

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

Дисциплина: Основы алгоритмизации и программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

«Система учета клиентов и продаж для магазина»

Выполнил:

гр. 421701 Федосов Е.К.

Проверила:

Панасик А.А.

Минск 2025

РЕФЕРАТ

СИСТЕМА УЧЁТА КЛИЕНТОВ И ПРОДАЖ ДЛЯ МАГАЗИНА :
курсовая работа / Е. К. Федосов. – Минск : БГУИР, 2025. – п.з. – 62 с.,
рисунков – 15, источников – 7, приложений – 2.

Цель курсовой работы: разработка программного обеспечения, предназначенного для автоматизации процессов хранения, обработки и анализа информации о клиентах и продажах магазина.

В ходе выполнения работы были определены и реализованы основные функциональные требования к системе учёта: ведение базы данных клиентов и их покупок, поддержка поиска и сортировки информации по различным критериям, формирование статистики по продажам. Программа позволяет выполнять линейный поиск по ФИО клиента и бинарный поиск по наименованию товара, реализована быстрая сортировка по товарам, сортировка выбором по дате покупки и сортировка вставками по алфавиту клиентов. Дополнительно реализован функционал поиска клиентов по товарам и дате, а также вычисление статистики: определение самого популярного товара и самого активного клиента по сумме покупок.

Разработано консольное приложение на языке программирования C++, использующее бинарные файлы для хранения данных и модульный подход в архитектуре. Проведено тестирование основных функций, подтвердившее корректность и устойчивость работы системы.

Область применения: разработанное программное обеспечение может использоваться в небольших торговых предприятиях для ведения клиентской и товарной базы, а также в учебных целях при изучении алгоритмов сортировки, поиска и базовых принципов построения информационных систем.

СОДЕРЖАНИЕ

Введение.....	4
1 Структуры и файлы.....	5
1.1 Структура программы.....	5
1.2 Используемые структуры данных.....	6
1.3 Хранение данных.....	6
2 Алгоритмы сортировки.....	7
2.1 Быстрая сортировка по названию товара.....	7
2.2 Сортировка выбором по дате покупки.....	8
2.3 Сортировка вставками по ФИО клиента.....	8
3 Алгоритмы поиска.....	10
3.1 Линейный поиск клиентов по ФИО.....	10
3.2 Линейный поиск клиентов по товару.....	10
3.3 Бинарный поиск клиентов по товару.....	11
3.4 Поиск клиентов по товару и дате.....	12
4 Пользовательские функции.....	13
4.1 Функции для работы со строками и датами.....	13
4.2 Функции валидации ввода и работы с файлом.....	14
4.4 Функции управления бинарными файлами.....	16
4.5 Функции генерация отчёта.....	16
4.6 Функции работы с клиентами.....	17
5 Описание работы программы.....	20
5.1 Краткое описание программы.....	20
5.2 Описание функциональных возможностей программы.....	20
Заключение.....	27
Список использованных источников.....	28
Приложение А (обязательное) Листинг кода.....	29
Приложение Б (обязательное) Блок-схема работы программы.....	61
Ведомость документов.....	62

ВВЕДЕНИЕ

В современных условиях автоматизация учета и анализа данных играет значимую роль в эффективности управления торговыми процессами. Особенно актуальной становится разработка программных решений, ориентированных на упрощение ведения клиентской базы и учета продаж в магазинах. Это способствует повышению качества обслуживания, оптимизации внутренних процессов и принятию обоснованных решений.

Данная курсовая работа написана с использованием языка программирования C++. Язык C++ — это ключ к современному объектно-ориентированному программированию. Он создан для разработки высокопроизводительного программного обеспечения и чрезвычайно популярен среди программистов [1].

В данном документе представлено описание курсовой работы «Система учета клиентов и продаж для магазина». Целью работы является создание системы, обеспечивающей учет информации о клиентах и совершенных ими покупках, а также реализация функций поиска, сортировки и анализа данных.

Для достижения поставленной цели нужно решить следующие задачи:

- реализация структуры хранения данных о клиентах и покупках;
- разработка алгоритмов линейного поиска по ФИО и бинарного поиска по наименованию товара;
- реализация функции генерации случайных данных клиентов;
- реализация базовых операций с клиентами, а именно: добавление, редактирование, удаление клиентов, подсчёт общего количества клиентов;
- реализация функции просмотра всех записей клиентов;
- внедрение алгоритмов сортировки: быстрой сортировки по товару, сортировки выбором по дате покупки и сортировки вставками по ФИО клиента;
- реализация алгоритма поиска клиентов, купивших определенный товар после заданной даты, с выводом результатов в алфавитном порядке;
- разработка алгоритмов линейного поиска по ФИО и бинарного поиска по наименованию товара;
- определение наиболее популярного товара и самого активного клиента на основе суммы покупки;
- формирование итогового отчета с определением самого популярного товара по количеству покупок и самого активного клиента по общей сумме покупок.

Создание подобной системы особенно важно для магазинов, где ежедневно совершается множество операций. Автоматизированное решение позволяет быстро находить нужные данные, сортировать их, собирать статистику и формировать отчеты. Это экономит время и снижает количество ошибок.

Важно отметить, что реализация надежного, читаемого и поддерживаемого программного кода имеет ключевое значение для успешной разработки любого программного обеспечения[2].

1 СТРУКТУРЫ И ФАЙЛЫ

Курсовая работа имеет простую и понятную структуру, разделенную на два основных каталога и на несколько файлов. В работе используются две основные папки `include/` и `source/` для организации заголовочных файлов и исходного кода. Также имеется основной файл `main.cpp`, который находится в корне работы, где реализован весь функционал. Каждое название файла в структуре продумано так, чтобы отображать его назначение и содержание.

1.1 Структура программы

Файл `main.cpp` является основным модулем программы. Функция `main()` реализует точку входа — начальный элемент выполнения программы. Он предназначен для инициализации пользовательского интерфейса в виде текстового меню, а также для организации взаимодействия пользователя с функциональными компонентами системы. Прямая реализация операций обработки данных клиентов в данном модуле отсутствует. Основная функция заключается в последовательном вызове функций для работы с клиентами: добавление, редактирование, удаление записей, сортировка и поиск информации, а также формирование отчетов.

Файл `charUtils.h`. Это заголовочный файл с объявлениями вспомогательных функций для работы со строками и датами. В нем реализованы функции для сравнения строк, копирования, проверки символов на принадлежность к алфавиту и цифрам, а также для обработки и сравнения дат. Реализация этих функций содержится в файле `charUtils.cpp`.

Файл `checker.h`. Этот файл содержит функции для проверки состояния данных: проверки на открытие файлов, выбор действия, а также проверка валидности даты и номера телефона. Реализация этих функций находится в файле `checker.cpp`.

Файл `client.h`. В этом файле определена структура `Client`, которая включает поля для хранения информации о клиенте: ФИО, телефон, товар, дата покупки, количество и сумма. Эти данные используются в программе для обработки информации о клиентах.

Файл `clientService.h`. Этот файл содержит объявления функций для работы с клиентами: добавление, редактирование, удаление, генерация случайных клиентов и сортировка по различным параметрам. Реализация функций находится в файле `clientService.cpp`.

Файл `fileManager.h`. В этом файле описаны функции для работы с бинарными файлами. В частности, функции для записи клиентов в файл, чтения из файла и удаления всех клиентов. Реализация этих функций представлена в файле `fileManager.cpp`.

Файл `generateReport.h`. Этот файл содержит объявление функции для генерации отчетов по данным о клиентах. Отчет создается и сохраняется в

текстовый файл `report.txt`, который формируется автоматически при вызове соответствующей функции.

Файл `searchAlgorithm.h`. В этом файле содержатся объявления функций для выполнения поиска клиентов. Программа поддерживает линейный и бинарный поиск по ФИО, товару и дате. Реализация поиска доступна в файле `searchAlgorithm.cpp`.

Файл `sortAlgorithm.h`. Этот файл содержит объявления функций для сортировки клиентов. В программе предусмотрены алгоритмы быстрой сортировки по товару, сортировки выбором и сортировки вставками. Реализация алгоритмов сортировки размещена в файле `sortAlgorithm.cpp`.

1.2 Используемые структуры данных

В рамках разработки системы учета клиентов и продаж для магазина использовались структуры для хранения и обработки информации. Для хранения информации о клиентах была разработана структура `Client`, которая содержит следующие поля как `name`, `phone`, `product`. Они представляют собой массивы символов. Они содержат ФИО клиента, номер телефона, и товар, который он приобрел. Следующие поля `day`, `month`, `year` имеют тип данных `int`, и хранят дату покупки, а именно: день, месяц, год. Последнее поле - `amount`, оно имеет тип данных `double`, и хранит сумму покупок. Структура позволяет объединить различные типы данных в одну логическую единицу. Это особенно полезно, когда необходимо хранить информацию, относящуюся к одному объекту, например, клиенту магазина, включая имя, номер телефона и дату покупки [1].

1.3 Хранение данных

Для долговременного хранения информации используется бинарный файл. Все данные, содержащиеся в структурах типа `Client`, сохраняются в файл `clients.bin`. К сожалению, для работы с бинарными файлами в языке C++ предусмотрены только низкоуровневые средства — методы `read/write` стандартных типов потоков `istream/ostream`. Кроме других очевидных недостатков этот факт не даёт использовать в полную силу программирование «в стиле STL» (то есть, в первую очередь, часть стандартной библиотеки C++, связанную с алгоритмами и итераторами)[3]. Это делает код сложнее, повышает риск ошибок и мешает писать красивый, обобщённый код.

Отчёт выводится в текстовый файл `report.txt`. Данный файл создаётся автоматически при выполнении соответствующей команды и содержит информацию, которая предварительно считывается из бинарного файла `clients.bin`.

2 АЛГОРИТМЫ СОРТИРОВКИ

При реализации алгоритмов сортировки особое внимание уделено структурной целостности и читаемости функций. Функции должны быть короткими. Функция должна делать только одну вещь. Она должна делать это хорошо. Она должна делать это только[2].

2.1 Быстрая сортировка по названию товара

Быстрая сортировка (Quick Sort) — эффективный алгоритм сортировки, используемый для упорядочивания массива данных, в моём случае, списка клиентов по полю `product` в лексикографическом порядке. Выбирается опорный элемент из массива, массив разбивается на две части: элементы меньше опорного и больше/равные ему. Рекурсивно сортируются обе части. Быстрая сортировка базируется на методе «разделяй и властвуй» и демонстрирует высокую эффективность при обработке больших массивов данных [1].

В основе алгоритма быстрой сортировке лежит процедура `partition`. `Partition` выбирает некоторый элемент массива и переставляет элементы участка массива таким образом, чтобы массив разбился на 2 части: левая часть содержит элементы, которые меньше этого элемента, а правая часть содержит элементы, которые больше или равны этого элемента. Такой разделяющий элемент называется пивотом[4].

Функция `sortQuickProduct` принимает массив объектов типа `Client`, который необходимо отсортировать по полю `product`, индекс начального элемента текущего подмассива (левая граница сортировки) и индекс конечного элемента текущего подмассива (правая граница сортировки). Эти параметры позволяют функции обрабатывать произвольные подмассивы исходного массива `arr` в процессе рекурсивного деления. Переменные `left` и `right` задают диапазон индексов, в котором выполняется сортировка на текущем шаге.

Первым делом проверяется условие выхода: если левый индекс больше либо равен правому, выполнение прекращается. Это означает, что текущий диапазон либо пуст, либо состоит из одного элемента, и дальнейшая сортировка не требуется. Такое условие помогает избежать ошибок при передаче в функцию некорректных границ массива. Затем создаются две переменные, которые позволяют одновременно двигаться от левого и правого края массива к центру. Также выбирается опорный элемент — тот, который находится в середине текущего диапазона. Этот подход помогает снизить риск ухудшения производительности алгоритма до квадратичной сложности. После этого начинается основной этап — деление массива. Два указателя начинают движение навстречу друг другу. Все элементы слева от опорного должны быть меньше его значения, а справа — больше. Если находятся

элементы, которые нарушают это правило, они меняются местами, и указатели продолжают движение.

Когда разделение завершено, алгоритм рекурсивно вызывается для левой и правой части массива, исключая уже отсортированный опорный элемент. Таким образом, каждая часть обрабатывается по тому же принципу до тех пор, пока весь массив не будет отсортирован.

Этот метод сортировки использует две функции для проверки порядка названий товаров и для обмена элементов массива: `my_strcmp` и `my_swap`, которые описаны в главе «Пользовательские функции».

2.2 Сортировка выбором по дате покупки

Смысл Сортировки выбором (Selection Sort) состоит в поиске минимального значения элемента в массиве, и перемещения этого значения в начало массива[5]. Сортировка выбором (Selection Sort) — алгоритм, применяемый для упорядочивания массива данных, в моём случае, списка клиентов по дате покупке. На каждом шаге из неотсортированной части массива выбирается элемент с минимальной датой, и он перемещается в конец отсортированной части массива.

Функция `sortSelection` принимает массив объектов типа `Client`, каждый из которых содержит дату покупки, по которой выполняется сортировка, и количество элементов массива. Переменная `size` задаёт диапазон индексов от 0 до `size - 1`, в пределах которого выполняется сортировка.

Сначала внешний цикл перебирает элементы массива от начала до предпоследнего. На каждом шаге текущая позиция указывает на начало неотсортированной части массива. Во внутреннем цикле ищется элемент с минимальной датой. Для этого используется вспомогательная функция, которая сравнивает года, затем месяцы и, при необходимости, дни. Индекс найденного минимального элемента сохраняется. Если найденный элемент отличается от текущего, происходит обмен: он перемещается в начало отсортированной части массива.

Таким образом, массив шаг за шагом выстраивается в порядке возрастания даты покупки.

Этот метод сортировки использует две функции для сравнения и обмена элементов массива: `compareDates` и `my_swap`, которые описаны в главе «Пользовательские функции».

2.3 Сортировка вставками по ФИО клиента

Сортировка вставками (Insertion Sort) — это простой и эффективный алгоритм, особенно хорошо подходящий для небольших массивов данных. В моём случае он применяется для сортировки списка клиентов по их имени. Алгоритм работает следующим образом: на каждом шаге текущий элемент

сравнивается с элементами, находящимися в отсортированной части массива, и вставляется в подходящую позицию, таким образом сохраняя порядок.

Функция `sortInsertion` принимает массив объектов типа `Client`, каждый из которых содержит имя клиента, по которому и производится сортировка, и параметр `size`, который указывает количество элементов в массиве. Эти параметры позволяют функции обрабатывать массив произвольной длины. Переменная `size` определяет диапазон индексов от 0 до `size - 1`, в пределах которого выполняется сортировка.

Принцип работы алгоритма заключается в том, что внешний цикл перебирает элементы массива начиная со второго, так как первый элемент уже считается отсортированным. На каждом шаге выбирается текущий элемент `key`, который требуется правильно разместить в отсортированной части массива слева от него. Затем во внутреннем цикле текущий элемент сравнивается с предыдущими с помощью функции `my_strcmp`. Если имя текущего клиента лексикографически меньше, чем у предыдущего, элементы сдвигаются на одну позицию вправо. Процесс продолжается до тех пор, пока не находится правильная позиция для `key`. После этого он вставляется в нужное место, сохраняя упорядоченность отсортированной части массива.

Для сравнения строк используется функция `my_strcmp`, которая возвращает отрицательное значение, если первое имя меньше второго, положительное — если больше, и ноль — если они равны.

3 АЛГОРИТМЫ ПОИСКА

В процессе проектирования поисковых функций важна не только корректность, но и чистота реализации. Плохой код может работать. Но если он не чистый, он со временем приведет проект к катастрофе[2].

3.1 Линейный поиск клиентов по ФИО

Линейный поиск (Linear Search) — это простой способ поиска информации в массиве или файле, не требующий предварительной сортировки данных. Этот алгоритм перебирает все элементы в массиве, сравнивая их с заданным ключом, из-за полного перебора скорость поиска намного меньше, чем в других алгоритмах[6]. В моём случае используется для поиска клиентов по полному имени. Алгоритм работает следующим образом: каждый клиент в файле последовательно проверяется на совпадение введённого имени с хранящимся в записи.

Функция `searchLinearByName` не принимает параметров. Она работает с файлом `clients.bin`, в котором содержатся записи клиентов.

Принцип работы алгоритма заключается в том, что пользователь вводит ФИО, по которому нужно выполнить поиск. В начале выполняется проверка на ошибки при открытии файла с помощью функций `checkFileEmpty()` и `checkFileOpen()`. Затем происходит поочередное чтение каждой записи из файла. Для сравнения ФИО используется функция `my_strcmp`, которая помогает найти совпадение с введённым значением. Если совпадение найдено, данные клиента выводятся на экран с использованием функции `printClient`. Если совпадений не найдено, пользователю выводится сообщение «Клиент не найден». Для вывода информации о найденном клиенте используется функция `printClient`.

3.2 Линейный поиск клиентов по товару.

Линейный поиск также может быть применён для нахождения клиентов по названию приобретённого товара. В этом случае алгоритм аналогичен предыдущему, но сравнение производится по полю `product`. Алгоритм работает следующим образом: каждая запись в файле последовательно проверяется на совпадение названия товара с введённым значением.

Функция `searchLinearByProduct` не принимает параметров и работает напрямую с файлом `clients.bin`. Принцип работы алгоритма заключается в том, что пользователь вводит наименование товара. В начале выполняется проверка на ошибки при открытии файла с помощью функций `checkFileEmpty()` и `checkFileOpen()`. Затем каждая запись читается из файла и сравнивается с введённым значением. Для этого используется функция `my_strcmp`, которая сравнивает строки. Все найденные совпадения выводятся на экран с помощью функции `printClient`, которая выводит

информацию о каждом клиенте. Если ни один клиент с таким товаром не найден, выводится сообщение «Продукт не найден».

Для сравнения строк используется функция `my_strcmp`. Порядок записей в файле не имеет значения, так как алгоритм обходит каждую из них. Для вывода информации о найденных клиентах используется функция `printClient`.

3.3 Бинарный поиск клиентов по товару.

Бинарный поиск - очень быстрый алгоритм с не сложной реализацией, который находит элемент с определенным значением в уже отсортированном массиве[7]. В моём случае он используется для поиска клиентов по товару, предварительно сортируя массив клиентов по названию продукта. Алгоритм работает следующим образом: массив клиентов сортируется по полю `product`, после чего выполняется бинарный поиск, позволяющий быстро находить совпадения.

Функция `searchBinaryByName` не принимает параметров. Она сначала считывает все данные из файла `clients.bin`, затем сортирует их, и только после этого выполняет бинарный поиск.

Принцип работы алгоритма заключается в том, что в начале выполняется проверка на ошибки при открытии файла с помощью функций `checkFileEmpty()` и `checkFileOpen()`. Далее данные из файла считываются в массив, который затем сортируется по полю `product` с помощью функции `sortClientsByProduct`. После этого с помощью бинарного поиска осуществляется нахождение одного из элементов с нужным товаром.

Алгоритм бинарного поиска работает следующим образом: после сортировки массива клиентов функция запрашивает у пользователя название товара. Затем она устанавливает границы поиска — левый индекс `left` и правый индекс `right`. В цикле вычисляется средний элемент массива, и его значение сравнивается с искомым товаром. В классическом бинарном поиске, как только элемент в середине совпадает с искомым, поиск останавливается. Однако в этом случае, так как нужно найти все совпадения, поиск продолжается. После нахождения первого совпадения алгоритм продолжает искать совпадения влево и вправо от найденного индекса. Сначала он проверяет элементы слева от найденного индекса, а затем — справа, чтобы вывести все записи с этим товаром. Если элемент в середине массива меньше искомого, то поиск продолжается в правой части массива, сдвигая левую границу. Если элемент больше, то поиск осуществляется в левой части массива, сдвигая правую границу.

Для вывода информации о найденных клиентах используется функция `printClient`, которая выводит полную информацию о каждом клиенте. Если совпадений не найдено, пользователю выводится сообщение «Товар не найден». Для сравнения строк используется функция `my_strcmp`.

3.4 Поиск клиентов по товару и дате.

Данная функция выполняет выборку клиентов, купивших определённый товар позже указанной даты. В отличие от других алгоритмов, здесь используется комбинированное условие поиска, а затем результаты сортируются по ФИО.

Суть алгоритма заключается в том, что сначала из файла выбираются только те клиенты, у которых совпадает наименование товара и дата покупки позже указанной, после чего найденные записи сортируются по имени.

Функция `searchByProductAndDate` не принимает параметров. Пользователь вводит наименование товара и дату. В начале выполняется проверка на ошибки при открытии файла с помощью функций `checkFileEmpty()` и `checkFileOpen()`. Далее происходит чтение каждой записи из файла и проверка условий: товар должен совпасть с введённым, а дата покупки должна быть позже указанной. Для проверки корректности даты используется функция `isValidDate()`, а сама дата парсится с помощью функции `parseDate()`. После завершения чтения массива выполняется сортировка результатов по ФИО методом сортировки вставками. Для сортировки строк используется функция `my_strcmp`. Затем все найденные и отсортированные записи выводятся пользователю с помощью функции `printClient`. Если подходящих записей не найдено, выводится сообщение «Покупки не найдены».

4 ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ

Ключевую роль в работе программы играют пользовательские функции, которые должны быть лаконичны и самодокументируемы. Хорошие имена функций и переменных — это не украшение, а необходимость. Плохо подобранные имена — причина множества багов[2].

4.1 Функции для работы со строками и датами

Данный блок содержит реализацию пользовательских функций для работы со строками (сравнение, копирование, определение длины, объединение), а также функцию валидации даты. Эти функции являются аналогами стандартных функций языка C++ и используются в других модулях программы для обеспечения независимости от библиотек. С целью улучшения читаемости кода и структурирования работы, реализация данных функций была вынесена в отдельные файлы: `charUtils.cpp` и `charUtils.h`. Такой подход обеспечивает логичное разделение кода и облегчает его поддержку и масштабирование.

Функция `my_strcmp` выполняет сравнение двух строк. Она принимает два указателя на строки `str1` и `str2` и поочередно сравнивает символы из обеих строк. Сравнение продолжается до тех пор, пока символы совпадают и не будет достигнут конец одной из строк. После этого функция возвращает разницу ASCII-кодов первых несовпадающих символов. Для корректного сравнения используется приведение символов к типу `unsigned char`, что помогает избежать ошибок при работе с символами в разных кодировках, включая русские буквы. Эта функция полностью имитирует стандартную функцию `strcmp` и работает корректно для строк в различных кодировках.

Функция `my_strcpy` выполняет копирование строки. Она принимает два указателя: `dest` — куда копировать, и `src` — откуда копировать. Строка `src` копируется в строку `dest`, включая завершающий нулевой символ. После копирования функция возвращает указатель на начало строки назначения. Эта функция аналогична стандартной функции `strcpy` и выполняет копирование строки по символам. Также предусмотрена возможность копировать строку в саму себя, то есть, когда `src` и `dest` указывают на одну и ту же строку.

Функция `my_strncpy` выполняет копирование строки с ограничением по количеству символов. Она принимает три параметра: `dest` — строку назначения, `src` — строку источника и `n` — максимальное количество символов для копирования. Строка `src` копируется в строку `dest` до достижения конца строки или до копирования не более `n` символов. Если в строке `src` меньше символов, чем `n`, оставшиеся символы в строке `dest` заполняются нулями (`\0`). Эта функция работает аналогично стандартной функции `strncpy`, обеспечивая корректное завершение строки нулевым символом, даже если исходная строка короче `n`.

Функция `my_strlen` принимает указатель на строку и считает количество символов до символа `\0`, возвращая длину строки. Аналогична стандартной функции `strlen`. Она подходит для строк любой длины и не зависит от содержимого строки, просто подсчитывает символы до первого нулевого символа.

Функция `my_isalpha` принимает символ `ch` и проверяет, принадлежит ли он к латинским или русским буквам. Включает поддержку кириллицы, что важно при работе с русскоязычными ФИО и товарами. Возвращает `true`, если символ является буквой, и `false`, если нет.

Функция `my_strncmp` сравнивает не более `n` символов из строк `str1` и `str2`. Посимвольно сравнивает строки до `n` символов или пока не встретится несовпадение. Возвращает разность первых несовпадающих символов. Эта функция аналогична стандартной `strncmp` и используется, например, при сортировке по первым нескольким символам строки или если нужно проверить, одинаковы ли строки в пределах определённого числа символов.

Функция `my_isdigit` принимает символ `ch` и проверяет, является ли символ цифрой в диапазоне от 0 до 9. Возвращает `true`, если символ является цифрой, и `false`, если нет. Это альтернатива стандартной функции `isdigit`.

Функция `my_strncat` добавляет не более `n` символов из строки `src` в конец строки `dest`. Сначала находит конец строки `dest`, затем добавляет символы из `src` (не более `n`), а в конце добавляется символ `\0`, чтобы сохранить корректную завершённость строки. Эта функция используется для безопасного объединения строк, не выходя за пределы буфера и не нарушая структуры строки.

Функция `parseDate` выполняет разбор строки с датой и проверку её корректности. Она принимает строку `data` и указатели на переменные `day`, `month` и `year`, в которые записываются значения после успешного анализа. Функция поддерживает несколько возможных разделителей — точку, дефис, косую черту и пробел — и приводит их к одному виду, заменяя на пробел. Ожидается формат DD MM YYYY. После разбиения строки извлекаются числовые значения и проверяется, что день находится в пределах от 1 до 31, месяц — от 1 до 12, а год — от 1 до 2025. Далее выполняется проверка соответствия количества дней в месяце, включая корректную обработку февраля в зависимости от високосности года. Год считается високосным, если он делится на 4, но не делится на 100, либо делится на 400. Функция подходит для строгой валидации пользовательского ввода: отказывает при наличии некорректных символов.

4.2 Функции валидации ввода и работы с файлом

Данный блок содержит функции, отвечающие за корректность пользовательского ввода и проверку состояния файла данных `clients.bin`. Реализованы проверки открытия и пустоты файла, обработка ввода чисел и строк, проверка телефонного номера и корректности даты. Все функции

вынесены в отдельные модули для обеспечения логичной архитектуры: `checker.h` и `checker.cpp`. Такой подход способствует улучшению читаемости и структурированности кода, делает его короче, избавляя основной модуль программы от повторяющихся проверок, и значительно упрощает сопровождение и тестирование логики обработки пользовательского ввода.

Функция `checkFileOpen` открывает файл `clients.bin` в бинарном режиме для проверки его доступности. Если файл не удаётся открыть, выводится сообщение об ошибке. Возвращает `true` при успешном открытии и `false` в противном случае. Применяется для предварительной проверки перед операциями чтения или записи, чтобы избежать ошибок обращения к несуществующему файлу.

Функция `checkFileEmpty` определяет, содержит ли файл записи структуры `Client`. Размер файла делится на размер структуры, чтобы получить количество записей. Если записей нет, выводится предупреждение. Эта проверка позволяет избежать бессмысленных операций с пустым файлом, например при попытке чтения или сортировки.

Функция `checkChoice` запрашивает ввод от пользователя и проверяет, что введено целое число в пределах от 0 до 14. Обработывает любые ошибки ввода, включая буквы, пустые строки и числа вне допустимого диапазона. Обеспечивает надёжность при работе с меню и защищает программу от аварийного завершения при ошибочном вводе.

Шаблонная функция `checkPositiveNum` предназначена для ввода положительных чисел с типами `int` и `double`. При вводе типа `double` автоматически заменяет запятые на точки, обеспечивая поддержку обоих вариантов записи. Функция выполняет проверку на корректность и возвращает число только в случае валидного ввода. Используется в ситуациях, где требуется контроль числовых параметров, например при вводе стоимости или количества.

Функция `checkPhoneNum` считывает строку и проверяет, что она содержит только цифры и, при необходимости, начинается с символа «+». Контролирует максимальную длину номера и отклоняет некорректные данные. При успешной проверке записывает результат в `output`. Используется для валидации номеров телефонов в анкете клиента, обеспечивая правильность формата контакта.

Функция `isValidDate` анализирует строку с датой, заменяя все допустимые разделители — точку, дефис, косую черту и пробел — на пробелы. После этого извлекает значения дня, месяца и года, проверяя их на соответствие допустимым диапазонам, включая корректную обработку високосных годов. Применяется для строгой валидации даты, введённой пользователем, и отклоняет строки с неправильным форматом или ошибками.

4.4 Функции управления бинарными файлами

Данный блок реализует функции работы с бинарным файлом клиентов: сохранение, отображение содержимого и удаление файла. Все функции сгруппированы в модулях `fileManager.cpp` и `fileManager.h`, что обеспечивает разделение ответственности и улучшает читаемость работы.

Функция `saveToBinaryFile` принимает константную ссылку на структуру `Client` и записывает её в файл `clients.bin`. Перед началом записи выполняется проверка на возможность открытия файла с использованием функции `checkFileOpen`. Затем файл открывается в бинарном режиме с добавлением (`ios::app`), что позволяет сохранить новые данные без потери предыдущих. Если файл отсутствует, он создаётся автоматически. Структура клиента записывается в файл целиком с использованием `reinterpret_cast`, обеспечивая точную запись всех полей. По завершении файл закрывается. Функция предназначена для безопасного и надёжного добавления клиентов в базу данных.

Функция `displayBinaryFile` открывает файл `clients.bin` и выводит его содержимое на экран. Перед этим выполняются проверки существования файла и наличия в нём данных с помощью функций `checkFileOpen` и `checkFileEmpty`. При успешном прохождении проверок файл открывается в бинарном режиме, и его содержимое считывается по одной структуре `Client` за раз. Каждая запись выводится на экран с помощью функции `printClient`. Реализация гарантирует стабильную работу и защищает от ошибок, связанных с отсутствием файла или данных.

Функция `removeClients` предназначена для удаления содержимого файла `clients.bin`. Она выполняет последовательные проверки с помощью функций `checkFileOpen` и `checkFileEmpty`, чтобы убедиться, что файл доступен и не содержит данных. Если оба условия выполняются, то из файла удаляются все записи. После успешной операции выводится сообщение об удалении всех записей клиентов. Такая реализация обеспечивает безопасное удаление содержимого файла, предотвращая попытки очистки несуществующего или пустого файла.

4.5 Функции генерация отчёта

Данный блок реализует функцию анализа и генерации текстового отчёта на основе содержимого бинарного файла `clients.bin`. Основные задачи: группировка клиентов по товарам, сортировка данных, а также определение самого популярного товара и самого активного клиента. Функция размещена в модулях `generateReport.cpp` и `generateReport.h`, а для сортировки и сравнения строк используются функции из модулей `sortAlgorithm` и `charUtils`. Проверка состояния файла осуществляется с помощью функций из модуля `checker`.

Функция `generateReport` не принимает параметров и не возвращает значение. На первом этапе производится проверка доступности бинарного файла с помощью функции `checkFileOpen`. Если файл пуст, выполнение прерывается с выводом соответствующего сообщения. Далее происходит считывание данных всех клиентов в динамический массив. После этого открывается текстовый файл `report.txt`, предназначенный для сохранения отчёта. В случае ошибки открытия выводится сообщение, и выделенные ресурсы освобождаются.

Сначала массив клиентов сортируется по названию товара с использованием алгоритма быстрой сортировки. Внутри каждой группы клиентов, относящихся к одному товару, применяется сортировка по дате покупки с помощью сортировки выбором. Отсортированные данные выводятся построчно в консоль и в файл: сначала указывается наименование товара, затем — клиенты с датой покупки и количеством.

В процессе обработки подсчитывается общее количество покупок для каждого товара, чтобы определить самый популярный. В отчёт добавляется информация о товаре с наибольшим числом продаж. Затем вычисляется суммарная стоимость покупок по каждому клиенту, на основе чего определяется клиент с максимальной суммой. Этот клиент считается самым активным, и его данные также включаются в отчёт.

По завершении выполнения все ресурсы освобождаются, файлы закрываются, и в консоль выводится сообщение об успешном сохранении отчёта.

4.6 Функции работы с клиентами

Данный блок реализует различные функции для управления базой данных клиентов. Основные задачи: добавление, редактирование, удаление, сортировка клиентов по различным параметрам, а также генерация случайных данных. Функции размещены в модулях `clientService.cpp` и `clientService.h`, и используют вспомогательные модули для валидации данных и записи в файл.

Функция `printClient` выводит детализированную информацию о клиенте. Она принимает ссылку на структуру `Client` и отображает все её поля: ФИО, телефон, товар, дата, количество и сумма. Это полезная функция для вывода полной информации о клиенте на экране.

Функция `addClient` предназначена для добавления нового клиента в базу данных. На первом этапе функция запрашивает у пользователя данные клиента: ФИО, телефон, товар, дату, количество и сумму. Для каждого поля выполняются проверки. Телефон проверяется с использованием функции `checkPhoneNum`, которая проверяет его корректность. Для ввода даты используется вспомогательная функция `setDate`, которая проверяет корректность введенной даты и устанавливает её. Количество проверяется и сумма с помощью функции `checkPositiveNum`. После прохождения всех

проверок, данные клиента сохраняются в бинарный файл `clients.bin` с помощью функции `saveToBinaryFile`.

Функция `setDate` предназначена для установки даты клиента. Она обрабатывает ввод даты пользователем в одном из трёх допустимых форматов: «ДД.ММ.ГГГГ», «ДД/ММ/ГГГГ» или «ДД-ММ-ГГГГ». Введенная строка затем проверяется с помощью функции `isValidDate`, которая удостоверяется в её корректности. Если дата валидна, она парсится в отдельные компоненты (день, месяц, год) с помощью функции `parseDate`, и эти данные сохраняются в структуре клиента. Если ввод некорректен, программа выводит сообщение об ошибке и предлагает ввести дату заново.

Функция `editClient` позволяет редактировать данные существующего клиента. Сначала проверяется доступность файла `clients.bin` с помощью функций `checkFileEmpty` и `checkFileOpen`. Затем выполняется поиск клиента по ФИО в файле. Если клиент найден, предлагается выбрать одно из полей для редактирования: телефон, товар, дату, количество или сумму. Для каждого поля перед внесением изменений выполняется соответствующая проверка. Например, телефон проверяется функцией `checkPhoneNum`, дата — функцией `setDate`, количество и сумма проверяются с использованием функций `checkPositiveNum`. После того как пользователь выбрал нужное поле и внес изменения, файл обновляется, и измененная запись перезаписывается в тот же файл. Если клиент с указанным ФИО не найден, программа выводит сообщение об ошибке.

Функция `deleteClient` выполняет удаление клиента из базы данных. Сначала проверяется наличие данных в файле и его доступность с помощью функций `checkFileEmpty` и `checkFileOpen`. Если файл не пуст и доступен, программа запрашивает ФИО клиента, которого нужно удалить. Затем создается временный файл `temp.bin`, в который записываются все клиенты, кроме того, который подлежит удалению. После завершения записи старый файл `clients.bin` удаляется, а временный файл переименовывается в `clients.bin`. Если клиент с указанным ФИО не найден, временный файл удаляется, а пользователю выводится соответствующее сообщение.

Функции `sortClientsByProduct`, `sortClientsByData` и `sortClientsByName` выполняют сортировку клиентов по различным параметрам: товару, дате или ФИО. В каждой из этих функций сначала проверяется доступность файла с данными клиентов с помощью `checkFileEmpty` и `checkFileOpen`. Затем все записи считываются в массив, после чего выполняется сортировка с использованием одного из алгоритмов сортировки: быстрая сортировка (в случае сортировки по товару), сортировка выбором (для сортировки по дате) и сортировка вставками (для сортировки по ФИО). После сортировки данные перезаписываются в файл `clients.bin`. Если файл не доступен или пуст, функция завершится без выполнения сортировки.

Функция `getCountClient` выводит количество клиентов в базе данных. Для этого открывается файл `clients.bin` и определяется количество записей

путем деления размера файла на размер одной записи структуры `Client`. Полученный результат выводится в консоль.

Функция `generateRandomClient` генерирует случайных клиентов с предопределенными данными: ФИО, телефон, товар, дата, количество и сумма. Для каждого клиента случайным образом выбирается пол, что влияет на выбор имени, фамилии и отчества. В случае мужского клиента выбираются имена и фамилии из списка мужских, в случае женского — из женского списка. Также для мужчин и женщин используются разные варианты отчеств. Сгенерированные данные затем записываются в структуру `Client`. Также для каждого клиента случайным образом генерируется телефон в формате "9XXXXXXXXX", выбирается товар из предустановленного списка, случайная дата (в диапазоне от 2000 до 2023 года), количество (от 1 до 100) и сумма (от 100 до 10000). Все сгенерированные клиенты сохраняются в файл `clients.bin`. После завершения генерации выводится сообщение о том, сколько клиентов было создано.

5 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

5.1 Краткое описание программы

Разработанная программа представляет собой консольное приложение для ведения базы данных клиентов с сохранением информации в бинарный файл. Программа предоставляет возможность добавления, редактирования, удаления клиентских записей, выполнять сортировку по различным параметрам, генерировать случайные данные, а также формировать отчёт. Ниже приводится инструкция по работе с программой и описание возможных исключительных ситуаций.

После запуска программы пользователю отображается текстовое меню с перечнем доступных действий (рисунок 5.1). Управление осуществляется путём ввода номера выбранного пункта меню. Для обеспечения корректного функционирования программы реализована проверка всех вводимых значений. Ввод недопустимых символов и чисел вне диапазона пунктов меню блокируется. При обнаружении ошибки на экран выводится соответствующее сообщение, информирующее пользователя о некорректном вводе. После этого предоставляется возможность повторного ввода корректного значения.

```
Меню:
1. Добавить клиента
2. Просмотреть записи
3. Редактировать клиента
4. Удалить клиента
5. Сортировать клиентов по наименованию товара(быстрая)
6. Сортировать клиентов по дате покупки(выбором)
7. Сортировать клиентов по ФИО(вставками)
8. Поиск по ФИО(линейный)
9. Поиск по товару (линейный)
10. Поиск по товару (бинарный)
11. Поиск по товару и дате(покупка товара после введенной даты)
12. Создать отчет
13. Сгенерировать случайных клиентов
14. Узнать количество клиентов
15. Удалить всех клиентов
0. Выход
Выберите действие: █
```

Рисунок 5.1 – Текстовое меню при запуске программы.

5.2 Описание функциональных возможностей программы

При выборе первого пункта меню программа последовательно запрашивает у пользователя следующие данные: ФИО, номер телефона, товар, дату покупки, количество и сумму (рисунок 5.2).

Каждое поле сопровождается проверкой корректности введенных данных. Дата проходит валидацию на существование с учётом високосных

лет, при этом допускаются различные разделители: точка, пробел, косая черта и дефис. Количество и сумма должны быть положительными числами. В случае ошибки выводится соответствующее сообщение, после чего пользователь повторяет ввод.

```
Выберите действие: 1
Введите ФИО: Раз Два Три
Телефон: 23424456
Товар: Книга
Дата (используйте . / -): 23.04.2024
Количество: 1
Сумма: 12
Клиент добавлен!
```

Рисунок 5.2 – Этапы ввода данных о клиенте.

Пункт 2 меню позволяет пользователю отобразить всех клиентов, хранящихся в файле (рисунок 5.3).

```
Выберите действие: 2
ФИО: Раз Два Три
Телефон: 23424456
Товар: Книга
Дата: 23.4.2024
Количество: 1
Сумма: 12
```

Рисунок 5.3 – Отображение клиентов.

Если записи о клиентах удалены, то выводится сообщение «Файл пуст». Если файл с клиентами отсутствует или не открывается, то выводится уведомление «Ошибка доступа к файлу, возможно, он не существует».

При выборе пункта 3 пользователь может изменить данные уже существующего клиента. Сначала программа запрашивает ФИО клиента, затем предлагает выбрать поле для редактирования: номер телефона, товар, дата, количество или сумма. Если ввести неверное значение пункт изменения, то выводится сообщение «Неверный выбор». Все новые значения проходят проверку, аналогичную при добавлении, если проверки не будут пройдены, то выводится соответствующее сообщение и запрашивается повторный ввод. После изменения данные перезаписываются в файл на прежнем месте и выводится сообщение «Данные успешно обновлены». Если запись клиента с таким ФИО отсутствует, то выводится сообщение «Клиент не найден». Перед выполнением команды проверяется доступность файла и наличие записей в нём. В случае ошибки отображается уведомление. Пример процесса изменения данных показан на рисунке 5.4.

```
Введите ФИО клиента для редактирования: Раз Два Три
Выберите поле для редактирования:
1. Телефон
2. Товар
3. Дата
4. Количество
5. Сумма
Ваш выбор: 1
Новый телефон: 45242
Данные успешно обновлены!
```

Рисунок 5.4 – Процесс изменения данных.

Пункт 4 позволяет удалить клиента по ФИО. Если клиент не будет найден, то выводится сообщение «Клиент не найден». Перед выполнением команды проверяется доступность файла и наличие записей в нём. В случае ошибки отображается уведомление. Процесс удаления клиента можно увидеть на рисунке 5.5.

```
Выберите действие: 4
Введите ФИО клиента для удаления: Раз Два Три
Запись успешно удалена
```

Рисунок 5.5 – Процесс удаления данных.

Пункт 5 выполняет сортировку клиентов по алфавиту названий товаров. После сортировки выводится сообщение: «Сортировка выполнена успешно»(рисунок 5.6). Перед выполнением команды проверяется доступность файла и наличие записей в нём. В случае ошибки отображается уведомление.

```
Меню:
1. Добавить клиента
2. Просмотреть записи
3. Редактировать клиента
4. Удалить клиента
5. Сортировать клиентов по наименованию товара(быстрая)
6. Сортировать клиентов по дате покупки(выбором)
7. Сортировать клиентов по ФИО(вставками)
8. Поиск по ФИО(линейный)
9. Поиск по товару (линейный)
10. Поиск по товару (бинарный)
11. Поиск по товару и дате(покупка товара после введенной даты)
12. Создать отчет
13. Сгенерировать случайных клиентов
14. Узнать количество клиентов
15. Удалить всех клиентов
0. Выход
Выберите действие: 5
Сортировка выполнена успешно
```

Рисунок 5.6 – Процесс сортировки.

При выборе пункта 6 выполняется сортировка клиентов по дате покупки. После сортировки выведется сообщение: «Сортировка выполнена успешно». Перед выполнением команды проверяется доступность файла и наличие записей в нём. В случае ошибки отображается уведомление.

Пункт 7 использует сортировку клиентов по фамилии, имени и отчеству. После сортировки выведется сообщение: «Сортировка выполнена успешно». Перед выполнением команды проверяется доступность файла и наличие записей в нём. В случае ошибки отображается уведомление.

Пункт 8 позволяет найти клиента по ФИО. При отсутствии совпадений программа выводит сообщение «Клиент не найден». Перед выполнением команды проверяется доступность файла и наличие записей в нём. В случае ошибки отображается уведомление. Пример поиска клиента по ФИО можно увидеть на рисунке 5.7.

```
Выберите действие: 8
Введите ФИО для поиска: Иванова Мария Алексеевна
ФИО: Иванова Мария Алексеевна
Телефон: 9462279637
Товар: Телевизор
Дата: 12.7.2016
Количество: 95
Сумма: 308.67
```

Рисунок 5.7 – Процесс поиска по ФИО.

Функция из пункта 9 выполняет поиск клиентов, купивших указанный товар. Если товар не найден, то выводится сообщение «Продукт не найден». Все подходящие записи отображаются в консоли. Перед выполнением команды проверяется доступность файла и наличие записей в нём. В случае ошибки отображается уведомление. Процесс поиска по товару показан на рисунке 5.8.

```
Выберите действие: 9
Введите наименование товара: Телевизор
ФИО: Смирнова Анна Ивановна
Телефон: 9720218019
Товар: Телевизор
Дата: 6.10.2017
Количество: 36
Сумма: 153.33

ФИО: Иванова Анна Дмитриевна
Телефон: 9574027731
Товар: Телевизор
Дата: 22.4.2016
Количество: 56
Сумма: 298.17

ФИО: Иванова Мария Алексеевна
Телефон: 9462279637
Товар: Телевизор
Дата: 12.7.2016
Количество: 95
Сумма: 308.67
```

Рисунок 5.8 – Поиск клиентов по указанному товару.

Пункт 10 выполняет поиск клиентов по товару, но бинарным способом. Такой метод быстрее, но требует отсортированных данных. После выбора этого пункта, происходит автоматическая сортировка для бинарного поиска. Перед выполнением команды проверяется доступность файла и наличие записей в нём. В случае ошибки отображается уведомление. Пример поиска представлен на рисунке 5.9.

```
Выберите действие: 10
Сортировка выполнена успешно
Введите наименование товара: Холодильник
ФИО: Кузнецов Петр Сергеевич
Телефон: 9298779218
Товар: Холодильник
Дата: 25.10.2022
Количество: 65
Сумма: 269.07

ФИО: Иванова Анна Алексеевна
Телефон: 9727050827
Товар: Холодильник
Дата: 29.5.2013
Количество: 14
Сумма: 309.29

ФИО: Васильев Иван Петрович
Телефон: 9529455608
Товар: Холодильник
Дата: 20.11.2006
Количество: 58
Сумма: 355.49
```

Рисунок 5.9 – Бинарный поиск клиентов по товару.

Пункт 11 позволяет найти клиентов, купивших определённый товар после указанной даты. Если введенный товар не найден, то выводится сообщение «Товар не найден». При вводе даты проверяется ее корректность, с учётом високосного года, если проверка не пройдена, то выводится соответствующее сообщение и требуется повторный ввод. Записи, не соответствующие фильтру, отбрасываются. Перед выполнением команды проверяется доступность файла и наличие записей в нём. В случае ошибки отображается уведомление. Пример такого поиска можно увидеть на рисунке 5.10.


```

Выберите действие: 11
Введите наименование товара: Телевизор
Введите дату(используйте . / -): 21.05.2008
ФИО: Иванова Анна Дмитриевна
Телефон: 9574027731
Товар: Телевизор
Дата: 22.4.2016
Количество: 56
Сумма: 298.17

ФИО: Иванова Мария Алексеевна
Телефон: 9462279637
Товар: Телевизор
Дата: 12.7.2016
Количество: 95
Сумма: 308.67

ФИО: Смирнова Анна Ивановна
Телефон: 9720218019
Товар: Телевизор
Дата: 6.10.2017
Количество: 36
Сумма: 153.33

```

Рисунок 5.10 – Поиск по товару и дате покупки.

Пункт 12 запускает генерацию текстового отчёта, сохраняемого в `report.txt`. Отчет содержит сгруппированные по товару записи клиентов, самую популярную продукцию, самого активного клиента по сумме покупок. Если записи о клиентах удалены, или файл с клиентами отсутствует, или не открывается, то выводится сообщение «Нет данных для отчета». Пример формирования отчета показан на рисунке 5.11.

```

Выберите действие: 12

=== ОТЧЕТ ===
Товар: Ноутбук
  17.11.2008 - Кузнецова Мария Алексеевна (28 шт.)
  24.1.2010 - Петрова Мария Алексеевна (43 шт.)
  7.4.2023 - Смирнов Дмитрий Петрович (19 шт.)

Товар: Смартфон
  27.4.2009 - Петров Дмитрий Сергеевич (44 шт.)
  25.2.2018 - Смирнов Дмитрий Петрович (4 шт.)
  16.6.2019 - Кузнецова Мария Алексеевна (78 шт.)

Товар: Телевизор
  10.5.2006 - Петров Петр Иванович (18 шт.)
  22.4.2016 - Иванова Анна Дмитриевна (56 шт.)
  12.7.2016 - Иванова Мария Алексеевна (95 шт.)
  6.10.2017 - Смирнова Анна Ивановна (36 шт.)

Товар: Холодильник
  20.11.2006 - Васильев Иван Петрович (58 шт.)
  29.5.2013 - Иванова Анна Алексеевна (14 шт.)
  25.10.2022 - Кузнецов Петр Сергеевич (65 шт.)

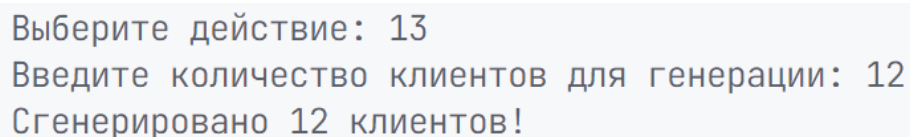
Самый популярный товар: Телевизор (205 шт.)
Самый активный клиент: Кузнецова Мария Алексеевна (Сумма: 487.36)

Отчет сохранен в report.txt

```

Рисунок 5.11 – Формирование отчёта.

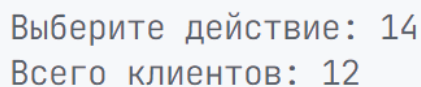
При выборе пункта 13 программа запрашивает количество записей, генерирует случайные данные (ФИО, товар, дата, количество, сумма) и добавляет их в базу. Эта функция может использоваться как для дополнения существующей базы клиентов, так и для создания нового файла с данными, если он отсутствует. Она полезна для тестирования и демонстрации обработки большого объёма данных. Этот процесс можно увидеть на рисунке 5.12.



Выберите действие: 13
Введите количество клиентов для генерации: 12
Сгенерировано 12 клиентов!

Рисунок 5.12 – Генерация клиентских данных.

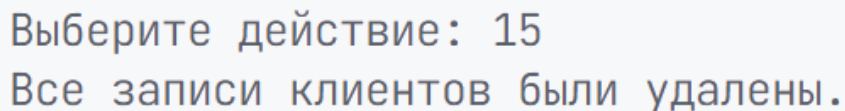
Пункт 14 отображает количество клиентов, как показано на рисунке 5.13. Перед выполнением команды проверяется доступность файла и наличие записей в нём. В случае ошибки отображается уведомление. Если все клиенты удалены, то выведет результат ноль.



Выберите действие: 14
Всего клиентов: 12

Рисунок 5.13 – Отображение количества клиентов.

Пункт 15 удаляет весь файл, где хранятся данные о клиентах. Этот процесс отображён на рисунке 5.14. Перед выполнением команды проверяется доступность файла и наличие записей в нём. В случае ошибки отображается уведомление.



Выберите действие: 15
Все записи клиентов были удалены.

Рисунок 5.14 – Удаление клиента из базы.

Пункт 0 завершает работу программы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана консольная программа «Система учёта клиентов и продаж для магазина», предназначенная для автоматизации хранения, обработки и анализа клиентской информации и данных о покупках.

Большое внимание уделялось качеству исходного кода. Чистый код — это результат тщательной работы и постоянного внимания. Хорошие программисты пишут его, потому что заботятся[2].

Курсовая работа полностью реализует поставленные во введении цели и задачи. Создана структура хранения данных, включающая персональные сведения о клиентах и информацию о приобретённых товарах. Реализован надёжный механизм ввода с валидацией данных, включая проверку корректности телефона, даты, количества и суммы покупки. Вся информация сохраняется в бинарный файл, что позволяет эффективно управлять данными.

В рамках работы были внедрены алгоритмы сортировки: быстрая сортировка по названию товара, сортировка выбором по дате покупки и сортировка вставками по ФИО клиента. Для поиска реализованы как простые, так и более сложные алгоритмы: линейный поиск по имени и товару, бинарный поиск по товару, а также комбинированный поиск по товару и дате.

Особое внимание уделено аналитическим возможностям программы. Создаётся отчёт, в котором по каждому товару формируется хронологический список покупок. Также автоматически определяется самый популярный товар и самый активный клиент, что позволяет использовать результаты для маркетингового и управленческого анализа.

Все функции программы были протестированы, в том числе в граничных и исключительных ситуациях: при отсутствии данных, при ошибочном вводе, при попытке удаления пустого файла и другие. Это повышает надёжность и устойчивость приложения в реальных условиях.

Таким образом, в рамках курсовой работы была создана полнофункциональная система учёта клиентов и продаж, способная существенно упростить ведение базы данных, ускорить поиск информации и предоставить информацию для анализа. Полученное решение может быть использовано как основа для более масштабных информационных систем, применимых в реальной торговой практике.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Шилдт, Г. С++: базовый курс / Г. Шилдт. – М. : Вильямс, 2003. – 640 с.
- [2] Мартин, Р. Чистый код: создание, анализ и рефакторинг / Р. Мартин. – СПб. : Питер, 2019. – 464 с.
- [4] Быстрая сортировка [Электронный ресурс] // Хабр. – 2021. – Режим доступа : <https://habr.com/ru/companies/otus/articles/524948/> (дата обращения: 29.05.2025).
- [3] Работа с бинарными файлами в С++ [Электронный ресурс] // Хабр. – 2011. – Режим доступа : <https://habr.com/ru/articles/134788/> (дата обращения: 29.05.2025).
- [5] Сортировка выбором С++ [Электронный ресурс] // Pure С++ – 2022. – Режим доступа : <https://purecodecpp.com/archives/2562> (дата обращения: 29.05.2025).
- [6] Линейный поиск в С++: подробное руководство [Электронный ресурс] // Code Lessons – 2022. – Режим доступа : <https://codelessons.dev/ru/linejnyj-poisk-po-massivu-c/> (дата обращения: 29.05.2025).
- [7] Бинарный поиск в С++: подробное руководство [Электронный ресурс] // Code Lessons – 2022. – Режим доступа : <https://codelessons.dev/ru/binarnyj-poisk-po-massivu-c/> (дата обращения: 29.05.2025).

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

main.cpp

```
#include <iostream>
#include <ctime>

#include "sortAlgorithm.h"
#include "searchAlgorithm.h"
#include "clientService.h"
#include "fileManager.h"
#include "searchAlgorithm.h"
#include "generateReport.h"
#include "checker.h"

using namespace std;

int main()
{
    srand(time(0));
    short choice = 0;
    do
    {

        cout << "\nМеню:\n";
        cout << "1. Добавить клиента\n";
        cout << "2. Просмотреть записи\n";
        cout << "3. Редактировать клиента\n";
        cout << "4. Удалить клиента\n";
        cout << "5. Сортировать клиентов по наименованию товара (быстрая) \n";
        cout << "6. Сортировать клиентов по дате покупки (выбором) \n";
        cout << "7. Сортировать клиентов по ФИО (вставками) \n";
        cout << "8. Поиск по ФИО (линейный) \n";
        cout << "9. Поиск по товару (линейный) \n";
        cout << "10. Поиск по товару (бинарный) \n";
        cout << "11. Поиск по товару и дате (покупка товара после введенной
даты) \n";
        cout << "12. Создать отчет\n";
        cout << "13. Сгенерировать случайных клиентов\n";
        cout << "14. Узнать количество клиентов\n";
        cout << "15. Удалить всех клиентов\n";
        cout << "0. Выход\n";
        cout << "Выберите действие: ";

        choice = checkChoice();

        switch (choice)
        {
            case 1:
                addClient();
```

Продолжение приложения А

```
break;
case 2:
displayBinaryFile();
break;
case 3:
editClient();
break;
case 4:
deleteClient();
break;
case 5:
sortClientsByProduct();
break;
case 6:
sortClientsByData();
break;
case 7:
sortClientsByName();
break;
case 8:
searchLinearByName();
break;
case 9:
searchLinearByProduct();
break;
case 10:
searchBinaryByName();
break;
case 11:
searchByProductAndDate();
break;
case 12:
generateReport();
break;

case 13:
generateRandomClient();
break;
case 14:
getCountClient();
break;
case 15:
removeClients();
break;
case 0:
exit(0);
default:
cout << "Неверный выбор\n";
}
} while (choice != 0);
}
```

Продолжение приложения А

charUtils.h

```
#ifndef CHARUTILS_H
#define CHARUTILS_H

#include <stddef> // for size_t

int my_strcmp(const char *str1, const char *str2);

char *my_strcpy(char *dest, const char *src);

char *my_strncpy(char *dest, const char *src, size_t n);

int my_strncmp(const char *str1, const char *str2, size_t n);

size_t my_strlen(const char *str);

bool my_isalpha(char ch);

bool my_isdigit(char ch);

char *my_strncat(char *dest, const char *src, size_t n);

bool parseDate(const char *data, int *day, int *month, int *year);

#endif
```

charUtils.cpp

```
#include "charUtils.h"
using namespace std;

int my_strcmp(const char *str1, const char *str2)
{
    while (*str1 != '\0' && (*str1 == *str2))
    {
        str1++;
        str2++;
    }

    return static_cast<int>(static_cast<unsigned char>(*str1) -
        static_cast<unsigned char>(*str2));
}

char *my_strcpy(char *dest, const char *src)
{
    char *original_dest = dest;
```

Продолжение приложения А

```
while ((*dest++ = *src++) != '\0')
;
return original_dest;
}

char *my_strncpy(char *dest, const char *src, size_t n)
{
char *original_dest = dest;
size_t i;

for (i = 0; i < n && src[i] != '\0'; ++i)
{
dest[i] = src[i];
}

for (; i < n; ++i)
{
dest[i] = '\0';
}

return original_dest;
}

size_t my_strlen(const char *str)
{
size_t length = 0;
while (str[length] != '\0')
{
++length;
}
return length;
}

bool my_isalpha(char ch)
{
if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z'))
{
return true;
}
// Русские буквы в кодировке Windows-1251
if ((ch >= '\xC0' && ch <= '\xDF') ||
(ch >= '\xE0' && ch <= '\xFF'))
{
return true;
}
return false;
}

int my_strncmp(const char *str1, const char *str2, size_t n)
{
for (size_t i = 0; i < n; ++i)
```


Продолжение приложения А

```
{
if (str1[i] == '\\0' || str2[i] == '\\0' || str1[i] != str2[i])
{
return static_cast<int>(static_cast<unsigned char>(str1[i]) -
static_cast<unsigned char>(str2[i]));
}
}
return 0;
}

bool my_isdigit(char ch)
{
return (ch >= '0' && ch <= '9');
}

char *my_strncat(char *dest, const char *src, size_t n)
{
char *ptr = dest;
while (*ptr)
++ptr;

size_t i = 0;
while (i < n && *src)
{
*ptr++ = *src++;
++i;
}
*ptr = '\\0';

return dest;
}

bool parseDate(const char *data, int *day, int *month, int *year)
{
if (data == nullptr || data[0] == '\\0')
{
return false;
}

int values[3] = {0};
int valueIndex = 0;
int pos = 0;
int length = my_strlen(data);

while (pos < length && valueIndex < 3)
{
while (pos < length && (data[pos] == ' ' || data[pos] == '.' ||
data[pos] == '/' || data[pos] == '-'))
{
pos++;
}
}
```

Продолжение приложения А

```
if (pos >= length)
{
break;
}

int start = pos;
int number = 0;

while (pos < length && my_isdigit(data[pos]))
{
number = number * 10 + (data[pos] - '0');
pos++;
}

if (start == pos)
{
return false;
}

values[valueIndex++] = number;

if (pos < length && data[pos] != ' ' && data[pos] != '.' && data[pos]
!= '/' && data[pos] != '-')
{
return false;
}

if (valueIndex != 3 || pos != length)
{
return false;
}

if (values[0] < 1 || values[0] > 31 ||
values[1] < 1 || values[1] > 12 ||
values[2] < 1 || values[1] > 2025)
{
return false;
}

*day = values[0];
*month = values[1];
*year = values[2];

return true;
}
```

Продолжение приложения А

checker.h

```
#ifndef CHECKER_H
#define CHECKER_H

#include <cstddef>

bool checkFileOpen();

bool checkFileEmpty();

short checkChoice();

bool isValidDate(char *data);

template <typename T>
T checkPositiveNum();

bool checkPhoneNum(char *output, int maxLength);

#endif
```

checker.cpp

```
#include <type_traits>
#include <iostream>
#include <fstream>
#include "charUtils.h"
#include "client.h"

using namespace std;

bool checkFileOpen()
{
    ifstream file("clients.bin", ios::binary);
    if (!file)
    {
        cout << "Ошибка доступа к файлу, возможно, он не существует\n";
        return false;
    }
    else
        return true;
}

bool checkFileEmpty()
{
    ifstream file("clients.bin", ios::binary);
    file.seekg(0, ios::end);
    int count = file.tellg() / sizeof(Client);
    if (count == 0)
```

Продолжение приложения А

```
{
    cout << "Файл пуст\n";
    return false;
} else
    return true;
}

short checkChoice()
{
    char *end;
    long choice;
    bool valid;

    do
    {
        char input[256];
        cin.getline(input, 256);

        valid = true;
        if (my_strlen(input) == 0)
        {
            cout << "Ошибка: введите число от 0 до 15." << endl;
            valid = false;
            continue;
        }

        choice = strtol(input, &end, 10);

        if (end == input || *end != '\0')
        {
            cout << "Ошибка: введите число от 0 до 15." << endl;
            valid = false;
        }
        else if (choice < 0 || choice > 15)
        {
            cout << "Ошибка: введите число от 0 до 15." << endl;
            valid = false;
        }

    } while (!valid);

    return static_cast<short>(choice);
}

template <typename T>
T checkPositiveNum()
{
    char input[256];
    char *end;
    T value;
    bool valid;
```

Продолжение приложения А

```
do
{
    valid = true;
    cin.getline(input, 256);

    if constexpr (is_same_v<T, double>)
    {
        for (char *c = input; *c; ++c)
        {
            if (*c == ',')
                *c = '.';
        }
    }

    if (my_strlen(input) == 0)
    {
        cout << "Ошибка: поле не может быть пустым." << endl;
        valid = false;
        continue;
    }

    if constexpr (is_same_v<T, int>)
    {
        value = strtol(input, &end, 10);
    }
    else
    {
        value = strtod(input, &end);
    }

    if (end == input || *end != '\0')
    {
        cout << "Ошибка: введите корректное число." << endl;
        valid = false;
    }
    else if (value <= 0)
    {
        cout << "Ошибка: число должно быть больше нуля." <<
endl;
        valid = false;
    }

} while (!valid);

return value;
}
template int checkPositiveNum<int>();
template double checkPositiveNum<double>();

bool checkPhoneNum(char *output, int maxLength)
{

```

Продолжение приложения А

```
const size_t MAX_LENGTH = maxLength - 1;
char input[256];
bool valid;

do
{
    valid = true;
    cin.getline(input, 256);

    if (my_strlen(input) == 0)
    {
        cout << "Ошибка: введите номер телефона." << endl;
        valid = false;
        continue;
    }

    if (my_strlen(input) > MAX_LENGTH)
    {
        cout << "Ошибка: номер не должен превышать " <<
MAX_LENGTH << " символов." << endl;
        valid = false;
        continue;
    }

    if (input[0] == '+')
    {
        for (int i = 1; input[i]; i++)
        {
            if (!isdigit(input[i]))
            {
                cout << "Ошибка: после '+' допускаются только
цифры." << endl;
                valid = false;
                break;
            }
        }
    }
    else
    {
        for (int i = 0; input[i]; i++)
        {
            if (!isdigit(input[i]))
            {
                cout << "Ошибка: введите только цифры или '+' в
начале." << endl;
                valid = false;
                break;
            }
        }
    }
}
```

Продолжение приложения А

```
if (!valid)
    continue;

    my_strncpy(output, input, maxLength);
    output[MAX_LENGTH] = '\0';

} while (!valid);

return true;
}

bool isValidDate(char *dateStr)
{
    char temp[11];
    int day = 0, month = 0, year = 0;
    int index = 0, numCount = 0;

    // заменяем разделители на пробел
    for (int i = 0; dateStr[i] && i < 10; i++)
    {
        if (my_isdigit(dateStr[i]))
        {
            temp[index++] = dateStr[i];
        }
        else if (dateStr[i] == '.' || dateStr[i] == '-' ||
dateStr[i] == ' ' || dateStr[i] == '/')
        {
            temp[index++] = ' ';
        }
    }
    temp[index] = '\0';

    for (int i = 0; temp[i]; i++)
    {
        if (my_isdigit(temp[i]))
        {
            int num = 0;
            while (temp[i] && my_isdigit(temp[i]))
            {
                num = num * 10 + (temp[i++] - '0'); // преобразуем
СИМВОЛЫ В ЧИСЛО
            }
            if (numCount == 0) // Первое число — день
                day = num;
            else if (numCount == 1)
                month = num;
            else if (numCount == 2)
                year = num;
            numCount++;
        }
    }
}
```

Продолжение приложения А

```
        if (numCount != 3)
            return false;
        if (year < 1 || month < 1 || month > 12 || day < 1 || year >
2025)
            return false;

        bool leap = (year % 4 == 0 && year % 100 != 0) || (year % 400
== 0); // проверка на високосный год
        if (month == 2)
        {
            if ((leap && day > 29) || (!leap && day > 28))
                return false;
        }
        else if (month == 4 || month == 6 || month == 9 || month == 11)
        {
            if (day > 30)
                return false;
        }
        else
        {
            if (day > 31)
                return false;
        }

        return true;
    }
}
```

client.h

```
#ifndef CLIENT_H
#define CLIENT_H

struct Client
{
    char name[100];
    char phone[20];
    char product[100];
    int day;
    int month;
    int year;
    int quantity;
    double amount;
};

#endif
```

clientService.h

```
#ifndef CLIENTSERVICE_H
#define CLIENTSERVICE_H
```


Продолжение приложения А

```
#include "charUtils.h"
#include "client.h"
#include "sortAlgorithm.h"
#include <fstream>

void printClient(const Client &client);

void editClient();

void deleteClient();

void sortClientsByProduct();

void sortClientsByData();

void sortClientsByName();

void generateRandomClient();

void setDate(Client &client);

void addClient();

void getCountClient();

#endif
```

clientService.cpp

```
#include <fstream>
#include <iostream>
#include "charUtils.h"
#include "clientService.h"
#include "client.h"
#include "sortAlgorithm.h"
#include "fileManager.h"
#include "checker.h"

using namespace std;

void printClient(const Client &client)
{
    cout << "ФИО: " << client.name << "\n"
         << "Телефон: " << client.phone << "\n"
         << "Товар: " << client.product << "\n"
         << "Дата: " << client.day << "." << client.month << "." <<
client.year << "\n"
         << "Количество: " << client.quantity << "\n"
         << "Сумма: " << client.amount << "\n\n";
}
```

Продолжение приложения А

```
void addClient()
{
    Client newClient;
    cout << "Введите ФИО: ";
    cin.getline(newClient.name, 100);

    cout << "Телефон: ";
    checkPhoneNum(newClient.phone, 20);

    cout << "Товар: ";
    cin.getline(newClient.product, 100);

    cout << "Дата (используйте . / -): ";
    setDate(newClient);

    cout << "Количество: ";
    newClient.quantity = checkPositiveNum<int>();

    cout << "Сумма: ";
    newClient.amount = checkPositiveNum<double>();

    saveToBinaryFile(newClient);
    cout << "Клиент добавлен!\n";
}

void setDate(Client &client)
{
    char inputData[256];
    bool valid = false;

    while (!valid)
    {
        cin.getline(inputData, 256);

        if (isValidDate(inputData))
        {
            if (parseDate(inputData, &client.day, &client.month, &client.year))
            {
                valid = true;
            }
            else
            {
                cout << "Ошибка парсинга даты, повторите ввод" <<
endl;
            }
        }
        else
        {
            cout << "Дата не корректна, повторите ввод.\n";
        }
    }
}
```

Продолжение приложения А

```
    }
    }
}

void editClient()
{
    if (checkFileEmpty() && checkFileOpen())
    {
        fstream file("clients.bin", ios::in | ios::out |
ios::binary);
        char searchName[100];
        cout << "Введите ФИО клиента для редактирования: ";
        cin.getline(searchName, 100);

        Client client;
        streampos pos;
        bool found = false;

        while (file.read(reinterpret_cast<char*>(&client),
sizeof(Client)))
        {
            if (my_strcmp(client.name, searchName) == 0)
            {
                pos = file.tellg() -
static_cast<streampos>(sizeof(Client)); // получаем позицию (конец
прочтения - размер объекта)
                found = true;
                break;
            }
        }

        if (!found)
        {
            cout << "Клиент не найден\n";
            file.close();
            return;
        }

        int editChoice;
        cout << "Выберите поле для редактирования:\n"
            << "1. Телефон\n"
            << "2. Товар\n"
            << "3. Дата\n"
            << "4. Количество\n"
            << "5. Сумма\n"
            << "Ваш выбор: ";
        cin >> editChoice;
        cin.ignore();

        switch (editChoice)
```

Продолжение приложения А

```
{
    case 1:
        cout << "Новый телефон: ";
        checkPhoneNum(client.phone, 20);
        break;
    case 2:
        cout << "Новый товар: ";
        cin.getline(client.product, 100);
        break;
    case 3:
        cout << "Дата (используйте . / -): ";
        setDate(client);
        break;
    case 4:
        cout << "Количество: ";
        client.quantity = checkPositiveNum<int>();
        break;
    case 5:
        cout << "Сумма: ";
        client.amount = checkPositiveNum<double>();
        break;
    default:
        cout << "Неверный выбор\n";
        file.close();
        return;
}

file.seekp(pos); //
перемещаемся в начало client
file.write(reinterpret_cast<const char*>(&client),
sizeof(Client)); // переписываем весь объект, с учетом изменённого поля
file.close();
cout << "Данные успешно обновлены!\n";
}
else
    return;
}

void deleteClient()
{
    if (checkFileEmpty() && checkFileOpen())
    {
        char deleteName[100];
        cout << "Введите ФИО клиента для удаления: ";
        cin.getline(deleteName, 100);

        ifstream inFile("clients.bin", ios::binary); // Исходный
        файл (для чтения)
        ofstream outFile("temp.bin", ios::binary); // Временный
        файл (для записи)
```

Продолжение приложения А

```
if (!inFile || !outFile)
{
    cout << "Ошибка при удалении\n";
    return;
}

Client client;
bool found = false;

while (inFile.read(reinterpret_cast<char*>(&client),
sizeof(Client)))
{
    if (my_strcmp(client.name, deleteName) != 0)
    {
        outFile.write(reinterpret_cast<const char*>(&client), sizeof(Client));
    }
    else
    {
        found = true;
    }
}

inFile.close();
outFile.close();

if (found)
{
    remove("clients.bin"); // Удаляем исходный
    rename("temp.bin", "clients.bin"); // Переименовываем
    cout << "Запись успешно удалена\n";
}
else
{
    remove("temp.bin"); // Удаляем временный файл, если клиент не
    cout << "Клиент не найден\n";
}
else
    return;
}

void sortClientsByProduct()
{
    if (checkFileEmpty() && checkFileOpen())
    {
```

Продолжение приложения А

```
fstream file("clients.bin", ios::in | ios::out | ios::binary);
file.seekg(0, ios::end);
int count = file.tellg() / sizeof(Client);

Client *clients = new Client[count];
file.seekg(0);
file.read(reinterpret_cast<char *>(clients), count *
sizeof(Client));
file.close();

sortQuickProduct(clients, 0, count - 1);

file.open("clients.bin", ios::out | ios::binary |
ios::trunc); // открываем для записи, если что-то есть - то удаляем
file.write(reinterpret_cast<const char *>(clients), count *
sizeof(Client));
file.close();

delete[] clients;
cout << "Сортировка выполнена успешно\n";
}
else
return;
}

void sortClientsByData()
{
if (checkFileEmpty() && checkFileOpen())
{

fstream file("clients.bin", ios::in | ios::out |
ios::binary);
file.seekg(0, ios::end);
int count = file.tellg() / sizeof(Client);

Client *clients = new Client[count];
file.seekg(0);
file.read(reinterpret_cast<char *>(clients), count *
sizeof(Client));
file.close();

sortSelection(clients, count);

file.open("clients.bin", ios::out | ios::binary |
ios::trunc); // открываем для записи, если что-то есть - то удаляем
file.write(reinterpret_cast<const char *>(clients), count *
sizeof(Client));
file.close();

delete[] clients;
cout << "Сортировка выполнена успешно\n";
```

Продолжение приложения А

```
    }
    else
        return;
}

void sortClientsByName()
{
    if (checkFileEmpty() && checkFileOpen())
    {
        fstream file("clients.bin", ios::in | ios::out |
ios::binary);
        file.seekg(0, ios::end);
        int count = file.tellg() / sizeof(Client);

        Client *clients = new Client[count];
        file.seekg(0);
        file.read(reinterpret_cast<char *>(clients), count *
sizeof(Client));
        file.close();

        sortInsertion(clients, count);

        file.open("clients.bin", ios::out | ios::binary |
ios::trunc); // открываем для записи, если что-то есть - то удаляем
        file.write(reinterpret_cast<const char *>(clients), count *
sizeof(Client));
        file.close();

        delete[] clients;
        cout << "Сортировка выполнена успешно\n";
    }
    else
        return;
}

void getCountClient() {
    if (checkFileOpen())
    {
        fstream file("clients.bin", ios::in | ios::out |
ios::binary);
        file.seekg(0, ios::end);
        int count = file.tellg() / sizeof(Client);

        cout << "Всего клиентов: " << count << endl;
    } else
        return;
}

void generateRandomClient()
{

```

Продолжение приложения

```
int count;
cout << "Введите количество клиентов для генерации: ";
cin >> count;
cin.ignore();

const char *maleFirstNames[] = {"Иван", "Петр", "Дмитрий"};
const char *femaleFirstNames[] = {"Мария", "Анна"};
const char *maleLastNames[] = {"Иванов", "Петров", "Смирнов",
"Кузнецов", "Васильев"};
const char *femaleLastNames[] = {"Иванова", "Петрова",
"Смирнова", "Кузнецова", "Васильева"};
const char *malePatronymics[] = {"Иванович", "Петрович",
"Сергеевич"};
const char *femalePatronymics[] = {"Алексеевна", "Дмитриевна",
"Ивановна"};
const char *products[] = {"Ноутбук", "Смартфон", "Телевизор",
"Холодильник", "Пылесос"};
for (int i = 0; i < count; i++)
{
    Client newClient;

    bool isMale = rand() % 2;

    const char *firstName;
    const char *lastName;
    const char *patronymic;

    if (isMale)
    {
        firstName = maleFirstNames[rand() % 3];
        lastName = maleLastNames[rand() % 5];
        patronymic = malePatronymics[rand() % 3];
    }
    else
    {
        firstName = femaleFirstNames[rand() % 2];
        lastName = femaleLastNames[rand() % 5];
        patronymic = femalePatronymics[rand() % 3];
    }

    char fullName[100] = "";
    my_strncat(fullName, lastName, 99);
    my_strncat(fullName, " ", 99 - my_strlen(fullName));
    my_strncat(fullName, firstName, 99 - my_strlen(fullName));
    my_strncat(fullName, " ", 99 - my_strlen(fullName));
    my_strncat(fullName, patronymic, 99 - my_strlen(fullName));

    my_strncpy(newClient.name, fullName, 99);
    newClient.name[99] = '\0';

    char phone[20] = "9";
```


Продолжение приложения А

```
for (int i = 0; i < 9; i++)
{
    char digit[2] = {static_cast<char>('0' + rand() % 10),
'\0'};

    my_strncat(phone, digit, 19 - my_strlen(phone));
}
my_strncpy(newClient.phone, phone, 19);
newClient.phone[19] = '\0';

my_strncpy(newClient.product, products[rand() % 5], 99);
newClient.product[99] = '\0';

newClient.day = 1 + rand() % 31;
newClient.month = 1 + rand() % 12;
newClient.year = 2000 + rand() % 24;

newClient.quantity = 1 + rand() % 100;
newClient.amount = 100.0 + (rand() % 990000) / 100.0;

saveToBinaryFile(newClient);
}
cout << "Сгенерировано " << count << " клиентов!\n";
}
```

fileManager.h

```
#ifndef FILEMANAGER_H
#define FILEMANAGER_H

#include "client.h"

void saveToBinaryFile(const Client &client);

void displayBinaryFile();

void removeClients();

#endif
```

fileManager.cpp

```
#include <iostream>
#include <fstream>

#include "fileManager.h"
#include "clientService.h"
#include "checker.h"

using namespace std;
```

Продолжение приложения А

```
void saveToBinaryFile(const Client &client)
{
    ofstream file("clients.bin", ios::binary | ios::app); // app -
    открываем файл в режиме добавления, если нет файлы - создаем
    if (file)
    {
        file.write(reinterpret_cast<const char*>(&client),
        sizeof(Client)); // преобразуем указатель и записываем столько байтов,
        сколько занимает объект
        file.close();
    }
}

void displayBinaryFile()
{
    if (checkFileEmpty() && checkFileOpen())
    {
        ifstream file("clients.bin", ios::binary);

        Client client;
        while (file.read(reinterpret_cast<char*>(&client),
        sizeof(Client)))
        {
            printClient(client);
        }

        file.close();
    }
    else
        return;
}

void removeClients()
{
    if (checkFileOpen() && checkFileEmpty())
    {
        ofstream file("clients.bin", ios::binary | ios::trunc); //
        очищаем содержимое файла
        if (file)
        {
            cout << "Все записи клиентов были удалены.\n";
            file.close();
        }
    }
}
```

generateReport.h

```
#ifndef GENERATEREPORT_H
#define GENERATEREPORT_H
```

```
void generateReport();
```

```
#endif
```

generateReport.cpp

```
#include <iostream>
```

```
#include <fstream>
```

```
#include "client.h"
```

```
#include "generateReport.h"
```

```
#include "sortAlgorithm.h"
```

```
#include "charUtils.h"
```

```
#include "checker.h"
```

```
using namespace std;
```

```
void generateReport()
```

```
{
```

```
    if (checkFileOpen())
```

```
    {
```

```
        ifstream inFile("clients.bin", ios::binary);
```

```
        inFile.seekg(0, ios::end);
```

```
        int count = inFile.tellg() / sizeof(Client);
```

```
        if (count == 0)
```

```
        {
```

```
            cout << "Нет данных для отчета\n";
```

```
            return;
```

```
        }
```

```
        Client *clients = new Client[count];
```

```
        inFile.seekg(0);
```

```
        inFile.read(reinterpret_cast<char *>(clients), count *  
sizeof(Client));
```

```
        inFile.close();
```

```
        ofstream outFile("report.txt");
```

```
        if (!outFile)
```

```
        {
```

```
            cout << "Ошибка создания отчета\n";
```

```
            delete[] clients;
```

```
            return;
```

```
        }
```

```
        cout << "\n=== ОТЧЕТ ===\n";
```

```
        outFile << "=== ОТЧЕТ ===\n\n";
```

```
        // сортируем по названию
```

```
        sortQuickProduct(clients, 0, count - 1);
```

Продолжение приложения А

```
// если один товар - сортируем по date
int startGroup = 0;
for (int i = 1; i <= count; ++i)
{
    if (i == count || my_strcmp(clients[i].product,
clients[startGroup].product) != 0)
    {
        sortSelection(clients + startGroup, i -
startGroup);
        startGroup = i;
    }
}

const char *currentProduct = clients[0].product;
outFile << "Товар: " << currentProduct << "\n";
cout << "Товар: " << currentProduct << "\n";

for (int i = 0; i < count; ++i)
{
    if (my_strcmp(clients[i].product, currentProduct) !=
0)
    {
        currentProduct = clients[i].product;
        outFile << "\nТовар: " << currentProduct << "\n";
        cout << "\nТовар: " << currentProduct << "\n";
    }
    outFile << "  " << clients[i].day << "." <<
clients[i].month << "." << clients[i].year
        << " - " << clients[i].name << " (" <<
clients[i].quantity << " шт.)\n";

    cout << "  " << clients[i].day << "." << clients[i].month << "."
<< clients[i].year
        << " - " << clients[i].name << " (" <<
clients[i].quantity << " шт.)\n";
}

int maxQuantity = 0;
const char *popularProduct = "";
for (int i = 0; i < count;)
{
    int total = 0; // общее количество покупок
    const char *product = clients[i].product;
    while (i < count && my_strcmp(clients[i].product,
product) == 0)
    {
        total += clients[i].quantity;
        ++i;
    }
    if (total > maxQuantity)
    {
```

Продолжение приложения А

```
        maxQuantity = total;
        popularProduct = product;
    }
}
outFile << "\nСамый популярный товар: " << popularProduct
<< " (" << maxQuantity << " шт.)\n";
cout << "\nСамый популярный товар: " << popularProduct <<
" (" << maxQuantity << " шт.)\n";

struct ClientTotal
{
    char name[100];
    double total;
};

ClientTotal *totals = new ClientTotal[count]; // Храним
имя и сумму покупок
int totalCount = 0;

for (int i = 0; i < count; ++i)
{
    bool found = false;
    for (int j = 0; j < totalCount; ++j)
    {
        if (my_strcmp(clients[i].name, totals[j].name) ==
0)
        {
            totals[j].total += clients[i].amount; //
увеличиваем сумму для клиента
            found = true;
            break;
        }
    }
    if (!found)
    {
        my_strcpy(totals[totalCount].name,
clients[i].name); // копируем имя
        totals[totalCount].total = clients[i].amount;
// копируем сумму
        ++totalCount;
    }
}

double maxTotal = 0;
const char *activeClient = "";
for (int i = 0; i < totalCount; ++i)
{
    if (totals[i].total > maxTotal)
    {
        maxTotal = totals[i].total;
        activeClient = totals[i].name;
    }
}
```

Продолжение приложения А

```
    }
    }
    outFile << "Самый активный клиент: " << activeClient << "
(Сумма: " << maxTotal << ")\n";
    cout << "Самый активный клиент: " << activeClient << "
(Сумма: " << maxTotal << ")\n";

    delete[] clients;
    delete[] totals;
    outFile.close();
    cout << "\nОтчет сохранен в report.txt\n";
}
else
    return;
}
```

searchAlgorithm.h

```
#ifndef SEARCHALGORITHM_H
#define SEARCHALGORITHM_H

#include "client.h"

void searchLinearByName();

void searchBinaryByName();

void searchByProductAndDate();
void searchLinearByProduct();
#endif
```

searchAlgorithm.cpp

```
#include <iostream>
#include <fstream>

#include "charUtils.h"
#include "client.h"
#include "clientService.h"
#include "searchAlgorithm.h"
#include "checker.h"

using namespace std;

void searchLinearByName()
{
    if (checkFileEmpty() && checkFileOpen())
    {

        char searchName[100];
```

Продолжение приложения А

```
cout << "Введите ФИО для поиска: ";
cin.getline(searchName, 100);

Client client;
bool found = false;
ifstream file("clients.bin", ios::binary);
while (file.read(reinterpret_cast<char *>(&client),
sizeof(Client)))
{
    if (my_strcmp(client.name, searchName) == 0)
    {
        printClient(client);
        found = true;
    }
}
file.close();

if (!found)
    cout << "Клиент не найден\n";
else
    return;
}

void searchLinearByProduct()
{
    if (checkFileEmpty() && checkFileOpen())
    {

        ifstream file("clients.bin", ios::binary);
        char searchProduct[100];
        cout << "Введите наименование товара: ";

        cin.getline(searchProduct, 100);

        Client client;
        bool found = false;
        while (file.read(reinterpret_cast<char *>(&client),
sizeof(Client)))
        {
            if (my_strcmp(client.product, searchProduct) == 0)
            {
                printClient(client);
                found = true;
            }
        }
        file.close();

        if (!found)
            cout << "Продукт не найден\n";
    }
}
```

Продолжение приложения А

```
else
    return;
}

void searchBinaryByName()
{
    if (checkFileEmpty() && checkFileOpen())
    {
        ifstream file("clients.bin", ios::binary);
        file.seekg(0, ios::end);
        int count = file.tellg() / sizeof(Client);
        file.seekg(0);
        Client *clients = new Client[count];

        file.seekg(0);
        file.read(reinterpret_cast<char *>(clients), count *
sizeof(Client));
        file.close();

        sortClientsByProduct(); // сортировка перед бинарной
char searchProduct[100];
        cout << "Введите наименование товара: ";
        cin.getline(searchProduct, 100);

        int left = 0, right = count - 1;
        bool found = false;
        while (left <= right)
        {
            int mid = left + (right - left) / 2;
            int cmp = my_strncmp(clients[mid].product,
searchProduct);

            if (cmp == 0)
            {
                int i = mid;
                while (i >= 0 && my_strncmp(clients[i].product,
searchProduct) == 0) // для того чтобы вывести все совпадения
                    --i;
                ++i;
                while (i < count && my_strncmp(clients[i].product,
searchProduct) == 0)
                {
                    printClient(clients[i]);
                    ++i;
                    found = true;
                }
                break;
            }
            if (cmp < 0)
                left = mid + 1;
        }
    }
}
```


Продолжение приложения А

```
else
    right = mid - 1;
}

delete[] clients;
if (!found)
    cout << "Товар не найден\n";
}
else
    return;
}

void searchByProductAndDate()
{
    if (checkFileEmpty() && checkFileOpen())
    {

        ifstream file("clients.bin", ios::binary);

        char searchProduct[100];
        int day, month, year;

        cout << "Введите наименование товара: ";
        cin.getline(searchProduct, 100);

        bool valid = false;
        char data[256];

        while (!valid)
        {
            cout << "Введите дату(используйте . / -): ";
            cin.getline(data, 256);

            valid = isValidDate(data);
            if (!valid)
                cout << "Дата не корректна, повторите ввод. \n";
        }
        parseDate(data, &day, &month, &year);

        Client *temp = new Client[1000];
        int count = 0;

        Client client;
        while (file.read(reinterpret_cast<char *>(&client),
sizeof(Client)))
        {
            if (my_strncmp(client.product, searchProduct) == 0 &&
                (client.year > year ||
                 (client.year == year && client.month > month) ||
                 (client.year == year && client.month == month &&
client.day > day)))
```

Продолжение приложения А

```
{
    temp[count++] = client;
}
}
file.close();

if (count == 0)
{
    cout << "Покупки не найдены\n";
    delete[] temp;
    return;
}
// сортируем по имени, методом вставок
for (int i = 1; i < count; ++i)
{
    Client key = temp[i];
    int j = i - 1;
    while (j >= 0 && my_strcmp(temp[j].name, key.name) >
0)
    {
        temp[j + 1] = temp[j];
        --j;
    }
    temp[j + 1] = key;
}

for (int i = 0; i < count; ++i)
{
    printClient(temp[i]);
}

delete[] temp;
}
else
    return;
}
```

sortAlgorithnm.h

```
#ifndef SORTALGORITHM_H
#define SORTALGORITHM_H
#include "client.h"

void sortQuickProduct(Client arr[], int left, int right);

int compareDates(Client a, Client b);

void sortSelection(Client arr[], int size);

void sortInsertion(Client arr[], int size);
```

Продолжение приложения А

```
template <typename T>
void my_swap(T &a, T &b);

#endif
```

sortAlgorithm.cpp

```
#include "sortAlgorithm.h"
#include "charUtils.h"
void sortQuickProduct(Client arr[], int left, int right)
{
    if (left >= right)
        return;
    int i = left, j = right;
    Client pivot = arr[left + (right - left) / 2];
    while (i <= j)
    {
        while (my_strcmp(arr[i].product, pivot.product) < 0)
            i++;
        while (my_strcmp(arr[j].product, pivot.product) > 0)
            j--;
        if (i <= j)
        {
            my_swap(arr[i], arr[j]);
            i++;
            j--;
        }
    }
    sortQuickProduct(arr, left, j);
    sortQuickProduct(arr, i, right);
}

int compareDates(Client a, Client b)
{
    if (a.year != b.year)
        return a.year - b.year;
    if (a.month != b.month)
        return a.month - b.month;
    return a.day - b.day;
}

void sortSelection(Client arr[], int size)
{
    for (int i = 0; i < size - 1; ++i)
    {
        int minDateIndex = i;
        for (int j = i + 1; j < size; ++j)
        {
            if (compareDates(arr[j], arr[minDateIndex]) < 0)
            {
                minDateIndex = j;
            }
        }
    }
}
```

Продолжение приложения А

```
    }
    }
    if (minDateIndex != i)
    {
        my_swap(arr[i], arr[minDateIndex]);
    }
}

void sortInsertion(Client arr[], int size)
{
    for (int i = 1; i < size; ++i)
    {
        Client key = arr[i];
        int j = i - 1;
        while (j >= 0 && my_strcmp(arr[j].name, key.name) > 0)
        {
            arr[j + 1] = arr[j];
            --j;
        }
        arr[j + 1] = key;
    }
}

template <typename T>
void my_swap(T &a, T &b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

ПРИЛОЖЕНИЕ Б **(обязательное)** **Блок-схема работы программы**

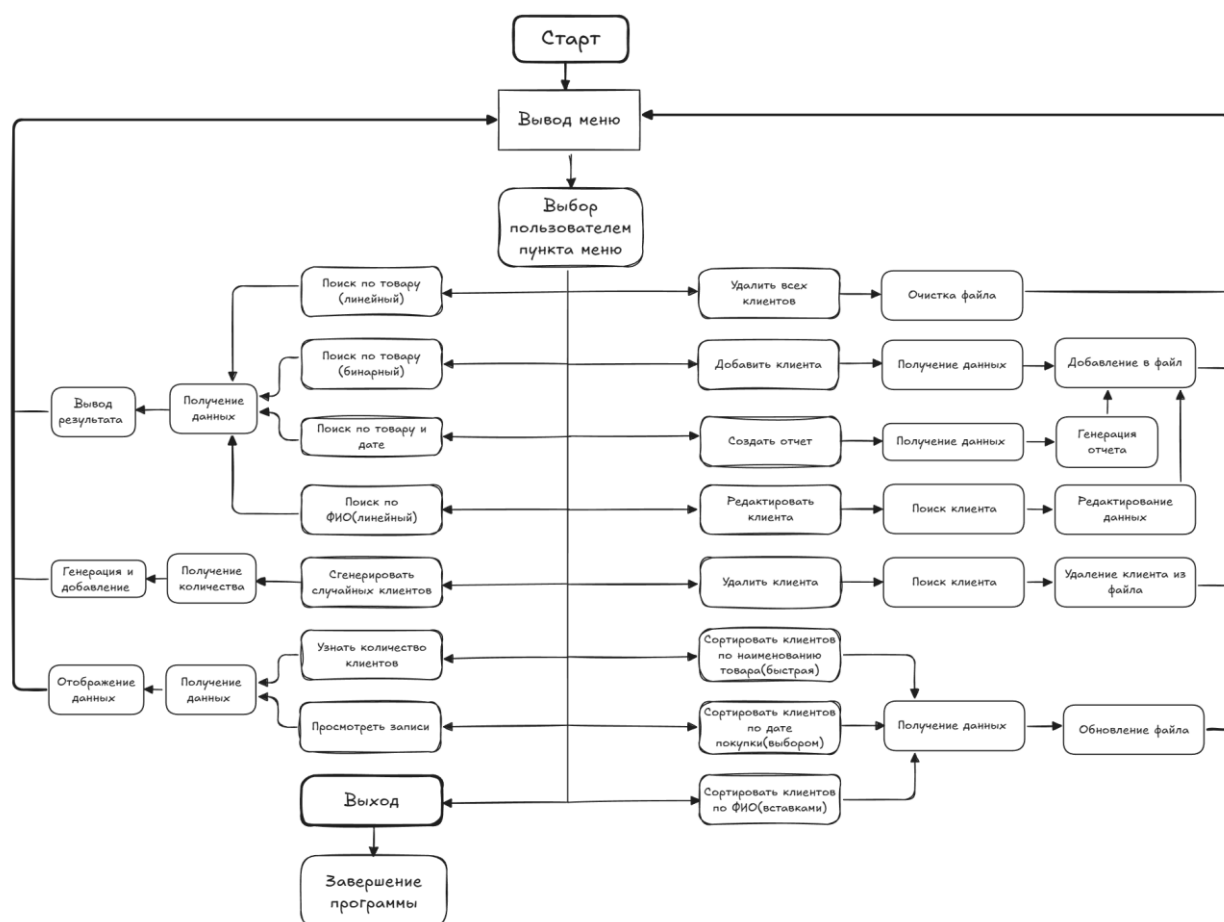


Рисунок Б.1 – Блок-схема работы программы

ВЕДОМОСТЬ ДОКУМЕНТОВ

Обозначение					Наименование					Дополнительные сведения		
					<u>Текстовые документы</u>							
БГУИР КП 6-05-0611-03 083 ПЗ					Пояснительная записка					62 с.		