A red blue and white star with a white flower

AI-generated content may be incorrect.

**TRIBHUVAN UNIVERSITY**

**INSTITUE OF ENGINEERING**

**PULCHOWK CAMPUS**

A REPORT ON

**SUBMITTED TO:**

Department of Electronics and Computer Engineering

**SUBMITTED BY:**

Nischal Khatri (081BCT046)

Nishant Bhatt (081BCT047)

Nugaw Ratna Shakya (081BCT048)

**Table of Content**

Acknowledgement………………………………………………………………………… 3

Abstract…………………………………………………………………………………….. 4

Introduction………………………………………………………………………………… 5

Problem Analysis………………………………………………………………………….. 6

1. Understanding the problem………………………………………………………. 6
2. Input requirement………………………………………………………………….. 6
3. Output requirement……………………………………………………………….. 6
4. Technical Feasibility………………………………………………………………. 7

Theory………………………………………………………………………………………. 8

1. Loops……………………………………………………………………………….. 8-9
2. File Handling……………………………………………………………………….. 9-10
3. Branching Statements…………………………………………………………….. 10
4. Other concepts……………………………………………………………………..10-11

Algorithm and Flowchart………………………………………………………………….12-28

Source Code and Output………………………………………………………………… 29-

Result………………………………………………………………………………………..

Discussion…………………………………………………………………………………

Conclusion and Future Scope…………………………………………………………….

**ACKNOWLEDGEMENT**

The successful completion of this project report, titled **"A Project Report on Taskbar Management System,"** would not have been possible without the support and guidance of several individuals.

We extend our heartfelt gratitude to **Mr. Santosh Giri** for his invaluable assistance, guidance, and encouragement throughout this project. His dedication, insightful feedback, and keen attention to detail have been instrumental in shaping this report. His unwavering support and constructive suggestions have greatly contributed to its refinement.

We are also sincerely thankful to the **library staff and faculty members** for their assistance in gathering essential resources for our research and documentation. Their dedication and willingness to help played a significant role in the successful completion of this report.

Furthermore, we deeply appreciate the continuous encouragement and support from our **friends and family**. Their belief in our abilities and constant motivation kept us inspired and committed throughout this journey.

We are truly grateful to everyone who contributed, directly or indirectly, to the completion of this project.

**ABSTRACT**

The **Taskbar Management System** is a simple C-based project that allows users to manage their tasks efficiently. It provides functionalities such as adding, deleting, updating, and viewing tasks. The system uses **file handling (fwrite & fread)** to store tasks permanently, ensuring data persistence even after the program is closed.

The program demonstrates fundamental principles of C programming, including file operations, string manipulation, user input handling, and basic security practices. It serves as a starting point for more advanced taskbar management system.

This program provides the functionalities to:

1. Users can create new tasks with details like title, priority, and deadline.
2. Displays all saved tasks in a structured format.
3. Users can remove completed or unwanted tasks.
4. Saves tasks in a file using fwrite() and fread() for later retrieval.

The system's design allows for easy expansion and customization in the future. This program increases user’s productivity by organizing and tracking tasks efficiently. In conclusion, the taskbar management system offers a simple and lightweight program which is easy to implement to increase productivity.

**Introduction**

This project, developed in **C programming,** allows users to perform essential task management operations such as adding, updating, deleting, and viewing tasks. By utilizing **file handling techniques,** the system ensures that task data remains stored even after the program is closed, providing a persistent and reliable task management solution.

This system is designed as a **console-based application** that follows a **user-friendly approach,** making it accessible to users of all levels. The project not only demonstrates fundamental C programming concepts such as **structures, file handling, loops, and functions**, but also provides a practical real-world application of task organization and tracking.

The user interacts with the taskbar management system through a simple and intuitive menu-driven interface. Upon launching the program, they are presented with a set of options to manage their tasks effectively. The entire interaction is based on numerical inputs, making it easy to navigate through the system. Once the user has completed their task management activities, they can exit the program, ensuring that all tasks are securely saved using file handling techniques for future access.

Overall, The **Taskbar Management System** is a structured and practical C programming project that highlights key concepts such as **data management, user interaction, and file handling.** This project is an excellent example of how basic C programming functionalities can be combined to create a meaningful and functional application.

**Problem Analysis**

* ***Understanding the problem***

Manually monitoring tasks can be inefficient and error-prone, resulting in missed deadlines and decreased productivity. The Taskbar Management System is intended to address this issue by offering a systematic method for managing tasks via a console-based program. The system allows users to create, change, view, and delete tasks while also assuring data persistence through file handling. By automating task tracking, the application enhances organization and helps users remain on top of their tasks.

* ***Input requirement***

The system requires users to provide specific inputs to manage their tasks effectively. These include:

* Task title (name or description of the task)
* Task priority (e.g., low, medium, high)
* Task deadline (date or time by which the task needs to be completed)
* Selection of actions (adding, updating, deleting, or viewing tasks)

These inputs are collected through a simple menu-based interface where users can enter their choices using numerical keys.

* ***Output requirement***

The program generates outputs based on user actions to ensure clarity and ease of use. These outputs include:

* Confirmation messages for successful task additions, updates, or deletions
* A well-structured list of all tasks when the user chooses to view them
* Error messages for invalid inputs or unsuccessful operations
* A notification when exiting the program to ensure that tasks are saved properly

The output is displayed in a clear and organized format, making it easy for users to understand and interact with the system.

* ***Technical feasibility***

The project is technically feasible as it is implemented using **C programming,** which provides a strong foundation for file handling and data management. The program makes use of **structures** to store task information, **file handling** to ensure data persistence, and **loops and conditional statements** to manage user interactions. The system requires minimal hardware resources and can run efficiently on any basic computing environment that supports C programming. Since it is a console-based application, it does not rely on any external libraries, making it simple, lightweight, and easy to implement.

**Theory**

* ***Loops Used***

1. ***While loop (Load tasks from file)***

 It reads task data (ID, description, priority, status, deadline, category) from a file line by line.

 It stores each task in an array (tasks) so that it can be accessed and modified later.

 It stops reading when:

* The maximum number of tasks is reached (MAX\_TASKS).
* fscanf() fails to read six tasks correctly.

1. ***For loop (Save tasks to file)***

 Saves task data permanently to a file so that it can be accessed later.

**** Ensures structured storage for easy retrieval.

 Preserves task details including ID, description, priority, status, deadline, and category.

1. ***While loop (load notes from file)***

 Reads saved notes from a file when the program starts, ensuring data persistence.

 Stores structured data (date and content) in an array for easy access and modification.

 Prevents exceeding the maximum number of notes, ensuring safe memory handling.

1. ***For loop (Save notes to file)***

 Saves note data permanently to a file so that it can be accessed later.

**** Ensures structured storage for easy retrieval.

1. ***For loop (Task display)***

 Enhances readability by presenting task details in a structured format.

 Incorporates deadline tracking by displaying time left for each task.

 Improves user experience by making it easy to see task priority, status, and deadlines at a glance.

1. ***Nested For loop (Task deletion)***

 Efficiently removes tasks while maintaining the array structure.

 Ensures there are no gaps in the task list after deletion.

 Prevents errors by updating the task count correctly.

1. ***For loop (Display notes)***

 Filter and display notes based on a specific date or show all notes if no date filter is applied.

 Improves user experience by allowing the user to view only relevant notes, enhancing the searchability.

 Displays notes in a clear, formatted manner that is easy to read.

1. ***For loop (Display tasks)***

 Filter and display tasks based on a specific filter or show all notes if no date filter is applied.

 Improves user experience by allowing the user to view only relevant tasks, enhancing the searchability.

 Displays tasks in a clear, formatted manner that is easy to read.

1. ***While loop (User input)***

 Take input from the user in numeral value.

 Based on this value, it redirects the operation of the program to the respective user defined functions for further operations.

* ***File handling***

***1. File Opening:***

● Files such as”tasks.txt”, “notes.txt” and "passwords.txt" are opened using fopen() for reading or writing purposes.

***2. File Reading/Writing:***

● fscanf() reads formatted data from a file, while fprintf() writes formatted data to a file.

● Used for tasks such as saving passwords, saving the input from the user of tasks and note, and printing the respective password, notes and tasks into the files.

***3. File Closing:***

● Files are closed using fclose() after their usage to release system resources and ensure data integrity.

* ***Branching statements***

***1. If-Else Statements:***

● Conditional execution is achieved using if statements, if-else statements, if-else-if statements and nested if-else statements.

***2. Switch Statements:***

● Switch statements handle different menu choices selected by the user, providing a concise way to manage multiple conditions.

* **Other concepts**

***1. String Manipulation:***

* + Functions like strcpy(), strcat(), and strcmp() from manipulate strings.

***2. ASCII color codes***

* + ASCII color codes are used to change the color of text output in terminals that support ANSI escape codes. These codes allow you to add color to your terminal output, making it more visually appealing or easier to read.

***3. User – Defined Functions***

* + In this program, **User-Defined Functions (UDFs)** are used to modularize code, making it more organized and reusable. For example, functions like calculateCountdown() handle specific tasks (like calculating time remaining) independently of the main logic, improving code clarity and maintainability. These functions are designed to take inputs, process them, and return results without cluttering the main program flow.

***4. #inclde<time.h>***

* + The time.h header file in this program is used to handle time-related tasks, such as calculating the **countdown** to a task's deadline. It provides functions like time(), difftime() to manage and manipulate time, allowing the program to compute the difference between the current time and the task's deadline. This ensures accurate tracking of how much time remains until a task is due.

***5. Structure***

* + A **structure** in C is a user-defined data type that groups related variables of different data types under one name. It helps organize complex data logically.
  + In this program;
* struct task: stores task details like ID, description, priority, status, deadline, and category.
* struct notes: stores notes with a date and content.

***6. Array***

* An **array** in C is a collection of elements of the **same data type**, stored in **contiguous memory locations**. It allows easy access and manipulation of multiple values using an index.
* In this program;
* tasks[MAX\_TASKS]: is an array to manage multiple tasks.
* tasks[MAX\_NOTES]: is an array to manage multiple notes.

# ALGORITHM & FLOWCHART

An algorithm is a systematic, step-by-step approach to solving a problem or completing a task. It consists of a sequence of instructions designed to achieve a specific outcome. Algorithms are widely used in various fields such as computer science, mathematics, and engineering.

A flowchart, in contrast, is a visual representation of an algorithm. It utilizes symbols and diagrams to depict the steps and control flow of the process. Flowcharts are beneficial for understanding and organizing complex operations, making them useful for both programming and explaining concepts to others.

Both algorithms and flowcharts play a crucial role in programming. Algorithms simplify problem-solving by breaking complex tasks into manageable steps, ensuring code is both correct and efficient. Flowcharts complement this by providing a clear, graphical illustration of the algorithm’s logic, improving comprehension and communication.

The Algorithms and the corresponding Flowcharts for this project are:

## Algorithm for clearScreen Function

Start

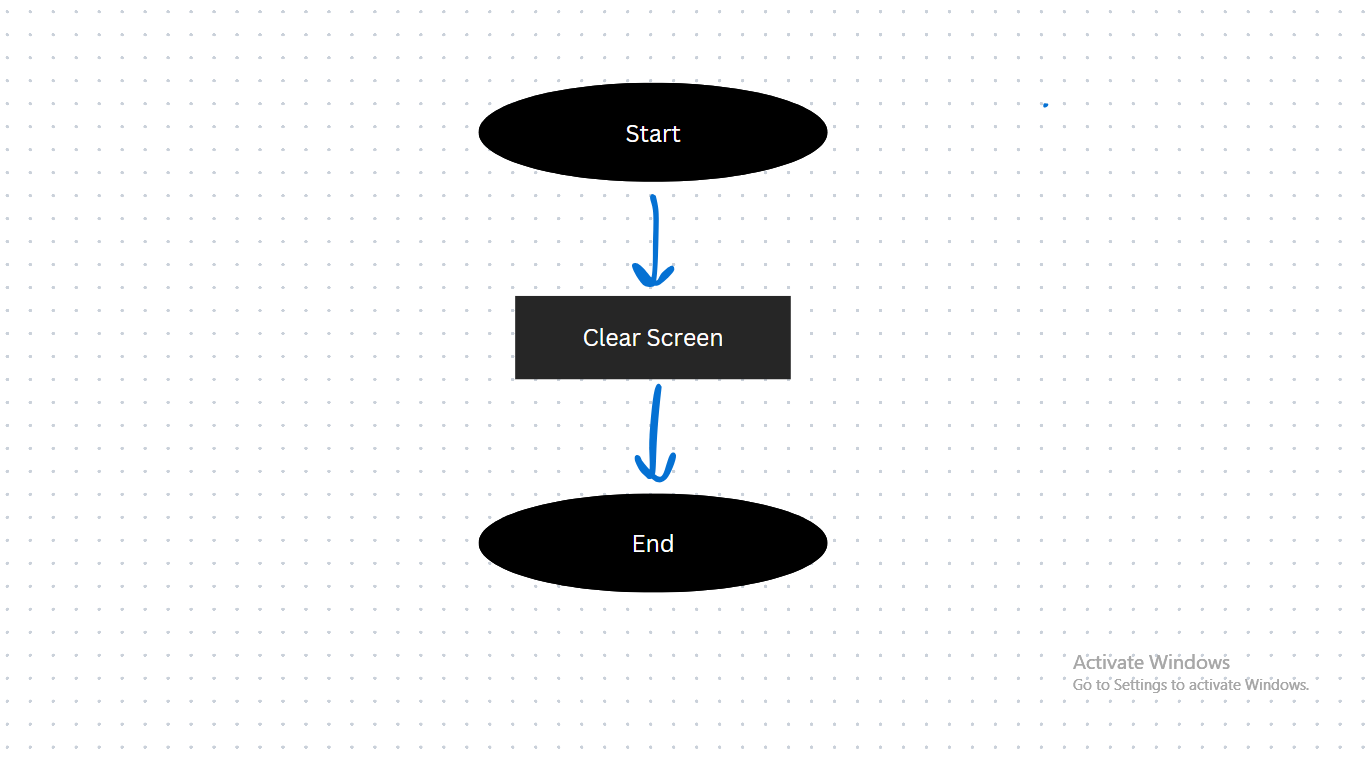
Step 1: Check the operating system.

Step 2: If the OS is Windows, execute the **cls** command to clear the screen.

Step 3: If the OS is not Windows (e.g., Linux/Mac), execute the **clear** command to clear the screen.

End

\



## Algorithm for getTodayDate Function

Start

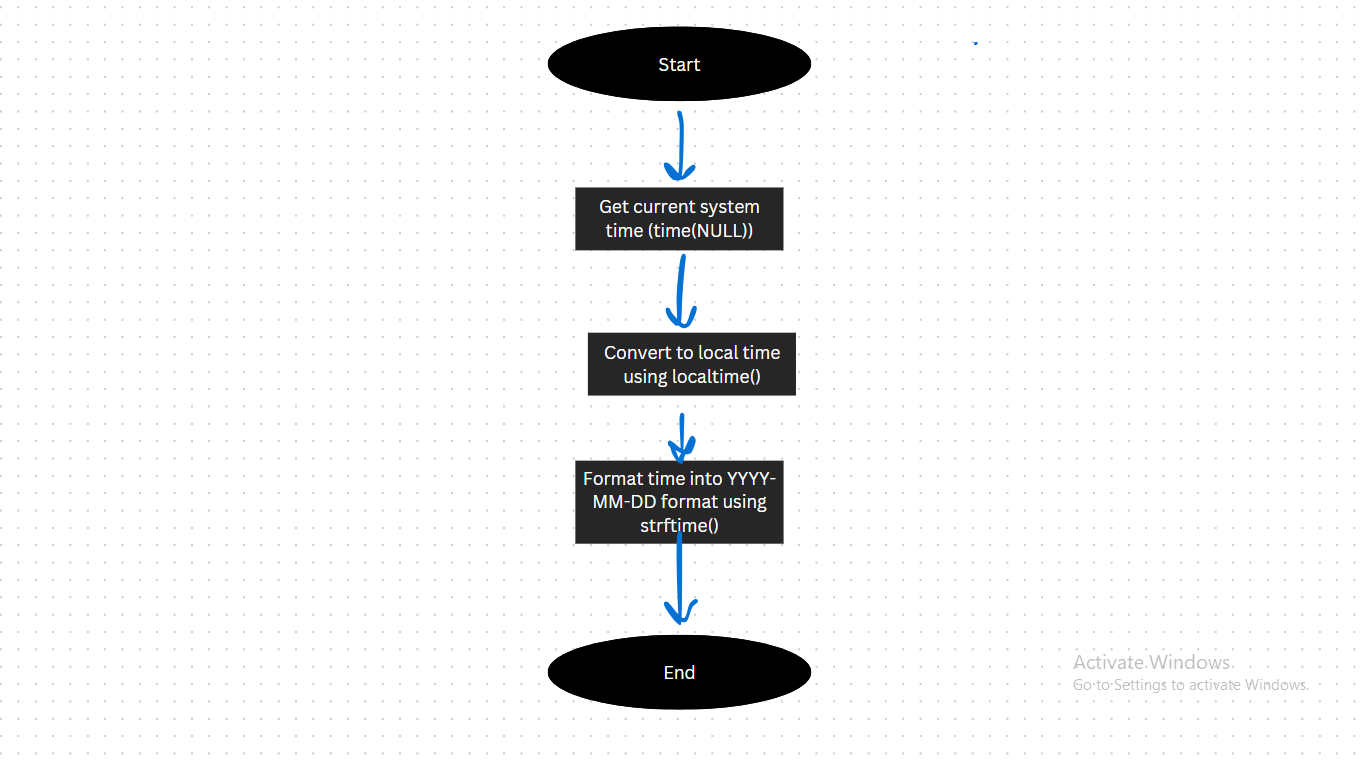
Step 1: Get the current time using **time(NULL).**

Step 2: Convert the time to a **tm** structure using **localtime**.

Step 3: Format the date as **YYYY-MM-DD** using **strftime**.

Step 4: Store the formatted date in the **todayDate** variable.

End



## Algorithm for loadTasks Function

Start

Step 1: Open the **tasks.txt** file in read mode.

Step 2: If the file cannot be opened, return 0.

Step 3: Initialize **taskCount** to 0.

Step 4: Repeat until **taskCount** reaches **MAX\_TASKS** or the end of the file is reached:

Step 4.1: Read task details (ID, description, priority, status, deadline, category) from the file.

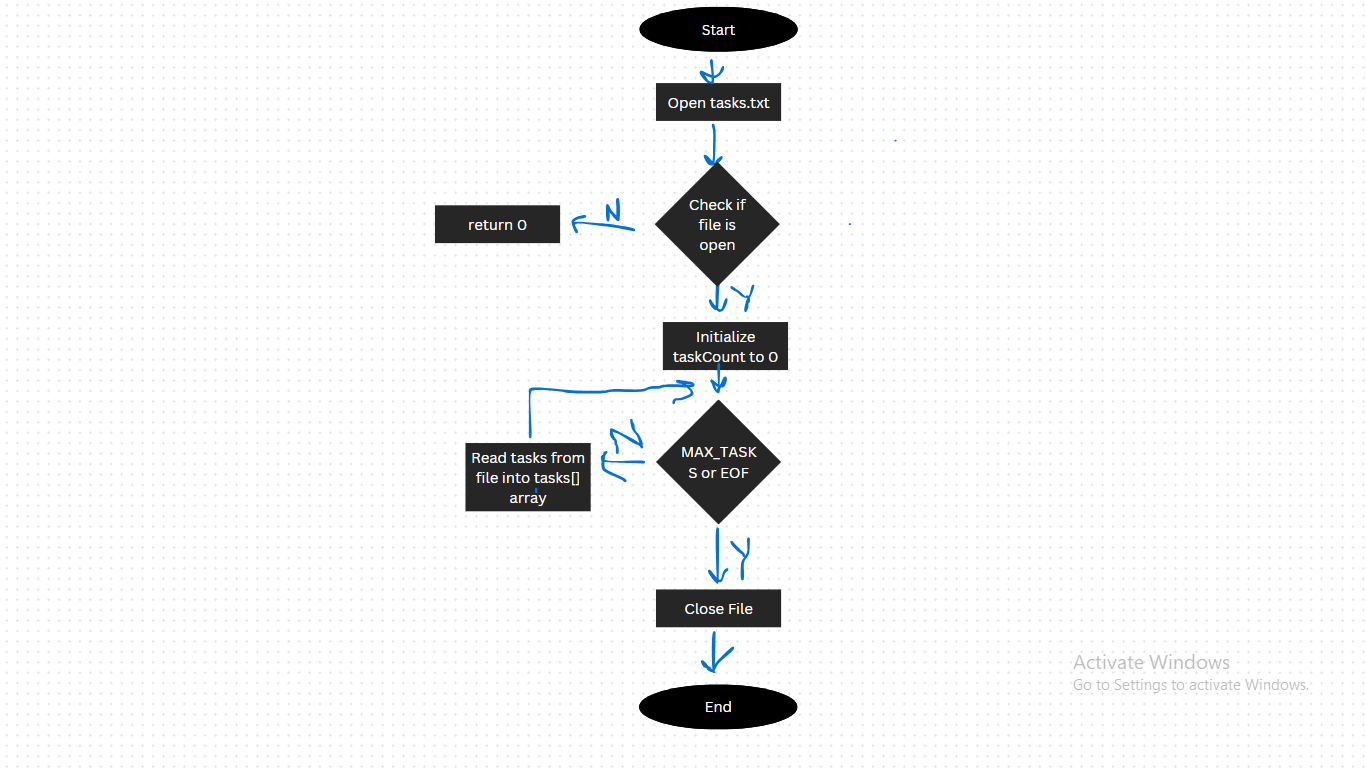
Step 4.2: Store the task in the **tasks** array.

Step 4.3: Increment **taskCount**.

Step 5: Close the file.

Step 6: Return 1 to indicate success.

End



## Algorithm for saveTasks Function

Start

Step 1: Open the **tasks.txt** file in write mode.

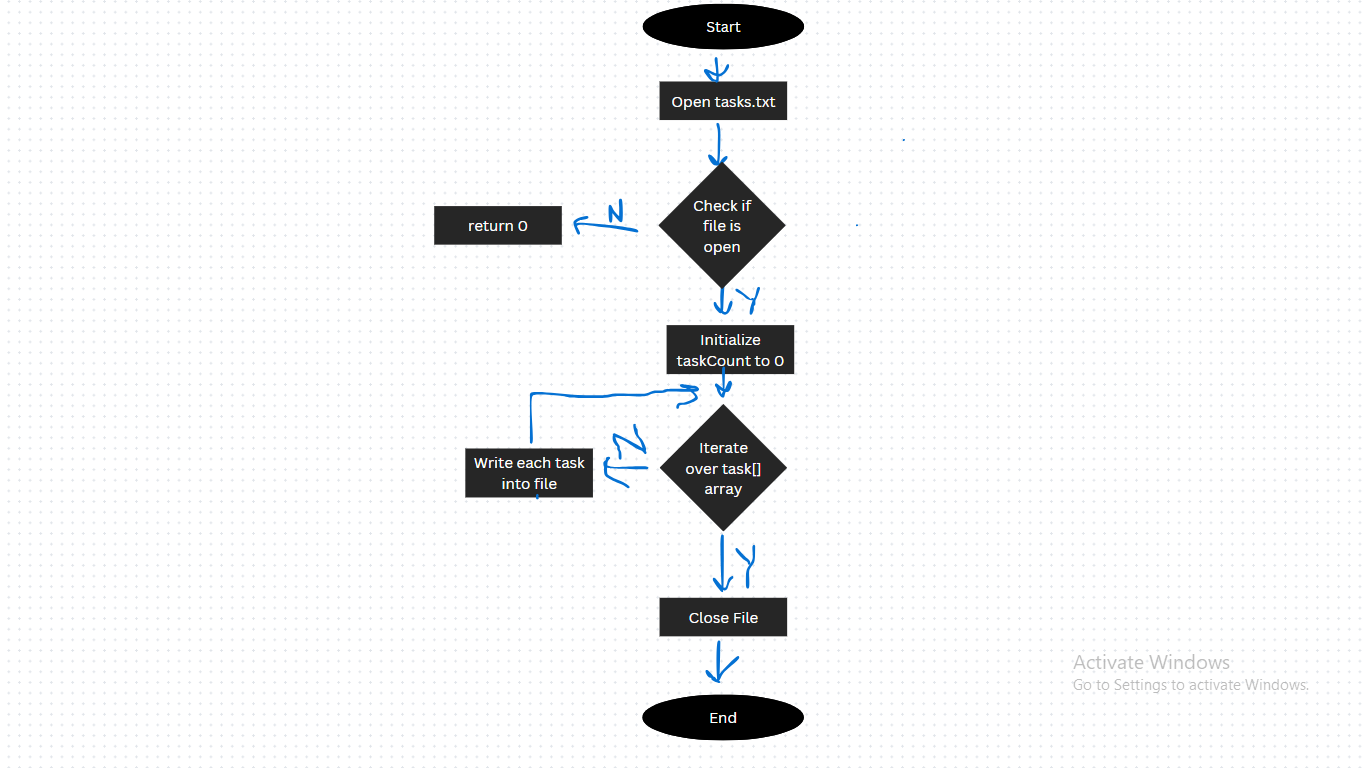
Step 2: If the file cannot be opened, print an error message and return.

Step 3: Repeat for each task in the **tasks** array:

Step 3.1: Write task details (ID, description, priority, status, deadline, category) to the file.

Step 4: Close the file.

End



## Algorithm for loadNotes Function

Start

Step 1: Open the **notes.txt** file in read mode.

Step 2: If the file cannot be opened, return 0.

Step 3: Initialize **noteCount** to 0.

Step 4: Repeat until **noteCount** reaches **MAX\_NOTES** or the end of the file is reached:

Step 4.1: Read note details (date and content) from the file.

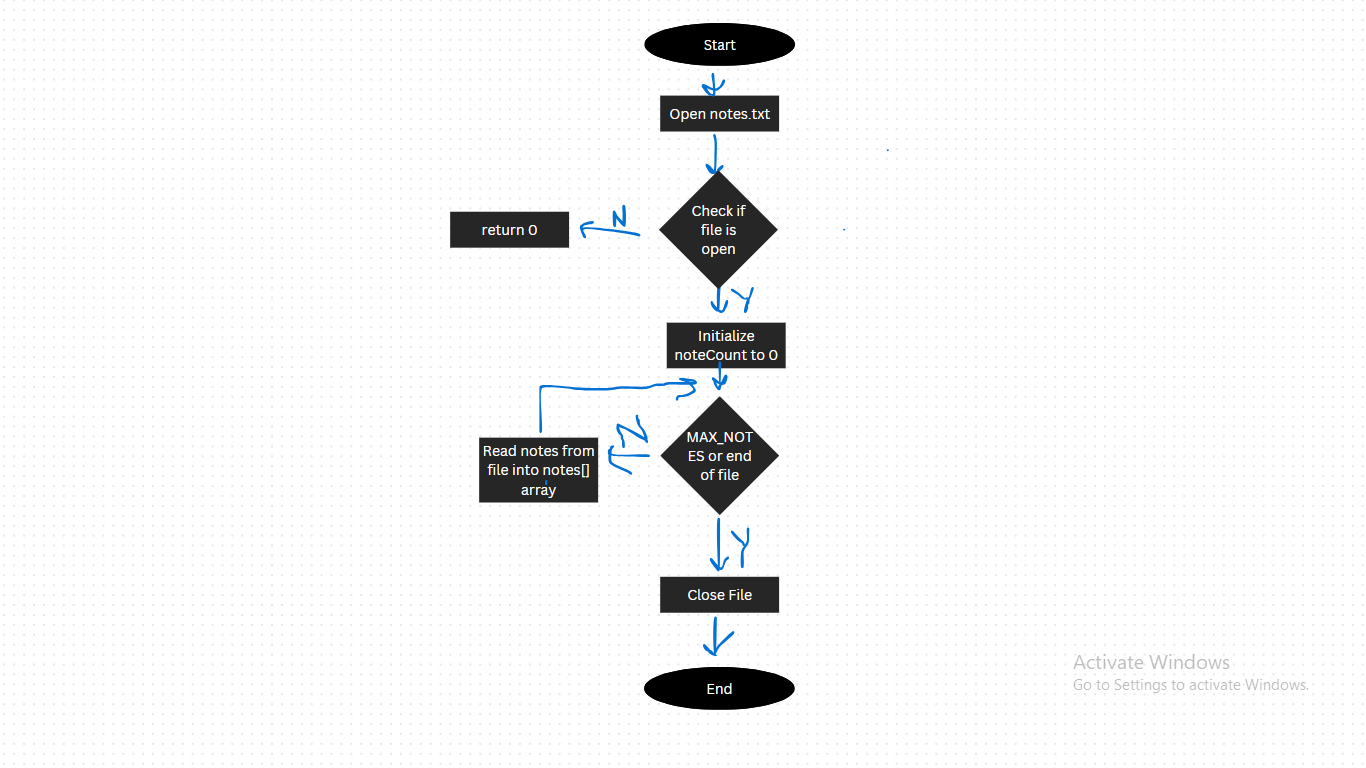
Step 4.2: Store the note in the **notes** array.

Step 4.3: Increment **noteCount**.

Step 5: Close the file.

Step 6: Return 1 to indicate success.

End



## Algorithm for saveNotes Function

Start

Step 1: Open the **notes.txt** file in write mode.

Step 2: If the file cannot be opened, print an error message and return.

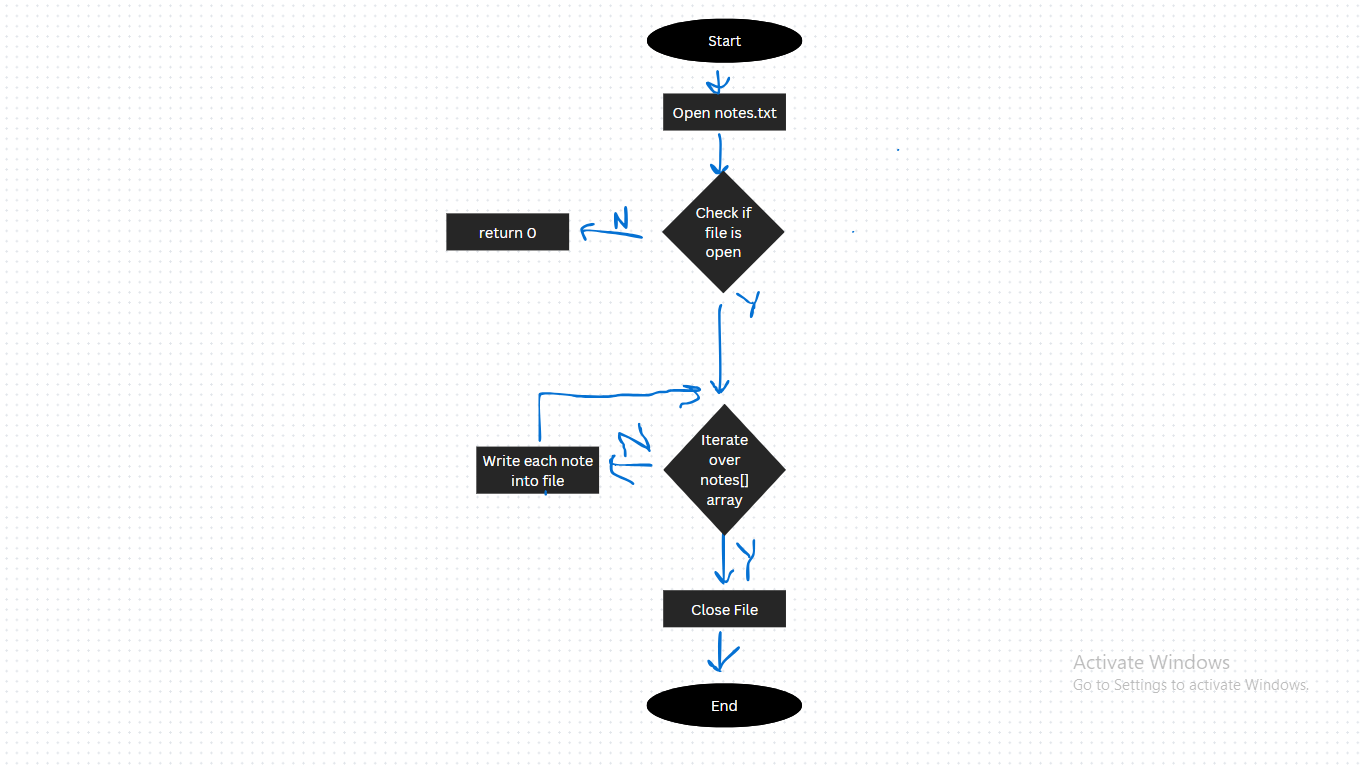
Step 3: Repeat for each note in the **notes** array:

Step 3.1: Write note details (date and content) to the file.

Step 4: Close the file.

Step 5: Print a success message indicating the number of notes saved.

End



## Algorithm for calculateCountdown Function

Start

Step 1: Parse the **deadline** string into year, month, and day.

Step 2: Convert the parsed values into a **tm** structure.

Step 3: Get the current time using **time(NULL).**

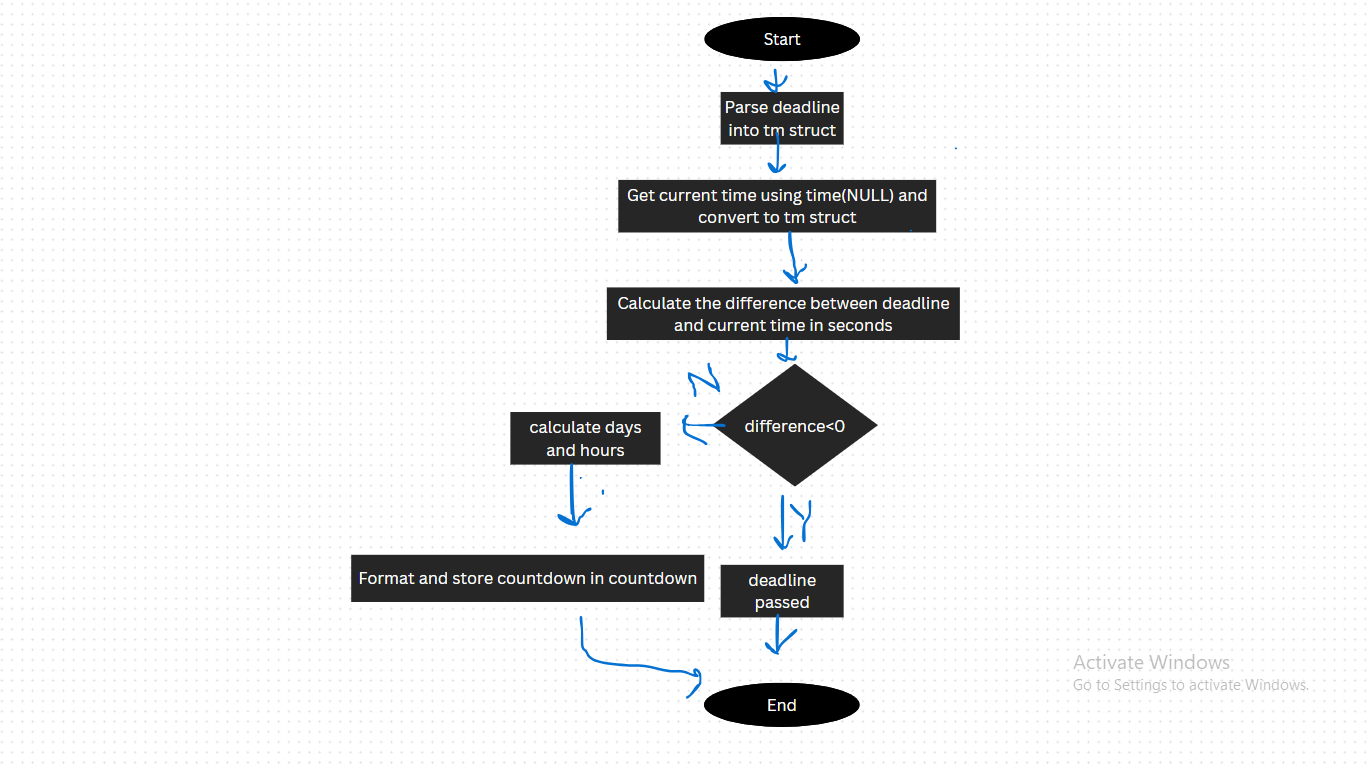
Step 4: Calculate the difference between the deadline and the current time.

Step 5: If the deadline has passed, set the countdown message to "Deadline passed".

Step 6: If the deadline is in the future, calculate the remaining days and hours.

Step 7: Store the countdown message in the **countdown** variable.

End



## Algorithm for addTask Function

Start

Step 1: Check if the task list is full. If full, print an error message and return.

Step 2: Initialize a new task with an auto-incremented ID.

Step 3: Prompt the user to enter the task description.

Step 4: Prompt the user to enter the task priority.

Step 5: Prompt the user to enter the task deadline.

Step 6: Prompt the user to enter the task category.

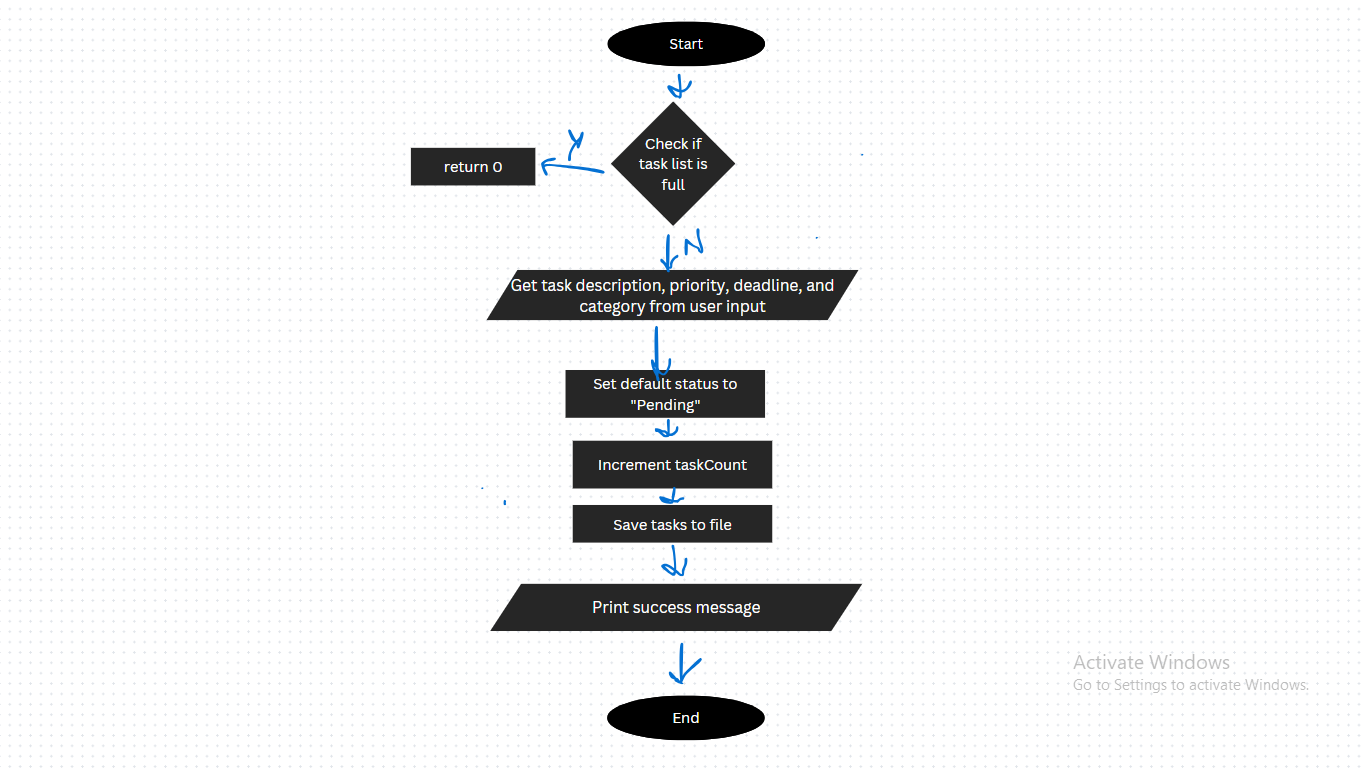
Step 7: Set the task status to "Pending".

Step 8: Add the task to the `tasks` array and increment **taskCount**.

Step 9: Save the updated task list to the file.

Step 10: Print a success message.

End



## Algorithm for displayTasks Function

Start

Step 1: Check if there are no tasks. If none, print a message and return.

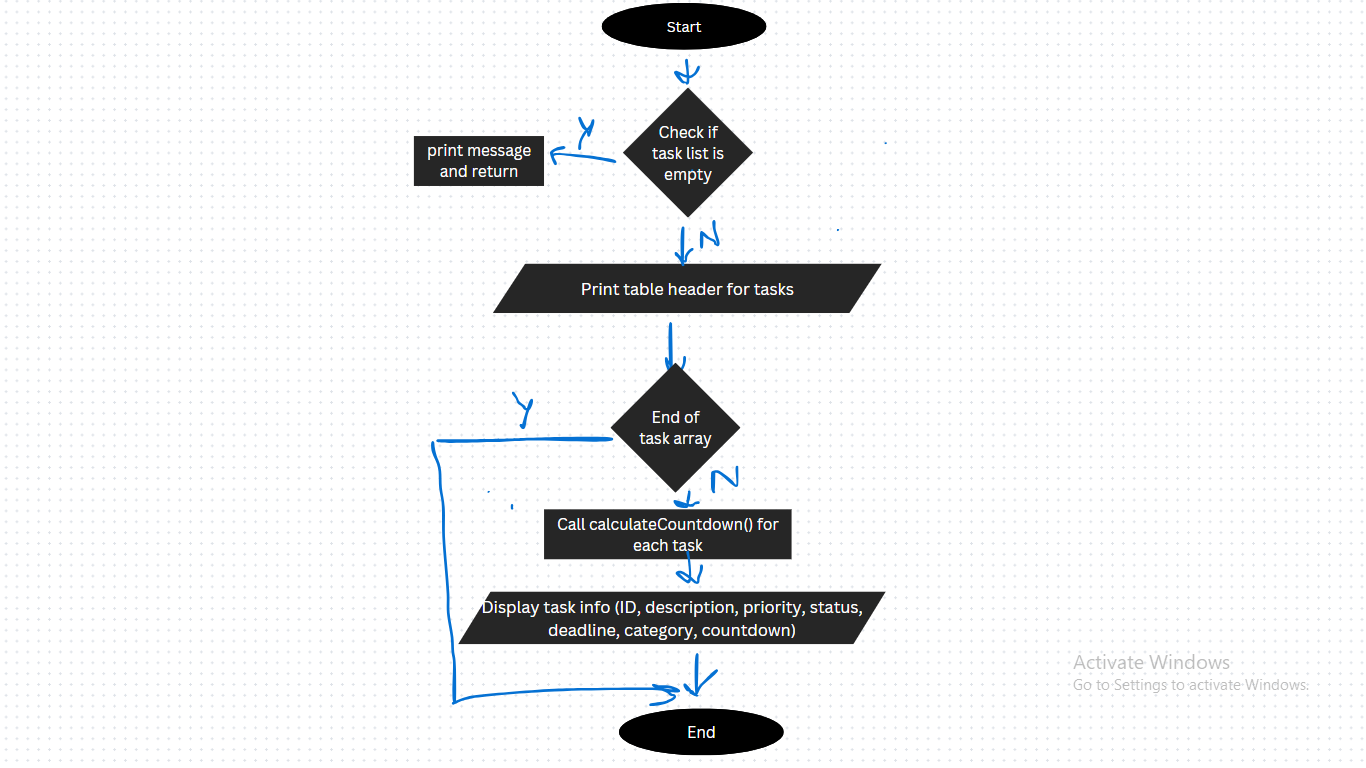
Step 2: Print the table header for tasks.

Step 3: Repeat for each task in the **tasks** array:

Step 3.1: Calculate the countdown for the task deadline.

Step 3.2: Print the task details (ID, description, priority, status, deadline, category, countdown).

End



## Algorithm for deleteTask Function

Start

Step 1: Check if there are no tasks. If none, print a message and return.

Step 2: Display the current list of tasks.

Step 3: Prompt the user to enter the ID of the task to delete.

Step 4: Search for the task with the specified ID.

Step 5: If the task is found:

Step 5.1: Shift all subsequent tasks to fill the gap.

Step 5.2: Decrement **taskCount**.

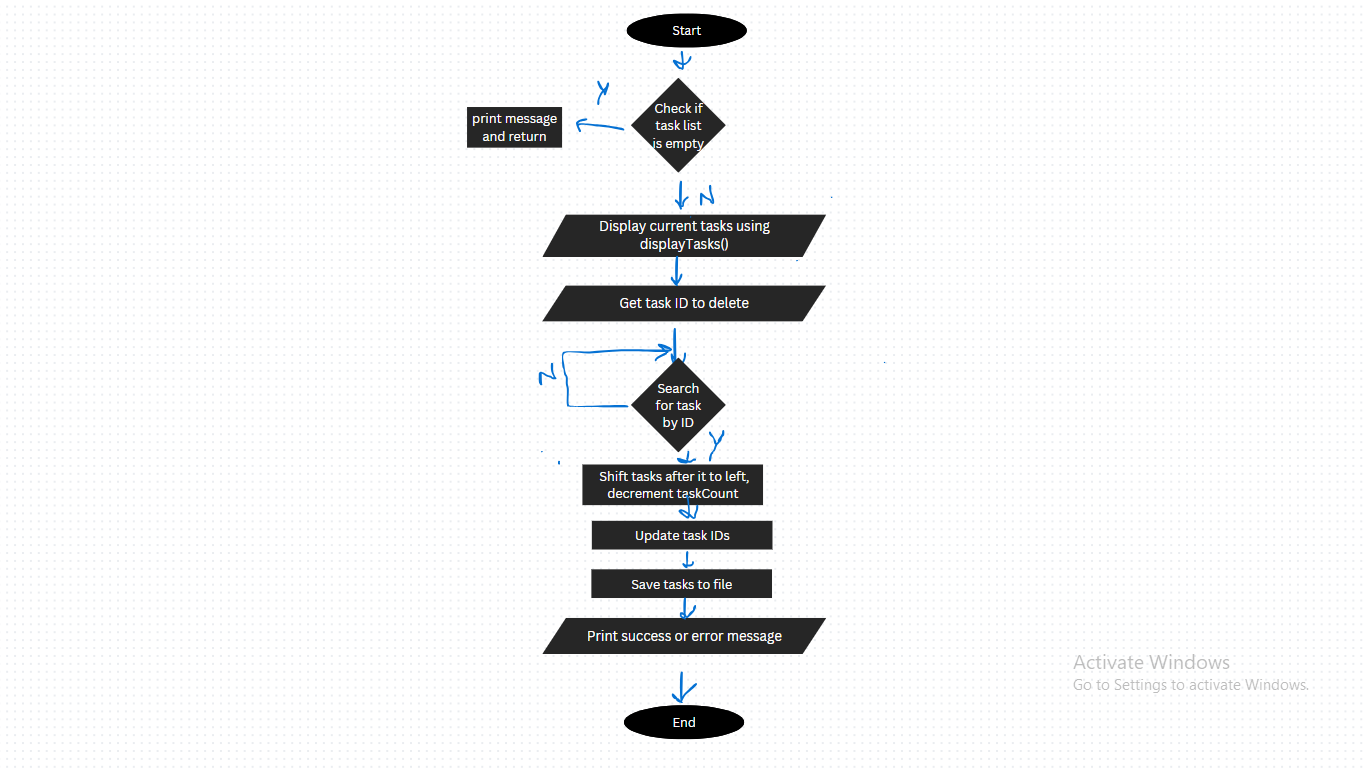
Step 5.3: Update the IDs of the remaining tasks.

Step 5.4: Save the updated task list to the file.

Step 5.5: Print a success message.

Step 6: If the task is not found, print an error message.

End



## Algorithm for markCompleted Function

Start

Step 1: Check if there are no tasks. If none, print a message and return.

Step 2: Display the current list of tasks.

Step 3: Prompt the user to enter the ID of the task to mark as completed.

Step 4: Search for the task with the specified ID.

Step 5: If the task is found:

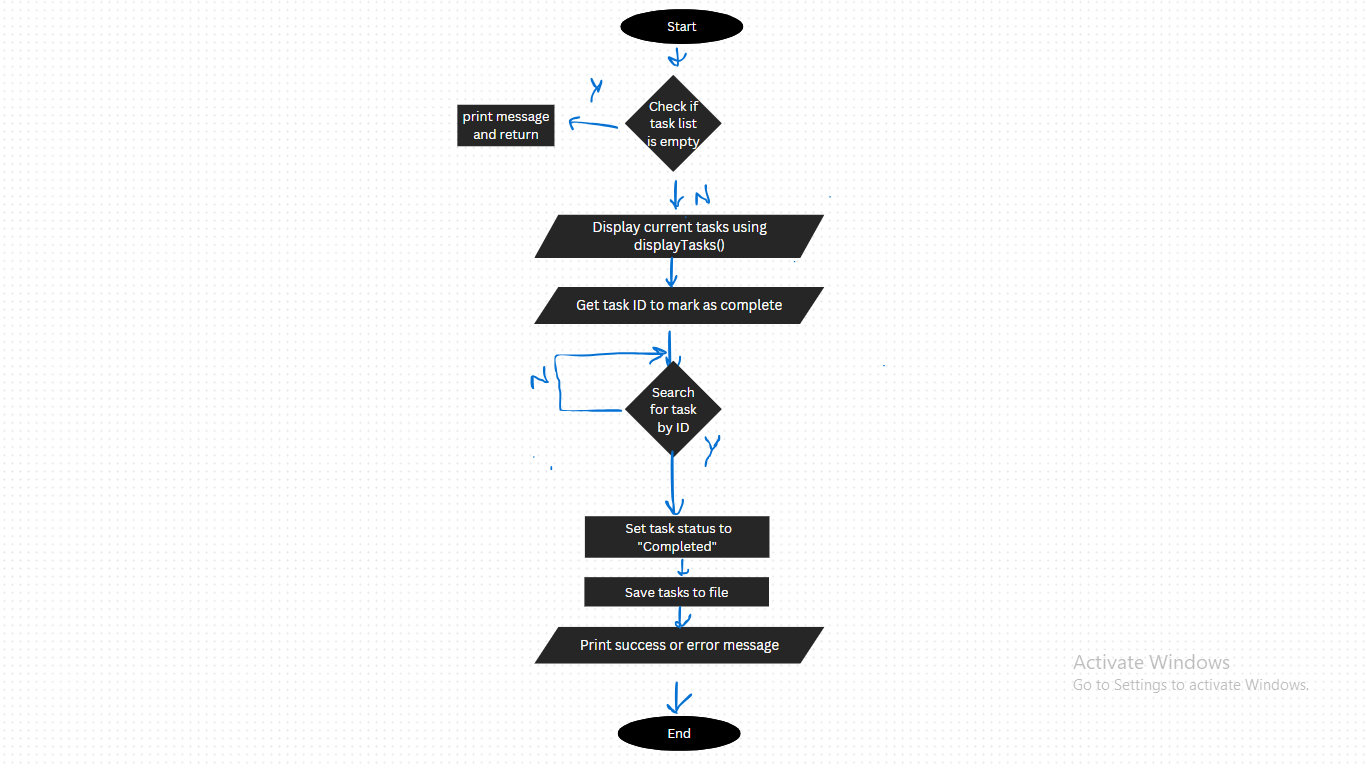
Step 5.1: Set the task status to "Completed".

Step 5.2: Save the updated task list to the file.

Step 5.3: Print a success message.

Step 6: If the task is not found, print an error message.

End



## Algorithm for addNote Function

Start

Step 1: Check if the note list is full. If full, print an error message and return.

Step 2: Prompt the user to enter the note date.

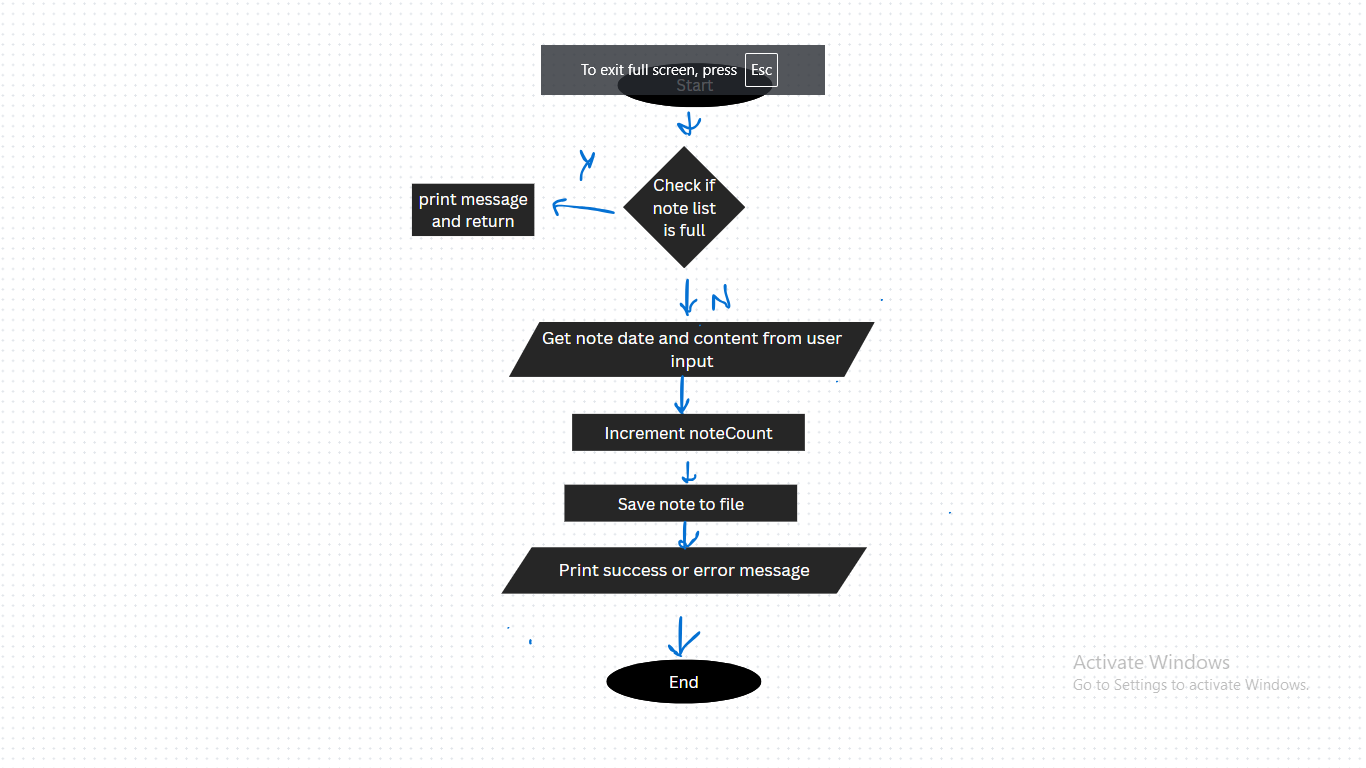
Step 3: Prompt the user to enter the note content.

Step 4: Add the note to the **notes** array and increment **noteCount**.

Step 5: Save the updated note list to the file.

Step 6: Print a success message.

End



## Algorithm for viewNotes Function

Start

Step 1: Check if there are no notes. If none, print a message and return.

Step 2: Prompt the user to enter a date to filter notes or leave blank to view all.

Step 3: Print the table header for notes.

Step 4: Repeat for each note in the **notes** array:

Step 4.1: If the filter date is empty or matches the note date, print the note details.

Step 5: If no notes match the filter date, print a message.

End

## C:\Users\acer\Pictures\Screenshots\Screenshot (29).png

## Algorithm for verifyPassword Function

Start

Step 1: Open the password.txt file in read mode.

Step 2: If the file does not exist, prompt the user to create a new password.

Step 3: If the file exists, read the stored password.

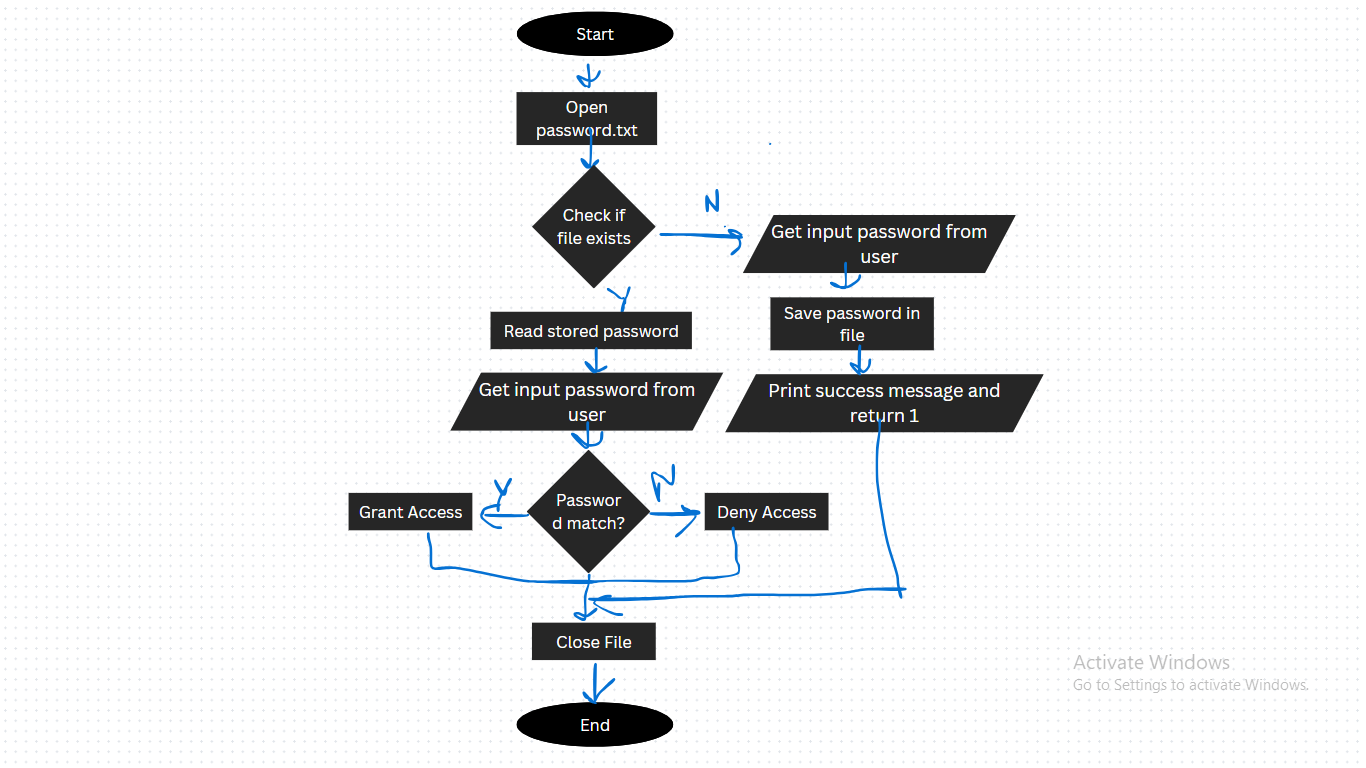
Step 4: Prompt the user to enter the password.

Step 5: Compare the entered password with the stored password.

Step 6: If the passwords match, grant access and return 1.

Step 7: If the passwords do not match, deny access and return 0.

End



## Algorithm for displayMenu Function

Start

Step 1: Clear the screen.

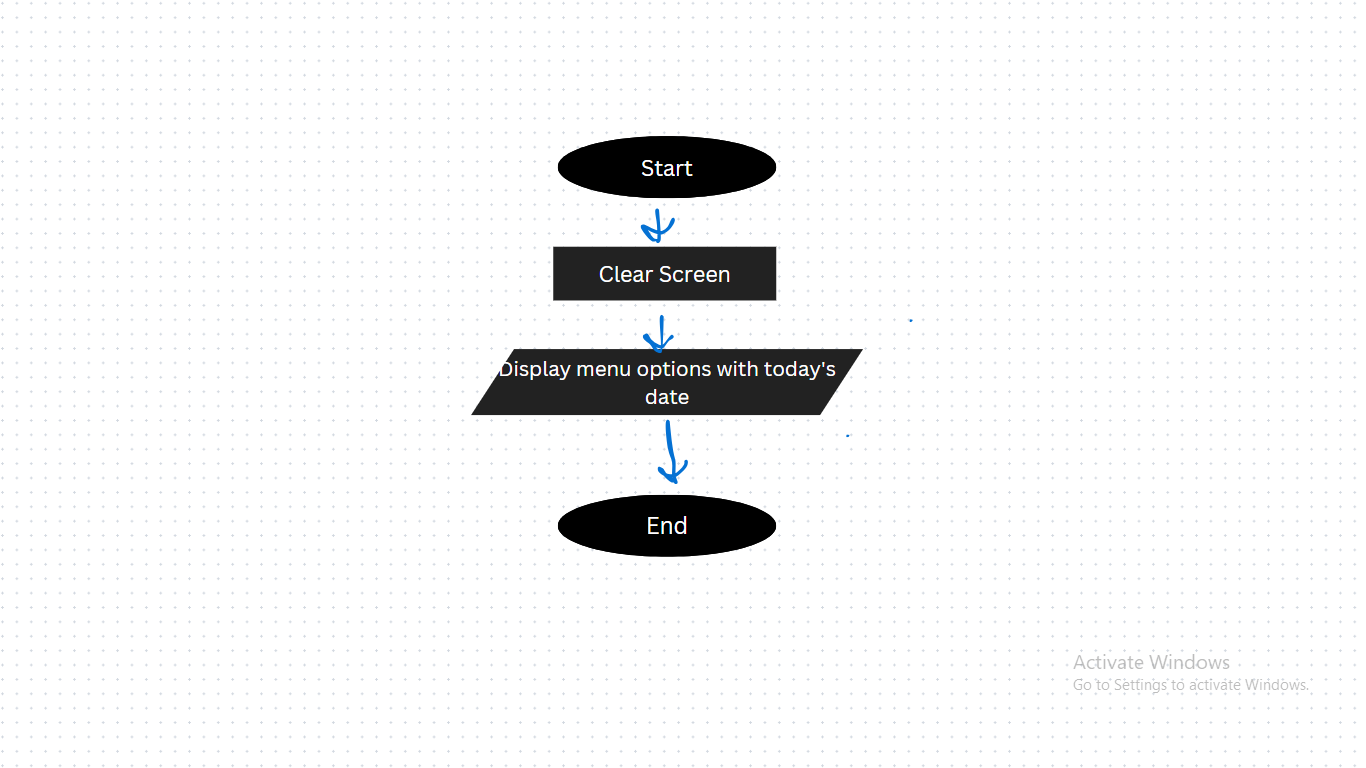
Step 2: Get today’s date using **getTodayDate**.

Step 3: Print the menu header with today’s date.

Step 4: Print the menu options (Add Task, View Tasks, Delete Task, etc.).

Step 5: Prompt the user to enter their choice.

End



## Algorithm for main Function

Start

Step 1: Verify the admin password using **verifyPassword**. If incorrect, exit the program.

Step 2: Load tasks from the file using **loadTasks**.

Step 3: Load notes from the file using **loadNotes**.

Step 4: Repeat until the user chooses to exit:

Step 4.1: Display the menu using **displayMenu**.

Step 4.2: Read the user’s choice.

Step 4.3: Perform the corresponding action based on the user’s choice:

Step 4.3.1: If choice is 1, call **addTask**.

Step 4.3.2: If choice is 2, call **displayTasks**.

Step 4.3.3: If choice is 3, call **deleteTask**.

Step 4.3.4: If choice is 4, call **markCompleted**.

Step 4.3.5: If choice is 5, call **addNote**.

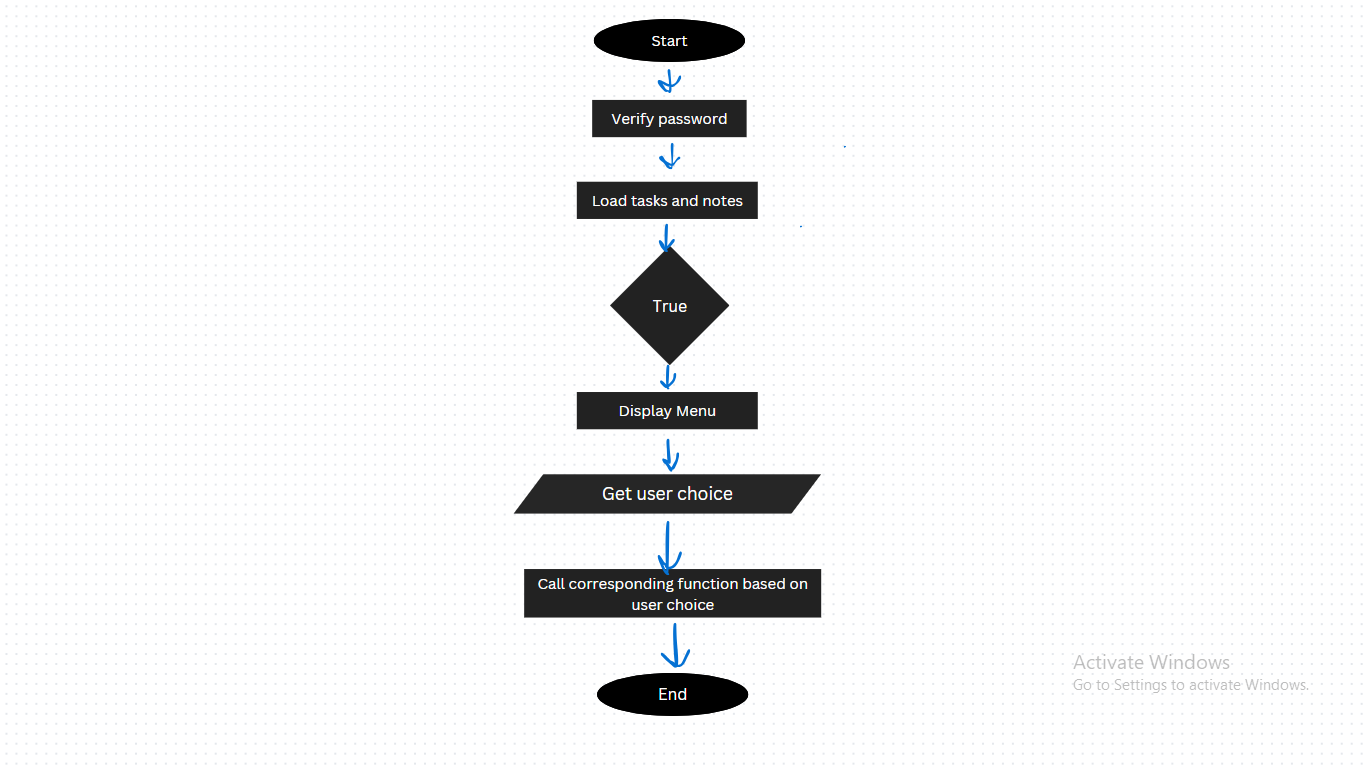
Step 4.3.6: If choice is 6, call **viewNotes**.

Step 4.3.7: If choice is 7, exit the program.

Step 4.3.8: If choice is invalid, print an error message.

Step 4.4: Wait for the user to press Enter before continuing.

End



# SOURCE CODE

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define MAX\_TASKS 100

#define MAX\_NOTES 1000

#define TASK\_FILE "tasks.txt"

#define NOTE\_FILE "notes.txt"

#define PASSWORD\_FILE "password.txt"

// ANSI escape codes for colors

#define COLOR\_RED "\x1b[31m"

#define COLOR\_GREEN "\x1b[32m"

#define COLOR\_YELLOW "\x1b[33m"

#define COLOR\_BLUE "\x1b[34m"

#define COLOR\_MAGENTA "\x1b[35m"

#define COLOR\_CYAN "\x1b[36m"

#define COLOR\_RESET "\x1b[0m"

struct Task {

int id;

char description[100];

char priority[10]; // High, Medium, Low

char status[10]; // Pending, Completed

char deadline[15]; // YYYY-MM-DD

char category[15]; // Work, Study, Personal, Other

};

struct Note {

char date[15]; // YYYY-MM-DD

char content[200]; // Note content

};

struct Task tasks[MAX\_TASKS];

struct Note notes[MAX\_NOTES];

int taskCount = 0;

int noteCount = 0;

// Function to clear the screen (cross-platform)

void clearScreen() {

#ifdef \_WIN32

system("cls");

#else

system("clear");

#endif

}

// Function to safely get today's date

void getTodayDate(char \*todayDate, size\_t size) {

time\_t now = time(NULL);

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

strftime(todayDate, size, "%Y-%m-%d", tm);

}

// Optimized function to load tasks from file

int loadTasks() {

FILE \*file = fopen(TASK\_FILE, "r");

if (!file) return 0;

taskCount = 0;

struct Task tempTask;

while (taskCount < MAX\_TASKS &&

fscanf(file, "%d\n%99[^\n]\n%9[^\n]\n%9[^\n]\n%14[^\n]\n%14[^\n]\n",

&tempTask.id,

tempTask.description,

tempTask.priority,

tempTask.status,

tempTask.deadline,

tempTask.category) == 6) {

tasks[taskCount++] = tempTask;

}

fclose(file);

return 1;

}

// Optimized function to save tasks to file

void saveTasks() {

FILE \*file = fopen(TASK\_FILE, "w");

if (!file) {

perror("Error saving tasks");

return;

}

for (int i = 0; i < taskCount; i++) {

fprintf(file, "%d\n%s\n%s\n%s\n%s\n%s\n",

tasks[i].id, tasks[i].description,

tasks[i].priority, tasks[i].status,

tasks[i].deadline, tasks[i].category);

}

fclose(file);

}

// Optimized function to load notes from file

int loadNotes() {

FILE \*file = fopen(NOTE\_FILE, "r");

if (!file) return 0;

noteCount = 0;

struct Note tempNote;

while (noteCount < MAX\_NOTES &&

fscanf(file, "%14[^:]: %199[^\n]\n",

tempNote.date, tempNote.content) == 2) {

notes[noteCount++] = tempNote;

}

fclose(file);

return 1;

}

// Optimized function to save notes to file

void saveNotes() {

FILE \*file = fopen(NOTE\_FILE, "w");

if (!file) {

printf(COLOR\_RED "Error: Could not save notes to file.\n" COLOR\_RESET);

return;

}

for (int i = 0; i < noteCount; i++) {

fprintf(file, "%s: %s\n", notes[i].date, notes[i].content);

}

fclose(file);

printf(COLOR\_GREEN "Successfully saved %d notes.\n" COLOR\_RESET, noteCount);

}

// Safely calculate deadline countdown

void calculateCountdown(const char \*deadline, char \*countdown, size\_t size) {

struct tm deadline\_tm = {0};

time\_t now = time(NULL);

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

if (sscanf(deadline, "%d-%d-%d",

&deadline\_tm.tm\_year, &deadline\_tm.tm\_mon, &deadline\_tm.tm\_mday) != 3) {

snprintf(countdown, size, "Invalid date");

return;

}

deadline\_tm.tm\_year -= 1900;

deadline\_tm.tm\_mon -= 1;

double seconds = difftime(mktime(&deadline\_tm), now);

if (seconds < 0) {

snprintf(countdown, size, "Deadline passed");

} else {

int days = seconds / 86400;

int hours = (seconds - days \* 86400) / 3600;

snprintf(countdown, size, "%d days, %d hours", days, hours);

}

}

// Safely add a task

void addTask() {

if (taskCount >= MAX\_TASKS) {

printf(COLOR\_RED "Task list is full!\n" COLOR\_RESET);

return;

}

struct Task \*newTask = &tasks[taskCount];

newTask->id = taskCount + 1;

printf(COLOR\_CYAN "Enter task description: " COLOR\_RESET);

if (fgets(newTask->description, sizeof(newTask->description), stdin) == NULL) {

printf(COLOR\_RED "Error reading description.\n" COLOR\_RESET);

return;

}

newTask->description[strcspn(newTask->description, "\n")] = '\0';

printf(COLOR\_CYAN "Enter priority (High/Medium/Low): " COLOR\_RESET);

if (scanf(" %9[^\n]", newTask->priority) != 1) {

printf(COLOR\_RED "Invalid priority.\n" COLOR\_RESET);

return;

}

printf(COLOR\_CYAN "Enter deadline (YYYY-MM-DD): " COLOR\_RESET);

if (scanf(" %14[^\n]", newTask->deadline) != 1) {

printf(COLOR\_RED "Invalid deadline.\n" COLOR\_RESET);

return;

}

printf(COLOR\_CYAN "Enter category (Work/Study/Personal/Other): " COLOR\_RESET);

if (scanf(" %14[^\n]", newTask->category) != 1) {

printf(COLOR\_RED "Invalid category.\n" COLOR\_RESET);

return;

}

strcpy(newTask->status, "Pending");

taskCount++;

saveTasks();

printf(COLOR\_GREEN "Task added successfully!\n" COLOR\_RESET);

}

// Improved task display function

void displayTasks() {

if (taskCount == 0) {

printf(COLOR\_YELLOW "No tasks available!\n" COLOR\_RESET);

return;

}

printf("\n" COLOR\_BLUE "%-5s %-30s %-10s %-10s %-12s %-15s %-20s\n" COLOR\_RESET,

"ID", "Task", "Priority", "Status", "Deadline", "Category", "Time Left");

printf(COLOR\_BLUE "------------------------------------------------------------------------------------------------\n" COLOR\_RESET);

char countdown[50];

for (int i = 0; i < taskCount; i++) {

calculateCountdown(tasks[i].deadline, countdown, sizeof(countdown));

printf("%-5d %-30s %-10s %-10s %-12s %-15s %-20s\n",

tasks[i].id, tasks[i].description,

tasks[i].priority, tasks[i].status,

tasks[i].deadline, tasks[i].category, countdown);

}

}

// Robust task deletion function

void deleteTask() {

if (taskCount == 0) {

printf(COLOR\_YELLOW "No tasks available to delete!\n" COLOR\_RESET);

return;

}

displayTasks(); // Show the current list of tasks

int id;

printf(COLOR\_CYAN "Enter the ID of the task to delete: " COLOR\_RESET);

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

printf(COLOR\_RED "Invalid input! Please enter a valid task ID.\n" COLOR\_RESET);

while (getchar() != '\n'); // Clear the input buffer

return;

}

int found = 0;

for (int i = 0; i < taskCount; i++) {

if (tasks[i].id == id) {

// Shift all tasks after the deleted task one position to the left

for (int j = i; j < taskCount - 1; j++) {

tasks[j] = tasks[j + 1];

}

taskCount--; // Reduce the task count

found = 1;

break;

}

}

if (found) {

// Update the IDs of the remaining tasks

for (int i = 0; i < taskCount; i++) {

tasks[i].id = i + 1; // Assign new sequential IDs

}

saveTasks(); // Save the updated task list to the file

printf(COLOR\_GREEN "Task with ID %d deleted successfully!\n" COLOR\_RESET, id);

} else {

printf(COLOR\_RED "Task with ID %d not found!\n" COLOR\_RESET, id);

}

}

// Improved mark completed function

void markCompleted() {

if (taskCount == 0) {

printf(COLOR\_YELLOW "No tasks available to mark as completed!\n" COLOR\_RESET);

return;

}

displayTasks(); // Show the current list of tasks

int id;

printf(COLOR\_CYAN "Enter the ID of the task to mark as completed: " COLOR\_RESET);

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

printf(COLOR\_RED "Invalid input! Please enter a valid task ID.\n" COLOR\_RESET);

while (getchar() != '\n'); // Clear the input buffer

return;

}

int found = 0;

for (int i = 0; i < taskCount; i++) {

if (tasks[i].id == id) {

strcpy(tasks[i].status, "Completed"); // Update the task status

found = 1;

break;

}

}

if (found) {

saveTasks(); // Save the updated task list to the file

printf(COLOR\_GREEN "Task with ID %d marked as completed!\n" COLOR\_RESET, id);

} else {

printf(COLOR\_RED "Task with ID %d not found!\n" COLOR\_RESET, id);

}

}

// Improved note addition function

void addNote() {

if (noteCount >= MAX\_NOTES) {

printf(COLOR\_RED "Note list is full!\n" COLOR\_RESET);

return;

}

struct Note \*newNote = &notes[noteCount];

printf(COLOR\_CYAN "Enter date (YYYY-MM-DD): " COLOR\_RESET);

if (scanf(" %14[^\n]", newNote->date) != 1) {

printf(COLOR\_RED "Invalid date!\n" COLOR\_RESET);

return;

}

printf(COLOR\_CYAN "Enter note: " COLOR\_RESET);

getchar(); // Clear previous newline

if (fgets(newNote->content, sizeof(newNote->content), stdin) == NULL) {

printf(COLOR\_RED "Error reading note.\n" COLOR\_RESET);

return;

}

newNote->content[strcspn(newNote->content, "\n")] = '\0';

noteCount++;

saveNotes();

printf(COLOR\_GREEN "Note added successfully!\n" COLOR\_RESET);

}

// Improved notes viewing function

void viewNotes() {

if (noteCount == 0) {

printf(COLOR\_YELLOW "No notes available!\n" COLOR\_RESET);

return;

}

char filterDate[15] = {0};

printf(COLOR\_CYAN "Enter date to filter notes (YYYY-MM-DD) or leave blank to view all: " COLOR\_RESET);

fgets(filterDate, sizeof(filterDate), stdin);

filterDate[strcspn(filterDate, "\n")] = '\0'; // Remove the newline character

printf("\n" COLOR\_BLUE "%-15s %-50s\n" COLOR\_RESET, "Date", "Note");

printf(COLOR\_BLUE "------------------------------------------------------------\n" COLOR\_RESET);

int found = 0; // Track if any notes match the filter

for (int i = 0; i < noteCount; i++) {

// If filterDate is empty or matches the note's date, display the note

if (strlen(filterDate) == 0 || strcmp(notes[i].date, filterDate) == 0) {

printf("%-15s %-50s\n", notes[i].date, notes[i].content);

found = 1; // At least one note matches

}

}

if (!found && strlen(filterDate) > 0) {

printf(COLOR\_YELLOW "No notes found for the date '%s'.\n" COLOR\_RESET, filterDate);

}

}

// Secure password verification

int verifyPassword() {

FILE \*file = fopen(PASSWORD\_FILE, "r");

char storedPassword[50] = {0};

char inputPassword[50] = {0};

if (file) {

if (fgets(storedPassword, sizeof(storedPassword), file) == NULL) {

printf(COLOR\_CYAN "No password set. Create a new password: " COLOR\_RESET);

if (scanf(" %49[^\n]", storedPassword) != 1) {

printf(COLOR\_RED "Invalid password.\n" COLOR\_RESET);

fclose(file);

return 0;

}

file = freopen(PASSWORD\_FILE, "w", file);

fprintf(file, "%s", storedPassword);

}

fclose(file);

} else {

printf(COLOR\_CYAN "No password set. Create a new password: " COLOR\_RESET);

if (scanf(" %49[^\n]", storedPassword) != 1) {

printf(COLOR\_RED "Invalid password.\n" COLOR\_RESET);

return 0;

}

file = fopen(PASSWORD\_FILE, "w");

fprintf(file, "%s", storedPassword);

fclose(file);

printf(COLOR\_GREEN "Password set successfully!\n" COLOR\_RESET);

return 1;

}

// Trim newline from stored password

storedPassword[strcspn(storedPassword, "\n")] = '\0';

printf("Enter password: ");

if (scanf(" %49[^\n]", inputPassword) != 1) {

printf(COLOR\_RED "Invalid input.\n" COLOR\_RESET);

return 0;

}

if (strcmp(inputPassword, storedPassword) == 0) {

printf("Access granted!\n");

return 1;

} else {

printf("Incorrect password. Access denied.\n");

return 0;

}

}

// Display menu

void displayMenu() {

clearScreen();

char todayDate[15];

getTodayDate(todayDate, sizeof(todayDate));

printf(COLOR\_MAGENTA "\n========================================\n");

printf(" To-Do List Manager \n");

printf("========================================\n" COLOR\_RESET);

printf(COLOR\_CYAN "Today's Date: %s\n\n" COLOR\_RESET, todayDate);

printf(COLOR\_CYAN "1. Add Task\n");

printf("2. View Tasks\n");

printf("3. Delete Task\n");

printf("4. Mark Task as Completed\n");

printf("5. Add Daily Note\n");

printf("6. View Notes\n");

printf("7. Exit\n" COLOR\_RESET);

printf(COLOR\_MAGENTA "========================================\n" COLOR\_RESET);

printf(COLOR\_CYAN "Enter your choice: " COLOR\_RESET);

}

// Main function

int main() {

if (!verifyPassword()) {

return 0; // Exit if password is incorrect

}

loadTasks();

loadNotes();

int choice;

while (1) {

displayMenu();

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

// Clear invalid input

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

continue;

}

while (getchar() != '\n'); // Clear input buffer after menu choice

switch (choice) {

case 1: addTask(); break;

case 2: displayTasks(); break;

case 3: deleteTask(); break;

case 4: markCompleted(); break;

case 5: addNote(); break;

case 6: viewNotes(); break;

case 7: printf(COLOR\_GREEN "Exiting...\n" COLOR\_RESET); return 0;

default: printf(COLOR\_RED "Invalid choice! Try again.\n" COLOR\_RESET);

}

printf(COLOR\_CYAN "\nPress Enter to continue..." COLOR\_RESET);

getchar(); getchar(); // Wait for user input

}

return 0;

}

**Output/Running of the code**

1. Password craeation if a new user /login:

A black screen with blue text

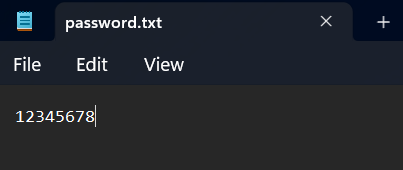
AI-generated content may be incorrect.

1. ADDING Password :

A black background with blue text

AI-generated content may be incorrect.

This file is saved in a new file **password.txt** as:



1. You are logged in and see our Home page which include Todays Date and features like

1. Add Task

2. View Tasks

3. Delete Task

4. Mark Task as Completed

5. Add Daily Note

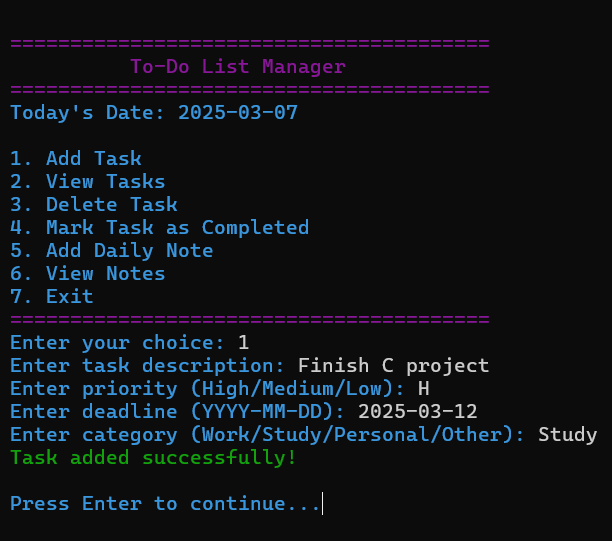
6. View Notes

7. Exit

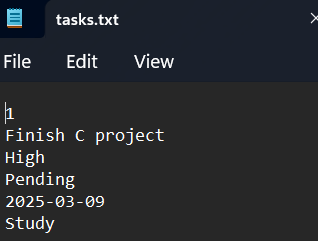
A screenshot of a computer program

AI-generated content may be incorrect.

4. Enter the choice firstly to add task click 1 then add description , select the priority order, deadline, also choose the category which the task belongs :



So a new file name **tasks.txt** is formed in which all this information is stored:

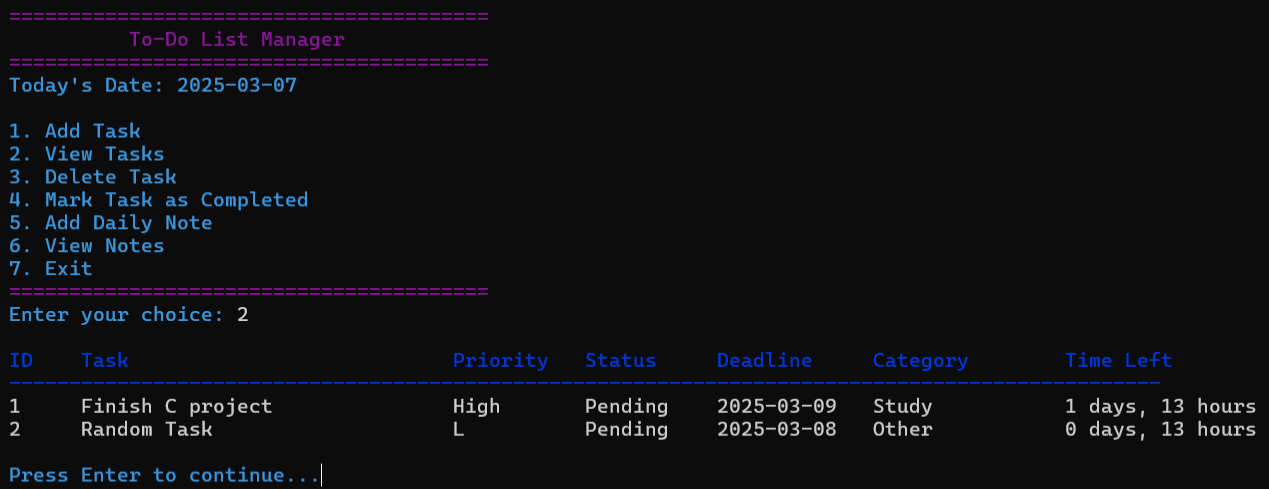


1. Now to view the task in a listing manner firstly enter to go to home screen then press 2:

A screenshot of a computer

AI-generated content may be incorrect.

1. Deleting Unwanted file :  
   Before delete



In file **task.txt before delete**  
A screenshot of a computer program

AI-generated content may be incorrect.

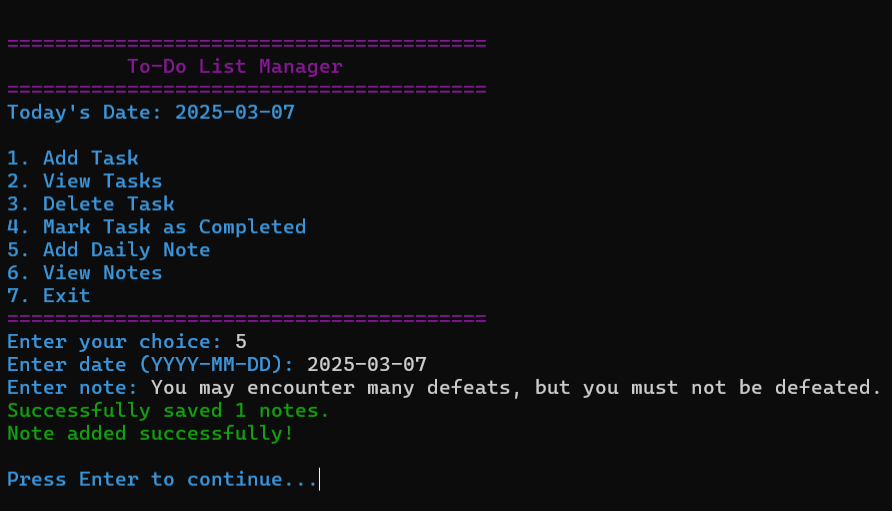
Deleting This with pressing 3 in the choice and pressing the Id of the task which is needed to be deleted in this case 2:  
A screenshot of a computer

AI-generated content may be incorrect.

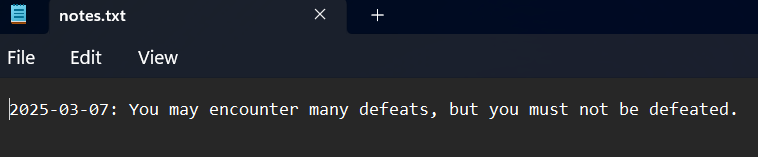
After deletion in **task.txt after delete .**

A screenshot of a computer program

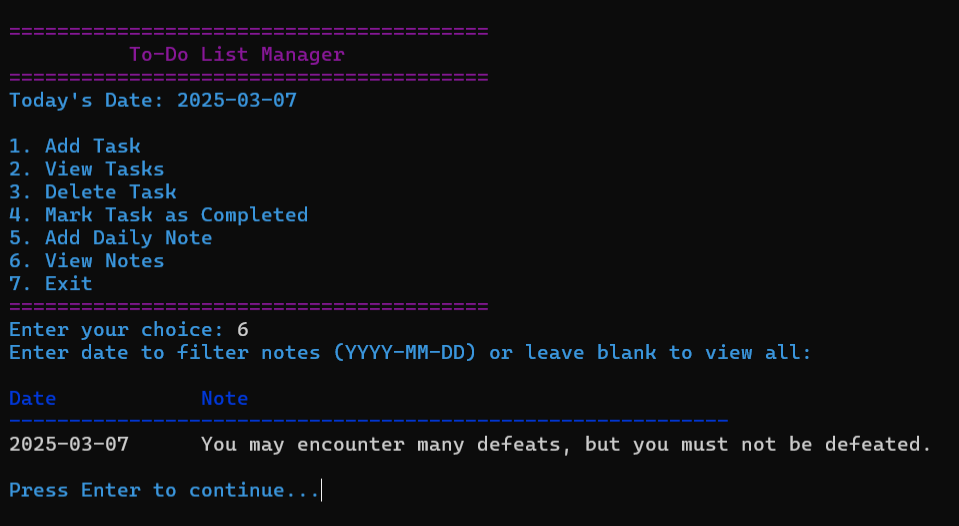
AI-generated content may be incorrect.

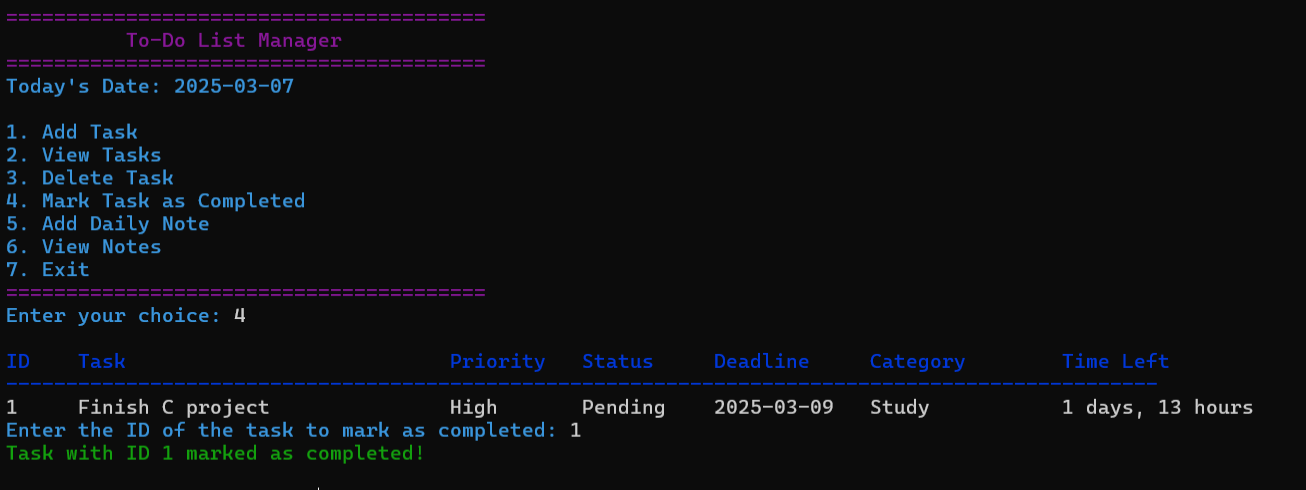
1. Adding Notes by Pressing 5  
   

A new file named **notes.txt** is made if no notes is created before it and the information is stored



1. To view note press 6 and enter the specific date if possible or just enter to view It all:



1. To mark completed task press 4:

Now it is marked completed in the status column

A screenshot of a computer

AI-generated content may be incorrect.



**Result**

The Taskbar Management System was successfully implemented using C programming language. The system provides a user-friendly interface to efficiently manage daily tasks and notes. The following key functionalities were tested and verified:

1. **Task Management:**
   * Users can add, view, delete, and mark tasks as completed.
   * Tasks include attributes like priority, status, deadline, and category.
   * A countdown timer for deadlines ensures timely task completion.
2. **Note Management:**
   * Users can add daily notes with timestamps.
   * Notes are stored and retrieved efficiently from a file.
   * Filtering notes by date allows easy organization.
3. **File Handling:**
   * Data persistence is achieved using file operations.
   * The application successfully loads and saves tasks and notes to external text files.
4. **Security Feature:**
   * A password protection system restricts unauthorized access.
   * Users are prompted to set and verify passwords, which are stored securely.
5. **User Interface & Usability:**
   * Clear instructions and a color-coded console UI enhance readability.
   * Proper input validation prevents crashes due to invalid inputs.
   * Cross-platform compatibility is ensured using standard C functions.

**Discussion**

The **Taskbar Management System** successfully fulfills the primary objective of organizing tasks and notes in an efficient manner. The system leverages fundamental **C programming concepts**, including **structures, file handling, loops, conditionals, and memory management.** Below is a discussion of the strengths and areas for improvement:

* ***Strengths:***
* **Efficient Task Handling:** The system efficiently manages a large number of tasks and notes (up to 100 tasks and 1000 notes).
* **Deadline Calculation:** The countdown feature helps users track their pending tasks effectively.
* **Data Persistence:** File handling techniques ensure that tasks and notes are not lost after exiting the program.
* **Security Integration:** A password system adds an extra layer of security to the task and note management system.
* **User-Friendly Interface:** The use of color-coded outputs and structured formatting makes interaction intuitive.
* ***Challenges and Limitations:***
* **Limited Storage Capacity:** The system is limited by predefined array sizes (MAX\_TASKS and MAX\_NOTES). Future improvements could use **dynamic memory allocation** to allow unlimited entries.
* **No GUI Support:** The application is **text-based**. Implementing a GUI (e.g., with GTK or C++ Qt) could enhance user experience.
* **Password Security:** The password system does not use encryption. Implementing **hashing techniques (e.g., SHA-256)** could improve security.
* **No Sorting or Search Features:** Tasks and notes are stored sequentially without sorting or searching capabilities. Implementing sorting by priority or deadline and adding a search feature would improve usability.

**Conclusion and Future Scope**

The Taskbar Management System is a robust C-based project that successfully manages tasks and notes while incorporating essential programming concepts. The project meets the fundamental requirements of task tracking and daily note-taking while ensuring data persistence and security.

For future enhancements, the following improvements could be considered:

1. Implementing a Graphical User Interface (GUI) for better usability.
2. Integrating Sorting and Searching features for efficient task management.
3. Using Database Storage (e.g., SQLite) instead of text files for better scalability.
4. Enhancing Security using encryption techniques for password protection.
5. Adding Notifications or Alerts for approaching deadlines using system APIs.

Overall, this project serves as an excellent foundation for further enhancements and real-world applications in task management systems.

**References**