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 товаров на складе»

БГУИР КП 6-05-0611-03 005 ПЗ

Студент: гр. 321702 Банкевич Я.Д.

Руководитель: Семижон Е.А.

Минск 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc500746007)

[1 СТРУКТУРЫ И ФАЙЛЫ](#_Toc500746008) 5

[1.1 Структуры данных](#_Toc500746009) 5

[1.2 Файлы](#_Toc500746009) 6

[2 АЛГОРИТМЫ СОРТИРОВКИ 8](#_Toc500746011)

[2.1 Сортировка методом пузырька 9](#_Toc500746012)

[2.2 Шейкер-сортировка 10](#_Toc500746013)

[2.3 Сортировка методом прямого выбора 12](#_Toc500746013)

[2.4 Сортировка методом простого включения 13](#_Toc500746013)

[2.5 Быстрая сортировка 14](#_Toc500746013)

[3 АЛГОРИТМЫ ПОИСКА 1](#_Toc500746015)6

[3.1 Линейный поиск 16](#_Toc500746013)

[3.2 Двоичный поиск 17](#_Toc500746013)

[3.3 Интерполяционный поиск 18](#_Toc500746013)

[4 ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ 20](#_Toc500746015)

[4.1 Основные функции 20](#_Toc500746013)

[4.1.1 Функция универсального пользовательского выбора ChooseFunction() 20](#_Toc500746013)

[4.1.2 Функция создания файла CreateNewFile() 21](#_Toc500746013)

[4.1.3 Функция просмотра содержимого файла ViewFile() 21](#_Toc500746013)

[4.1.4 Функция для добавления информации о товаре AddInfo() 21](#_Toc500746013)

[4.1.5 Функция для ввода даты InputDate() 22](#_Toc500746013)

[4.1.6 Функция линейного поиска LinearSearch() 22](#_Toc500746013)

[4.1.7 Функции бинарного поиска 23](#_Toc500746013)

[4.1.7.1 Функция BinarySearchShow() 23](#_Toc500746013)

[4.1.7.2 Функция BinarySearch() 23](#_Toc500746013)

[4.1.8 Функции сортировки методом прямого выбора 24](#_Toc500746013)

[4.1.8.1 Функция DirectSelectionSortShow() 24](#_Toc500746013)

[4.1.8.2 Функция DirectSelectionSort() 24](#_Toc500746013)

[4.1.9 Функции быстрой сортировки 25](#_Toc500746013)

[4.1.9.1 Функция QuickSortShow() 25](#_Toc500746013)

[4.1.9.2 Функция QuickSort() 25](#_Toc500746013)

[4.1.10 Функция отбора товаров SeeResult() 25](#_Toc500746013)

[4.1.11 Функция редактирования информации о товаре Editor() 26](#_Toc500746013)

[4.1.12 Функция удаления информации о товаре DeleteProduct() 26](#_Toc500746013)

[4.2 Вспомогательные функции 27](#_Toc500746013)

[4.2.1 Функция для считывания данных из файла GetFileIntoStruct() 27](#_Toc500746013)

[4.2.2 Функция для преобразования данных из массива структура в файл StructIntoFileAndConsole() 28](#_Toc500746013)

[4.2.3 Функция для определения високосного года IsLeapYear() 28](#_Toc500746013)

[4.2.4 Функция для определения максимального количества дней в месяце MaxDays() 29](#_Toc500746013)

[4.2.5 Функция нахождения первого нецифрового элемента FirstNotDigit.29](#_Toc500746013)

[5 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ 30](#_Toc500746015)

[ЗАКЛЮЧЕНИЕ 32](#_Toc500746020)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33](#_Toc500746021)

[ПРИЛОЖЕНИЕ А *(листинг кода)* 34](#_Toc500746021)

[ПРИЛОЖЕНИЕ Б *(блок-схема)* 52](#_Toc500746020)

ВВЕДЕНИЕ

В современном мире эффективное управление товарами на складе является неотъемлемым элементом успешной деятельности любого предприятия или организации, работающих в сфере торговли и производства. Оно обеспечивает не только оптимальное использование ресурсов, но и удовлетворение потребностей клиентов в своевременной поставке продукции, а также позволяет работникам логистических центров и обычных складских помещений отслеживать статус товара в режиме реального времени. Однако, для достижения этой эффективности необходимо не только наличие товаров на складе и информации о них, но и правильная организация их структуры.

Структура списка товаров на складе играет ключевую роль в обеспечении порядка и эффективности операций по отслеживанию, хранению и распределению товарных позиций. Этот аспект управления не просто определяет местоположение товаров в складском помещении, но и влияет на скорость доступа к ним, сроки их обработки и, следовательно, уровень удовлетворенности клиентов.

Мой курсовой проект является одним из примеров программ, позволяющих повысить эффективность управления товарами на складе. С его помощью пользователь может с легкостью отслеживать статус товаров на складе, работать с ними, а также быстро находить определенную продукцию.

Целью данного курсового проекта является разработка программы, которая автоматизирует процессы управления запасами на складе и оптимизирует структуру списка товаров.

Задачи, которые необходимо выполнить в ходе выполнения курсового проекта для достижения поставленных целей:

* исследовать технологии и тенденции в области управления запасами на складах;
* ознакомиться с уже существующими программами и приложениями, позволяющими оптимизировать управление товарной продукцией на складе;
* разработать функциональную модель основного процесса;
* написать код программы, реализующий функционал, необходимый для управления структурой списка товаров на складе;
* провести тестирование разработанного программного продукта;
* выполнить отладку ошибок, обнаруженных в процессе тестирования.

Это исследование имеет большое значение для практики, поскольку позволяет повысить эффективность управления запасами на складе, а также способствует совершенствованию процессов складского хозяйства в целом и повышению качества обслуживания клиентов.

1. СТРУКТУРЫ И ФАЙЛЫ

**1.1 Структуры данных**

Структуры данных представляют собой важную программную концепцию, которая обеспечивает возможность эффективного хранения и обработки данных, связанных как логически, так и однотипно; т.е. это способ организации и хранения данных в компьютере.

Структуры данных являются неотъемлемой частью разработки алгоритмов, предназначенных для решения сложных задач, и играют важную роль в формировании производительности и функциональности программного обеспечения. Они предоставляют эффективные методы организации данных, что позволяет оптимизировать использование доступной памяти компьютера. Эффективное управление памятью, в свою очередь, делает программы более быстрыми и эффективными, особенно в случаях работы с обширными объемами данных.

**Классификация структур данных**

Таблица 1. Классификация структур данных

В языках программирования существуют различные встроенные библиотеки, которые упрощают работу со структурами данных. Например, в языке С++ это контейнеры STL.

Структуры данных предоставляют широкий спектр возможностей:

* Логическая структуризация и группировка информации, следовательно, код становится более понятным и легко поддерживаемым.
* Эффективное хранение уникальных элементов, исключая дублирование информации (например, с помощью хеш-таблиц).
* Оптимизация процессов поиска, добавления, удаления и обновления данных, что повышает производительность программ.
* Экономное использование памяти (например, с помощью битовых и компактных массивов, что особенно важно при работе с большими объемами данных).
* Упрощение выявления и устранения ошибок, так как правильно спроектированные структуры данных могут предотвратить множество потенциальных проблем.

**1.2 Файлы**

Файл ̶ это совокупность данных, организованных и хранящихся на внешнем носителе, которые рассматриваются программами как единое целое в процессе их обработки. Они играют ключевую роль в хранении и передаче информации, предназначенной для долгосрочного сохранения и использования.

Файлы являются неотъемлемой частью работы с компьютером, позволяя пользователям сохранять, обрабатывать и передавать данные между различными приложениями и устройствами. Они также обеспечивают удобный способ хранения информации, обеспечивая доступ к данным в любое время и из любого места.

**Виды файлов**

Таблица 2. Виды файлов

Во многих языках программирования существуют стандартные библиотеки, которые обеспечивают удобные и эффективные средства для работы с файлами, и они часто используются для чтения, записи, создания и удаления файлов, а также для манипуляций с файловой системой. Например, в С++ можно использовать встроенную библиотеку <fstream>.

**Основные функции работы с файлами в ЯП:**

* Открытие файлов;
* Чтение данных из файлов;
* Запись данных в файлы;
* Закрытие файлов;
* Управление текущей позицией в файле;
* Управление атрибутами файлов и директорий

2.АЛГОРИТМЫ СОРТИРОВКИ

*Сортировка* представляет собой процесс упорядочивания элементов по определенному критерию. Этот процесс может включать последовательное распределение элементов по возрастанию или убыванию значений, либо разбиение на группы в соответствии с выбранным критерием. *Алгоритм сортировки* - это набор инструкций, разработанных для систематического переупорядочивания элементов в списке с целью достижения желаемого порядка.

В случае, если элементы списка представляют собой структуры с несколькими полями данных, сортировка обычно выполняется по одному из этих полей, называемому *ключом сортировки*. Ключ сортировки определяет порядок, в котором элементы должны быть упорядочены.

Допустим, данные для сортировки представлены символьным типом, тогда алгоритм сортировки может упорядочить элементы в списке по алфавиту, основываясь на значении символов в ключе сортировки. Например, для списка слов или строк, каждое из которых содержит символы, алгоритм сортировки может организовать элементы таким образом, чтобы они следовали в алфавитном порядке. Это означает, что элементы списка будут расположены в порядке, определяемом последовательностью символов в каждом элементе. Если ключом сортировки является первая буква слова или строки, то элементы будут упорядочены по алфавиту исходя из их первых символов. Если ключом сортировки является целая строка, то элементы будут упорядочены в соответствии с алфавитным порядком этой строки.

Алгоритмы сортировки играют важную роль в компьютерных науках и информатике, поскольку они широко используются в различных областях, включая базы данных, анализ данных, компьютерную графику, алгоритмы машинного обучения и многое другое, эффективно организовывая информацию. Оптимальный выбор алгоритма сортировки зависит от характеристик данных, которые требуется отсортировать, таких как объем данных, тип данных и требуемая производительность.

Важно отметить, что существует множество алгоритмов сортировки, каждый из которых имеет свои особенности и подходит для разных целей. Они оцениваются двум главным критериям:

* Время (вычислительная сложность) – параметр, определяющий быстродействие алгоритма;
* Память (эффективность использования памяти) – требует ли алгоритм использования дополнительной памяти для временного хранения данных.

Кроме того, каждая сортировка характеризуется такими свойствами, как *устойчивость* (меняются ли элементы с одинаковыми ключами), *естественность поведения* (эффективность при работе с уже ранее упорядоченными данными), *использование операции сравнения, сфера применения алгоритма*.

**2.1** **Сортировка методом пузырька**

*Сортировка методом «пузырька» (пузырьковая сортировка bubble sort)* – самый известный метод сортировки, который прост для понимания и реализации, но эффективен лишь для небольших массивов.

*Алгоритм:*

* Проходим по массиву и сравниваем элементы попарно:
  + Если порядок элементов в паре неправильный (например, для сортировки по возрастанию: если первый элемент больше второго), то меняем их местами;
  + Если порядок в паре правильный или равные элементы, оставляем их на своих местах;
* После первого прохода самый крупный (или самый маленький, в зависимости от порядка сортировки) элемент оказывается на последнем месте списка;
* Повторяем этот процесс для оставшихся элементов списка (исключая уже отсортированные на каждой итерации);
* Продолжаем проходить по списку до тех пор, пока все элементы не будут упорядочены.

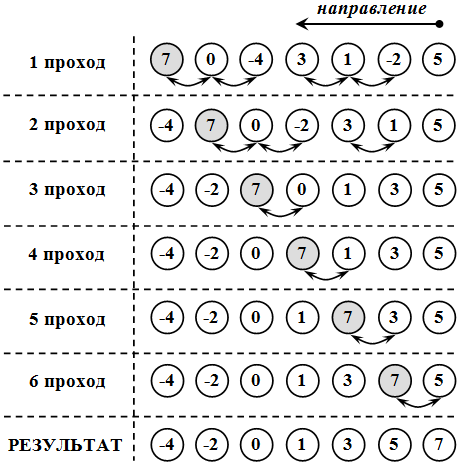


Рисунок 1. Пример работы сортировки методом пузырька

*Реализация метода пузырьковой сортировки на С++:*

void BubbleSort(int mas[], int size)

{

int per = 0;

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

{

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

{

if (mas[j] > mas[j + 1])

{

per = mas[j];

mas[j] = mas[j + 1];

mas[j + 1] = per;

}

}

}

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

{

cout << mas[i] << " ";

}

cout << endl;

}

**2.2 Шейкер-сортировка**

*Шейкер-сортировка (shaker sort, сортировка взбалтыванием)* – улучшенная версия пузырьковой сортировки. В отличие от метода BubbleSort, шейкер-сортировка проходит по списку то в одном направлении, то в другом, что может улучшить производительность в некоторых случаях.

*Алгоритм:*

* Начинаем сравнивать и менять элементы попарно, как в пузырьковой сортировке, с левого конца списка к правому.
* После прохода от левого конца списка к правому, начинаем проходить по списку в обратном направлении от правого конца к левому.
* Процесс повторяется до тех пор, пока не будет совершено ни одной перестановки элементов.

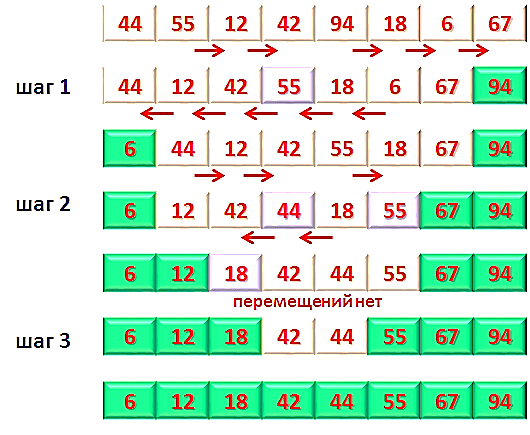


Рисунок 2. Пример работы шейкер-сортировки

*Реализация метода шейкер-сортировки на С++:*

void ShakerSort(int k, int x[max])

{

int i, t;

bool exchange;

do

{

exchange = false;

for (i = k - 1; i > 0; --i)

{

if (x[i - 1] > x[i])

{

t = x[i - 1];

x[i - 1] = x[i];

x[i] = t;

exchange = true;

}

}

}

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

{

if (x[i - 1] > x[i])

{

t = x[i - 1];

x[i - 1] = x[i];

x[i] = t;

exchange = true;

}

}

} while (exchange);

**2.3 Сортировка методом простого выбора**

*Метод простого выбора (сортировка выбора)* – пример простых алгоритмов сортировки, эффективен на небольших массивах. Его суть заключается в следующем: проходим по списку несколько раз, выбирая минимальный (иди максимальный, в зависимости от нужного результата) элемент, переставляя его на соответствующее место.

*Алгоритм:*

* Находим минимальный (максимальный) элемент в списке;
* Меняем найденный элемент с первым элементом списка;
* Повторяем процесс для оставшейся части списка, исключая уже отсортированные элементы;
* Сортировка продолжается до тех пор, пока не будут отсортированы все элементы.

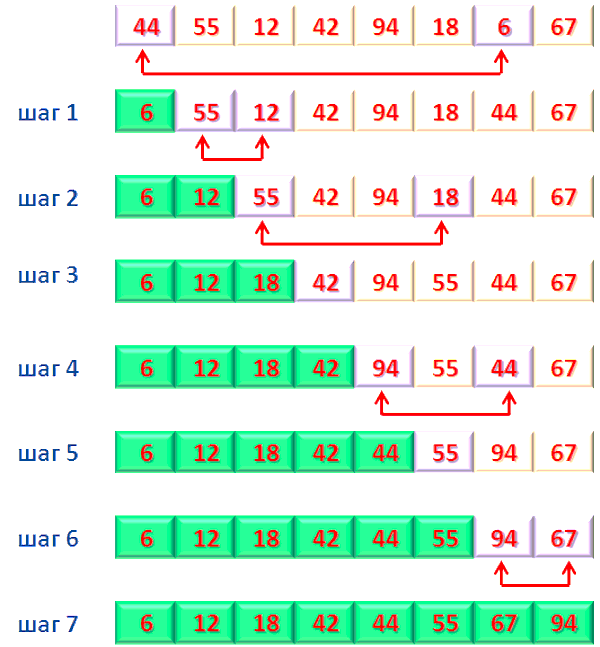


Рисунок 3. Пример работы сортировки методом простого выбора

*Реализация метода сортировки простого выбора на С++:*

void SelectionSort(int k, int x[max])

{

int i, j, min, temp;

for (i = 0; i < k - 1; i++)

{

min = i;

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

{

if (x[j] < x[min])

min = j;

}

temp = x[i];

x[i] = x[min];

x[min] = temp;

}

}

**2.4 Сортировка методом простого включения**

*Метод простого включения (сортировка вставками)* –пример устойчивого простого алгоритма сортировки, который эффективен на уже отсортированных наборах данных и не требует дополнительной памяти. В отличие от метода простого выбора, вместо нахождения минимального (максимального) элемента и его перемещения в начало списка, метод простого включения поочередно добавляет каждый элемент на его правильное место в уже отсортированной части массива.

*Алгоритм:*

* Начинаем сортировку с первого элемента, считая его уже отсортированным;
* Берем следующий элемент и вставляем его на правильное место в уже отсортированной части списка;
* Повторяем это действие для оставшихся элементов списка, переставляя каждый элемент на правильное место;
* Сортировка продолжается, пока все элементы массива не окажутся на своем месте.

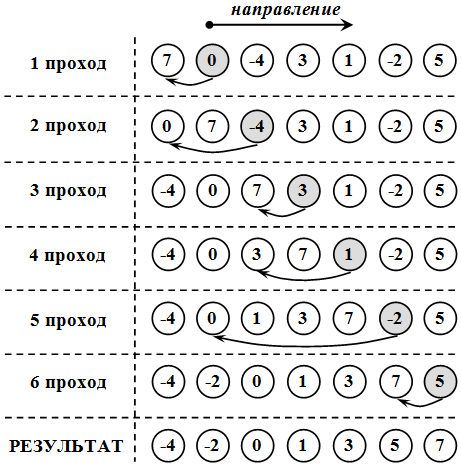


Рисунок 4. Пример работы сортировки методом простого включения

*Реализация сортировки методом простого включения на С++:*

void InsertSort(int k, int x[max])

{

int i, j, temp;

for (i = 0; i < k; i++)

{

//цикл проходов, i - номер прохода

temp = x[i]; //поиск места элемента

for (j = i - 1; j >= 0 && x[j] > temp; j--)

x[j + 1] = x[j];

//сдвигаем элемент вправо, пока не дошли

//место найдено, вставить элемент

x[j + 1] = temp;

}

}

**2.5 Быстрая сортировка**

*Быстрая сортировка (QuickSort)* – один из наиболее эффективных алгоритмов сортировки. Она разбивает масив на меньшие подмассивы, сортирует их, а затем обратно объединяет в один. За счет этого намного быстрее можно отсортировать даже очень большие списки, конечно, если был правильно выбран опорный элемент.

*Алгоритм*:

* Индексы l и r инициализируются минимальным и максимальным значениями для массива;
* Определяется индекс опорного элемента (например, m=(l+r)/2);
* l увеличивается до m, пока не превысит опорный элемент;
* r уменьшается до m, пока не будет меньше опорного элемента;
* Если r и l становятся равными, операция завершена, оба индекса указывают на опорный элемент;
* Если l меньше r, обмениваются значениями и продолжается операция с новыми l и r. Причем, если граница (l или r) достигает опорного элемента, то при обмене значение m изменяется на элемент с индексом r или l соответственно;
* Рекурсивно сортируем подмассивы, расположенные слева и справа от опорного элемента.
* Базовым случаем рекурсии являются подмассивы, содержащие один или два элемента. В первом случае возвращаем подмассив без изменений, во втором случае производим перестановку элементов, если это необходимо. Таким образом, все такие подмассивы уже упорядочены в процессе разделения.

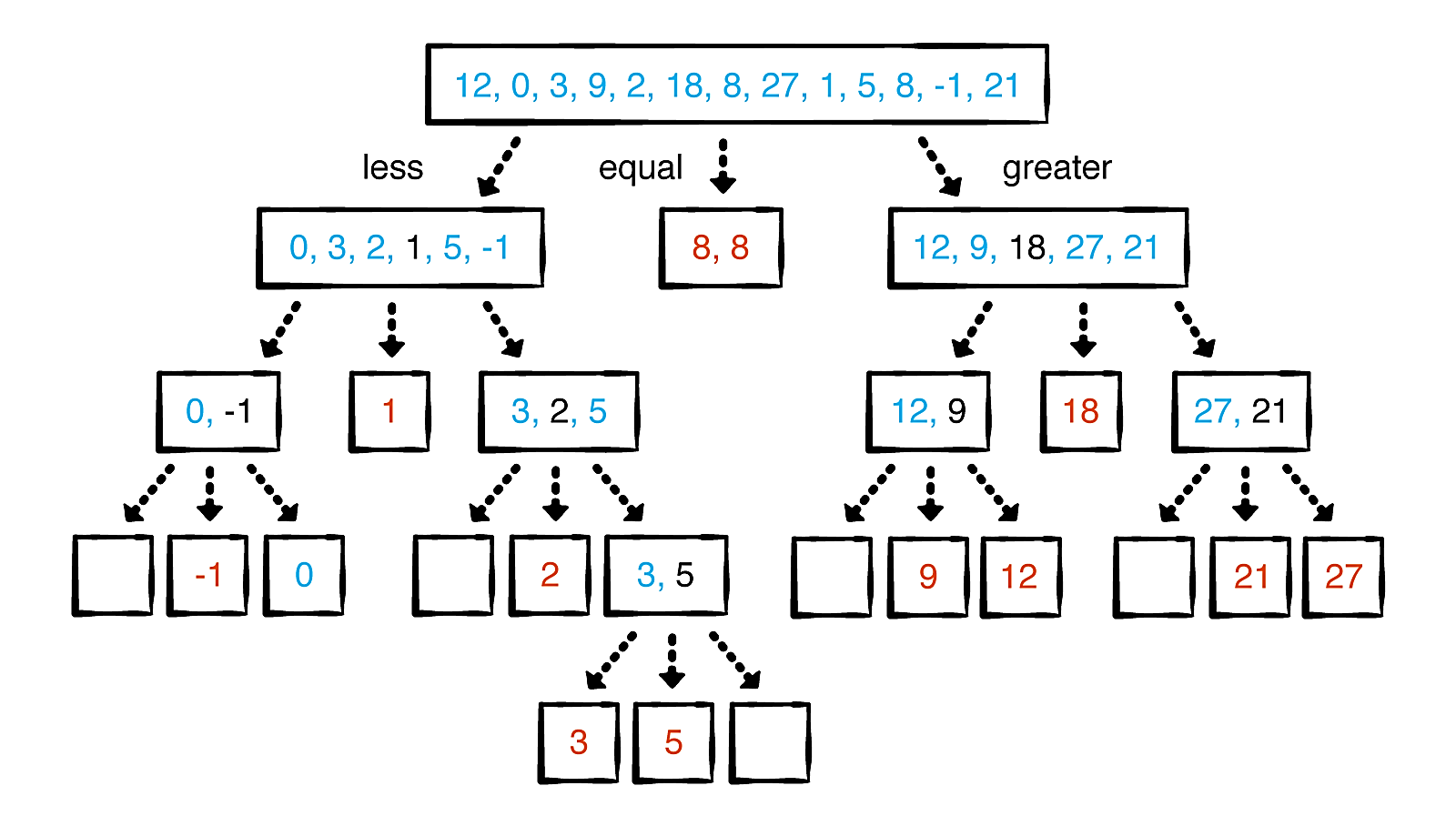


Рисунок 5. Пример работы быстрой сортировки

*Реализация сортировки методом* *QuickSort на С++:*

void QuickSort (int l, int r)

{

int w, x, i, j;

i = l;

j = r;

x = a[(l + r) / 2];

while (i <= j)

{

while (a[i] < x) i++;

while (x < a[j]) j--;

if (i <= j)

{

w = a[i]; a[i] = a[j]; a[j] = w;

i++; j--;

}

}

if (l < j) QuickSort (l, j);

if (i < r) QuickSort (i, r);

}

3. АЛГОРИТМЫ ПОИСКА

*Поиск* – это процесс нахождения нужной информации в структуре данных, таких как массив, дерево или список. Он используется для нахождения элемента с определенным значением (ключом) или выявления факта наличия конкретного элемента в структуре.

Поиск активно используется в различных областях программирования, например, при обработке данных, работе с базами данных, разработке алгоритмов и т.д. Он позволяет эффективно работать с большими объемами данных, обеспечивая быстрый доступ к необходимой информации.

Существуют разные виды поисков, используемых в программировании, которые отличаются друг от друга способом организации и характеристиками поискового алгоритма.

**3.1 Линейный поиск**

*Линейный поиск* – один из простых алгоритмов поиска, который эффективен в массивах с небольшим количеством элементов; работает и с неотсортированными и неупорядоченными массивами. Он последовательно просматривает каждый элемент в структуре данных до тех пор, пока не будет найден искомый элемент или пока не достигнет конца списка.

*Алгоритм:*

* Берем первый элемент;
* Сравниваем текущий элемент с искомым значением:
  + Если текущий элемент равен искомому, то возвращаем его индекс или отображаем сам элемент;
  + Если текущий элемент не равен исходному, то переходим к следующему элементу и повторяем те же шаги;
* Поиск продолжается до тех пор, пока не будет найден искомый элемент или пока не кончатся все элементы.

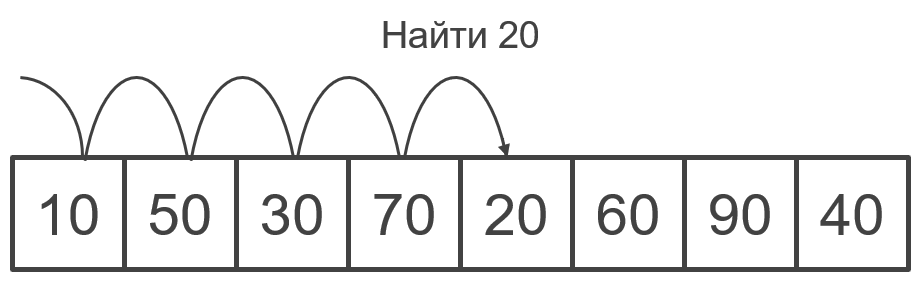


Рисунок 6. Пример работы линейного поиска

*Реализация линейного поиска на С++:*

* *1 способ:*

int LinearSearch1(int a[],int n,int x)

{

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

if (a[i] == x) return i;

return -1;

}

* *2 способ:*

int LinearSearch2(int a[],int n,int x){

a[n] = x;

int i = 0;

while (a[i] != x) i++;

if (i == n) return -1;

else return i;

}

**3.2 Двоичный поиск**

*Двоичный (бинарный) поиск* – эффективный алгоритм поиска элемента, который работает только с отсортированными массивами и позволяет эффективно работать с большими объемами данных. Он не используется для поиска максимального/минимального значения в структуре, так как они всегда находятся в начале и конце отсортированной структуры данный. Следовательно, бинарный поиск используется только в тех случаях, когда необходимо найти элемент по ключу (определенному значению в массиве).

*Алгоритм:*

* Берем отсортированный массив данных;
* Находим середину массива (mid=(left+right)/2, где left и right это крайние значения индексов элементов в массиве);
* Сравниваем элемент, находящийся в середине массива (а[mid]) со значением ключа:
  + Если искомое значение равно значению «серединного элемента», то возвращаем его индекс (или его значение);
  + Если искомое значение не совпадает со значением элемента:
    - Если искомое значение меньше чем элемент, повторяем поиск в левой половине массива;
    - Если искомое значение больше сем элемент, повторяем поиск в правой половине массива;
* Повторяем этот процесс для новой половины списка;
* Поиск продолжается до тех пор, пока не будет найден элемент, значение которого совпадает со значением ключа или пока не закончатся все элементы.

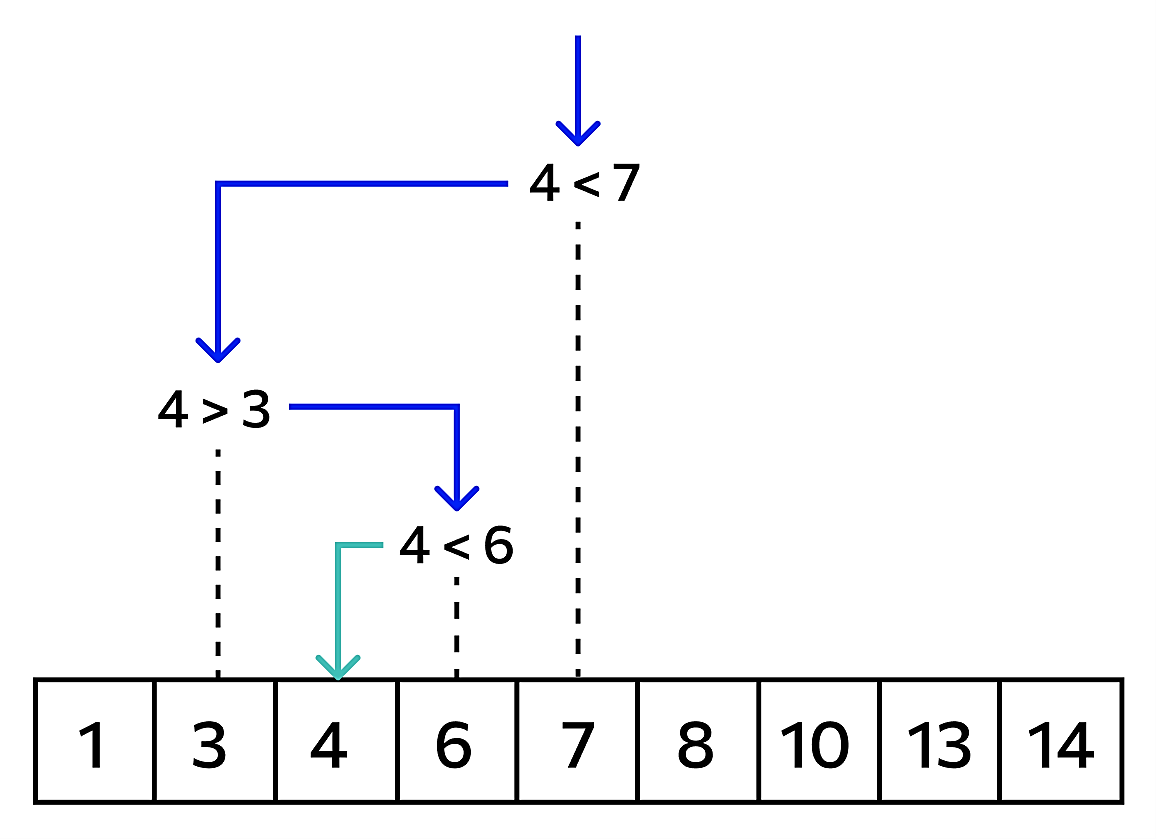


Рисунок 7. Пример работы бинарного поиска

*Реализация бинарного поиска на С++:*

int BinarySearch(int a[], int n, int x)

{

int i = 0, j = n - 1, m;

while (i < j)

{

m = (i + j) / 2;

if (x > a[m])

i = m + 1;

else j = m;

}

if (a[i] == x) return i;

else return -1;

}

**3.3 Интерполяционный поиск**

*Интерполяционный (интерполирующий) поиск* – это улучшенная и более быстрая версия бинарного поиска, который работает с отсортированными структурами данных и является более эффективным на равномерно распределенных данных. В отличие от двоичного поиска, он использует оценку для нахождения примерного местоположения искомого элемента. Также его отличием является то, что с его помощью можно находить и числовые значения, и текстовую информацию.

*Алгоритм:*

* Берем отсортированный массив данных;
* Оцениваем примерное местоположение искомого элемента по формуле (*m=i+(j-i)\*(x-a[i])/(a[j]-a[i])*, где i и j – индексы начального и конечного элементов, a[i] и a[j] – значения этих элементов, а х – искомое значение);
* Сравниваем искомое значение с оцененным элементов:
  + Если искомое значение равно оцененному значению, то возвращается его индекс (или значение элемента);
  + Если искомое значение не совпадает с оцененным:
    - Если искомое значение меньше чем оцененное, то поиск проводится в левой части массива;
    - Если искомое значение больше чем оцененное, то поиск проводится в правой части массива;
* Повторяем те же самые действия для новой части массива;
* Поиск продолжается до тех пор, пока не будет найден искомый элемент или пока не достигнет конца структуры.



Рисунок 8. Пример работы бинарного и интерполирующего поиска

*Реализация интерполяционного поиска на С++:*

int InterpolSearch(int a[], int n, int x)

{

int i = 0, j = n - 1, m;

while (i < j)

{

if (a[i] == a[j])

if (a[i] == x) return i;

else

return -1;

m = i + (j - i) \* (x - a[i]) / (a[j] - a[i]);

if (a[m] == x) return m;

else

if (x > a[m]) i = m + 1;

else

j = m - 1;

}

return -1;

}

4. ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ

В рамках данного курсового проекта были разработаны и реализованы пользовательские функции, направленные на управление и обработку информации о товарах на складе. Эти функции охватывают различные аспекты работы с данными, включая создание, просмотр и добавление информации в файлы, а также редактирование и удаление существующих записей о товарах.

Дополнительно к функциям работы с файлами, в проекте также реализованы алгоритмы сортировки и поиска, изученные в предыдущих разделах. Эти алгоритмы включают в себя метод сортировки прямого выбора, быструю сортировку, линейный и бинарный поиск.

Кроме того, была реализована универсальная функция выбора действия пользователем, которая обеспечивает гибкость в управлении программой. Пользователь может выбрать необходимое действие из предложенного списка, что делает работу с программой более удобной и понятной.

**4.1 Основные функции**

Основные функции выполняют ключевые задачи и операции в программе, обеспечивая её основную функциональность. Обычно они являются точками входа для выполнения задач и могут вызывать вспомогательные функции для выполнения отдельных частей своих задач.

**4.1.1 Функция универсального пользовательского выбора ChooseFunction()**

Функция **ChooseFunction()** – это пользовательская функция, которая предназначена для управления основным функционалом программы, позволяя пользователю выбирать различные операции, которые необходимо выполнить со списком товаров. На вход принимает строку у **filename,** которая содержит имя файла, над которым выполняются операции.

Пользователю предоставляется список операций, которые можно выполнить со списком товаров, каждому из которых соответствует свой уникальный номер. После отображения меню пользователь должен выбрать номер операции, который вводится с клавиатуры и поступает в переменную **main\_choose**. Далее программа использует конструкцию **switch-case** для определения, какая операция была выбрана пользователем. Она проверяет значение переменной **main\_choose** и в зависимости от него выполняет соответствующую функцию или предлагает пользователю выполнить еще один выбор. Функция работает до тех пор, пока пользователь не нажмет 0 (выход из программы). Также, если в **switch-case** поступает значение переменной **main\_choose,** для которой не предусмотрено выполнение никакой из функций, то срабатывает «ветка» **default,** которая выводит сообщение об ошибке выбора и пользователю предлагается сделать новый выбор.

Например, если пользователь нажал 1 (работа с файлом), то ему снова предлагается выбрать, что необходимо сделать с файлом (создание/ добавление/ просмотр/ выход). Этот выбор записывается в переменную **file\_choose,** после чего начинается выполнение соответствующей ветки **switch-case,** который на вход принимает эту переменную. Т.е вызывается определенная функция или, если пользователь нажал 0, он возвращается в главное меню.

**4.1.2 Функция создания файла CreateNewFile()**

Эта функция принимает строку **filename** в качестве аргумента и создает файл с указанным именем. Для этого она использует объект **ofstream**, который предоставляет возможность записи данных в файл. Сначала функция пытается открыть файл с указанным именем для записи. Если файл не удается открыть, она выводит сообщение об ошибке и завершает выполнение функции без создания файла. В случае успешного открытия файла для записи, выводится сообщение об успешном создании файла в стандартный вывод. После этого файл закрывается вызовом метода **close()** объекта **ofstream**.

**4.1.3 Функция просмотра содержимого файла ViewFile()**

Эта функция принимает строку **filename** в качестве аргумента и выводит содержимое файла в консоль. Сначала она пытается открыть файл для чтения с помощью объекта **ifstream**. Если файл не удается открыть, на экран выводится сообщение об ошибке и выполнение функции завершается. Если файл удается открыть, с помощью цикла **while** и метода **getline()** по отдельностисчитывается каждая строка в файле и записывается в переменную **line**, после чего ее содержимое выводится в консоль. Цикл завершает свое выполнение, когда достигнет конца файла. После этого, файл закрывается вызовом метода **close()** объекта **file**.

**4.1.4 Функция для добавления информации о товаре AddInfo()**

Функция **AddInfo() –** это пользовательская функция, которая добавляет информацию о товарах. В качестве аргумента принимает строку **filename**, содержащую имя файла, в который будут записаны новые элементы. Файл открывается в помощью объекта **ofstream** с опцией **os\_base::app**, которая позволяет записывать новые данные в конец существующего файла. Если файл не удается открыть, в консоль выводится сообщение об ошибке и завершается выполнение функции.

Если файл успешно открыт, у пользователя запрашивают количество товаров, которые необходимо добавить, и записывают введенное число в переменную **tov**. Далее создается динамический массив структур **spisok** типа **product,** в котором будет храниться информация обо всех введенных товарах. Затем начинается выполнение цикла **for**, который поочередно запрашивает у пользователя информацию о каждом товаре (наименование, цена, количество и дата поступления), которая записывается в массив **spisok**, после чего данные из массива записываются в файл. Цикл завершается, когда будет введена информация обо всех товарах общим количеством **tov.** После записи всех товаров файл закрывается вызовом метода **close()** объекта **file**. Функция завершается удалением памяти, занятой динамическим массивом **spisok**.

**4.1.5 Функция ввода даты InputDate()**

Функция **InputDate(**) предназначена для ввода даты пользователем. Она запрашивает у пользователя год, месяц и день, проверяет корректность введенных значений и возвращает структуру **date**, содержащую введенную дату. Сначала в цикле **do-while** функция запрашивает у пользователя год получения товара; цикл работает до тех пор, пока не будет введен год в диапазоне от 1900 до 2024. Затем аналогично запрашивается ввод месяца (диапазон от 1 до 12). Далее на основе введенных года и месяца определяется максимальное количество дней в этом месяце, и запрашивается ввод дня, который должен находиться в пределе от 1 до этого максимального значения. После этого функция рассчитывает общее количество дней, прошедший с 1 января 1900 года, и сохраняет это значение в поле **all\_in\_days** структуры **date**. Функция завершает свое выполнение, возвращая заполненную структуру типа **date**.

**4.1.6 Функция линейного поиска LinearSearch()**

Функция **LinearSearch()** функция, которая реализует простой линейный поиск по ключу наименование. Она принимает в качестве аргумента строку **filename**, содержащую имя файла. Сначала у пользователя запрашивают наименование товара, который необходимо найти, и оно записывается в переменную **find\_name**. Далее с помощью функции **GetFileIntoStruct()** данные из файла преобразовываются в динамический массив структур **SKLAD**. Затем с помощью цикла **for** мы обходим все товары и проверяем на наличие необходимого. Если товар (или товары, если их несколько) с заданным наименованием были найдены, информация о них выводится в консоль и флаг **found\_n** (сигнал о том, что товары найдены) становится **true**. Цикл завершается, когда он обошел все товары. Если после завершения цикла флаг **found\_n** остался **false**, выводится сообщение об отсутствии товаров с таким наименованием. Функция завершается удалением памяти, занятой массивом структур **SKLAD**.

**4.1.7 Функции бинарного поиска**

**4.1.7.1 Функция BinarySearchShow()**

Функция **BinarySearchShow()** предназначена для «подготовки» массива структур к выполнению функции бинарного поиска и отображению его результата. Она принимает в качестве аргумента строку **filename**, содержащую имя файла. Далее с помощью функции **GetFileIntoStruct()** данные из файла преобразовываются в динамический массив структур **SKLAD**. Далее полученный массив сортируется с помощью функции **QuickSort()** (\*т.к. бинарный поиск работает только с отсортированными массивами). Затем пользователя просят ввести наименование товара, который необходимо найти (оно записывается в переменную **key**) и создается целочисленный массив **same\_name\_idxs**, который будет хранить индексы элементов, наименование которых соответствует ключу. Затем вызывается функция **BinarySearch().** После завершения выполнения функции **BinarySearch(),** с помощью цикла **for** выводится информация обо всех товарах, имя которых соответствует ключу. Если же количество таких товаров **count** равняется нулю, на экран выводится сообщение о том, что товаров с таким именем на складе нет. Функция завершается очищением памяти, занятой массивом **SKLAD**.

**4.1.7.2 Функция BinarySearch()**

Функция **BinarySearch()** реализует алгоритм бинарного поиска в упорядоченном массиве **SKLAD**по их наименованию (поле **prod\_name**). В качестве аргументов в нее поступаютмассив структур **SKLAD,** «левые» и «правые» границы массива (**left** и **right**), ключ поиска ***key***, массив индексов **same\_name\_idxs** и переменная **count** для подсчета общего количества товаров с одним именем.

Алгоритм функции:

* Запускаем цикл **while**, который будет работать до тех пор, пока левая и правая границы массива не «схлопнутся»; это будет означать, что мы обошли весь массив;
* Начинаем с определения среднего индекса **mid** между **left** и **right**;
* Сравниваем ключ **key** с названием «серединного» товара **SKLAD[mid].prod\_name***:*
  + Если **key** меньше названия товара, сдвигаем правую границу (**right**) влево;
  + Если **key** больше названия товара, сдвигаем левую границу (**left**) вправо;
  + Если **key** совпадает с названием товара, находим все товары с этим названием, просматривая их индексы в обе стороны от найденного среднего индекса **mid**;
* Записываем найденные индексы товаров с одинаковыми названиями в массив **same\_name\_idxs[]** и увеличиваем счетчик **count**;
* Возвращаемся к шагу 1 до тех пор, пока **left** не станет больше **right** (условие завершение выполнения внешнего цикла **while**).

**4.1.8 Функции сортировки методом прямого выбора**

**4.1.8.1 Функция DirectSelectionSortShow()**

Функция **DirectSelectionSortShow()** используется для сортировки товаров по их наименованиям с помощью метода прямого выбора и отображения результата сортировки. На вход принимает строку *filename*, содержащую имя файла. Затем с помощью функции **GetFileIntoStruct**() считывает содержимое файла и записывает в динамический массив структур **SKLAD**. Далее сортирует склад, используя функцию **DirectSelectionSort()**, и записывает результат сортировки в новый файл *"DirectSelectionSortResult.txt",* используя функцию **StructIntoFileAndConsole().** Функция завершается удалением памяти, занятой массивом **SKLAD**.

**4.1.8.2 Функция** **DirectSelectionSort()**

Эта функция реализует алгоритм сортировки прямого выбора для массива товаров по их наименованиям.

Алгоритм функции:

* Внешний цикл **for** перебирает каждый элемент массива **SKLAD[]** от начала до конца;
* Внутренний цикл **for** ищет минимальный элемент в оставшейся части массива, начиная с элемента **i+1**;
* Если найденный минимальный элемент имеет наименование, «меньшее», чем у текущего минимального элемента **SKLAD[min].prod\_name**, то он становится новым минимальным элементом;
* После завершения внутреннего цикла, если найденный минимальный элемент отличается от текущего элемента **i**, они меняются местами;
* Таким образом, на каждом шаге внешнего цикла наименьший элемент массива сдвигается в начало, а **i** увеличивается на 1;
* Процесс повторяется до тех пор, пока не будет достигнут конец массива.

**4.1.9 Функции быстрой сортировки**

**4.1.9.1 Функция QuickSortShow()**

В качестве аргумента в функцию поступает строка **filename**, содержащая имя файла. Затем с помощью функции **GetFileIntoStruct**() считывает содержимое файла и записывает в динамический массив структур **SKLAD**. Далее начинается выполнение основной функции **QuickSort(),** которая сортирует массив структур. После чего результат сортировки записывается в новый файл *"QuickSortResult.txt"* и выводится в консоль с помощью функции **StructIntoFileAndConsole().** Функция завершается удалением памяти, занятой динамическим массивом структур **SKLAD.**

**4.1.9.2 Функция** **QuickSort()**

Это основная функция, реализующая алгоритм быстрой сортировки. Она принимает на вход левую и правую границы массива структур **SKLAD** и сам массив**.**

Алгоритм:

* Внутри функции выбирается опорный элемент **x**, который находится в середине массива;
* Затем во внешнем цикле **while** происходит перестановка элементов в массиве таким образом, чтобы элементы, меньшие x, располагались слева от него, а элементы, большие **x**, — справа от него. Он работает до тех пор, пока границы массива не «схлопнутся»;
* После разделения массива на две части вызывается рекурсивно функция **QuickSort**() для обеих частей;
* Рекурсивные вызовы продолжаются до тех пор, пока не будет обработан весь массив (пока левая граница не станет правой и наоборот).

**4.1.10 Функция отбора товаров SeeResult()**

Функция **SeeResult()** реализует отбор товаров, т.е. выводит в алфавитном порядке лишь те товары, соответствующие определенному условию (хранятся больше месяца и цена более 100000 р.). В качестве аргумента в функцию поступает строка **filename**, содержащая имя файла. Затем с помощью функции **GetFileIntoStruct**() считывает содержимое файла и записывает в динамический массив структур **SKLAD**. Далее у пользователя запрашивается контрольная дата, относительно которой будет проведена проверка товаров; она будет записана в переменную **curr\_prod** с помощью функции **InputDate().** Затем запускается цикл **for**, который подсчитывает количество товаров, подходящих под условие; это количество хранится в переменной **ot\_count**. После чего создается динамический массив структур **OTBOR** размером **ot\_count,** в который будет записан результат отбора товаров. Далее вновь запускается цикл **for**, который записывает все товары, подходящие по условию в массив **OTBOR**; он действует до тех пор, пока не пройдет все товары. С помощью функции **QuickSort**() отобранные товары сортируются в алфавитном порядке, после чего с помощью функции **StructIntoFileAndConsole()** отобранные товары выводятся на экран и записываются в файл с результатом отбора товаров *"Result.txt".* Функция завершается удалением памяти, занятой динамическими массивами структур **OTBOR** и **SKLAD**.

**4.1.11 Функция редактирования информации о товаре Editor()**

Функция **Editor**() предназначена для редактирования информации о товаре в файле. Она принимает в качестве аргумента имя файла (**filename**), содержащего информацию о товарах.

Сначала пользователю предлагается ввести наименование товара, который нужно отредактировать. Затем с помощью функции **GetFileIntoStruct()** из файла считывается информация о всех товарах и записывается в динамический массив структур **SKLAD**. После этого осуществляется поиск товаров с заданным наименованием. Если такие товары найдены, их индексы записываются в массив **idxs**, а флаг **found\_n** становится равным **true**. Если товары не найдены, выводится сообщение об отсутствии товаров с указанным наименованием, и выполнение функции завершается.

Если товары были найдены, то на экран выводится информация о них, а пользователю предлагается выбрать номер товара для редактирования. Если найден только один товар, его индекс автоматически присваивается переменной **chosen\_index.**

Затем, запускается цикл **while**, пользователю предлагается выбрать поле товара для редактирования: наименование, количество единиц, цену или дату получения. В зависимости от выбора пользователя, производится соответствующее редактирование поля товара. После каждого редактирования у пользователя спрашивают, желает ли он продолжить редактирование данного товара. В случае нажатия 1 редактирование продолжается, иначе цикл **while** завершает работу и сделанные изменения сохраняются в файл и выводятся в консоль с помощью функции **StructIntoFileAndConsole().** В конце функции память, выделенная под массив структур **SKLAD**, освобождается с помощью оператора **delete[].**

**4.1.12 Функция удаления информации о товаре DeleteProduct()**

Функция **DeleteProduct()** предназначена для удаления информации о конкретном товаре. Она принимает в качестве аргумента имя файла (**filename**), содержащего информацию о товарах.

Сначала пользователю предлагается ввести наименование товара, который нужно отредактировать. Затем с помощью функции **GetFileIntoStruct()** из файла считывается информация о всех товарах и записывается в динамический массив структур **SKLAD**. После этого осуществляется поиск товаров с заданным наименованием. Если такие товары найдены, их индексы записываются в массив **idxs**, а флаг **found\_n** становится равным **true**. Если товары не найдены, выводится сообщение об отсутствии товаров с указанным наименованием, и выполнение функции завершается.

Если товары были найдены, то на экран выводится информация о них, а пользователю предлагается выбрать номер товара для удаления. Если найден только один товар, тогда с помощью цикла **for** все товары, стоящие после удаляемого сдвигаются на шаг назад, тем стирая нужный товар. После чего с помощью функции **StructIntoFileAndConsole()** полученный массив выводится в консоль и записывается в файл и на экран выводится сообщение об успешном удалении товара; выполнение функции завершается.

Если таких товаров было несколько, пользователю предлагается выбрать, какой именно необходимо удалить; выбор записывается в переменную **choice** (она хранит индекс выбранного товара в массиве структура). Далее все товары, стоящие после удаляемого, смещаются на шаг назад и отредактированный массив выводится на экран и записывается в файл. В конце функции память, выделенная под массив структур **SKLAD**, освобождается с помощью оператора **delete[].**

**4.2 Вспомогательные функции**

Вспомогательные функции поддерживают основные функции, выполняя более мелкие, часто повторяющиеся задачи. Они используются для упрощения кода основных функций, улучшая его читаемость и снижая дублирование кода.

**4.2.1 Функция для считывания данных из файла GetFileIntoStruct()**

Функция **GetFileIntoStruct()** предназначена для преобразования данных, находящихся в файле в массив структур для дальнейшей работы с ними. Она считывает данные из файла (имя файла поступает в функцию в качестве аргумента) и записывает их в массив структур. Она открывает файл с помощью **ifstream** и, если файл не удается открыть, выводит сообщение об ошибке и возвращает **nullptr**. Далее функция подсчитывает количество товаров в файле, проверяя строки на наличие фразы *"Наименование: ",* и записывает общее количество товаров в переменную **products\_amount**. Если товаров нет, выводится сообщение о пустом складе и возвращается nullptr.

Если товары найдены, создается динамический массив структур **SKLAD** типа **product**. Затем с помощью методов **clear()** и **seekg(0, ios::beg)** файл перематывается в начало для повторного чтения данных. Далее запускается цикл **while**, который построчно считывает данные; он действует, пока количество считанных товаров не достигнет общего количества **products\_amount**. Метод **find()** используется для нахождения и записи информации в соответствующие поля структуры. После того, как было найдено соответствующее поле, полученная информация о товаре преобразовывается в необходимый тип данных и записывается в соответствующее поле в массиве структур.

После преобразования данных о товарах в массив структур, файл закрывается, и возвращается указатель на массив структур **SKLAD**.

**4.2.2 Функция для преобразования данных из массива структур в файл StructIntoFileAndConsole()**

Функция **StructIntoFileAndConsole()** предназначена для вывода данных из структуры, с которыми ранее производились некоторые операции, в консоль и файл. Она принимает в качестве аргументов строку **filename**, содержащую имя файла, целочисленную переменную **products\_amount**, равную количеству товаров в массиве, и массив структур **SKLAD[],** который будет преобразован.

Сначала функция пересоздает файл с помощью функции **CreateNewFile()**, очищая данные, которые ранее в нем содержались. Далее открывает файл для записи с помощью объекта **ofstream**, и в случае неудавшегося открытия выбрасывает сообщение ошибке и завершает выполнение функции. Если файл был успешно открыт и количество товаров **products\_amount** равняется нулю, то в консоль и файл с помощью оператора **<<**  записывается сообщение об отсутствии товаров. Иначе с помощью цикла **for** обходим все товары, записывая всю информацию о них. При этом для более удобной различимости информации используются разделительные линии **"---"**. После того, как информация обо всех товарах была выведена в консоль и записана в файл, цикл прекращается и функция завершается закрытием файла с помощью метода **close().**

**4.2.3 Функция для определения високосного года IsLeapYear()**

Функция **IsLeapYear()** предназначена для определения, является ли введенный год високосным (это необходимо для преобразования даты получения товара в дни). Она принимает в качестве параметра целочисленную переменную **year** (год, который нужно проверить на високосность). Функция возвращает значение **true (1),** если год является високосным, и **false (0)** в противном случае.

Високосный год определяется по следующим правилам:

* Год делится на 4;
* Год не делится на 100, за исключением случаев, когда делится на 400.

**4.2.4 Функция для определения максимального количества дней в месяце MaxDays()**

Функция **MaxDays()** вычисляет количество дней в указанном месяце для заданного года. Она принимает два параметра: **year** и **month**, и возвращает целое число, представляющее максимальное количество дней в введенном месяце.

Функция использует конструкцию **switch** для определения количества дней в месяце:

* Если на вход поступил февраль (месяц 2), то проверяется, является ли год високосным, в зависимости от этого возвращается значение 28 или 29;
* Для месяцев 4, 6, 9 и 11 функция возвращает 30 дней;
* Для всех остальных возвращает 31 день.

**4.2.5 Функция нахождения первого нецифрового элемента строки FirstNotDigit()**

Функция **FirstNotDigit()** используется для нахождения первого символа в строке, который не является цифрой. Она проходит по каждому символу строки от начала до конца и проверяет, является ли он цифрой (от 1 до 9). Если найден символ, который не является цифрой, то функция возвращает его индекс. В случае, если в строке нет таких символов, функция возвращает значение -1.

5. ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

Данная программа предназначена для управления информацией о товарах на складе. Она предоставляет пользователю широкий спектр функциональных возможностей, включая создание нового файла для хранения данных, добавление информации о новых товарах, поиск товаров, сортировку данных и редактирование информации о существующих товарах.

**Основные возможности программы:**

* Создание нового файла.
* Добавление информации о товарах.
* Просмотр содержимого файла.
* Поиск товара (линейный и бинарный).
* Сортировка товаров (прямая выборка и быстрая сортировка).
* Редактирование информации о товарах.
* Удаление товара.
* Отбор товаров по заданным критериям (хранится более 30 дней и стоимость более 100000, в алфавитном порядке).
* Выход (завершение работы программы).

**Структуры данных, используемые в программе :**

* Структура **date –** структура, содержащая поля даты:
* Год (целочисленное значение prod\_name);
* Месяц (целочисленное значение prod\_count);
* День (целочисленное значение price);
* Дата поступления полностью в днях (целочисленное all\_in\_days).

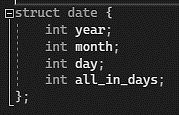
****

Рисунок 9. Фрагмент кода, содержащий реализацию структуры **date**

* Структура **product –** структура, содержащая поля с информацией о товарах:
* Наименование (строковое значение prod\_name);
* Количество единиц товара (целочисленное значение prod\_count);
* Единичная стоимость (целочисленное значение price);
* Дата поступления (get\_prod типа структуры **date**).

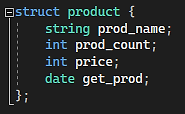


Рисунок 10. Фрагмент кода, содержащий реализацию структуры **product**

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы была разработана программа на языке C++, предназначенная для управления информацией о товарах на складе. Программа реализует функционал создания файла с данными о товарах и работы с ним, а также выполнения различных операций по их поиску, сортировке, редактированию и удалению.

Таким образом, данная программа позволяет эффективно управлять данными о товарах на складе, что делает ее мощным инструменталом для организации информации и обеспечивает пользователю возможность легко выполнять свои задачи и достигать поставленных целей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лафоре Р. Объектно-ориентированное программирование в С++. Классика Computer Science. 4 изд. — СПб.: Питер, 2018. — 928 с.
2. Информационные технологии в управлении. Учебное пособие. – М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2008. – 336с.
3. Stack Overflow [Электронный ресурс] // stackoverflow.com: [сайт]. - URL: <https://stackoverflow.com>.
4. Документация по С++ [Электронный ресурс] // cplusplus.com: [сайт]. - URL: <https://www.cplusplus.com>.
5. Алгоритмы сортировки и поиска, структуры данных [Электронный ресурс] // prog-cpp.ru: [сайт]. - URL: <https://prog-cpp.ru>.
6. НОУ ИНИТ. Структуры и алгоритмы компьютерной работы [Электронный ресурс] // intuit.ru: [сайт]. - URL: <https://intuit.ru/studies/courses/648/504/lecture/11466>
7. Г.В. Ваныкина, Т.О. Сундукова Структуры и алгоритмы компьютерной обработки данных [Текст] / Г.В. Ваныкина, Т.О. Сундукова — 1. — Новосибирск: ИНТИУИТ, 2011 — 219 c.
8. Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. Алгоритмы: построение и анализ / Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. — М.: МГТУ им. Н.Э. Баумана, 2016.

ПРИЛОЖЕНИЕ А

*(листинг кода)*

#include <iostream>

#include <iomanip>

#include <fstream>

#include <algorithm>

#include <Windows.h>

#include <string>

using namespace std;

struct date {

int year;

int month;

int day;

int all\_in\_days;

};

struct product {

string prod\_name;

int prod\_count;

int price;

date get\_prod;

};

void CreateNewFile(string&);

bool IsLeapYear(int);

int MaxDays(int, int);

date InputDate();

void ViewFile(string&);

void AddInfo(string&);

void LinearSearch(string&);

void DirectSelectionSort(product[], int);

void DirectSelectionSortShow(string&);

int FirstNotDigit(string&);

product\* GetFileIntoStruct(string&, int&);

void StructIntoFileAndConsole(string&, int, product[]);

void QuickSort(int, int, product[]);

void QuickSortShow(string&);

void BinarySearch(product[], int , int , string , int [], int& );

void BinarySearchShow(string&);

void Editor(string&);

void DeleteProduct(string&);

void SeeResult(string&);

void ChooseFunction(string&);

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "ru");

string filename = "file.txt";

ChooseFunction(filename);

return 0;

}

void CreateNewFile(string& filename)

{

ofstream file(filename);

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

cout << "Ошибка создания файла " << filename << " !\n";

return;

}

cout << "Файл " << filename << " создан успешно!\n";

file.close();

}

bool IsLeapYear(int year)

{

return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

}

int MaxDays(int year, int month)

{

switch (month)

{

case 2:

return IsLeapYear(year) ? 29 : 28;

case 4: case 6: case 9: case 11:

return 30;

default:

return 31;

}

}

date InputDate()

{

date input\_date;

do

{

cout << "Год: ";

cin >> input\_date.year;

} while (input\_date.year < 1900 || input\_date.year>2024);

do

{

cout << "Месяц: ";

cin >> input\_date.month;

} while (input\_date.month <= 0 || input\_date.month > 12);

int max\_days = MaxDays(input\_date.year, input\_date.month);

do

{

cout << "День: ";

cin >> input\_date.day;

} while (input\_date.day<1 || input\_date.day>max\_days);

input\_date.all\_in\_days = input\_date.day + ((input\_date.month - 1) \* MaxDays(input\_date.year, input\_date.month)) + ((input\_date.year - 1899) \* (365 + IsLeapYear(input\_date.year)));

return input\_date;

}

void ViewFile(string& filename)

{

ifstream file(filename);

if (!file.is\_open())

{

cout << "Файл не найден!" << filename << endl;

return;

}

string line;

while (getline(file, line))

{

cout << line << endl;

}

file.close();

}

void AddInfo(string& filename)

{

ofstream file(filename, ios\_base::app);

if (!file.is\_open())

{

cout << "Файл не найден!\n";

return;

}

else

{

int tov;

cout << "\nВведите количество товаров, которые вы хотите добавить: ";

cin >> tov;

product\* spisok = new product[tov];

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

{

cout << " Товар № " << i + 1 << endl;

cout << " Введите название товара: ";

cin >> spisok[i].prod\_name;

cout << " Введите кол-во единиц товара: ";

cin >> spisok[i].prod\_count;

cout << " Введите единичную стоимость: ";

cin >> spisok[i].price;

cout << " Введите дату поступления товара: " << endl;

spisok[i].get\_prod = InputDate();

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

file << "\nНаименование: " << spisok[i].prod\_name << endl;

file << "Количество единиц: " << spisok[i].prod\_count << endl;

file << "Цена: " << spisok[i].price << endl;

file << "Дата получения: " << setw(2) << setfill('0') << spisok[i].get\_prod.day << "."

<< setw(2) << setfill('0') << spisok[i].get\_prod.month << "." << spisok[i].get\_prod.year << endl;

file << "\n----------------------------------------------------------------\n";

}

file.close();

delete[] spisok;

}

}

void LinearSearch(string& filename)

{

string find\_name;

cout << "\nВведите наименование товара: ";

cin >> find\_name;

int products\_amount = 0;

product\* SKLAD = GetFileIntoStruct(filename, products\_amount);

bool found\_n = false;

for (int i = 0; i < products\_amount; i++)

{

if (SKLAD[i].prod\_name == find\_name)

{

found\_n = true;

cout << "\nТовар " << find\_name << " имеется на складе!\n";

cout << "Наименование: " << SKLAD[i].prod\_name << endl;

cout << "Количество единиц: " << SKLAD[i].prod\_count << endl;

cout << "Цена: " << SKLAD[i].price << endl;

cout << "Дата получения: " << setw(2) << setfill('0') << SKLAD[i].get\_prod.day << "."

<< setw(2) << setfill('0') << SKLAD[i].get\_prod.month << "." << SKLAD[i].get\_prod.year << endl;

}

}

if (!found\_n)

cout << "На складе нет товара с таким наименованием!\n";

cout << endl;

delete[] SKLAD;

}

void DirectSelectionSort(product SKLAD[], int products\_amount)

{

int min;

product t;

for (int i = 0; i < products\_amount; i++)

{

min = i;

for (int j = i + 1; j < products\_amount; j++)

{

if (SKLAD[j].prod\_name < SKLAD[min].prod\_name)

min = j;

}

if (min == i)

continue;

t = SKLAD[i];

SKLAD[i] = SKLAD[min];

SKLAD[min] = t;

}

}

void DirectSelectionSortShow(string& filename)

{

int products\_amount = 0;

product\* SKLAD = GetFileIntoStruct(filename, products\_amount);

DirectSelectionSort(SKLAD, products\_amount);

string q\_result = "DirectSelectionSortResult.txt";

StructIntoFileAndConsole(q\_result, products\_amount, SKLAD);

cout << "\nРезультат записан в файл DirectSelectionSortResult.txt\n";

delete[] SKLAD;

}

int FirstNotDigit(string& str)

{

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

{

if (!(str[i] >= '0' && str[i] <= '9'))

return i;

}

return -1;

}

product\* GetFileIntoStruct(string& filename, int& products\_amount)

{

ifstream file(filename);

if (!file.is\_open())

{

cout << "\nОшибка открытия файла!\n";

return nullptr;

}

string str;

while (getline(file, str))

{

if (str.find("Наименование: ") != string::npos)

products\_amount++;

}

if (products\_amount == 0)

{

cout << "\nСклад пуст!\n";

return nullptr;

}

product\* SKLAD = new product[products\_amount];

file.clear();

file.seekg(0, ios::beg);

int curr\_product = 0;

while (getline(file, str) && curr\_product < products\_amount)

{

if (str.find("Наименование: ") != string::npos)

SKLAD[curr\_product].prod\_name = str.substr(14);

if (str.find("Количество единиц: ") != string::npos)

{

string count\_str = str.substr(19);

int count\_end = FirstNotDigit(count\_str) - 1;

if (count\_end != -1)

{

count\_str = count\_str.substr(0, count\_end);

SKLAD[curr\_product].prod\_count = stoi(count\_str);

}

}

if (str.find("Цена: ") != string::npos)

{

string count\_str = str.substr(6);

int price\_end = FirstNotDigit(count\_str) - 2;

if (price\_end != -1)

{

count\_str = count\_str.substr(0, price\_end);

SKLAD[curr\_product].price = stoi(count\_str);

}

}

if (str.find("Дата получения: ") != string::npos)

{

string count\_str = str.substr(16);

int day\_end = FirstNotDigit(count\_str);

if (day\_end != -1)

{

string day\_str = count\_str.substr(0, day\_end);

SKLAD[curr\_product].get\_prod.day = stoi(day\_str);

}

count\_str = count\_str.substr(day\_end + 1);

int month\_end = FirstNotDigit(count\_str);

if (month\_end != -1)

{

string month\_str = count\_str.substr(0, month\_end);

SKLAD[curr\_product].get\_prod.month = stoi(month\_str);

}

count\_str = count\_str.substr(month\_end + 1);

string year\_str = count\_str.substr(0, 4);

SKLAD[curr\_product].get\_prod.year = stoi(year\_str);

curr\_product++;

}

}

file.close();

return SKLAD;

}

void StructIntoFileAndConsole(string& filename, int products\_amount, product SKLAD[])

{

CreateNewFile(filename);

ofstream result(filename);

if (!result.is\_open())

{

cout << "\nОшибка открытия файла!\n";

return;

}

else

{

if (products\_amount == 0)

{

cout << "\nТоваров нет!\n";

result << "\nТоваров нет!\n";

}

else

{

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

result << "\n----------------------------------------------------------------\n";

for (int i = 0; i < products\_amount; i++)

{

cout << "\nНаименование: " << SKLAD[i].prod\_name << endl;

cout << "Количество единиц: " << SKLAD[i].prod\_count << endl;

cout << "Цена: " << SKLAD[i].price << endl;

cout << "Дата получения: " << setw(2) << setfill('0') << SKLAD[i].get\_prod.day << "."

<< setw(2) << setfill('0') << SKLAD[i].get\_prod.month << "." << SKLAD[i].get\_prod.year << endl;

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

result << "\nНаименование: " << SKLAD[i].prod\_name << endl;

result << "Количество единиц: " << SKLAD[i].prod\_count << endl;

result << "Цена: " << SKLAD[i].price << endl;

result << "Дата получения: " << setw(2) << setfill('0') << SKLAD[i].get\_prod.day << "."

<< setw(2) << setfill('0') << SKLAD[i].get\_prod.month << "." << SKLAD[i].get\_prod.year << endl;

result << "\n----------------------------------------------------------------\n";

}

}

}

result.close();

}

void QuickSort(int left, int right, product SKLAD[])

{

product t;

int i = left;

int j = right;

string x = SKLAD[(left + right) / 2].prod\_name;

while (i <= j)

{

while (SKLAD[i].prod\_name < x)

i++;

while (SKLAD[j].prod\_name > x)

j--;

if (i <= j)

{

t = SKLAD[i];

SKLAD[i] = SKLAD[j];

SKLAD[j] = t;

i++;

j--;

}

}

if (left < j)

QuickSort(left, j, SKLAD);

if (i < right)

QuickSort(i, right, SKLAD);

}

void QuickSortShow(string& filename)

{

int products\_amount = 0;

product\* SKLAD = GetFileIntoStruct(filename, products\_amount);

int left = 0;

int right = products\_amount - 1;

QuickSort(left, right, SKLAD);

string q\_result = "QuickSortResult.txt";

StructIntoFileAndConsole(q\_result, products\_amount, SKLAD);

cout << "\nРезультат записан в файл QuickSortResult.txt\n";

delete[] SKLAD;

}

void BinarySearch(product SKLAD[], int left, int right, string key, int same\_name\_idxs[], int& count)

{

int mid = 0;

while (left <= right)

{

mid = (left + right) / 2;

if (key < SKLAD[mid].prod\_name)

right = mid - 1;

else if (key > SKLAD[mid].prod\_name)

left = mid + 1;

else

{

same\_name\_idxs[count++] = mid;

int i = mid - 1;

while (i >= left && SKLAD[i].prod\_name == key)

{

same\_name\_idxs[count++] = i;

i--;

}

i = mid + 1;

while (i <= right && SKLAD[i].prod\_name == key)

{

same\_name\_idxs[count++] = i;

i++;

}

return;

}

}

}

void BinarySearchShow(string& filename)

{

int products\_amount = 0;

product\* SKLAD = GetFileIntoStruct(filename, products\_amount);

int left = 0;

int right = products\_amount - 1;

QuickSort(left, right, SKLAD);

string key;

cout << "\nВведите наименование товара: ";

cin >> key;

int same\_name\_idxs[123];

int count = 0;

BinarySearch(SKLAD, left, right, key, same\_name\_idxs, count);

if (count == 0)

{

cout << "На складе нет товара с таким наименованием!\n\n";

return;

}

else

{

cout << "\nНайдено " << count << " товаров с именем " << key << ":" << endl;

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

{

int idx = same\_name\_idxs[i];

cout << "\nПорядковый номер товара на складе: " << idx + 1 << endl;

cout << "Наименование: " << SKLAD[idx].prod\_name << endl;

cout << "Количество единиц: " << SKLAD[idx].prod\_count << endl;

cout << "Цена: " << SKLAD[idx].price << endl;

cout << "Дата получения: " << setw(2) << setfill('0') << SKLAD[idx].get\_prod.day << "." << setw(2) << setfill('0') << SKLAD[idx].get\_prod.month << "." << SKLAD[idx].get\_prod.year << endl;

}

}

cout << endl;

delete[] SKLAD;

}

void Editor(string& filename)

{

string find\_name;

cout << "\nВведите наименование товара : ";

cin >> find\_name;

int products\_amount = 0;

product\* SKLAD = GetFileIntoStruct(filename, products\_amount);

bool found\_n = false;

int idxs[123];

int count = 0;

for (int i = 0; i < products\_amount; i++)

{

if (SKLAD[i].prod\_name == find\_name)

{

found\_n = true;

idxs[count++] = i;

}

}

if (!found\_n)

{

cout << "\nНа складе нет товара с таким наименованием!" << endl;

return;

}

int chosen\_index;

if (count == 1)

{

cout << "\nНайден 1 товар с именем " << find\_name << endl;

chosen\_index = idxs[0];

}

else

{

cout << "\nНайдены следующие товары с именем '" << find\_name << "':" << endl;

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

{

cout << "\nТовар №" << i + 1 << ". \n"<< "\nНаименование: " << SKLAD[idxs[i]].prod\_name<< "\nКоличество единиц: " << SKLAD[idxs[i]].prod\_count<< "\nЦена: " << SKLAD[idxs[i]].price<< "\nДата получения: " << setw(2) << setfill('0') << SKLAD[idxs[i]].get\_prod.day << "." << setw(2) << setfill('0') << SKLAD[idxs[i]].get\_prod.month << "." << SKLAD[idxs[i]].get\_prod.year << endl;

cout << endl;

}

int choice;

cout << "\nВыберите номер товара, который хотите изменить: ";

cin >> choice;

choice--;

if (choice < 0 || choice >= count)

{

cout << "\nНеверный выбор товара." << endl;

return;

}

chosen\_index = idxs[choice];

}

bool continue\_edit = true;

while (continue\_edit)

{

int choose;

cout << "\nКакое поле вы хотите изменить?\n"<<

" 1-Наименование\n" << " 2-Количество единиц\n" << " 3-Цена\n" << " 4-Дата получения\n";

cin >> choose;

string new\_name;

int new\_count;

int new\_price;

switch (choose)

{

case 1:

cout << "Введите новое наименование: ";

cin >> new\_name;

SKLAD[chosen\_index].prod\_name = new\_name;

break;

case 2:

cout << "Введите новое количество единиц: ";

cin >> new\_count;

SKLAD[chosen\_index].prod\_count = new\_count;

break;

case 3:

cout << "Введите новую цену: ";

cin >> new\_price;

SKLAD[chosen\_index].price = new\_price;

break;

case 4:

cout << "Введите новую дату получения: ";

SKLAD[chosen\_index].get\_prod = InputDate();

break;

default:

cout << "Ошибка: Некорректный выбор." << endl;

return;

}

int continue\_edit;

cout << "Выберите 1, если хотите продолжить редактирование этого товара: ";

cin >> continue\_edit;

switch (continue\_edit)

{

case 1:

continue\_edit = false;

break;

default:

StructIntoFileAndConsole(filename, products\_amount, SKLAD);

return;

}

}

delete[] SKLAD;

}

void DeleteProduct(string& filename)

{

string find\_name;

cout << "\nВведите наименование товара, который необходимо удалить: ";

cin >> find\_name;

int products\_amount = 0;

product\* SKLAD = GetFileIntoStruct(filename, products\_amount);

bool found\_n = false;

int idxs[123];

int count = 0;

for (int i = 0; i < products\_amount; i++)

{

if (SKLAD[i].prod\_name == find\_name)

{

found\_n = true;

idxs[count++] = i;

}

}

if (!found\_n)

{

cout << "\nНа складе нет товара с таким наименованием!" << endl;

return;

}

if (count == 1)

{

int idx = idxs[0];

for (int i = idx; i < products\_amount - 1; i++)

{

SKLAD[i] = SKLAD[i + 1];

}

products\_amount--;

StructIntoFileAndConsole(filename, products\_amount, SKLAD);

cout << "\nТовар '" << find\_name << "' успешно удален!" << endl;

return;

}

cout << "\nНайдены следующие товары с именем '" << find\_name << "':" << endl;

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

{

cout << "Товар №" << i + 1 << ". \n"<< "\nНаименование: " << SKLAD[idxs[i]].prod\_name<< "\nКоличество единиц: " << SKLAD[idxs[i]].prod\_count<< "\nЦена: " << SKLAD[idxs[i]].price<< "\nДата получения: " << setw(2) << setfill('0') << SKLAD[idxs[i]].get\_prod.day << "." << setw(2) << setfill('0') << SKLAD[idxs[i]].get\_prod.month << "." << SKLAD[idxs[i]].get\_prod.year << endl;

cout << endl;

}

int choice;

cout << "\nВыберите номер товара, который хотите удалить: ";

cin >> choice;

choice--;

if (choice < 0 || choice >= count)

{

cout << "\nНеверный выбор товара." << endl;

return;

}

for (int i = idxs[choice]; i < products\_amount - 1; i++)

{

SKLAD[i] = SKLAD[i + 1];

}

products\_amount--;

cout << "\nРезультат удаления информации о товаре " << find\_name << endl;

StructIntoFileAndConsole(filename, products\_amount, SKLAD);

delete[] SKLAD;

}

void SeeResult(string& filename)

{

int products\_amount = 0;

product\* SKLAD = GetFileIntoStruct(filename, products\_amount);

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

cout << "\n Введите контрольную дату: " << endl;

date curr\_prod = InputDate();

cout << "Контрольная дата: " << setw(2) << setfill('0') << curr\_prod.day << "."<< setw(2) << setfill('0') << curr\_prod.month << "." << curr\_prod.year << endl;

cout << "----------------------------------------------------------------" << endl;

cout << "Результат: \n\n";

int ot\_count = 0;

for (int i = 0; i < products\_amount; i++)

{

if ((curr\_prod.all\_in\_days - SKLAD[i].get\_prod.all\_in\_days) > 30 && SKLAD[i].price > 100000)

ot\_count++;

}

product\* OTBOR = new product[ot\_count];

int ot\_idx = 0;

for (int i = 0; i < products\_amount; i++)

{

if ((curr\_prod.all\_in\_days - SKLAD[i].get\_prod.all\_in\_days) > 30 && SKLAD[i].price > 100000)

{

OTBOR[ot\_idx] = SKLAD[i];

ot\_idx++;

}

}

int left = 0;

int right = ot\_count - 1;

QuickSort(left, right, OTBOR);

string result = "Result.txt";

StructIntoFileAndConsole(result, ot\_idx, OTBOR);

cout << "\nРезультат записан в файл Result.txt\n";

delete[] OTBOR;

delete[] SKLAD;

}

void ChooseFunction(string& filename)

{

while (true)

{

system("cls");

cout << "\nВыберите, что вы хотите сделать:\n";

int main\_choose;

cout << " 1 - Работа с файлом\n" << " 2 - Поиск в файле\n"<< " 3 - Сортировка\n" << " 4 - Вывести результат отбора товаров\n"<< " 5 - Редактировать информацию о товаре\n" << " 6 - Удалить товар\n" << " 0 - Выход\n";

cin >> main\_choose;

switch (main\_choose)

{

case 1:

{

int file\_choose;

do

{

cout << "\nВыберите, что вы хотите сделать с файлом:\n";

cout << " 1 - Создать файл\n" << " 2 - Добавить в файл новую запись\n"<< " 3 - Просмотреть содержимое файла\n" << " 0 - Вернуться в главное меню\n";

cin >> file\_choose;

switch (file\_choose)

{

case 1:

system("cls");

cout << " СОЗДАНИЕ ФАЙЛА\n";

CreateNewFile(filename);

break;

case 2:

system("cls");

cout << " ДОБАВЛЕНИЕ ИНФОРМАЦИИ\n";

AddInfo(filename);

break;

case 3:

system("cls");

cout << " ПРОСМОТР СОДЕРЖИМОГО ФАЙЛА\n";

ViewFile(filename);

break;

case 0:

break;

default:

cout << "Ошибка выбора!\n";

break;

}

} while (file\_choose != 0);

break;

}

case 2:

{

int search\_choose;

do

{

cout << "\nВыберите метод поиска в файле:\n";

cout << " 1 - Линейный поиск\n" << " 2 - Бинарный поиск\n" << " 0 - Вернуться в главное меню\n";

cin >> search\_choose;

switch (search\_choose)

{

case 1:

system("cls");

cout << " ЛИНЕЙНЫЙ ПОИСК\n";

LinearSearch(filename);

break;

case 2:

system("cls");

cout << " БИНАРНЫЙ ПОИСК\n";

BinarySearchShow(filename);

break;

case 0:

break;

default:

cout << "Ошибка выбора!\n";

break;

}

} while (search\_choose != 0);

break;

}

case 3:

{

int sort\_choose;

do

{

cout << "\nВыберите метод сортировки: \n" << endl;

cout << " 1 - Быстрая сортировка (метод QuickSort)\n" << " 2 - Метод сортировки прямого выбора\n" << " 0 - Вернуться в главное меню\n";

cin >> sort\_choose;

switch (sort\_choose)

{

case 1:

system("cls");

cout << " РЕЗУЛЬТАТ БЫСТРОЙ СОРТИРОВКИ\n";

QuickSortShow(filename);

cout << "Нажмите любую клавишу, чтобы продолжить...\n";

system("pause > nul");

break;

case 2:

system("cls");

cout << " РЕЗУЛЬТАТ СОРТИРОВКИ МЕТОДОМ ПРЯМОГО ВЫБОРА\n";

DirectSelectionSortShow(filename);

cout << "Нажмите любую клавишу, чтобы продолжить...\n";

system("pause > nul");

break;

case 0:

break;

default:

cout << "Ошибка выбора!\n";

break;

}

} while (sort\_choose != 0);

break;

}

case 4:

system("cls");

cout << " РЕЗУЛЬТАТ ОТБОРА ТОВАРОВ\n";

SeeResult(filename);

cout << "Нажмите любую клавишу, чтобы продолжить...\n";

system("pause > nul");

break;

case 5:

system("cls");

cout << " РЕДАКТИРОВАНИЕ ТОВАРОВ\n";

Editor(filename);

cout << "Нажмите любую клавишу, чтобы продолжить...\n";

system("pause > nul");

break;

case 6:

system("cls");

cout << " УДАЛЕНИЕ ТОВАРА\n";

DeleteProduct(filename);

cout << "Нажмите любую клавишу, чтобы продолжить...\n";

system("pause > nul");

break;

case 0:

return;

default:

cout << "Ошибка выбора!\n"; break;

}

}

}