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»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  Старший преподаватель  кафедры ВМиП  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.В.Гуревич |
|  | \_\_\_.\_\_\_\_.2024 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему:

**«структура списка изделий мебельного магазина»**

БГУИР КР 6-05-0611-03 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 321702  Казаченко Вадим Александрович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовая работа представлена на проверку \_\_\_.\_\_\_\_.2024  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2024

**РЕФЕРАТ**

БГУИР КР 6-05-0611-03 ПЗ

**Пучинская, П.В.** Структура списка изделий мебельного магазина / П.В.  Пучинская. — Минск : БГУИР, 2024. — 35 с.

Пояснительная записка 35 с., 36 рис., 1 табл., 5 источников, 1 приложение

структура списка изделий мебельного магазина, схемы алгоритмов, программное средство, модели *UML*

*Цель* *курсовой работы*: улучшение процесса учета информации о продукции в мебельном магазине. Программа предоставляет возможность внесения изменений в уже существующие данные, создания новых записей, удаления устаревших и сохранения отредактированных записей.

*Методология проведения работы*: в процессе решения поставленных задач использованы принципы системного подхода, аналитические методы, методы компьютерной обработки экспериментальных данных и компьютерного моделирования.

*Результаты работы*: выполнена постановка задачи и определены основные методы ее решения; в ходе объектного моделирования системы построен ряд *UML*-диаграмм; описаны основные алгоритмы работы программного средства; выполнено тестирование программного средства, показавшее его соответствие функциональным требованиям, поставленным в задании на разработку.

Программный продукт разработан на языке *C++* с применением *MS Visual Studio 2022*.

*Область применения результатов*: разработанное программное средство позволяет анализировать учет продукции в мебельном магазине.

Разработанное программное средство полностью соответствует всем функциональным требованиям, необходимым при учете и анализе информации о продукции в мебельном магазине.

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 6](#_Toc166755131)

[**1 СТРУКТУРЫ И ФАЙЛЫ** 7](#_Toc166755132)

[**1.1 Описание предметной области и функциональных требований** 7](#_Toc166755133)

[**1.2 Выбор способа организации данных** 7](#_Toc166755134)

[**1.3 UML-модели представления программного средства и их описание** 8](#_Toc166755135)

[**2 АЛГОРИТМЫ СОРТИРОВКИ И ПОИСКА** 10](#_Toc166755136)

[**2.1 Оценка сложности алгоритмов. Big O** 10](#_Toc166755137)

[**2.2 Selection sort** 11](#_Toc166755138)

[**2.3 Quick sort** 13](#_Toc166755139)

[**2.4 Linear search** 15](#_Toc166755140)

[**2.5 Binary search** 16](#_Toc166755141)

[**3 ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ** 17](#_Toc166755142)

[**4 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ** 18](#_Toc166755143)

[**4.1 Разработка схем алгоритмов программы** 18](#_Toc166755144)

[**4.2 Тестирование и проверка работоспособности программного средства** 20](#_Toc166755145)

[**4.3 Сквозное тестирование** 22](#_Toc166755146)

[**ЗАКЛЮЧЕНИЕ** 26](#_Toc166755147)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 27](#_Toc166755148)

[**ПРИЛОЖЕНИЕ А** 28](#_Toc166755149)

# **ВВЕДЕНИЕ**

В современном мире мебельные магазины играют важную роль в розничной торговле. Они предоставляют широкий ассортимент товаров для обустройства дома и офиса. Организация и эффективное ведение таких магазинов требует тщательного учета товаров, их количества, цен и состояния на складе. В связи с этим, разработка программы для учета информации о мебельном магазине является актуальной задачей, которая поможет автоматизировать процесс сбора и обработки данных, упростить работу персонала и повысить качество обслуживания покупателей.

Курсовой проект затрагивает очень важную проблему — обработка большого количества информации. Одним из способов решения данной проблемы является автоматизация работы с информацией.

Целью данной курсовой работы является разработка приложения для упрощения хранения информации о товарах мебельного магазина с возможностью внесения, редактирования, удаления данных, а также вывода информации в необходимом формате.

Для достижения поставленных целей, необходимо выполнить следующие задачи:

* ознакомиться с предметной областью;
* проанализировать процесс учета товаров и его особенности.
* разработать алгоритмы работы программного средства для учета товаров мебельного магазина.
* протестировать работу программного средства для подтверждения его функциональности и эффективности.

Объектом исследования является учет информации о товарах в мебельном магазине.

# **СТРУКТУРЫ И ФАЙЛЫ**

## **Описание предметной области и функциональных требований**

В предметной области программы учета данных о мебельном магазине важно понять, что магазин представляет собой место, где предлагается широкий ассортимент мебели для дома и офиса. Каждая единица товара обладает уникальными характеристиками, такими как название, тип, количество, стоимость и статус.

Требования к разрабатываемому программному средству основаны на потребностях продавцов-консультантов, которые используют приложение для управления ассортиментом товаров, консультирования покупателей и осуществления продаж.

Приложение для мебельного магазина предоставляет следующий набор функциональностей для продавцов-консультантов: просмотр доступных товаров с возможностью фильтрации и сортировки, добавление новых товаров в ассортимент магазина, удаление или редактирование информации о товарах, поиск товаров по различным параметрам, предоставление консультаций покупателям и совершение продаж. Эти функциональности обеспечивают удобный и эффективный способ управления магазином, повышая качество обслуживания покупателей и оптимизируя процессы продаж.

## **Выбор способа организации данных**

Структура в C++ представляет собой производный тип данных, обычно служат для хранения каких-то общедоступных данных в виде публичных переменных. Для проектирования приложения были выбраны структуры, в связи с их простотой использования. На рисунке 1.1 показана структура Furniture.

Таблица 1.1 – Описание структуры Furniture

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Назначение поля |
| id | int | ID изделия |
| name | string | Название изделия |
| type | string | Тип изделия |
| quantity | int | Количество изделия |
| price | double | Стоимость |
| status | string | Статус |

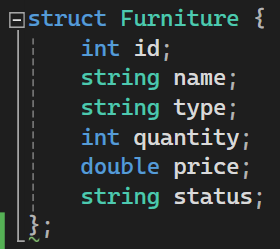
****

Рисунок 1.1 – Структура Furniture

## **UML-модели представления программного средства и их описание**

UML является графическим языком для визуализации, специфицирования, конструирования и документирования систем, в которых большая роль принадлежит программному обеспечению. С помощью UML можно разработать детальный план создаваемой системы.

На рисунке 1.2 представлена данная диаграмма, которая описывают систему с точки зрения возможностей ее использования продавцом-консультантом.



Рисунок 1.2 – UML диаграмма вариантов использования

На рисунке 1.3 представлена диаграмма компонентов. Этот тип диаграмм используется для представления организации совокупности компонентов и существующих между ними зависимостей. Данный тип диаграмм описывает систему с точки зрения реализации. При проектировании больших систем может оказаться, что система должна быть разложена на несколько сотен или тысяч компонентов. Именно данный тип диаграмм позволяет не потеряться в обилии модулей и связей между ними.

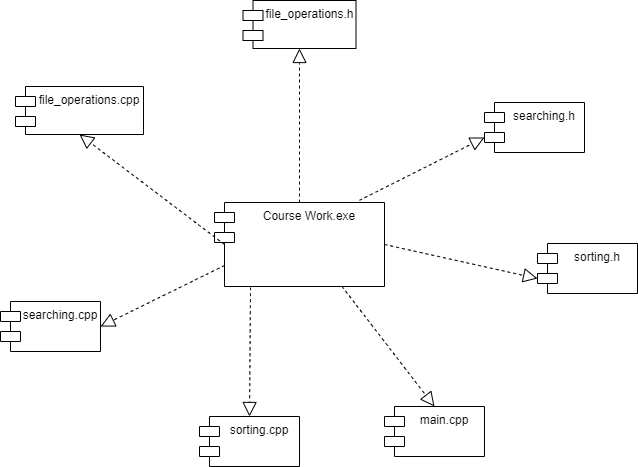


Рисунок 1.3 – UML диаграмма компонентов

На рисунке 1.4 показана диаграмма развертывания.

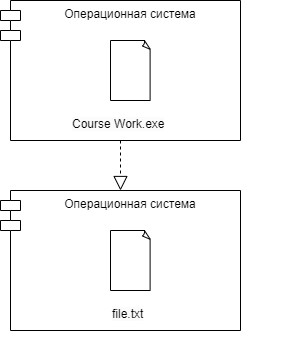


Рисунок 1.4 – UML диаграмма развертывания

Файл "file.txt" в программе используется для хранения информации о мебели, такой как ID, название, тип, количество, стоимость и статус каждого предмета мебели. В этом файле каждая строка представляет отдельный предмет мебели, а каждый столбец разделен пробелами и содержит определенные характеристики предмета мебели. Этот файл играет важную роль в хранении данных о мебели и обеспечивает постоянный доступ к этой информации в программе. На рисунке 1.5 показан пример возможного содержимого данного файла.

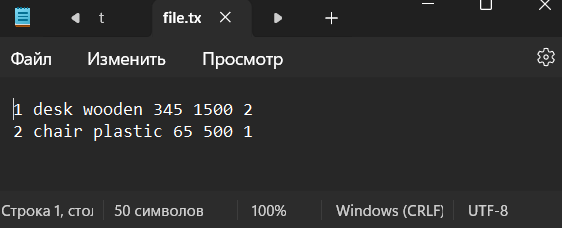
****

Рисунок 1.5 – пример содержимого файла file.txt

# **АЛГОРИТМЫ СОРТИРОВКИ И ПОИСКА**

## **Оценка сложности алгоритмов. Big O**

Структура нотации Big O в компьютерной науке является основным инструментом для оценки производительности алгоритмов при увеличении объема данных. Она описывает изменение скорости выполнения кода при росте объема данных и не зависит от конкретной аппаратной платформы. Анализируя количество шагов, необходимых для выполнения алгоритма, Big O позволяет сравнивать разные алгоритмы независимо от оборудования. Важно отметить, что незначительные операции обычно игнорируются из-за их незначительного влияния на общую производительность. Каждый алгоритм имеет три основных случая временной сложности: лучший, худший и средний. Лучший случай — это идеальная ситуация с минимальным количеством шагов или временем выполнения, в то время как худший случай описывает максимальное количество шагов или время выполнения. Средний случай представляет типичную ситуацию с определенным количеством шагов или временем выполнения.

На рисунке 2.1 представлен график, отображающий зависимость временной сложности алгоритмов от объема данных. Алгоритмы с разной временной сложностью имеют различные скорости роста времени выполнения при увеличении объема данных. Например, linear search имеет линейную вычислительную сложность алгоритма O(n), binary search — логарифмическое время log(n), quick sort — линейно-логарифмическое время n log(n), selection sort — квадратичное время n^2.

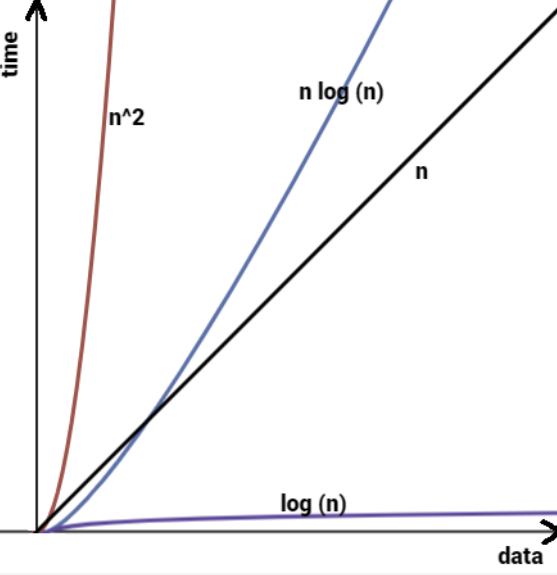


Рисунок 2.1 – График сложности алгоритмов

## **Selection sort**

Сортировка выбором (Selection sort) — метод при котором проходим по массиву и на каждой итерации ищем минимальное значение и перемещаем его в начало массива. Затем снова начинаем искать минимальный элемент среди оставшихся элементов и продолжаем этот процесс до тех пор, пока весь массив не будет отсортирован.

Например, дан массив, который надо отсортировать (рисунок 2.2) .



Рисунок 2.2

На первой итерации из всех элементов массива находим минимальный и перемещаем его в начало (рисунок 2.3) .



Рисунок 2.3

Далее продолжаем поиск следующего минимального элемента среди оставшихся элементов, начиная с индекса 1 и до последнего элемента. Как только находим его, перемещаем его на позицию следующую за предыдущим минимальным элементом, который уже находится на своем месте в начале массива (рисунок 2.4) .



Рисунок 2.4

В итоге, пройдя через каждую итерацию, мы расположим все элементы в порядке возрастания (рисунок 2.5) .



Рисунок 2.5

## **Quick sort**

Быстрая сортировка (QuickSort) — это эффективный алгоритм сортировки, который использует метод разделения и властвующего принципа. Он разбивает массив на две подмассива вокруг опорного элемента, так чтобы элементы слева от опорного были меньше него, а справа — больше. Затем данный процесс рекурсивно применяется к подмассивам до достижения базового случая.

Воспользуемся псевдокодом для описания работы алгоритма:

1. Устанавливаем опорный элемент (pivot) на последний элемент массива, индекс j в начало массива, а индекс i на одну позицию перед началом массива.
2. Если значение элемента с индексом j меньше опорного элемента (pivot):
   1. Увеличиваем индекс i на 1.
   2. Меняем местами элементы под индексами i и j.
   3. Увеличиваем индекс j на 1.
3. Если значение элемента с индексом j больше или равно опорному элементу (pivot):
   1. Увеличиваем индекс j на 1.
4. Если значение элемента с индексом j равно опорному элементу (pivot):
   1. Увеличиваем индекс i на 1.
   2. Меняем местами элементы под индексами i и pivot, тем самым получаем новый pivot.
5. Повторяем эти шаги до завершения процесса сортировки.

Например, дан массив, который надо отсортировать (рисунок 2.6) .



Рисунок 2.6

Устанавливаем pivot, i и j (рисунок 2.7) .



Рисунок 2.7

Сравниваем j и pivot. Так как 8 > 5 инкрементируем j (рисунок 2.8) .



Рисунок 2.8

Снова сравниваем j и pivot.Так как 2 < 5 инкрементируем i, меняем местами i и j, а потом инкрементируем j (рисунок 2.9) .

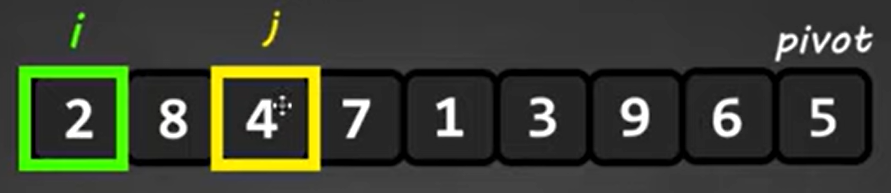


Рисунок 2.9

Проделываем эти шаги дальше. Когда j станет равным pivot, инкрементируем i и меняем местами i и pivot, тем самым получим новый pivot (рисунок 2.10) .

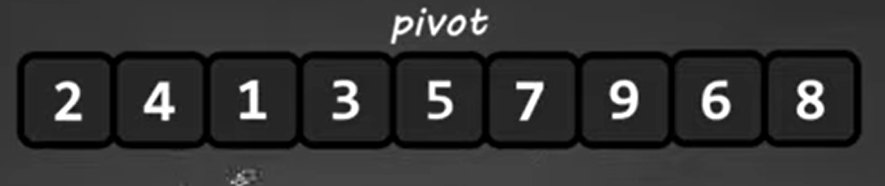


Рисунок 2.10

Заметим, что элементы, стоящие справа, меньше, чем pivot, а слева – больше. Так как quick sort — это рекурсивный алгоритм, то получим два подмассива, для которых проделаем те же шаги (рисунок 2.11) .

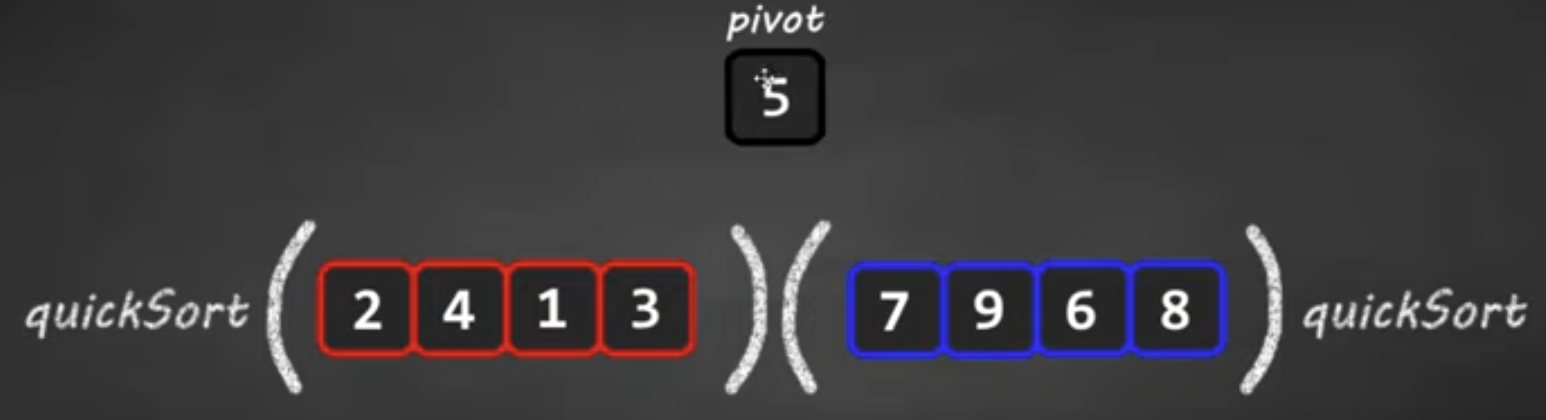


Рисунок 2.11

В конце выполения алгоритма получим такой результат (рисунок 2.12) .



Рисунок 2.12

## **Linear search**

Линейный поиск (linear search) — это простой метод поиска в массиве, который последовательно проверяет каждый элемент массива на соответствие заданному значению.

Например, для следующего массива, чтобы найти 3, надо выполнить 6 итераций, на каждой из которых мы ищем заданное значение (рисунок 2.13) .



Рисунок 2.13

## **Binary search**

Бинарный поиск (binary search) — это алгоритм поиска, который находит позицию заданного значения в отсортированном массиве. За каждый «шаг» этого алгоритма половина массива устраняется, что делает его очень эффективным для поиска в больших отсортированных данных.

Опишем работу алгоритма с помощью псевдокода:

1. Сортируем массив
2. Делим массив на две части и определяем элемент в середине
3. Если элемент, который мы ищем, равен элементу в середине массива, нужный индекс найден.
4. Если элемент, который мы ищем, больше элемента в середине массива, продолжаем поиск в правой части массива.
5. Если элемент, который мы ищем, меньше элемента в середине массива, продолжаем поиск в левой части массива.
6. Повторяем эти действия, пока не найдем искомый элемент или пока не останется только один элемент в подмассиве.

Например, для следующего массива надо найти элемент «H» (рисунок 2.14) .

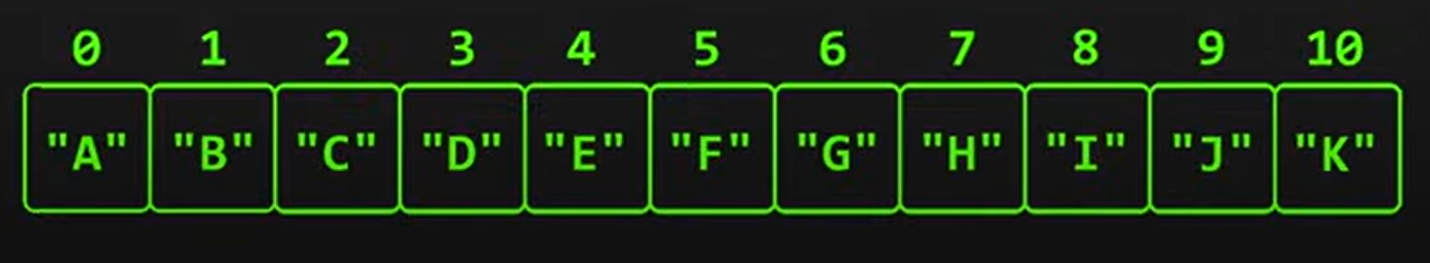


Рисунок 2.14

Ищем середину массива (рисунок 2.15) .



Рисунок 2.15

Так как «F» ≠ «H» и «H» > «F», то переходим в правый подмассив (рисунок 2.16) .



Рисунок 2.16

Далее выполняем те же действия, пока не найдем нужный элемент (рисунок 2.17) .



Рисунок 2.17

# **ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ**

Функции в программе учета данных о мебельном магазине играют ключевую роль. Они обеспечивают модульность кода, разделяя его на логические блоки и делая его более понятным для разработчиков. Кроме того, функции обеспечивают удобство и повторное использование кода, поскольку один и тот же блок кода может быть вызван из разных частей программы, что экономит время разработки. Использование функций также способствует организации и структурированию кода, делая его более читаемым и легко поддерживаемым.

Для работы программного обеспечения были разработаны и использованы следующие функции:

* void printItem(const Furniture& item);// Вывода информации о товаре на экран
* void selectionSort(Furniture array[], int size);// Сортировка товаров по названию
* void quickSort(Furniture array[], int start, int end);// Сортировка товаров по стоимости
* int partition(Furniture array[], int start, int end);// Вспомогательная функция для quickSort для разделения массива на подмассивы
* int linearSearch(Furniture array[], int size, string itemName);// Поиск по имени товара
* int binarySearch(Furniture array[], int length, string itemName);// Поиск по имени товара
* void create();//Создание файла
* void view(); //Просмотр товаров из файла
* void add(Furniture array[], int& size);// Добавление новых товаров в файл
* void outputAvailableItems(Furniture array[], int size);// Вывод товаров, которые доступны в наличии и имеют цену менее 1000 рублей
* void removeItem(Furniture array[], int& size);//Удаление товара из файла

# **ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ**

## **Разработка схем алгоритмов программы**

Алгоритм в программировании — это набор последовательных инструкций, которые описывают порядок поведения программы для достижения нужной цели.

Сам алгоритм можно удобно представить в виде диаграммы. На рисунке 3.1 показан алгоритм реализации функции selectionSort.

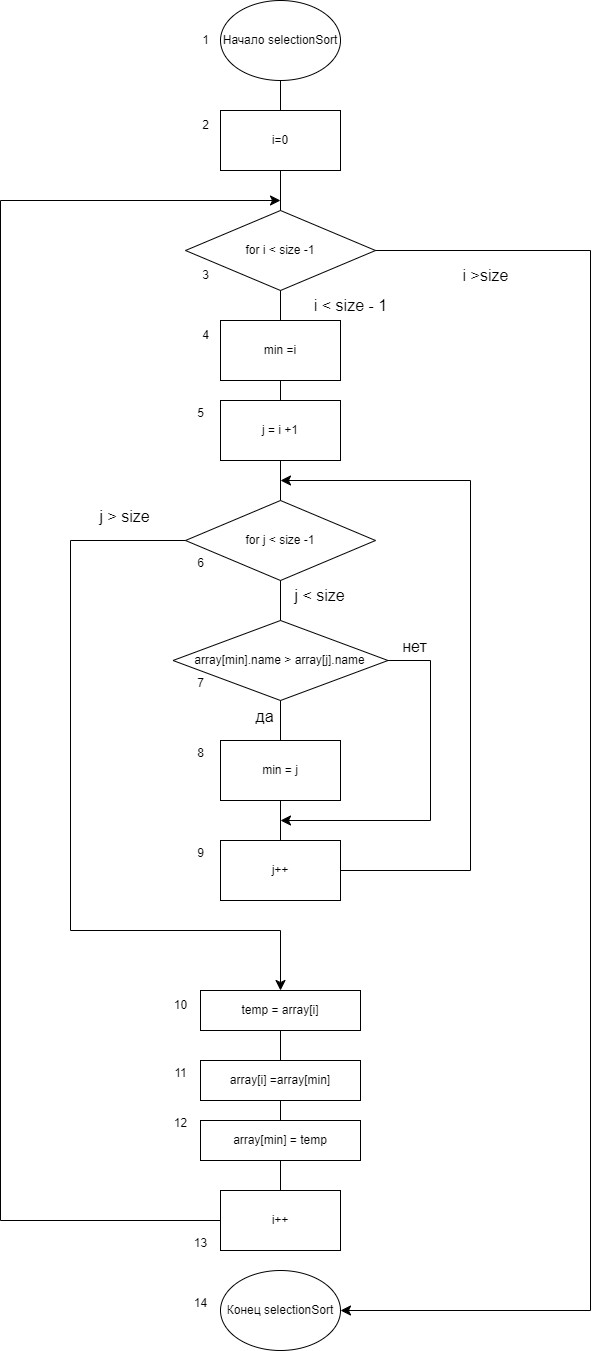


Рисунок 3.1 – Алгоритм функции selectionSort

Следующий алгоритм показывает выполнение функции main (рисунок 3.2).

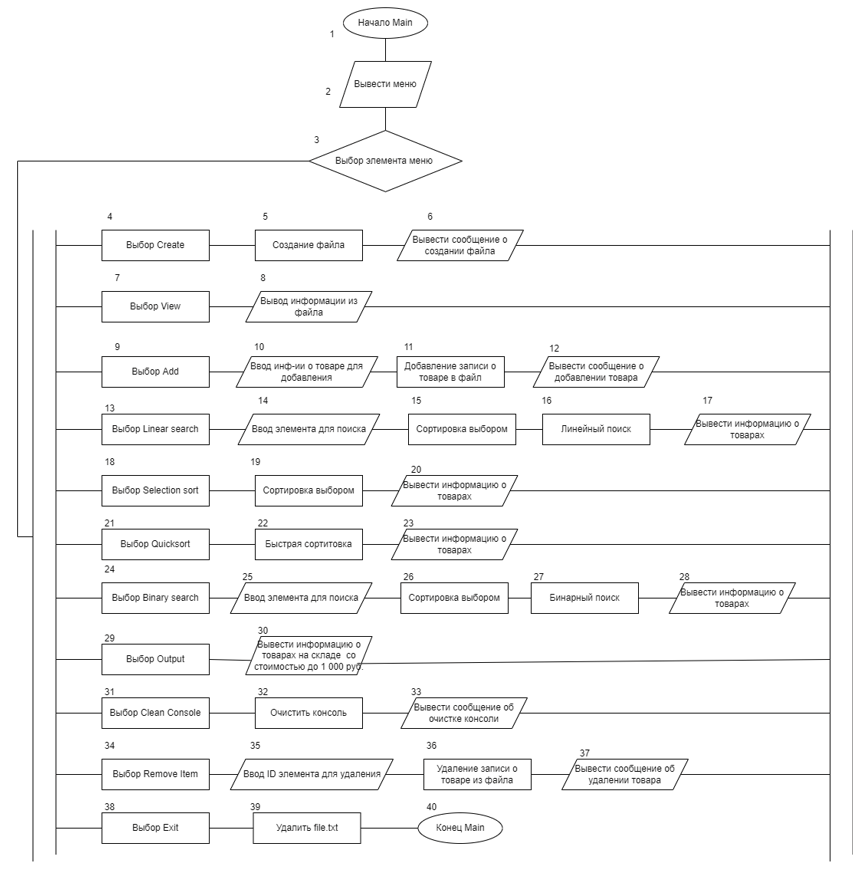


Рисунок 3.2 – Алгоритм главной функции main

## **Тестирование и проверка работоспособности программного средства**

Исключительные ситуации — ситуации, возникающие во время работы системы, выходящие за пределы, предусмотренные спецификацией.

При вводе неправильных данных о товаре, программа выведет сообщение об ошибке и будет просить ввод правильных данных, пока информация не будет введена верно (рисунок 4.1).

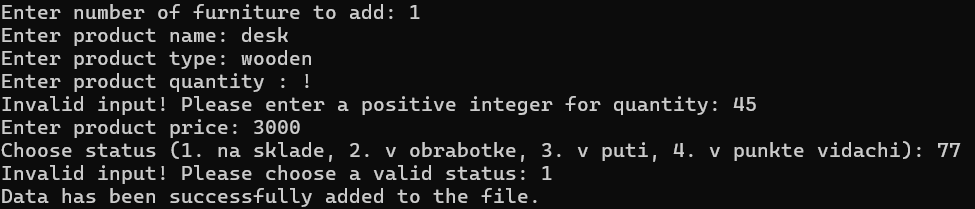


Рисунок 4.1 – Ввод названия, типа, количества, стоимости и статуса товара

На рисунке 4.2 показано меню продавца-консультанта.

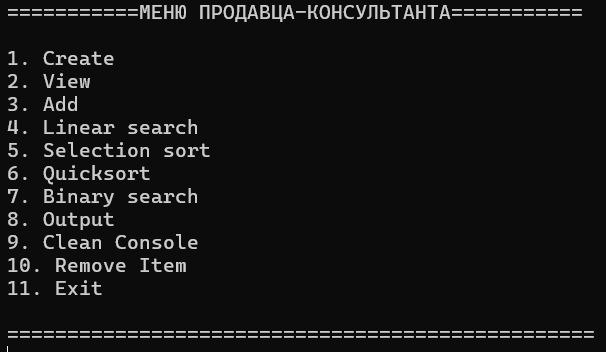


Рисунок 4.2 – Меню продавца-консультанта

Если пользователю необходимо просмотреть все записи о товарах, а в файле ничего нет, то при попытке выбрать данный пункт меню выведется сообщение об ошибке (рисунок 4.3).

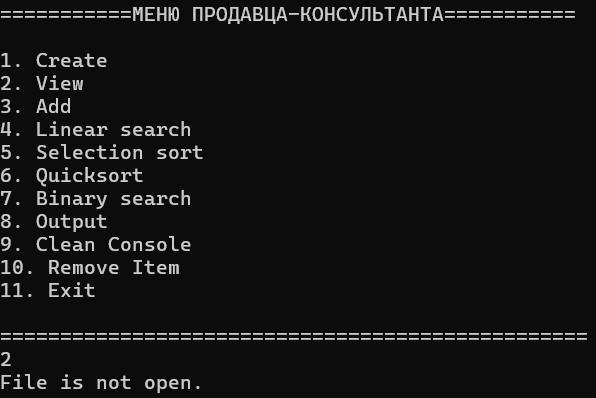


Рисунок 4.3 – Меню продавца-консультанта, когда нет информации о товарах

При удалении несуществующего товара выводится сообщение, что продукта с данным ID не существует (рисунок 4.4).

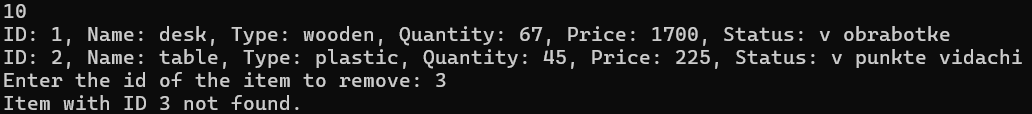


Рисунок 4.4 – Удаление несуществующей записи

## **Сквозное тестирование**

Сквозное тестирование — это процесс проверки всей системы или приложения от начала до конца, как если бы оно использовалось реальными пользователями. Основная цель этого тестирования - убедиться, что все компоненты системы работают вместе правильно и без ошибок

На рисунке 4.5 показано меню продавца-консультанта.

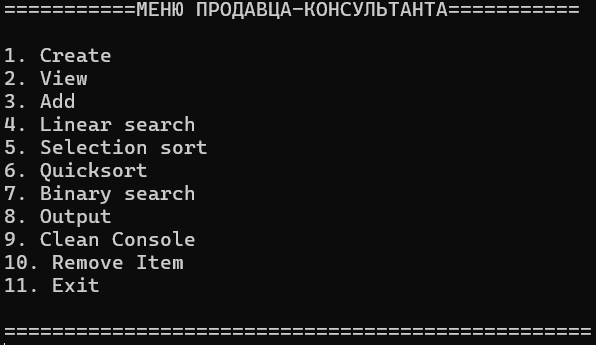


Рисунок 4.5 – Меню продавца консультанта

При входе в программу создаем файл и добавляем в него инфомацию о мебели (рисунок 4.6).

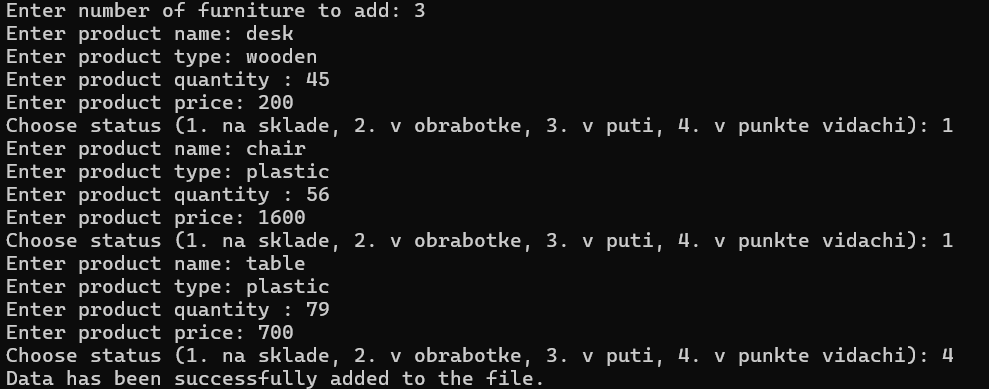


Рисунок 4.6 – Добавление информации о мебели

Информацию о мебели, которая хранится в файле можно вывести на экран (рисунок 4.7).

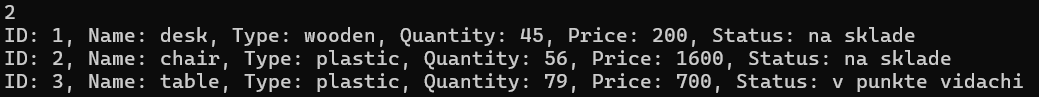


Рисунок 4.7 – Просмотр информации из файла

С помощью функции selection sort на экран выведется информация о мебели, отсортированной по названию (рисунок 4.8).

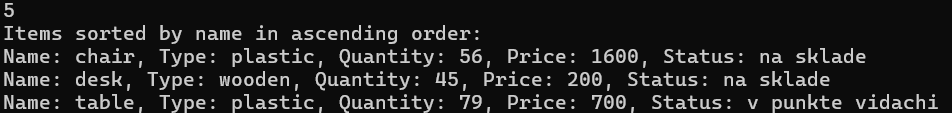


Рисунок 4.8 – Вывод информации после вызова функции selection sort

С помощью функции quick sort на экран выведется информация о мебели, отсортированнной по стоимости (рисунок 4.9).

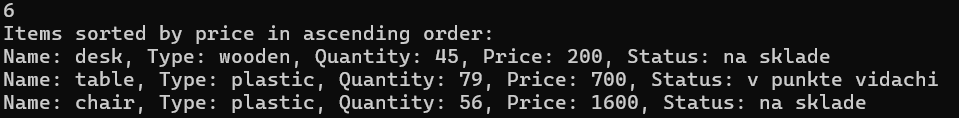


Рисунок 4.9 – Вывод информации после вызова функции quick sort

Функция linear search ищет информацию о мебели в массиве, который отсортирован по имени, и выводит индекс, где была найдена соответствующая мебель, начиная с 0. Ключом поиска является название мебели, которое пользователь вводит с клавиатуры (рисунок 4.10).

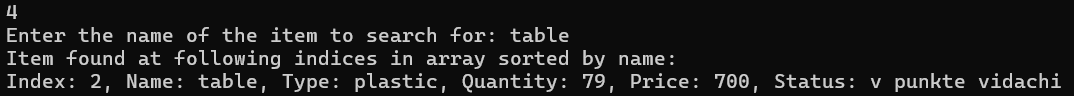


Рисунок 4.10 – Вывод информации после вызова функции linear search

Аналогично функции linear search выведется информация о мебели при вызове функции binary search (рисунок 4.11).

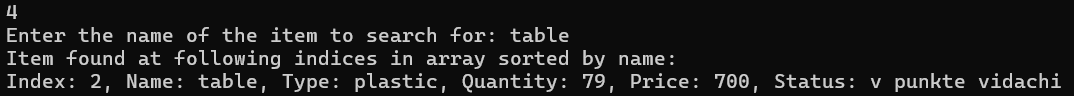


Рисунок 4.11 – Вывод информации после вызова функции binary search

Восьмой пункт — это индивидуальное задание. Согласно условию, надо вывести полную информацию об изделии стоимостью до 1000 рублей и имеющим статус на складе. На рисунке 4.12 можно увидеть работоспособность кода.

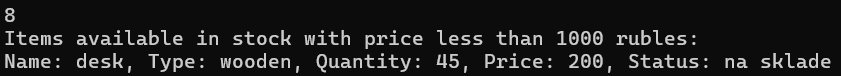


Рисунок 4.12 – Индивидуальное задание

Также предусмотрена очистка консоли для улучшения читаемости кода (рисунок 4.13).

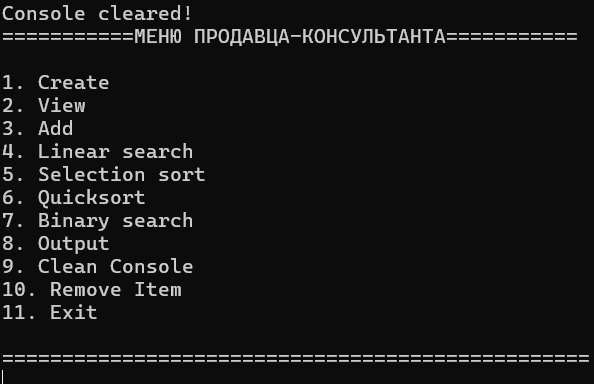


Рисунок 4.13 – Очистка консоли

На рисунке 4.14 продемонстрирована возможность удаления записи о товаре из файла по его ID.

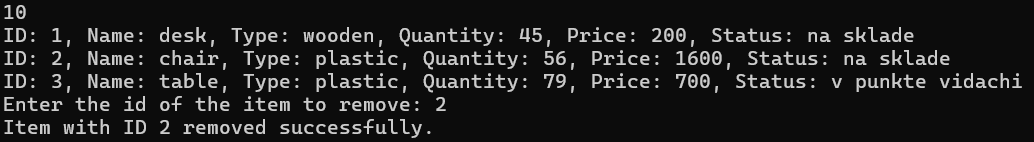


Рисунок 4.14 – Удаление записи о мебели

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения данной курсовой работы была разработана программа для управления ассортиментом мебельного магазина. Программа автоматизирует процесс учёта и обработки информации о наличии товаров, их характеристиках и статусах. Имеется возможность добавлять новые товары, просматривать имеющийся ассортимент, удалять позиции из списка, а также осуществлять поиск товаров по их названиям.

Программа обладает простым и интуитивно понятным интерфейсом, что делает её использование удобным для продавцов-консультантов магазина. В процессе разработки были применены методы сортировки и поиска данных, такие как сортировка выбором и быстрая сортировка, а также линейный и двоичный поиск. Эти методы обеспечивают эффективную работу с большим объёмом данных о товарах.

Были рассмотрены исключительные ситуации и их обработка с помощью функций проверок (на основе языка С++), создан удобный интерфейс.

Также были рассмотрены различные сценарии работы с программой, включая создание файла для хранения данных, вывод информации на экран, удаление товаров из списка, а также очистка консоли для удобства работы.

В ходе разработки программного средства были выполнены следующие шаги:

* разработаны проектные решения программного средства на основе языка UML;
* разработаны блок-схемы алгоритмов работы программы;
* разработан интерфейс приложения;
* разработаны структуры и модули приложения;
* проведено тестирование программного продукта.

Несомненно, данное приложение можно усовершенствовать, добавив больше информации по теме, функций, что могло бы позволить сделать спектр решаемых задач шире.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Бхаргава, А. Грокаем алгоритмы : иллюстрированное пособие для программистов и любопытствующих / А. Бхаргава. — СПб : Питер, 2017. — 288 с.
2. Сложность алгоритмов. Разбор Big O [Электронный ресурс]. — Режим доступа : <https://habr.com/ru/articles/782608/>. — Дата доступа : 12.05.2024.
3. Нижельская, М. UML диаграммы / М. Нижельская [Электронный ресурс]. — Режим доступа : <https://practicum.yandex.ru/blog/uml-diagrammy/>. — Дата доступа : 10.05.2024.
4. Основы алгоритмизации и программирования (язык С/С++). Лабораторный практикум в 2 ч. Ч. 2 : учеб.-метод. пособие / Беспалов С. А. [и др.] . — Минск : БГУИР, 2018. — 114 с.
5. Как улучшить блок-схемы алгоритмов? [Электронный ресурс]. — Режим доступа : <https://habr.com/ru/articles/541478/>. — Дата доступа : 10.05.2024.

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

Файл main.cpp

#include "file\_operations.h"

#include "sorting.h"

#include "searching.h"

int main() {

const int MAX\_ITEMS = 100;

Furniture items[MAX\_ITEMS];

int itemCount = 0;

int ch;

bool active = true;

while (active) {

cout << "===========МЕНЮ ПРОДАВЦА-КОНСУЛЬТАНТА===========\n\n";

cout << "1. Create\n";

cout << "2. View\n";

cout << "3. Add\n";

cout << "4. Linear search\n";

cout << "5. Selection sort\n";

cout << "6. Quicksort\n";

cout << "7. Binary search\n";

cout << "8. Output\n";

cout << "9. Clean Console\n";

cout << "10. Remove Item\n";

cout << "11. Exit\n\n";

cout << "=================================================\n";

cin >> ch;

switch (ch) {

case 1:

create();

itemCount = 0;

break;

case 2:

view();

break;

case 3:

add(items, itemCount);

break;

case 4:

{

string itemName;

cout << "Enter the name of the item to search for: ";

cin >> itemName;

selectionSort(items, itemCount);

int count;

int\* indices = linearSearch(items, itemCount, itemName, count);

if (indices != nullptr) {

cout << "Item found at following indices in array sorted by name:" << endl;

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

int foundIndex = indices[i];

cout << "Index: " << foundIndex << ", ";

printItem(items[foundIndex]);

}

delete[] indices;

}

else {

cout << "Item not found." << endl;

}

break;

}

case 5:

selectionSort(items, itemCount);

cout << "Items sorted by name in ascending order:" << endl;

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

printItem(items[i]);

}

break;

case 6:

quickSort(items, 0, itemCount - 1);

cout << "Items sorted by price in ascending order: " << endl;

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

printItem(items[i]);

}

break;

case 7:

{

string itemName;

cout << "Enter the name of the item to search for: ";

cin >> itemName;

selectionSort(items, itemCount);

int count;

int\* indices = binarySearch(items, itemCount, itemName, count);

if (indices != nullptr) {

cout << "Item found at following indices in array sorted by name:" << endl;

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

int foundIndex = indices[i];

cout << "Index: " << foundIndex << ", ";

printItem(items[foundIndex]);

}

delete[] indices;

}

else {

cout << "Item not found." << endl;

}

break;

}

case 8:

outputAvailableItems(items, itemCount);

break;

case 9:

system("cls");

cout << "Console cleared!\n";

break;

case 10:

removeItem(items, itemCount);

break;

case 11:

remove("file.txt");

active = false;

break;

}

}

return 0;

}

Файл file\_operations.h

#ifndef FILE\_OPERATIONS\_H

#define FILE\_OPERATIONS\_H

#include <fstream>

#include <string>

#include <iostream>

using namespace std;

struct Furniture {

int id;

string name;

string type;

int quantity;

double price;

string status;

};

void printItem(const Furniture& item);

void add(Furniture array[], int& size);

void view();

void create();

void removeItem(Furniture array[], int& size);

#endif

Файл file\_operations.cpp

#include "file\_operations.h"

void printItem(const Furniture& item) {

cout << "Name: " << item.name << ", Type: " << item.type

<< ", Quantity: " << item.quantity << ", Price: " << item.price

<< ", Status: ";

if (item.status == "1") cout << "na sklade" << endl;

else if (item.status == "2") cout << "v obrabotke" << endl;

else if (item.status == "3") cout << "v puti" << endl;

else if (item.status == "4") cout << "v punkte vidachi" << endl;

}

void add(Furniture array[], int& size) {

ofstream file("file.txt", ios::out | ios::app);

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

cout << "File is not open." << endl;

return;

}

else {

int n;

cout << "Enter number of furniture to add: ";

cin >> n;

if (!cin || n <= 0) {

cout << "Invalid input! Please enter a valid number." << endl;

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return;

}

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

Furniture item;

item.id = size + 1; // Присваиваем ID

cout << "Enter product name: ";

cin.ignore();

do {

getline(cin, item.name);

if (item.name.find(' ') != string::npos) {

cout << "Product name must be a single word. Please enter again: ";

}

} while (item.name.find(' ') != string::npos);

cout << "Enter product type: ";

do {

getline(cin, item.type);

if (item.type.find(' ') != string::npos) {

cout << "Product type must be a single word. Please enter again: ";

}

} while (item.type.find(' ') != string::npos);

cout << "Enter product quantity : ";

do {

cin >> item.quantity;

if (cin.fail() || item.quantity <= 0) {

cout << "Invalid input! Please enter a positive integer for quantity: ";

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

} while (cin.fail() || item.quantity <= 0);

cout << "Enter product price: ";

do {

cin >> item.price;

if (cin.fail() || item.price <= 0) {

cout << "Invalid input! Please enter a positive number for price: ";

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

} while (cin.fail() || item.price <= 0);

cout << "Choose status (1. na sklade, 2. v obrabotke, 3. v puti, 4. v punkte vidachi): ";

do {

cin >> item.status;

if (item.status != "1" && item.status != "2" && item.status != "3" && item.status != "4") {

cout << "Invalid input! Please choose a valid status: ";

}

} while (item.status != "1" && item.status != "2" && item.status != "3" && item.status != "4");

file << item.id << " " << item.name << " " << item.type << " " << item.quantity << " " << item.price << " " << item.status << endl;

array[size++] = item;

}

cout << "Data has been successfully added to the file." << endl;

}

file.close();

}

void view() {

ifstream file("file.txt");

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

cout << "File is not open." << endl;

return;

}

else {

Furniture item;

while (file >> item.id >> item.name >> item.type >> item.quantity >> item.price >> item.status) {

cout << "ID: " << item.id << ", ";

printItem(item);

}

}

file.close();

}

void create() {

ofstream newFile("file.txt", ios::trunc);

if (newFile.is\_open()) {

cout << "File has been successfully created." << endl;

}

else {

cout << "Failed to create file." << endl;

}

newFile.close();

}

void removeItem(Furniture array[], int& size) {

view();

int id;

cout << "Enter the id of the item to remove: ";

cin >> id;

int foundIndex = -1;

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

if (array[i].id == id) {

foundIndex = i;

break;

}

}

if (foundIndex != -1) {

for (int i = foundIndex; i < size - 1; ++i) {

array[i] = array[i + 1];

}

size--;

cout << "Item with ID " << id << " removed successfully." << endl;

ifstream inFile("file.txt");

ofstream tempFile("temp.txt");

Furniture item;

while (inFile >> item.id >> item.name >> item.type >> item.quantity >> item.price >> item.status) {

if (item.id != id) {

tempFile << item.id << " " << item.name << " " << item.type << " " << item.quantity << " " << item.price << " " << item.status << endl;

}

}

inFile.close();

tempFile.close();

remove("file.txt");

rename("temp.txt", "file.txt");

}

else {

cout << "Item with ID " << id << " not found." << endl;

}

}

Файл searching.h

#ifndef SEARCHING\_H

#define SEARCHING\_H

#include "file\_operations.h"

int\* linearSearch(Furniture array[], int size, string itemName, int& count);

int\* binarySearch(Furniture array[], int length, string itemName, int& count);

void outputAvailableItems(Furniture array[], int size);

#endif

Файл searching.cpp

#include "searching.h"

int\* linearSearch(Furniture array[], int size, string itemName, int& count) {

int\* indices = new int[size];

count = 0;

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

if (array[i].name == itemName) {

indices[count++] = i;

}

}

if (count == 0) {

delete[] indices;

return nullptr;

}

return indices;

}

int\* binarySearch(Furniture array[], int length, string itemName, int& count) {

int\* indices = new int[length];

count = 0;

int low = 0;

int high = length - 1;

while (low <= high) {

int middle = (low + high) / 2;

string itemNameInArray = array[middle].name;

if (itemNameInArray < itemName)

low = middle + 1;

else if (itemNameInArray > itemName)

high = middle - 1;

else {

indices[count++] = middle;

int left = middle - 1;

while (left >= low && array[left].name == itemName) {

indices[count++] = left;

left--;

}

int right = middle + 1;

while (right <= high && array[right].name == itemName) {

indices[count++] = right;

right++;

}

return indices;

}

}

delete[] indices;

return nullptr;

}

void outputAvailableItems(Furniture array[], int size) {

cout << "Items available in stock with price less than 1000 rubles:" << endl;

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

if (array[i].status == "1" && array[i].price <= 1000) {

printItem(array[i]);

}

}

}

Файл sorting.h

#ifndef SORTING\_H

#define SORTING\_H

#include "file\_operations.h"

void selectionSort(Furniture array[], int size);

void quickSort(Furniture array[], int start, int end);

int partition(Furniture array[], int start, int end);

#endif

Файл sorting.cpp

#include "sorting.h"

void selectionSort(Furniture array[], int size) {

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

int min = i;

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

if (array[min].name > array[j].name) {

min = j;

}

}

Furniture temp = array[i];

array[i] = array[min];

array[min] = temp;

}

}

void quickSort(Furniture array[], int start, int end) {

if (end <= start) return;

int pivot = partition(array, start, end);

quickSort(array, start, pivot - 1);

quickSort(array, pivot + 1, end);

}

int partition(Furniture array[], int start, int end) {

double pivot = array[end].price;

int i = start - 1;

for (int j = start; j <= end; j++) {

if (array[j].price < pivot) {

i++;

Furniture temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

i++;

Furniture temp = array[i];

array[i] = array[end];

array[end] = temp;

return i;

}