A
Project Report
Of
Software Development Fundamentals Lab – I
[15B17CI171]

SDF MINI PROJECT {20-25 Nov 2023}

Jaypee Institute of Information Technology
Sector-62, Noida
(U.P.-201309)

<u>TEAM Members:</u>

S.No.	Enrolment No. (Roll No.)	Name
1	23103278	Kush Kansal
2	23103298	Aaditya Pratap Singh
3	23103374	Vasu Tayal

Batch - B10(B.Tech CSE) **SEM-1** **(ODD 2023)**

Language Used: C

Title: TaskMaster-X

Synopsis:-

Do you feel overwhelmed by the many tasks, deadlines, and commitments in your life? Do you wish you had a simple way to organize and manage everything? If so, we have a solution **“TaskMaster X”** that helps you keep track of your goals. TaskMaster X will be written in C, a fast-efficient programming language that can handle complex problems. With TaskMaster X, you will be able to easily create, edit, and prioritize your tasks, set reminders and notifications, and track your progress and performance. **TaskMaster X is more than just a to-do list app; it is a smart and reliable solution that can help you achieve more in less time.**

Our to-do list project boasts several key features:

✔ **Task Management:** Users can efficiently add tasks, providing a title, description, due date, and assigned time. This helps users organize their responsibilities effectively.
✔ **Task Editing and Removal:** Users can easily edit or remove tasks as needed to adapt to changing requirements and priorities.
✔ **User Authentication:** To enhance security and user-friendliness, the application includes a login system where users can access their tasks securely using a username and password.
✔ **Reminder Feature:** Taskmaster X incorporates a reminder system that alerts users to complete tasks within the assigned time limit. This ensures that important deadlines are not missed.
✔ **Task Sorting and Filtering:** Users can sort their tasks by priority, due date, and categories, making it simple to prioritize and manage their to-do list effectively.

✔ **Task Status Tracking:** The application allows users to view the completion status of their tasks, providing a clear overview of their progress.
✔ **Data Persistence:** Data backup and persistence are crucial for security and data integrity. Taskmaster X ensures that task data is stored correctly.
✔ **Feedback Support:** To improve the app's usability and features, Taskmaster X includes a feedback system where users can provide input and suggestions for enhancements.

By incorporating these features into our C-based project we will create a robust and versatile task management tool that caters to various user needs and preferences.

Topics of SDF-1 used:

This project would include topics used in the course curriculum of 'C' language such as

- File Handling
- Control Flow
- Arrays
- Strings
- Functions
- Structures and Unions
- Pointers
- Other fundamentals of 'C' like data types, statements, operators etc.

Design of the project:

System Architecture

The To-Do List application is implemented in C. The main components include task management functions, user interface functions, and the main program loop.

1. User Interface

The user interface (UI) component is crucial for providing a seamless and intuitive interaction between the user and the To-Do List application. In this console-based project, the UI is primarily text-based, utilizing the console window to display information and capture user input. The homepage and the mainMenu() function is responsible for presenting a clear and organized menu to the user. The menu outlines the available options, providing a visual guide for the user to navigate through the application. Each option is numbered, making it easy for the user to input their choice. The UI prompts the user with meaningful messages and instructions, guiding them through the different functionalities of the application.

2. Structures

The structure Task encapsulates the essential attributes of a task. The description field is an array of characters to store the task's description, and the completed field is an integer indicating whether the task is completed (1) or not (0),etc. This structure provides a convenient way to organize and manipulate task-related data.

Similarly,the structure Credentials encapsulates the essential attributes of a user login details.

The struct Task encapsulates all relevant information about a task in a single unit. This design choice promotes clarity, simplicity, and ease of manipulation within the program. Each element of the structure plays a specific role in representing and managing tasks:

- Task Description (description): This field stores a textual description of the task.
- Task Completion Status (completed): This field provides a binary indicator of whether the task has been completed.
- If additional information about tasks needs to be tracked in the future (e.g., due dates, priorities), the struct Task can be extended without significantly impacting the existing codebase. This makes the data structure adaptable to future requirements.

The use of a struct allows for a straightforward and readable representation of tasks. The fields are self-explanatory, making the code more accessible to developers and contributors.

3. Functionality

3.1 Adding a Task (addTask)

The addTask function captures the user's input for a task description using the scanf function. It then adds a new task to the task file, initializing the completion status to 0. This functionality allows users to easily populate their to-do list with new tasks.

3.2 Viewing Tasks (seeToDoList)

The seeTasks function iterates through the task list, displaying each task's description and completion status. If there are no tasks, a message informs the user that no tasks are available. This functionality provides users with a quick overview of their current tasks.

3.3 Marking a Task as Completed (markCompleted)

The markCompleted function prompts the user to select a task by displaying the current list of tasks. Upon user input, it updates the completion status of the selected task. This feature enables users to track their progress by marking tasks as completed.

4. Key Features

4.1 Task Priority

To enhance task management, future iterations of the application could include a priority system for tasks. This would allow users to assign different levels of importance to tasks, aiding in better organization and prioritization. The struct Task could be extended to include a priority field, and functions such as addTask and seeToDoList could be updated accordingly.

4.2 File I/O Operations

Implementing file I/O operations would provide users with the ability to save their to-do list to a file and load it later. This feature ensures data persistence between program runs. The application could use functions like fopen, fwrite, and fread to write and read task data to and from a file. This enhancement would make the application more practical for long-term use.

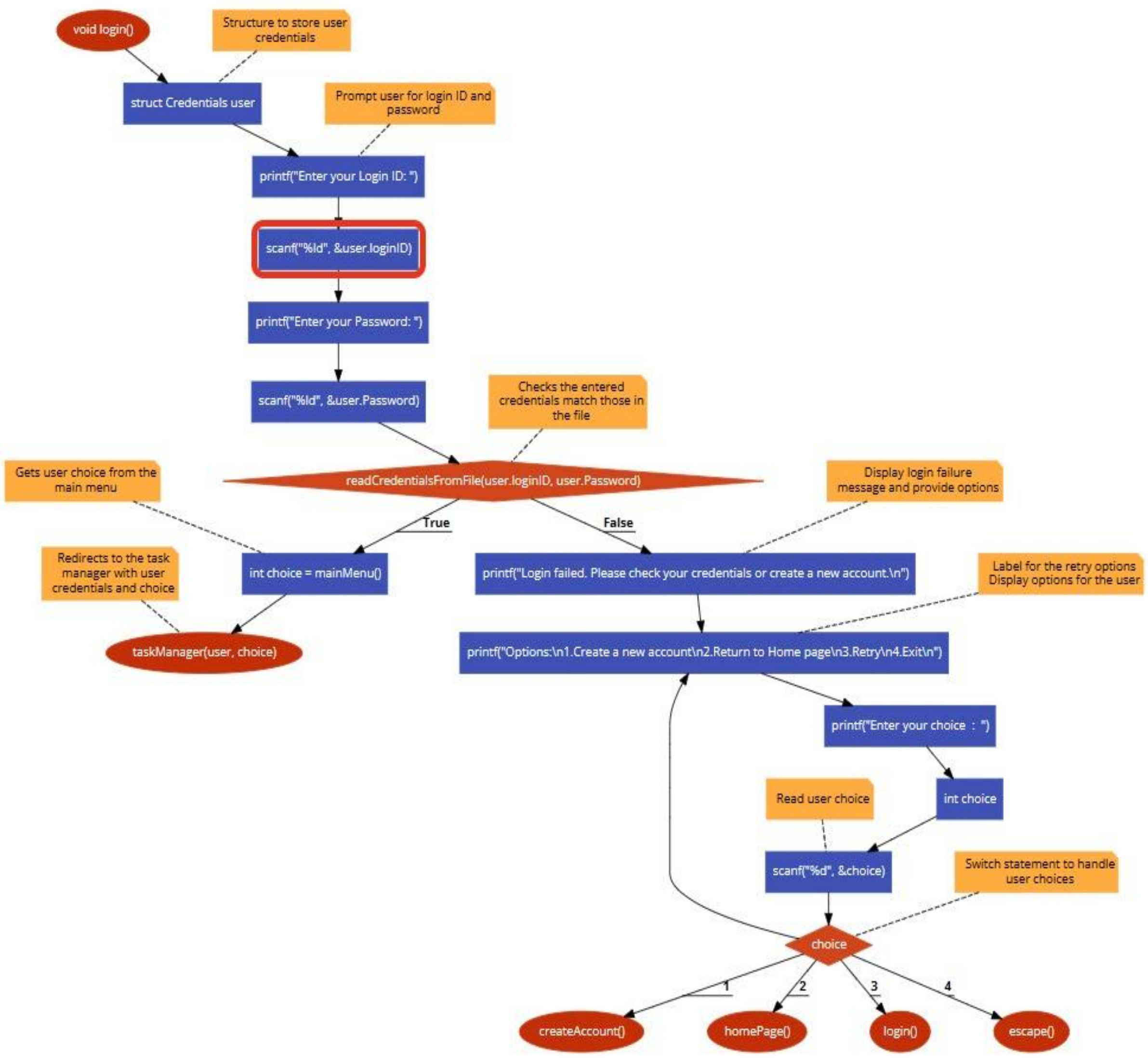
4.3 Due Dates and Times

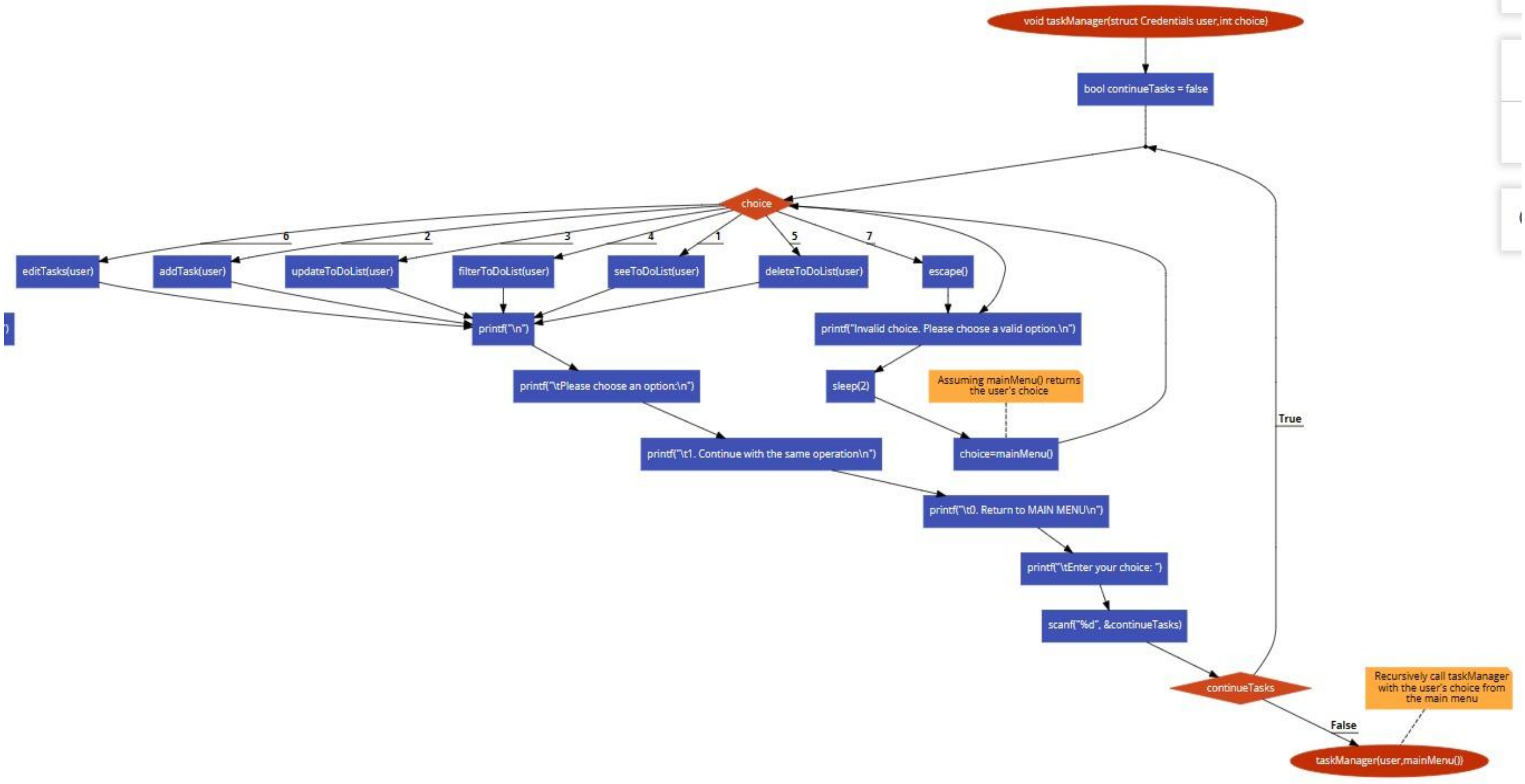
Adding due dates and times to tasks would introduce a temporal dimension to the to-do list. Users could specify when tasks need to be completed, and the application could provide reminders or sort tasks based on their due dates by using filterToDoList() function. The struct Task would need to be expanded to include fields for due dates and times, and the corresponding functions would be updated to accommodate this additional information.This is how the calender integration is being implemented.

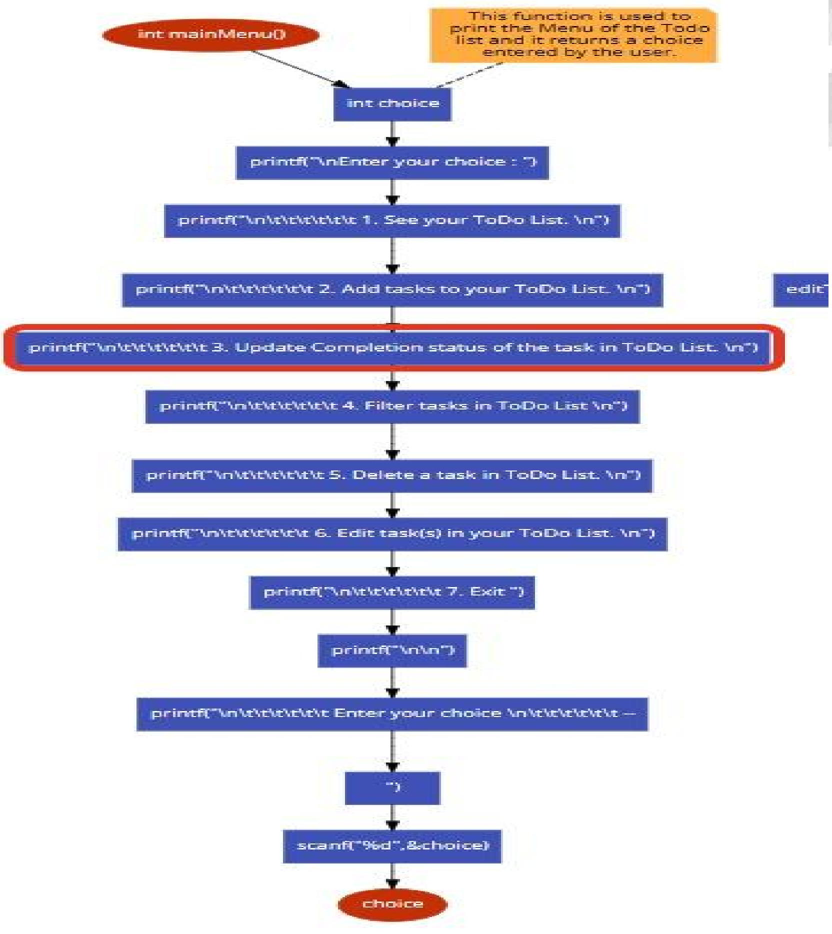
5. Conclusion

The To-Do List project in C serves as a foundational console application for task management. Its modular design allows for easy maintenance and future enhancements. The current functionality provides users with a straightforward means of adding, viewing, and marking tasks as completed. As the application evolves, the incorporation of additional features will further enhance its utility and user experience.

Flow charts representing the design of the above project are shown below:


void login()
Structure to store user credentials
struct Credentials user
Prompt user for login ID and password
printf("Enter your Login ID: ")
scanf("%ld", &user.loginID)
printf("Enter your Password: ")
scanf("%ld", &user.Password)
Checks the entered credentials match those in the file
readCredentialsFromFile(user.loginID, user.Password)
Gets user choice from the main menu
True
False
Display login failure message and provide options
Redirects to the task manager with user credentials and choice
int choice = mainMenu()
printf("Login failed. Please check your credentials or create a new account.\n")
Label for the retry options Display options for the user
taskManager(user, choice)
printf("Options:\n1.Create a new account\n2.Return to Home page\n3.Retry\n4.Exit\n")
printf("Enter your choice : ")
Read user choice
int choice
scanf("%d", &choice)
Switch statement to handle user choices
choice
1
2
3
4
createAccount()
homePage()
login()
escape()


void taskManager(struct Credentials user,int choice)
bool continueTasks = false
choice
6
2
3
4
1
5
7
editTasks(user)
addTask(user)
updateToDoList(user)
filterToDoList(user)
seeToDoList(user)
deleteToDoList(user)
escape()
printf("\n")
printf("Invalid choice. Please choose a valid option.\n")
printf("\tPlease choose an option:\n")
sleep(2)
Assuming mainMenu() returns the user's choice
printf("\t1. Continue with the same operation\n")
choice=mainMenu()
printf("\t0. Return to MAIN MENU\n")
printf("\tEnter your choice: ")
scanf("%d", &continueTasks)
True
continueTasks
Recursively call taskManager with the user's choice from the main menu
False
taskManager(user,mainMenu())


int mainMenu()
This function is used to print the Menu of the Todo list and it returns a choice entered by the user.
int choice
printf("\nEnter your choice : ")
printf("\n\t\t\t\t\t\t 1. See your ToDo List. \n")
printf("\n\t\t\t\t\t\t 2. Add tasks to your ToDo List. \n")
printf("\n\t\t\t\t\t\t 3. Update Completion status of the task in ToDo List. \n")
printf("\n\t\t\t\t\t\t 4. Filter tasks in ToDo List \n")
printf("\n\t\t\t\t\t\t 5. Delete a task in ToDo List. \n")
printf("\n\t\t\t\t\t\t 6. Edit task(s) in your ToDo List. \n")
printf("\n\t\t\t\t\t\t 7. Exit ")
printf("\n\n")
printf("\n\t\t\t\t\t\t Enter your choice \n\t\t\t\t\t\t --
")
scanf("%d",&choice)
choice

Implementation Details

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>
#include<unistd.h>
#include<conio.h>
#include<math.h>

// it Defines a structure for user credentials
struct Credentials
{
    long loginID;
    long Password;
    bool verified;
    char name[50];
};

// Define a structure for tasks
struct Task
{
    char taskname[20];
    char description[500];
    int percent_complete;
    bool completed;
    int priority;    // Set priority on a scale of 1-5
    int lastDate;    // Last date to complete the task
    int time;        // Last time to complete the task
    char category[20];
};

// Function declarations
void escape();
void login();
void createAccount();
void guest();
void homePage();
int mainMenu();
bool validate(int dob);
bool loginIDExists(long loginID);
void taskManager(struct Credentials user, int choice);

int seeToDoList(struct Credentials user);
void updateToDoList(struct Credentials user);
void filterToDoList(struct Credentials user);
void deleteToDoList(struct Credentials user);
void editTasks(struct Credentials user);
bool validateName(char* name);
void addTask(struct Credentials user);

int main()
{
   homePage();
   return 0;
}

// Function  displays the home page and take user input actions
void homePage()
{
   system("cls");
   system("color 17");

   int choice;

   // Displaying the TaskMaster X logo and menu options
   printf("\n
_______________________________________________________________________\n");
   printf("||                                                              ||\n");
   printf("||  \\      /\\      / |===== |      |====== /=====\\ |\\    /| |=====    ||\n");
   printf("||   \\    /  \\    /  |_____ |      |       |     | | \\  / | |_____    ||\n");
   printf("||    \\  /    \\  /   |      |      |       |     | |  \/  | |         ||\n");
   printf("||     \/      \/    |===== |_____ |====== \\=====/ |      | |=====    ||\n");
   printf("||                          TO TaskMaster X                       ||\n");
   printf("||     MENU:-                                                     ||\n");
```

```
    printf("||   1. Login                                        ||\n");
    printf("||   2. Create a new account                                 ||\n");
    printf("||   3. CLOSE TASKMASTER X                                     ||\n");

printf("||___________________________________________________________||
\n");
    printf("ENTER YOUR CHOICE: ");
    scanf("%d", &choice);
    printf("\nLoading.....\n");
    sleep(1);

 // Switch case made to handle user choices
    switch(choice)
    {
        case 1:
            login();
            break;
        case 2:
            createAccount();
            break;
        case 3:
            escape();
            break;
        default:
            escape();
    }
}

// Function to display an exit confirmation screen
void escape()
{
    // asks user for exit confirmation
    printf("\nAre you sure you want to exit?(YES/NO)---");
    char ch;

    // Read the user's choice (consume newline characters)
    scanf("%c", &ch);
    scanf("%c",&ch);

    // Check if the user chose to exit
```

```
  if(ch == 'Y' || ch == 'y')
 {

    system("cls");
    system("color 3E");
      printf("\n\t\t\t\t\t\t
___________________________________________________________________________");
      printf("\n\t\t\t\t\t\t||                                                    ||");
      printf("\n\t\t\t\t\t\t|| ---------- |    |   /\\   |\\   | |  /     \\    / |=====| |    |  ||");
      printf("\n\t\t\t\t\t\t||    ||    |    |  /  \\  | \\  | | /      \\  /  |    | |   |  ||");
      printf("\n\t\t\t\t\t\t||    ||    |=====|  /====\\  |  \\  | |/        \\=/  |    | |   |  ||");
      printf("\n\t\t\t\t\t\t||    ||    |    | /     \\ |   \\ | |\\         |   |    | |   |  ||");
      printf("\n\t\t\t\t\t\t||    ||    |    |/       \\|    \\| | \\        |   |=====| |====|   ||");

printf("\n\t\t\t\t\t\t||___________________________________________________________________|
|\n");

     // Pause for 2 seconds
     sleep(1);
     system("cls");

     // Display a feedback prompt
     printf("\n\n");
     system("color 2E");
     int choice = 1;
     printf("Would you like to provide feedback? (1: Yes, 2: No): ");
     fflush(stdin);
     scanf("%d", &choice);

     // If the user chooses to provide feedback
     if(choice == 1)
     {
        // Feedback form
        char feedback[1000];
        printf("\nPlease provide your feedback (up to 999 characters):\n");
        fflush(stdin);
        fgets(feedback, sizeof(feedback), stdin);

      // Saves the feedback to a file
     size_t feedback_length = strlen(feedback);
     if(feedback_length > 0 && feedback[feedback_length - 1] == '\n')
     {
            feedback[feedback_length - 1] = '\0';
        }

        FILE *feedbackFile = fopen("Program Data/feedback.txt", "a");
        if(feedbackFile == NULL)
        {
            printf("Error saving feedback. Please try again later.\n");
        }
        else
        {
            fprintf(feedbackFile,"\n%s\n",feedback);
            fclose(feedbackFile);
            printf("Thank you for your feedback! It has been saved in
feedback.txt.\n");
        }

            // Prompt the user to rate the program
            printf("\nRATE US NOW!!\n");
            printf("1. *\n");
            printf("2. **\n");
            printf("3. ***\n");
            printf("4. ****\n");
            printf("5. *****\n");
            int rate;
            scanf("%d", &rate);
        }

        // Display a goodbye message
        printf("\nGoodbye! Have a great day!\n");
        // Pause for 2 seconds
        sleep(1);
        // Clears the screen
        system("cls");
        // Exit the program
        exit(0);
    }

  else
  {
    // If the user chooses not to exit, it goes back to the home page
    homePage();
  }
  return;
}

// Function to display the main menu and get user choice
int mainMenu()
{
    //This function is used to print the Menu of the Todo list and it
returns a choice entered by the user.
    int choice;

    system("cls");
    system("color 4F");

    printf("\n");

    printf("\t\t\t\t\t****************************************\n");
    printf("\t\t\t\t\t***                                  ***\n");
    printf("\t\t\t\t\t***           MAIN MENU              ***\n");
    printf("\t\t\t\t\t***                                  ***\n");
    printf("\t\t\t\t\t****************************************\n");

    printf("\nEnter your choice : ");
    printf("\n\t\t\t\t\t\t\t 1. See your ToDo List. \n");
    printf("\n\t\t\t\t\t\t\t 2. Add tasks to your ToDo List. \n");
    printf("\n\t\t\t\t\t\t\t 3. Update Completion status of the task in ToDo
List. \n");
    printf("\n\t\t\t\t\t\t\t 4. Filter tasks in ToDo List \n");
    printf("\n\t\t\t\t\t\t\t 5. Delete a task in ToDo List. \n");
    printf("\n\t\t\t\t\t\t\t 6. Edit task(s) in your ToDo List. \n");
    printf("\n\t\t\t\t\t\t\t 7. Exit ");
    printf("\n\n");
    printf("\n\t\t\t\t\t\t\t Enter your choice \n\t\t\t\t\t\t\t --> ");
    scanf("%d",&choice);
    return choice;
}

void writeCredentialsToFile(long loginID, long Password)
{
    // Open file for appending in the "Program Data" directory
    FILE *file=fopen("Program Data/Credentials.txt","a");

    if(file==NULL)
    {
        printf("Error opening the file for writing.\n");
        return;
    }
    // Write login credentials in the following format "loginID,Password"
    fprintf(file,"%ld,%ld\n",loginID,Password);
     // Flush here ensures immediate writing to the file
    fflush(file);
    fclose(file);
}


bool readCredentialsFromFile(long loginID, long Password)
{
    // Open file for reading in the "Program Data" directory
    FILE *file = fopen("Program Data/Credentials.txt","r");
     // Variables to store login credentials read from the file
    long storedLoginID, storedPassword;
    // Flag here indicate whether matching credentials are found
    bool found = false;
    char line[100];

     // Read each line in the file
    while(fgets(line,sizeof(line),file)!=NULL)
    {
         // Extract stored loginID and Password from the line
        sscanf(line,"%ld,%ld",&storedLoginID,&storedPassword);
        // Check if entered credentials match the stored credentials
        if(loginID==storedLoginID&&Password==storedPassword)
        {  //sets the found flag to be true and exits the loop
            found=true;
            break;
        }
    }
    fclose(file);
```

```c
     // Return true if matching credentials are found, false otherwise
    return found;
}

// Function to  make use of login process
void login()
{
    // Pause for 2 seconds, clear screen, and set console text color
    sleep(1);
    system("cls");
    system("color E1");

    // Structure to store user credentials
    struct Credentials user;

    // Display login header
    printf("******************************************\n");
    printf("*                  Login                 *\n");
    printf("******************************************\n");

    // Prompt user for login ID and password
    printf("Enter your Login ID: ");
    scanf("%ld", &user.loginID);
    printf("Enter your Password: ");
    scanf("%ld", &user.Password);

    // Checks the entered credentials match those in the file
    if (readCredentialsFromFile(user.loginID, user.Password))
    {
        // Displays successful login message
        printf("******************************************\n");
        printf("*          Login Successful      *\n");
        printf("******************************************\n");
        printf("\nMAIN MENU Loading.....   \n");

        // Pause for 5 seconds
        sleep(2);

        // Gets user choice from the main menu
        int choice = mainMenu();
        // Redirects to the task manager with user credentials and choice
        taskManager(user, choice);
    }
    else
    {
        // Display login failure message and provide options
        printf("Login failed. Please check your credentials or create a new
account.\n");

        // Label for the retry options
        DEFAULT:

        // Display options for the user
        printf("Options:\n1.Create a new account\n2.Return to Home
page\n3.Retry\n4.Exit\n");
        printf("Enter your choice  :  ");

        int choice;

        // Read user choice
        scanf("%d", &choice);

        // Switch statement to handle user choices
        switch (choice)
        {
            case 1:
                createAccount();
                break;
            case 2:
                homePage();
                break;
            case 3:
                login();
                break;
            case 4:
                escape();
                break;
            default:
                goto DEFAULT;
        }
    }
}

// Function to create a new user account
void createAccount()
{   //variables stores user info
    char name[50];
    int dob = 0;
    struct Credentials newUser;
    sleep(1);
    system("cls");
    system("color 2f");
    struct Credentials user;

   // Display create account header
    printf("***************************************************\n");
    printf("*         CREATE AN ACCOUNT          *\n");
    printf("***************************************************\n");
    printf("To Create a New Account\n");
    retry1:
    printf("Enter your first name: ");
    scanf("%49s", name);
     // Validate the entered name
    if(validateName(name))
    {
        // Copy validated name to the newUser structure
        strcpy(newUser.name,name);
        printf("----------Name Validated Successfully----------- \n");
    }
    else
    {   // Display error message for an invalid name and retry
        printf("\nInvalid Name. Please use only letters and spaces.Retry.....");
        goto retry1;
    }

     // Label for the date of birth validation retry
    retry2:
    printf("Enter your date of birth (format DDMMYY): ");
    scanf("%d", &dob);
```

```
 // Validate the entered date of birth
if(!validate(dob))
{
   // Display error message for an invalid date of birth and retry
   printf("\nWrong Date of birth......Retry.....");
   goto retry2;
}
else
{
   printf("-----------Date Validated Successfully------------ \n");
}
long newLoginID;

while(1)
{
   printf("Enter a new Login ID: ");
   scanf("%ld", &newLoginID);

   // Check if the entered login ID already exists
   if(loginIDExists(newLoginID))
      printf("Login ID already exists. Please choose a different one.\n");
   else
   {
     // Set the new login ID in the newUser structure and break the
loop
      newUser.loginID = newLoginID;
      break;
   }
}
printf("Enter a new Password: ");
scanf("%ld", &newUser.Password);

 // Set the verified flag to true
newUser.verified = true;

// Write the new user credentials to the file
writeCredentialsToFile(newUser.loginID, newUser.Password);
printf("\nAccount created successfully!\n");
```

```
    printf("Now you will be guided to the login page.\nEnter your
credentials there in order to LOGIN\n");
    // Pause for 2 seconds and redirect to the login page
    sleep(2);
    login();
}

// Function to validate date of birth
bool validate(int dob)
{   //extracting day,month,year from dob
    int day=dob/10000;
    int month=(dob/100)%100;
    int year=dob%100;
    // Checkinging for valid year, month, and day
    if(year<0||year>99||month<1||month>12||day<1||day>31)
    {
         return false;
    }
   // Checking for specific month-day combinations

if((month==2&&day>29)||((month==4||month==6||month==9||month==11)&&
day>30))
    {
        return false; //invalid date
    }
    return true;   // valid date
}

// Function to validate a name
bool validateName(char* name)
{
    int i;
    // Checking if the name is not empty
    if(name[0]=='\0')
        return false;
    // Iterating through each character in the name
    for(i=0;name[i]!='\0';i++)
    {
        char ch=name[i];
        //checking character is uppercase and lowercase
        if(!((ch>='A'&&ch<='Z')||(ch>='a'&&ch<='z')||ch==' '))
            return false;
    }
    return true;
}

// Function to check if a login ID exists in the credentials file
bool loginIDExists(long loginID)
{
    FILE *file = fopen("Program Data/Credentials.txt","r");
    long storedLoginID, storedPassword;
    bool found=false;
    char line[100];

     // Reading each line from the file
    while(fgets(line,sizeof(line),file)!=NULL)
    {
        // Extracting stored login ID and password from the line
        sscanf(line,"%ld,%ld",&storedLoginID,&storedPassword);
        // Checking if the provided login ID matches any stored login ID
        if(loginID==storedLoginID)
        {
            found = true;
            break;
        }
    }
    fclose(file);
    return found; // Return true if login ID exists, false otherwise
}

// Function to manage tasks based on user's choice
void taskManager(struct Credentials user,int choice)
{
    bool continueTasks = false;
    do
    {
        retry:
        switch(choice)
        {
            case 1:
                seeToDoList(user);
                break;
            case 2:
                addTask(user);
                break;
             case 3:
                updateToDoList(user);
                break;
            case 4:
                filterToDoList(user);
                break;
            case 5:
                deleteToDoList(user);
                break;
            case 6:
                editTasks(user);
                break;
            case 7:
                escape();
            default:
                printf("Invalid choice. Please choose a valid option.\n");
                sleep(2);
                choice=mainMenu();    // Assuming mainMenu() returns the user's choice
                goto retry;
        }

        printf("\n");
        printf("\tPlease choose an option:\n");
        printf("\t1. Continue with the same operation\n");
        printf("\t0. Return to MAIN MENU\n");
        printf("\tEnter your choice: ");
        scanf("%d", &continueTasks);

    }while(continueTasks);

    taskManager(user,mainMenu()); // Recursively call taskManager with the user's choice from the main menu
}
```

```
// Function to display the ToDo list for a user
int seeToDoList(struct Credentials user)
{
   int serial=0;
   char filename[20];
   sprintf(filename, "Program Data/%ld.txt", user.loginID); // Generate the
file's name based on the user's loginID
   system("cls");
   system("color 5a");
   FILE *file =fopen(filename,"r");

   if(file==NULL)
   {
      printf("Your ToDo List is empty.\n");
   }

   else
   {
      serial = 1;


printf("******************************************************************************************
*****************************************************\n");
      printf("*   Serial No |   Task Name                   |  %% Complete  |
Completed  |   Priority   |   Due Date  |  Due Time  |    Category        *\n");

printf("******************************************************************************************
*****************************************************\n");

      struct Task task;
      char line[1000];

      while(fgets(line, sizeof(line), file) != NULL)
      {
         if(sscanf(line, "Task Name: %19s", task.taskname) == 1)
         {
            // Successfully read the task name, now process the rest of the
data
            fgets(line, sizeof(line), file); // Read the description line
            sscanf(line, "Description: %499[^\n]", task.description); // Read
the description

            fgets(line, sizeof(line), file); // Read the next line
            sscanf(line, "Completion Percentage: %d",
&task.percent_complete);

            fgets(line, sizeof(line), file);
            sscanf(line, "Completed: %d", &task.completed);

            fgets(line, sizeof(line), file);
            sscanf(line, "Priority: %d", &task.priority);

            fgets(line, sizeof(line), file);
            sscanf(line, "Last Date: %d", &task.lastDate);

            fgets(line, sizeof(line), file);
            sscanf(line, "Last Time: %d", &task.time);

            fgets(line, sizeof(line), file);
            sscanf(line, "Category: %19s", task.category);

            printf("* %10d  | %-28s | %11d%%  | %-11s  | %11d  | %10d  | %9d  |
%-20s *\n",
                 serial, task.taskname, task.percent_complete,
task.completed ? "Yes" : "No",
                 task.priority, task.lastDate, task.time, task.category);

printf("*********************************************************************************************************
***************************************************************\n");

            serial++; // Increment the serial number
         }
      }

      fclose(file);
      return serial;  // Return the total number of tasks displayed

   }
}
```

```
// Function to add a new task in todo list
void addTask(struct Credentials user)
{
   system("cls");
   printf("************************************************************************\n");
   printf("*                             Add New Task                             *\n");
   printf("************************************************************************\n");

   char fileName[50];
   sprintf(fileName, "Program Data/%ld.txt", user.loginID); // Generate the
file's name based on the user's loginID

   FILE *file = fopen(fileName, "a"); // Open file in append mode
   if(file==NULL)
   {
      printf("Error opening file.\n");
      return;
   }
 // Create a new Task structure to store task details
   struct Task newTask;
   // Prompt the user to enter task details
   printf("Enter task name (up to 19 characters): ");
   scanf("%19s", newTask.taskname);

   printf("Enter task description (up to 499 characters): ");
   scanf(" %[^\n]", newTask.description);

   printf("Enter task completion percentage: ");
   scanf("%d", &newTask.percent_complete);

   // Determine task completion status based on percentage
   if(newTask.percent_complete<100)
   {
      newTask.completed = false; // Initialize to false
   }
   else
   {
      newTask.completed = true;
   }
   printf("Enter task priority (1-5): ");
```

```
    scanf("%d", &newTask.priority);

    printf("Enter last date to complete the task: ");
    scanf("%d", &newTask.lastDate);

    printf("Enter last time to complete the task: ");
    scanf("%d", &newTask.time);

    printf("Enter task category (up to 19 characters): ");
    scanf("%19s", newTask.category);

     // Write task details to the file
    fprintf(file, "Task Name: %s\n", newTask.taskname);
    fprintf(file, "Description: %s\n", newTask.description);
    fprintf(file, "Completion Percentage: %d\n",
newTask.percent_complete);
    fprintf(file, "Completed: %d\n", newTask.completed);
    fprintf(file, "Priority: %d\n", newTask.priority);
    fprintf(file, "Last Date: %d\n", newTask.lastDate);
    fprintf(file, "Last Time: %d\n", newTask.time);
    fprintf(file, "Category: %s\n", newTask.category);
    fprintf(file, "\n");
    fclose(file);
    //prints success message
    printf("Task added successfully.\n");
    sleep(1);
}

// Function to update the To-Do List for a given user
void updateToDoList(struct Credentials user)
{
    system("cls");
    printf("******************************************************************\n");
    printf("*                         Update To-Do List                      *\n");
    printf("******************************************************************\n");

    char filename[20];
    sprintf(filename, "Program Data/%ld.txt", user.loginID);
   // Open the file in read mode
    FILE *file = fopen(filename, "r");
```

```
if(file==NULL)
{
    printf("Error opening the file.\n");
    return;
}
sleep(1);
// Display the current To-Do List and get the total number of tasks
int taskNumber = seeToDoList(user);
int taskChoice;

  // Prompt the user to enter the task number to update
printf("Enter the task number you want to update (0 to exit): ");
scanf("%d", &taskChoice);

// Checks if user enters a valid tasknumber
if(taskChoice<=0||taskChoice>taskNumber)
{
    printf("No task selected or invalid task number.\n");
    fclose(file);
    return;
}

char line[1000];
int serial = 1;

 // Open a temporary file for writing
FILE *tempFile = fopen("Program Data/temp.txt", "w");
if(tempFile==NULL)
{
    printf("Error creating the temporary file.\n");
    fclose(file);
    return;
}
// Iterate through each line in the original file
while(fgets(line, sizeof(line), file) != NULL)
{
    int percent;
    bool completed;
     // Check if the line contains "Completion Percentage"
    if(strstr(line, "Completion Percentage")!=NULL)
```

```
        {
            int percent;
            // If the current line corresponds to the selected task
            if(serial==taskChoice)
            {
                printf("Enter updated completion percentage: ");
                scanf("%d", &percent);

                // Write the updated completion percentage to the temporary
file
                fprintf(tempFile, "Completion Percentage: %d\n",percent);
                printf("To-Do List updated successfully.\n");

                // Determine the completion status based on the updated
percentage
                if(percent>=100)
                {
                    completed=true;
                    fgets(line, sizeof(line), file);
                    fprintf(tempFile, "Completed: %d\n", completed);
                }
                else
                {
                    completed=false;
                    fgets(line, sizeof(line), file);
                    fprintf(tempFile, "Completed: %d\n", completed);
                }

            }
            else
            {   // If the line does not correspond to the selected task, copy it
to the temporary file
                fprintf(tempFile, "%s", line);
            }
            serial++;
        }
        else
        {          // If the line does not contain "Completion Percentage," copy
it to the temporary file
                    fprintf(tempFile, "%s", line);
        }
    }

    // Close the files
    fclose(file);
    fclose(tempFile);

    //Reopen the original file for writing
    file = fopen(filename, "w");
    if(file==NULL)
    {
        printf("Error opening the file.\n");
        return;
    }
      // Reopen the temporary file for reading
    tempFile = fopen("Program Data/temp.txt", "r");

    if(tempFile==NULL)
    {
        printf("Error creating the temporary file.\n");
        fclose(file);
        return;
    }
 // Copy the contents of the temporary file to the original file
    while(fgets(line, sizeof(line), tempFile) != NULL)
    {
        fprintf(file, "%s", line);
    }

    //closing the file
    fclose(file);
    fclose(tempFile);
    sleep(2);
}
// Function to delete a task from the to-do list
void deleteToDoList(struct Credentials user)
{
// Clear the console screen for a cleaner UI
    system("cls");
 // Display a header for the delete to-do list section
```

```
    printf("**********************************************************************\n");
    printf("*                          Delete To-Do List                          *\n");
    printf("**********************************************************************\n");

// Define variables for file and line handling
    char filename[20];
    char line[1000];
// Construct the filename based on the user's loginID
    sprintf(filename, "Program Data/%ld.txt", user.loginID);
// Introduce a brief delay for user experience
    sleep(1);

// Get the total number of tasks and display them for the user
    int taskNumber=seeToDoList(user);
    int taskChoice, serial = 0;
    char task[100];

// Open the original file in read mode
    FILE *file = fopen(filename, "r");
    if(file == NULL)
    {
        // Print an error message if the file cannot be opened
        printf("Error opening file!\n");
        return ;
    }

    // Open a temporary file in write mode
    FILE *tempFile = fopen("Program Data/temp.txt", "w");
    if(tempFile == NULL)
    {
                // Print an error message if the temp file cannot be opened
        printf("Error opening file!\n");
                // Close the original file before returning
        fclose(file);
        return ;
    }
// Prompt the user to enter the task number to delete
    printf("Enter task number to delete: ");
    // Validate user input for taskChoice
    scanf("%d", &taskChoice);
    if(taskChoice<=0||taskChoice>taskNumber)
    {
// Print an error message for invalid taskChoice
        printf("No task selected or invalid task number.\n");
        fclose(file);
        return;
    }

// Iterate through lines in the original file
    while(fgets(line, sizeof(line), file) != NULL)
    {
        // Check if the line contains task information
        if(strstr(line, "Task Name:")!=NULL)
        {
            serial++;
// If the current task matches the chosen task for deletion, skip its lines
            if(serial==taskChoice)
            {
                fgets(line, sizeof(line), file);
                fgets(line, sizeof(line), file);
                fgets(line, sizeof(line), file);
                fgets(line, sizeof(line), file);
                fgets(line, sizeof(line), file);
                fgets(line, sizeof(line), file);
                fgets(line, sizeof(line), file);
                fgets(line, sizeof(line), file);
            }
            else
         // If the task doesn't match, write it to the temp file
        fprintf(tempFile, "%s", line);
        }
        else
            // If the line doesn't contain task information, write it to the temp
file
          fprintf(tempFile, "%s", line);
    }

    // Close both the original and temp files
    fclose(file);
    fclose(tempFile);
    printf("Task deleted successfully.\n");
    sleep(2);
}

// Function to edit tasks in the to-do list
void editTasks(struct Credentials user)
{
     // Clear the console screen for a cleaner UI
    system("cls");
        // Display a header for the edit tasks section
    printf("********************************************************************************\n");
    printf("*                              Edit Tasks                                      *\n");
    printf("********************************************************************************\n");

    // Define variables for file and line handling
    char filename[20];
    char line[1000];
        // Construct the filename based on the user's loginID
    sprintf(filename, "Program Data/%ld.txt", user.loginID);
        // Introduce a brief delay for user experience
    sleep(1);

    // Open the original file in read mode
    FILE *file = fopen(filename, "r");
    if(file == NULL)
    {
    // Print an error message if the file cannot be opened
        printf("Error opening the original file.\n");
        return;
    }

    // Get the total number of tasks and display them for the user
    int taskNumber=seeToDoList(user);
    int taskChoice,currentTask= 0;

    // Open an editable file in write mode
    FILE *edit = fopen("Program Data/edit.txt", "w");
    if(edit == NULL)
    {
```

```
 // Print an error message if the editable file cannot be created
   printf("Error creating the editable file.\n");
 // Close the original file before returning
   fclose(file);
   return;
}

// Prompt the user to enter the task number to edit
printf("Enter the task number you want to edit (0 to exit): ");
scanf("%d", &taskChoice);

// Validate user input for taskChoice
if(taskChoice<=0||taskChoice>taskNumber)
{
// Print an error message for invalid taskChoice
   printf("INVALID INPUT!! Exiting without changes.\n");
// Close the editable file before returning
   fclose(edit);
   return;
}

// Display options for editing a task
printf("Enter what you want to edit in the selected task: \n");
printf("1.) Task name\n");
printf("2.) Task description\n");
printf("3.) Task priority\n");
printf("4.) Task last date\n");
printf("5.) Task last time\n");
printf("6.) Task category\n");
printf("0.) Exit\n");
printf("Enter your choice : ");
int z;
scanf("%d", &z);
   // Iterate through lines in the original file
while(fgets(line, sizeof(line), file) != NULL)
{
  // Check if the line contains task information
   if(strstr(line, "Task Name:")!=NULL)
   {
      currentTask++;
```

```
// If the current task matches the chosen task for editing
        if(currentTask == taskChoice)
        {
            switch(z)
            {
// Handle user choices for editing
                case 0:
// Exit editing
                    escape();
                    break;
// Add cases for other editing options as needed
                case 1:
                {
                    // Edit task name
                    char newTaskName[20];
                    printf("Enter the new task name (up to 19 characters): ");
                    scanf("%19s", newTaskName);
                    fprintf(edit, "Task Name: %s\n", newTaskName);
                    break;
                }
                case 2:
                {// Edit task description
                    fprintf(edit, "%s", line);
                    fgets(line, sizeof(line),file);
                    char newDescription[500];
                    printf("Enter the new task description (up to 499
characters): ");
                    scanf(" %[^\n]", newDescription);
                    fprintf(edit, "Description: %s\n", newDescription);
                    break;
                }
                case 3:
                {// Edit task priority
                    fprintf(edit, "%s", line);
                    fgets(line, sizeof(line),file);
                    fprintf(edit, "%s", line);
                    fgets(line, sizeof(line),file);
                    fprintf(edit, "%s", line);
                    fgets(line, sizeof(line),file);
                    fprintf(edit, "%s", line);
```

```
    fgets(line, sizeof(line),file);
    int newPriority;
    printf("Enter the new task priority (1-5): ");
    scanf("%d", &newPriority);
    fprintf(edit, "Priority: %d\n", newPriority);
    break;
}
case 4:
{// Edit task last date
    fprintf(edit, "%s", line);
    fgets(line, sizeof(line),file);
    fprintf(edit, "%s", line);
    fgets(line, sizeof(line),file);
    fprintf(edit, "%s", line);
    fgets(line, sizeof(line),file);
    fprintf(edit, "%s", line);
    fgets(line, sizeof(line),file);
    fprintf(edit, "%s", line);
    fgets(line, sizeof(line),file);

    int newLastDate;
    printf("Enter the new last date for the task: ");
    scanf("%d", &newLastDate);
    fprintf(edit, "Last Date: %d\n", newLastDate);
    break;
}
case 5:
{// Edit task last time
    fprintf(edit, "%s", line);
    fgets(line, sizeof(line),file);
    fprintf(edit, "%s", line);
    fgets(line, sizeof(line),file);
    fprintf(edit, "%s", line);
    fgets(line, sizeof(line),file);
    fprintf(edit, "%s", line);
    fgets(line, sizeof(line),file);
    fprintf(edit, "%s", line);
    fgets(line, sizeof(line),file);
    fprintf(edit, "%s", line);
    fgets(line, sizeof(line),file);
```

```
            int newLastTime;
            printf("Enter the new last time for the task: ");
            scanf("%d", &newLastTime);
            fprintf(edit, "Last Time: %d\n", newLastTime);
            break;
        }
        case 6:
        {       // Edit task category
            fprintf(edit, "%s", line);
            fgets(line, sizeof(line),file);
            fprintf(edit, "%s", line);
            fgets(line, sizeof(line),file);
            fprintf(edit, "%s", line);
            fgets(line, sizeof(line),file);
            fprintf(edit, "%s", line);
            fgets(line, sizeof(line),file);
            fprintf(edit, "%s", line);
            fgets(line, sizeof(line),file);
            fprintf(edit, "%s", line);
            fgets(line, sizeof(line),file);
            fprintf(edit, "%s", line);
            fgets(line, sizeof(line),file);

            char newCategory[20];
            printf("Enter the new task category (up to 19 characters): ");
            scanf("%19s", newCategory);
            fprintf(edit, "Category: %s\n", newCategory);
            break;
        }
        default:
        {   // Handle invalid option
            printf("Invalid option.\n");
        // Introduce a brief delay for user experience
            sleep(2);
        // Return to the task manager
            taskManager(user,mainMenu());
    }
    }
}
```

```
            else
            {    // Continue writing lines to the edited file
                fprintf(edit, "%s", line);
            }
        }
        // Continue writing lines to the edited file
        else
        {
            fprintf(edit, "%s", line);
        }
    }

// Close both the original and edited files after completing the editing
process
    fclose(file);
    fclose(edit);

// Reopen the original file in write mode to overwrite its contents
    file = fopen(filename, "w");
    if(file == NULL)
    {
        printf("Error opening the file.\n");
        return;
    }

// Reopen the edited file in read mode
    edit = fopen("Program Data/edit.txt", "r");

    if(edit == NULL)
    {
        printf("Error creating the temporary file.\n");
        fclose(file);
        return;
    }

// Copy the edited content back to the original file
    while(fgets(line, sizeof(line), edit) != NULL)
    {
        fprintf(file, "%s", line);
    }
```

```
// Close both the original and edited files after completing the copying
process
    fclose(file);
    fclose(edit);
    printf("Task editing complete.\n");
}

// Function to filter tasks in the to-do list
void filterToDoList(struct Credentials user)
{
        // Clear the console screen for a cleaner UI
    system("cls");
        // Set console text color to green on black
    system("color 0B");

    // Display a header for the filter to-do list section
    printf("**********************************************************************\n");
    printf("*                          Filter To-Do List                         *\n");
    printf("**********************************************************************\n");

    // Prompt the user to choose a filter option
    Retry:
    printf("Filter Options:\n");
    printf("1. Show Only Completed Tasks\n");
    printf("2. Show Only Incomplete Tasks\n");
    printf("3. Filter by Priority\n");
    printf("4. Filter by Last Date\n");
    printf("5. Filter by Category\n");
    printf("0. Return to MAIN MENU\n");

    // Read the user's choice
    printf("Enter your choice: ");
    int filterChoice;
    scanf("%d", &filterChoice);

    // Define the filename based on the user's loginID
    char filename[20];
    sprintf(filename, "Program Data/%ld.txt", user.loginID);
        // Open the original file in read mode
    FILE *file = fopen(filename, "r");

    // Check if the file can be opened
    if(file == NULL)
    {
        printf("Error opening file!\n");
        return;
    }

    // Initialize variables for task information and line reading
    int serial = 1;
    char line[1000];

    // Switch statement to handle different filter options
    switch (filterChoice)
    {
    case 0:
    {   // Case 0: Return to the MAIN MENU
        fclose(file);
        system("color 07");
        taskManager(user,mainMenu());
    }
    case 1:
    {   // Case 1: Show Only Completed Tasks
        system("color 0A");

printf("****************************************************************************************************
************************************************************\n");
        printf("*   Serial No |   Task Name                    |   %% Complete  |
Completed  |   Priority   |   Due Date  |  Due Time  |    Category         *\n");

printf("****************************************************************************************************
************************************************************\n");

        struct Task task;

        while(fgets(line, sizeof(line), file) != NULL)
        {
            if(sscanf(line, "Task Name: %19s", task.taskname) == 1)
            {
                // Successfully read the task name, now process the rest of the
data
                fgets(line, sizeof(line), file); // Read the description line
                sscanf(line, "Description: %499[^\n]", task.description); // Read
the description

                fgets(line, sizeof(line), file); // Read the next line
                sscanf(line, "Completion Percentage: %d",
&task.percent_complete);

                fgets(line, sizeof(line), file);
                sscanf(line, "Completed: %d", &task.completed);

                fgets(line, sizeof(line), file);
                sscanf(line, "Priority: %d", &task.priority);

                fgets(line, sizeof(line), file);
                sscanf(line, "Last Date: %d", &task.lastDate);

                fgets(line, sizeof(line), file);
                sscanf(line, "Last Time: %d", &task.time);

                fgets(line, sizeof(line), file);
                sscanf(line, "Category: %19s", task.category);

                fgets(line, sizeof(line), file);
                if(task.completed)
                {
                    printf("* %10d  | %-28s | %11d%%  | %-11s  | %11d  | %10d  | %9d  |
%-20s *\n",serial, task.taskname, task.percent_complete, task.completed
? "Yes" : "No", task.priority, task.lastDate, task.time, task.category);

printf("*******************************************************************************************************
*************************************************************\n");
                    serial++; // Increment the serial number
                }
            }
        }
        break;
```

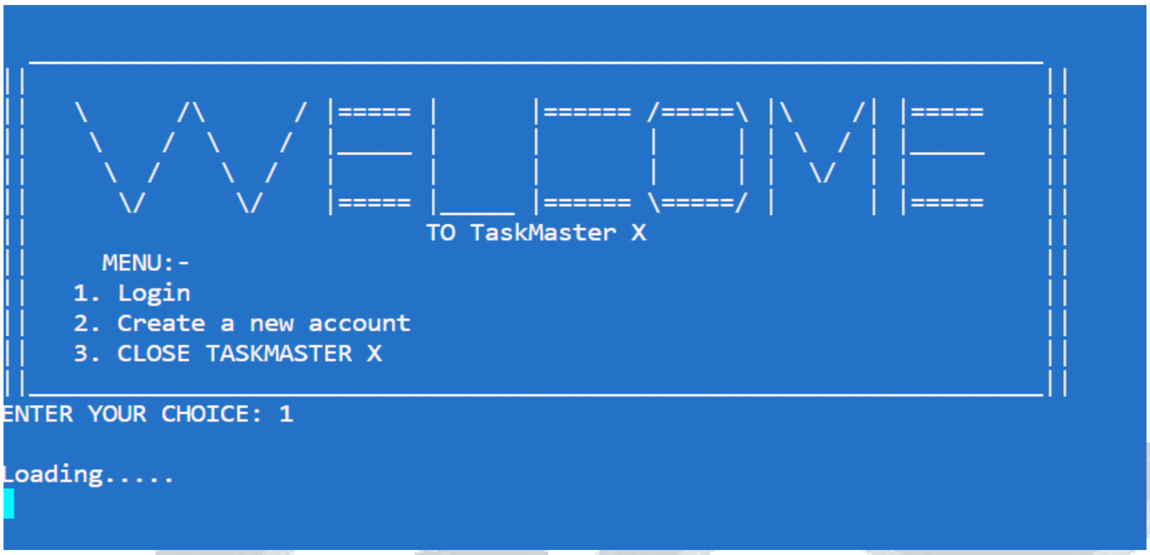
```
    }

    case 2:
    {
        // Show Only Incomplete Tasks
        system("color 0C");  // Change text to light red and background to
black


printf("*****************************************************************************************
*****************************************************\n");
        printf("*   Serial No |   Task Name                 |   %% Complete  |
Completed  |   Priority   |   Due Date  |  Due Time  |    Category        *\n");

printf("*****************************************************************************************
*****************************************************\n");

        struct Task task;

        while(fgets(line, sizeof(line), file) != NULL)
        {
            if(sscanf(line, "Task Name: %19s", task.taskname) == 1)
            {
                // Successfully read the task name, now process the rest of the
data
                fgets(line, sizeof(line), file); // Read the description line
                sscanf(line, "Description: %499[^\n]", task.description); // Read
the description

                fgets(line, sizeof(line), file); // Read the next line
                sscanf(line, "Completion Percentage: %d",
&task.percent_complete);

                fgets(line, sizeof(line), file);
                sscanf(line, "Completed: %d", &task.completed);

                fgets(line, sizeof(line), file);
                sscanf(line, "Priority: %d", &task.priority);

                fgets(line, sizeof(line), file);
                sscanf(line, "Last Date: %d", &task.lastDate);

                fgets(line, sizeof(line), file);
                sscanf(line, "Last Time: %d", &task.time);

                fgets(line, sizeof(line), file);
                sscanf(line, "Category: %19s", task.category);

                fgets(line, sizeof(line), file);

                if(!task.completed)
                {
                    printf("* %10d  | %-28s | %11d%%  | %-11s  | %11d  | %10d  | %9d  |
%-20s *\n",serial, task.taskname, task.percent_complete, task.completed
? "Yes" : "No",task.priority, task.lastDate, task.time, task.category);

printf("*******************************************************************************************
*****************************************************\n");
                    serial++; // Increment the serial number
                }
            }
        }

        break;
    }

    case 3:
    {
        // Filter by Priority
        int priority;
        printf("Enter priority to filter tasks: ");
        scanf("%d", &priority);

        system("color 0E");  // Change text to yellow and background to
black


printf("*******************************************************************************************
*****************************************************\n");
```

```
    printf("*   Serial No |   Task Name                 |   %% Complete   |
Completed  |   Priority   |   Due Date  |  Due Time  |    Category         *\n");

printf("******************************************************************************************
*****************************************************\n");

    struct Task task;

    while(fgets(line, sizeof(line), file) != NULL)
    {
       if(sscanf(line, "Task Name: %19s", task.taskname) == 1)
       {
          // Successfully read the task name, now process the rest of the
data
          fgets(line, sizeof(line), file); // Read the description line
          sscanf(line, "Description: %499[^\n]", task.description); // Read
the description

          fgets(line, sizeof(line), file); // Read the next line
          sscanf(line, "Completion Percentage: %d",
&task.percent_complete);

          fgets(line, sizeof(line), file);
          sscanf(line, "Completed: %d", &task.completed);

          fgets(line, sizeof(line), file);
          sscanf(line, "Priority: %d", &task.priority);

          fgets(line, sizeof(line), file);
          sscanf(line, "Last Date: %d", &task.lastDate);

          fgets(line, sizeof(line), file);
          sscanf(line, "Last Time: %d", &task.time);

          fgets(line, sizeof(line), file);
          sscanf(line, "Category: %19s", task.category);

          fgets(line, sizeof(line), file);

          if(task.priority == priority)
```

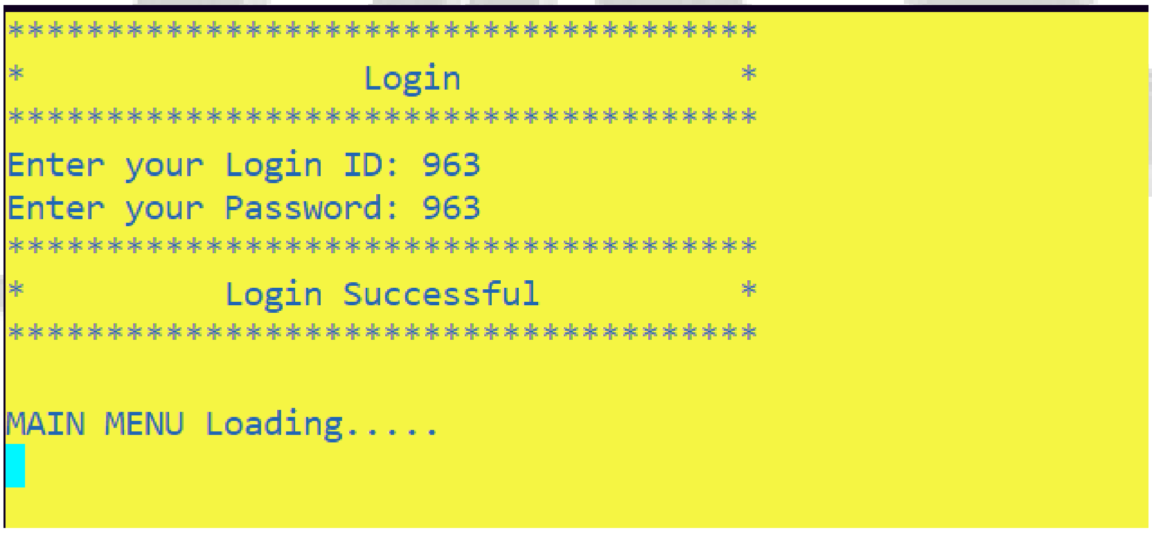
```
            {
                printf("* %10d  | %-28s | %11d%%  | %-11s  | %11d  | %10d  | %9d  |
%-20s *\n",serial, task.taskname, task.percent_complete, task.completed
? "Yes" : "No",task.priority, task.lastDate, task.time, task.category);

printf("*****************************************************************************************************************************
***************************************************************************\n");
                serial++; // Increment the serial number
            }
        }
    }

    break;
  }
  case 4:
  {
  // Filter by Last Date
    int lastDate;
    printf("Enter last date to filter tasks: ");
    scanf("%d", &lastDate);

    system("color 0D");  // Change text to light purple and background
to black

printf("*****************************************************************************************************************************
***************************************************************************\n");
    printf("*   Serial No |   Task Name                  |   %% Complete  |
Completed  |   Priority   |  Due Date  |  Due Time  |    Category        *\n");

printf("*****************************************************************************************************************************
***************************************************************************\n");

    struct Task task;

    while(fgets(line, sizeof(line), file) != NULL)
    {
      if(sscanf(line, "Task Name: %19s", task.taskname) == 1)
      {
            // Successfully read the task name, now process the rest of the
data
            fgets(line, sizeof(line), file); // Read the description line
            sscanf(line, "Description: %499[^\n]", task.description); // Read
the description

            fgets(line, sizeof(line), file); // Read the next line
            sscanf(line, "Completion Percentage: %d",
&task.percent_complete);

            fgets(line, sizeof(line), file);
            sscanf(line, "Completed: %d", &task.completed);

            fgets(line, sizeof(line), file);
            sscanf(line, "Priority: %d", &task.priority);

            fgets(line, sizeof(line), file);
            sscanf(line, "Last Date: %d", &task.lastDate);

            fgets(line, sizeof(line), file);
            sscanf(line, "Last Time: %d", &task.time);

            fgets(line, sizeof(line), file);
            sscanf(line, "Category: %19s", task.category);

            fgets(line, sizeof(line), file);

            if(task.lastDate == lastDate)
            {
               printf("* %10d  | %-28s | %11d%%  | %-11s  | %11d  | %10d  | %9d  |
%-20s *\n",serial, task.taskname, task.percent_complete, task.completed
? "Yes" : "No",task.priority, task.lastDate, task.time, task.category);

printf("*********************************************************************************************
*******************************************************\n");
               serial++; // Increment the serial number
            }
         }
      }
```

```
        break;
    }

    case 5:
    {
        // Filter by Category
        char category[20];
        printf("Enter category to filter tasks: ");
        scanf("%19s", category);

        system("color 0F");  // Change text to bright white and background
to black

printf("***************************************************************************************************************
******************************************************************\n");
        printf("*   Serial No |   Task Name                    |   %% Complete  |
Completed  |   Priority   |   Due Date  |  Due Time  |    Category          *\n");

printf("***************************************************************************************************************
******************************************************************\n");

        struct Task task;

        while(fgets(line, sizeof(line), file) != NULL)
        {
            if(sscanf(line, "Task Name: %19s", task.taskname) == 1)
            {
                // Successfully read the task name, now process the rest of the
data
                fgets(line, sizeof(line), file); // Read the description line
                sscanf(line, "Description: %499[^\n]", task.description); // Read
the description

                fgets(line, sizeof(line), file); // Read the next line
                sscanf(line, "Completion Percentage: %d",
&task.percent_complete);

                fgets(line, sizeof(line), file);
                sscanf(line, "Completed: %d", &task.completed);
                fgets(line, sizeof(line), file);
                sscanf(line, "Priority: %d", &task.priority);

                fgets(line, sizeof(line), file);
                sscanf(line, "Last Date: %d", &task.lastDate);

                fgets(line, sizeof(line), file);
                sscanf(line, "Last Time: %d", &task.time);

                fgets(line, sizeof(line), file);
                sscanf(line, "Category: %19s", task.category);
                fgets(line, sizeof(line), file);

                if(strcmp(task.category, category) == 0)
                {
                  printf("* %10d  | %-28s | %11d%%  | %-11s  | %11d  | %10d  | %9d  |
%-20s *\n",serial, task.taskname, task.percent_complete, task.completed
? "Yes" : "No",task.priority, task.lastDate, task.time, task.category);
printf("********************************************************************************************
*******************************************************\n");
                  serial++; // Increment the serial number
                }
            }
        }
        break;
    }
  default:
        printf("Invalid choice. Please enter a number between 0 and 5.\n");
        goto Retry;
    }
    fclose(file);
    sleep(2);
```

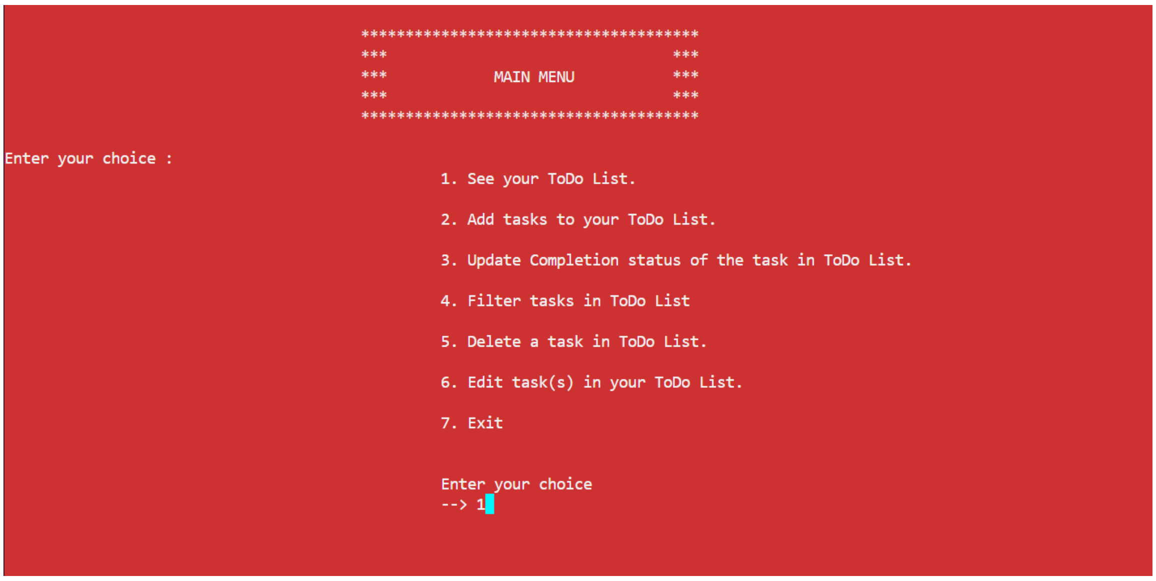
Output:



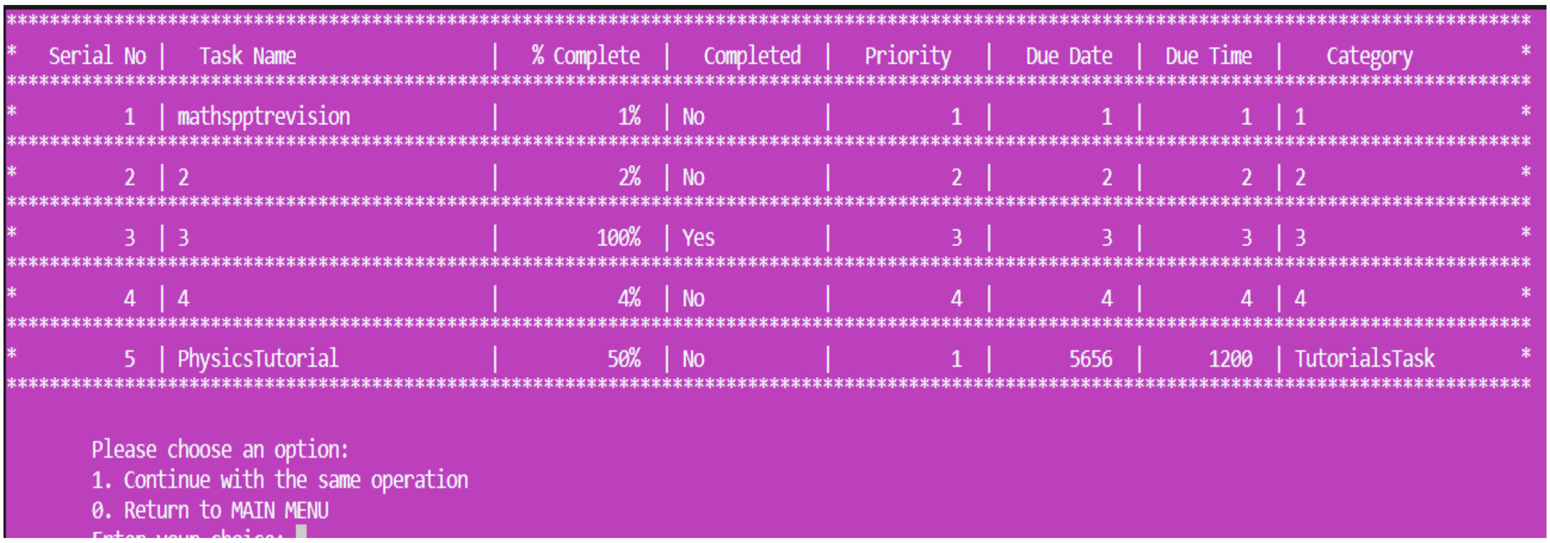
```
  ______________________________________________________________________
||                                                                      ||
||   \      /\      / |===== |      |====== /=====\ |\     /| |=====    ||
||    \    /  \    /  |_____ |      |       |     | | \   / | |_____    ||
||     \  /    \  /   |      |      |       |     | |  \ /  | |         ||
||      \/      \/    |===== |_____ |====== \=====/ |   V   | |=====    ||
||                           TO TaskMaster X                            ||
||      MENU:-                                                          ||
||    1. Login                                                          ||
||    2. Create a new account                                           ||
||    3. CLOSE TASKMASTER X                                             ||
||______________________________________________________________________||
ENTER YOUR CHOICE: 1

Loading.....
```



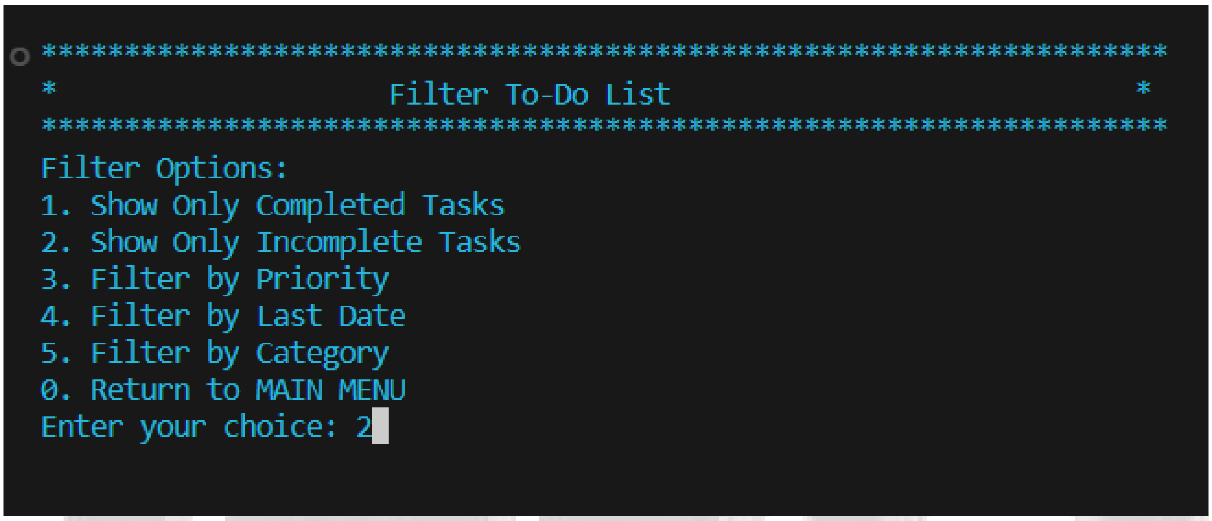
```
*************************************
*                Login              *
*************************************
Enter your Login ID: 963
Enter your Password: 963
*************************************
*          Login Successful         *
*************************************


MAIN MENU Loading.....
```



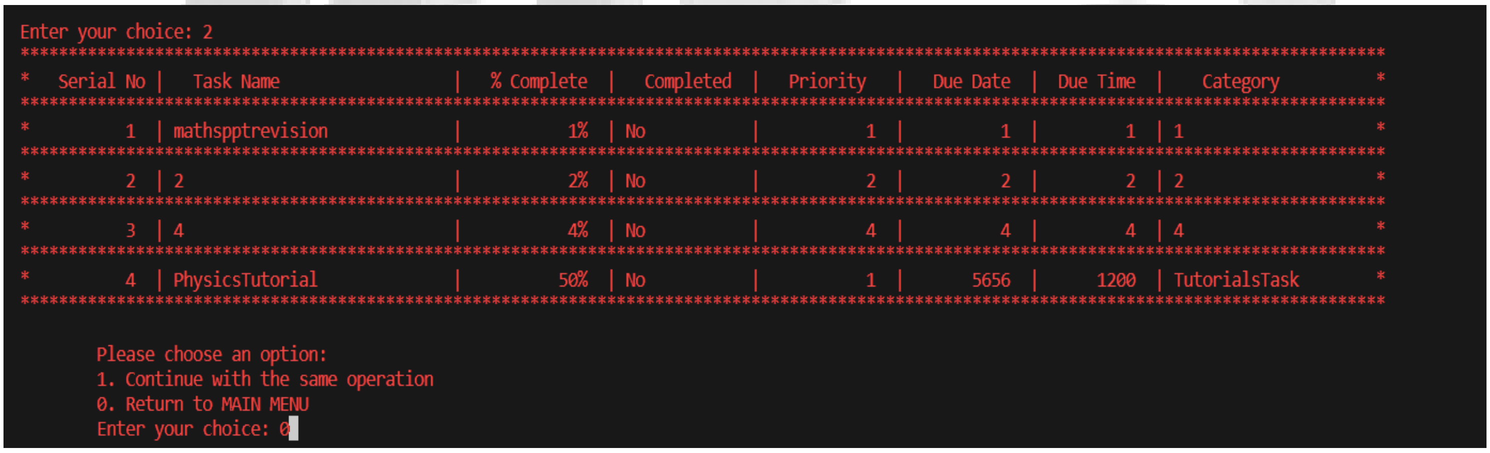
```
                                  ************************************
                                  ***                              ***
                                  ***          MAIN MENU           ***
                                  ***                              ***
                                  ************************************

Enter your choice :
                                          1. See your ToDo List.

                                          2. Add tasks to your ToDo List.

                                          3. Update Completion status of the task in ToDo List.

                                          4. Filter tasks in ToDo List

                                          5. Delete a task in ToDo List.

                                          6. Edit task(s) in your ToDo List.

                                          7. Exit


                                          Enter your choice
                                          --> 1
```



```
*************************************************************************************************************
*  Serial No |  Task Name                  |  % Complete  |  Completed  |  Priority  |  Due Date  |  Due Time  |   Category        *
*************************************************************************************************************
*          1 | mathspptrevision            |           1% | No          |          1 |          1 |          1 | 1                 *
*************************************************************************************************************
*          2 | 2                           |           2% | No          |          2 |          2 |          2 | 2                 *
*************************************************************************************************************
*          3 | 3                           |         100% | Yes         |          3 |          3 |          3 | 3                 *
*************************************************************************************************************
*          4 | 4                           |           4% | No          |          4 |          4 |          4 | 4                 *
*************************************************************************************************************
*          5 | PhysicsTutorial             |          50% | No          |          1 |       5656 |       1200 | TutorialsTask     *
*************************************************************************************************************


      Please choose an option:
      1. Continue with the same operation
      0. Return to MAIN MENU
      Enter your choice:
```



```
*********************************************************************
*                        Filter To-Do List                          *
*********************************************************************
Filter Options:
1. Show Only Completed Tasks
2. Show Only Incomplete Tasks
3. Filter by Priority
4. Filter by Last Date
5. Filter by Category
0. Return to MAIN MENU
Enter your choice: 2
```



```
Enter your choice: 2
*************************************************************************************************************
*  Serial No |  Task Name                  |  % Complete  |  Completed  |  Priority  |  Due Date  |  Due Time  |   Category        *
*************************************************************************************************************
*          1 | mathspptrevision            |           1% | No          |          1 |          1 |          1 | 1                 *
*************************************************************************************************************
*          2 | 2                           |           2% | No          |          2 |          2 |          2 | 2                 *
*************************************************************************************************************
*          3 | 4                           |           4% | No          |          4 |          4 |          4 | 4                 *
*************************************************************************************************************
*          4 | PhysicsTutorial             |          50% | No          |          1 |       5656 |       1200 | TutorialsTask     *
*************************************************************************************************************


      Please choose an option:
      1. Continue with the same operation
      0. Return to MAIN MENU
      Enter your choice: 0
```

References

1. Online Learning Platforms:

YouTube tutorials as resources that help to understand some important concepts used in creating to-do lists.

2. GitHub:

Explore some open-source projects on GitHub written in C. Looked for projects related to task management or to-do lists and similar working model for reference. We learn a lot by studying the codebase of existing projects.

3. Programming Forums:

Websites like Stack Overflow were really helpful. There, we asked specific questions related to your project and got various good advice from the community. It was also used to solve various bugs in the code.

4. C Programming Books:

Books like "The Complete Reference C" by Herbert Schildt" and “Let us C” by Yashwant Kanetkar provided foundational knowledge , syntax and examples that we applied in our project.

5. Online Tutorials:

Platforms like GeeksforGeeks, TutorialsPoint also provided the guidance for the project.