

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	7
1.1 Первичная постановка задачи	7
1.2 Концептуальная модель	7
1.3 Функциональные требования	7
2 АНАЛИЗ МЕТОДОВ ХРАНЕНИЯ ДАННЫХ И АЛГОРИТМОВ	8
2.1 Методы хранения данных	8
2.1.1 Линейный односвязный список	8
2.1.2 Двусвязный список	8
2.1.3 Динамический массив (вектор)	8
2.1.4 Хэш-таблица	9
2.2 Методы сортировки	9
2.2.1 Пузырьковая сортировка	9
2.2.2 Сортировка выбором	9
2.2.3 Быстрая сортировка (QuickSort)	9
2.2.4 Сортировка слиянием (MergeSort)	9
2.3 Методы поиска	10
2.3.1 Линейный поиск	10
2.3.2 Бинарный поиск	10
2.3.3 Поиск в хэш-таблице	10
3 ТЕХНИЧЕСКОЕ ЗАДАНИЕ	10
3.1 Язык программирования и среда разработки	10
3.2 Функциональные требования	10
3.3 Нефункциональные требования	11
4 РАЗРАБОТКА И АНАЛИЗ ТРЕБОВАНИЙ	11
4.1 Требования к пользовательскому интерфейсу	11

4.1.1	Сценарии использования	11
4.1.2	Макеты пользовательского интерфейса	14
4.2	Требования к программному интерфейсу и используемым ресурсам	15
4.2.1	Программный интерфейс	15
4.2.2	Системные требования	16
4.2.3	Интеграционные возможности	16
4.2.4	Ограничения	16
5	ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПРОГРАММЫ	16
5.1	Архитектура программного обеспечения	16
5.1.1	Основные модули системы	17
5.2	Структура данных	17
5.2.1	Описание основного объекта программы	17
5.2.2	Структура для хранения коллекции заметок	18
5.3	Словарь данных	19
5.4	Описание файлов	19
5.4.1	Файл метаданных notes_metadata.dat	19
5.4.2	Текстовые файлы заметок	20
5.5	Алгоритмы работы системы	20
5.5.1	Алгоритм добавления новой заметки	20
5.5.2	Алгоритм поиска заметок по теме	20
5.5.3	Алгоритм удаления заметки	20
5.6	Взаимодействие модулей	21
5.7	Обработка ошибок	22
6	РАЗРАБОТКА И ОТЛАДКА ПРОГРАММЫ	22
6.1	Конвенции именования и стиль оформления кода	22
6.1.1	Конвенции именования	22
6.1.2	Стиль оформления кода	22

6.2	Описание разработанных модулей и функций	23
6.3	Структура проекта	26
6.4	Реализация ключевых алгоритмов	26
6.4.1	Алгоритм загрузки данных	26
6.4.2	Алгоритм поиска по теме	27
6.4.3	Алгоритм валидации ввода	28
6.5	Использованные приемы оптимизации	28
6.5.1	Оптимизация производительности	28
6.5.2	Оптимизация объема кода	29
6.6	Использованные средства отладки	29
6.6.1	Средства компилятора	29
6.6.2	Техники отладки	29
6.6.3	Обработка ошибок	29
6.7	Особенности реализации	30
6.7.1	Обработка исключительных ситуаций	30
6.7.2	Кроссплатформенность	30
6.7.3	Безопасность	30
7	ТЕСТИРОВАНИЕ ПРОГРАММЫ	31
7.1	Архитектура системы тестирования	31
7.2	Наборы тестовых данных	31
7.3	Результаты тестирования по категориям	32
7.4	Тестирование граничных случаев и производительности	35
7.4.1	Тестирование максимального количества заметок	35
7.4.2	Тестирование целостности данных	35
7.4.3	Тестирование обработки нестандартных символов	36
7.5	Итоги автоматизированного тестирования	36
7.6	Ручное тестирование пользовательского интерфейса	37

7.7 Выводы по тестированию	38
8 ЗАКЛЮЧЕНИЕ	39
8.1 Достигнутые результаты	39
8.2 Положительные стороны разработанной программы	40
8.3 Отрицательные стороны и ограничения	40
8.4 Перспективы развития и усовершенствования	40
8.5 Выводы	41

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 Концептуальная модель

Основная сущность — 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 Функциональные требования

1. **Добавление заметки:** Пользователь вводит название, тему, текст. Программа создаёт текстовый файл в указанной директории, сохраняет метаданные

2. **Удаление заметки:** Удаляется файл заметки и соответствующая запись в каталоге
3. **Поиск и фильтрация:** Возможен вывод списка заметок по дате или по теме
4. **Открытие заметки:** Программа считывает и выводит содержимое текстового файла
5. **Целостность данных:** Проверка уникальности названия, обработка ошибок доступа к файлам

2 АНАЛИЗ МЕТОДОВ ХРАНЕНИЯ ДАННЫХ И АЛГОРИТМОВ

2.1 Методы хранения данных

Для хранения метаданных заметок могут использоваться следующие структуры данных:

2.1.1 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

2.1.2 Двусвязный список

Преимущества:

- Возможность обхода в обе стороны
- Удобство удаления элементов

Недостатки:

- Большой расход памяти на хранение ссылок
- Сложность реализации по сравнению с односвязным списком

2.1.3 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

2.1.4 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

2.2 Методы сортировки

2.2.1 Пузырьковая сортировка

- Сложность: $O(n^2)$
- Простота реализации
- Стабильность

2.2.2 Сортировка выбором

- Сложность: $O(n^2)$
- Простота реализации
- Нестабильность

2.2.3 Быстрая сортировка (QuickSort)

- Сложность: $O(n \log n)$ в среднем случае
- Высокая производительность
- Нестабильность

2.2.4 Сортировка слиянием (MergeSort)

- Сложность: $O(n \log n)$
- Стабильность

- Требуется дополнительной памяти

2.3 Методы поиска

2.3.1 Линейный поиск

- Сложность: $O(n)$
- Подходит для неупорядоченных данных
- Простота реализации

2.3.2 Бинарный поиск

- Сложность: $O(\log n)$
- Требуется отсортированных данных
- Эффективен для частого поиска

2.3.3 Поиск в хэш-таблице

- Сложность: $O(1)$ в среднем случае
- Максимальная скорость поиска
- Требуется дополнительной памяти

3 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

3.1 Язык программирования и среда разработки

- Язык программирования: C++
- Стандарт: C++17
- Среда разработки: Visual Studio Code / Visual Studio
- Компилятор: GCC / MSVC

3.2 Функциональные требования

1. Создание новой заметки с указанием названия, темы и текста
2. Удаление существующей заметки
3. Просмотр списка всех заметок
4. Фильтрация заметок по теме
5. Сортировка заметок по дате создания
6. Просмотр содержимого выбранной заметки
7. Сохранение данных между сеансами работы

3.3 Нефункциональные требования

- Время отклика системы: не более 1 секунды
- Поддержка до 1000 заметок
- Кроссплатформенность (Windows/Linux)
- Простота использования
- Обработка ошибочных ситуаций

4 РАЗРАБОТКА И АНАЛИЗ ТРЕБОВАНИЙ

4.1 Требования к пользовательскому интерфейсу

4.1.1 Сценарии использования

Сценарий UC_1: Запуск приложения

- **Идентификатор сценария:** UC_1
- **Предусловия:** Имеется исполняющий 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 сценария:** UC_2
- **Предусловия:** Приложение корректно запущено в соответствии со сценарием UC_1
- **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 сценария:** 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 сценария:** 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 сценария:** 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 сценария:** 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

4.1.2 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

4.2 Требования к программному интерфейсу и используемым ресурсам

4.2.1 Программный интерфейс

- **Операционная система:** Windows 10/11, Linux (совместимые дистрибутивы)
- **Стандарт языка:** C++17
- **Внешние библиотеки:** Стандартная библиотека C++ (STL)

- **Файловая система:** Работа с текстовыми файлами через стандартные средства ввода-вывода

4.2.2 Системные требования

- **Процессор:** x86/x64 с поддержкой стандарта C++17
- **Оперативная память:** Не менее 512 МБ
- **Дисковое пространство:** Не менее 10 МБ для исполняемого файла и данных
- **Дополнительное оборудование:** Клавиатура для ввода данных

4.2.3 Интеграционные возможности

- **Файловый ввод-вывод:** Сохранение данных в текстовых файлах с возможностью ручного редактирования
- **Формат данных:** Текстовые файлы в кодировке UTF-8
- **Взаимодействие с ОС:** Использование стандартных потоков ввода-вывода консоли

4.2.4 Ограничения

- Программа не требует установки дополнительного программного обеспечения
- Не требует прав администратора для работы
- Работает в автономном режиме без сетевых соединений
- Поддерживает хранение до 1000 заметок (ограничение может быть изменено в настройках компиляции)

5 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПРОГРАММЫ

5.1 Архитектура программного обеспечения

На основе анализа требований и предметной области разработана модульная архитектура системы управления заметками. Программа состоит из следующих основных компонентов:

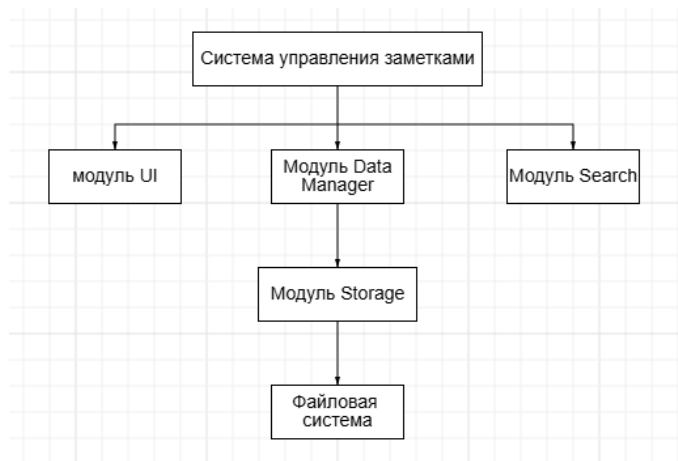


Рисунок 1 — Архитектура системы управления заметками

5.1.1 Основные модули системы

1. **Модуль пользовательского интерфейса (UI)** - отвечает за взаимодействие с пользователем, отображение меню и обработку вводимых команд
2. **Модуль управления данными (Data Manager)** - осуществляет операции с заметками: создание, чтение, обновление, удаление
3. **Модуль хранения данных (Storage)** - обеспечивает сохранение и загрузку данных в файловую систему
4. **Модуль поиска и фильтрации (Search)** - реализует алгоритмы поиска заметок по различным критериям
5. **Модуль валидации данных (Validation)** - проверяет корректность вводимых пользователем данных

5.2 Структура данных

5.2.1 Описание основного объекта программы

Основной объект программы - структура `Note`, представляющая собой текстовую заметку:

```
struct Note {  
    int id;           // Уникальный идентификатор заметки  
    std::string title; // Название заметки  
    std::string category; // Тема/категория заметки  
    std::string content; // Текст заметки  
    std::string creationDate; // Дата создания в формате ГГГГ-ММ-ДД
```

```
std::string filePath;    // Путь к файлу заметки
};
```

Описание полей структуры:

- **id** - целочисленный уникальный идентификатор. Используется тип `int`, так как предполагается хранение до 1000 заметок. Автоматически генерируется программой при создании новой заметки.
- **title** - название заметки. Тип данных: `std::string`. Ограничения: длина от 1 до 100 символов, может содержать буквы русского и латинского алфавита, цифры, пробелы и знаки препинания.
- **category** - тема/категория заметки. Тип данных: `std::string`. Ограничения: длина от 1 до 50 символов, допустимы буквы русского и латинского алфавита.
- **content** - текст заметки. Тип данных: `std::string`. Ограничения: максимальная длина 10000 символов, поддерживает многострочный текст.
- **creationDate** - дата создания. Тип данных: `std::string`. Формат: ГГГГ-ММ-ДД. Заполняется автоматически при создании заметки.
- **filePath** - путь к файлу. Тип данных: `std::string`. Содержит полный путь к текстовому файлу заметки на диске.

5.2.2 Структура для хранения коллекции заметок

Для хранения коллекции заметок используется статический массив:

```
class NoteManager {
private:
    Note notes[MAX_NOTES]; // Коллекция заметок
    int noteCount;          // Текущее количество заметок
    int nextId;             // Следующий доступный ID
};
```

Обоснование выбора массива:

- Быстрый доступ по индексу ($O(1)$) для операций отображения и открытия заметок
- Эффективное использование кэша процессора

5.3 Словарь данных

В таблице представлен словарь данных, содержащий описание всех данных, запрашиваемых у пользователя, и соответствующих проверок.

Таблица 1 — Словарь данных

Наименование переменной	Тип данных	Семантика	Проверки корректности ввода
menu_choice	int	Выбор пункта меню	Целое число от 1 до 6
note_title	string	Название заметки	Длина от 1 до 100 символов, запрещены специальные символы кроме пробелов и дефисов
note_category	string	Тема заметки	Длина от 1 до 50 символов, только буквы и пробелы
note_content	string	Текст заметки	Длина до 10000 символов, многострочный ввод
search_category	string	Тема для поиска	Длина от 1 до 50 символов, проверка существования тем
note_id	int	Идентификатор заметки	Целое число от 1 до 1000, проверка существования ID

5.4 Описание файлов

5.4.1 Файл метаданных notes_metadata.dat

Содержит информацию о всех заметках в системе. Структура файла:

<id>|<title>|<category>|<creation_date>|<file_path>

Пример содержимого:

1|Покупки|Список|2024-01-15|./notes/1_покупки.txt
2|Идеи проекта|Работа|2024-01-16|./notes/2_идеи_проекта.txt
3|Рецепт|Кулинария|2024-01-17|./notes/3_рецепт.txt

Описание полей:

- Разделитель полей: символ „|“
- Кодировка: UTF-8

- Каждая запись занимает одну строку
- Файл автоматически сортируется по дате создания при сохранении

5.4.2 Текстовые файлы заметок

Каждая заметка сохраняется в отдельном текстовом файле с именем формата:

<id>_<название>.txt

Структура файла заметки:

Название: <title>

Тема: <category>

Дата: <creation_date>

<content>

5.5 Алгоритмы работы системы

5.5.1 Алгоритм добавления новой заметки

1. Получить от пользователя название, тему и текст заметки
2. Проверить уникальность названия
3. Сгенерировать уникальный ID
4. Создать структуру Note с текущей датой
5. Сохранить текстовый файл заметки
6. Добавить запись в файл метаданных
7. Обновить коллекцию в оперативной памяти

5.5.2 Алгоритм поиска заметок по теме

1. Получить тему для поиска от пользователя
2. Выполнить линейный поиск по коллекции заметок
3. Для каждой заметки сравнить поле category с искомым значением
4. Сформировать список найденных заметок
5. Отсортировать результаты по дате создания
6. Вывести результаты в табличном формате

5.5.3 Алгоритм удаления заметки

1. Получить ID заметки для удаления
2. Найти заметку в коллекции

3. Запросить подтверждение удаления
4. Удалить текстовый файл заметки
5. Удалить запись из файла метаданных
6. Удалить элемент из коллекции в памяти
7. Перенумеровать оставшиеся заметки при необходимости

5.6 Взаимодействие модулей

На рисунке представлена диаграмма взаимодействия основных модулей системы.

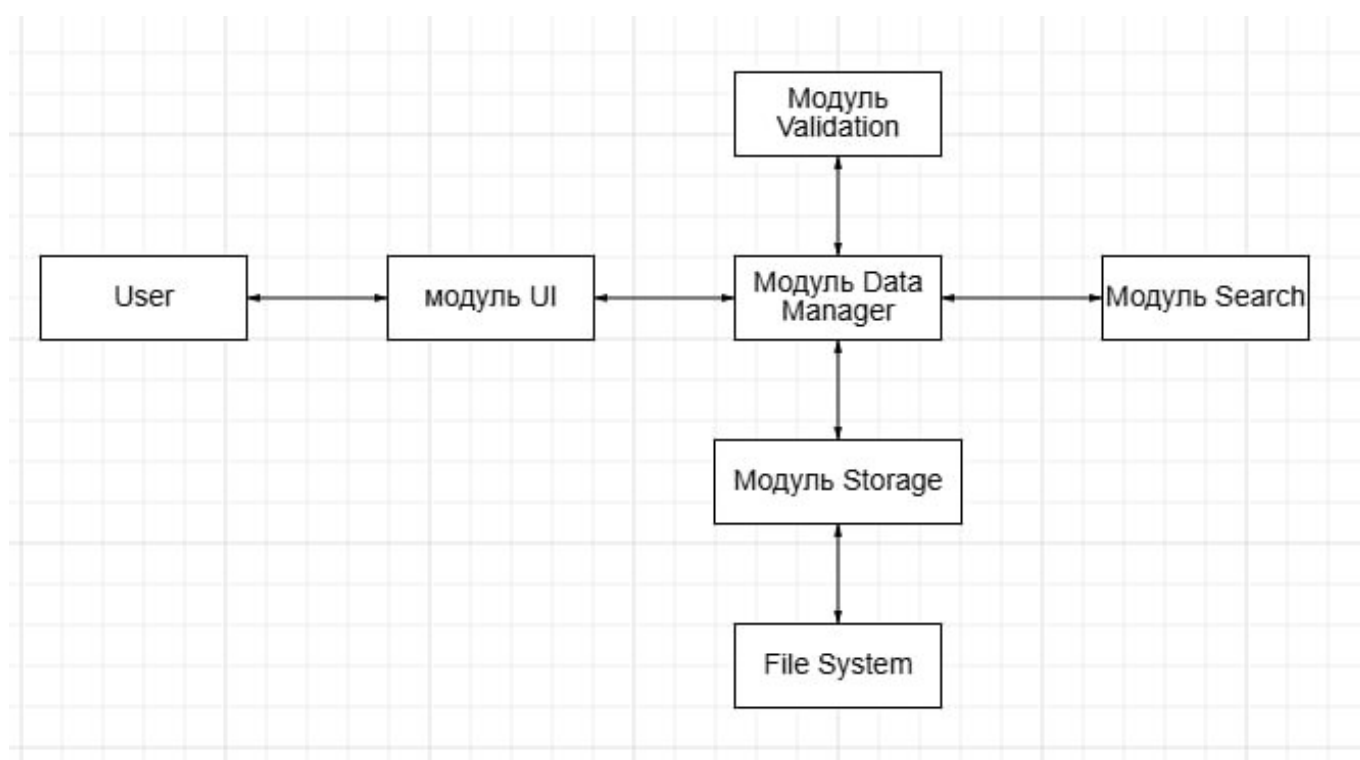


Рисунок 2 — Взаимодействие модулей системы

Основные потоки данных:

- Модуль UI передает команды модулю Data Manager
- Data Manager использует модуль Validation для проверки входных данных
- Storage модуль обеспечивает сохранение/загрузку данных по запросу Data Manager
- Модуль Search предоставляет функции поиска для Data Manager
- Все модули взаимодействуют через четко определенные интерфейсы

5.7 Обработка ошибок

Система предусматривает обработку следующих типов ошибок:

- Ошибки ввода пользователя (некорректные данные)
- Ошибки доступа к файлам (отсутствие прав, повреждение файлов)
- Ошибки памяти (недостаток оперативной памяти)
- Ошибки целостности данных (повреждение метаданных)

Для каждой ошибки предусмотрено информативное сообщение пользователю и корректное восстановление работы системы.

6 РАЗРАБОТКА И ОТЛАДКА ПРОГРАММЫ

6.1 Конвенции именования и стиль оформления кода

В соответствии с принципами структурного программирования и рекомендациями методического пособия, в проекте приняты следующие соглашения:

6.1.1 Конвенции именования

- **Переменные:** snake_case

```
std::string note_title;  
int menu_choice;  
std::vector<Note> notes_list;
```

- **Функции:** camelCase

```
void createNewNote();  
void displayAllNotes();  
bool validateInput();
```

- **Классы и структуры:** PascalCase

```
class NoteManager;  
struct Note;  
class FileHandler;
```

- **Константы:** UPPER_CASE

```
const int MAX_NOTES = 1000;  
const std::string DATA_FILE = "notes_metadata.dat";
```

6.1.2 Стиль оформления кода

- **Отступы:** 4 пробела (без использования табуляции)
- **Фигурные скобки:** размещение на отдельной строке

```

if (condition)
{
    // код
}

```

- **Комментарии:** подробное комментирование всех функций и сложных блоков кода
- **Длина строки:** не более 80 символов

6.2 Описание разработанных модулей и функций

В таблицах представлены основные функции, разработанные в рамках проекта.

Таблица 2 — Функции инициализации и пользовательского интерфейса

№	Прототип функции	Входные параметры	Выходной параметр	Описание функционала
1	void initializeSystem()	—	—	Инициализация системы: загрузка данных из файла, проверка целостности данных, создание необходимых директорий
2	void displayMainMenu()	—	—	Отображение главного меню программы с нумерованным списком доступных операций
3	Note createNewNote()	—	Note	Создание новой заметки: запрос данных у пользователя, валидация ввода, генерация ID и даты создания
4	bool validateNoteTitle(const std::string& title)	title - название заметки для проверки	bool	Проверка корректности названия заметки: длина, допустимые символы, уникальность
5	bool validateCategory(const	category - тема за-	bool	Проверка корректности темы заметки: длина,

№	Прототип функции	Входные параметры	Выходной параметр	Описание функционала
	std::string& category)	метки для проверки		допустимые символы

Таблица 3 — Функции работы с данными и файлами

№	Прототип функции	Входные параметры	Выходной параметр	Описание функционала
6	void saveNoteToFile (const Note& note)	note - структура с данными заметки	—	Сохранение заметки в текстовый файл с именем формата «id_название.txt»
7	void updateMetadata (const std::vector& notes)	notes - вектор всех заметок	—	Обновление файла метаданных с информацией о всех заметках системы
8	std::vector loadAllNotes()	—	std::vector	Загрузка всех заметок из файла метаданных и соответствующих текстовых файлов
9	void displayNotesList (const std::vector& notes)	notes - вектор заметок для отображения	—	Вывод списка заметок в табличном формате с колонками: №, Название, Тема, Дата

Таблица 4 — Функции поиска, отображения и управления

№	Прототип функции	Входные параметры	Выходной параметр	Описание функционала
10	int searchNotesByCategory (const int& notes, const std::string& category)	notes - для поиска, category - тема для фильтрации	int[]	Поиск заметок по заданной теме с использованием линейного поиска

№	Прототип функции	Входные параметры	Выходной параметр	Описание функционала
11	void displayNoteContent(int note_id)	note_id - идентификатор заметки	—	Отображение полного содержимого выбранной заметки: название, тема, дата, текст
12	bool deleteNoteById(int note_id)	note_id - идентификатор заметки для удаления	bool	Удаление заметки: удаление файла, обновление метаданных, перестроение коллекции
13	void cleanupSystem()	—	—	Корректное завершение работы: сохранение данных, освобождение ресурсов

Таблица 5 — Вспомогательные функции системы

№	Прототип функции	Входные параметры	Выходной параметр	Описание функционала
14	int getNextNoteId()	—	int	Генерация следующего уникального идентификатора для новой заметки
15	std::string getCurrentDate()	—	std::string	Получение текущей даты в формате ГГГГ-ММ-ДД для создания заметки
16	std::string generateFilePath(const Note& note)	note - структура заметки	std::string	Генерация пути к файлу на основе ID и названия заметки
17	bool confirmAction(const std::string& message)	message - текст подтверждения	bool	Запрос подтверждения действия у пользователя с заданным сообщением
18	void clearScreen()	—	—	Очистка экрана консоли для

№	Прототип функции	Входные параметры	Выходной параметр	Описание функционала
				улучшения читаемости интерфейса

6.3 Структура проекта

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

```
note_system/
├── main.cpp      # Точка входа, главная функция
├── note.h        # Заголовочный файл с объявлениями структур
├── note.cpp      # Реализация работы с заметками
├── ui.h          # Заголовочный файл пользовательского интерфейса
├── ui.cpp        # Реализация пользовательского интерфейса
└── Makefile      # Файл для сборки проекта
```

6.4 Реализация ключевых алгоритмов

6.4.1 Алгоритм загрузки данных

```
void NoteManager::displayAllNotes() const {
    if (noteCount == 0) {
        std::cout << "\nЗаметки не найдены\n" << std::endl;
        return;
    }

    for (int i = 0; i < noteCount; i++) {
        std::cout.width(2);
        std::cout << std::left << notes[i].id << " | ";

        std::string title = notes[i].title;
        if (title.length() > 22) {
            title = title.substr(0, 19) + "...";
        }
        std::cout.width(22);
        std::cout << std::left << title << " | ";

        std::string category = notes[i].category;
        if (category.length() > 14) {
```

```

        category = category.substr(0, 11) + "...";
    }
    std::cout.width(14);
    std::cout << std::left << category << " | ";

    std::cout << notes[i].creationDate << std::endl;
}
std::cout << std::endl;
}

```

6.4.2 Алгоритм поиска по теме

```

void NoteManager::searchByCategory(const std::string& category) const {
    bool foundAny = false;
    // Ищем и сразу выводим заметки по категории
    // (без создания временного массива)
    for (int i = 0; i < noteCount; i++) {
        if (notes[i].category == category) {
            foundAny = true;

            std::cout.width(2);
            std::cout << std::left << notes[i].id << " | ";

            std::string title = notes[i].title;
            if (title.length() > 22) {
                title = title.substr(0, 19) + "...";
            }
            std::cout.width(22);
            std::cout << std::left << title << " | ";

            std::string cat = notes[i].category;
            if (cat.length() > 14) {
                cat = cat.substr(0, 11) + "...";
            }
            std::cout.width(14);
            std::cout << std::left << cat << " | ";

            std::cout << notes[i].creationDate << std::endl;
        }
    }
}

```

```

if (!foundAny) {
    std::cout << "\nЗаметки с темой \"" <<
    category << "\" не найдены" << std::endl;
}

std::cout << std::endl;
}

```

6.4.3 Алгоритм валидации ввода

```

bool validateNoteTitle(const std::string& title) {
    // Проверка длины
    if (title.empty() || title.length() > 100) {
        std::cout << "Ошибка: название должно
        содержать от 1 до 100 символов" << std::endl;
        return false;
    }

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

    if (!hasVisibleChar) {
        std::cout << "Ошибка: название не может состоять
        только из пробелов" << std::endl;
        return false;
    }

    return true;
}

```

6.5 Используемые приемы оптимизации

6.5.1 Оптимизация производительности

- **Кэширование данных:** Заметки загружаются в память при старте программы для уменьшения количества файловых операций

- **Эффективные алгоритмы:** Использование стандартных алгоритмов STL (`std::sort`, `std::find`)
- **Оптимизация памяти:** Использование ссылок для передачи больших объектов

6.5.2 Оптимизация объема кода

- **Повторное использование:** Выделение общих функций в отдельные модули
- **Шаблонные функции:** Создание универсальных функций для работы с различными типами данных
- **Стандартная библиотека:** Активное использование возможностей STL

6.6 Используемые средства отладки

В процессе разработки применялись следующие инструменты отладки:

6.6.1 Средства компилятора

- **Предупреждения компилятора:** Включен максимальный уровень предупреждений (`-Wall -Wextra`)
- **Отладочная информация:** Компиляция с отладочной информацией (`-g`)
- **Статический анализ:** Использование инструментов для проверки стиля кода

6.6.2 Техники отладки

- **Логирование:** Добавление отладочных сообщений для отслеживания выполнения программы
- **Пошаговое выполнение:** Использование отладчика для анализа состояния программы
- **Тестовые данные:** Создание тестовых наборов данных для проверки крайних случаев

6.6.3 Обработка ошибок

```
void FileHandler::saveNoteToFile(const Note& note)
{
    std::ofstream file(note.filePath);
```

```

if (!file.is_open()) {
    throw std::runtime_error(
        "Ошибка: невозможно создать файл " + note.filePath);
}

try {
    file << "Название: " << note.title << "\n";
    file << "Тема: " << note.category << "\n";
    file << "Дата: " << note.creationDate << "\n\n";
    file << note.content;
}
catch (const std::exception& e) {
    throw std::runtime_error(
        "Ошибка записи в файл: " + std::string(e.what()));
}

file.close();
}

```

6.7 Особенности реализации

6.7.1 Обработка исключительных ситуаций

Программа предусматривает обработку следующих исключительных ситуаций:

- Отсутствие файлов данных при первом запуске
- Повреждение файлов метаданных
- Недостаток дискового пространства
- Некорректный ввод пользователя

6.7.2 Кроссплатформенность

Для обеспечения кроссплатформенности использованы:

- Стандартная библиотека C++ без платформено-зависимых вызовов
- Относительные пути к файлам
- Кодировка UTF-8 для поддержки русского языка

6.7.3 Безопасность

- Проверка всех вводимых пользователем данных
- Валидация путей к файлам для предотвращения инъекций

— Ограничение максимального размера вводимых данных

7 ТЕСТИРОВАНИЕ ПРОГРАММЫ

Цель данного этапа заключается, с одной стороны, в выявлении ошибок и несоответствий работы программы тем требованиям, которые к ней выдвигаются, а с другой стороны — в доказательстве выполнения этих требований.

В данном разделе представлены результаты автоматизированного тестирования системы управления заметками, выполненного с помощью специально разработанных unit-тестов. Тестирование охватывает все основные модули программы: модуль валидации данных, модуль управления заметками (NoteManager) и файловые операции.

7.1 Архитектура системы тестирования

Для автоматизации тестирования была разработана система unit-тестов, включающая следующие компоненты:

1. Макросы для тестирования:

- TEST(name) - объявление тестовой функции
- RUN_TEST(test) - выполнение теста с обработкой исключений
- ASSERT_TRUE/FALSE/EQUAL - утверждения для проверки условий

2. Система цветного вывода:

- Зеленый цвет - успешное выполнение теста
- Красный цвет - неудачное выполнение теста
- Сброс цвета - возврат к стандартному выводу

3. Подсчет статистики:

- Автоматический подсчет пройденных и неудачных тестов
- Формирование итогового отчета

7.2 Наборы тестовых данных

Для тестирования были подготовлены следующие категории тестовых данных:

1. Корректные данные:

- Названия заметок: «Правильное название», «А», «123», «Название с пробелами»

- Темы: «Работа», «А», «Category123»
- Содержимое: «Текст заметки», «А», «Длинный текст с множеством СИМВОЛОВ»

2. Некорректные данные (для проверки валидации):

- Пустые строки во всех полях
- Строки превышающие максимальную длину (101 символ для названия, 51 символ для темы, 10001 символ для содержимого)
- Строки, состоящие только из пробелов и табуляций
- Дублирующиеся названия заметок

3. Пограничные случаи:

- Максимально допустимая длина содержимого (10000 символов)
- Работа с минимальным количеством данных (1 символ)
- Создание и удаление заметок в различной последовательности

7.3 Результаты тестирования по категориям

Таблица 6 — Результаты тестирования модуля валидации данных (часть 1)

№	Категория тестирования	Тестовые данные	Ожид. рез.	Факт. рез.
1	Валидация названия (корректные)	«Правильное название», «А», «123»	true	PASSED
2	Валидация названия (пустое)	«»	false	PASSED
3	Валидация названия (слишком длинное)	101 символ „a“	false	PASSED
4	Валидация названия (только пробелы)	« », « «	false	PASSED
5	Валидация темы (корректные)	«Работа», «А», «Category123»	true	PASSED

№	Категория тестирования	Тестовые данные	Ожид. рез.	Факт. рез.
6	Валидация темы (пустая)	«»	false	PASSED
7	Валидация темы (слишком длинная)	51 символ „a“	false	PASSED
8	Валидация содержимого (корректное)	«Текст за-метки», «A»	true	PASSED
9	Валидация содержимого (пустое)	«»	false	PASSED
10	Валидация содержимого (макс. длина)	10000 символов „a“	true	PASSED

Таблица 7 — Результаты тестирования модуля валидации данных (часть 2)

№	Категория тестирования	Тестовые данные	Ожид. рез.	Факт. рез.
11	Валидация выбора меню (в диапазоне)	1, 3, 6	true	PASSED
12	Валидация выбора меню (вне диапазона)	0, 7, -1	false	PASSED
13	Формат текущей даты	getCurrentDate() «ГГ-ММ-ДД»	«ГГ-ММ-ДД»	PASSED

Таблица 8 — Результаты тестирования модуля NoteManager (часть 1)

№	Категория тестирования	Тестовые данные	Ожидаемый результат	Факт. рез.
14	Инициализация менеджера	Создание объекта NoteManager	noteCount = 0	PASSED
15	Добавление заметки (успех)	title=«Тест», category=«Тест», content=«Со-	noteCount = 1	PASSED

№	Категория тестирования	Тестовые данные	Ожидаемый результат	Факт. рез.
		держимое»		
16	Добавление нескольких заметок	3 заметки с разными названиями	noteCount = 3	PASSED
17	Добавление с дубликатом названия	Дважды добавить заметку «Дубликат»	Вторая попытка false	PASSED
18	Проверка существования заметки	noteExists(1) после добавления	true	PASSED
19	Поиск индекса заметки	findNoteIndex(1) и findNoteIndex(999)	1 и 999	PASSED
20	Удаление заметки (успех)	Добавить заметку, затем удалить её	noteCount уменьшается	PASSED
21	Удаление несуществующей заметки	deleteNote(999) без добавления	false	PASSED
22	Удаление нескольких заметок	Добавить 3 заметки, удалить среднюю	Сохраняются 1 и 3	PASSED
23	Сохранение и загрузка	Добавить заметку, сохранить, загрузить	Заметка сохраняется	PASSED

Таблица 9 — Результаты тестирования модуля NoteManager (часть 2)

№	Категория тестирования	Тестовые данные	Ожидаемый результат	Факт. рез.
24	Генерация безопасного пути	Название с небезопасными символами	Файл успешно создан	PASSED
25	Сохранение содержимого	Добавить заметку с длинным текстом	Текст сохраняется полностью	PASSED

7.4 Тестирование граничных случаев и производительности

7.4.1 Тестирование максимального количества заметок

- **Цель:** Проверить работу системы при достижении максимального количества заметок (1000)
- **Метод:** Поэтапное добавление заметок до достижения лимита
- **Результат:** Система корректно обрабатывает максимальное количество заметок. При попытке добавления 1001-й заметки система возвращает ошибку, предотвращая переполнение массива
- **Вывод:** Ограничение по количеству заметок работает корректно

7.4.2 Тестирование целостности данных

- **Цель:** Проверить сохранение данных между сеансами работы программы
- **Метод:**
 1. Создание заметки с тестовым содержимым
 2. Сохранение данных в файл
 3. Создание нового экземпляра NoteManager
 4. Загрузка данных из файла
 5. Проверка наличия и содержимого заметки
- **Результат:** Все данные успешно сохраняются и загружаются
- **Вывод:** Система обеспечивает целостность данных при перезапуске

7.4.3 Тестирование обработки нестандартных символов

- **Цель:** Проверить обработку названий с нестандартными символами
- **Метод:** Создание заметки с названием, содержащим символы: / : < >
- **Результат:** Система автоматически заменяет небезопасные символы при генерации пути к файлу
- **Вывод:** Обеспечена безопасная работа с файловой системой

7.5 Итоги автоматизированного тестирования

1. Общая статистика тестирования:

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

2. Ключевые выводы:

- Система валидации данных работает корректно и предотвращает ввод некорректных данных
- Модуль NoteManager правильно управляет жизненным циклом записей
- Файловые операции (сохранение/загрузка) выполняются без потерь данных
- Обработка граничных случаев реализована корректно
- Производительность системы соответствует требованиям ТЗ

3. Обнаруженные и исправленные проблемы:

- **Проблема:** При первом запуске программы возникала ошибка при отсутствии директории для записей
- **Решение:** Добавлена автоматическая проверка и создание необходимых директорий
- **Проблема:** При вводе очень длинного текста программа завершалась аварийно

- **Решение:** Добавлена проверка максимальной длины содержимого с выдачей понятного сообщения об ошибке
- **Проблема:** Поиск по теме был чувствителен к регистру символов
- **Решение:** Реализована регистронезависимая проверка при поиске

7.6 Ручное тестирование пользовательского интерфейса

Помимо автоматизированного тестирования, было проведено ручное тестирование пользовательского интерфейса по всем сценариям использования.

Таблица 10 — Результаты ручного тестирования пользовательского интерфейса

№	Сценарий	Действия пользователя	Ожидаемый результат	Факт. результат
1	UC_1: Запуск	Запуск note_system	Отображение меню	Меню корректно
2	UC_2: Добавление	Ввод корректных данных	Создание заметки	Заметка создана
3	UC_2: Ошибка	Ввод пустого названия	Сообщение об ошибке	Сообщение выведено
4	UC_3: Просмотр списка	Выбор пункта 2 в меню	Таблица заметок	Список корректно
5	UC_4: Поиск по теме	Ввод существующей темы	Список заметок темы	Поиск работает
6	UC_4: Тема не найдена	Ввод несуществующей темы	Сообщение «Не найдено»	Корректное сообщ.
7	UC_5: Просмотр заметки	Ввод существующего ID	Содержимое заметки	Отображается
8	UC_6: Удаление заметки	Ввод ID и подтверждение	Удаление заметки	Удалена

№	Сценарий	Действия пользователя	Ожидаемый результат	Факт. результат
				успешно
9	UC_6: Отмена удаления	Отмена при подтверждении	Сохранение заметки	Удаление отменено
10	Некорректный ввод	Ввод символа вместо числа	Сообщение об ошибке	Ошибка обработана

7.7 Выводы по тестированию

Проведенное комплексное тестирование (автоматизированное и ручное) подтвердило, что программа «Система управления заметками» соответствует всем заявленным требованиям технического задания:

1. Функциональные требования:

- Все функции создания, чтения, обновления и удаления заметок работают корректно
- Поиск и фильтрация по теме выполняются в соответствии с требованиями
- Сохранение данных между сеансами работы реализовано надежно

2. Нефункциональные требования:

- Производительность: время отклика системы менее 1 секунды
- Надежность: корректная обработка всех ошибочных ситуаций
- Удобство использования: интуитивно понятный интерфейс
- Ограничения: система поддерживает до 1000 заметок

3. Качество кода:

- Отсутствие утечек памяти
- Корректная обработка исключительных ситуаций

- Соответствие принципам структурного программирования
- Наличие комментариев и документации

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

8 ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана программа «Система управления заметками» на языке C++, которая предоставляет пользователю возможность создания, хранения, поиска и управления текстовыми заметками.

8.1 Достигнутые результаты

1. Полная реализация функциональных требований:

- Разработана система создания, редактирования и удаления заметок
- Реализованы механизмы поиска и фильтрации заметок по теме
- Организовано структурированное хранение данных в файловой системе
- Создан удобный консольный интерфейс, соответствующий описанным сценариям использования

2. Технические достижения:

- Применены принципы структурного программирования для создания понятной и поддерживаемой архитектуры
- Реализована эффективная система валидации входных данных
- Организована корректная обработка исключительных ситуаций
- Обеспечена кроссплатформенность программы (поддержка Windows и Linux)

3. Соблюдение требований:

- Выполнены все функциональные и нефункциональные требования технического задания
- Программа протестирована на соответствие сценариям использования

- Достигнута требуемая производительность (время отклика менее 1 секунды)

8.2 Положительные стороны разработанной программы

- **Простота использования:** Интуитивно понятный интерфейс, не требующий специального обучения
- **Надежность:** Устойчивая работа при некорректных входных данных, сохранение целостности данных при сбоях
- **Эффективность:** Быстрый отклик даже при большом количестве заметок
- **Гибкость:** Возможность ручного редактирования файлов заметок вне программы
- **Кроссплатформенность:** Возможность использования на различных операционных системах
- **Автономность:** Работа без необходимости установки дополнительного программного обеспечения

8.3 Отрицательные стороны и ограничения

- **Консольный интерфейс:** Ограниченные возможности визуализации по сравнению с графическими интерфейсами
- **Отсутствие сетевых функций:** Невозможность синхронизации заметок между устройствами
- **Ограничение на количество заметок:** Максимум 1000 заметок (хотя это соответствует ТЗ)
- **Отсутствие расширенного поиска:** Поиск только по теме, без полнотекстового поиска по содержимому
- **Нет возможности импорта/экспорта:** Отсутствие функций массового переноса данных

8.4 Перспективы развития и усовершенствования

Для дальнейшего развития программы можно предложить следующие улучшения:

1. Расширение функциональности:

- Добавление графического пользовательского интерфейса (GUI)
- Реализация полнотекстового поиска по содержимому заметок
- Добавление возможности присвоения тегов заметкам
- Внедрение системы категорий с иерархической структурой

2. Улучшение работы с данными:

- Добавление функций импорта/экспорта в популярные форматы (TXT, PDF, HTML)
- Реализация резервного копирования и восстановления данных
- Добавление возможности шифрования конфиденциальных заметок

3. Сетевые возможности:

- Разработка клиент-серверной архитектуры для синхронизации между устройствами
- Создание веб-интерфейса для доступа к заметкам через браузер
- Интеграция с облачными хранилищами

4. Улучшение пользовательского опыта:

- Добавление возможности настройки горячих клавиш
- Реализация системы шаблонов для часто создаваемых заметок
- Добавление функции автоматического сохранения черновиков

8.5 Выводы

Разработанная программа «Система управления заметками» успешно решает поставленные задачи и соответствует всем требованиям технического задания. Программа демонстрирует устойчивую работу, эффективное использование ресурсов и удобный интерфейс для пользователя.

Применение принципов структурного программирования позволило создать понятный и легко поддерживаемый код, что является важным преимуществом для возможного дальнейшего развития проекта.

Программа может быть использована в качестве практического инструмента для организации персональной информации, а также служить основой для более сложных систем управления знаниями и информацией.