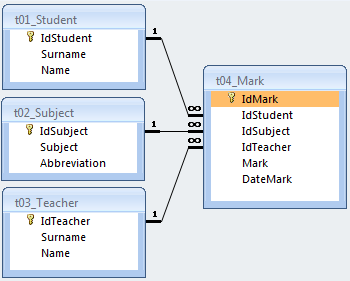
1. Введение

Цель данной лабораторной работы заключается в создании программы для управления базой данных, а также разработке меню для взаимодействия пользователя с базой данных. В ходе работы будут использованы теоретические знания по теме "Схема БД. Меню программы".

2. Теоретическая основа

2.1. Схема базы данных (БД)

Схема БД представляет собой графическое описание структуры базы данных, включая таблицы, ключи, связи и т. д. Это позволяет разработчикам и администраторам БД лучше понимать структуру данных и связи между ними.



CREATE TABLE Customers (

CustomerID int PRIMARY KEY,

CustomerName varchar(255),

ContactName varchar(255),

Country varchar(255)

);

CREATE TABLE Orders (

OrderID int PRIMARY KEY,

CustomerID int,

OrderDate date,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

2.2. Меню программы

Меню программы является пользовательским интерфейсом, предоставляющим выбор различных функций и действий, доступных в программе. Меню обычно включает в себя текстовые пункты, среди которых пользователь может выбирать, выполняя соответствующие команды.

#include <iostream>

**using** **namespace** std;

**int** **main**() {

**int** choice;

cout << "Menu:" << endl;

cout << "1. Option 1" << endl;

cout << "2. Option 2" << endl;

cout << "3. Option 3" << endl;

cout << "Enter your choice: ";

cin >> choice;

**switch** (choice) {

**case** **1**:

// Call function for option 1

**break**;

**case** **2**:

// Call function for option 2

**break**;

**case** **3**:

// Call function for option 3

**break**;

**default:**

cout << "Invalid choice" << endl;

**break**;

}

**return** **0**;

}

3. Реализация программы

Программа будет разработана на языке C++ и представляет собой текстовый интерфейс с меню для управления базой данных. В программе предусмотрены следующие функции:

1. Ввод данных
2. Изменение данных
3. Вывод данных

4. Вывод данных по индексу

5. Сохранение данных в файл

6. Чтение данных из файла

7. Удаление файла

8. Удаление элементов

9. Сортировка данных

10. Поиск данных

Для каждой из этих функций разработать соответствующие подменю, позволяющие пользователю вводить необходимые параметры для выполнения выбранной команды.

Заключение:

В результате выполнения лабораторной работы будет разработана программа управления базой данных с использованием теоретических знаний по теме "Схема БД. Меню программы". Программа предоставляет пользователю удобный текстовый интерфейс для выполнения различных действий с базой данных.

Задания к лабораторной работе:

Основываясь на вашем выбранном варианте (номере в списке), вам необходимо разработать базу данных в соответствии с указанной темой:

1. База данных книжного магазина: данные о книгах, авторах, ценах и количестве на складе.

2. База данных аптеки: информация о лекарствах, их ценах, количестве и сроке годности.

3. База данных мебельного магазина: данные о мебели, материалах, размерах, ценах и наличии.

4. База данных туристического агентства: информация о турах, ценах, датах и местах проведения.

5. База данных детского сада: данные о детях, родителях, воспитателях и графике занятий.

6. База данных футбольного чемпионата: информация о командах, игроках, расписании матчей и результатах.

7. База данных школы: данные об учителях, учениках, предметах и оценках.

8. База данных ветеринарного пункта: информация о пациентах, услугах, ценах и расписании приема.

9. База данных «Банкомат»: данные о банкоматах, операциях, пользователях и лимитах.

10. База данных продюсерского центра: информация о проектах, участниках, договорах и оплате.

11. База данных овощной базы: данные о товарах, складе, поставщиках и движении товара.

12. База данных отдела по занятости населения: информация о соискателях, вакансиях и услугах.

13. База данных паспортного стола: данные о гражданах, паспортах, сроках действия и замены.

14. База данных по договорам: информация о договорах, сторонах, условиях и оплате.

15. База данных по подготовительным курсам: данные о курсах, преподавателях, учениках и расписании занятий.

16. База данных «Расписание»: информация о мероприятиях, датах, времени и местах проведения.

17. База данных «Телефонный справочник организации»: данные о сотрудниках, их контактах и должностях.

18. База данных по ремонту автомобилей: информация о мастерских, услугах, ценах и графике работы.

19. База данных аэропорта: данные о рейсах, билетах, ценах и расписании.

20. База данных почтового отделения: информация о посылках, отправителях, получателях, статусе доставки и оплате услуг.

21. База данных ЗАГСа: информация о регистрации браков, разводов, рождений и смертей, а также участники и даты событий.

22. База данных преступлений: данные о преступлениях, подозреваемых, свидетелях и результаты расследований.

23. База данных учета движения вычислительной техники: информация о компьютерах, их состоянии, местоположении и истории перемещений.

24. База данных абонентов сотовой компании: данные о клиентах, их тарифах, оплате и использовании услуг.

25. База данных оздоровительного центра: информация о занятиях, тренерах, клиентах и расписании.

26. База данных системы проверки олимпиадных заданий: данные о заданиях, участниках, результатах и критериях оценки.

27. База данных учета занятий преподавателя: информация о преподавателях, предметах, группах и расписании занятий.

28. База данных помещений вуза: данные о аудиториях, их паспортах, состоянии и регистрации ремонтов.

29. База данных аэропорта/жд (рейсы, билеты): информация о рейсах, станциях, ценах билетов и расписании движения.

30. База данных студенческого общежития: данные о корпусах, комнатах, проживающих студентах и оплате за проживание.

Программный код:

void Menu() {

cout << "--------КиберМаркет--------**\n**" << endl;

cout << "1. Ввод данных" << endl;

cout << "3. Вывод данных" << endl;

cout << "0. Выход" << endl;

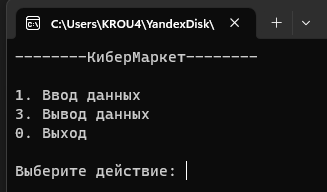
cout << "**\n**Выберите действие: ";

GetValueFromUser(\_stateMenu);

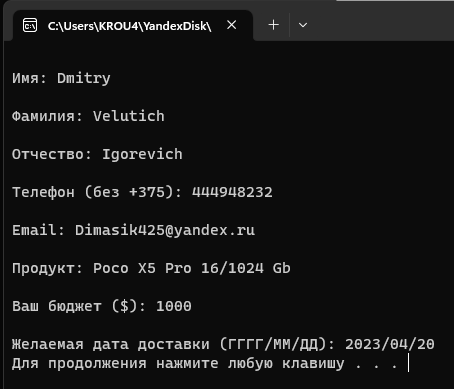
}

Скриншоты выполнения программы:

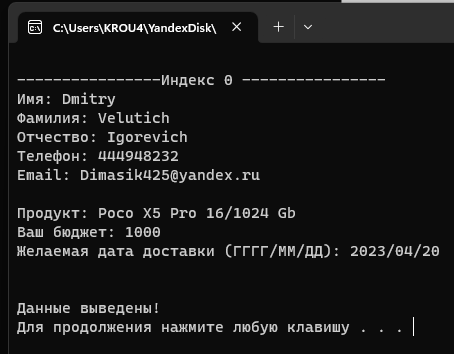
Меню программы



Ввод данных о клиенте и продуктах



Вывод данных



Дополнительные задания:  
  
1. Красиво распределите функции программы методом древа (корневой каталог включает в себя несколько дочерних элементов)

2. Добавьте проверку, чтобы пользователь не мог ввести данные типа *string* в поле типа *int*, *double*

3. Добавьте проверку, чтобы пользователь не мог ввести данные типа *int*, *double* в поле типа *string*

Контрольные вопросы:

1. Какие ключевые элементы составляют схему базы данных, и как эти элементы помогают разработчикам и администраторам понимать структуру данных и связи между ними?

2. Какие функции могут быть включены в меню программы для управления базой данных, и какие действия должны быть доступны пользователям для эффективного взаимодействия с данными?

3. Какие подходы к проектированию пользовательского интерфейса можно использовать для создания интуитивно понятного и удобного меню программы, обеспечивающего быстрый доступ к основным функциям работы с базой данных?

# **Лабораторная работа 3**

**Тема: "Двоичные и текстовые файлы"**

**Цель: Научится работать с текстовыми и двоичными файлами.**

1. Введение

Цель данной лабораторной работы заключается в разработке и реализации программы на языке C++ для работы с текстовыми файлами. В ходе работы используются теоретические знания по теме "Двоичные и текстовые файлы".

2. Теоретическая основа

2.1. Текстовые файлы

Текстовые файлы — это файлы, содержащие данные, представленные в виде символов. Они хранят информацию в виде последовательности символов, разделённых специальными символами (например, перевод строки). Текстовые файлы удобны для чтения и редактирования человеком, так как они представлены в виде обычного текста.



2.2. Двоичные файлы

Двоичные файлы — это файлы, содержащие данные в формате, который не может быть прочитан или редактирован напрямую человеком. Они хранят информацию в виде последовательности байт, и для работы с ними требуются специализированные инструменты или программы.



3.Реализация программы

Программа разрабатывается на языке C++ и реализует следующие функции:

1. Создание текстового файла, содержащего список студентов с их именами, возрастом и номером группы.

2. Чтение данных из текстового файла и вывод этих данных на экран.

3. Добавление новых записей о студентах в текстовый файл.

4. Создание двоичного файла, содержащего список студентов с их именами, возрастом и номером группы.

5. Чтение данных из двоичного файла и вывод этих данных на экран.

6. Добавление новых записей о студентах в двоичный файл.

Реализация функций осуществляется с помощью структуры библиотеки <fstream> с операторами ifstream и ofstream.

Функция `save\_to\_txt` сохраняет данные о студентах в текстовый файл. Она принимает на вход вектор структур с данными студентов и имя файла для сохранения информации. В функции открывается поток записи ofstream с флагом `ios::app`, который позволяет добавлять данные в файл без удаления существующих данных. Затем данные о студентах записываются в файл.

void save\_to\_txt(const vector<Student>& students, const string& fileName) {

ofstream file(fileName, ios::app);

for (const auto& student : students) {

file << student.name << " " << student.age << " " << student.group << endl;

}

file.close();

}

Функция `read\_from\_txt` считывает данные о студентах из текстового файла и выводит их на экран. Она принимает на вход имя файла и открывает поток чтения ifstream. В случае ошибки при открытии файла выводится сообщение об ошибке. Затем функция считывает данные из файла и выводит их на экран.

void read\_from\_txt(const string& fileName) {

ifstream file(fileName);

Student temp;

if (!file.is\_open()) {

cerr << "Невозможно открыть файл " << fileName << endl;

return;

}

while (file >> temp.name >> temp.age >> temp.group) {

cout << temp.name << " " << temp.age << " " << temp.group << endl;

}

file.close();

}

Функции `save\_to\_bin` и `read\_from\_bin` аналогичны функциям `save\_to\_txt` и `read\_from\_txt`, но работают с двоичными файлами. Для записи и чтения данных используются флаги `**ios::binary**` и `ios::app` для работы с двоичными файлами. Для записи и чтения структур студентов используются функции `write` и `read` с приведением типов данных к указателям на char.

4. Заключение

В результате выполнения лабораторной работы будет разработана программа на языке C++ для работы с текстовыми и двоичными файлами, используя теоретические знания по теме "Двоичные и текстовые файлы". Программа реализует функции сохранения, чтения и добавления записей о студентах в текстовые и двоичные файлы.

Задания к лабораторной работе:

Разработать программу на языке C++ для работы с текстовыми и двоичными файлами, которая будет выполнять следующие задачи:

1. Создать текстовый файл, содержащий список студентов с их именами, возрастом, и номером группы.

2. Реализовать функцию для чтения данных из текстового файла, созданного на первом шаге, и вывода этих данных на экран.

3. Реализовать функцию для добавления новых записей о студентах в текстовый файл.

4. Создать двоичный файл, содержащий список студентов с их именами, возрастом, и номером группы.

5. Реализовать функцию для чтения данных из двоичного файла, созданного на четвёртом шаге, и вывода этих данных на экран.

6. Реализовать функцию для добавления новых записей о студентах в двоичный файл.

Программный код:

void read\_from\_bin(const string& fileName) {

ifstream file(fileName, ios::binary);

}

void read\_from\_txt(const string& fileName) {

ifstream file(fileName);

}

void save\_to\_bin(const vector<Student>& students, const string& fileName) {

ofstream file(fileName, ios::binary | ios::app);

}

void save\_to\_txt(const vector<Student>& students, const string& fileName) {

ofstream file(fileName, ios::app);

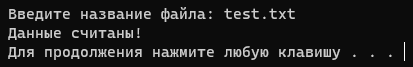
}

Скриншоты выполнения программы:

Экспорт структуры в файл



Импорт структуры из файла



Дополнительные задания:

1 задание: Разработайте программу на C++, которая позволяет шифровать и дешифровать содержимое файла. Пользователь должен иметь возможность вводить название файла и выбирать операцию (шифрование или дешифрование). Программа может использовать простой алгоритм шифрования, такой как шифр Цезаря или XOR.

2 задание: Написать программу, которая считывает из двоичного файла целые числа, сортирует их по возрастанию и записывает отсортированные числа в другой текстовый файл.

3 задание: Написать программу, которая считывает из текстового файла данные о температуре воздуха (например, за каждый день месяца), вычисляет среднюю температуру и записывает результаты в бинарный файл.

Контрольные вопросы:

1. Каковы основные отличия между текстовыми и двоичными файлами, и в каких случаях предпочтительнее использовать один тип файла вместо другого при разработке программы на C++?

2. Какие инструменты и методы предоставляет язык C++ для работы с текстовыми файлами, и как их можно использовать для реализации функций сохранения, чтения и удаления данных?

3. В чем состоит роль библиотеки <fstream> при работе с файлами в C++, и какие операторы и объекты она предоставляет для эффективного взаимодействия с текстовыми и двоичными файлами?

# **Лабораторная работа 4**

**Тема: "Индексирование записей. Простой/сложный индекс"**

**Цель: Изучение индексирования записей и реализация его на практике.**

1. Теоретические сведения:

Индексирование записей — это процесс создания индексов для быстрого доступа к записям в базе данных. Индекс — это структура данных, содержащая информацию о расположении записей в базе данных.

Простой индекс — это индекс, созданный для одного поля таблицы. Он упрощает поиск данных по этому полю, но не дает преимущества при поиске по другим полям.

Сложный индекс — это индекс, созданный для двух или более полей таблицы. Он упрощает поиск данных по комбинации полей, указанных в индексе.

#include <iostream>

#include <algorithm>

#include <vector>

**using** **namespace** std;

**struct** Record {

**int** id;

string name;

**int** age;

string country;

};

**int** **main**() {

vector<Record> records = {

{**1**, "John", **25**, "USA"},

{**2**, "Alice", **30**, "Canada"},

{**3**, "Bob", **35**, "UK"},

{**4**, "Emily", **40**, "USA"}

};

// Создаем простой индекс для поля id

sort(records.begin(), records.end(), [](**const** Record& r1, **const** Record& r2) {

**return** r1.id < r2.id;

});

// Создаем сложный индекс для полей name и country

sort(records.begin(), records.end(), [](**const** Record& r1, **const** Record& r2) {

**return** r1.name < r2.name;

});

stable\_sort(records.begin(), records.end(), [](**const** Record& r1, **const** Record& r2) {

**return** r1.country < r2.country;

});

**return** **0**;

}

2. Реализация:

В данном коде представлены две функции: "print\_all\_struct" и "findin".

Функция "print\_all\_struct" выводит на экран все данные, содержащиеся в массиве структур "Client", включая информацию о продуктах, которые заказал каждый клиент. Для этого используется цикл "for", который проходится по каждому элементу массива "t", содержащего данные о клиентах. Затем используется вложенный цикл "for", который выводит информацию о продуктах, которые заказал каждый клиент.

void print\_all\_struct(Client\* t, int kol, int kolvo)

{

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

{

cout << "**\n**Имя: " << t[i].Name

<< "**\n**Фамилия: " << t[i].Fam

<< "**\n**Отчество: " << t[i].Otch

<< "**\n**Телефон: " << t[i].TelephoneNumber

<< "**\n**Email: " << t[i].Email;

for (int k = 0; k < kolvo; k++)

{

cout << "**\n**Продукт: " << t[i].tovar[k].product

<< "**\n**Ваш бюджет: " << t[i].tovar[k].cost

<< "**\n**Желаемая дата доставки (ГГГГ/ММ/ДД): " << t[i].tovar[k].data\_dost;

cout << endl;

}

}

}

Функция "findin" позволяет найти данные по заданному индексу в массиве структур "Client". Для этого используется цикл "for", который проходится по каждому элементу массива "t". Если индекс i совпадает с заданным индексом, то функция выводит на экран информацию о клиенте и его заказах.

void findin(Client\* t, int index, int kolvo)

{

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

{

if (i == index) {

cout << "**\n**Имя: " << t[i].Name

<< "**\n**Фамилия: " << t[i].Fam

<< "**\n**Отчество: " << t[i].Otch

<< "**\n**Телефон: " << t[i].TelephoneNumber

<< "**\n**Email: " << t[i].Email;

for (int k = 0; k < kolvo; k++)

{

cout << "**\n**Продукт: " << t[i].tovar[k].product

<< "**\n**Ваш бюджет: " << t[i].tovar[k].cost

<< "**\n**Желаемая дата доставки (ГГГГ/ММ/ДД): " << t[i].tovar[k].data\_dost;

cout << endl;

}

}

}

}

3. Выводы:

Индексирование записей является важным инструментом для обеспечения быстрого доступа к данным в базе данных. Простой индекс упрощает поиск данных по одному полю, а сложный индекс - по комбинации нескольких полей.

В данном коде индексирование записей не реализовано явно, однако функция "findin" может быть использована для быстрого поиска данных по заданному индексу.

Задание к лабораторной работе:

Для базы данных, которую вы создали во 2-ой лабораторной работе, необходимо создать индексы для каждой структуры данных и добавить возможность выполнения линейного поиска данных по этим индексам. Когда пользователь вводит индекс, программа должна выводить всю структуру данных, связанную с этим индексом.

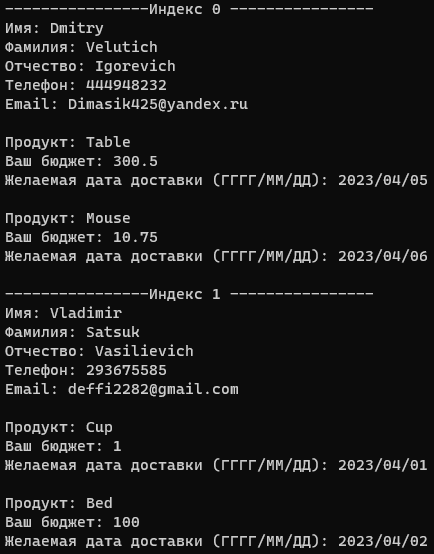
Программный код:

void print\_all\_struct(Client\* t, int kol, int kolvo)

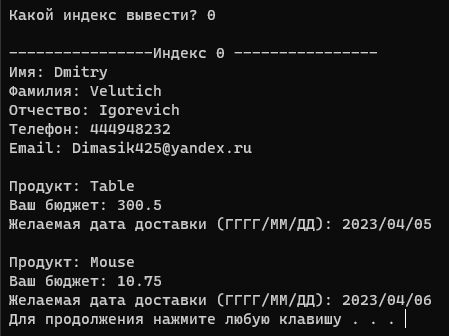
void findin(Client\* t, int index, int kolvo)

Скриншоты выполнения программы:

Исходная структура



Вывод нулевого индекса (Dmitry)



Дополнительные задания:

1 задание: Постраничное отображение данных.

Разработайте программу на C++, которая будет отображать данные из базы данных постранично, используя индексы. Пользователь должен иметь возможность указывать количество записей на странице и выбирать страницу для просмотра. Программа должна корректно обрабатывать ситуации, когда пользователь вводит некорректные значения (например, номер страницы, которой не существует).

2 задание: Написать программу, которая хранит информацию о книгах в библиотеке (название, автор, издательство, год издания) в сложном индексированном массиве. Реализовать методы для добавления новых книг, редактирования информации о существующих книгах и поиска книг по заданным критериям (например, по автору, году издания).

3 задание: Написать программу, которая хранит информацию о заказах в интернет-магазине (номер заказа, дата оформления, сумма заказа, статус заказа) в простом индексированном массиве. Реализовать методы для добавления новых заказов, изменения статуса существующих заказов и вывода списка заказов с заданным статусом.

Контрольные вопросы:

1. Какие преимущества предоставляет использование индексов при работе с базами данных, и какие типы индексов существуют в зависимости от количества полей, на которые они созданы?

2. В чем состоит разница между простыми и сложными индексами, и каковы их особенности при поиске данных по различным полям в таблице?

3. Какие подходы и методы можно использовать для создания и работы с простыми и сложными индексами на примере языка программирования C++, и как они могут быть применены для оптимизации поиска и сортировки данных?

# **Лабораторная работа 5**

**Тема: "Запись/чтение массива структур в файл"**

**Цель: Изучение методов записи и чтения массива структур в файл и их реализация на практике.**

1. Теоретические сведения:

Запись/чтение массива структур в файл — это процесс сохранения данных из массива структур в файл на диске или чтения данных из файла в массив структур.

Для записи массива структур в файл необходимо открыть файл для записи (ofstream), перебрать все элементы массива и записать их значения в файл в нужном формате.

Для чтения массива структур из файла необходимо открыть файл для чтения (ifstream), прочитать первое значение из файла, которое является количеством элементов массива, затем создать массив структур нужного размера и прочитать значения для каждого элемента из файла.

#include <iostream>

#include <fstream>

**using** **namespace** std;

**struct** Record {

**int** id;

string name;

};

**int** **main**() {

**const** **int** size = **3**;

Record records[size] = {{**1**, "John"}, {**2**, "Alice"}, {**3**, "Bob"}};

// Записываем массив структур в файл

ofstream outFile("data.txt");

**for** (**int** i = **0**; i < size; ++i) {

outFile << records[i].id << " " << records[i].name << endl;

}

outFile.close();

// Читаем массив структур из файла

ifstream inFile("data.txt");

Record readRecords[size];

**for** (**int** i = **0**; i < size; ++i) {

inFile >> readRecords[i].id >> readRecords[i].name;

}

inFile.close();

// Выводим прочитанные значения

**for** (**int** i = **0**; i < size; ++i) {

cout << readRecords[i].id << " " << readRecords[i].name << endl;

}

**return** **0**;

}

2. Реализация:

В данном коде представлены две функции: "savin" и "read".

Функция "savin" записывает данные из массива структур "Client" в файл. Она открывает файл для записи и записывает в него количество элементов массива "n", а затем перебирает каждый элемент массива и записывает его поля, включая количество продуктов "l" и информацию о каждом продукте.

void savin(Client\* d, int n, string fileName, int l)

{

ofstream record(fileName, ios::out);

record.close();

}

Функция "read" считывает данные из файла и сохраняет их в массив структур "Client". Она открывает файл для чтения, считывает количество элементов массива "n", создает массив структур нужного размера и затем перебирает каждый элемент массива и считывает его поля, включая количество продуктов "l" и информацию о каждом продукте.

void read(Client\* (&d), int& n, int& l, const std::string& fileName)

{

ifstream record(fileName);

record.close();

}

3. Выводы:

Запись/чтение массива структур в файл является важным инструментом для сохранения данных в базе данных на диске и их последующего использования. В данном коде использованы функции "savin" и "read" для записи и чтения массива структур "Client".

Для успешной записи и чтения данных из файла необходимо использовать правильный формат записи и чтения. В данном коде используется метод записи и чтения, который позволяет сохранять и считывать данные из файла в нужном формате.

Знание методов записи/чтения массива структур в файл является важным для создания эффективных баз данных. В дальнейшем, эти методы могут использоваться для сохранения данных в других форматах, таких как JSON, XML и т.д.

Задание к лабораторной работе:

В соответствии с заданием из 2-ой лабораторной работы нужно создать массив структур для базы данных и реализовать функции считывания из файла, вывода структуры на экран, записи в файл и добавления данных в массив. Для функций считывания и записи необходимо добавить меню, где пользователь может ввести название файла вручную или использовать уже готовый файл. Также необходимо разработать функции удаления содержимого файла и полного удаления файла.

Программный код:

void savin(Client\* d, int n, string fileName, int l)

{

ofstream record(fileName, ios::out);

record.close();

}

void read(Client\* (&d), int& n, int& l, const std::string& fileName)

{

ifstream record(fileName);

record.close();

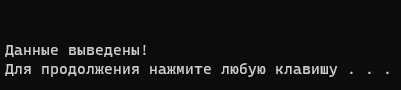
}

Скриншоты выполнения программы:

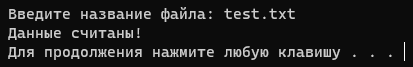
Экспорт структуры в файл



Отчистка структуры для примера



Импорт структуры из файла



Дополнительные задания:

1 задание: Копирование содержимого одного файла в другой.

Создайте программу на C++, которая копирует содержимое одного файла в другой. Имена обоих файлов должны быть введены пользователем с клавиатуры. После успешного копирования содержимого, программа должна выводить сообщение об успешной операции.

2 задание: Написать программу, которая записывает информацию о студентах (имя, фамилия, номер зачетной книжки, средний балл) в массив структур, а затем сохраняет его в файл в двоичном формате. Реализовать методы для чтения информации из файла и вывода списка студентов.

3 задание: Написать программу, которая хранит информацию о продуктах в магазине (название, цена, количество на складе) в массиве структур. Реализовать методы для записи этого массива структур в текстовый файл, чтения этого файла в массив структур и вывода информации о продуктах.

Контрольные вопросы:

1. Каковы основные этапы записи и чтения массива структур в файл на языке программирования C++, и какие инструменты и объекты используются для этого?

2. В чем состоит роль объектов ifstream и ofstream при работе с файлами, и какие основные операции они предоставляют для чтения и записи данных из/в файлы?

3. Как обеспечить корректное сохранение и восстановление данных при записи и чтении массива структур из файла, и какие особенности формата файла следует учесть для успешного взаимодействия с данными?

# **Лабораторная работа 6**

**Тема: "Редактирование файлов: удаление, изменение поля"**

**Цель: Изучение методов удаления и изменения данных в файле и их реализация на практике.**

1. Теоретические сведения:

Редактирование файлов — это процесс изменения данных в существующем файле. В контексте баз данных это может быть изменение или удаление полей записей в файле.

Изменение поля — это процесс изменения значения конкретного поля записи в файле. Для этого необходимо найти нужную запись в файле и изменить ее поле на новое значение.

Удаление поля — это процесс удаления конкретного поля записи в файле. Для этого необходимо найти нужную запись в файле и удалить ее поле.

#include <iostream>

#include <fstream>

using namespace std;

struct Record {

string name;

int age;

};

int main() {

const string fileName = "data.txt";

Record record = {"John", 25};

// Записываем структуру в файл

ofstream outFile(fileName);

outFile << record.name << " " << record.age << endl;

outFile.close();

// Читаем структуру из файла

ifstream inFile(fileName);

inFile >> record.name >> record.age;

inFile.close();

// Изменяем поле и записываем обновленную структуру в файл

record.age = 30;

ofstream outFile2(fileName);

outFile2 << record.name << " " << record.age << endl;

outFile2.close();

// Читаем обновленную структуру из файла

ifstream inFile2(fileName);

inFile2 >> record.name >> record.age;

inFile2.close();

// Удаляем поле и записываем обновленную структуру в файл

record.age = 0;

ofstream outFile3(fileName);

outFile3 << record.name << " " << record.age << endl;

outFile3.close();

// Читаем обновленную структуру из файла

ifstream inFile3(fileName);

inFile3 >> record.name >> record.age;

inFile3.close();

// Выводим значения

cout << record.name << " " << record.age << endl;

return 0;

}

2. Реализация:

В данном коде представлены две функции: "Change" и "del".

Функция "Change" позволяет изменять поля в выбранной записи массива структур "Client". Она запрашивает у пользователя номер записи, которую нужно изменить, затем запрашивает новые значения для каждого поля. В конце функция сообщает, что данные изменены.

void Change(Client\* (&t), int n, int l)

{

int i;

cout << "Введите номер элемента (от 0 до " << n - 1 << "): ";

GetValueFromUser(i);

system("cls");

if (i >= 0 && i < n) {

cout << "**\n**Имя: "; GetStringValueFromUser(t[i].Name);

cout << "**\n**Фамилия: "; GetStringValueFromUser(t[i].Fam);

cout << "**\n**Отчество: "; GetStringValueFromUser(t[i].Otch);

cout << "**\n**Телефон: "; GetValueFromUser(t[i].TelephoneNumber);

cout << "**\n**Email: "; cin >> t[i].Email;

for (int k = 0; k < l; k++)

{

cout << "**\n**Продукт: "; GetStringValueFromUser(t[i].tovar[k].product);

cout << "**\n**Желаемая цена ($): "; GetValueFromUser(t[i].tovar[k].cost);

cout << "**\n**Желаемая дата доставки (ГГГГ/ММ/ДД): "; GetValueFromUser(t[i].tovar[k].data\_dost);

}

system("cls");

cout << "Данные изменены!" << endl;

}

}

Функция "del" позволяет удалить выбранную запись массива структур "Client". Она запрашивает у пользователя номер записи, которую нужно удалить, и затем пересоздает массив структур без этой записи. В конце функция сообщает, что данные удалены.

void del(Client\* (&d), int& n)

{

int \_n;

cout << "Введите номер элемента (от 0 до " << n - 1 << "): ";

GetValueFromUser(\_n);

system("cls");

if (\_n >= 0 && \_n < n) {

Client\* buf = new Client[n];

Copy(buf, d, n);

--n;

d = new Client[n];

int q = 0;

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

if (i != \_n) {

d[q] = buf[i];

++q;

}

}

system("cls");

delete[]buf;

cout << "Данные удалены!" << endl;

}

else

cout << "Номер введён неверно!" << endl;

}

В коде также присутствует строка "delete[]t", которая удаляет все данные в массиве структур. Это может использоваться для полного удаления всех данных из файла.

size = 0;

delete[]t;

t = new Client[size];

3. Выводы:

Редактирование файлов — это важный инструмент для управления данными в базе данных. В данном коде представлены функции "Change" и "del", которые позволяют изменять и удалять записи в массиве структур "Client".

Для успешного редактирования файлов необходимо использовать правильные методы удаления и изменения данных. В данном коде используются циклы и условные операторы для поиска и изменения или удаления нужных данных.

Задание к лабораторной работе:

Необходимо написать функции для удаления данных из массива структур вашей базы данных (лабораторная работа 2), а также функцию для полной очистки всего массива. Также нужно создать функцию для изменения данных в массиве структур.

Программный код:

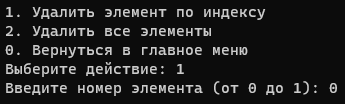
void Change(Client\* (&t), int n, int l)

void del(Client\* (&d), int& n)

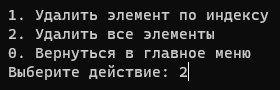
delete[]t;

Скриншоты выполнения программы:

Удаление клиента с нулевым индексом



Удаление всех клиентов



Дополнительные задания:

1 задание: Поиск дубликатов в массиве структур.

Создайте программу на C++, которая анализирует массив структур и находит дубликаты записей. Дубликатами считаются записи с одинаковыми значениями в ключевых полях (например, одинаковый идентификационный номер или имя). При нахождении дубликатов программа должна выводить их на экран и предлагать пользователю удалить повторяющиеся записи.

2 задание: Напишите программу на языке C++, которая открывает текстовый файл, заменяет все вхождения определенного слова на другое слово и сохраняет изменения в тот же файл.

3 задание: Напишите программу на C++, которая открывает файл для чтения, изменяет заданное поле в каждой строке файла, и сохраняет измененный файл под тем же именем.

Контрольные вопросы:

1. Каким образом осуществляется редактирование файлов, таких как изменение и удаление полей записей, в языке программирования C++?

2. Какие особенности работы с файлами на языке программирования C++ следует учесть при редактировании данных, таких как изменение и удаление полей записей?

3. Как обеспечить корректное сохранение и восстановление данных в файле после изменения или удаления полей записей на языке программирования C++?

# **Лабораторная работа 7**

**Тема: "Сортировка записей"**

**Цель: Изучение методов удаления и изменения данных в файле и их реализация на практике.**

1. Теоретическая часть:

Запись/чтение массива структур в файл

Запись и чтение массива структур в файл — это процесс сохранения и восстановления данных структур в файле на жестком диске компьютера. При записи массива структур в файл, каждая структура сохраняется последовательно в файле. При чтении массива структур из файла, данные структур читаются последовательно из файла и сохраняются в массив.

Для записи и чтения данных из файла в C++ используются потоки ввода-вывода (fstream). При записи данных в файл, используется оператор <<, а при чтении данных из файла - оператор >>.

Редактирование файлов: удаление, изменение поля

Редактирование файлов — это процесс изменения или удаления данных в файле. Для удаления данных из файла необходимо найти номер строки, которую необходимо удалить, создать временный массив без этой строки и записать временный массив в файл. Для изменения поля данных в файле необходимо найти номер строки, изменить поле и записать массив в файл.

#include <iostream>

#include <fstream>

using namespace std;

struct Record {

string name;

int age;

};

int main() {

const string fileName = "data.txt";

const int numRecords = 3;

Record records[numRecords];

// Записываем структуры в файл

ofstream outFile(fileName);

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

Record record = {"John", 25 + i};

outFile << record.name << " " << record.age << endl;

}

outFile.close();

// Читаем структуры из файла

ifstream inFile(fileName);

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

inFile >> records[i].name >> records[i].age;

}

inFile.close();

// Выводим значения

cout << "Before editing:" << endl;

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

cout << records[i].name << " " << records[i].age << endl;

}

// Удаляем элемент из файла

int numToDelete = 1;

Record tempRecords[numRecords - 1];

int j = 0;

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

if (i != numToDelete) {

tempRecords[j++] = records[i];

}

}

ofstream outFile2(fileName);

for (int i = 0; i < numRecords - 1; i++) {

outFile2 << tempRecords[i].name << " " << tempRecords[i].age << endl;

}

outFile2.close();

// Изменяем поле и записываем обновленный массив в файл

int numToEdit = 0;

records[numToEdit].age = 30;

ofstream outFile3(fileName);

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

outFile3 << records[i].name << " " << records[i].age << endl;

}

outFile3.close();

// Читаем обновленные структуры из файла

ifstream inFile2(fileName);

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

inFile2 >> records[i].name >> records[i].age;

}

inFile2.close();

// Выводим значения

cout << "After editing:" << endl;

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

cout << records[i].name << " " << records[i].age << endl;

}

return 0;

}

2. Сортировка записей

Сортировка записей — это процесс упорядочивания данных в файле в определенном порядке. Существует несколько методов сортировки массивов: Selection sort, Bubble sort, Insertion sort, Merge sort, Quick sort.

Selection sort — это простой алгоритм сортировки, который находит минимальный элемент из оставшейся части массива и перемещает его в начало неотсортированной части массива. Он продолжает делать это до тех пор, пока весь массив не будет отсортирован. Время выполнения этого алгоритма - O(n^2).

void selectionSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

if (minIndex != i) {

swap(arr[i], arr[minIndex]);

}

}

}

Bubble sort — это алгоритм сортировки, который сравнивает пары соседних элементов и меняет их местами, если они находятся в неправильном порядке. Алгоритм продолжает проходить по массиву до тех пор, пока все элементы не будут отсортированы. Время выполнения этого алгоритма - O(n^2).

void bubbleSort(int arr[], int n) {

bool swapped = true;

while (swapped) {

swapped = false;

for (int i = 0; i < n - 1; i++) {

if (arr[i] > arr[i + 1]) {

swap(arr[i], arr[i + 1]);

swapped = true;

}

}

}

}

Insertion sort — это алгоритм сортировки, который проходит по массиву и каждый элемент вставляет в правильное место в уже отсортированной части массива. Алгоритм продолжает делать это, пока весь массив не будет отсортирован. Время выполнения этого алгоритма - O(n^2).

void insertionSort(int arr[], int n) {

for (int i = 1; i < n; i++) {

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

Merge sort — это алгоритм сортировки, который использует стратегию "разделяй и властвуй". Он делит массив на две части, рекурсивно сортирует каждую часть и затем сливает их вместе. Время выполнения этого алгоритма - O(n log n).

void merge(int arr[], int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for(int i = 0; i < n1; i++) {

L[i] = arr[l+i];

}

for(int j = 0; j < n2; j++) {

R[j] = arr[m+1+j];

}

int i = 0;

int j = 0;

int k = l;

while(i < n1 && j < n2) {

if(L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while(i < n1) {

arr[k] = L[i];

i++;

k++;

}

while(j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r) {

if(l < r) {

int m = l + (r-l)/2;

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

Quick sort — это алгоритм сортировки, который также использует стратегию "разделяй и властвуй". Он выбирает элемент, называемый опорным, и перемещает все элементы, меньшие опорного, перед ним, а все элементы, большие опорного, после него. Затем алгоритм рекурсивно применяется к двум подмассивам. Время выполнения этого алгоритма - O(n log n).

void quickSort(int arr[], int left, int right) {

int i = left, j = right;

int pivot = arr[(left + right) / 2];

while (i <= j) {

while (arr[i] < pivot) {

i++;

}

while (arr[j] > pivot) {

j--;

}

if (i <= j) {

std::swap(arr[i], arr[j]);

i++;

j--;

}

}

if (left < j) {

quickSort(arr, left, j);

}

if (i < right) {

quickSort(arr, i, right);

}

}

3. Заключение

В ходе выполнения данной лабораторной работы будут изучены различные методы работы с массивами структур данных. Будут реализованы операции сортировки элементов в массиве структур данных.

Будут изучены различные методы сортировки массивов, такие как Selection sort, Bubble sort, Insertion sort, Merge sort и Quick sort.

Результатом выполнения лабораторной работы является полноценный набор функций для работы с массивами структур данных, что позволяет выполнять различные операции с данными и удобно хранить их в файлах. Реализация данных функций может быть использована в различных приложениях, работающих с большим объемом информации.

4. Выводы о скорости каждого из алгоритмов:

Сравниваем пять алгоритмов сортировки: выбором (Selection sort), пузырьковый (Bubble sort), вставками (Insertion sort), слиянием (Merge sort) и быстрая сортировка (Quick sort). Сравнение проводится на основе их среднего времени выполнения и асимптотической сложности.

1. Selection sort (сортировка выбором):

Среднее время выполнения: O(n^2)

Самый медленный алгоритм сортировки в данном списке. Он неэффективен на больших наборах данных.

2. Bubble sort (пузырьковая сортировка):

Среднее время выполнения: O(n^2)

Это также медленный алгоритм сортировки. Он не рекомендуется для больших наборов данных.

3. Insertion sort (сортировка вставками):

Среднее время выполнения: O(n^2)

Данный алгоритм работает немного быстрее на частично отсортированных данных, но в целом имеет квадратичную сложность и не подходит для больших наборов данных.

4. Merge sort (сортировка слиянием):

Среднее время выполнения: O(n\*log(n))

Это эффективный алгоритм сортировки с гарантированной логарифмической сложностью. Он стабилен и хорошо работает на больших наборах данных, но требует дополнительной памяти для хранения промежуточных результатов.

5. Quick sort (быстрая сортировка):

Среднее время выполнения: O(n\*log(n))

Быстрая сортировка - один из самых быстрых алгоритмов сортировки в данном списке. Он имеет среднее время выполнения O(n\*log(n)) и работает на месте (in-place), что снижает требования к памяти. Однако в худшем случае время выполнения может увеличиться до O(n^2).

Вывод: Самым быстрым алгоритмом сортировки является быстрая сортировка (Quick sort), а самыми медленными являются сортировка выбором (Selection sort), пузырьковая сортировка (Bubble sort) и сортировка вставками (Insertion sort). Сортировка слиянием (Merge sort) также является быстрым алгоритмом, но требует дополнительной памяти для хранения промежуточных результатов.

Задание к лабораторной работе:

Используя свою базу данных из 2 лабораторной работы, нужно реализовать сортировку данных по каким-то параметрам. Можно использовать любую сортировку, но обязательно должно быть не менее двух параметров для сортировки, а также не менее трёх видов сортировок.

Программный код:

void InsertSort(Client\* (&t), int n)

void BubbleSort(Client\* (&t), int n)

void SelectionSort(Client\* (&t), int n)

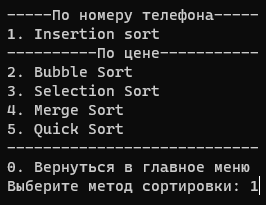
void Merge(Client\* (&t), int left, int mid, int right)

void MergeSort(Client\* (&t), int left, int right)

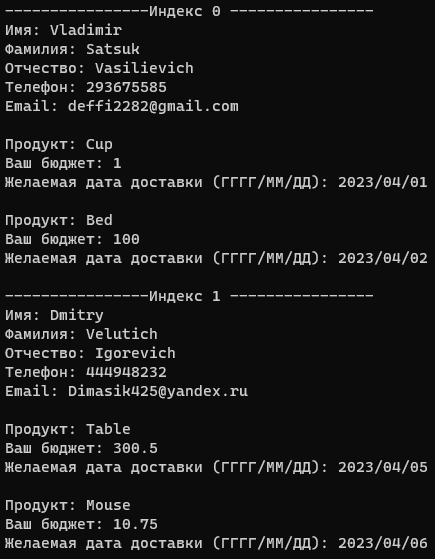
void QuickSort(Client\* (&t), int low, int high)

Скриншоты выполнения программы:

Выбор метода сортировки



Результат сортировки



Дополнительные задания:

1 задание: Сравнение производительности разных алгоритмов сортировки.

Реализуйте в вашей программе на C++ несколько алгоритмов сортировки данных (например, сортировка пузырьком, быстрая сортировка, сортировка слиянием). Пользователь должен иметь возможность выбирать алгоритм сортировки для каждой операции. Программа должна замерять время выполнения каждой операции сортировки и выводить результаты на экран, чтобы пользователь мог сравнить производительность разных алгоритмов.

2 задание: Напишите программу на языке C++, которая считывает набор записей из файла и сортирует их по возрастанию значения определенного поля. Отсортированные записи затем сохраняются в тот же файл.

3 задание: Напишите программу на языке C++, которая считывает несколько строк из текстового файла, сортирует их по алфавиту и выводит отсортированный список строк в новый файл.

Контрольные вопросы:

1. Какие основные методы сортировки массивов существуют, и каковы их характеристики с точки зрения времени выполнения?

2. Какие особенности имеют сортировки Merge sort и Quick sort по сравнению с другими алгоритмами сортировки, и каковы их преимущества?

3. В каких ситуациях стоит использовать каждый из алгоритмов сортировки (Selection sort, Bubble sort, Insertion sort, Merge sort, Quick sort) для упорядочивания данных в массиве?

# **Лабораторная работа 8**

**Тема: "Фильтрация данных"**

**Цель: Изучение методов фильтрации данных.**

В процессе выполнения лабораторной работы будет реализована функция FindStr, которая осуществляет поиск клиента в списке по имени.

Основной метод фильтрации данных, реализованный в данной функции — это поиск подстроки в строке. Для этого мы последовательно сравниваем символы строки клиента и искомой подстроки.

Если встречаем несовпадение, то сбрасываем счетчик и начинаем сравнение заново. Если находим совпадение, то увеличиваем счетчик. Если длина найденной подстроки совпадает с длиной искомой подстроки, то выводим информацию о клиенте.

При фильтрации данных необходимо учитывать особенности хранения данных и выбрать оптимальный алгоритм для конкретной задачи. В данном случае, для поиска по имени клиента мы выбрали метод последовательного сравнения символов, что позволяет искать клиентов не только по полному соответствию имени, но и по его части.

bool FindStr(string str, string sub\_str) {

int str\_len = str.length();

int sub\_str\_len = sub\_str.length();

int count = 0;

for (int i = 0; i < str\_len; i++) {

if (str[i] == sub\_str[count]) {

count++;

} else {

count = 0;

}

if (count == sub\_str\_len) {

return true;

}

}

return false;

}

Также, при фильтрации данных необходимо учитывать возможные ошибки при вводе пользователем данных и обрабатывать их, чтобы не происходило непредвиденных сбоев в работе программы.

Таким образом, реализуем функцию FindStr, осуществляющую поиск клиента в списке по имени, и изучим методы фильтрации данных, необходимые для решения данной задачи.

void FindStr(Client\* (&a), int n)

{

string Name;

cout << "Введите имя клиента: ";

GetStringValueFromUser(Name);

system("cls");

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

{

int l = a->Name.size();

int l2 = Name.size();

int fg = 0;

string x = a[i].Name;

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

if (j + l2 > l + fg)break;

if (Name[fg] != x[j])

{

fg = 0;

continue;

}

fg++;

if (fg == l2)

{

cout << "**\n**Имя: " << a[i].Name

<< "**\n**Фамилия: " << a[i].Fam

<< "**\n**Отчество: " << a[i].Otch

<< "**\n**Телефон: " << a[i].TelephoneNumber

<< "**\n**Email: " << a[i].Email;

cout << endl;

break;

}

}

}

}

Задание к лабораторной работе:

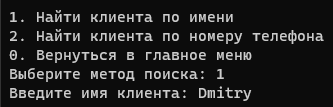
Для базы данных, которую вы создали во 2 лабораторной работе, необходимо реализовать функцию поиска подстроки в строке, например, по фамилии. Эта функция позволит фильтровать данные в базе данных по заданной подстроке и выводить только те записи, которые соответствуют заданному критерию поиска.

Программный код:

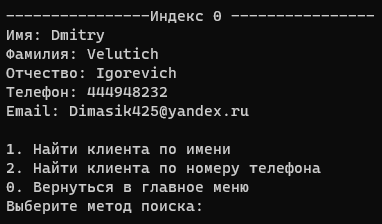
void FindStr(Client\* (&a), int n)

Скриншоты выполнения программы:

Ищем клиента с именем Dmitry



Вывод исходного результата



Дополнительные задания:

1 задание: Реализация расширенного поиска по нескольким полям.

Разработайте программу на C++, которая позволит пользователю осуществлять расширенный поиск в массиве структур по нескольким полям одновременно. Пользователь должен иметь возможность вводить значения для каждого поля и выбирать условие соответствия (например, точное совпадение, частичное совпадение, совпадение по началу или концу строки). Программа должна выводить на экран все записи, удовлетворяющие заданным критериям поиска.

2 задание: Напишите программу на языке C++, которая считывает данные из текстового файла, содержащего список чисел, и выводит только те числа, которые делятся на 2 без остатка.

3 задание: Напишите программу на языке C++, которая считывает данные из текстового файла, содержащего список имен и возрастов людей, и выводит только те записи, где возраст больше заданного значения.

Контрольные вопросы:

1. Какие другие методы фильтрации данных могут быть использованы для поиска клиентов в списке, и какие преимущества и недостатки они имеют по сравнению с методом последовательного сравнения символов?

2. Как обеспечить обработку ошибок при вводе данных пользователем, и какие типы ошибок могут возникнуть в процессе использования функции FindStr?

3. В каких ситуациях метод последовательного сравнения символов для поиска подстроки в строке может быть неэффективным, и какие альтернативные методы могут быть использованы для улучшения производительности поиска?

# **Лабораторная работа 9**

**Тема: "Поиск записи по значению/индексу"**

**Цель: Разработать программу, которая позволяет осуществлять поиск записей в массиве структур по значению и индексу.**

Теоретические знания:

Поиск записей в массиве является важной задачей в информатике. Существуют различные методы поиска, каждый из которых имеет свои преимущества и недостатки.

Интерпретирующий поиск можно реализовать с помощью цикла и условных операторов в языке программирования C++.

Пример кода поиска элемента в массиве с использованием интерпретирующего поиска на языке C++:

#include <iostream>

using namespace std;

int main()

{

int array[10] = {3, 7, 10, 14, 23, 29, 36, 44, 51, 60};

int search = 29;

bool found = false;

for (int i = 0; i < 10; i++)

{

if (array[i] == search)

{

found = true;

cout << "Element " << search << " found at index " << i << endl;

break;

}

}

if (!found)

{

cout << "Element not found in the array" << endl;

}

return 0;

}

В данном коде задается массив из 10 элементов. Ищем элемент со значением 29. В начале цикла поиска создаем переменную found (найден), которая в начале равна false (ложно). Затем проходимся по всем элементам массива и если находим элемент, равный значению search, то устанавливаем значение found в true, выводим сообщение об успешном нахождении элемента и прерываем цикл. Если цикл закончился, а элемент не был найден, то выводится сообщение о неудаче поиска.

Интерполяционный поиск (Interpolation search) является оптимизированной версией бинарного поиска, который позволяет быстрее находить элементы в упорядоченном массиве при равномерном распределении значений.

Интерполяционный поиск работает следующим образом:

1. Ищем элемент по формуле: pos = lo + ((hi - lo) / (arr[hi] - arr[lo])) \* (x - arr[lo]), где lo - индекс первого элемента в массиве, hi - индекс последнего элемента в массиве, x - искомый элемент.

2. Если arr[pos] == x, то элемент найден.

3. Если arr[pos] < x, то ищем в правой части массива.

4. Если arr[pos] > x, то ищем в левой части массива.

5. Повторяем пункты 1-4, пока не найдем элемент.

int interpolationSearch(int arr[], int size, int x) {

int lo = 0;

int hi = size - 1;

int pos;

while (lo <= hi && x >= arr[lo] && x <= arr[hi]) {

pos = lo + ((hi - lo) / (arr[hi] - arr[lo])) \* (x - arr[lo]);

if (arr[pos] == x) {

return pos;

} else if (arr[pos] < x) {

lo = pos + 1;

} else {

hi = pos - 1;

}

}

return -1;

}

Поиск по индексу:

Для поиска элементов по индексу в массиве используют прямой доступ по индексу. Это означает, что элемент находится в массиве по определенному индексу, и чтобы его найти, нужно просто обратиться к этому индексу.

Задание к лабораторной работе:

Согласно структуре вашей базы данных, которую вы создали во 2-ой лабораторной работе, нужно реализовать алгоритм интерполирующего поиска индекса в массиве структур. Также необходимо реализовать бинарный поиск по любому из типов данных, например, по фамилии, если в базе данных есть соответствующее поле.

Программный код:

int InterpolationSearch(Client\* (&t), int size, int x)

{

int lo = 0, hi = size - 1;

int range = t[hi].TelephoneNumber - t[lo].TelephoneNumber;

while (lo < hi) {

int pos = lo + ((double)(hi - lo) / range) \* (x - t[lo].TelephoneNumber);

if (t[pos].TelephoneNumber == x)

return pos;

if (t[pos].TelephoneNumber < x)

lo = pos + 1;

else

hi = pos - 1;

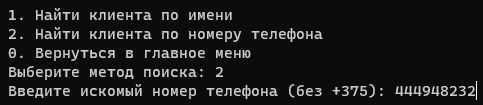
}

return (t[lo].TelephoneNumber == x) ? lo : -1;

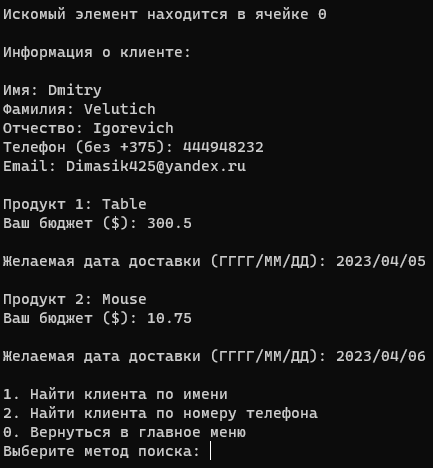
}

Скриншоты выполнения программы:

Ищем клиента с номером телефона ~~+375~~ 444948232



Вывод работы программы



Дополнительные задания:

1 задание: Сравнение производительности различных алгоритмов поиска.

Добавьте в вашу программу на C++ функционал для сравнения производительности различных алгоритмов поиска (например, линейный поиск, бинарный поиск, интерполяционный поиск, поиск с использованием хэш-таблиц или дерева поиска). Пользователь должен иметь возможность выбирать алгоритмы для сравнения и указывать параметры для поиска. Программа должна замерять время выполнения каждой операции поиска и выводить результаты на экран или сохранять их в файл по выбору пользователя.

2 задание: Напишите программу на языке C++, которая считывает список имен и возрастов людей из текстового файла и запрашивает у пользователя имя человека для поиска. Программа выводит возраст этого человека, если он найден в списке.

3 задание: Напишите программу на языке C++, которая считывает данные из текстового файла, содержащего список чисел, и позволяет пользователю искать записи по индексу. Если индекс найден, программа выводит соответствующее число, в противном случае выводится сообщение об ошибке.

Контрольные вопросы:

1. Какие преимущества и недостатки имеет интерполяционный поиск по сравнению с другими методами поиска, такими как бинарный или линейный поиск, и в каких ситуациях его использование будет наиболее эффективным?

2. Каковы условия и ограничения применимости интерполяционного поиска, и какие факторы могут повлиять на его производительность, например, структура и размер массива, а также характер данных в нем?

3. В каких случаях использование поиска по индексу является наиболее предпочтительным способом поиска элементов в массиве, и какие альтернативные методы могут быть применены для оптимизации процесса поиска, если прямой доступ по индексу не является возможным или неэффективным?