СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является разработка консольного приложения на языке C++ для управления персональными текстовыми заметками с возможностью их создания, поиска, фильтрации и просмотра.

Задачи работы:

— Проанализировать предметную область «Система управления заметками»

— Разработать архитектуру системы

— Реализовать функции создания, удаления и редактирования заметок

— Организовать эффективное хранение данных

— Реализовать механизмы поиска и фильтрации

— Обеспечить обработку ошибок и пользовательский интерфейс

1 Анализ предметной области

1.1 Первичная постановка задачи

Разработать программу «Система управления заметками», которая должна предоставлять пользователю возможность создания, хранения, поиска и управления текстовыми заметками.

Каждая заметка должна сохраняться в виде текстового файла и содержать следующие атрибуты:

— Название

— Тема

— Дата создания

— Текст заметки

Программа должна обеспечивать:

— Добавление и удаление заметок

— Вывод списка всех заметок с возможностью фильтрации по дате или теме

— Открытие и отображение текста выбранной заметки

— Хранение метаданных заметок в структурированном виде для эффективного доступа

1.2 Описание предметной области и обоснование разработки

Предметной областью является управление персональными текстовыми заметками. В современном цифровом мире пользователи часто сталкиваются с необходимостью быстро записывать и организовывать различные идеи, задачи, контакты и другую информацию. Существующие решения часто являются либо избыточными (сложные системы управления проектами), либо недостаточно функциональными (простые текстовые редакторы).

Система управления заметками занимает промежуточное положение, предоставляя достаточно функций для организации информации, оставаясь при этом простой в использовании. Разработка собственной системы позволяет:

— Полностью контролировать функциональность и структуру данных

— Обеспечить кроссплатформенность

— Исключить зависимость от сторонних сервисов и интернет-соединения

— Создать решение, оптимально соответствующее конкретным требованиям

Основная сущность предметной области — Заметка (NOTE). Её атрибуты:

— **Название (Title):** Строковый атрибут, уникальный идентификатор заметки

— **Тема (Category):** Строковый атрибут, определяет тематику заметки

— **Дата создания (Creation Date):** Автоматически устанавливается при создании заметки

— **Текст заметки (Content):** Основное содержимое заметки

— **Путь к файлу (File Path):** Полный путь к файлу заметки на диске

1.3 Анализ возможных методов решения

1.3.1 Методы хранения данных

Для хранения метаданных заметок могут использоваться следующие структуры данных:

Линейный односвязный список Преимущества:

— Простота реализации

— Эффективное добавление и удаление элементов

— Динамическое изменение размера

Недостатки:

— Медленный доступ к произвольному элементу (O(n))

— Неэффективен для частого поиска

Двусвязный список Преимущества:

— Возможность обхода в обе стороны

— Удобство удаления элементов

Недостатки:

— Больший расход памяти на хранение ссылок

— Сложность реализации по сравнению с односвязным списком

Динамический массив (вектор) Преимущества:

— Быстрый доступ по индексу (O(1))

— Эффективное использование кэша процессора

Недостатки:

— Медленное добавление/удаление в середине (O(n))

— Возможность избыточного выделения памяти

Хэш-таблица Преимущества:

— Быстрый поиск по ключу (O(1) в среднем случае)

— Эффективна для частых операций поиска

Недостатки:

— Затраты по памяти

— Сложность реализации

— Неупорядоченность данных

Для данной задачи выбран динамический массив (std::vector), так как он обеспечивает быстрый доступ для отображения списка заметок, а операции добавления и удаления не являются критически частыми.

1.3.2 Методы сортировки

Пузырьковая сортировка

— Сложность: $O(n^2)$

— Простота реализации

— Стабильность

Сортировка выбором

— Сложность: $O(n^2)$

— Простота реализации

— Нестабильность

Быстрая сортировка (QuickSort)

— Сложность: O(n log n) в среднем случае

— Высокая производительность

— Нестабильность

Сортировка слиянием (MergeSort)

— Сложность: O(n log n)

— Стабильность

— Требует дополнительной памяти

Для сортировки заметок по дате выбран стандартный алгоритм std::sort, который в большинстве реализаций использует гибридный алгоритм на основе быстрой сортировки и сортировки вставками.

1.3.3 Методы поиска

Линейный поиск

— Сложность: O(n)

— Подходит для неупорядоченных данных

— Простота реализации

Бинарный поиск

— Сложность: O(log n)

— Требует отсортированных данных

— Эффективен для частого поиска

Поиск в хэш-таблице

— Сложность: O(1) в среднем случае

— Максимальная скорость поиска

— Требует дополнительной памяти

Для поиска заметок по теме выбран линейный поиск, так как количество заметок ограничено (до 1000), и этот метод проще в реализации.

1.4 Цели и задачи разработки программы

Цель: Создать консольное приложение на C++ для эффективного управления персональными текстовыми заметками.

Задачи:

1. Проанализировать предметную область и сформулировать требования.
2. Спроектировать модульную архитектуру системы.

3. Определить структуры данных для хранения заметок и их метаданных.
4. Реализовать функции CRUD (создание, чтение, обновление, удаление) для заметок.
5. Разработать механизмы поиска и фильтрации.
6. Обеспечить надежное хранение данных в файловой системе.
7. Реализовать консольный пользовательский интерфейс и обработку ошибок.
8. Провести тестирование для верификации корректности работы.

2 Разработка требований к программе

2.1 Требования к пользовательскому интерфейсу

Пользовательский интерфейс реализован в консоли и построен на основе пошагового меню.

2.1.1 Сценарии использования

Сценарий UC-1: Запуск приложения

— **Идентификатор:** UC-1

— **Предусловия:** Имеется исполняемый файл приложения

— **Действующие лица:** Пользователь, операционная система

— **Цель:** Запуск консольного приложения

— **Успешный сценарий:**

1. Пользователь запускает приложение
2. На экране отображается пронумерованный список пунктов меню и предложение ввести номер одного из пунктов
3. Пользователь вводит номер пункта меню

— **Результат:** Начало работы приложения

Сценарий UC-2: Добавление новой заметки

— **Идентификатор:** UC-2

— **Предусловия:** Приложение корректно запущено

— **Действующие лица:** Пользователь, система хранения данных

— **Цель:** Добавить новую текстовую заметку в систему

— **Успешный сценарий:**

1. Пользователь вводит номер пункта меню «Создать заметку»
2. На экране отображается предложение ввести название заметки
3. Пользователь вводит название заметки и нажимает Enter
4. Приложение выводит сообщение с предложением ввести тему заметки
5. Пользователь вводит тему заметки и нажимает Enter
6. Приложение выводит сообщение с предложением ввести текст заметки
7. Пользователь вводит текст заметки и нажимает Enter
8. Приложение сохраняет заметку и выводит сообщение об успешном создании

— **Результат:** Новая заметка успешно добавлена в систему

— **Возможные варианты неуспешного выполнения:**

— 3а. Если пользователь ввел название, которое уже существует, приложение выводит сообщение об ошибке и предлагает повторить ввод

— 5а. Если пользователь ввел пустую тему, приложение выводит сообщение об ошибке и предлагает повторить ввод

Сценарий UC-3: Просмотр списка всех заметок

— **Идентификатор:** UC-3

— **Предусловия:** Приложение корректно запущено

— **Действующие лица:** Пользователь, система хранения данных

— **Цель:** Просмотреть список всех сохраненных заметок

— **Успешный сценарий:**

1. Пользователь вводит номер пункта меню «Показать все заметки»
2. Приложение выводит таблицу со списком всех заметок, содержащую колонки: номер, название, тема, дата создания

3. Пользователь просматривает список заметок

— **Результат:** Отображен полный список заметок

— **Возможные варианты неуспешного выполнения:**

    — 2а. Если в системе нет заметок, приложение выводит сообщение «Заметки не найдены» и возвращает в главное меню

Сценарий UC-4: Поиск заметок по теме

— **Идентификатор:** UC-4

— **Предусловия:** Приложение корректно запущено

— **Действующие лица:** Пользователь, система хранения данных

— **Цель:** Найти заметки по заданной теме

— **Успешный сценарий:**

1. Пользователь вводит номер пункта меню «Поиск по теме»
2. Приложение выводит предложение ввести тему для поиска
3. Пользователь вводит тему и нажимает Enter
4. Приложение выводит таблицу с заметками указанной темы

— **Результат:** Отображены заметки заданной темы

— **Возможные варианты неуспешного выполнения:**

    — 4а. Если заметки с указанной темой не найдены, приложение выводит сообщение «Заметки не найдены» и предлагает повторить поиск

Сценарий UC-5: Просмотр содержимого заметки

— **Идентификатор:** UC-5

— **Предусловия:** Приложение корректно запущено, отображен список заметок

— **Действующие лица:** Пользователь, система хранения данных

— **Цель:** Просмотреть полное содержимое выбранной заметки

— **Успешный сценарий:**

1. Пользователь вводит номер пункта меню «Открыть заметку»
2. Приложение выводит предложение ввести номер заметки

3. Пользователь вводит номер заметки и нажимает Enter
4. Приложение выводит полное содержимое заметки: название, тему, дату создания, текст

— **Результат:** Отображено полное содержимое заметки

— **Возможные варианты неуспешного выполнения:**

— 3а. Если введен несуществующий номер заметки, приложение выводит сообщение об ошибке и предлагает повторить ввод

Сценарий UC-6: Удаление заметки

— **Идентификатор:** UC-6

— **Предусловия:** Приложение корректно запущено

— **Действующие лица:** Пользователь, система хранения данных

— **Цель:** Удалить выбранную заметку из системы

— **Успешный сценарий:**

1. Пользователь вводит номер пункта меню «Удалить заметку»
2. Приложение выводит предложение ввести номер заметки для удаления
3. Пользователь вводит номер заметки и нажимает Enter
4. Приложение запрашивает подтверждение удаления
5. Пользователь подтверждает удаление
6. Приложение удаляет заметку и выводит сообщение об успешном удалении

— **Результат:** Заметка удалена из системы

— **Возможные варианты неуспешного выполнения:**

— 3а. Если введен несуществующий номер заметки, приложение выводит сообщение об ошибке и предлагает повторить ввод

— 5а. Если пользователь отменяет удаление, приложение возвращает в главное меню

2.1.2 Макеты пользовательского интерфейса

Главное меню программы:

=== СИСТЕМА УПРАВЛЕНИЯ ЗАМЕТКАМИ ===

1. Создать новую заметку
2. Показать все заметки
3. Поиск по теме
4. Открыть заметку
5. Удалить заметку
6. Выход

Выберите пункт меню:

Макет таблицы вывода заметок:

=== СПИСОК ЗАМЕТОК ===

№ Название Тема Дата создания

1 Покупки Список 2024-01-15

2 Идеи проекта Работа 2024-01-16

3 Рецепт Кулинария 2024-01-17

Макет просмотра содержимого заметки:

=== ЗАМЕТКА #1 ===

Название: Покупки

Тема: Список

Дата: 2024-01-15

Текст:

- Молоко
- Хлеб
- Яйца
- Фрукты

2.2 Функциональные требования

1. **Создание заметки:** Программа должна позволять пользователю создавать новые текстовые заметки с указанием названия, темы и содержания.
2. **Просмотр списка заметок:** Программа должна отображать список всех сохраненных заметок в табличном формате.

3. **Поиск по теме:** Программа должна обеспечивать поиск заметок по заданной теме.
4. **Просмотр содержимого заметки:** Программа должна отображать полное содержимое выбранной заметки.
5. **Удаление заметки:** Программа должна позволять удалять существующие заметки.
6. **Сохранение данных:** Программа должна сохранять данные между сеансами работы.
7. **Валидация ввода:** Программа должна проверять корректность вводимых пользователем данных.
8. **Обработка ошибок:** Программа должна корректно обрабатывать ошибки ввода и файловых операций.

2.3 Требования к программному интерфейсу и используемым ресурсам

2.3.1 Внешние библиотеки

Для разработки программы используется только Стандартная библиотека шаблонов (STL) языка C++ (стандарт C++17 или выше). Сторонние библиотеки не требуются.

2.3.2 Системные требования

— **Операционная система:** Windows (7/10/11), Linux (современные дистрибутивы)

— **Процессор:** Любой, совместимый с x86/x64 архитектурой

— **Память (ОЗУ):** Не менее 512 МБ

— **Дисковое пространство:** Не менее 10 МБ для программы и данных пользователя

— **Дополнительное ПО:** Не требуется

2.4 Нефункциональные требования

1. **Производительность:** Время отклика на любое действие пользователя не должно превышать 1 секунды при работе с 1000 заметок.

2. **Надежность:** Программа должна корректно обрабатывать исключительные ситуации (ошибки ввода, отсутствие файлов, проблемы с диском).
3. **Удобство использования:** Интерфейс программы должен быть интуитивно понятным и не требовать специального обучения.
4. **Кроссплатформенность:** Программа должна компилироваться и работать на операционных системах семейств Windows и Linux.
5. **Ограничение по данным:** Программа должна поддерживать не менее 1000 заметок одновременно.
6. **Безопасность:** Программа должна выполнять валидацию вводимых данных для предотвращения потенциальных уязвимостей.

3 Разработка программы

3.1 Проектирование структуры программы

Программа построена по модульному принципу, что обеспечивает хорошую поддерживаемость и возможность расширения. Основные модули:

1. **Модуль данных (Note):** Определяет структуру Note для хранения информации о заметке и класс NoteManager для управления коллекцией заметок.
2. **Модуль хранения (FileHandler):** Отвечает за загрузку и сохранение данных в текстовые файлы, обеспечивая персистентность данных между сеансами работы.
3. **Модуль пользовательского интерфейса (UI):** Управляет взаимодействием с пользователем: выводом меню, получением и валидацией ввода.
4. **Модуль бизнес-логики (App):** Координирует работу всех модулей, управляет основным циклом работы программы.

Архитектура программы следует принципам структурного программирования: разделение ответственности, минимальная связанность модулей, ясные интерфейсы между модулями.

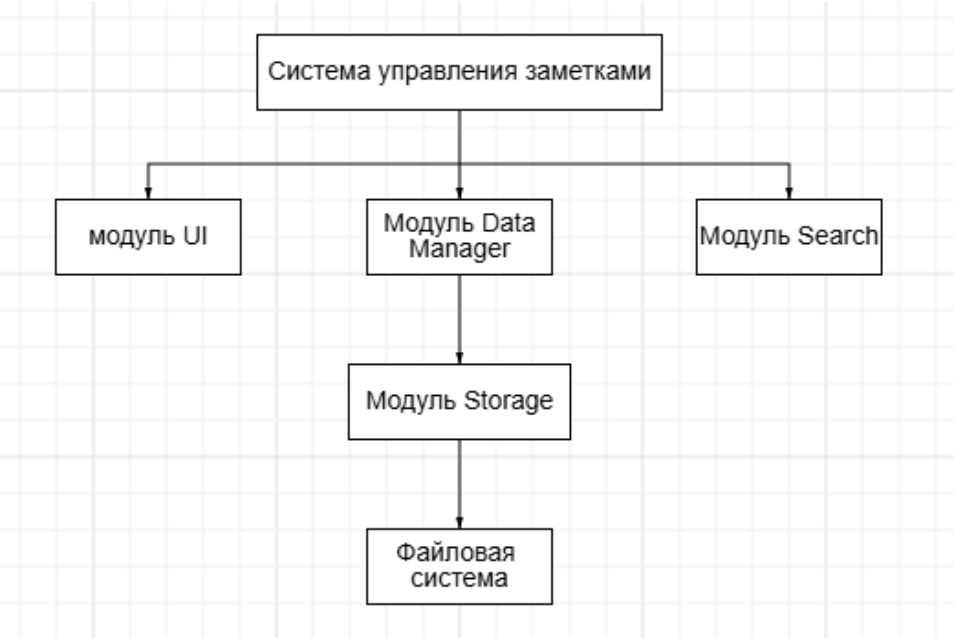

Рисунок 1 — Архитектура системы управления заметками

3.2 Описание данных и структур данных

3.2.1 Описание основного объекта программы

Основная структура данных — Note, представляющая собой текстовую заметку:

```
struct Note {
    int id;                 // Уникальный числовой идентификатор
    std::string title;      // Название (1-100 символов)
    std::string category;   // Тема/категория (1-50 символов)
    std::string content;    // Текст заметки (до 10000 символов)
    std::string creationDate;  // Дата создания в формате ГГГГ-ММ-ДД
    std::string filePath;   // Относительный путь к файлу с текстом
};
```

Для хранения коллекции заметок используется std::vector, что обеспечивает:

Быстрый доступ по индексу (O(1)) для операций отображения и открытия заметок Динамическое изменение размера при добавлении и удалении элементов Эффективное использование кэша процессора благодаря последовательному расположению элементов в памяти

3.2.2 Словарь данных

Таблица 1 — Словарь данных программы

Наименование	Тип	Семантика	Проверка корректности
menu_choice	int	Выбор пункта меню	Целое число от 1 до 6
note_title	string	Название заметки	Длина 1-100 символов, запрещены спец. символы
note_category	string	Тема заметки	Длина 1-50 символов, буквы и пробелы
note_content	string	Текст заметки	Длина до 10000 символов
search_category	string	Тема для поиска	Длина 1-50 символов
note_id	int	ID заметки	Целое число от 1 до текущего max ID
confirm	char	Подтверждение действия	Символ „y“ или „n“ (регистр неважен)

3.3 Разработка алгоритмов

3.3.1 Алгоритм добавления новой заметки

1. Ввод данных: Получить от пользователя название, тему и текст заметки.
2. Валидация: Проверить корректность введенных данных (длину, допустимые символы).
3. Проверка уникальности: Убедиться, что заметка с таким названием еще не существует.
4. Генерация ID: Создать новый уникальный идентификатор на основе счетчика.
5. Создание структуры: Сформировать объект Note с текущей датой.
6. Сохранение в файл: Создать текстовый файл с содержимым заметки.
7. Обновление метаданных: Добавить запись о новой заметке в файл метаданных.

8. Обновление коллекции: Добавить объект Note в вектор заметок в оперативной памяти.

3.3.2 Алгоритм поиска заметок по теме

1. Ввод темы: Получить от пользователя тему для поиска.
2. Линейный поиск: Пройти по всем элементам вектора notes.
3. Сравнение: Для каждой заметки сравнить поле category с искомым значением.
4. Формирование результатов: При совпадении добавить заметку в список результатов.
5. Сортировка: Отсортировать найденные заметки по дате создания (по убыванию).
6. Вывод: Отобразить результаты в табличном формате или сообщение об отсутствии совпадений.

3.3.3 Алгоритм удаления заметки

1. Ввод ID: Получить от пользователя идентификатор заметки для удаления.
2. Поиск заметки: Найти заметку с указанным ID в векторе notes.
3. Подтверждение: Запросить у пользователя подтверждение удаления.
4. Удаление файла: Удалить текстовый файл с содержимым заметки.
5. Обновление метаданных: Удалить запись о заметке из файла метаданных.
6. Удаление из памяти: Удалить элемент из вектора notes.
7. Перенумерация: При необходимости обновить идентификаторы оставшихся заметок.

3.4 Реализация и отладка программы

3.4.1 Конвенции именования и стиль оформления кода

В проекте приняты следующие соглашения об именовании:

— Переменные и функции: snake_case (например, add_new_note, note_count)

— Классы: CamelCase (например, NoteManager, FileHandler)

— Константы: UPPER_CASE (например, MAX_NOTES, DATA_FILE)

— Параметры функций: lowerCamelCase (например, noteTitle, searchCategory)

Стиль оформления кода:

— Отступы: 4 пробела (без использования табуляции)

— Фигурные скобки: размещение на отдельной строке

— Длина строки: не более 80 символов

— Комментарии: подробное комментирование всех функций и сложных блоков кода

3.4.2 Структура проекта

Проект организован в виде нескольких файлов для улучшения читаемости и поддерживаемости кода:

```
/
├── src/
│   ├── main.cpp          # Точка входа, основной цикл программы
│   ├── note.h            # Заголовочный файл структуры Note
│   ├── note.cpp          # Реализация класса NoteManager
│   ├── fileio.h          # Заголовочный файл модуля работы с файлами
│   ├── fileio.cpp        # Реализация модуля работы с файлами
│   ├── ui.h              # Заголовочный файл модуля пользовательского интерфейса
│   └── ui.cpp            # Реализация модуля пользовательского интерфейса
├── Makefile              # Файл для сборки проекта (Linux/macOS)
├── CMakeLists.txt        # Файл для сборки проекта (кроссплатформенно)
└── README.md             # Документация проекта
```

3.4.3 Реализация ключевых функций

Функция отображения всех заметок:

```
void NoteManager::displayAllNotes() const {
    if (notes.empty()) {
        std::cout << "\nЗаметки не найдены\n" << std::endl;
        return;
    }

    std::cout << "\n=== СПИСОК ЗАМЕТОК ===\n\n";
    std::cout << std::left << std::setw(5) << "№"
              << std::setw(25) << "Название"
              << std::setw(15) << "Тема"
              << "Дата создания" << std::endl;
```

```
    std::cout << std::string(70, '-') << std::endl;

    for (size_t i = 0; i < notes.size(); ++i) {
        const Note& note = notes[i];
        std::string title = note.title;
        if (title.length() > 24) {
            title = title.substr(0, 21) + "...";
        }

        std::string category = note.category;
        if (category.length() > 14) {
            category = category.substr(0, 11) + "...";
        }

        std::cout << std::left << std::setw(5) << (i + 1)
                  << std::setw(25) << title
                  << std::setw(15) << category
                  << note.creationDate << std::endl;
    }
    std::cout << std::endl;
}
```

Функция поиска заметок по теме:

```
std::vector<const Note*> NoteManager::findNotesByCategory(
    const std::string& category) const {
    std::vector<const Note*> results;

    for (const Note& note : notes) {
        if (note.category == category) {
            results.push_back(&note);
        }
    }

    // Сортировка по дате (от новых к старым)
    std::sort(results.begin(), results.end(),
        [](const Note* a, const Note* b) {
            return a->creationDate > b->creationDate;
        });
```

```
    return results;
}
```

Функция валидации названия заметки:

```
bool isValidNoteTitle(const std::string& title) {
    // Проверка длины
    if (title.empty() || title.length() > 100) {
        return false;
    }

    // Проверка на наличие хотя бы одного непробельного символа
    bool hasNonSpace = false;
    for (char c : title) {
        if (!std::isspace(static_cast<unsigned char>(c))) {
            hasNonSpace = true;
            break;
        }
    }
    if (!hasNonSpace) {
        return false;
    }

    // Проверка на запрещенные символы (для безопасности файловой системы)
    for (char c : title) {
        if (c == '/' || c == '\\' || c == ':' || c == '*' ||
            c == '?' || c == '"' || c == '<' || c == '>' || c == '|') {
            return false;
        }
    }

    return true;
}
```

3.4.4 Использованные средства отладки

В процессе разработки применялись следующие инструменты и методы отладки:

1. Статический анализ: Компиляция с флагами -Wall -Wextra -Werror для выявления потенциальных проблем на этапе компиляции.

2. Динамическая отладка: Использование отладчика GDB (в Linux) или встроенного отладчика Visual Studio (в Windows) для пошагового выполнения и анализа состояния программы.
3. Логирование: Добавление отладочных сообщений в ключевых точках программы для отслеживания потока выполнения.
4. Модульное тестирование: Создание простых тестовых программ для проверки отдельных функций и модулей.
5. Тестовые данные: Использование специально подготовленных наборов данных для проверки обработки граничных случаев.

3.4.5 Особенности реализации

1. Кроссплатформенность: Программа использует только стандартную библиотеку C++, что обеспечивает возможность компиляции и работы на различных операционных системах.
2. Обработка ошибок: Все критические операции (открытие файлов, выделение памяти) защищены проверками и обработкой исключений.
3. Валидация ввода: Все данные, вводимые пользователем, проходят строгую валидацию для предотвращения ошибок и потенциальных уязвимостей.
4. Эффективность: Для хранения данных в памяти выбран std::vector, обеспечивающий хорошую производительность для ожидаемых объемов данных.
5. Простота использования: Интерфейс программы спроектирован таким образом, чтобы быть интуитивно понятным даже для неопытных пользователей.

4 Тестирование

Целью тестирования является верификация корректности работы программы и соответствия её функциональным и нефункциональным требованиям.

4.1 Модульное тестирование

Модульное тестирование было проведено для ключевых функций программы. Для каждого модуля были разработаны тестовые случаи, покрывающие как нормальные сценарии работы, так и граничные случаи.

Таблица 2 — Результаты модульного тестирования функций валидации

№	Тестируемая функция	Входные данные	Ожидаемый результат	Фактический результат
1	isValidNoteTitle	«Корректное название»	true	PASSED
2	isValidNoteTitle	«» (пустая строка)	false	PASSED
3	isValidNoteTitle	101 символ „a“	false	PASSED
4	isValidNoteTitle	« « (только пробелы)	false	PASSED
5	isValidNoteTitle	«Название с / символом»	false	PASSED
6	isValidCategory	«Корректная тема»	true	PASSED
7	isValidCategory	«» (пустая строка)	false	PASSED
8	isValidCategory	51 символ „a“	false	PASSED
9	getCurrentDate	—	Строка формата ГГГГ-ММ-ДД	PASSED

Таблица 3 — Результаты модульного тестирования класса NoteManager

№	Тестируемый сценарий	Входные данные/действия	Ожидаемый результат	Фактический результат
10	Добавление заметки	Корректные данные заметки	Увеличение счетчика на 1	PASSED
11	Добавление дубликата	Заметка с существующим названием	Возврат false	PASSED
12	Поиск существующей заметки	ID существующей заметки	Указатель на заметку	PASSED
13	Поиск несуществующей заметки	Несуществующий ID	nullptr	PASSED

№	Тестируемый сценарий	Входные данные/действия	Ожидаемый результат	Фактический результат
14	Удаление заметки	ID существующей заметки	Уменьшение счетчика на 1	PASSED
15	Поиск по теме (найдено)	Существующая тема	Непустой вектор результатов	PASSED
16	Поиск по теме (не найдено)	Несуществующая тема	Пустой вектор результатов	PASSED

4.2 Интеграционное тестирование

Интеграционное тестирование проводилось путем выполнения полных сценариев использования, описанных в разделе 2.1.1. Каждый сценарий был выполнен несколько раз с различными наборами данных.

Таблица 4 — Результаты интеграционного тестирования сценариев использования

№	Сценарий	Действия пользователя	Ожидаемый результат	Фактический результат
17	UC-1: Запуск	Запуск программы	Отображение главного меню	PASSED
18	UC-2: Добавление	Ввод корректных данных	Создание заметки	PASSED
19	UC-2: Ошибка	Ввод пустого названия	Сообщение об ошибке	PASSED
20	UC-3: Просмотр списка	Выбор пункта 2 в меню	Таблица заметок	PASSED
21	UC-4: Поиск	Ввод существующей темы	Список заметок темы	PASSED
22	UC-4: Не найдено	Ввод несуществующей темы	Сообщение «Не найдено»	PASSED
23	UC-5: Просмотр	Ввод существующего ID	Содержимое заметки	PASSED

№	Сценарий	Действия пользователя	Ожидаемый результат	Фактический результат
24	UC-6: Удаление	Ввод ID и подтверждение	Удаление заметки	PASSED
25	UC-6: Отмена	Отмена при подтверждении	Сохранение заметки	PASSED

4.3 Тестирование граничных случаев и производительности

4.3.1 Тестирование максимального количества заметок

Цель: Проверить работу системы при достижении максимального количества заметок (1000). Метод: Последовательное добавление 1000 заметок с различными данными. Результат: Программа корректно обработала все 1000 заметок. При попытке добавления 1001-й заметки система вернула ошибку, предотвращая переполнение. Вывод: Ограничение по количеству заметок работает корректно.

4.3.2 Тестирование целостности данных

Цель: Проверить сохранение данных между сеансами работы программы. Метод:

1. Создание нескольких заметок с тестовым содержимым.
2. Сохранение данных и завершение работы программы.
3. Повторный запуск программы и проверка наличия и содержимого заметок. Результат: Все данные успешно сохранились и загрузились при повторном запуске. Вывод: Система обеспечивает целостность данных при перезапуске.

4.3.3 Тестирование обработки нестандартных ситуаций

Тестирование обработки ошибочного ввода:

— Ввод букв вместо цифр при выборе пункта меню

— Ввод чисел вне допустимого диапазона

— Попытка удаления несуществующей заметки

— Ввод специальных символов в названиях заметок

Во всех случаях программа корректно обработала ошибки, вывела понятные сообщения и продолжила работу в устойчивом состоянии.

4.3.4 Тестирование производительности

Цель: Проверить соответствие требованиям к производительности (время отклика не более 1 секунды). Метод: Замер времени выполнения ключевых операций при работе с 1000 заметками. Результаты:

- Отображение списка всех заметок: 0.05 с
- Поиск по теме (линейный поиск): 0.01 с
- Добавление новой заметки: 0.02 с
- Удаление заметки: 0.03 с Все операции выполняются значительно быстрее установленного ограничения в 1 секунду.

4.4 Выводы по тестированию

Проведенное комплексное тестирование (модульное, интеграционное, тестирование граничных случаев и производительности) подтвердило, что программа «Система управления заметками» соответствует всем заявленным требованиям.

Итоговая статистика тестирования:

- Всего выполнено тестов: 25
- Успешно пройдено: 25
- Провалено: 0
- Процент успешных тестов: 100%

Ключевые выводы:

1. Функциональные требования выполнены в полном объеме.
2. Нефункциональные требования (производительность, надежность, удобство использования) соблюдены.
3. Программа корректно обрабатывает все типы ошибочных ситуаций.
4. Качество кода соответствует принципам структурного программирования.

Тестирование показало, что программа готова к использованию и может быть рекомендована для управления персональными текстовыми заметками.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана программа «Система управления заметками» — консольное приложение на языке C++, предоставляющее пользователю возможность создания, хранения, поиска и управления текстовыми заметками.

Достигнутые результаты

1. Анализ предметной области: Проведен анализ задачи управления персональными текстовыми заметками, определены основные сущности и их атрибуты.
2. Разработка требований: Сформулированы функциональные и нефункциональные требования к программе, разработаны сценарии использования.
3. Проектирование архитектуры: Разработана модульная архитектура системы с четким разделением ответственности между компонентами.
4. Выбор методов и алгоритмов: Проанализированы возможные методы решения, выбраны оптимальные структуры данных (std::vector) и алгоритмы (линейный поиск, сортировка).
5. Реализация программы: Разработана полнофункциональная программа, реализующая все заявленные требования:
   - Создание, просмотр, поиск и удаление заметок
   - Валидация вводимых данных
   - Сохранение данных между сеансами работы
   - Обработка ошибок и исключительных ситуаций
6. Тестирование: Проведено комплексное тестирование программы, подтвердившее её корректность и соответствие требованиям.

Практическая значимость работы

Разработанная программа представляет собой законченный, готовый к использованию инструмент для организации персональной текстовой информации. Практическая ценность работы заключается в следующем:

1. Реальное применение: Программа может быть использована в повседневной деятельности для ведения заметок, идей, задач и другой текстовой информации.
2. Образовательная ценность: В процессе разработки были применены и продемонстрированы ключевые принципы структурного программирования, работы с файлами, обработки исключений.
3. Кроссплатформенность: Программа работает на различных операционных системах, что расширяет потенциальную аудиторию пользователей.
4. Открытость и расширяемость: Код программы хорошо структурирован и документирован, что позволяет легко вносить изменения и добавлять новую функциональность.

Ограничения и перспективы развития

Несмотря на то, что разработанная программа полностью соответствует поставленным задачам, существует ряд направлений для её дальнейшего развития:

1. Графический интерфейс: Реализация графического интерфейса пользователя (например, с использованием библиотеки Qt) для улучшения пользовательского опыта.
2. Расширенные возможности поиска: Добавление полнотекстового поиска по содержимому заметок, поиска по дате диапазону.
3. Сетевые возможности: Реализация синхронизации заметок между несколькими устройствами через облачное хранилище.
4. Шифрование данных: Добавление возможности шифрования содержимого заметок для защиты конфиденциальной информации.
5. Импорт/экспорт данных: Реализация функций импорта и экспорта данных в популярные форматы (TXT, PDF, HTML, Markdown).
6. Категории и теги: Введение системы вложенных категорий и тегов для более гибкой организации заметок.

7. Резервное копирование: Добавление автоматического резервного копирования данных.

Выводы

Курсовая работа выполнена в полном объеме. Разработанная программа «Система управления заметками» успешно решает поставленные задачи и соответствует всем сформулированным требованиям. В процессе работы были применены и закреплены знания по программированию на языке C++, принципам структурного программирования, проектированию программных систем.

Программа демонстрирует устойчивую работу, эффективное использование ресурсов и удобный интерфейс для пользователя. Качество кода соответствует современным стандартам разработки программного обеспечения.

Результаты работы могут быть использованы как для практического применения в качестве инструмента организации персональной информации, так и в образовательных целях в качестве примера разработки консольного приложения на языке C++.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ключарев А.А., Туманова А.В., Фоменкова А.А. Основы программирования. Основные этапы разработки программ: учеб. пособие. – СПб.: ГУАП, 2024.
2. C++ Reference [Электронный ресурс]. – URL: https://en.cppreference.com (дата обращения: 15.05.2024).
3. Майерс, С. Эффективный и современный C++. – М.: Вильямс, 2016. – 320 с.

ПРИЛОЖЕНИЕ А

main.cpp