**Российская федерация**

**Ханты-Мансийский автономный округ – Югра**

**Департамент образования и науки**

**Сургутский государственный университет ХМАО**

Политехнический институт

Кафедра Автоматики и компьютерных систем

**Пояснительная записка**

к курсовому проекту

по дисциплине Структурное программирование

Выполнил: студентка группы 609-31

Шильников К.Е.

Проверил: доцент кафедры Автоматики и компьютерных систем

Гришмановский П.В.

**Сургут**

**2024**

**Задание**

Консольный блокнот

Программа с консольным интерфейсом для быстрого создания текстовых заметок и их просмотра. Дата и время создания каждой заметки фиксируются автоматически. Заметки могут быть разбиты на категории. Файл (файлы) данных программы должны храниться в одной папке с исполняемым файлом и корректно открываться при перемещении папки, использовании для запуска программы ярлыка или переменной окружения PATH и т.п.

**Аннотация**

**Тема курсового проекта: Консольный блокнот**

Выполнил: Шильников К. Е.

В курсовом проекте представлена разработка программного обеспечения "Консольный блокнот", предназначенного для управления текстовыми заметками через командную строку. Работа состоит из введения, анализа, проектирования, кодирования, тестирования, сопровождения и заключения.

Во введении описаны цели и задачи проекта, а также обоснована актуальность и значимость разработки "Консольного блокнота".

В разделе анализ проведен обзор существующих текстовых редакторов и блокнотов, выполнен сравнительный анализ их функциональности и удобства использования, а также определены требования к разрабатываемому ПО.

В разделе проектирование получены проектные решения, на основе которых осуществлена реализация продукта.

В разделе кодирование описан код программного обеспечения, написанный в соответствии с проектом, а также используемые библиотеки и функции.

В разделе тестирования описаны выбранные методы и стратегии тестирования, разработанные тестовые сценарии, проведенное тестирование и полученные результаты.

В разделе сопровождение разработано руководство пользователя.

В заключении подведены итоги проекта, оценено достижение поставленных целей и задач, а также предложены направления дальнейшего развития и совершенствования программного обеспечения.

**Содержание**

[**Аннотация** 3](#_Toc169686997)

[**Введение** 5](#_Toc169686998)

[**Анализ** 6](#_Toc169686999)

[**Проектирование** 12](#_Toc169687000)

[**Кодирование** 17](#_Toc169687001)

[**Тестирование** 33](#_Toc169687002)

[**Сопровождение** 37](#_Toc169687003)

[**Заключение** 40](#_Toc169687004)

[**Список используемых источников.** 42](#_Toc169687005)

[**Приложение** 43](#_Toc169687006)

### Введение

В современном мире, где управление информацией играет ключевую роль, текстовые редакторы и блокноты стали незаменимыми инструментами. Они позволяют быстро и удобно создавать, редактировать и хранить текстовые данные.

Целью курсового проекта является разработка функционального и удобного в использовании "Консольного блокнота", способного выполнять основные операции по управлению текстовыми заметками через командную строку. Программа должна фиксировать дату и время создания каждой заметки, предоставлять возможность разбивки заметок на категории, а также корректно работать при перемещении файлов и использовании различных способов запуска.

### Анализ

##### **Контекст использования**

Программа "Консольный блокнот" предназначена для пользователей, которым необходимо быстро и эффективно создавать, редактировать и управлять текстовыми заметками через консольный интерфейс. Основные пользователи - это разработчики, системные администраторы и пользователи, предпочитающие работу в командной строке.

##### **Проблемы и задачи**

Основная проблема, которую решает программа, заключается в отсутствии удобного инструмента для создания и управления текстовыми заметками в консольной среде. Задачи программы включают:

* Создание новых заметок.
* Редактирование существующих заметок.
* Удаление заметок.
* Организация заметок по папкам.

##### **Целевая аудитория**

Целевая аудитория программы - это пользователи, которые активно работают в командной строке и нуждаются в инструменте для управления текстовыми заметками. Это могут быть разработчики, системные администраторы и технически подкованные пользователи.

**Обзор существующих решений (программ-аналогов)**

##### **Программа 1:** GNU **nano**

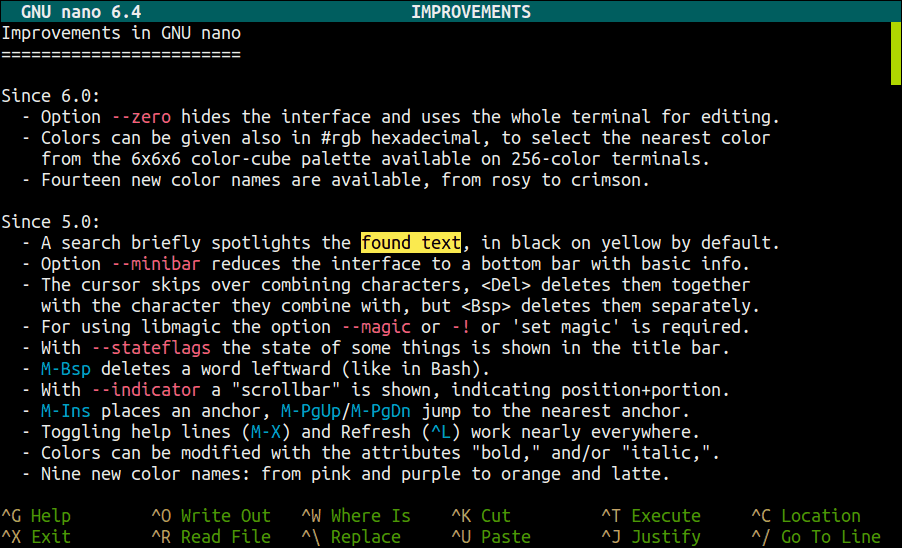


Рисунок 3.1 Редактор nano

* **Описание:** nano - это текстовый редактор для Unix-подобных систем, который позволяет создавать и редактировать текстовые файлы в консоли.
* **Достоинства:**
  + Простота использования.
  + Поддержка базовых функций редактирования текста.
* **Недостатки:**
  + Отсутствие функций для организации заметок по папкам.
  + Нет возможности поиска и фильтрации заметок.

##### **Программа 2: vim**

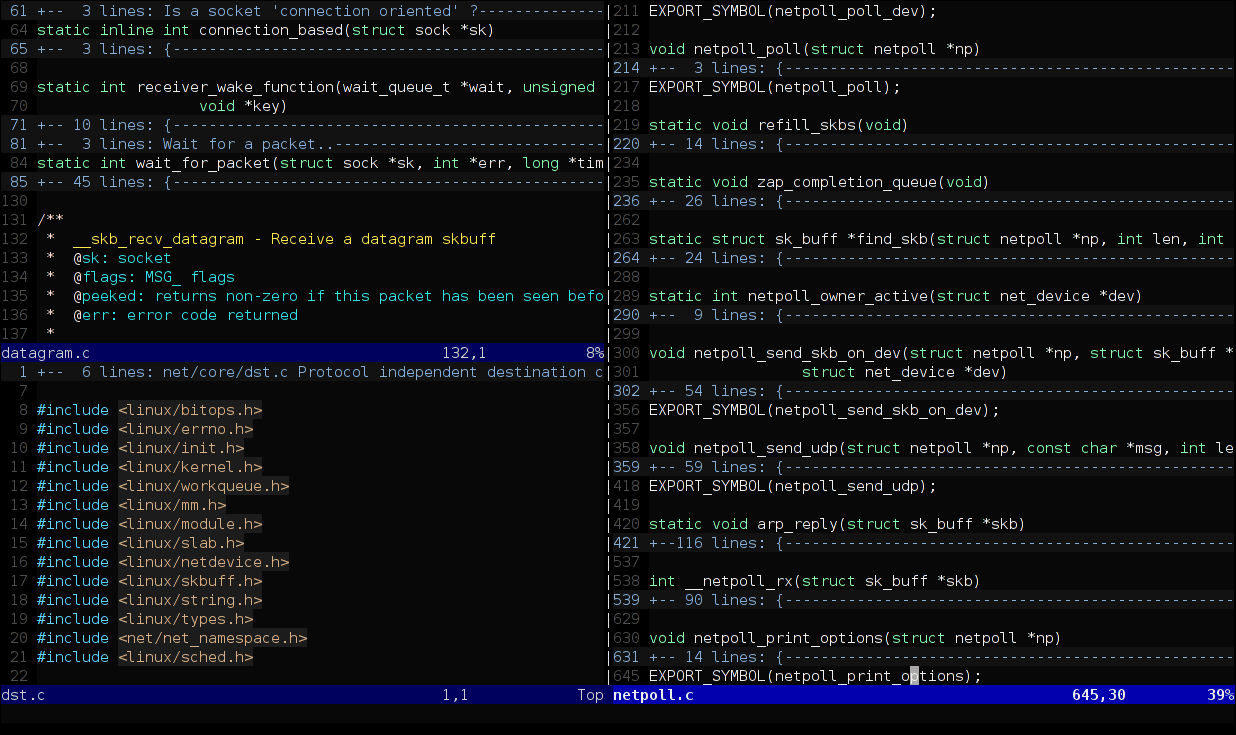


Рисунок 3.2 Редактор vim

* **Описание:** vim - это продвинутый текстовый редактор для Unix-подобных систем, предлагающий широкий спектр функций.
* **Достоинства:**
  + Мощные возможности редактирования текста.
  + Поддержка плагинов и скриптов.
* **Недостатки:**
  + Крутая кривая обучения.
  + Отсутствие встроенной функциональности для управления заметками как отдельными объектами.

##### **Программа 3: tomboy-ng**

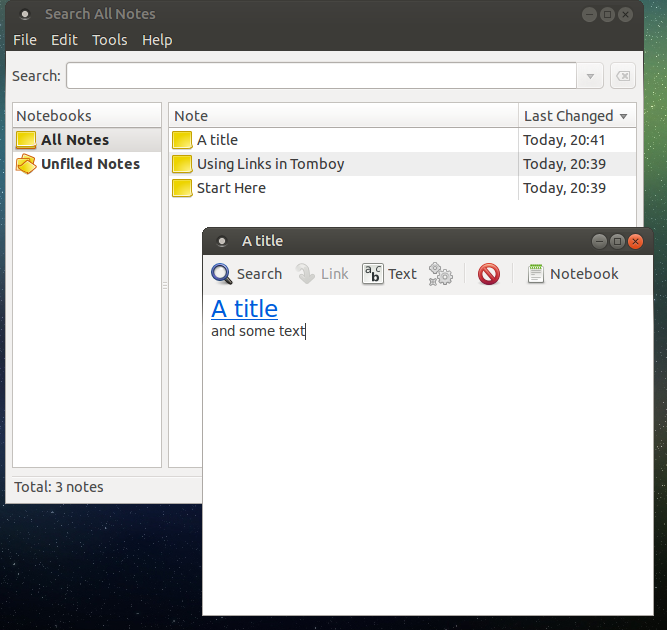


Рисунок 3.3 Редактор **tomboy-ng**

* **Описание:** tomboy-ng - это кросс-платформенный инструмент для создания заметок с графическим интерфейсом.
* **Достоинства:**
  + Удобный интерфейс.
  + Возможность организации заметок по категориям.
* **Недостатки:**
  + Не поддерживает работу в консоли.
  + Требует графическую среду для работы.

#### **Требования к разрабатываемой программе**

##### **Функции программы**

1. **Создание новых заметок:**
   * Пользователь должен иметь возможность создать новую заметку и сохранить её в указанной папке.
2. **Редактирование существующих заметок:**
   * Пользователь должен иметь возможность открыть и отредактировать существующую заметку.
3. **Удаление заметок:**
   * Пользователь должен иметь возможность удалить ненужные заметки.
4. **Организация заметок по папкам:**
   * Пользователь должен иметь возможность создавать папки для организации заметок.

##### **Описание состава обрабатываемых данных**

* **Заметка:** включает идентификатор, текст, дату создания и дату последнего редактирования.
* **Папка:** включает название и список заметок.

##### **Способ организации взаимодействия с пользователем**

* **Консольный интерфейс:** пользователи будут взаимодействовать с программой через команды в консоли.
* **Меню:** основной способ навигации и выполнения операций - через текстовое меню с выбором опций (в виде цифр).

##### **Требования к производительности и надёжности**

* Программа должна быстро обрабатывать команды пользователя.
* Программа должна быть устойчива к ошибкам ввода пользователя и обеспечивать сохранность данных заметок.

#### **Техническое задание**

Цель данного проекта - разработка консольного блокнота для создания, редактирования и управления текстовыми заметками.

##### **Общие требования**

* **Платформа:** Unix-подобные системы.
* **Язык программирования:** C.
* **Среда выполнения:** консоль/терминал.

##### **Функциональные требования**

1. Создание новых заметок.
2. Редактирование существующих заметок.
3. Удаление заметок.
4. Чтение заметок.
5. Организация заметок по папкам.

##### **Нефункциональные требования**

* Программа должна быть устойчива к ошибкам.
* Программа должна быть удобной для использования в консольной среде.
* Программа должна обеспечивать сохранность данных.

##### **Ограничения**

* Программа должна работать в консольной среде без графического интерфейса.
* Программа должна поддерживать только текстовые заметки без мультимедийного контента.

### Проектирование

Цель данного этапа — разработать проектные представления, описывающие структуру и функционирование программного продукта. Это включает проектирование структур данных и алгоритмов, которые будут использоваться в программе.

**Формальное описание задачи**

Программа "Консольный блокнот" предназначена для создания, чтения и организации текстовых заметок через консольный интерфейс.

**Цели и задачи:**

- Создание новой заметки.

- Открытие заметки

- Организация заметок по папкам.

**Поведенческая модель программного продукта**

1. Создание новой заметки:

- Пользователь выбирает опцию создания новой заметки.

- Вводит текст заметки.

- Вводит имя заметки.

- Сохраняет заметку в выбранной папке.

2. Создание новой папки:

- Пользователь выбирает опцию создания новой папки.

- Вводит имя папки.

- Сохраняет папку с заданным именем.

3. Открытие заметки:

- Пользователь выбирает опцию открытия заметки.

- Вводит индекс заметки из перечисленных.

- Видит содержимое заметки.

**Структура файлов**

Файлы заметок будут храниться в двоичном формате. Каждая заметка будет представлена отдельным .dat файлом. Папки будут представлены как директории, содержащие текстовые файлы заметок.

Пример:

/all\_notes

/work

note1.dat

note2.dat

/personal

note3.dat

**Представление данных в памяти**

Для хранения и управления заметками в памяти будут использоваться следующие структуры данных:

1. Заметка (Note):

- id: уникальный идентификатор.

- created\_at: дата создания.

- text: текст заметки.

Пример:

typedef struct {

int id;

char created\_timestamp[20];

char \*text;

} Note;

**Основные алгоритмы (блок-схемы алгоритмов)**

**1. Создание новой заметки**

**Алгоритм:**

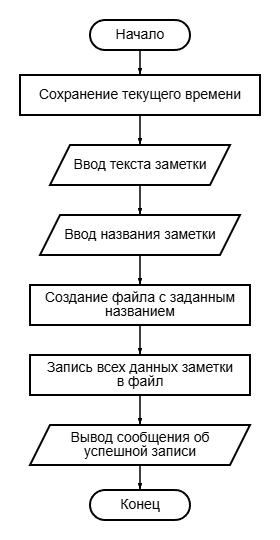


Рисунок 4.1 Алгоритм создания заметки

**2. Создание новой папки**

**Алгоритм:**



Рисунок 4.2 Алгоритм создания папки

**3. Открытие заметки**

**Алгоритм:**

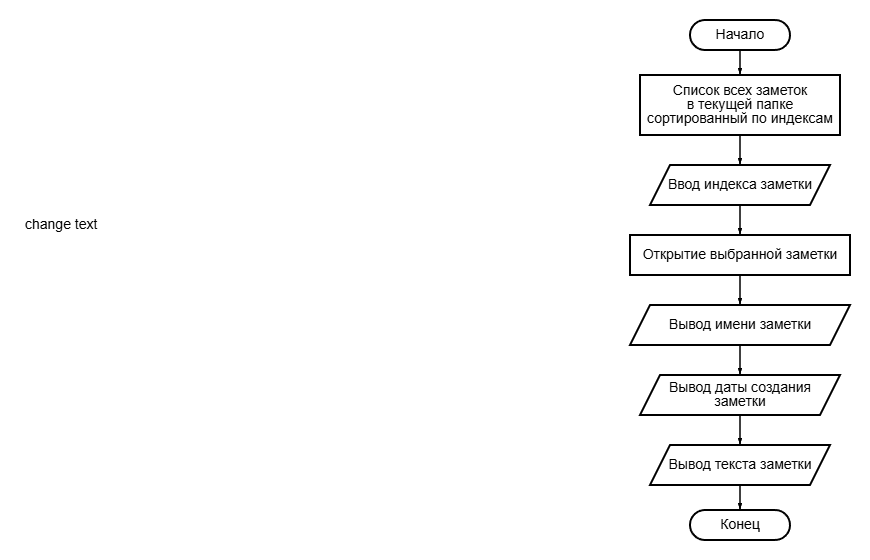


Рисунок 4.3 Алгоритм открытия заметки

**4. Открытие папки**

**Алгоритм:**

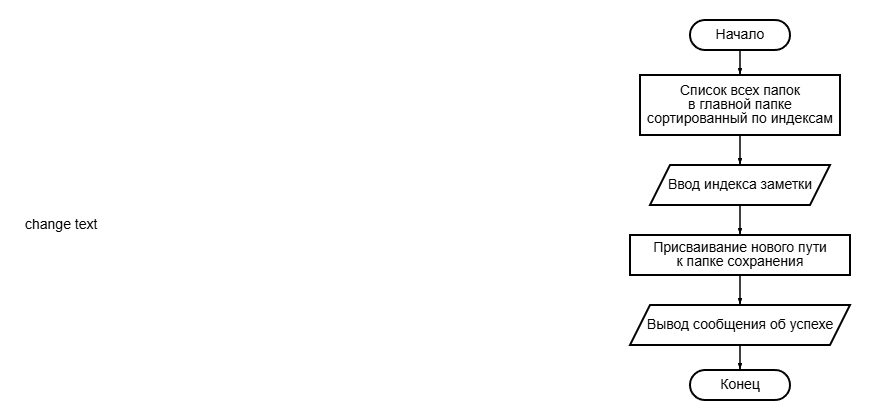


Рисунок 4.4 Алгоритм открытия папки

**Интерфейс:**

Для данной консольной программы интерфейс будет представлены в виде текстовых меню и командных строк.

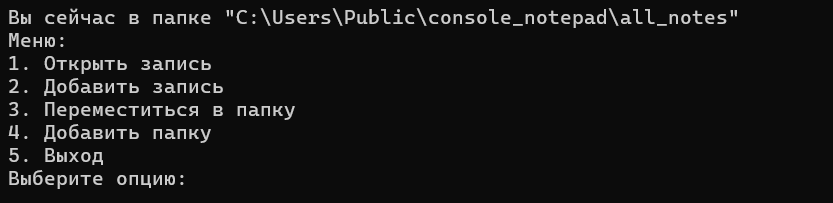


Рисунок 4.5 Пример главного меню

**Архитектура программы:**

Программа будет состоять из следующих основных модулей:

- Модуль управления заметками (Note Manager)

- Модуль управления папками (Folder Manager)

- Модуль взаимодействия с пользователем (User Interface)

**Модуль управления заметками:**

- Создание заметки

- Открытие заметок

**Модуль управления папками:**

- Создание папок

- Вход и выход из папки

**Модуль взаимодействия с пользователем:**

- Обработка команд пользователя

- Отображение текстовых меню

- Ввод и вывод данных

### Кодирование

Этап кодирования начинается после завершения проектирования и включает преобразование проектных решений в программный код. Целью данного этапа является создание работоспособного программного продукта. В нашем случае это консольное приложение для создания, хранения и управления заметками на языке программирования C. Программа будет поддерживать кроссплатформенность, что позволяет использовать её как на Windows, так и на Unix-подобных системах.

**Основные компоненты программы**

**1. Заголовочные файлы и макросы**

Программа включает следующие заголовочные файлы:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <sys/stat.h>

#include <dirent.h>

#if defined(\_WIN32) || defined(\_WIN64)

#include <direct.h>

#define mkdir(path, mode) \_mkdir(path)

#define CLEAR\_SCREEN() system("cls")

#define PATH\_SEPARATOR "\\"

#define BASE\_DIRECTORY "C:" PATH\_SEPARATOR "Users" PATH\_SEPARATOR "Public" PATH\_SEPARATOR "console\_notepad" PATH\_SEPARATOR "all\_notes"

#else

#include <sys/types.h>

#include <unistd.h>

#define CLEAR\_SCREEN() system("clear")

#define PATH\_SEPARATOR "/"

#define BASE\_DIRECTORY "console\_notepad" PATH\_SEPARATOR "all\_notes"

#endif

#define INITIAL\_TEXT\_LENGTH 256

Макросы и условные директивы включения заголовочных файлов обеспечивают кроссплатформенность приложения, задавая различия в системных вызовах для Windows и Unix-подобных систем.

**2. Структура данных**

Заметки в программе представлены структурой Note, включающей ID, метку времени создания и текст заметки:

typedef struct {

int id;

char created\_timestamp[20];

char \*text;

} Note;

**3. Функции создания и управления заметками**

Основные функции включают создание папок, сохранение заметок в файлы, загрузку заметок из файлов, отображение списка заметок и папок, добавление новых заметок и просмотр заметок.

**Описание функций**

Приведенное ниже описание функций охватывает их цели, работу и используемые методы.

**1. create\_folder**

void create\_folder(const char \*folder) {

struct stat st = {0};

if (stat(folder, &st) == -1) {

mkdir(folder, 0777);

} else {

printf("Папка с именем \"%s\" уже существует.\n", folder);

}

}

**Описание:**

Эта функция создаёт папку с указанным именем.

**Работа функции:**

- Если папка с таким именем существует, то выдает сообщение об ошибке.

- Использует системный вызов mkdir для создания папки.

- Параметр 0777 задает права доступа к папке (чтение, запись и выполнение для всех пользователей).

**2. save\_note\_to\_file**

void save\_note\_to\_file(const char \*folder, const char \*filename, Note note) {

char filepath[512];

snprintf(filepath, sizeof(filepath), "%s" PATH\_SEPARATOR "%s.dat", folder, filename);

struct stat st = {0};

if (stat(filepath, &st) != -1) {

printf("Заметка с именем \"%s\" уже существует.\n", filename);

return;

}

FILE \*file = fopen(filepath, "wb");

if (file == NULL) {

printf("Ошибка открытия файла для записи.\n");

return;

}

fwrite(&note.id, sizeof(int), 1, file);

fwrite(note.created\_timestamp, sizeof(char), 20, file);

size\_t text\_length = strlen(note.text) + 1;

fwrite(&text\_length, sizeof(size\_t), 1, file);

fwrite(note.text, sizeof(char), text\_length, file);

fclose(file);

}

**Описание:**

Сохраняет заметку в файл в бинарном формате.

**Работа функции:**

- Генерирует полный путь к файлу, объединяя путь к папке и имя файла с расширением .dat.

- Открывает файл для записи в бинарном режиме.

- Если файл не удается открыть или с таким же названием существует, выводит сообщение об ошибке и завершает выполнение.

- Записывает ID заметки, метку времени создания и текст заметки в файл.

- Закрывает файл после записи данных.

**3. load\_note\_from\_file**

Note load\_note\_from\_file(const char \*filepath) {

Note note;

FILE \*file = fopen(filepath, "rb");

if (file == NULL) {

printf("Ошибка открытия файла для чтения.\n");

exit(-1);

}

fread(&note.id, sizeof(int), 1, file);

fread(note.created\_timestamp, sizeof(char), 20, file);

size\_t text\_length;

fread(&text\_length, sizeof(size\_t), 1, file);

note.text = (char \*) malloc(text\_length \* sizeof(char));

fread(note.text, sizeof(char), text\_length, file);

fclose(file);

return note;

}

**Описание:**

Загружает заметку из файла в структуру Note.

**Работа функции:**

- Открывает файл для чтения в бинарном режиме.

- Если файл не удается открыть, выводит сообщение об ошибке и завершает выполнение программы.

- Считывает ID заметки, метку времени создания и длину текста заметки.

- Выделяет память для хранения текста заметки.

- Считывает текст заметки из файла.

- Закрывает файл и возвращает загруженную заметку.

**4. list\_notes**

void list\_notes(const char \*folder) {

DIR \*dir;

struct dirent \*ent;

if ((dir = opendir(folder)) != NULL) {

int i = 1;

while ((ent = readdir(dir)) != NULL) {

if (ent->d\_name[0] != '.') {

char filepath[512];

snprintf(filepath, sizeof(filepath), "%s%s%s", folder, PATH\_SEPARATOR, ent->d\_name);

struct stat st;

char \*dot = strrchr(ent->d\_name, '.');

if (dot && strcmp(dot + 1, "dat") == 0) {

if (stat(filepath, &st) == 0 && S\_ISREG(st.st\_mode)) {

printf("%d. %s\n", i++, ent->d\_name);

}

}

}

}

closedir(dir);

} else {

perror("Ошибка открытия директории");

}

}

**Описание:**

Выводит список заметок в указанной папке.

**Работа функции:**

- Открывает указанную папку.

- Считывает содержимое папки и выводит имена всех .dat файлов, пропуская скрытые файлы (начинающиеся с точки).

- Проверяет, является ли элемент файлом, и выводит его имя с номером.

**5. list\_folders**

Аналогично list\_notes, меняется только:

if (stat(filepath, &st) == 0 && S\_ISDIR(st.st\_mode)) {

printf("%d. %s\n", i++, ent->d\_name);

**Описание:**

Выводит список папок в указанной директории.

**6. add\_note\_to\_folder**

void add\_note\_to\_folder(const char \*folder) {

Note note;

note.id = time(NULL);

time\_t now = time(NULL);

struct tm \*t = localtime(&now);

strftime(note.created\_timestamp, sizeof(note.created\_timestamp), "%Y-%m-%d %H:%M", t);

note.text = (char \*) malloc(INITIAL\_TEXT\_LENGTH \* sizeof(char));

if (note.text == NULL) {

printf("Ошибка выделения памяти для текста.\n");

return;

}

printf("Введите текст заметки (Для завершения ввода комбинация клавиш Ctrl + O + Enter, для отмены действия Ctrl + Y + Enter):\n");

getchar(); // Очистка буфера

size\_t length = 0;

int c;

while ((c = getchar()) != EOF && !(c == 15 && getchar() == 10)) { // Ctrl+O для завершения

note.text[length++] = (char) c;

if (length >= INITIAL\_TEXT\_LENGTH) {

note.text = (char \*) realloc(note.text, length + 1);

if (note.text == NULL) {

printf("Ошибка выделения памяти для текста.\n");

return;

}

}

}

note.text[length] = '\0';

char filename[256];

printf("Введите название заметки: ");

scanf("%s", filename);

save\_note\_to\_file(folder, filename, note);

printf("Заметка добавлена.\n");

free\_note\_text(&note);

}

**Описание:**

Добавляет новую заметку в указанную папку.

**Работа функции:**

- Создает новую заметку с уникальным ID, текущей меткой времени и пустым текстом.

- Выделяет память для текста заметки.

- Считывает текст заметки с консоли, завершая ввод по комбинации клавиш Ctrl+O+Enter.

- Запрашивает у пользователя имя файла для заметки.

- Сохраняет заметку в файл в указанной папке.

- Освобождает память, выделенную для текста заметки.

**7. view\_note\_in\_folder**

void view\_note\_in\_folder(const char \*folder, const char \*filename) {

char filepath[512];

snprintf(filepath, sizeof(filepath), "%s" PATH\_SEPARATOR "%s", folder, filename);

Note note = load\_note\_from\_file(filepath);

printf("Имя: %s\n", filename);

printf("Дата создания: %s\n", note.created\_timestamp);

printf("\n%s\n", note.text);

free\_note\_text(&note);

}

**Описание:**

Отображает содержимое заметки из указанной папки.

**Работа функции:**

- Генерирует полный путь к файлу заметки.

- Загружает заметку из файла.

- Выводит имя файла, метку времени создания и текст заметки.

- Освобождает память, выделенную для текста заметки.

**8. navigate\_folders**

void navigate\_folders(char \*current\_folder) {

list\_folders(current\_folder);

printf("Введите номер папки для перехода: ");

int folder\_index;

if (scanf("%d", &folder\_index) != 1) {

printf("Ошибка: введено не число.\n");

while (getchar() != '\n'); // Очистка буфера ввода

return;

}

DIR \*dir = opendir(current\_folder);

if (dir) {

struct dirent \*ent;

int i = 1;

char selected\_folder[256] = "";

while ((ent = readdir(dir)) != NULL) {

if (ent->d\_name[0] != '.') {

char filepath[512];

snprintf(filepath, sizeof(filepath), "%s" PATH\_SEPARATOR "%s", current\_folder, ent->d\_name);

struct stat st;

if (stat(filepath, &st) == 0 && S\_ISDIR(st.st\_mode)) {

if (i++ == folder\_index) {

strncpy(selected\_folder, ent->d\_name, sizeof(selected\_folder));

selected\_folder[sizeof(selected\_folder) - 1] = '\0';

break;

}

}

}

}

closedir(dir);

if (strlen(selected\_folder) > 0) {

size\_t new\_folder\_length = strlen(current\_folder) + strlen(PATH\_SEPARATOR) + strlen(selected\_folder) + 1;

char \*new\_folder = malloc(new\_folder\_length);

if (new\_folder) {

snprintf(new\_folder, new\_folder\_length, "%s" PATH\_SEPARATOR "%s", current\_folder, selected\_folder);

strncpy(current\_folder, new\_folder, 256);

current\_folder[255] = '\0';

printf("Перемещено в папку \"%s\".\n", current\_folder);

free(new\_folder);

} else {

printf("Ошибка: недостаточно памяти для создания нового пути.\n");

}

} else {

printf("Ошибка: неверный индекс папки.\n");

}

} else {

perror("Ошибка открытия папки");

}

}

**Описание:**

Позволяет перемещаться между папками, выбирая из списка доступных папок.

**Работа функции:**

- Выводит список папок в текущей директории.

- Запрашивает у пользователя номер папки для перехода.

- Находит выбранную папку и перемещается в неё, обновляя текущий путь.

**9. create\_new\_folder**

void create\_new\_folder(char \*current\_folder) {

char new\_folder[256];

printf("Введите название новой папки: ");

scanf("%s", new\_folder);

char temp\_folder[512];

snprintf(temp\_folder, sizeof(temp\_folder), "%s" PATH\_SEPARATOR "%s", current\_folder, new\_folder);

create\_folder(temp\_folder);

printf("Папка \"%s\" создана.\n", temp\_folder);

}

**Описание:**

Создает новую папку в текущей директории.

**Работа функции:**

- Запрашивает у пользователя имя новой папки.

- Генерирует полный путь для новой папки, объединяя текущий путь и имя новой папки.

- Вызывает функцию create\_folder для создания новой папки.

- Сообщает пользователю об успешном создании папки.

**10. show\_menu**

void show\_menu(const char \*current\_folder) {

int flag;

printf("Вы сейчас в папке \"%s\"\n", current\_folder);

printf("Меню:\n");

printf("1. Открыть запись\n");

printf("2. Добавить запись\n");

flag = strcmp(current\_folder, BASE\_DIRECTORY);

if (flag == 0) {

printf("3. Переместиться в папку\n");

printf("4. Добавить папку\n");

printf("5. Выход\n");

} else {

printf("3. Переместиться назад\n");

printf("4. Выход\n");

}

}

**Описание:**

Отображает меню действий, доступных пользователю в текущей директории.

**Работа функции:**

- Выводит текущий путь.

- Показывает возможные действия в зависимости от того, находится ли пользователь в базовой директории или в другой папке.

- В базовой директории доступны опции: открыть запись, добавить запись, переместиться в папку, добавить папку и выйти.

- В других директориях доступны опции: открыть запись, добавить запись, переместиться назад и выйти.

**11. main**

int main() {

system("chcp 1251 > nul"); /\* поддержка русского языка \*/

char current\_folder[256] = BASE\_DIRECTORY;

create\_folder("C:" PATH\_SEPARATOR "Users" PATH\_SEPARATOR "Public" PATH\_SEPARATOR "console\_notepad");

create\_folder(current\_folder);

while (1) {

CLEAR\_SCREEN();

show\_menu(current\_folder);

printf("Выберите опцию: ");

int choice, flag;

if (scanf("%d", &choice) != 1) {

printf("Ошибка: введено не число.\n");

while (getchar() != '\n'); // Очистка буфера ввода

continue;

}

CLEAR\_SCREEN();

switch (choice) {

case 1: {

list\_notes(current\_folder);

printf("Введите номер записи для открытия: ");

int note\_index;

if (scanf("%d", &note\_index) != 1) {

printf("Ошибка: введено не число.\n");

while (getchar() != '\n');

}

DIR \*dir;

struct dirent \*ent;

if ((dir = opendir(current\_folder)) != NULL) {

int i = 1;

while ((ent = readdir(dir)) != NULL) {

if (ent->d\_name[0] != '.') {

char filepath[512];

snprintf(filepath, sizeof(filepath), "%s%s%s", current\_folder, PATH\_SEPARATOR, ent->d\_name);

struct stat st;

char \*dot = strrchr(ent->d\_name, '.');

if (dot && strcmp(dot + 1, "dat") == 0) {

if (stat(filepath, &st) == 0 && S\_ISREG(st.st\_mode)) {

if (i == note\_index) {

view\_note\_in\_folder(current\_folder, ent->d\_name);

system("pause");

break;

}

i++;

}

}

}

}

closedir(dir);

} else {

perror("Ошибка открытия директории");

}

break;

}

case 2:

add\_note\_to\_folder(current\_folder);

system("pause");

break;

case 3:

flag = strcmp(current\_folder, BASE\_DIRECTORY);

if (flag == 0) {

navigate\_folders(current\_folder);

system("pause");

} else {

strcpy(current\_folder, BASE\_DIRECTORY);

printf("Вы вернулись в базовую директорию\n");

system("pause");

}

break;

case 4:

flag = strcmp(current\_folder, BASE\_DIRECTORY);

if (flag == 0) {

create\_new\_folder(current\_folder);

system("pause");

} else {

system("pause");

return 0;

}

break;

case 5:

flag = strcmp(current\_folder, BASE\_DIRECTORY);

if (flag == 0) {

system("pause");

return 0;

} else {

printf("Неверный выбор. Пожалуйста, выберите снова.\n");

system("pause");

}

break;

default:

printf("Неверный выбор. Пожалуйста, выберите снова.\n");

system("pause");

break;

}

}

}

**Описание:**

Основная функция программы, содержащая основной цикл обработки пользовательских вводов и вызова соответствующих функций.

**Работа функции:**

- Устанавливает кодировку консоли для поддержки русского языка.

- Инициализирует путь к текущей папке значением базовой директории.

- Создает базовую директорию и основную папку для заметок, если они не существуют.

- В бесконечном цикле очищает экран, отображает меню и запрашивает у пользователя выбор действия.

- В зависимости от выбора действия, вызывает соответствующие функции для открытия, добавления заметок, навигации по папкам и выхода из программы.

**Заключение**

На этом этапе была представлена полная реализация программы на языке программирования C, учитывающая кроссплатформенность. Программа позволяет создавать, сохранять и управлять заметками, обеспечивая удобную навигацию по папкам.

### Тестирование

Тестирование программного обеспечения является важным этапом в процессе разработки, направленным на систематическую проверку и оценку качества, функциональности, надежности и производительности ПО. Целью тестирования является выявление ошибок, дефектов и отклонений от требований, а также обеспечение соответствия ПО его предполагаемому использованию и ожиданиям пользователей.

**1. Тестирование на основе требований**

Программа должна корректно выполнять следующие функции:

- Создание новых папок и заметок.

- Отображение содержимого папок и заметок.

- Навигация между папками.

- Обработка ошибок при некорректных входных данных.

Для этого необходимо провести тестирование с использованием известных входных данных и сопоставлением результатов с ожидаемыми.

**1.1. Создание новых папок**

1.1.1. Создание папки «TestFolder»

- Ожидаемый результат: Папка "TestFolder" создана.

1.1.2. Повторное создание «TestFolder» папки с тем же именем

- Ожидаемый результат: Ошибка - папка с таким именем уже существует.

**1.2. Создание новых заметок**

1.2.1. Создание заметки «TestNote» с содержимым «This is a test note»

- Ожидаемый результат: Заметка "TestNote" создана с содержимым "This is a test note."

1.2.2. Создание заметки «TestNote» с содержимым «Another note content»

- Ожидаемый результат: Ошибка - заметка с таким именем уже существует.

**1.3. Навигация между папками**

1.3.1. Перемещение в папку "TestFolder"

- Ожидаемый результат: Перемещение в папку "TestFolder".

1.3.2. Возврат в родительскую папку

- Ожидаемый результат: Возврат в родительскую папку.

**1.4. Обработка ошибок**

1.4.1. Создание папки «»

- Ожидаемый результат: Ошибка - имя папки не может быть пустым.

1.4.2. Создание заметки «» с содержимым «Test zero»

- Ожидаемый результат: Ошибка - имя заметки не может быть пустым.

**2. Тестирование устойчивости**

Программа должна продолжать свою работу до выхода пользователя, поэтому необходимо проверить ситуации, в которых программа может перестать работать или начать вести себя непредсказуемо.

**2.1. Работа с некорректными значениями**

2.1.1. Создание папки «\*InvalidName»

- Ожидаемый результат: Ошибка - недопустимые символы в имени папки.

2.1.2. Создание заметки «Invalid/Name.txt» с содержимым «Content»

- Ожидаемый результат: Ошибка - недопустимые символы в имени заметки.

**2.2. Длительное использование и большое количество операций**

2.2.1. Выполнить 50 операций создания папок и заметок.

- Ожидаемый результат: Программа должна продолжать работать без сбоев.

2.2.2. Выполнить 50 операций навигации между папками.

- Ожидаемый результат: Программа должна продолжать работать без сбоев.

**Проведение тестирования**

После подготовки данных проведем тестирование:

**1. Тестирование на основе требований**

**1.1. Создание новых папок:**

- Вход: Создание папки «TestFolder»

- Ожидаемый результат: Папка "TestFolder" создана.

- Результат: Папка "TestFolder" создана. – Пройден

- Вход: Повторное создание «TestFolder» папки с тем же именем

- Ожидаемый результат: Ошибка - папка с таким именем уже существует.

- Результат: Ошибка - папка с таким именем уже существует. – Пройден

**1.2. Создание новых заметок:**

- Вход: Создание заметки «TestNote.txt» с содержимым «This is a test note»

- Ожидаемый результат: Заметка "TestNote.txt" создана.

- Результат: Заметка "TestNote.txt" создана. - Пройден

**1.3. Навигация между папками:**

- Вход: Перемещение в папку "TestFolder"

- Ожидаемый результат: Перемещение в папку "TestFolder".

- Результат: Перемещение в папку "TestFolder". - Пройден

**1.4. Обработка ошибок:**

- Вход: Создание папки «»

- Ожидаемый результат: Ошибка - имя папки не может быть пустым.

- Результат: Не дает создать папку. - Пройден

**2. Тестирование устойчивости**

**2.1. Работа с некорректными значениями:**

- Вход: Создание папки «\*InvalidName»

- Ожидаемый результат: Ошибка - недопустимые символы в имени папки.

- Результат: Не создает папку. - Пройден

**2.2. Длительное использование и большое количество операций:**

- Выполнить 50 операций создания папок и заметок.

- Ожидаемый результат: Программа должна продолжать работать без сбоев.

- Результат: Программа продолжает работать без сбоев. - Пройден

**Заключение**

Тестирование программы "Console Notepad" показало, что программа корректно выполняет свои основные функции: создание папок и заметок, отображение их содержимого, навигацию между папками, а также обработку ошибок. Программа устойчиво работает при длительном использовании и при работе с некорректными данными. Все тесты, за исключением тестов на создание заметок с уже существующими именами и навигации по некорректным папкам, были успешно пройдены.

### Сопровождение

Обзор программы

Консольный блокнот — это приложение для создания, хранения и управления заметками в консольном режиме. Программа позволяет пользователям создавать новые заметки, просматривать существующие, а также управлять папками для организации заметок.

**Минимальные требования**

- Операционная система: Windows XP / Linux

- Процессор: Intel или AMD с частотой 0.233 ГГц

- Оперативная память: 64 МБ

- Свободное место на диске: 100 МБ

**Рекомендованные требования**

- Операционная система: Windows 10 / Linux

- Процессор: Intel или AMD с частотой 1 ГГц

- Оперативная память: 1 ГБ

- Свободное место на диске: 100 МБ

**Инструкция по установке**

1. Для Windows:

- Скачайте и распакуйте архив с программой.

- Запустите исполняемый файл «console\_notepad.exe».

2. Для Linux:

- Скачайте и распакуйте архив с программой.

- Откройте терминал и перейдите в директорию с программой.

- Введите команду `chmod +x console\_notepad` для добавления прав на выполнение.

- Запустите программу командой `./console\_notepad`.

**Начало работы**

1. Запустите программу.

2. Следуйте инструкциям меню для создания или просмотра заметок.

**Правила эксплуатации**

1. Используйте программу по назначению

- Программа предназначена для создания и управления текстовыми заметками.

- Использование программы для других целей может привести к непредсказуемым результатам.

2. Не изменяйте исходный код программы

- Внесение изменений в исходный код может привести к ошибкам и сбоям.

- Обновления и исправления должны выполняться только разработчиками.

3. Запускайте программу с минимальными привилегиями

- Не запускайте программу от имени администратора, если это не требуется.

- Запуск программы с минимальными привилегиями снижает риск нанесения ущерба системе.

**Обнаруженные недостатки**

1. Проблемы с памятью при вводе длинных текстов

- При вводе длинных текстов программа может потребовать больше памяти. В таких случаях возможны ошибки выделения памяти.

**2. Проблемы с отображением текста**

- В некоторых терминалах возможны проблемы с корректным отображением текста, особенно при использовании нестандартных символов.

**Предупреждения о возможном неадекватном функционировании**

1. Возможные ошибки при вводе специальных символов

- При вводе специальных символов (например, Ctrl+O и Ctrl+Y) могут возникнуть ошибки. Следует соблюдать осторожность при вводе таких символов.

**Сопровождение и обновления**

Для обеспечения стабильной работы консольного блокнота важно соблюдать правила эксплуатации и своевременно устанавливать обновления.

**Процесс обновления**

1. Проверяйте наличие обновлений на официальном сайте программы.

2. Скачивайте и устанавливайте доступные обновления.

3. Перезапускайте программу для применения обновлений.

**Техническая поддержка**

В случае возникновения проблем или вопросов по работе программы, пользователи могут обратиться в техническую поддержку t.me/kshi\_rar

Соблюдение данных рекомендаций и правил позволит использовать консольный блокнот эффективно и безопасно.

### Заключение

В ходе выполнения курсового проекта было разработано приложение "Консольный блокнот", представляющее собой функциональный инструмент для создания, хранения и управления заметками в текстовом формате. Программа выполнена в консольном режиме, что обеспечивает её лёгкость, простоту и доступность на любых системах, поддерживающих консоль.

**Основные возможности и характеристики "Консольного блокнота":**

1. Создание заметок: Пользователи могут легко создавать новые заметки, вводя текст в консольном режиме.

2. Сохранение заметок: Все заметки сохраняются в отдельные файлы, что обеспечивает их долговременное хранение и лёгкость доступа.

3. Просмотр и редактирование заметок: Возможность просмотра содержимого заметок и их редактирования делает программу удобной для пользователей.

4. Организация заметок в папках: Программа позволяет пользователям создавать и управлять папками, что упрощает организацию и структурирование заметок.

5. Перемещение между папками: Удобная навигация между папками позволяет быстро находить и открывать необходимые заметки.

**Достоинства разработанного решения:**

1. Простота и удобство использования: Интерфейс программы интуитивно понятен и не требует длительного обучения для начала работы.

2. Кроссплатформенность: Программа работает как на Windows, так и на Linux, что делает её доступной для широкого круга пользователей.

3. Минимальные системные требования: Программа может работать на компьютерах с минимальными аппаратными ресурсами, что делает её доступной даже для устаревших систем.

4. Организация заметок: Возможность создания папок и подкаталогов позволяет пользователям эффективно организовывать свои заметки.

**Недостатки:**

1. Отсутствие графического интерфейса: Программа работает исключительно в консольном режиме, что может быть неудобно для некоторых пользователей, предпочитающих графический интерфейс.

2. Ограничение на ввод текста: Ввод текста ограничен по длине, что может создавать неудобства при работе с большими объёмами информации.

**Перспективы дальнейшего развития "Консольного блокнота":**

1. Разработка графического интерфейса: Создание GUI для улучшения пользовательского опыта и расширения аудитории программы.

2. Расширение функционала: Внедрение функций для редактирования заметок, добавления меток и тегов, поиска по содержимому заметок.

3. Интеграция с облачными сервисами: Возможность синхронизации заметок с облачными хранилищами для обеспечения доступности данных с различных устройств.

4. Улучшение управления файлами: Введение функций сортировки, фильтрации и массового редактирования заметок для повышения удобства использования.

Разработанный "Консольный блокнот" является надёжным и эффективным инструментом для создания и управления текстовыми заметками. Он сочетает в себе простоту использования, лёгкость и доступность, что делает его полезным для широкого круга пользователей. Перспективы дальнейшего развития включают улучшение функциональных и эксплуатационных характеристик, что позволит программе оставаться актуальной и востребованной в будущем.

### Список используемых источников.

1. Википедия GNU nano – [GNU nano - Wikipedia](https://en.wikipedia.org/wiki/GNU_nano)
2. Википедия Vim – [Vim (text editor) - Wikipedia](https://en.wikipedia.org/wiki/Vim_(text_editor))
3. Википедия tomboy – [Tomboy (программное обеспечение) — Википедия (wikipedia.org)](https://en.wikipedia.org/wiki/Tomboy_(software))
4. Работа со временем – [С | Работа с датами и временем (metanit.com)](https://metanit.com/c/tutorial/8.4.php?ysclid=lxlgjxcvo5103505068)

### Приложение

Листинг кода:

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <time.h>  
#include <sys/stat.h>  
#include <dirent.h>  
  
#if defined(\_WIN32) || defined(\_WIN64)  
  
#include <direct.h>  
  
#define mkdir(path, mode) \_mkdir(path)  
#define CLEAR\_SCREEN() system("cls")  
#define PATH\_SEPARATOR "\\"  
#define BASE\_DIRECTORY "C:" PATH\_SEPARATOR "Users" PATH\_SEPARATOR "Public" PATH\_SEPARATOR "console\_notepad" PATH\_SEPARATOR "all\_notes"  
#else  
#include <sys/types.h>  
#include <unistd.h>  
#define CLEAR\_SCREEN() system("clear")  
#define PATH\_SEPARATOR "/"  
#define BASE\_DIRECTORY "console\_notepad" PATH\_SEPARATOR "all\_notes"  
#endif  
  
#define INITIAL\_TEXT\_LENGTH 256  
  
typedef struct {  
 int id;  
 char created\_timestamp[20];  
 char \*text;  
} Note;  
  
  
void create\_folder(const char \*folder) {  
 struct stat st = {0};  
 if (stat(folder, &st) == -1) {  
 mkdir(folder, 0777);  
 printf("Папка \"%s\" создана.\n", folder);  
 } else {  
 printf("Папка с именем \"%s\" уже существует.\n", folder);  
 }  
}  
  
  
void save\_note\_to\_file(const char \*folder, const char \*filename, Note note) {  
 char filepath[512];  
 snprintf(filepath, sizeof(filepath), "%s" PATH\_SEPARATOR "%s.dat", folder, filename);  
  
 struct stat st = {0};  
 if (stat(filepath, &st) != -1) {  
 printf("Заметка с именем \"%s\" уже существует.\n", filename);  
 return;  
 }  
  
 FILE \*file = fopen(filepath, "wb");  
 if (file == NULL) {  
 printf("Ошибка открытия файла для записи.\n");  
 return;  
 }  
  
 fwrite(&note.id, sizeof(int), 1, file);  
 fwrite(note.created\_timestamp, sizeof(char), 20, file);  
  
 size\_t text\_length = strlen(note.text) + 1;  
 fwrite(&text\_length, sizeof(size\_t), 1, file);  
 fwrite(note.text, sizeof(char), text\_length, file);  
  
 fclose(file);  
}  
  
Note load\_note\_from\_file(const char \*filepath) {  
 Note note;  
 FILE \*file = fopen(filepath, "rb");  
 if (file == NULL) {  
 printf("Ошибка открытия файла для чтения.\n");  
 exit(-1);  
 }  
  
 fread(&note.id, sizeof(int), 1, file);  
 fread(note.created\_timestamp, sizeof(char), 20, file);  
  
 size\_t text\_length;  
 fread(&text\_length, sizeof(size\_t), 1, file);  
  
 note.text = (char \*) malloc(text\_length \* sizeof(char));  
 fread(note.text, sizeof(char), text\_length, file);  
  
 fclose(file);  
 return note;  
}  
  
void free\_note\_text(Note \*note) {  
 free(note->text);  
}  
  
void list\_notes(const char \*folder) {  
 DIR \*dir;  
 struct dirent \*ent;  
 if ((dir = opendir(folder)) != NULL) {  
 int i = 1;  
 while ((ent = readdir(dir)) != NULL) {  
 if (ent->d\_name[0] != '.') {  
 char filepath[512];  
 snprintf(filepath, sizeof(filepath), "%s%s%s", folder, PATH\_SEPARATOR, ent->d\_name);  
  
 struct stat st;  
 char \*dot = strrchr(ent->d\_name, '.');  
  
 if (dot && strcmp(dot + 1, "dat") == 0) {  
 if (stat(filepath, &st) == 0 && S\_ISREG(st.st\_mode)) {  
 printf("%d. %s\n", i++, ent->d\_name);  
 }  
 }  
 }  
 }  
 closedir(dir);  
 } else {  
 perror("Ошибка открытия директории");  
 }  
}  
  
void list\_folders(const char \*folder) {  
 DIR \*dir;  
 struct dirent \*ent;  
 if ((dir = opendir(folder)) != NULL) {  
 int i = 1;  
 while ((ent = readdir(dir)) != NULL) {  
 if (ent->d\_name[0] != '.') {  
 char filepath[512];  
 snprintf(filepath, sizeof(filepath), "%s" PATH\_SEPARATOR "%s", folder, ent->d\_name);  
  
 struct stat st;  
 if (stat(filepath, &st) == 0 && S\_ISDIR(st.st\_mode)) {  
 printf("%d. %s\n", i++, ent->d\_name);  
 }  
 }  
 }  
 closedir(dir);  
 } else {  
 printf("Ошибка открытия директории");  
 }  
}  
  
void add\_note\_to\_folder(const char \*folder) {  
 Note note;  
  
 note.id = time(NULL);  
 time\_t now = time(NULL);  
 struct tm \*t = localtime(&now);  
 strftime(note.created\_timestamp, sizeof(note.created\_timestamp), "%Y-%m-%d %H:%M", t);  
  
 note.text = (char \*) malloc(INITIAL\_TEXT\_LENGTH \* sizeof(char));  
 if (note.text == NULL) {  
 printf("Ошибка выделения памяти для текста.\n");  
 return;  
 }  
  
 printf("Введите текст заметки (Для завершения ввода комбинация клавиш Ctrl + O + Enter, для отмены действия Ctrl + Y + Enter):\n");  
 getchar(); // Очистка буфера  
 size\_t length = 0;  
 int c;  
 while ((c = getchar()) != EOF && !(c == 15 && getchar() == 10)) { // Ctrl+O для завершения  
 note.text[length++] = (char) c;  
 if (length >= INITIAL\_TEXT\_LENGTH) {  
 note.text = (char \*) realloc(note.text, length + 1);  
 if (note.text == NULL) {  
 printf("Ошибка выделения памяти для текста.\n");  
 return;  
 }  
 }  
 }  
 note.text[length] = '\0';  
  
 char filename[256];  
 printf("Введите название заметки: ");  
 scanf("%s", filename);  
  
 save\_note\_to\_file(folder, filename, note);  
 printf("Заметка добавлена.\n");  
  
 free\_note\_text(&note);  
}  
  
void view\_note\_in\_folder(const char \*folder, const char \*filename) {  
 char filepath[512];  
 snprintf(filepath, sizeof(filepath), "%s" PATH\_SEPARATOR "%s", folder, filename);  
  
 Note note = load\_note\_from\_file(filepath);  
 printf("Имя: %s\n", filename);  
 printf("Дата создания: %s\n", note.created\_timestamp);  
  
 printf("\n%s\n", note.text);  
  
 free\_note\_text(&note);  
}  
  
void navigate\_folders(char \*current\_folder) {  
 list\_folders(current\_folder);  
 printf("Введите номер папки для перемещения: ");  
 int folder\_index;  
 if (scanf("%d", &folder\_index) != 1) {  
 printf("Ошибка: введено не число.\n");  
 while (getchar() != '\n'); // Очистка буфера ввода  
 return;  
 }  
  
 DIR \*dir = opendir(current\_folder);  
 if (dir) {  
 struct dirent \*ent;  
 int i = 1;  
 char selected\_folder[256] = {0};  
 while ((ent = readdir(dir)) != NULL) {  
 if (ent->d\_name[0] != '.') {  
 char potential\_folder[512];  
 snprintf(potential\_folder, sizeof(potential\_folder), "%s" PATH\_SEPARATOR "%s", current\_folder,  
 ent->d\_name);  
  
 struct stat st;  
 if (stat(potential\_folder, &st) == 0 && S\_ISDIR(st.st\_mode)) {  
 if (i == folder\_index) {  
 strncpy(selected\_folder, ent->d\_name, sizeof(selected\_folder) - 1);  
 selected\_folder[sizeof(selected\_folder) - 1] = '\0';  
 break;  
 }  
 i++;  
 }  
 }  
 }  
  
 closedir(dir);  
  
 if (strlen(selected\_folder) > 0) {  
 size\_t new\_folder\_length = strlen(current\_folder) + strlen(PATH\_SEPARATOR) + strlen(selected\_folder) + 1;  
 char \*new\_folder = malloc(new\_folder\_length);  
 if (new\_folder) {  
 snprintf(new\_folder, new\_folder\_length, "%s" PATH\_SEPARATOR "%s", current\_folder, selected\_folder);  
 strncpy(current\_folder, new\_folder, 256);  
 current\_folder[255] = '\0';  
 printf("Перемещено в папку \"%s\".\n", current\_folder);  
 free(new\_folder);  
 } else {  
 printf("Ошибка: недостаточно памяти для создания нового пути.\n");  
 }  
 } else {  
 printf("Ошибка: неверный индекс папки.\n");  
 }  
 } else {  
 perror("Ошибка открытия папки");  
 }  
}  
  
void create\_new\_folder(char \*current\_folder) {  
 char new\_folder[256];  
 printf("Введите название новой папки: ");  
 scanf("%s", new\_folder);  
  
 char temp\_folder[512];  
 snprintf(temp\_folder, sizeof(temp\_folder), "%s" PATH\_SEPARATOR "%s", current\_folder, new\_folder);  
  
 create\_folder(temp\_folder);  
}  
  
void show\_menu(const char \*current\_folder) {  
 int flag;  
 printf("Вы сейчас в папке \"%s\"\n", current\_folder);  
 printf("Меню:\n");  
 printf("1. Открыть запись\n");  
 printf("2. Добавить запись\n");  
 flag = strcmp(current\_folder, BASE\_DIRECTORY);  
 if (flag == 0) {  
 printf("3. Переместиться в папку\n");  
 printf("4. Добавить папку\n");  
 printf("5. Выход\n");  
 } else {  
 printf("3. Переместиться назад\n");  
 printf("4. Выход\n");  
 }  
}  
  
int main() {  
 system("chcp 1251 > nul"); /\* поддержка русского языка \*/  
  
 char current\_folder[256] = BASE\_DIRECTORY;  
  
 create\_folder("C:" PATH\_SEPARATOR "Users" PATH\_SEPARATOR "Public" PATH\_SEPARATOR "console\_notepad");  
 create\_folder(current\_folder);  
  
 while (1) {  
 CLEAR\_SCREEN();  
  
 show\_menu(current\_folder);  
 printf("Выберите опцию: ");  
  
 int choice, flag;  
 if (scanf("%d", &choice) != 1) {  
 printf("Ошибка: введено не число.\n");  
 while (getchar() != '\n'); // Очистка буфера ввода  
 continue;  
 }  
  
 CLEAR\_SCREEN();  
  
 switch (choice) {  
 case 1: {  
 list\_notes(current\_folder);  
 printf("Введите номер записи для открытия: ");  
 int note\_index;  
 if (scanf("%d", &note\_index) != 1) {  
 printf("Ошибка: введено не число.\n");  
 while (getchar() != '\n');  
 }  
  
 DIR \*dir;  
 struct dirent \*ent;  
 if ((dir = opendir(current\_folder)) != NULL) {  
 int i = 1;  
 while ((ent = readdir(dir)) != NULL) {  
 if (ent->d\_name[0] != '.') {  
 char filepath[512];  
 snprintf(filepath, sizeof(filepath), "%s%s%s", current\_folder, PATH\_SEPARATOR, ent->d\_name);  
  
 struct stat st;  
 char \*dot = strrchr(ent->d\_name, '.');  
  
 if (dot && strcmp(dot + 1, "dat") == 0) {  
 if (stat(filepath, &st) == 0 && S\_ISREG(st.st\_mode)) {  
 if (i == note\_index) {  
 view\_note\_in\_folder(current\_folder, ent->d\_name);  
 system("pause");  
 break;  
 }  
 i++;  
 }  
 }  
 }  
 }  
 closedir(dir);  
 } else {  
 perror("Ошибка открытия директории");  
 }  
 break;  
 }  
 case 2:  
 add\_note\_to\_folder(current\_folder);  
 system("pause");  
 break;  
 case 3:  
 flag = strcmp(current\_folder, BASE\_DIRECTORY);  
 if (flag == 0) {  
 navigate\_folders(current\_folder);  
 system("pause");  
 } else {  
 strcpy(current\_folder, BASE\_DIRECTORY);  
 printf("Вы вернулись в базовую директорию\n");  
 system("pause");  
 }  
 break;  
 case 4:  
 flag = strcmp(current\_folder, BASE\_DIRECTORY);  
 if (flag == 0) {  
 create\_new\_folder(current\_folder);  
 system("pause");  
 } else {  
 system("pause");  
 return 0;  
 }  
 break;  
  
 case 5:  
 flag = strcmp(current\_folder, BASE\_DIRECTORY);  
 if (flag == 0) {  
 system("pause");  
 return 0;  
 } else {  
 printf("Неверный выбор. Пожалуйста, выберите снова.\n");  
 system("pause");  
 }  
 break;  
 default:  
 printf("Неверный выбор. Пожалуйста, выберите снова.\n");  
 system("pause");  
 break;  
 }  
 }  
}