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 курсовому проекту

на тему:

**«Структура списка участников танцевального конкурса»**

Выполнил: гр.321701 Перминова В.В.

Проверила: Матюшкин С.И.

Минск 2024

**РЕФЕРАТ**

СТРУКТУРА СПИСКА УЧАСТНИКОВ ТАНЦЕВАЛЬНОГО КОНКУРСА: курсовой проект / В.В. Перминова – Минск: БГУИР, 2024, – п.з. – 40 с.

Kурсовая работа посвящена созданию структуры списка участников танцевального конкурса.

Цель работы: создание консольного приложения для управления данными об участниках.

Oбъект исследования: алгоритмы поиска и сортировки для списка участников танцевального конкурса.

Задачей данного курсового проекта является разработать программу для управления данными об участниках танцевального конкурса.

B курсовой работе проводится детальный разбор всех составляющих программы и описание работы программы в целом.

Данная работа содержит документацию, которая описывает работу программы, программный код и блок-схему, которая наглядно показывает работу основной функции программы. В документации представлены все основные средства, использованные при разработке программы, логика работы программы и наглядные примеры выполнения различных задач.

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc168007653)

[1. Структуры и файлы 6](#_Toc168007654)

[1.1. Описание структуры 6](#_Toc168007655)

[1.2. Описание работы с файлами 7](#_Toc168007656)

[2. Алгоритмы сортировки 8](#_Toc168007657)

[2.1. Метод прямого обмена (BubbleSort) 8](#_Toc168007658)

[2.2. Быстрая сортировка (QuickSort) 9](#_Toc168007659)

[3. Алгоритмы поиска 12](#_Toc168007660)

[3.1. Линейный поиск 12](#_Toc168007661)

[3.2. Бинарный поиск 13](#_Toc168007662)

[4. Пользовательские функции 14](#_Toc168007663)

[4.1. Функция для считывания данных из файла 14](#_Toc168007664)

[4.2. Функция для добавления данных в файл 14](#_Toc168007665)

[4.3. Функция удаления данных из файла 15](#_Toc168007666)

[4.4. Функция для просмотра содержимого файла 16](#_Toc168007667)

[4.5. Функции для сортировки методом прямого обмена по количеству участников 17](#_Toc168007668)

[4.6. Функция для обмена элементов массива 17](#_Toc168007669)

[4.7. Функции для разделения массива и возврата индекса опорного элемента 18](#_Toc168007670)

[4.8. Функции для реализации быстрой сортировки 18](#_Toc168007671)

[4.9. Функции для вызова сортировки методом QuickSort 19](#_Toc168007672)

[4.10. Функция для бинарного поиска по количеству участников 19](#_Toc168007673)

[4.11. Функция для линейного поиска по возрастной категории 20](#_Toc168007674)

[4.12. Функция для выполнения задания 21](#_Toc168007675)

[5. Описание работы программы 22](#_Toc168007676)

[Заключение 23](#_Toc168007677)

[Список использованных источников 24](#_Toc168007678)

[Приложение А (обязательное) Листинг кода 25](#_Toc168007679)

[Приложение Б Блок-схема работы программы 39](#_Toc168007680)

Введение

В современном мире мы имеем дело с огромным объёмом данных, в том числе, обладающих низким качеством. Поэтому обработка данных и их структурирование имеет важное значение. Это помогает извлекать ценную информацию из огромных объемов, улучшает качество данных, позволяет принимать обоснованные решения, автоматизировать рутинные задачи. Структурирование информации позволяет нам увидеть полную картину и выявить связи между элементами. Это повышает эффективность поиска и анализа, автоматизирует машинное обучение, облегчает визуализацию и понимание, а также обеспечивает безопасность и конфиденциальность. Поэтому в данной курсовой работе рассматривается разработка программного обеспечения на языке C++, которое автоматизирует некоторые процессы работы со списком участников танцевального конкурса, что должно упростить и ускорить решение многих административных задач.

Цель данной курсовой работы заключается в разработке консольного приложения на языке C++, которое позволит управлять списком участников танцевального конкурса. Приложение будет обладать следующими функциями: просмотр файла с данными о участниках, добавление новых записей, удаление необходимых коллективов, поиск опредёленных записей и выполнение сортировок.

В ходе данной работы исследовано применение файлов для организации и хранения данных коллективов. Рассмотрены различные операции с файлами, включая чтение, запись, поиск и удаление информации о участниках. Так же изучены и использованы на практике алгоритмы линейного и бинарного поиска. Для сортировки данных о коллективах-участниках, полученных из файла, изучена работа следующих алгоритмов: метод прямого обмена (BubbleSort) и быстрая сортировка (QuickSort). Эти алгоритмы так же использованы в работе.

Главная задача курсовой работы — разработать программу, которая будет давать возможность редактирования, просмотра и сортировки данных в файле со списком участников танцевального конкурса, поиска необходимых коллективов, а также будет выводить результаты на экран и сохранять их в файл.

В ходе работы улучшим навыки работы с файлами и структурами данных, изучим алгоритмы поиска и сортировки. Приобретенные знания и опыт также могут быть применены в различных сферах программирования, где необходима работа с данными и их анализ.

1. Структуры и файлы

Структуры данных — это способ организации информации для более эффективного использования. В программировании структурой обычно называют набор данных, связанных определённым образом. Структуры данных позволяют логически структурировать и группировать информацию, делая код более понятным и легко поддерживаемым. Они используются для упорядочивания, поиска, анализа и манипулирования данными с помощью алгоритмов программирования. Структуры данных оптимизируют доступ к данным и их обработку, что приводит к созданию более быстрого и эффективного программного обеспечения.

Файлы играют ключевую роль в разработке программного обеспечения, предоставляя возможности долгосрочного хранения данных, ввода-вывода, управления конфигурацией и обмена данными между приложениями. Они обеспечивают гибкость, удобство и расширяемость, позволяя программам считывать и записывать данные, сохранять настройки и обмениваться информацией с легкостью.

В целом, структуры данных и файлы обеспечивают эффективное управление информацией и являются важными составляющими разработки программного обеспечения.

* 1. Описание структуры

В данной курсовой работе для удобного хранения и управления данными о каждом коллективе была разработана структураParticipant*.*

Структура Participant имеет следующие поля:

1. name: данное поле является строковым типом (string) и предназначено для хранения названия коллектива.

2. nomination: данное поле является строковым типом (string) и предназначено для хранения названия номинации, в которой участвует коллектив.

3. age: данное поле является целочисленным типом (int) и предназначено для хранения возрастной категории.

4. quantity: данное поле является целочисленным типом (int) и предназначено для хранения количества участников в коллективе.

Структура Participantпредоставляет удобный формат для представления информации о коллективах-участниках танцевального конкурса. Каждый объект типа Participantсодержит данные о названии коллектива, номинации, в котором он участвует, возрастной категории и количестве участников. Это позволяет эффективно оперировать данными о коллективах и выполнять различные операции с ними. Преимущества использования структуры Participant включают организацию данных, удобство работы с данными, читаемость кода, расширяемость и гибкость, а также удобство передачи и использования.

* 1. Описание работы с файлами

Библиотека <fstream> на C++ позволяет вводить/выводить данные в файлы с помощью потоков. Она включает ifstream для чтения, ofstream для записи и fstream как для чтения, так и для записи. С помощью <fstream>, можно открывать файлы, выполнять чтение и запись данных, а также управлять файлами в бинарном или текстовом формате. [Это позволяет создавать программы, которые могут сохранять результаты, загружать данные и взаимодействовать с файловой системой](https://metanit.com/cpp/tutorial/8.2.php).

С использованием этой библиотеки в данной курсовой работе будет осуществляться работа с данными в файлах «list.txt» и «result.txt». В первом файле будет находится исходный список участников конкурса, а во второй будет выводится информация о коллективах, участвующих в определённой номинации, вводимой пользователем.

2. Алгоритмы сортировки

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент в списке имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

Сортировки в программировании помогают оптимизировать хранение данных, ускорять поиск информации, прогнозировать выполнение сложных операций и экономить ресурсы. Но при этом есть большой выбор алгоритмов сортировки разной сложности и скорости. У каждого свои плюсы и минусы. Определение оптимального алгоритма сортировки требует учета множества аспектов, включая объем и характеристики данных, необходимый уровень эффективности и специфику применения. В этом исследовании анализируются такие методы сортировки, как метод прямого обмена (BubbleSort) и быстрая сортировка (QuickSort), чтобы выявить их преимущества и ограничения в различных условиях.

2.1. Метод прямого обмена (BubbleSort)

Сортировка пузырьком (*Bubble Sort*) является простым алгоритмом сортировки, который используется для упорядочивания элементов в массиве или списке.

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим наибольшим элементом («всплывает» до нужной позиции, как пузырёк в воде), а наименьший элемент перемещается на одну позицию к началу массива.

Этот алгоритм считается учебным и почти не применяется на практике из-за низкой эффективности. Сложность данного алгоритма в худшем случае O(n2), в лучшем — O(n).

В программе данный алгоритм используется для сортировки списка танцевальных коллективов по количеству участников(как в порядке возрастания, так и в порядке убывания) и реализован в функциях void bubbleSortAsc(const string& filename)иvoid bubbleSortDesc(const string& filename). Пример сортировки изображены на рисунках 2.1 и 2.2:



Рисунок 2.1. Список коллективов до сортировки методом прямого обмена.



Рисунок 2.2. Список коллективов после сортировки методом прямого обмена.

2.2. Быстрая сортировка (QuickSort)

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы. Таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов.

Кратко алгоритм можно описать следующим образом:

1. На очередном шаге выбирается опорный элемент — им может быть любой элемент массива.
2. Все остальные элементы массива сравниваются с опорным и те, которые меньше него, ставятся слева от него, а которые больше или равны — справа.
3. Для двух получившихся блоков массива производится точно такая же операция — выделяется опорный элемент и всё идёт точно так же, пока в блоке не останется один элемент.

Сложность быстрой сортировки в среднем случае составляет O(n logn), где n — количество элементов в массиве. В худшем случае, когда выбирается плохой опорный элемент, сложность может быть O(n2), но вероятность его возникновения невысока.

В целом, *QuickSort* остается одним из самых популярных алгоритмов сортировки благодаря своей высокой производительности в большинстве сценариев.

В программе данный алгоритм используется для сортировки списка танцевальных коллективов по возрастной категории (как в порядке возрастания, так и в порядке убывания) и реализован в функциях void sortbyQuantityAsc(const string& filename)иvoid sortbyQuantityDesc(const string& filename).Пример работы алгоритма в программе изображён на рисунках 2.3 и 2.4:



Рисунок 2.3. Список коллективов до быстрой сортировки



Рисунок 2.4. Список коллективов после быстрой сортировки

Таким образом пузырьковая сортировка – это простой, интуитивно понятный алгоритм с временной сложностью O(n2), неэффективный для больших объемов данных. Быстрая сортировка – один из самых быстрых алгоритмов с временной сложностью O (n logn), использующий принцип "разделяй и властвуй" и эффективный для больших массивов. Именно быстрая сортировка обычно предпочтительнее для использования в программе из-за своей высокой эффективности и меньшего количества необходимых обменов элементов.

3. Алгоритмы поиска

Алгоритмы поиска играют важную роль в программировании и используются для поиска конкретных данных. Они помогают решать различные задачи, такие как поиск, определение наличия и нахождение минимальных/максимальных элементов. Эффективные алгоритмы поиска имеют решающее значение при работе с большими данными или базами данных, особенно когда данные упорядочены. Они играют ключевую роль в реализации других алгоритмов и структур данных, повышая производительность программ и сокращая время их выполнения. Знание различных алгоритмов поиска позволяет разработчикам выбирать наиболее эффективные решения, создавая оптимизированные программы для обработки данных и управления ими.

В данной курсовой работе были изучены два алгоритма поиска: линейный и бинарный. Самым простым из них является линейный поиск, который проверяет каждый элемент данных по порядку, пока не будет найден правильный элемент данных. Линейный поиск легко реализовать, но его эффективность снижается по мере увеличения объема данных. Именно здесь на помощь приходят более сложные алгоритмы (такие как бинарный поиск), которые значительно ускоряют процесс за счет разделения данных на более мелкие сегменты и устранения ненужных частей на каждом этапе.

* 1. Линейный поиск

Линейный поиск — это простой и интуитивно понятный метод поиска определенного элемента в списке, массиве или структуре данных.

Принцип работы линейного поиска: алгоритм линейного поиска проходит через все элементы массива (или списка) в последовательном порядке, начиная с первого элемента и заканчивая последним. В процессе он сравнивает каждый элемент с искомым значением.

Линейный поиск прост, понятен и может применяться к неупорядоченным наборам данных, но он неэффективен при работе с большими данными и занимает O(n) времени. Он подходит для небольших наборов данных или неупорядоченных данных, но для больших данных или данных с частым поиском могут быть предпочтительны более эффективные алгоритмы, такие как бинарный поиск.

В программе линейный поиск реализован в функции void linearSearch(const string& filename, int targetAge) для поиска танцевальных коллективов в определённой танцевальной категории.

* 1. Бинарный поиск

Бинарный поиск — процесс нахождения индекса элемента с целевым значением в отсортированном массиве путем его дробления на половину на каждом шаге новой итерации. Изначально алгоритм поиска сравнивает искомое значение со средним элементом в массиве. Если значения не равны, то он отбрасывает ту часть массива, в которой целевое значение гарантированно не может находиться. Далее поиск продолжается в оставшейся части элементов путем сравнения средних элементов с искомым значением до тех пор, пока оно не будет найдено, либо пока оставшаяся часть не станет пустой. В этом случае мы можем сказать, что элемент не найден.

Бинарный поиск имеет логарифмическую временную сложность O(logN). На больших массивах бинарный поиск работает быстрее линейного поиска, однако список изначально должен быть отсортирован.

Двоичный поиск часто используется в задачах, где важно быстро найти элемент в отсортированном списке, таких как базы данных, индексы или бинарные деревья. Благодаря своей эффективности он является одним из базовых алгоритмов, изучаемых в информатике и программировании.

В программе бинарный поиск реализован в функции void binarySearch(const string& filename, int targetQuantity) для поиска танцевальных коллективов с определённым количеством участников.

4. Пользовательские функции

Пользовательские функции — это функции, которые не являются встроенными и создаются самим пользователем во время работы с данными при необходимости получить желаемый результат. Такие функции обычно создаются для дополнительной обработки и могут содержать в себе несколько встроенных функций одновременно.

В данной программе определён ряд пользовательских функций, реализующих алгоритмы поиска, сортировки и выполнение прочих задач.

4.1. Функция для считывания данных из файла

vector<Participant> inputData(const string& filename), где filename *—* это файл со списком всех танцевальных коллективов. Данная функция открывает файл «list.txt», из которого она считывает информацию про коллективы в структуры. Список танцевальных коллективов она записывает в переменную list и возвращает её.

4.2. Функция для добавления данных в файл

void addDataToFile(const string& filename), где filename *—* это файл со списком всех танцевальных коллективов. Данная функция выводит в консоль просьбу ввести информацию о коллективе, который необходимо добавить, открывает файл «list.txt», в который записывает необходимую информацию, и выводит сообщение о том, что данные успешно добавлены в файл. Если файл закрыт, выводит соответствующее сообщение.

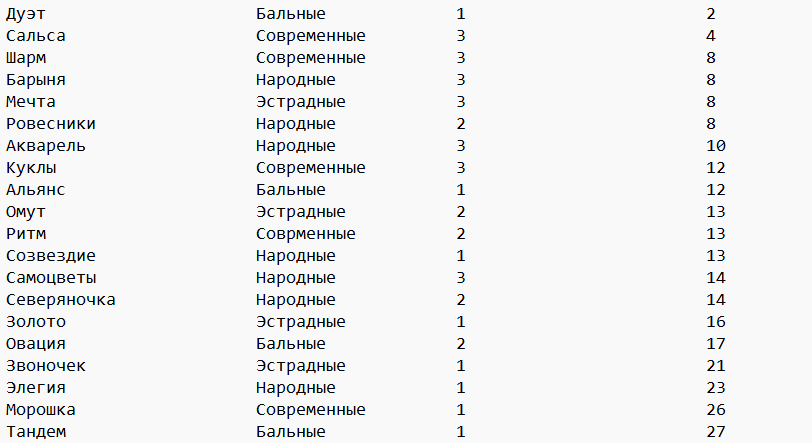


Рисунок 4.1. Содержимое файла до вызова функции

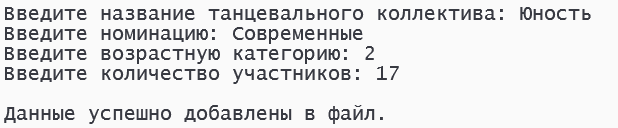


Рисунок 4.2. Пример работы функции в консоли

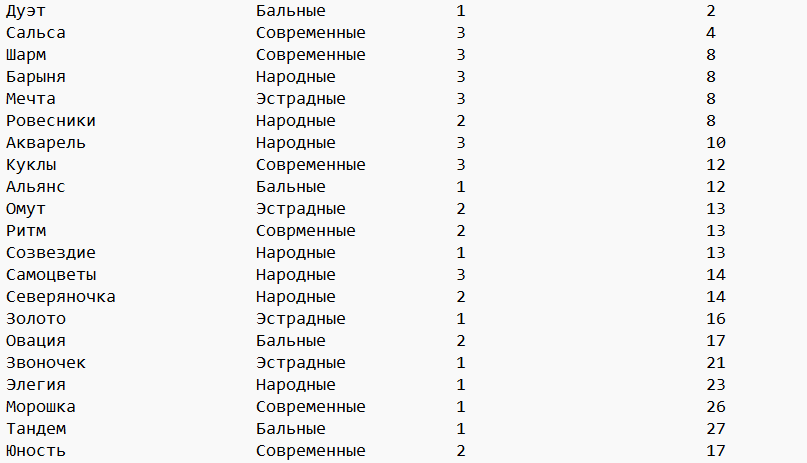


Рисунок 4.3. Содержимое файла после использования функции

4.3. Функция удаления данных из файла

void deleteDataInFile(const string& filename), где filename — это файл со списком всех танцевальных коллективов. Данная функция требует от пользователя ввод названия коллектива, который необходимо удалить из списка. Далее вызывается функция для считывания данных из файла в vector<Participant> list. После открывается файл «list.txt» и в него записываются данные о всех коллективах, название которых не соответствует искомому. Если удаляемый коллектив не найден, выводится соответствующее сообщение. Если файл закрыт, выводит сообщение об ошибке при удалении.

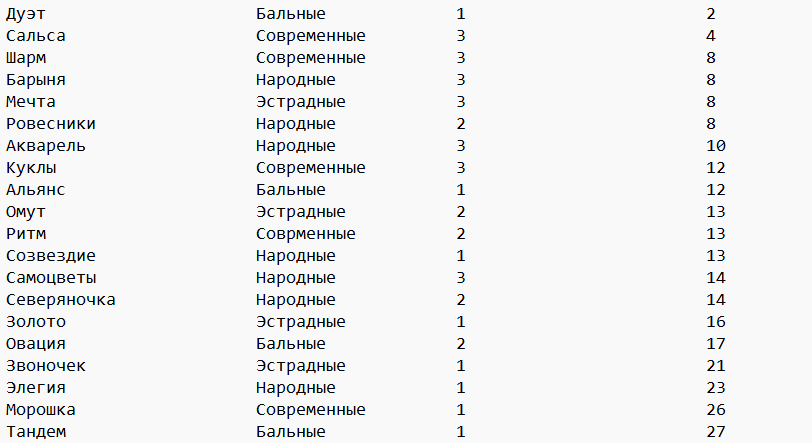


Рисунок 4.4. Содержимое файла до вызова функции

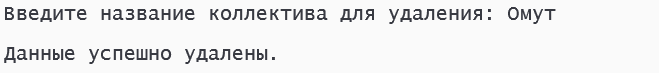


Рисунок 4.5. Пример работы функции в консоли

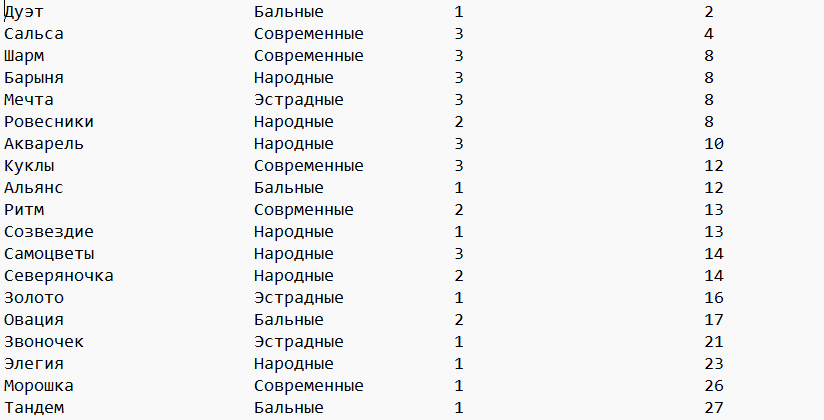


Рисунок 4.6. Содержимое файла после вызова функции

4.4. Функция для просмотра содержимого файла

void viewFile(const string& filename), где filename *—* это файл со списком всех танцевальных коллективов. Эта функция открывает файл и построчно выводит информацию на консоль в виде таблицы. Если файл закрыт, выводит соответствующее сообщение.

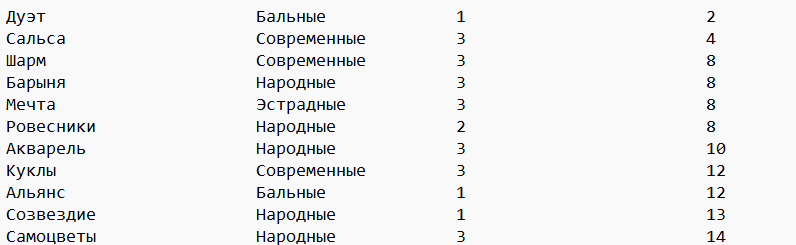


Рисунок 4.7. Содержимое файла

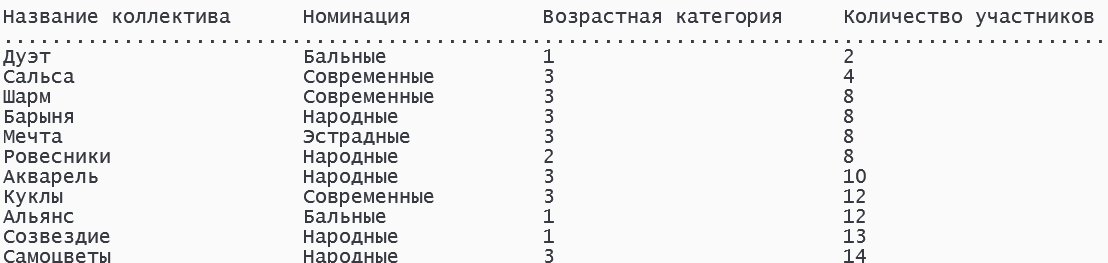


Рисунок 4.8. Вид списка в консоли

4.5. Функции для сортировки методом прямого обмена по количеству участников

В программе присутствуют две функции для сортировки методом прямого обмена по количеству участников: одна сортирует в порядке возрастания – void bubbleSortAsc(const string& filename), а вторая в порядке убывания – void bubbleSortDesc(const string& filename), где filename *—* это файл со списком всех танцевальных коллективов.

В первую очередь данные функции вызывают функцию для считывания данных из файла. После они сортируют полученные данные в порядке возрастания и в порядке убывания количества участников соответственно с помощью метода прямого обмена. Далее они открывают файл «list.txt», в который перезаписываются отсортированные данные. В конце работы обе функции вызывают функцию просмотра файла и выводят сообщение об успешной сортировке.

Пример работы функции void bubbleSortAsc(const string& filename)был показан на рисунках 2.1 и 2.2.

4.6. Функция для обмена элементов массива

Для осуществления быстрой сортировки необходимо обменивать элементы массива местами, для этой задачи в программе используется функция void swap(Participant& a, Participant& b), где Participant& a и Participant& b – ссылки на элементы, которые необходимо поменять местами. Данная функция объявляет дополнительную переменную, с помощью которой и перезаписывает элементы на новые места.

4.7. Функции для разделения массива и возврата индекса опорного элемента

Для осуществления быстрой сортировки необходимо разделять массив на два, в одном из которых находятся элементы меньше опорного, а в другом – больше опорного и равные ему. Для выполнения этой задачи в программе используются функции int partitionAsc(vector<Participant>& arr, int low, int high) и int partitionDesc(vector<Participant>& arr, int low, int high), где vector<Participant>& arr – ссылка на массив, содержащий данные из файла, int low – номер первого сравниваемого элемента массива, int high – номер последнего сравниваемого элемента массива.

В качестве опорного элемента будет использоваться значение возрастной категории коллектива, информация о котором содержится в последнем сравниваемом элементе массива. Сначала объявляется переменная int i, используемая в будущем как счётчик элементов, меньших опорного. Далее циклично сравниваются все необходимые элементы с опорным и, если элемент удовлетворяет условию, счётчик i увеличивается на 1 и вызывается функция swap(Participant& a, Participant& b) для обмена местами элементов под номерами i и значением переменной из цикла. После окончания цикла вызывается функция для обмена местами опорного и первого элемента, который больше либо равен ему (для partitionDesc(vector<Participant>& arr, int low, int high): меньше или равен). В конце работы функции возвращают номер элемента, оказавшегося на последнем месте вместо опорного.

4.8. Функции для реализации быстрой сортировки

В программе присутствуют две функции для быстрой сортировки по возрастной категории: одна сортирует в порядке возрастания – void QuickSortAsc(vector<Participant>& arr, int low, int high), а вторая в порядке убывания – void QuickSortDesc(vector<Participant>& arr, int low, int high), где vector<Participant>& arr – ссылка на массив, содержащий данные из файла, int low – номер первого сравниваемого элемента массива, int high – номер последнего сравниваемого элемента массива.

Обе функции пока элементов в массиве больше одного, сначала вызывают функцию для разделения массива и получения индекса опорного элемента, а потом рекурсивно вызывают себя же, но для двух новых массивов.

4.9. Функции для вызова сортировки методом QuickSort

В программе присутствуют две функции для вызова быстрой сортировки по возрастной категории: одна сортирует в порядке возрастания – void sortbyQuantityAsc (const string& filename), а вторая в порядке убывания – void sortbyQuantityDesc (const string& filename), где filename *—* это файл со списком всех танцевальных коллективов.

Сначала функции вызывают функцию для считывания данных из файла. Потом вызывают функции непосредственно для быстрой сортировки в порядке возрастания и в порядке убывания соответственно. Далее они открывают файл «list.txt» и записывают в него отсортированные данные. В конце обе функции вызывают функцию просмотра файла и выводят сообщение об успешной сортировке.

Пример работы функции void sortbyQuantityAsc (const string& filename) был показан на рисунках 2.3 и 2.4.

4.10. Функция для бинарного поиска по количеству участников

void binarySearch(const string& filename), где filename — это файл со списком всех танцевальных коллективов. Данная функция просит ввести количество участников в коллективе для поиска. Далее она вызывает фунцию для сортировки по возрастанию методом прямого обмена и функцию для считывания данных из файла. После в соответствии с алгоритмом происходит бинарный поиск необходимой записи. При нахождении нужного коллектива, информация о нём выводится на экран. Если такой записи нет, выводится соответстующее сообщение.

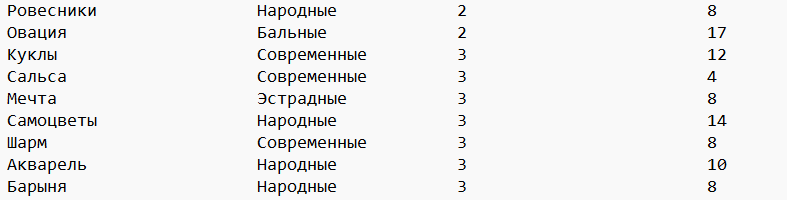


Рисунок 4.9. Содержимое файла

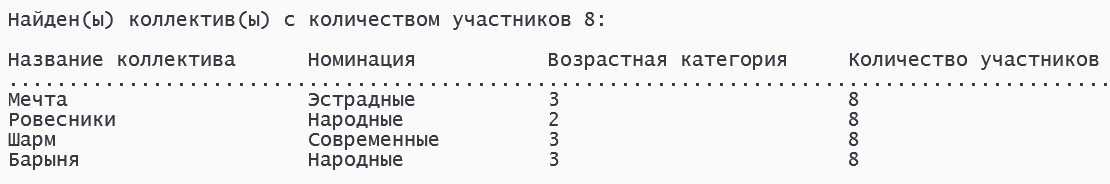


Рисунок 4.10. Результат работы функции в консоли

4.11. Функция для линейного поиска по возрастной категории

void linearSearch(const string& filename), где filename — это файл со списком всех танцевальных коллективов. Данная функция требует ввести возрастную категорию, вызывает функцию для считывания данных из файла и с помощью цикла ищет коллектив, возрастная категория которого соответствует введённой. Если такой коллектив найден, функция выводит информацию о нём в консоль. Если же такого коллектива нет, выводится соответствующее сообщение.

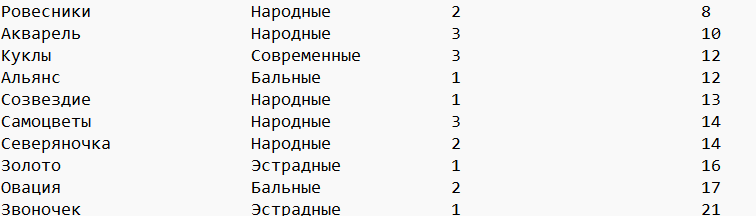


Рисунок 4.11. Содержимое файла

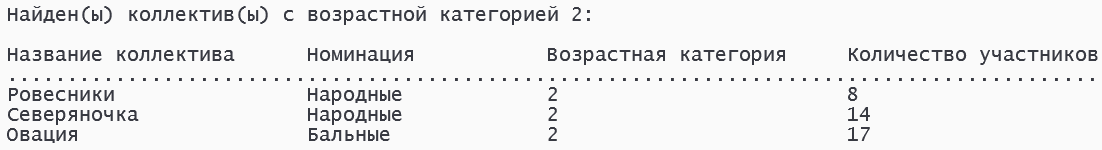


Рисунок 4.12. Результат работы функции в консоли

4.12. Функция для выполнения задания

Функция, которая выводит в консоль и записывает в файл информацию о всех коллективах, участвующих в определённой номинации, в порядке возрастания возрастной категории: void resultTask(const string& filename), где filename — это файл со списком всех танцевальных коллективов. Сначала необходимо ввести название номинации. Далее функция вызывает функцию для считывания данных из файла и открывает файл «result.txt». После функция производит линейный поиск по номинации, записывает все найденные коллективы в файл «result.txt» и вызывает функцию для быстрой сортировки данных в этом файле по возрастной категории. Если коллектив в такой номинации не найден, выводится соответствующее сообщение.



Рисунок 4.11. Содержимое файла

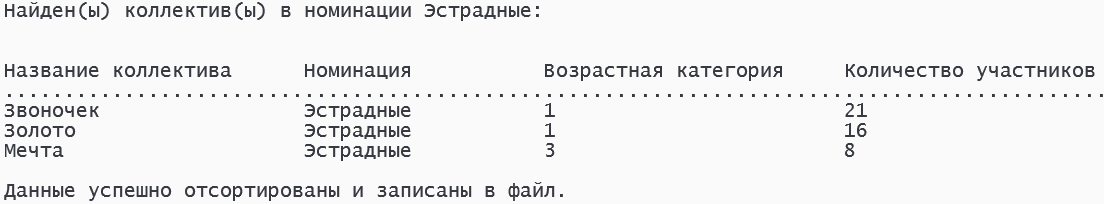


Рисунок 4.12. Результат работы функции в консоли

5. Описание работы программы

Данная программа разработана для качественного и эффективного управления информацией об участниках танцевального конкурса. Она даёт возможность редактирования, просмотра и сортировки данных в файле со списком участников танцевального конкурса, поиска необходимых коллективов. Также программа выводит результаты на экран и сохраняет их в файл.

Сначала подключаются необходимые стандартные библиотеки языка С++, а именно:

* iostream — для реализации консольного ввода/вывода;
* fstream — для реализации файлового ввода/вывода;
* string — для работы со строками;
* Windows.h— для настройки корректного ввода/вывода информации на русском языке;
* iomanip — для форматирования консольного вывода;
* vector — для работы с динамическими массивами.

Дальше в коде идет структура Participant, которая предназначена для хранения информации о каждом коллективе-участнике. Структура содержит такие поля как: name (название коллектива), nomination (номинация, в которой участвует коллектив), age (возрастная категория), quantity (количество участников).

Программа начинает свою работу с вывода меню на консоль.

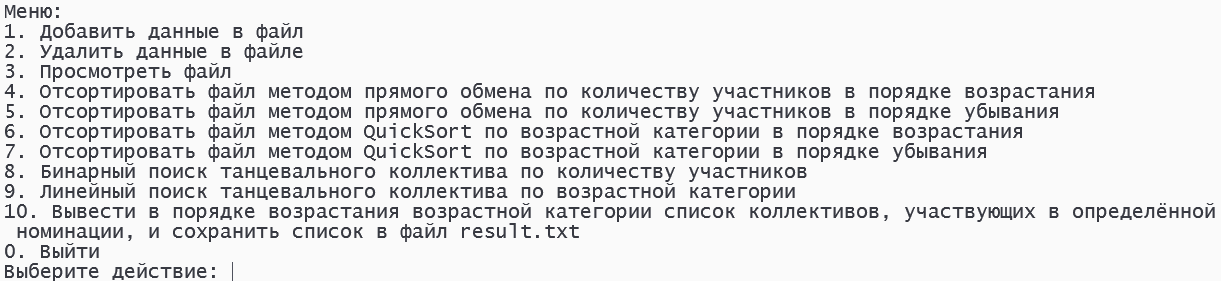


Рисунок 5.1. Вид меню в консоли

После выбора пользователем необходимой команды вызывается соответствующая пользовательская функция, которая выполняет определённые задачи.

Заключение

В ходе курсовой работы мы научились работе с структурами данных и файлами. Исследовали различные алгоритмы сортировки и поиска. Мы научились читать данные из текстового файла и записывать их обратно, создавать структуру для представления информации о коллективах-участниках танцевального конкурса, использовать различные алгоритмы сортировки для упорядочивания данных. Мы реализовали линейный и бинарный поиск для быстрого поиска необходимых коллективов в большом объёме данных. Также мы создали текстовый интерфейс в консоли, который позволяет пользователю выбирать различные операции, для работы со список коллективов-участников танцевального конкурса.

В целом, эта курсовая работа позволила нам изучить различные фундаментальные знания в программировании и научиться эффективно обрабатывать и анализировать данные, что является важным навыком в современном мире программирования.

Список использованных источников

[1] Работа с файлами [электронный ресурс] // URL: <https://metanit.com/c/tutorial/7.3.php?ysclid=lv1dss68y4468107293>

[2] Быстрая сортировка [электронный ресурс] // URL: <https://favtutor.com/blogs/quick-sort-cpp>

[3] Метод прямого обмена [электронный ресурс] // URL: https://codelessons.dev/ru/puzyrkovaya-sortirovka-v-c-glavnye-momenty/

[4] Бинарный поиск [электронный ресурс] // URL: https://agorinenko.github.io/data-structures-and-algorithms/tutorial/binary\_search.html#:~:text=%D0%91%D0%B8%D0%BD%D0%B0%D1%80%D0%BD%D1%8B%D0%B9%20%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%20%E2%80%94%20%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%20%D0%BD%D0%B0%D1%85%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F,%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B3%D0%B0%D1%80%D0%B0%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%20%D0%BD%D0%B5%20%D0%BC%D0%BE%D0%B6%D0%B5%D1%82%20%D0%BD%D0%B0%D1%85%D0%BE%D0%B4%D0%B8%D1%82%D1%8C%D1%81%D1%8F

Приложение А ****(обязательное)****

****Листинг кода****

#include <iostream>

#include <fstream>

#include <string>

#include <Windows.h>

#include <iomanip>

#include <vector>

using namespace std;

struct Participant

{

string name;

string nomination;

int age;

int quantity;

} list;

// Функция для считывания данных из файла

vector<Participant> inputData(const string& filename)

{

int number\_of\_lines = 0;

string line;

ifstream file(filename);

while (getline(file, line))

++number\_of\_lines;

ifstream inputFile(filename);

vector<Participant> list(number\_of\_lines); // Объявление массива структур

if (inputFile.is\_open())

{

int count = 0; // Количество считанных записей

while (count<number\_of\_lines)

{

inputFile >> list[count].name >> list[count].nomination >> list[count].age >> list[count].quantity;

++count;

}

}

else

cout << "Файл для чтения не открыт" << endl;

inputFile.close();

return list;

}

// Функция для добавления данных в файл

void addDataToFile(const string& filename)

{

Participant stru;

cout << "Введите название танцевального коллектива: ";

cin >> stru.name;

cout << "Введите номинацию: ";

cin >> stru.nomination;

cout << "Введите возрастную категорию: ";

while (true)

{

cin >> stru.age;

if (stru.age != 1 && stru.age != 2 && stru.age != 3)

{

cin.clear();

cin.ignore();

cout << "Пожалуйста, введите корректное значение." << endl;

}

else

break;

}

cout << "Введите количество участников: ";

while (true)

{

cin >> stru.quantity;

if (cin.fail())

{

cin.clear();

cin.ignore();

cout << "Пожалуйста, введите корректное значение." << endl;

}

else

break;

}

ofstream file(filename, ios::app);

if (file.is\_open())

{

file << setw(25) << left << stru.name << setw(20) << stru.nomination << setw(25) << stru.age << setw(10) << stru.quantity << endl;

file.close();

cout << endl << "Данные успешно добавлены в файл." << endl;

}

else

{

cout << "Ошибка при добавлении данных в файл." << endl;

}

cout << endl;

}

// Функция для удаления данных в файле

void deleteDataInFile(const string& filename) {

string targetName;

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

cin >> targetName;

cout << endl;

vector<Participant> list = inputData(filename);

// Выполнить линейный поиск по названию

bool found = false;

ofstream outputFile(filename);

if (outputFile.is\_open())

{

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

{

if (list[i].name != targetName)

{

outputFile << setw(25) << left << list[i].name << setw(20) << list[i].nomination << setw(25) << list[i].age << setw(10) << list[i].quantity << endl;

}

else

found = true;

}

if (!found)

{

cout << "Коллектив с названием " << targetName << " не найден." << endl;

}

else

cout << "Данные успешно удалены." << endl;

}

else

{

cout << "Ошибка при удалении данных из файла." << endl;

}

cout << endl;

}

// Функция для просмотра содержимого файла

void viewFile(const string& filename)

{

ifstream file(filename);

string line;

cout << endl;

cout << setw(25) << left << "Название коллектива" << setw(20) << "Номинация" << setw(25) << "Возрастная категория" << setw(10) << "Количество участников" << endl;

cout << "........................................................................................................" << endl;

if (file)

{

while(getline(file, line))

cout << line << endl;

}

else

{

cout << "Ошибка при просмотре файла" << endl;

}

cout << endl;

}

// Функция для сортировки методом прямого обмена по количеству участников в порядке возрастания

void bubbleSortAsc(const string& filename)

{

vector<Participant> list = inputData(filename);

// Сортировка

bool swapped;

do

{

swapped = false;

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

{

if (list[i].quantity > list[i + 1].quantity)

{

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

swapped = true;

}

}

} while (swapped);

// Запись отсортированных данных в файл

ofstream outputFile(filename);

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

{

outputFile << setw(25) << left << list[i].name << setw(20) << list[i].nomination << setw(25) << list[i].age << setw(10) << list[i].quantity << endl;

}

}

// Функция для сортировки методом прямого обмена по количеству участников в порядке убывания

void bubbleSortDesc(const string& filename)

{

vector<Participant> list = inputData(filename);

// Сортировка

bool swapped;

do

{

swapped = false;

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

{

if (list[i].quantity < list[i + 1].quantity)

{

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

swapped = true;

}

}

} while (swapped);

// Запись отсортированных данных в файл

ofstream outputFile(filename);

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

{

outputFile << setw(25) << left << list[i].name << setw(20) << list[i].nomination << setw(25) << list[i].age << setw(10) << list[i].quantity << endl;

}

viewFile(filename);

cout << "Данные успешно отсортированы и записаны в файл." << endl << endl << endl;

}

// Функция для обмена элементов массива

void swap(Participant& a, Participant& b)

{

Participant temp = a;

a = b;

b = temp;

}

// Функции для разделения массива и возврата индекса опорного элемента

int partitionAsc(vector<Participant>& arr, int low, int high)

{

int i = low - 1;

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

{

if (arr[j].age < arr[high].age)

{

i++;

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

}

}

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

return i + 1;

}

int partitionDesc(vector<Participant>& arr, int low, int high)

{

int i = low - 1;

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

{

if (arr[j].age > arr[high].age)

{

i++;

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

}

}

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

return i + 1;

}

// Функции для реализации QuickSort

void QuickSortAsc(vector<Participant>& arr, int low, int high)

{

if (low < high)

{

int index = partitionAsc(arr, low, high);

QuickSortAsc(arr, low, index - 1);

QuickSortAsc(arr, index + 1, high);

}

}

void QuickSortDesc(vector<Participant>& arr, int low, int high)

{

if (low < high)

{

int index = partitionDesc(arr, low, high);

QuickSortDesc(arr, low, index - 1);

QuickSortDesc(arr, index + 1, high);

}

}

// Функция для сортировки методом QuickSort по возрасту в порядке возрастания

void sortbyQuantityAsc(const string& filename)

{

vector<Participant> list = inputData(filename);

// Сортировка

QuickSortAsc(list, 0, list.size() - 1);

// Запись отсортированных данных в файл

ofstream outputFile(filename);

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

{

outputFile << setw(25) << left << list[i].name << setw(20) << list[i].nomination << setw(25) << list[i].age << setw(10) << list[i].quantity << endl;

}

viewFile(filename);

cout << "Данные успешно отсортированы и записаны в файл." << endl << endl << endl;

}

// Функция для сортировки методом QuickSort по возрасту в порядке убывания

void sortbyQuantityDesc(const string& filename)

{

vector<Participant> list = inputData(filename);

// Сортировка

QuickSortDesc(list, 0, list.size() - 1);

// Запись отсортированных данных в файл

ofstream outputFile(filename);

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

{

outputFile << setw(25) << left << list[i].name << setw(20) << list[i].nomination << setw(25) << list[i].age << setw(10) << list[i].quantity << endl;

}

viewFile(filename);

cout << "Данные успешно отсортированы и записаны в файл." << endl << endl << endl;

}

// Функция для бинарного поиска по количеству участников

void binarySearch(const string& filename)

{

cout << "Введите количество участников: ";

int targetQuantity;

while (true)

{

cin >> targetQuantity;

if (cin.fail())

{

cin.clear();

cin.ignore();

cout << "Пожалуйста, введите корректное значение." << endl;

}

else

break;

}

cout << endl << endl;

bubbleSortAsc(filename);

vector<Participant> list = inputData(filename);

// Выполнить бинарный поиск по количеству участников

int low = 0;

int high = list.size() - 1;

bool found = false;

while (low <= high)

{

int mid = low + (high - low) / 2;

if (list[mid].quantity == targetQuantity)

{

// Вывести информацию о найденном коллективе

cout << "Найден(ы) коллектив(ы) с количеством участников " << targetQuantity << ":" << endl << endl;

cout << setw(25) << left << "Название коллектива" << setw(20) << "Номинация" << setw(25) << "Возрастная категория" << setw(10) << "Количество участников" << endl;

cout << "........................................................................................................" << endl;

cout << setw(25) << left << list[mid].name << setw(20) << list[mid].nomination << setw(25) << list[mid].age << setw(10) << list[mid].quantity << endl;

found = true;

for (int i = mid-1; list[i].quantity == targetQuantity; i--)

{

cout << setw(25) << left << list[i].name << setw(20) << list[i].nomination << setw(25) << list[i].age << setw(10) << list[i].quantity << endl;

}

for (int i = mid+1; list[i].quantity == targetQuantity; i++)

{

cout << setw(25) << left << list[i].name << setw(20) << list[i].nomination << setw(25) << list[i].age << setw(10) << list[i].quantity << endl;

}

break;

}

else if (list[mid].quantity < targetQuantity)

{

low = mid + 1;

}

else

{

high = mid - 1;

}

}

if (!found)

{

cout << "Коллектив с количеством участников " << targetQuantity << " не найден." << endl << endl;

}

cout << endl << endl;

}

// Функция для линейного поиска по возрастной категории

void linearSearch(const string& filename)

{

cout << "Введите возрастную категорию: ";

int targetAge;

while (true) {

cin >> targetAge;

if (targetAge != 1 && targetAge != 2 && targetAge != 3)

{

cin.clear();

cin.ignore();

cout << "Пожалуйста, введите корректное значение." << endl;

}

else

break;

}

cout << endl << endl;

vector<Participant> list = inputData(filename);

// Выполнить линейный поиск по возрасту

bool found = false;

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

{

if (list[i].age == targetAge)

{

cout << "Найден(ы) коллектив(ы) с возрастной категорией " << targetAge << ":" << endl << endl;

cout << setw(25) << left << "Название коллектива" << setw(20) << "Номинация" << setw(25) << "Возрастная категория" << setw(10) << "Количество участников" << endl;

cout << "........................................................................................................" << endl;

found = true;

break;

}

}

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

{

if (list[i].age == targetAge)

{

// Вывести информацию о найденном коллективе

cout << setw(25) << left << list[i].name << setw(20) << list[i].nomination << setw(25) << list[i].age << setw(10) << list[i].quantity << endl;

}

}

cout << endl << endl;

if (!found)

{

cout << "Коллектив в возрастной категории " << targetAge << " не найден." << endl;

}

}

//Функция выполнения задания

void resultTask(const string& filename)

{

cout << "Введите номинацию: ";

string targetNomination;

cin >> targetNomination;

cout << endl;

vector<Participant> list = inputData(filename);

ofstream outputFile("result.txt");

// Выполнить линейный поиск по номинации

bool found = false;

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

{

if (list[i].nomination == targetNomination)

{

cout << "Найден(ы) коллектив(ы) в номинации " << targetNomination << ":" << endl << endl;

found = true;

break;

}

}

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

{

if (list[i].nomination == targetNomination)

{

outputFile << setw(25) << left << list[i].name << setw(20) << list[i].nomination << setw(25) << list[i].age << setw(10) << list[i].quantity << endl;

}

}

if (!found)

{

cout << "Коллектив в номинации " << targetNomination << " не найден." << endl;

}

else

{

sortbyQuantityAsc("result.txt");

}

cout << endl;

}

int main()

{

string filename = "list.txt";

int a;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

do

{

cout << "Меню:" << endl;

cout << "1. Добавить данные в файл" << endl;

cout << "2. Удалить данные в файле" << endl;

cout << "3. Просмотреть файл" << endl;

cout << "4. Отсортировать файл методом прямого обмена по количеству участников в порядке возрастания" << endl;

cout << "5. Отсортировать файл методом прямого обмена по количеству участников в порядке убывания" << endl;

cout << "6. Отсортировать файл методом QuickSort по возрастной категории в порядке возрастания" << endl;

cout << "7. Отсортировать файл методом QuickSort по возрастной категории в порядке убывания" << endl;

cout << "8. Бинарный поиск танцевального коллектива по количеству участников" << endl;

cout << "9. Линейный поиск танцевального коллектива по возрастной категории" << endl;

cout << "10. Вывести в порядке возрастания возрастной категории список коллективов, участвующих в определённой номинации, и сохранить список в файл result.txt" << endl;

cout << "0. Выйти" << endl;

cout << "Выберите действие: ";

while (true)

{

cin >> a;

if (cin.fail())

{

cin.clear();

cin.ignore();

cout << "Пожалуйста, введите корректное значение." << endl;

}

else

break;

}

cout << endl;

switch (a)

{

case 1:

{

addDataToFile(filename);

break;

}

case 2:

{

deleteDataInFile(filename);

break;

}

case 3:

{

viewFile(filename);

break;

}

case 4:

{

bubbleSortAsc(filename);

viewFile(filename);

cout << "Данные успешно отсортированы и записаны в файл." << endl << endl << endl;

break;

}

case 5:

{

bubbleSortDesc(filename);

break;

}

case 6:

{

sortbyQuantityAsc(filename);

break;

}

case 7:

{

sortbyQuantityDesc(filename);

break;

}

case 8:

{

binarySearch(filename);

break;

}

case 9:

{

linearSearch(filename);

break;

}

case 10:

{

resultTask(filename);

break;

}

}

} while (a != 0);

return 0;

}

Приложение Б

Блок-схема работы программы

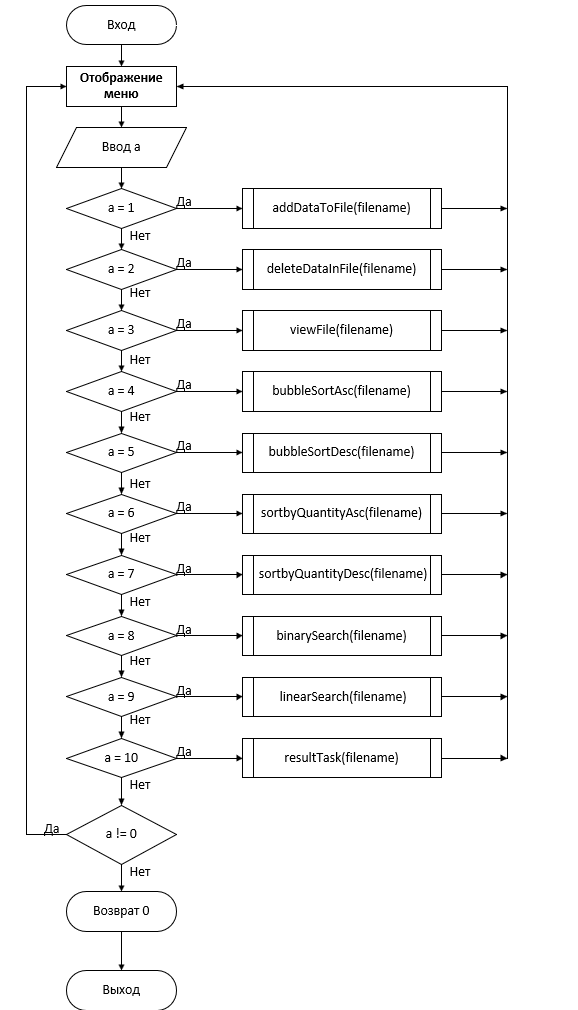


Рисунок Б.1 – Блок-схема работы программы

ВЕДОМОСТЬ ДОКУМЕНТОВ

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | | | | *Дополнительные сведения* | | | |
|  | | | | |  | | | |  | | | |
|  | | | | | *Текстовые документы* | | | |  | | | |
|  | | | | |  | | | |  | | | |
| *БГУИР ДП 6-05-0611-03 053* | | | | | *Пояснительная записка* | | | | *40с.* | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  | | | | |  | | | |  | | | |
|  |  |  |  |  | *БГУИР ДП 6-05-0611-03 053 Д1* | | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм.* | *Л.* | *№ докум.* | *Подп.* | *Дата* | Структура списка участников танцевального конкурса  Ведомость курсового проекта | *Лит* | | | *Лист* | | *Листов* | |
| *Разраб.* | | *Перминова В. В.* |  |  |  | *У* |  | *40* | *40* | |
| *Пров.* | | *Матюшкин С. И.* |  |  | *Кафедра ВМиП*  *гр. 321701* | | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |