Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

О Т Ч Е Т

ПРОИЗВОДСТВЕННАЯ ПРАКТИКА

«Эксплуатационная практика»

Студент гр. ИУК4-42Б _____ (Урманова Г.И.)
(подпись) (Ф.И.О.)

Руководитель _____ (Ameличева К.А.)
(подпись) (Ф.И.О.)

Оценка руководителя _____ баллов _____
30-50 (дата)

Оценка защиты _____ баллов _____
30-50 (дата)

Оценка практики _____ баллов _____
(оценка по пятибалльной шкале)

Комиссия: _____ (Гагарин Ю.Е.)
(подпись) (Ф.И.О.)

_____ (Пчелинцева Н.И.)
(подпись) (Ф.И.О.)

_____ (Ameличева К.А.)
(подпись) (Ф.И.О.)

Калуга, 2024

Калужский филиал федерального государственного бюджетного образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой **__ИУК4___**
_____(Гагарин Ю.Е.)
«01» июля 2024 г.

З А Д А Н И Е
на ПРОИЗВОДСТВЕННУЮ ПРАКТИКУ, Эксплуатационную

За время прохождения практики студенту необходимо:

1. Определить предметную область, цель, задачи и основные результаты прохождения практики.
2. Выбирать необходимый математический аппарат для реализации задания, выбирать интегрированную среду разработки программного обеспечения, ознакомиться с технологиями создания графического интерфейса в реализуемом программном обеспечении; подобрать стандартные библиотеки; спроектировать компоненты программного продукта.
3. Разработать тестовое окружение, создать тестовые сценарии.
4. Разработать программу валидации входящих строк.
5. Подготовить отчет и защитить результаты практики.

Дата выдачи задания «01» июля 2024 г.

Руководитель практики _____ Амеличева К.А.

Задание получил _____ Урманова Г.И.

Оглавление

ВВЕДЕНИЕ

Целью практики является сформировать первичные умения и навыки:

- выбора предметной области на основе имеющихся знаний в сфере разработки программного обеспечения, умение грамотно сформулировать тему проекта, область его применения и главные отличия относительно уже имеющихся проектов;
- по подбору и оценке технологий разработки и вспомогательного программного обеспечения для реализации итогового проекта в соответствии с выбранной темой;
- оценки функциональности проекта, которая будет реализована к соответствующему сроку, умение грамотно распределить роли и задачи внутри команды; по представлению и оценке качества разработанного приложения.

Для достижения поставленной цели решаются следующие задачи:

- осуществить выбор языка(-ов) программирования, фреймворков и библиотек, среду(-ы) разработки;
- подготовить доклад с использованием инструментов визуализации в котором будет представлен отчёт о проделанной работе, общая информация о разработанном проекте, его преимущества, недостатки и перспективы дальнейшего развития.

1. ОБЩИЕ СВЕДЕНИЯ

1.1Исследование предметной области задачи и постановка задачи

Предметная область:

Валидация данных – это процесс проверки информации на соответствие определенным правилам и стандартам. Представьте, что вы заполняете онлайн-форму. Валидация проверяет, правильно ли вы ввели свое имя (только буквы), дату рождения (существует ли такая дата) и номер телефона (соответствует ли формату).

Важность валидации:

Качество данных: Валидация обеспечивает высокое качество данных, что является основой для принятия обоснованных решений. Неверные данные могут привести к ошибочным выводам и серьезным последствиям.

Безопасность**:** Валидация данных помогает предотвратить кибератаки, такие как SQL-инъекции и XSS. Проверка вводимых данных на соответствие определенным шаблонам помогает защитить системы от вредоносного кода.

Производительность систем**:** Корректные данные позволяют системам работать более эффективно. Например, оптимизированные запросы к базе данных, основанные на валидных данных, ускоряют обработку информации.

Соответствие стандартам: Многие отрасли имеют свои стандарты и требования к качеству данных. Валидация помогает обеспечить соответствие этим стандартам.

Пользовательский опыт: Валидация делает взаимодействие с приложениями более удобным. Сообщения об ошибках при неправильном вводе данных помогают пользователям избежать ошибок и быстрее достичь цели.

Регулярные выражения — это мощный инструмент для валидации данных в C++. Они позволяют создавать гибкие и эффективные проверки на соответствие заданным шаблонам. Однако, важно помнить о сложности синтаксиса и потенциальных проблемах с производительностью.

Регулярные выражения – это последовательности символов, которые описывают определенный поиск. Они используются для поиска и манипулирования текстом. Например, с помощью регулярных выражений можно:

Проверить, соответствует ли строка определенному формату (например, email, номер телефона).

Найти все вхождения определенного слова или фразы в тексте.

Заменить все вхождения подстроки на другую.

Разбить текст на отдельные части по определенным правилам.

Постановка задачи:

1. Реализовать функции приема, валидации данных.
2. Функция валидации должна содержать в себе – регулярные выражения.
3. Разработать алгоритм работы функций, добавить возможность считывания данных с файлов.

Основная цель – реализовать программу, для валидации номера рейса и кода авиакомпаний из данных в файле, с целью отобрать необходимую часть данных.

1.2 Обоснование выбора средств реализации

Для разработки программы был выбран язык C++, библиотека стандартных шаблонов STL.

Они имеют следующие преимущества:

1. Мощность и гибкость

Сложные шаблоны: Регулярные выражения позволяют описывать чрезвычайно сложные и гибкие шаблоны поиска, от простых последовательностей символов до сложных грамматических конструкций.

Универсальность: Подходят для широкого спектра задач, от простой проверки формата данных до сложного синтаксического анализа.

2. Эффективность

Оптимизированные алгоритмы: Стандартная библиотека C++ предоставляет высокоэффективные реализации алгоритмов для работы с регулярными выражениями.

Компиляция: Регулярные выражения компилируются в эффективный машинный код, что обеспечивает высокую скорость выполнения.

3. Интеграция с языком

Стандартная библиотека: Регулярные выражения являются частью стандартной библиотеки C++, что упрощает их использование и интеграцию в различные проекты.

4. Чтение и понимание

Ясный синтаксис: Синтаксис регулярных выражений, хотя и может показаться сложным на первый взгляд, достаточно интуитивен и позволяет создавать читаемый код.

Документация: Существует обширная документация и множество примеров использования регулярных выражений, что облегчает обучение и использование.

5. Повторное использование

Модульность: Регулярные выражения можно легко выносить в отдельные функции или классы, что способствует повторному использованию кода.

Библиотеки регулярных выражений: Существуют специализированные библиотеки, предоставляющие дополнительные возможности и оптимизации для работы с регулярными выражениями.

Примеры использования

Валидация форм: Проверка корректности вводимых данных в веб-формах (email, пароли, номера телефонов).

Парсинг данных: Извлечение информации из текстовых файлов (например, лог-файлов, конфигурационных файлов).

Анализ текстов: Поиск и замена подстрок, разбиение текста на токены.

Сравнение с другими методами валидации

По сравнению с другими методами валидации данных, регулярные выражения обладают большей гибкостью и выразительностью. Они позволяют создавать более сложные и точные правила проверки.

1.3 Актуальность решаемой проблемы и возможные области применения данной разработки

Качество данных: Валидация обеспечивает высокое качество данных, что является основой для принятия обоснованных решений. Неверные данные могут привести к ошибочным выводам и серьезным последствиям.

Безопасность: Валидация данных помогает предотвратить кибератаки, такие как SQL-инъекции и XSS. Проверка вводимых данных на соответствие определенным шаблонам помогает защитить системы от вредоносного кода.

Производительность систем: Корректные данные позволяют системам работать более эффективно. Например, оптимизированные запросы к базе данных, основанные на валидных данных, ускоряют обработку информации.

Соответствие стандартам: Многие отрасли имеют свои стандарты и требования к качеству данных. Валидация помогает обеспечить соответствие этим стандартам.

Пользовательский опыт: Валидация делает взаимодействие с приложениями более удобным. Сообщения об ошибках при неправильном вводе данных помогают пользователям избежать ошибок и быстрее достичь цели.

Сферы применения валидации:

Разработка программного обеспечения: Валидация входных данных, проверка форматов файлов, контроль целостности данных в базах данных.

Анализ данных: Очистка данных, проверка на выбросы, подготовка данных для моделирования.

Электронная коммерция: Проверка корректности адресов, номеров кредитных карт, данных о заказах.

Финансовая сфера: Контроль точности финансовых данных, предотвращение мошенничества.

Научные исследования: Проверка достоверности экспериментальных данных.

Последствия отсутствия валидации:

Ошибочные результаты: Неправильные вычисления, некорректные отчеты.

Сбои в работе системы: Нестабильность приложения, потеря данных.

Уязвимости безопасности: Риск взлома системы, утечки конфиденциальной информации.

Потеря репутации: Недоверие пользователей, снижение конкурентоспособности.

Методы валидации

Проверка типов данных: Убедиться, что данные соответствуют ожидаемому типу (например, число, строка).

Проверка диапазонов: Ограничить значения данных допустимым диапазоном.

Регулярные выражения: Проверять данные на соответствие определенным шаблонам (например, формат email, номера телефона).

Сравнение с эталонными значениями: Сравнивать данные с заранее известными правильными значениями.

Проверка на NULL: Убедиться, что значения не являются пустыми.

1.4 Описание используемой библиотеки regex

Библиотека regex (регулярные выражения) – это мощный инструмент, который широко используется в программировании для поиска, замены и валидации текстовых данных. Она позволяет создавать сложные шаблоны, описывающие структуру искомых данных, и эффективно применять их к большим объемам текста.

Стандартная библиотека C++ предоставляет мощный инструмент для работы с регулярными выражениями — заголовочный файл `<regex>`. Он позволяет эффективно осуществлять поиск, замену и валидацию текстовых данных.

Как библиотека regex используется для валидации данных?

Создание шаблонов: Вы создаете регулярное выражение, которое описывает формат корректных данных.

Сопоставление с данными: Библиотека сравнивает вводимые данные с созданным шаблоном.

Принятие решения: Если данные соответствуют шаблону, они считаются валидными, иначе – нет.

Основные компоненты библиотеки:

std::regex**:** Представляет собой компилированное регулярное выражение. При создании объекта этого класса регулярное выражение преобразуется в внутреннее представление, оптимизированное для поиска.

std::smatch**:** Используется для хранения результатов поиска. Содержит информацию о найденных совпадениях, их позициях и подгруппах.

Функции для поиска и сопоставления:

std::regex_match: Проверяет, совпадает ли вся строка с регулярным выражением.

std::regex_search: Ищет первое подстроку в строке, которая соответствует регулярному выражению.

std::regex_replace: Заменяет все подстроки, соответствующие регулярному выражению, на новую строку.

Особенности регулярных выражений в C++:

Синтаксис: Используется синтаксис регулярных выражений, совместимый с ECMAScript.

Флаги: Позволяют изменять поведение поиска (например, игнорировать регистр).

Подгруппы: Разделяют регулярное выражение на части для извлечения отдельных элементов.

Преимущества использования библиотеки regex для валидации данных

Гибкость: Позволяет создавать сложные и гибкие шаблоны для проверки данных.

Эффективность: Обеспечивает высокую скорость выполнения благодаря оптимизированным алгоритмам.

Стандартизация: Синтаксис регулярных выражений является стандартизированным, что облегчает их использование в разных языках программирования.

Повторное использование: Созданные шаблоны можно легко использовать в разных частях программы.

Интеграция с другими инструментами: Многие текстовые редакторы, IDE и языки программирования поддерживают регулярные выражения.

Типичные задачи, решаемые с помощью regex

Валидация форм: Проверка корректности вводимых данных в веб-формах (email, пароли, номера телефонов).

Парсинг данных: Извлечение информации из текстовых файлов (например, лог-файлов, конфигурационных файлов).

Анализ текстов: Поиск и замена подстрок, разбиение текста на токены.

Создание лексических анализаторов: Разработка языков программирования и компиляторов.

2 ПРОЕКТИРОВАНИЕ КОМПОНЕНТОВ ПРОГРАММНОГО ПРОДУКТА

2.1 Общие сведения о программе

Программа валидации данных – набор функций, используемый для обработки информации в файлах путём считывания и валидации исходных данных.

Как работает программа:

На вход подаются названия файлов, первый из которых с исходной информацией, содержащие код авиакомпании и номер рейса, а второй пустой, куда должны попасть данные, прошедшие валидацию. Данные сохраняются в файле и дается ответ пользователю.

Программа позволяет проверять уникальность и валидность номера рейса и кода авиакомпании, тем самым отсеивая неактуальные или неверные данные в файлах.

Такая программа может быть использована в ряду авиакомпаний с диспетчерскими службами, контролирующие полеты самолетов, для проведения отчета полетов определенных точек.

2.2 Стратегии валидации

Валидация данных — это критически важный этап разработки программного обеспечения, особенно в C++, где ошибки ввода могут привести к неопределенному поведению и даже к краху программы. Существует несколько стратегий для обеспечения корректности данных:

Проверка типов данных

Использование сильных типов: C++ — язык со статической типизацией. Это позволяет компилятору выявлять многие ошибки еще на этапе компиляции.

Ограничение диапазонов: Для числовых типов можно проверять, что значения находятся в допустимом диапазоне.

Проверка на NULL: Убеждаться, что указатели не являются нулевыми перед их использованием.

Регулярные выражения

Библиотека regex: Предоставляет мощные инструменты для проверки строк на соответствие определенным шаблонам.

Примеры: Проверка формата email, номеров телефонов, дат и т.д.

Пользовательские функции валидации

Специализированные функции: Создавать функции, которые проверяют данные на соответствие конкретным требованиям.

Пример: Функция для проверки пароля на сложность.

Исключения

Обработка ошибок: Использовать механизм исключений для сигнализации о некорректных данных.

Пользовательские исключения: Создавать собственные классы исключений для более детальной информации об ошибке.

Ассерты

Отладка: Использовать ассерты для проверки условий, которые должны всегда выполняться.

Пример: Проверка индексов массивов на выход за границы.

Валидация входных данных

Проверка пользовательского ввода: Тщательно проверять все данные, вводимые пользователем.

Фильтрация: Использовать фильтры для удаления недопустимых символов или форматирования данных.

Валидация выходных данных

Проверка результатов вычислений: Убеждаться, что результаты вычислений находятся в разумных пределах.

Сравнение с ожидаемыми значениями: Сравнивать результаты с известными правильными значениями.

Использование библиотек валидации

Специализированные библиотеки: Существуют библиотеки, предоставляющие готовые инструменты для валидации данных.

Валидация перед сохранением: Проверка данных перед записью их в базу данных или файл.

Валидация во время выполнения: Периодическая проверка данных на соответствие условиям.

Читаемость кода: Код валидации должен быть понятным и легко поддерживаемым.

Выбор подходящего модуля валидации зависит от конкретной задачи и требований проекта. Для простых проверок могут быть достаточно встроенные механизмы, а для сложных сценариев может потребоваться

создание пользовательских функций или использование специализированных библиотек.

Стратегии комбинирования методов валидации

Многоуровневая валидация: Комбинировать различные методы для достижения максимальной надежности.

Ранняя валидация: Проверять данные как можно раньше в процессе обработки.

Валидация на разных уровнях: Проверять данные на уровне ввода, обработки и вывода.

2.3Обработка исключений

Обработка исключений является ключевым аспектом при валидации данных в C++. Она позволяет эффективно реагировать на ситуации, когда вводимые данные не соответствуют заданным критериям.

Повышение надежности: Обработка исключений предотвращает аварийное завершение программы и позволяет выполнить альтернативные действия.

Улучшение пользовательского опыта: Можно выводить информативные сообщения об ошибках, помогая пользователю исправить неверные данные.

Локализация ошибок: Точная идентификация места возникновения ошибки и ее причины облегчает отладку.

Обработка может осуществляться следующими способами:

Определение пользовательских исключений: Создайте класс исключения, наследуемый от std::exception.

Добавьте конструктор для передачи сообщения об ошибке.

Переопределите метод what(), чтобы возвращать более подробную информацию об исключении.

Генерация исключений: В функции валидации, если данные не прошли проверку, генерируется исключение.

Обработка исключений: Используйте блок try-catch для перехвата исключений.

Важные моменты:

Иерархия исключений: Создавайте иерархию исключений для более детальной обработки ошибок.

Сообщения об ошибках: Делайте сообщения об ошибках информативными и понятными для пользователя.

Локализация: Размещайте обработку исключений как можно ближе к месту их возникновения.

Использование стандартных исключений: Для некоторых ошибок можно использовать стандартные исключения, такие как std::invalid_argument или std::out_of_range.

ЗАКЛЮЧЕНИЕ

В ходе проведении эксплуатационной практики был разработан проект валидации входной информации. В результате практики были сформированы умения и навыки:

- выбора предметной области на основе имеющихся знаний в сфере разработки программного обеспечения, умение грамотно сформулировать тему проекта, область его применения и главные отличия относительно уже имеющихся проектов;
- по подбору и оценке технологий разработки и вспомогательного программного обеспечения для реализации итогового проекта в соответствии с выбранной темой;
- оценки функциональности проекта, которая будет реализована к соответствующему сроку, умение грамотно распределить роли и задачи внутри команды; по представлению и оценке качества разработанного приложения.

По достижению поставленной цели были решены следующие задачи:

- осуществить выбор языка программирования, фреймворков и библиотек, среду разработки;
- подготовить доклад с использованием инструментов визуализации в котором будет представлен отчёт о проделанной работе, общая информация о разработанном проекте, его преимущества, недостатки и перспективы дальнейшего развития.

Список использованных источников

1. Моделирование информационных ресурсов [Электронный ресурс]: учебно-методический/ Составитель Огнев Э.Н. - Кемерово: Кемеровский государственный университет культуры и искусств, 2013. - 36 с.: ил., табл. - URL: http://biblioclub.ru/index.php?page=book&id=274218
2. Коваленко, Ю.В. Информационно-поисковые системы [Электронный ресурс]: учебно-методическое пособие / Ю.В. Коваленко, Т.А. Сергиенко. — Омск: Омская юридическая академия, 2017. — 38 с.— Режим доступа: http://www.iprbookshop.ru/66817.html
3. Маюрникова, Л. А. Основы научных исследований в научно-технической сфере [Электронный ресурс]: учебно-методическое пособие / Л. А. Маюрникова, С. В. Новосёлов. — Кемерово: Кемеровский технологический институт пищевой промышленности, 2009. — 123 с. — Режим доступа: http://www.iprbookshop.ru/14381.html
4. Вайнштейн, М. З. Основы научных исследований [Электронный ресурс] : учебное пособие / М. З. Вайнштейн, В. М. Вайнштейн, О. В. Кононова. — Йошкар-Ола: Марийский государственный технический университет, Поволжский государственный технологический университет, ЭБС АСВ, 2011. — 216 с. — Режим доступа: http://www.iprbookshop.ru/22586.html
5. Мокий, М.С. Методология научных исследований[Текст]: учебник / М.С. Мокий, А.Л. Никифоров, В.С. Мокий. - М.: Юрайт, 2015. - 255 с.
6. Рогов, В.А. Методика и практика технических экспериментов[Текст]: учеб.пособие / В.А. Рогов, А.В. Антонов, Г.Г. Поздняк. – М.: Академия, 2005. – 288 с.
7. Щербаков, А. Интернет-аналитика [Электронный ресурс]: поиск и оценка информации в web-ресурсах: практическое пособие / А. Щербаков. - М.: Книжный мир, 2012. - 78 с. - URL: http://biblioclub.ru/index.php?page=book&id=89693.
8. Моделирование систем[Текст]: учебник для вузов / С.И. Дворецкий, Ю.Л. Муромцев, В.А. Погонин, А.Г. Схиртладзе. – М.: Академия, 2009. – 320 с.
9. Порсев, Е. Г. Организация и планирование экспериментов [Электронный ресурс]: учебное пособие / Е. Г. Порсев.— Новосибирск : Новосибирский государственный технический университет, 2010. — 155 с. — Режим доступа: http://www.iprbookshop.ru/45415.html
10. Маюрникова, Л. А. Основы научных исследований в научно-технической сфере [Электронный ресурс] : учебно-методическое пособие / Л. А. Маюрникова, С. В. Новосёлов. — Кемерово : Кемеровский технологический институт пищевой промышленности, 2009. — 123 с. — Режим доступа: http://www.iprbookshop.ru/14381.html
11. Вайнштейн, М. З. Основы научных исследований [Электронный ресурс] : учебное пособие / М. З. Вайнштейн, В. М. Вайнштейн, О. В. Кононова. — Йошкар-Ола : Марийский государственный технический университет,

Поволжский государственный технологический университет, ЭБС АСВ, 2011. — 216 с. — Режим доступа: http://www.iprbookshop.ru/22586.html

12. Мокий, М.С. Методология научных исследований [Текст]: учебник / М.С. Мокий, А.Л. Никифоров, В.С. Мокий. - М.: Юрайт, 2015. - 255 с.

13. Рогов, В.А. Методика и практика технических экспериментов [Текст]: учеб.пособие / В.А. Рогов, А.В. Антонов, Г.Г. Поздняк. – М.: Академия, 2005. – 288 с. 6. Васильев, А.Н. Самоучитель С++ с примерами и задачами/ А.Н. Васильев.– СПб.: Наука и Техника, 2010. – 480с.

14. Кузнецов, М.В. С++. Мастер-класс в задачах и примерах / М.В. Кузнецов,И.В. Симдянов. – СПб.: БХВ-Петербург, 2007. – 480с.

15. Культин, Н. Microsoft Visual C++ в задачах и примерах / Н. Культин. – Издательство: BHV, 2010. – 274 с.

```
#include <iostream>
#include<fstream>
#include <thread>
#include<string>
#include<regex>
#include<map>


using namespace std;


static regex reg1("([A-Z]{3})([0-9]{1,4})");

static regex reg2("([A-Z]{2})(([\\s]{1})([0-9]{1,4}))");

static regex reg3("([A-Z]{2})([0-9]{1,5})");

static regex reg4("([A-Z][A-Z0-9]|[0-9][A-Z])(([\\s]{1})([0-9]{1,4}))");

static regex reg5("([A-Z][A-Z0-9]|[0-9][A-Z])([0-9]{1,5})");

static regex reg6("([0-9]{1,5})");



/* Функция приведения строки к нормальному/стандартному виду
    - возвращает строку вида: AFL1234, AF12345, 12345,
    без пробелов и ведущих нулей */
string getFullStringForEquality(const string& str)
{
    cmatch result;
    string strNumOfFlight;
    string strAviaCode;


    // Получение номера рейса и кода авиакомпании
    if(regex_match(str.c_str(),result,reg1) ||
        regex_match(str.c_str(),result,reg2) ||
        regex_match(str.c_str(),result,reg3) ||
        regex_match(str.c_str(),result,reg4) ||
        regex_match(str.c_str(),result,reg5))
    {
        strAviaCode = result[1].str();
        strNumOfFlight = result[2].str();
    }
    else if(regex_match(str.c_str(),result,reg6))
    {
        strAviaCode = "";
        strNumOfFlight = result[1].str();
    }

    // Очистка от пробелов и ведущих нулей
    strNumOfFlight.erase(remove(strNumOfFlight.begin(),strNumOfFlight.end(),'
'),strNumOfFlight.end());

    size_t pos = strNumOfFlight.find_first_not_of('0');
    if(pos !=string::npos)
    {
        strNumOfFlight = strNumOfFlight.substr(pos);
    }
```

```
    return strAviaCode + strNumOfFlight;
}


void readStrFromFile(const string& fileNameIn,const string& fileNameOut)
/*Считываем данные из файла, приводим к "нормальному виду",
если есть нормальный ключ, то затираем строчку, иначе добавляем,
добавляем в файл непустые строчки,
на вход - название файлов
 входных и выходных данных*/
{
    //"/home/user/task_2/1_in.txt"

    ifstream fin(fileNameIn);
    ofstream fout(fileNameOut);

    map<string,string>dataFromFile;

    string str;
    string normalStr;

    if(fin.is_open()&&fout.is_open())
    {
        cout<<"File is open\n";
        while(!fin.eof())
        {
            str = "";
            getline(fin,str);
            normalStr = getFullStringForEquality(str);

            if(dataFromFile.find(normalStr) != dataFromFile.end())
            {
                dataFromFile[normalStr] = "";
            }
            else
            {
                dataFromFile[normalStr] = str;
            }
        }

        for(const auto& [normalStr,str] : dataFromFile)
        {
            if(str.empty() == false)
            {
                fout<<str<<'\n';
            }
        }

        cout<<"Data in file\n";
    }
    else
    {
        cout<<"File is not open\n";
    }
    fin.close();
    fout.close();

}

int main()
{
    //чтение из файла

    string fileNameInFirst = "/home/user/task_2/1_in.txt";
```

```
    string fileNameOutFirst = "/home/user/task_2/1_out.txt";

    string fileNameInSecond = "/home/user/task_2/2_in.txt";
    string fileNameOutSecond = "/home/user/task_2/2_out.txt";


    thread firstFile(readStrFromFile,fileNameInFirst,fileNameOutFirst);
    thread secondFile(readStrFromFile,fileNameInSecond,fileNameOutSecond);
    firstFile.join();
    secondFile.join();

    return 0;
}
```

Приложение 2

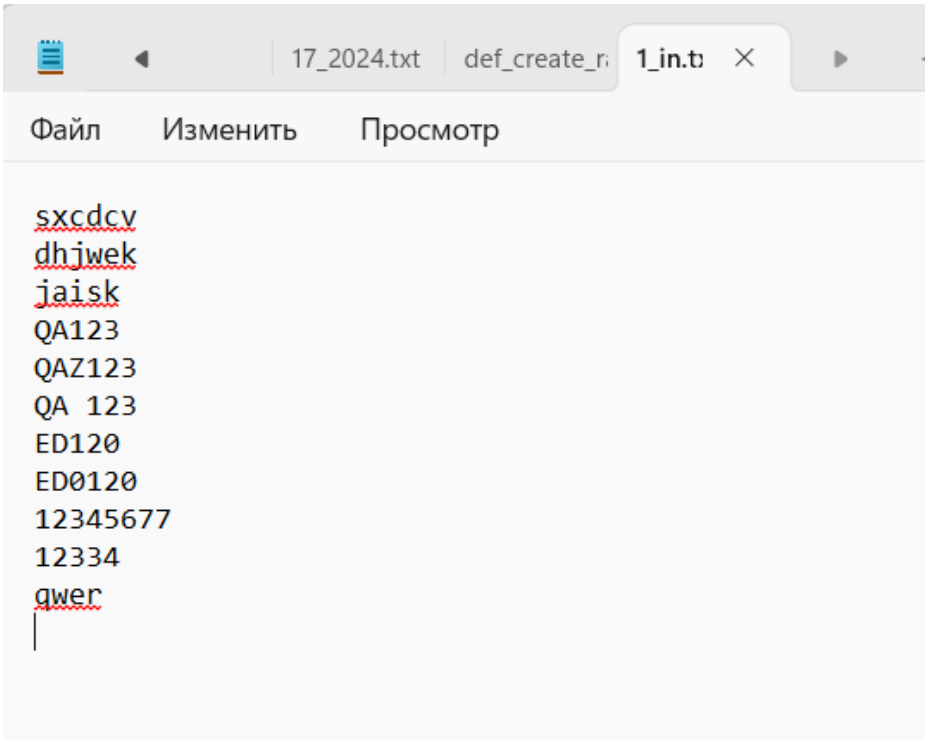

Рис.1 Содержимое файла 1_in.txt

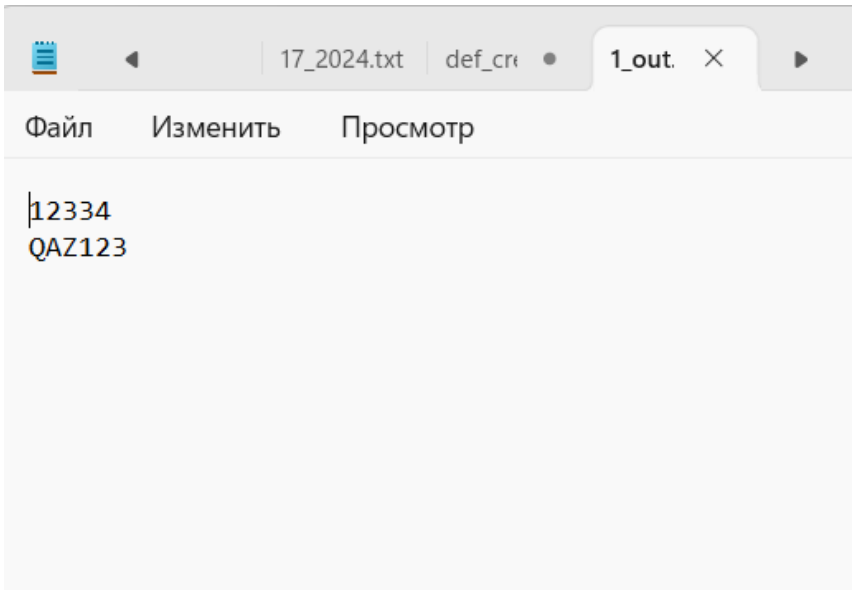

Рис.2 Результат работы в файле 1_out.txt

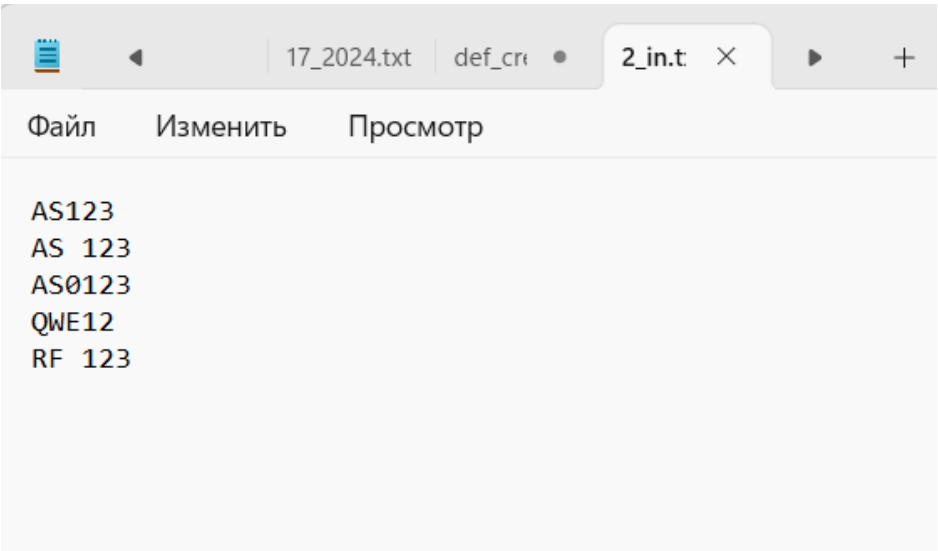

Рис.3 Содержимое файла 2_in.txt

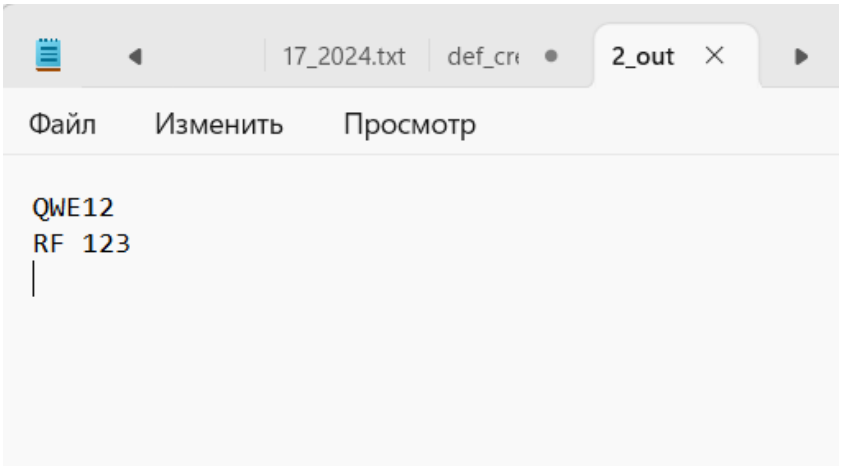

Рис.4 Результат работы в файле 2_out.txt