Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

Дисциплина: Основы Алгоритмизации и Программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

АЛГОРИТМЫ ПОИСКА И СОРТИРОВКИ В МАССИВАХ

Студент гр. 321702 Кислицын И.А.

Руководитель: Семижон Е.А.

Минск

2024

**Пояснительная записка к курсовой работе на тему алгоритмы поиска и сортировки в массивах: пояснительная записка к курсовой работе / Кислицын И.А. – Минск: БГУИР, 2024. — 34с**

В курсовой работе представлено описание работы алгоритмов поиска и сортировки в массивах на примере программы. Для данной курсовой работы пример выполнения работы программы выполняется на ведомости абитуриентов. Предложены различные решения для данной задачи.

Материал предназначен всем, кто интересуется принципом работы и реализацией алгоритмов и структур данных.

Автор: И.А.Кислицын

Рецензенты:

ассистент кафедры вычислительных методов и программирования учреждения образования «Белорусский государственный университет информатики и

радиоэлектроники» Е.А.Семижон.

СОДЕРЖАНИЕ

Введение……………………………………………………………………………5

1.Структуры и файлы…………………………………………………………..….6

2.Алгоритмы сортировки…………………………………………………..….…..7

2.1 Общие сведения…………………………………………………………7

2.2 Метод прямого обмена…………………………………………………7

2.3 Быстрая сортировка……………………………………………………..8

3.Алгоритмы поиска……………………………………………………………..11

3.1 Общие сведения……..………………………………………………...11

3.2 Линейный поиск …..…………………………………………………..11

3.3 Бинарный поиск ………………………..……………….……………13

4.Пользовательские функции…………………………………………….….…..15

4.1 Общие сведения ……………..………………………………………..15

4.2 Функция создания файла………… …………………………………...15

4.3 Функция заполнения данными файла………………………………..15

4.4 Функция просмотра данных файла…………………………………..16

4.5 Функция удаления данных файла …………………………………....16

4.6 Функция сортировки прямым выбором……………………………...17

4.7 Функция быстрой сортировки…….… ………....…………………….17

4.8 Функция бинарного поиска…………………….......……...................18

5.Описание работы программы………………………………………………...19

6.Заключение……………………………………………………………….…......20

Список использованных источников …………………………………………...22

Приложение А........................ …………………………………………………...23

Приложение Б........................ …………………………………………………...34

ВВЕДЕНИЕ

В современном образовательном процессе автоматизация играет ключевую роль, обеспечивая эффективное управления информацией и улучшая процессы обработки данных. Одним из важных аспектов этой автоматизации является создание программ для обработки информации об абитуриентах.

**Цель:** разработка программы, способной обрабатывать ведомость абитуриентов. Программа будет предоставлять удобное хранение, редактирование и анализ данных.

**Объекты** **исследования:** алгоритмы поиска и сортировки для массивов, так как наиболее частыми операциями при работе с базами данных являются «поиск» и «сортировка». При обработке баз данных часто применяются такой тип данных, как структуры. База данных накапливается и хранится на диске в файле или создается пользователем. К ней часто приходится обращаться и обновлять. Элемент данных (структура) содержит ключевое поле (ключ), по которому его можно найти. Ключом может являться любое поле структуры, например, фамилия.

**Предметами исследования** являются следующие алгоритмы:

* Быстрая сортировка;
* Сортировка методом прямого обмена;
* Линейный поиск;
* Бинарный поиск.

Все вышеперечисленные алгоритмы применялись в процессе написания программы для различных целей.

В процессе написания курсового проекта использовались следующие **методы исследования**: анализ литературы, интернет-ресурсов и различных документов по выбранной теме.

1. СТРУКТУРЫ И ФАЙЛЫ

В процессе написания структуры мы имеем готовую ведомость абитуриентов.

Также пользователь может создать собственную ведомость и внести нужные в нее данные.

Список абитуриентов имеет такой вид:

Фамилия: Xxxxxxx, где Х —заглавная буква фамилии, x- любая прописная буква

Номер группы: XXXXXX, где Х — любая цифра

Город: Ddddddd, где D - заглавная буква, города, d любая прописная буква.

Средний балл: В, где В -любое средний балл

Во время выполнения программы данные абитуриентов хранятся в виде структуры **Vedomost**, содержащую следующие поля: string **name** для хранения Фамилии абитуриента,int **group** для хранения группы абитуриента, string **adress** для хранения данных о городе проживания абитуриента и double **average\_grade** для хранения среднего балла абитуриента**.**.

В процессе написания курсового проекта использовались разные стандартные библиотеки языка С++,такие как:

* **iostream** — для реализации консольного ввода/вывода;
* **vector** — для создания динамического массива;
* **fstream** — для реализации файлового ввода/вывода;
* **string** — для работы со строками;
* **iomanip** — для создания читабельного консольного вывода списка покупателей.
* **windows.h**— для создания эффекта загрузки (функция sleep()).

1. АЛГОРИТМЫ СОРТИРОВКИ

2.1 ОБЩИЕ СВЕДЕНИЯ

Программа позволяет выполнить два типа сортировки: метод прямого обмена и быстрая сортировка.

Каждый алгоритм сортирует ведомость абитуриентов по ключу, а именно по фамилии абитуриентов.

После выполнения сортировки список абитуриентов не меняет свой порядок, а выводит результат сортировок в консоль, для возможности

корректности выполнения других операций.

2.2 МЕТОД ПРЯМОГО ОБМЕНА

Алгоритм сортировки прямым обменом, также известный как сортировка пузырьком, является одним из самых простейших алгоритмов сортировки.Он работает путем многократного прохода по массиву, сравнивая соседние элементы и меняя их местами, если они стоят в неправильном порядке. Этот процесс повторяется до тех пор пока массив не будет отсортирован .

Принцип работы алгоритма сортировки пузырьком можно описать следующим образом:

1. Начнем сравнивать первый и второй элементы массива. Если они стоят в неправильном порядке(меньший элемент стоит после большего), то меняем их местами.
2. Переходим к следующей паре элементов (второй и третий) и снова сравниваем их меняя местами при необходимости.
3. Продолжаем этот процесс до конца массива, после чего самый большой элемент “всплывает” на правильную позицию в конце массива.
4. Повторяем все шаги для всех элементов массива, пока весь массив не будет отсортирован.

Алгоритм сортировки имеет сложность О(n^2) в худшем и средних случаях, что делает его не эффективным для больших массивов. Однако, он прост в реализации и может быть полезен для наборов данных или в учебных целях

В программе данный алгоритм используется для сортировки ведомости абитуриентов по фамилии (по алфавиту) и реализован в функции void dc\_sort(vector<string>arr).

Пример сортировки изображены на рис.1 и рис.2:

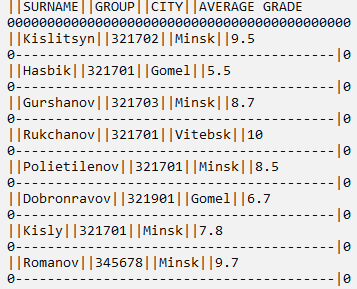


Рис. 1 «Список абитуриентов до сортировки методом прямого обмена»

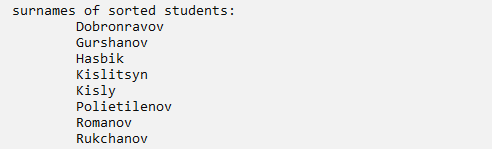


Рис. 2 «Вывод списка абитуриентов после сортировки методом прямого обмена»

2.3 БЫСТРАЯ СОРТИРОВКА

Быстрая сортировка является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена. Это эффективный алгоритм сортировки, который использует стратегию “разделяй и властвуй”. Он работает следующим образом

Общая идея алгоритма состоит в следующем:

1. Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. Обычно это средний элемент массива .

2. Массив разделяется на две подгруппы: одна содержит элементы, меньшие опорного, а вторая - элементы, большие опорного .

3. Рекурсивно применяется алгоритм к каждой из подгрупп.

4. Результаты сортировки объединяются в один отсортированный массив

5. Рекурсия останавливается, когда размер подмассива становится равным 1 или 0.

Быстрая сортировка быстро сортирует данные, особенно когда данные хорошо разделены на подмассивы при каждом шаге разделения. В лучшем случае его сложность составляет O(n log n).

В худшем случае (если опорный элемент выбирается неудачно), сложность алгоритма может составлять O(n^2) .

Алгоритм быстрой сортировки применяется, где требуется эффективная сортировка данных.

В программе алгоритм сортирует ведомость абитуриентов по их фамилиям и реализован в функциях

void quickSort(vector<string>& arr, int low, int high) и int partition(vector<string>& arr, int low, int high).

Пример работы алгоритма в программе изображён на рис. 3 и рис. 4:

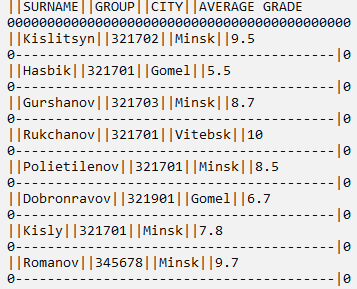


Рис. 3 «Список абитуриентов до быстрой сортировки»

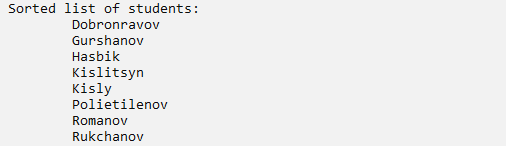


Рис. 4 «Вывод списка абитуриентов после быстрой сортировкиа»

3. АЛГОРИТМЫ ПОИСКА

3.1 ОБЩИЕ СВЕДЕНИЯ

Существует несколько основных алгоритмов поиска, каждый из которых имеет свои особенности и применение в различных ситуациях.

Рассмотрим некоторые из них:

1. **Линейный поиск** (Linear Search):

Описание: этот алгоритм прост и понятен. Он последовательно провяряет каждый элемент в списке, пока не будет найден искомый элемент или пока не будет пройден весь список.

Применение: Линейный поиск применяется при небольшом наборе даных.

1. **Бинарный поиск** (Binary Search)

Описание: Этот алгоритм работает на отсортированных списках. Он искомое значение с элементом в середине списка и, исходя из результата сравнения, продолжает поиск либо в левой , либо в правой половине списка.

Применение: Бинарный поиск эффективен на больших отсортированных наборах данных

Каждый из этих алгоритмов имеет свои преимущества и недостатки, и выбор конкретного алгоритма зависит от конкретной задачи и характеристик данных, с которыми мы работаем.

В программе использовались два алгоритма поиска: линейный и бинарный.

3.2 ЛИНЕЙНЫЙ ПОИСК

Алгоритм линейного поиска просматривает каждый элемент в последовательности данных по порядку, начиная с первого, пока не найдет целевой элемент или не достигнет конца последовательности.Если целевой элемент найден, то обычно возвращается его индекс.

При выполнении программы, алгоритм линейного поиска применяется для нахождения количества абитуриентов у которых средний балл выше 8.5 и которые проживают в Минске.

Как происхоидит выполнение алгоритсма линейного поиска в программе:

1. Берем первый элемент массива, который состоит из абитуриентов ведомости абитуриентов
2. Сравниваем город в котором проживает абитуриент и наш искомый город и также сравниваем средний балл абитуриента с нашим искомым средним баллом
3. Если условие 2 выполняется, то счетчик становится на единицу больше.

Для большей наглядности алгоритма так же выводим фамилии абитуриентов (рис. 5, рис. 6).

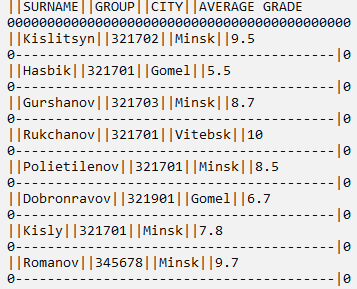


Рис. 5 «Список абитуриентов до линейного поиска»

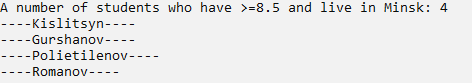


Рис. 6 «Вывод списка абитуриентов после линейного поиска»

3.3 БИНАРНЫЙ ПОИСК

Алгоритм бинарного поиска работает на отсортированном массиве данных. Он сравнивает целевой элемент с элементом в середине массива.

Процесс повторяется до тех пор, пока целевой элемент не будет найден или не останется элементов для проверки. Если элемент найден, возвращается его индекс. Иначе возвращается -1;

Сложность алгоритма бинарного поиска по времени выполнения O(logN) (так как мы уменьшаем срез массива на 2 на каждой итерации и проверяем только 1 элемент), и O(1) по памяти.

Алгоритм будет работать правильно, только с отсортированным массивом. Если по случайности забыть отсортировать массив перед его использованием, то в большинстве случаев тот ответ, который подсчитал алгоритм, будет неверным

Принцип работы бинарного поиска с ведомостью абитуриентов:

1. Сортируем список абитуриентов по алфавиту с помощью быстрой сортировки
2. Формируем границы списка в переменных left и right
3. Пока left<= right:
   1. Если текущий элемент равен искомому значение возвращаем индекс элемента
   2. Если текущий элемент меньше искомого значения присваеваем левой границе значение индекса текущего элемента и прибавляем единицу
   3. Если текущий элемент больше искомого значения присваеваем правой границе значение индекса текущего элемента и отнимаем единицу.Если ничего не выполнилось из вышеперечисленного, то возвращаем -1, то есть такого элемента в списке нет.

Пример работы алгоритма бинарного поиска на рис.7 и на рис.8:

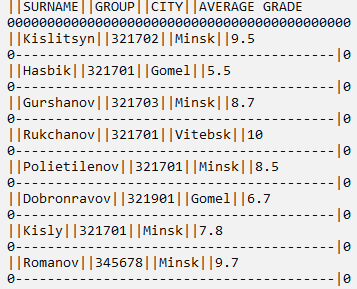


Рис. 7 «Список абитуриентов до бинарного поиска»

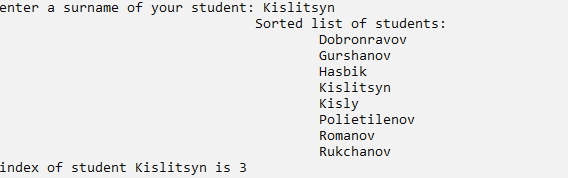


Рис. 8 «Вывод списка абитуриентов после бинарного поиска»

4. ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ

* 1. ОБЩИЕ СВЕДЕНИЯ

Пользовательские функции в программировании - это блоки кода, которые создаются пользователем для выполнения определенной задачи или набора задач. Они позволяют упростить код, избежать дублирования и повторного использования одного и того же кода в разных частях программы

* 1. ФУНКЦИЯ СОЗДАНИЯ ФАЙЛА

Функция void createFile(const string& filename) предназначена ддя создания файла с именем, которое функция принимает как аргумент.

Пример работы функции представлен на рис.9.

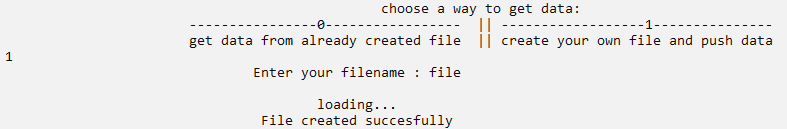


Рис.9 «Пример работы функции создания файла»

* 1. ФУНКЦИЯ ЗАПОЛНЕНИЯ ДАННЫМИ ФАЙЛА

Функция void addTofFile( const string&filename, Vedomost& data) предназначена для заполнения данными файла. Аргументы функции: имя файла, данные(ведомость абитуриентов).

Пример работы функции представлен на рис.10.

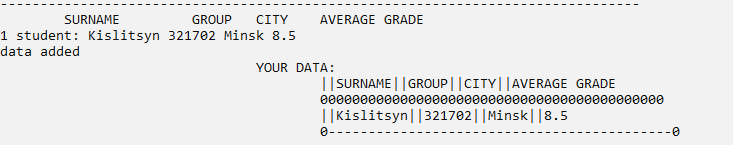


Рис.10 «Пример работы функции заполнения данными файла»

4.4 ФУНКЦИЯ ПРОСМОТРА ДАННЫХ ФАЙЛА

Функция void looakAt( const string& filename) предназначена для просмотра данных файла. Аргумент функции - имя файла.

Пример работы функции представлен на рис.11.

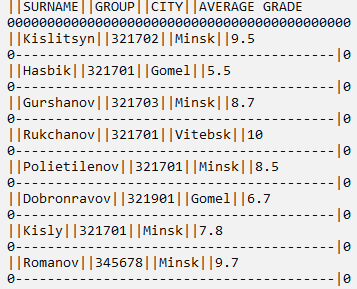


Рис.11 «Пример работы функции просмотра данных файла»

4.5 ФУНКЦИЯ УДАЛЕНИЯ ДАННЫХ В ФАЙЛЕ

Функция int delete\_from\_file(const string& filename,string toDel) предназначена для удаления определенных данных файла.Аргументы функции: имя файла, данные предназначеные для удаления, которые вводит пользователь (имя абитуриента).

Пример работы функции представлен на рис.12 .

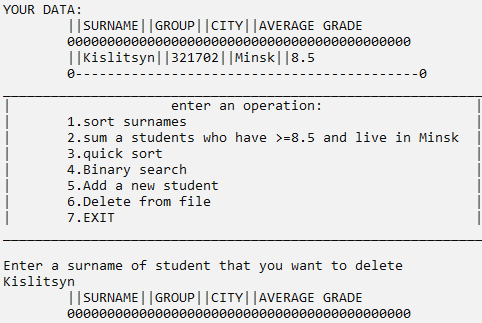


Рис.12 «Пример работы функции удаления данных файла»

4.6 ФУНКЦИЯ СОРТИРОВКИ ПРЯМЫМ ВЫБОРОМ

Функция void dc\_sort(vector<string>arr). предназначена для выполнения операции сортировки “пузырьком”. Аргументом данной функции является массив абитуриентов. Функция возвращает тот же массив, который на данном этапе будет отсортированный.

Более подробное описание задачи и принцип работы функции находится в главе 2.2.

4.7 ФУНКЦИЯ БЫСТРОЙ СОРТИРОВКИ

Функция void quickSort(vector<string>& arr, int low, int high). предназначена для выполнения операции быстрой сортировки. Аргументами данной функции является массив абитуриентов, некоторая левая и правая части половина массива абитуриентов. Функция возвращает тот же массив, который на данном этапе будет отсортированный. Функции необходимы последние два аргумента, так как принцип работы быстрой сортировки основан на ее рекурсивном вызове, что обеспечивает хорошее понимание и чтение кода.

Более подробное описание задачи и принцип работы функции находится в главе 2.3.

4.8 ФУНКЦИЯ БИНАРНОГО ПОИСКА

Функция int binary\_search(const string&filename, string& target) предназначена для выполнения операции бинарного поиска в массиве абитуриентов. Аргументами данной функции являются имя файла из которого берутся данные об абитуриентах и фамилия того абитуриента, которого необходимо обнаружить. Перед выполнением функции binary\_search необходимо отсортировать массив данных, так как алгоритм выполнения операции бинарного поиска заключается в разделения массива на две части и дальнейшем анализе конкретной части, а именно её элементов.Функция возвращает индекс абитуриента, если он был обнаружен в списке данных, в противном случае, возвращается нуль.

Более подробное описание задачи и принцип работы функции находится в главе 3.3.

5.ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

Выполнение программы начинается с запроса пользователю о выборе места хранения данных о ведомости абитуриентов.Это может быть либо запрос данных с локального диска или создание собственных данных. После того, как пользователь заполнил нужные поля ввода, предлагается консольное меню с выбором операций :

1. Сортировать абитуриентов методом прямого выбора
2. Сосчитать количество абитуриентов, которые проживают в г.Минск и имеют средний балл больше 8.5 включая.
3. Сортировать абитуриентов с помощью алгоритма быстрой сортировки
4. Поиск абитуриента по фамилии с помощью алгоритма бинарного поиска
5. Добавление в список нового абитуриента
6. Удаление из списка абитуриента
7. Завершение программы

Если пользователь выбрал пункт 4, то необходимо ввести фамилию

абитуриента, индекс которого в отсортированном списке необходимо найти.

Если пользователь выбрал пункт 5, то необходимо ввести дополнительные

данные, такие как фамилия, номер группы, город и средний балл нового абитуриента.

Если пользователь выбрал пункт 6, то необходимо ввести фамилию

абитуриента, которого необходимо удалить из ведомости.

Программа не завершит своё выполнение до тех пор, пока пользователь не выберет операцию под номером 7 («Exit»).

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта была разработана программа для обработки ведомости абитуриентов. Целью данного проекта было создание инструмента, обеспечивающего удобное хранение, редактирование и анализ о поступающих.

Программа успешно реализует поставленные задачи, обеспечивая пользователям простой и интуитивно понятный интерфейс для работы с информацией об абитуриентах. Благодаря функционалу программы, пользователи могут эффективно управлять данными, проводить анализ и получать необходимую информацию в удобном формате.

Для того чтобы представить список абитуриентов в виде базы данных была разработана структура файла для ввода/вывода списка и удобного использования её в ходе выполнения программы. Структура подробно описана в главе 1.

Для операций с записями были использованы следующие алгоритмы сортировки:

* + - Сортировка методом прямого обмена — сортировка пузырьком обеспечивает сортировку списка(массива) данных с помощью простого прохождения вдоль всего списка, поочередно сравнивая каждый элемент с текущим.Этот алгоритм не является эффективным при больших объемах данных
    - Быстрая сортировка — сортировка, которая эффективно и корректно предоставляет пользователю отсортировать список(массив), с помощью метода “разделяй и властвуй”. При выполнении этой сортировки массив выбирается опорный элемент, относительно которого формируются две оставшиеся части из элементов которые имеют какое либо отношение к опорному (к примеру, у чисел это отношение может являться большим или меньшим).

Все алгоритмы сортировки и их применение были подробно описаны во 2 главе.

Использовались следующие алгоритмы поиска:

* Линейный поиск — это алгоритм поиска, который проходит вдоль всего списка(массива) элементов и ищет нужный пользователю. Обычно как результат выводится значение номера или индекса в списке, иначе выводится -1. Этот алгоритм на текущий момент не является эффективным и может использоваться только для небольших объемов данных;
* Бинарный поиск — это алгоритм поиска, благодаря которому осуществляется наиболее эффективный поиск элементов в отсортированном списке, в отличие от алгоритма линейного поиска. Этот алгоритм, как и быстрая сортировка, основан по принципу “разделяй и властвуй”, так как для его успешного выполнения необходимо чтобы список элементов был отсортирован, а дальше работает принцип выбора опорного элемента и разделения массива на две части. Обычно опорным элементом служит элемент в середине для более быстрого поиска “в две стороны”.

Все алгоритмы поиска и их применение были подробно описаны в 3 главе.

В 4 главе были подробно описаны цель и принцип работы различных функций для удобного построения и использования программы .

В 5 главе был описан процесс работы программы и описано её использование пользователем.

В процессе написания курсового проекта были подробно изучены алгоритмы поиска и сортировки, ситуации их оптимального использования. Все полученные знания были использованы на практике.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

**1.** Бхаргава А. Грокаем алгоритмы [Текст] / А. Бхаргава. - СПб.: Питер, 2017. - 288с. (дата обращения: 23.03.2024)

**2.** Binary tree [электронный ресурс] // URL: **<https://en.wikipedia.org/wiki/Binary_tree#Types_of_binary_trees>** (дата обращения 23.03.2024).

**3.** Быстрая сортировка [электронный ресурс] // URL: **[https://ru.wikipedia.org/wiki/Быстрая\_сортировка](https://ru.wikipedia.org/wiki/%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0)** (дата обращения 23.03.2024).

**4.** Binary Insertion Sort in C++ [электронный ресурс] // URL: **<https://www.tutorialspoint.com/binary-insertion-sort-in-cplusplus>** (дата обращения 23.03.2024).

**5.** Бинарный поиск [электронный ресурс] // URL :

<https://purecodecpp.com/archives/1977> (дата обращения 24.03.2024 )

ПРИЛОЖЕНИЕ А

ЛИСТИНГ КОДА

#include <iostream>

#include <string>

#include <vector>

#include<fstream>

#include <conio.h>

#include <windows.h>

#include <stdlib.h>

using namespace std;

struct Vedomost

{

//ФИО, номер группы, адрес, оценки.

string name;

int group;

string adress;

double average\_grade;

};

void createFile(const string& filename)

{

ofstream file(filename);

if (file.is\_open()) {

cout << "\t\t\t\t File created succesfully" <<'\n';

file.close();

}

else

{

cout << "FAILED " << '\n';

}

}

void addTofFile( const string&filename, Vedomost& data) {

ofstream file(filename,ios::app);

if (file.is\_open())

{

file << data.name <<' ' << data.group <<' ' << data.adress << ' ' << data.average\_grade << '\n';

cout << "data added" << '\n';

file.close();

}

else cout << "failed to add data" << '\n';

}

void looakAt( const string& filename)

{

std::ifstream file(filename,ios::binary);

if (file.is\_open())

{

Vedomost data;

cout << "\t\t\t\t\t||SURNAME||GROUP||CITY||AVERAGE GRADE \n";

cout << "\t\t\t\t\t0000000000000000000000000000000000000000000\n";

while ( file >> data.name>>data.group>>data.adress>>data.average\_grade)

{

cout << "\t\t\t\t\t";

cout << "||" << data.name << "||" <<

data.group << "||" <<

data.adress << "||" <<

data.average\_grade << '\n';

cout << "\t\t\t\t\t0-------------------------------------------0\n";

}

file.close();

}

}

int delete\_from\_file(const string& filename,string toDel) {

ifstream file(filename);

string line;

string fileContents;

if (file.is\_open())

{

Vedomost data;

bool isHaveName = false;

while (std::getline(file,line))

{

if (line.find(toDel)!=std::string::npos)

{

isHaveName = true;

continue;

}

fileContents += line + "\n";

}

file.close();

}

ofstream output(filename);

if(output.is\_open())

output << fileContents;

output.close();

looakAt(filename);

return 0;

}

///direct choice sort/////////////////////////////////////////////////

void dc\_sort(vector<string>arr)

{

for (int i = 0; i < arr.size(); i++)

{

for (int j = 0; j < arr.size() - i - 1; j++)

{

if (arr[j] >= arr[j + 1])

{

string temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

cout << "\t\t\t\tsurnames of sorted students:\n ";

for (string el : arr)

{

cout <<"\t\t\t\t\t" << el << '\n';

}

}

//// quick sort /////////////////////////////////////////////

void swap(string& a, string& b)

{

string temp = a;

a = b;

b = temp;

}

int partition(vector<string>& arr, int low, int high)

{

string pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++)

{

if (arr[j] <= pivot)

{

i++;

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

}

}

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

return (i + 1);

}

void quickSort(vector<string>& arr, int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

/////binary search////////////////////////////////////////////////

int binary\_search(const string&filename, string& target)

{

ifstream file(filename);

vector<string>names;

if (file.is\_open())

{

Vedomost data;

while (file >> data.name >> data.group >> data.adress >> data.average\_grade)

{

names.push\_back(data.name);

}

}

int left = 0;

int right = names.size() - 1;

quickSort(names, left, right);

cout << "\t\t\t\tSorted list of students:\n";

for (string stud : names)

{

cout << "\t\t\t\t\t" << stud << " ";

cout << '\n';

}

while (left <= right)

{

int mid = left + (right - left) / 2;

if (names[mid] == target)

{

return mid;

}

else if (names[mid] < target)

{

cout << names[mid] << ' ' << target;

left = mid + 1;

}

else

{

right = mid - 1;

}

}

return -1; // Имя не найдено

}

int main()

{

system("color F0");

// создание файла

bool inputstart = false;

cout << "\t\t\t\t\t\tchoose a way to get data: \n";

cout << "\t\t\t----------------0----------------- || ------------------1---------------\n";

cout << "\t\t\tget data from already created file || create your own file and push data\n "; cin >> inputstart;

switch (inputstart) {

case 0: {

cout << " enter a way to file: \n";

cout << '\n';

string way\_to\_file;

cin >> way\_to\_file;

cout << "\t\t\t\tprocess of searching for a file in the system...\n";

Sleep(2000);

cout << "\t\t\t\t\t||SURNAME||GROUP||CITY||AVERAGE GRADE \n";

cout << "\t\t\t\t\t0000000000000000000000000000000000000000000\n";

ifstream file(way\_to\_file);

Vedomost data;

while (file >> data.name >> data.group >> data.adress >> data.average\_grade) {

cout << "\t\t\t\t\t||" << data.name << "||" <<

data.group << "||" <<

data.adress << "||" <<

data.average\_grade << '\n';

cout << "\t\t\t\t\t0----------------------------------------|0\n";

}

bool flag = true;

while (flag) {

cout << "\t\t\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << '\n';

cout << "\t\t\t\t| enter an operation: |" << '\n';

cout << "\t\t\t\t|\t1.sort surnames\t\t\t\t\t |\n" <<

"\t\t\t\t|\t2.sum a students who have >=8.5 and live in Minsk |\n" <<

"\t\t\t\t| \t3.quick sort\t\t\t\t\t | \n" <<

"\t\t\t\t| \t4.Binary search\t\t\t\t\t | \n" <<

"\t\t\t\t| \t5.Add a new student\t\t\t\t | \n" <<

"\t\t\t\t| \t6.Delete from file\t\t\t\t | \n" <<

"\t\t\t\t| \t7.EXIT\t\t\t\t\t\t | \n";

cout << "\t\t\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << '\n';

int input = 0;

cin >> input;

switch (input)

{

case 1:

{

ifstream file(way\_to\_file);

if (file.is\_open())

{

int count = 0;

Vedomost student;

vector<string>students;

while (file >> student.name >> student.group >> student.adress >> student.average\_grade)

{

students.push\_back(student.name);

}

dc\_sort(students);

file.close();

}

break;

}

case 2:

{

int count = 0;

vector<string>students;

ifstream file(way\_to\_file);

if (file.is\_open())

{

Vedomost student;

while (file >> student.name >> student.group >> student.adress >> student.average\_grade)

{

if ((student.adress == "Minsk" || student.adress == "minsk") && student.average\_grade >= 8.5) {

count++;

students.push\_back(student.name);

}

}

}

file.close();

cout << "A number of students who have >=8.5 and live in Minsk: " << count << '\n';

for (string stud : students)

cout << "----" << stud << "----" << "\n";

break;

}

case 3:

{

vector<string> surname;

ifstream file(way\_to\_file);

if (file.is\_open())

{

Vedomost student;

while (file >> student.name >> student.group >> student.adress >> student.average\_grade)

{

surname.push\_back(student.name);

}

quickSort(surname, 0, surname.size() - 1);

cout << "\t\t\t\tSorted list of students:\n";

for (string stud : surname)

{

cout << "\t\t\t\t\t" << stud << " "; cout << '\n'; }

file.close();

}

break;

}

case 4:

{

cout << "enter a surname of your student: ";

string input\_name;

cin >> input\_name;

cout << "index of student " << input\_name << " is " << binary\_search(way\_to\_file, input\_name) << '\n';

break;

}

case 5:

{

cout << "\t\t\t\t" << "NAME\t" << "GROUP\t" << "CITY\t" << "AVERAGE GRADE\n";

Vedomost data;

cout << "\t\t\t\t";

cin >> data.name >> data.group >> data.adress >> data.average\_grade;

addTofFile(way\_to\_file, data);

looakAt(way\_to\_file);

break;

}

case 6:

{

cout << "\t\t\t\tEnter a surname of student that you want to delete" << '\n';

cout << "\t\t\t\t";

string todel;

cin >> todel;

delete\_from\_file(way\_to\_file, todel);

input = 10;

break;

}

case 7:

{

cout << "\t\t\t\tthe system is being logged out...\n";

Sleep(2000);

cout << "\t\t\t\t\tBYE!";

flag = false;

}

}

continue;

}

break;

}

case 1:

{

cout << "\t\t\t\tEnter your filename : ";

string filename = "some name";

cin >> filename;

cout << '\n' << "\t\t\t\t\tloading..." << '\n';

Sleep(3000);

createFile(filename);

// добавление в файл

bool flag = true;

cout << "\t\t\t\tEnter an amount of your students: ";

int n;

cin >> n;

cout << "\t\t\t\tEnter data of your " << n << " students\n";

cout << "--------------------------------------------------------------------------------\n";

cout <<"\t" << "NAME\t" << "GROUP\t" << "CITY\t" << "AVERAGE GRADE\n";

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

{

cout << i + 1 << " student: ";

Vedomost data;

cin >> data.name >> data.group >> data.adress >> data.average\_grade;

addTofFile(filename, data);

}

cout << "\t\t\t\tYOUR DATA: \n";

looakAt(filename);

while (flag)

{

cout << "\t\t\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << '\n';

cout << "\t\t\t\t| enter an operation: |" << '\n';

cout << "\t\t\t\t|\t1.sort surnames\t\t\t\t\t |\n" <<

"\t\t\t\t|\t2.sum a students who have >=8.5 and live in Minsk |\n" <<

"\t\t\t\t| \t3.quick sort\t\t\t\t\t | \n" <<

"\t\t\t\t| \t4.Binary search\t\t\t\t\t | \n" <<

"\t\t\t\t| \t5.Add a new student\t\t\t\t | \n" <<

"\t\t\t\t| \t6.Delete from file\t\t\t\t | \n" <<

"\t\t\t\t| \t7.EXIT\t\t\t\t\t\t | \n";

cout << "\t\t\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << '\n';

int input = 0;

cin >> input;

switch (input)

{

case 1:

{

ifstream file(filename);

if (file.is\_open())

{

int count = 0;

Vedomost student;

vector<string>students;

while (file >> student.name >> student.group >> student.adress >> student.average\_grade)

{

students.push\_back(student.name);

}

dc\_sort(students);

file.close();

}

break;

}

case 2:

{

int count = 0;

vector<string>students;

ifstream file(filename);

if (file.is\_open())

{

Vedomost student;

while (file >> student.name >> student.group >> student.adress >> student.average\_grade)

{

if ((student.adress == "Minsk"||student.adress=="minsk") && student.average\_grade >= 8.5)

{

count++;

students.push\_back(student.name);

}

}

}

file.close();

cout << "\t\t\tA number of students who have >=8.5 and live in Minsk: " << count << '\n';

cout <<"\t\t\t\t\t" << "( "; for (string stud : students)cout << stud << " "; cout << " )" << '\n';

break;

}

case 3: {

vector<string> surname;

ifstream file(filename);

if (file.is\_open()) {

Vedomost student;

while (file >> student.name >> student.group >> student.adress >> student.average\_grade)

{

surname.push\_back(student.name);

}

quickSort(surname, 0, surname.size() - 1);

cout << "\t\t\t\tSorted list of students:\n";

for (string stud : surname) { cout << "\t\t\t\t\t" << stud << " "; cout << '\n'; }

file.close();

}

break;

}

case 4:

{

cout << "\t\t\t\tenter a name of your student: ";

string input\_name;

cin >> input\_name;

cout << "\t\t\t\tindex of student " << input\_name << " is " << binary\_search(filename, input\_name) << '\n';

break;

}

case 5:

{

cout << "\t\t\t\t" << "NAME\t" << "GROUP\t" << "CITY\t" << "AVERAGE GRADE\n";

Vedomost data;

cout << "\t\t\t\t";

cin >> data.name >> data.group >> data.adress >> data.average\_grade;

addTofFile(filename, data);

looakAt(filename);

break;

}

case 6:

{

cout << "\t\t\t\tEnter a surname of student that you want to delete" << '\n';

cout << "\t\t\t\t";

string todel;

cin >> todel;

delete\_from\_file(filename, todel);

input = 10;

break;

}

case 7:

{

cout << "\t\t\t\tthe system is being logged out...\n";

Sleep(2000);

cout << "\t\t\t\t\tBYE!";

flag = false;

break;

}

}

continue;

}

break;

}

}

}

**ПРИЛОЖЕНИЕ Б**

**БЛОК-СХЕМА АЛГОРИТМА РАБОТЫ ПРОГРАММЫ**

