PROJECT REPORT

PROGRAMING C

University of Petroleum amd Energy Studies

School of computer science

INSTRUCTOR: DR. PRASHANT TRIVEDI

COURSE: PROGRAMMING IN C

PROJECT TITLE .

To-Do List Application (with Priority & Deadlines)

• Add, update, delete tasks.

• Assign priority levels (High/Medium/Low).

• Set deadlines with date and time.

• Display pending vs completed tasks.

• Save tasks persistently to a file.

SUBMITTED BY : RASHI BANYAL

ABSTARCT :

The TO-DO application is written in C programming language. It's a basic task management system that includes features…

- for setting deadline with date and time.
- Displaying tasks(with their status (pending or completed).
- Saving tasks to a file.
- It use structure for data representation.
- A menu-driven interface.

This report covers the project's objectives

Problem Definition , System Design (flowchart),

Implementation Details(with snippets) ,

Testing and Results , Conclusion and Future Work ,

References , Appendix…

PROBLEM DEFINITON

INTRODUCTION

➢ PURPOSE

The purpose of this project is to enhance a basic TO-DO LIST APPLICATION with the C programming concepts file

Handling, string validation, and conditional data display.

This application allow users to set deadlines along with the time , allow the users to edit any existing task (but deadline date and time line cannot edit it's fixed once u set) , allow the user to change priority and change the status of tasks…..

➢ OBJECTIVES

The major objectives of the application are:

To store and manage tasks using C structures and arrays.

To provide CRUD operation (Create, Read, Update, Delete).

To validate date, time, and input values.

To save tasks to a file and load them .

To implement user-friendly menu interface.

PROGRAM OVERVIEW

The To-Do list Application uses:

- Structure to represent individual tasks.
- Array of Task to hold all tasks in memory
- Utility Function to handle input validation.
- File operations for saving and loading tasks.
- Menu-driven loop for user interaction.

ALOGORITHM :

1 . Start

2 . Initialize System

1. Declare array tasks[MAX_TASKS] of Struc Task
2. Declare Variables:
   - Taskcount – 0
   - nextID – 0
   - Running – 1
3. Call initTasks() to reset all tasks.

4.   call LoadtasksFromFile() to load saved tasks from “tasks.txt”

3. Main Loop (Runs Until Exit)

Repeat while Running = 1;

Display Menu :

Call printMenu().

Read user choice

Use readIntInRange(1-7) to get an integer choice.

4. Process User Choice

Case 1:   Add New Task

Call addTask()

1 . If task list is full print error.

2 . Read:

- Task title
- Task description
- Deadline date (loop until valid)
- Deadline time (loop until valid)

- Priority(1-5)

3 . Assign:

- Id = nextId
- Isdone = 0

4 . Append task to array.

5 . Increment taskCount and nextId.

6 . Print Added successfully.

Case 2:   List All Tasks

Call listTasks()

1 . If no task (Show the printf statement)

2 . Else, print a formatted table of:

- Id
- Done status
- Title
- Priority
- Deadline date
- Deadline time

- Description

Case 3: Mark as Done

Call markTaskAsDone()

1 . If no Tasks found   (display printf statement).

2 . Show task list.

3 . Read task ID.

4 . Search Using findTaskIndexById()

5 . If found: set isdone = 1

Call saveTasksToFile()

6 . Else show “Task not found.”

Case 4: Edit an Existing Task

Call editTask()

1 . if taskCount = 0 display message

2 . Show task list

3 . Read Id

4 . Search for task

5 . Display edit menu:

- 1: Edit title

- 2: Edit description
- 3: Edit priorty
- 4: Cancle

6 . Update selected field.

Case 5: Delete a Task

Call deleteTask()

1 . if no task (print message)

2 . Show exixting tasks.

3 . Read id

4 . Search Using findTaskIdexByID()

5 . If Found:

- Shift all tasks after index to the left.
- Decrease taskCount.
- Save updated list to file.

6. Else print "not found"

Case 6: Save Task Manually

Call saveTasksToFile()

1 . Open "tasks.txt" in write mode.

2 . For each task, write:

- Id
- Title
- Description
- Date
- Time
- Priority
- Isdone

3 . Close file.

4 . Show “Tasks saved.”

Case 7: Exit

1 . Print exit message

2 . Set Running = 0

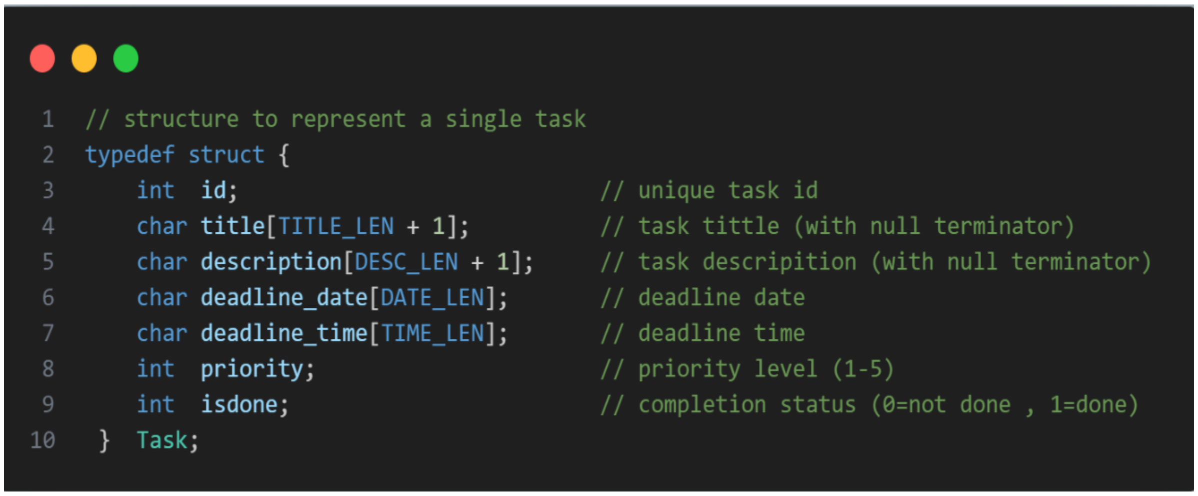
5. END PROGRAM

IMPLEMENTATION DETAILS:

(WITH SNIPPETS)

1 . DATA STRUCTURE

Each task is represented by Task struct that holds all relevent details about tasks such as title, description,deadline,priority, and complete status.Every option (add,edit,mark,done,delete) works on thid structure.

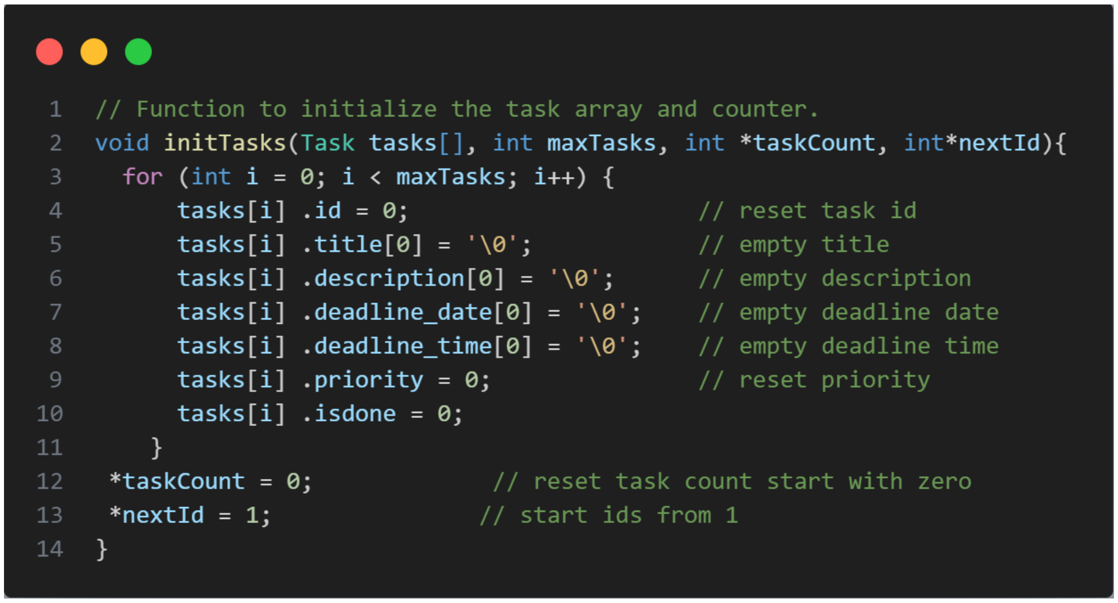


```c
// structure to represent a single task
typedef struct {
    int  id;                          // unique task id
    char title[TITLE_LEN + 1];        // task tittle (with null terminator)
    char description[DESC_LEN + 1];   // task descripition (with null terminator)
    char deadline_date[DATE_LEN];     // deadline date
    char deadline_time[TIME_LEN];     // deadline time
    int  priority;                    // priority level (1-5)
    int  isdone;                      // completion status (0=not done , 1=done)
 }  Task;
```

2 . INITIALIZATION

The program begins by resetting all stored tasks and counters.

nextID is set to 1 so each task recieves a unique ID.

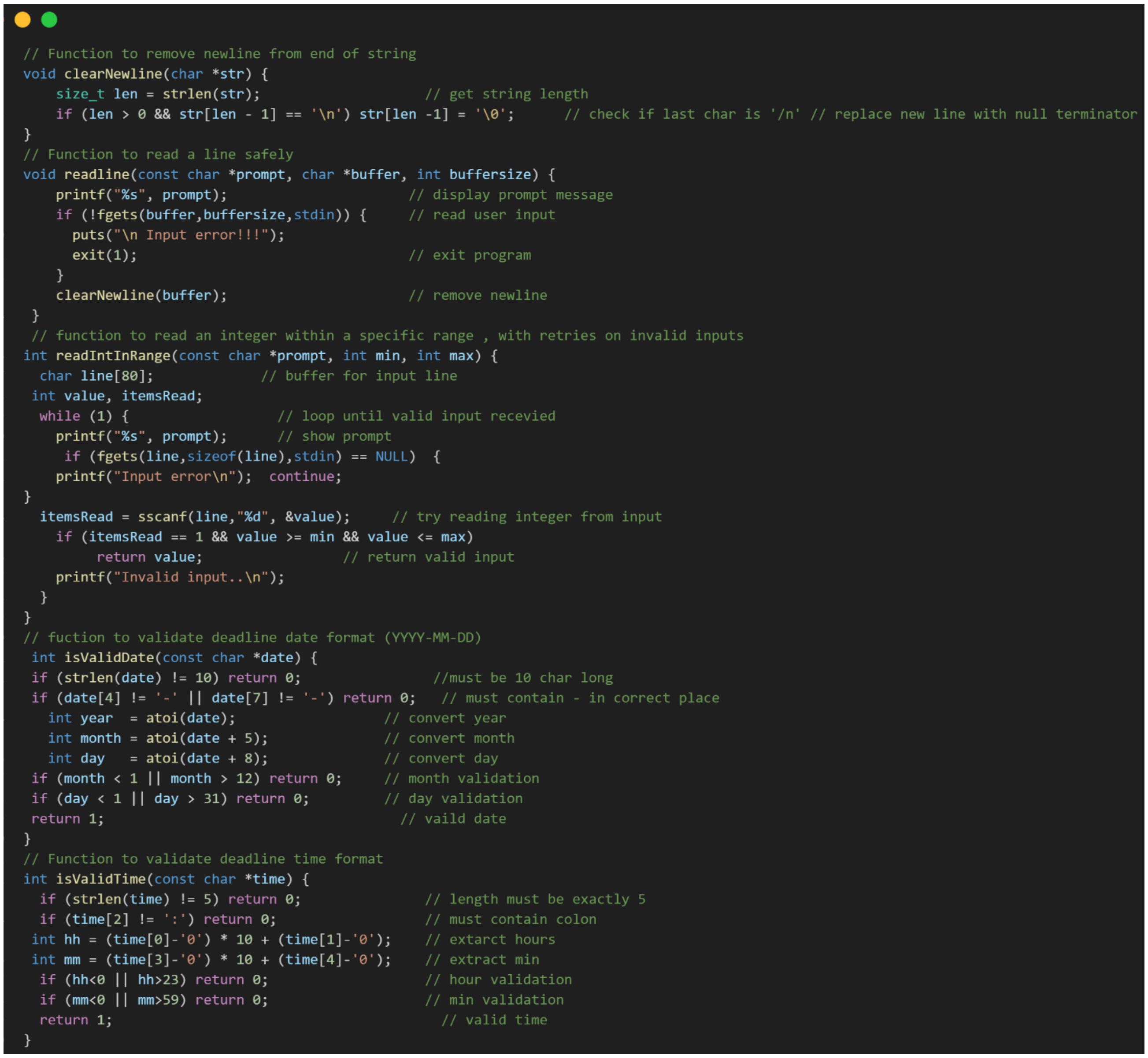


```c
// Function to initialize the task array and counter.
void initTasks(Task tasks[], int maxTasks, int *taskCount, int*nextId){
  for (int i = 0; i < maxTasks; i++) {
      tasks[i] .id = 0;                     // reset task id
      tasks[i] .title[0] = '\0';            // empty title
      tasks[i] .description[0] = '\0';      // empty description
      tasks[i] .deadline_date[0] = '\0';    // empty deadline date
      tasks[i] .deadline_time[0] = '\0';    // empty deadline time
      tasks[i] .priority = 0;               // reset priority
      tasks[i] .isdone = 0;
    }
 *taskCount = 0;             // reset task count start with zero
 *nextId = 1;               // start ids from 1
}
```

3 . INPUT HANDLING AND VALIDATION

The program includes helper functions to read safe input and validate formats. To ensure the program does not crash due to invalid inputs.

readline() reads a string safely and removes newline characters, while readInRange() ensure only valid integers are accepted. To avoid worng date/time entries function check proper format suchh sa YYYY-MM-DD and HH:MM



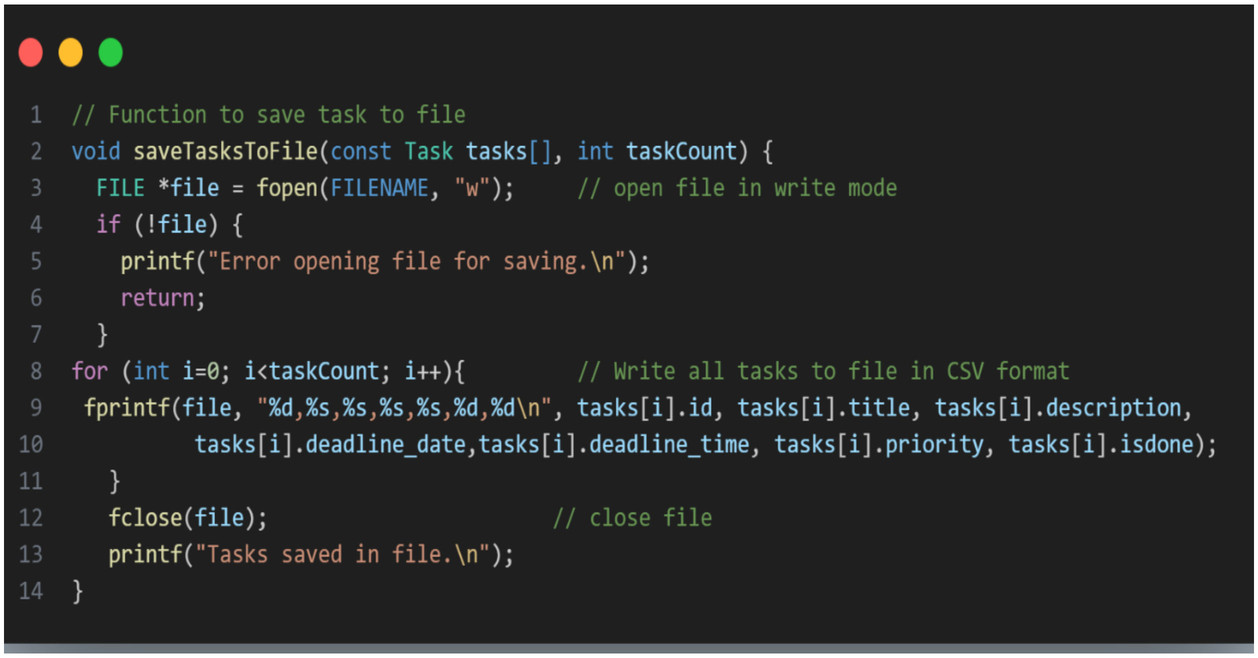
```c
// Function to remove newline from end of string
void clearNewline(char *str) {
    size_t len = strlen(str);                    // get string length
    if (len > 0 && str[len - 1] == '\n') str[len -1] = '\0';      // check if last char is '/n' // replace new line with null terminator
}
// Function to read a line safely
void readline(const char *prompt, char *buffer, int buffersize) {
    printf("%s", prompt);                      // display prompt message
    if (!fgets(buffer,buffersize,stdin)) {     // read user input
      puts("\n Input error!!!");
      exit(1);                                 // exit program
    }
    clearNewline(buffer);                      // remove newline
 }
 // function to read an integer within a specific range , with retries on invalid inputs
int readIntInRange(const char *prompt, int min, int max) {
  char line[80];             // buffer for input line
 int value, itemsRead;
  while (1) {                  // loop until valid input recevied
    printf("%s", prompt);      // show prompt
     if (fgets(line,sizeof(line),stdin) == NULL)  {
    printf("Input error\n");  continue;
}
  itemsRead = sscanf(line,"%d", &value);     // try reading integer from input
    if (itemsRead == 1 && value >= min && value <= max)
         return value;                 // return valid input
    printf("Invalid input..\n");
  }
}
// fuction to validate deadline date format (YYYY-MM-DD)
 int isValidDate(const char *date) {
 if (strlen(date) != 10) return 0;                //must be 10 char long
 if (date[4] != '-' || date[7] != '-') return 0;   // must contain - in correct place
   int year  = atoi(date);                      // convert year
   int month = atoi(date + 5);                  // convert month
   int day   = atoi(date + 8);                  // convert day
 if (month < 1 || month > 12) return 0;      // month validation
 if (day < 1 || day > 31) return 0;          // day validation
 return 1;                                     // vaild date
}
// Function to validate deadline time format
int isValidTime(const char *time) {
  if (strlen(time) != 5) return 0;               // length must be exactly 5
  if (time[2] != ':') return 0;                  // must contain colon
 int hh = (time[0]-'0') * 10 + (time[1]-'0');    // extarct hours
 int mm = (time[3]-'0') * 10 + (time[4]-'0');    // extract min
  if (hh<0 || hh>23) return 0;                   // hour validation
  if (mm<0 || mm>59) return 0;                   // min validation
  return 1;                                        // valid time
}
```

4 . FILE HANDLING (SAVING AND LOADING)

Tasks are stored in tasks.txt using a line-based CSV format.

SAVING TASKS

Saves the tasks in the file so that they can be preserved even after the program exits.

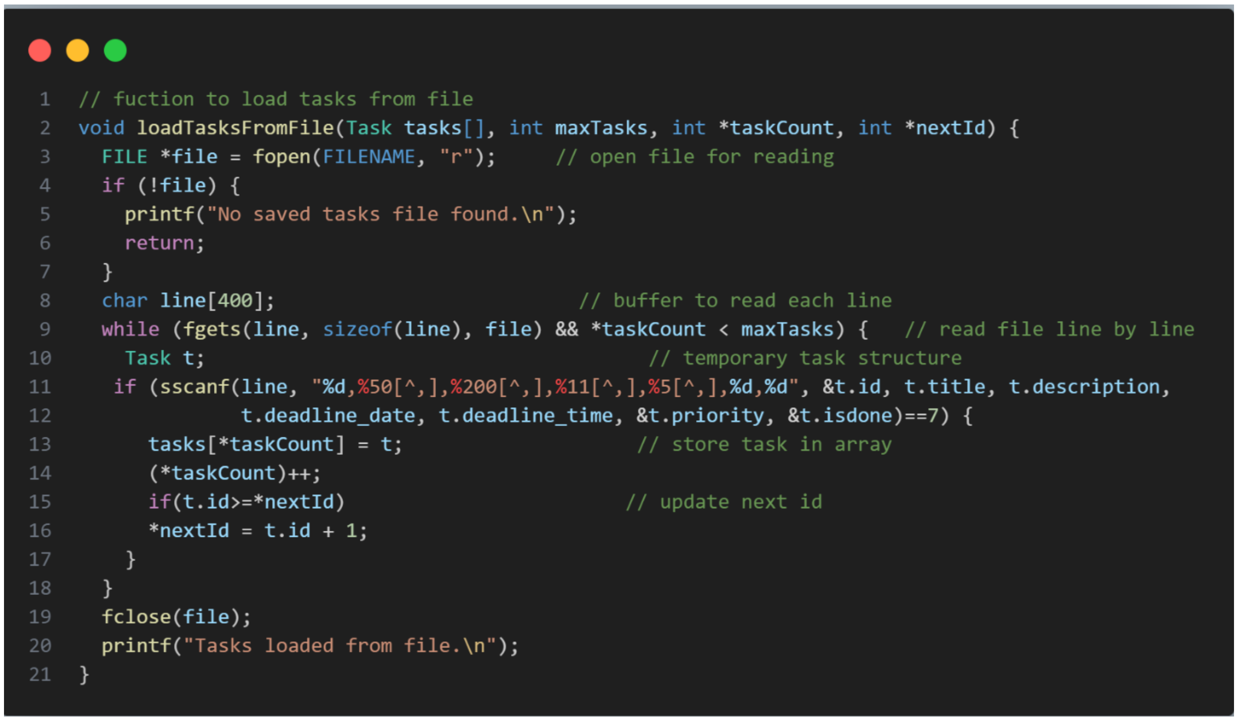


```c
// Function to save task to file
void saveTasksToFile(const Task tasks[], int taskCount) {
  FILE *file = fopen(FILENAME, "w");     // open file in write mode
  if (!file) {
    printf("Error opening file for saving.\n");
    return;
  }
for (int i=0; i<taskCount; i++){         // Write all tasks to file in CSV format
 fprintf(file, "%d,%s,%s,%s,%s,%d,%d\n", tasks[i].id, tasks[i].title, tasks[i].description,
         tasks[i].deadline_date,tasks[i].deadline_time, tasks[i].priority, tasks[i].isdone);
   }
   fclose(file);                       // close file
   printf("Tasks saved in file.\n");
}
```

LOADING TASKS

The function loads the saved tasks that was saved in file

Automatically after changes or manually.



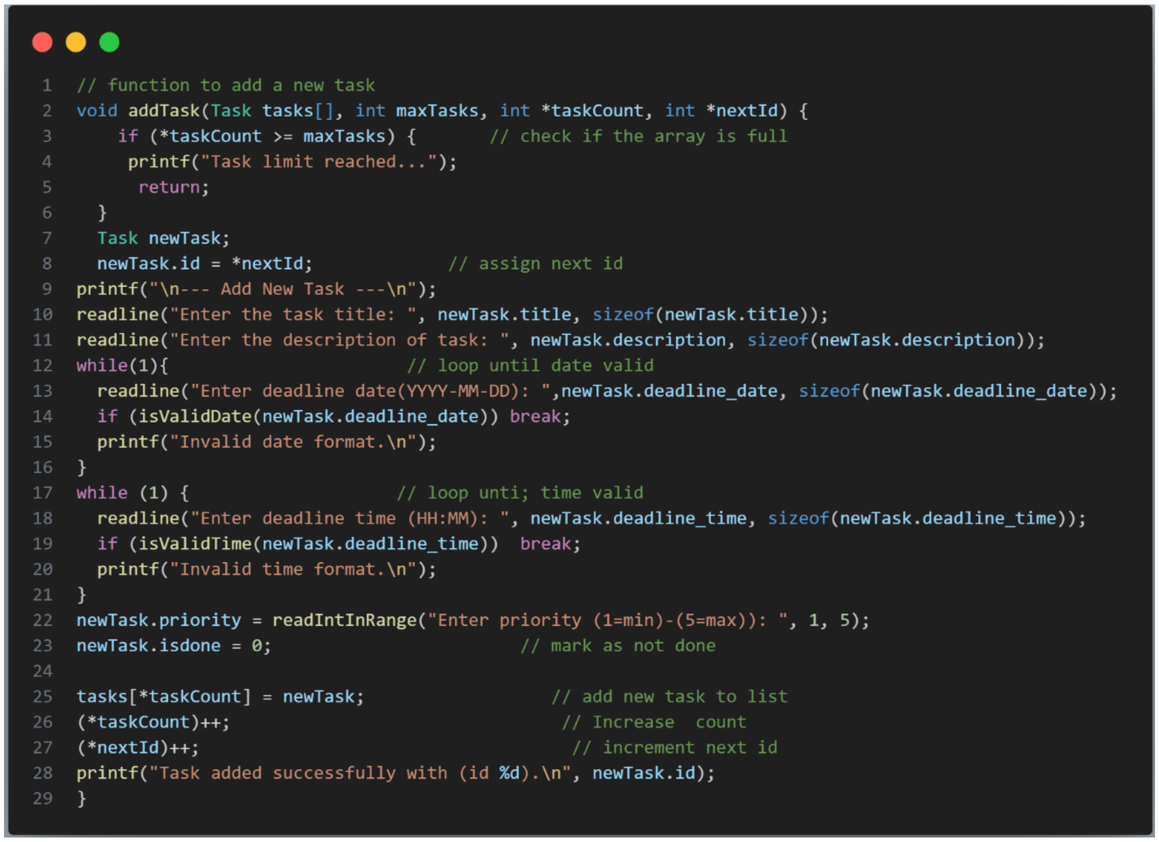
```c
// fuction to load tasks from file
void loadTasksFromFile(Task tasks[], int maxTasks, int *taskCount, int *nextId) {
  FILE *file = fopen(FILENAME, "r");     // open file for reading
  if (!file) {
    printf("No saved tasks file found.\n");
    return;
  }
  char line[400];                          // buffer to read each line
  while (fgets(line, sizeof(line), file) && *taskCount < maxTasks) {   // read file line by line
    Task t;                                  // temporary task structure
   if (sscanf(line, "%d,%50[^,],%200[^,],%11[^,],%5[^,],%d,%d", &t.id, t.title, t.description,
              t.deadline_date, t.deadline_time, &t.priority, &t.isdone)==7) {
      tasks[*taskCount] = t;                    // store task in array
      (*taskCount)++;
      if(t.id>=*nextId)                        // update next id
      *nextId = t.id + 1;
    }
  }
  fclose(file);
  printf("Tasks loaded from file.\n");
}
```

5 . CORE FEATURES

A . ADD A TASK

The function addTask() collects all the required details

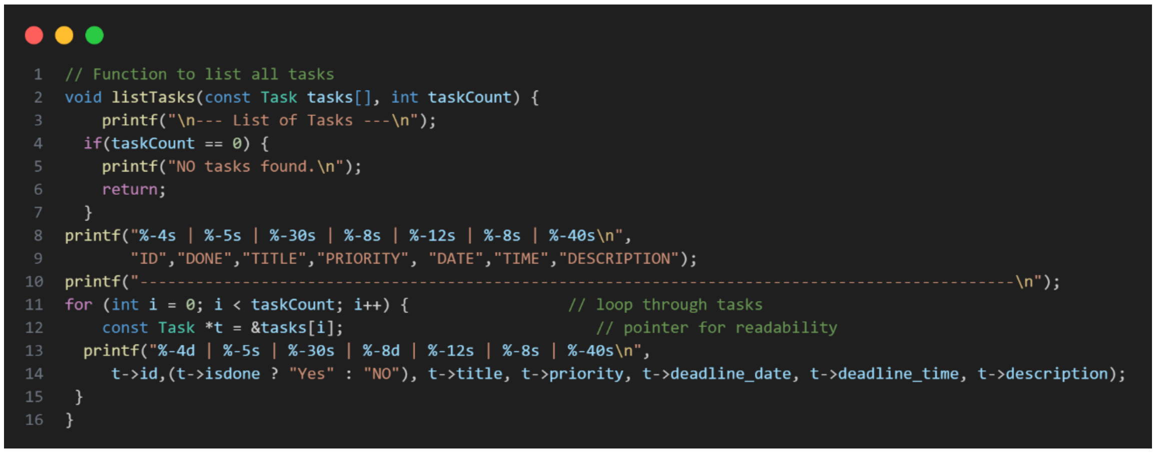
From the user with their required validation and store them in array.



```c
// function to add a new task
void addTask(Task tasks[], int maxTasks, int *taskCount, int *nextId) {
    if (*taskCount >= maxTasks) {       // check if the array is full
     printf("Task limit reached...");
      return;
  }
  Task newTask;
  newTask.id = *nextId;              // assign next id
printf("\n--- Add New Task ---\n");
readline("Enter the task title: ", newTask.title, sizeof(newTask.title));
readline("Enter the description of task: ", newTask.description, sizeof(newTask.description));
while(1){                       // loop until date valid
  readline("Enter deadline date(YYYY-MM-DD): ",newTask.deadline_date, sizeof(newTask.deadline_date));
  if (isValidDate(newTask.deadline_date)) break;
  printf("Invalid date format.\n");
}
while (1) {                    // loop unti; time valid
  readline("Enter deadline time (HH:MM): ", newTask.deadline_time, sizeof(newTask.deadline_time));
  if (isValidTime(newTask.deadline_time))  break;
  printf("Invalid time format.\n");
}
newTask.priority = readIntInRange("Enter priority (1=min)-(5=max)): ", 1, 5);
newTask.isdone = 0;                        // mark as not done

tasks[*taskCount] = newTask;                  // add new task to list
(*taskCount)++;                                // Increase  count
(*nextId)++;                                    // increment next id
printf("Task added successfully with (id %d).\n", newTask.id);
}
```

B . LIST ALL TASKS

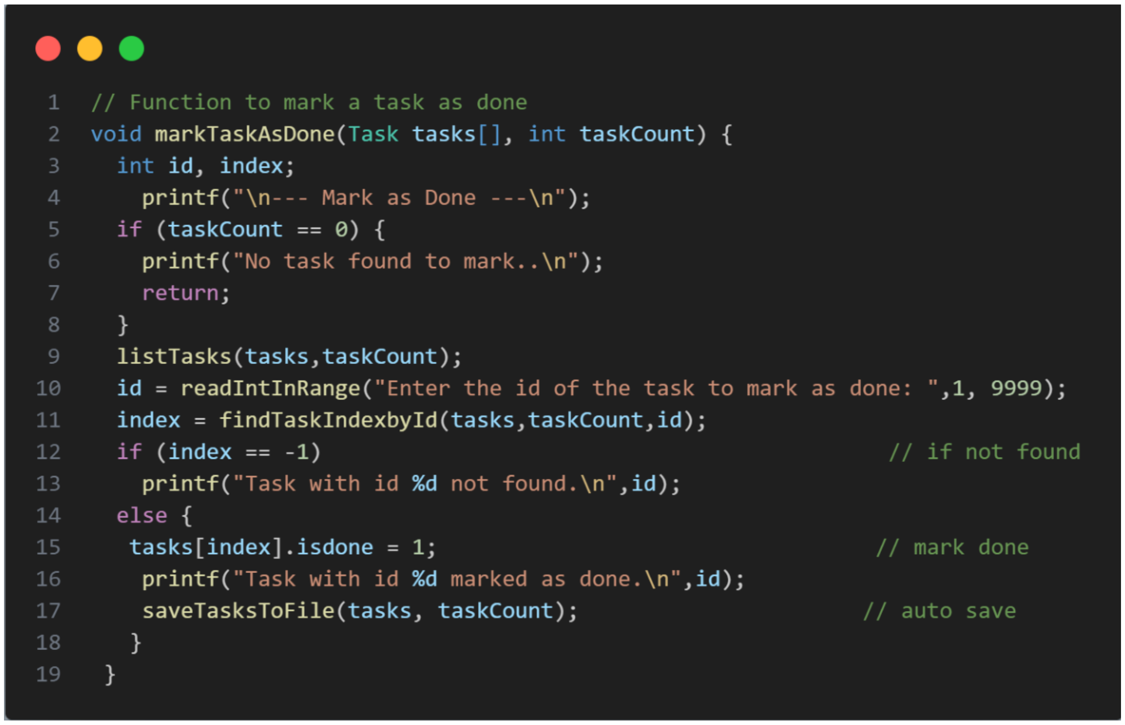
Displays all tasks in a formatted table.



```c
// Function to list all tasks
void listTasks(const Task tasks[], int taskCount) {
    printf("\n--- List of Tasks ---\n");
  if(taskCount == 0) {
    printf("NO tasks found.\n");
    return;
  }
printf("%-4s | %-5s | %-30s | %-8s | %-12s | %-8s | %-40s\n",
       "ID","DONE","TITLE","PRIORITY", "DATE","TIME","DESCRIPTION");
printf("----------------------------------------------------------------------------------------------\n");
for (int i = 0; i < taskCount; i++) {                 // loop through tasks
    const Task *t = &tasks[i];                           // pointer for readability
  printf("%-4d | %-5s | %-30s | %-8d | %-12s | %-8s | %-40s\n",
     t->id,(t->isdone ? "Yes" : "NO"), t->title, t->priority, t->deadline_date, t->deadline_time, t->description);
 }
}
```

C . MARK AS DONE

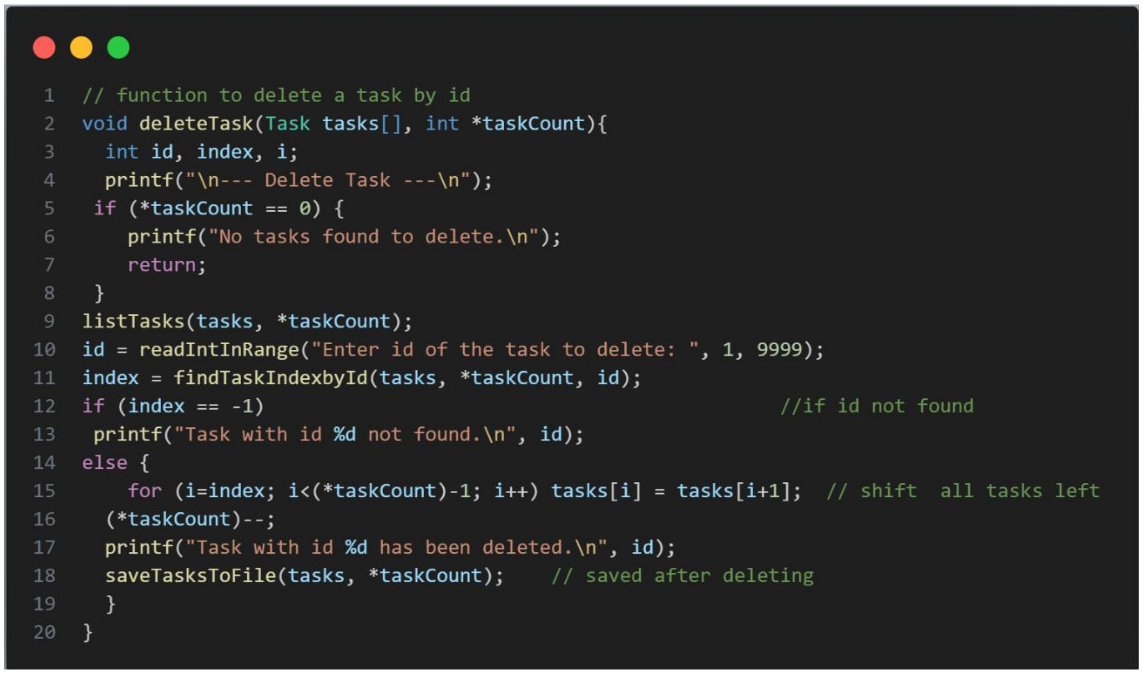
This fuction help the user to mark the task status.



```c
// Function to mark a task as done
void markTaskAsDone(Task tasks[], int taskCount) {
  int id, index;
    printf("\n--- Mark as Done ---\n");
  if (taskCount == 0) {
    printf("No task found to mark..\n");
    return;
  }
  listTasks(tasks,taskCount);
  id = readIntInRange("Enter the id of the task to mark as done: ",1, 9999);
  index = findTaskIndexbyId(tasks,taskCount,id);
  if (index == -1)                                            // if not found
    printf("Task with id %d not found.\n",id);
  else {
   tasks[index].isdone = 1;                                  // mark done
    printf("Task with id %d marked as done.\n",id);
    saveTasksToFile(tasks, taskCount);                      // auto save
   }
 }
```

D . DELETE A TASK

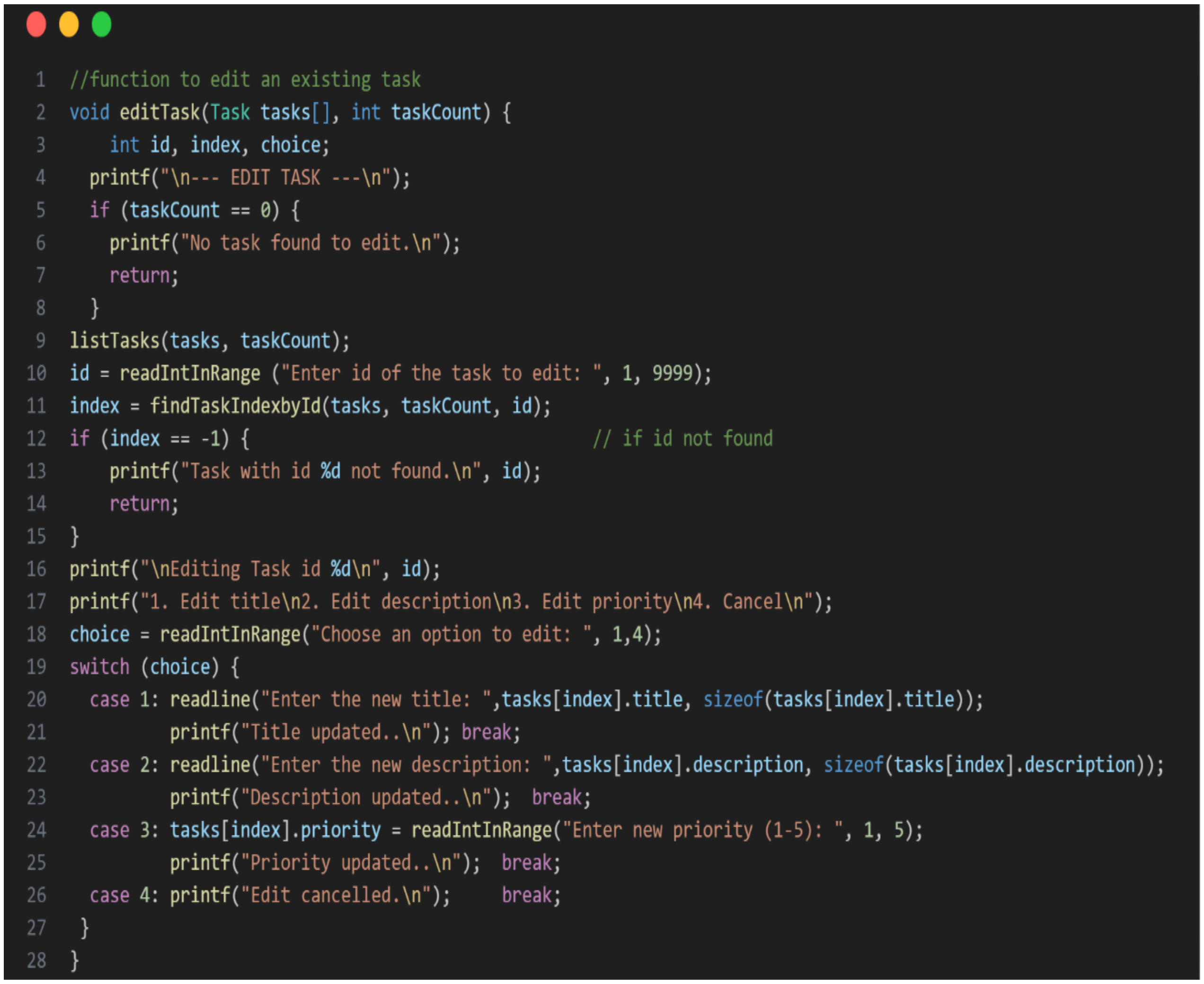
Allow the user to delete any saved task with its id. The program shifts all tasks one position left after deleting any task.



```c
// function to delete a task by id
void deleteTask(Task tasks[], int *taskCount){
  int id, index, i;
  printf("\n--- Delete Task ---\n");
 if (*taskCount == 0) {
    printf("No tasks found to delete.\n");
    return;
 }
listTasks(tasks, *taskCount);
id = readIntInRange("Enter id of the task to delete: ", 1, 9999);
index = findTaskIndexbyId(tasks, *taskCount, id);
if (index == -1)                                          //if id not found
 printf("Task with id %d not found.\n", id);
else {
    for (i=index; i<(*taskCount)-1; i++) tasks[i] = tasks[i+1];  // shift  all tasks left
  (*taskCount)--;
  printf("Task with id %d has been deleted.\n", id);
  saveTasksToFile(tasks, *taskCount);    // saved after deleting
  }
}
```

E . EDIT A TASK

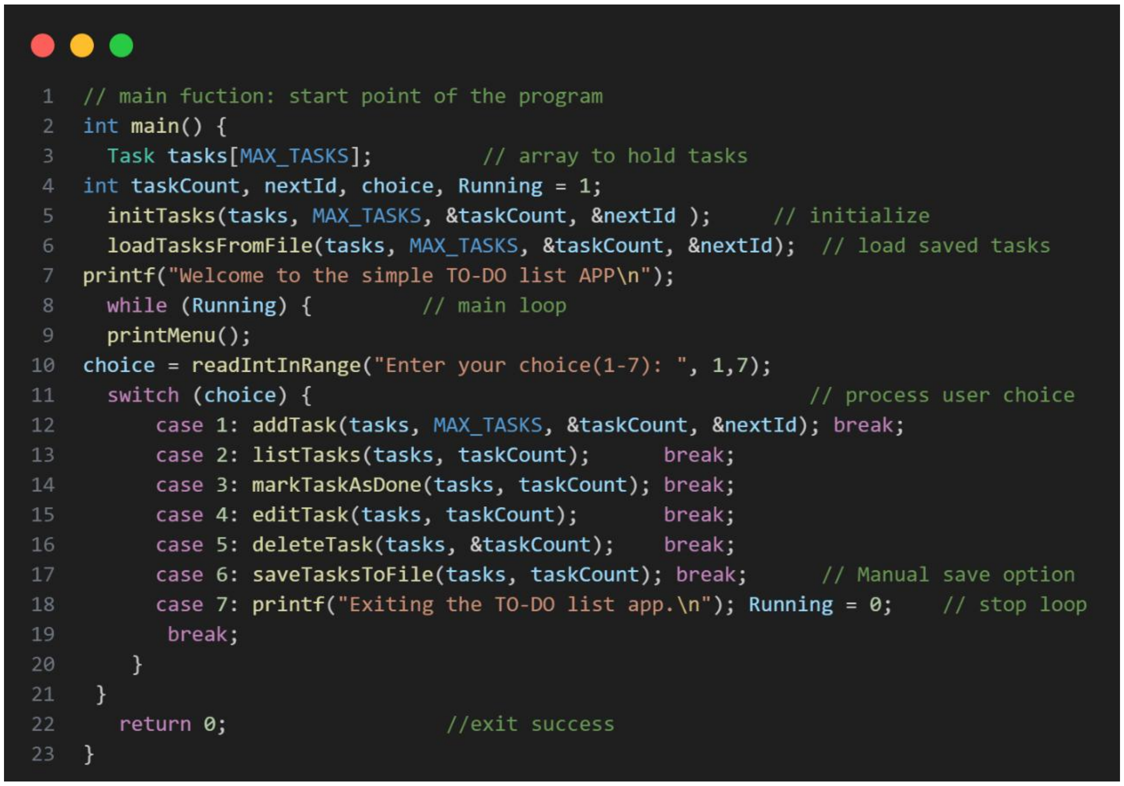
Allow the user to update title, description, or priority.



```c
//function to edit an existing task
void editTask(Task tasks[], int taskCount) {
    int id, index, choice;
  printf("\n--- EDIT TASK ---\n");
  if (taskCount == 0) {
    printf("No task found to edit.\n");
    return;
  }
listTasks(tasks, taskCount);
id = readIntInRange ("Enter id of the task to edit: ", 1, 9999);
index = findTaskIndexbyId(tasks, taskCount, id);
if (index == -1) {                               // if id not found
    printf("Task with id %d not found.\n", id);
    return;
}
printf("\nEditing Task id %d\n", id);
printf("1. Edit title\n2. Edit description\n3. Edit priority\n4. Cancel\n");
choice = readIntInRange("Choose an option to edit: ", 1,4);
switch (choice) {
  case 1: readline("Enter the new title: ",tasks[index].title, sizeof(tasks[index].title));
          printf("Title updated..\n"); break;
  case 2: readline("Enter the new description: ",tasks[index].description, sizeof(tasks[index].description));
          printf("Description updated..\n");  break;
  case 3: tasks[index].priority = readIntInRange("Enter new priority (1-5): ", 1, 5);
          printf("Priority updated..\n");  break;
  case 4: printf("Edit cancelled.\n");     break;
 }
}
```

6 . MAIN LOOP OF THE PROGRAM

The main loop repeatedly display the menu with the 7 options until the user select the exit option.

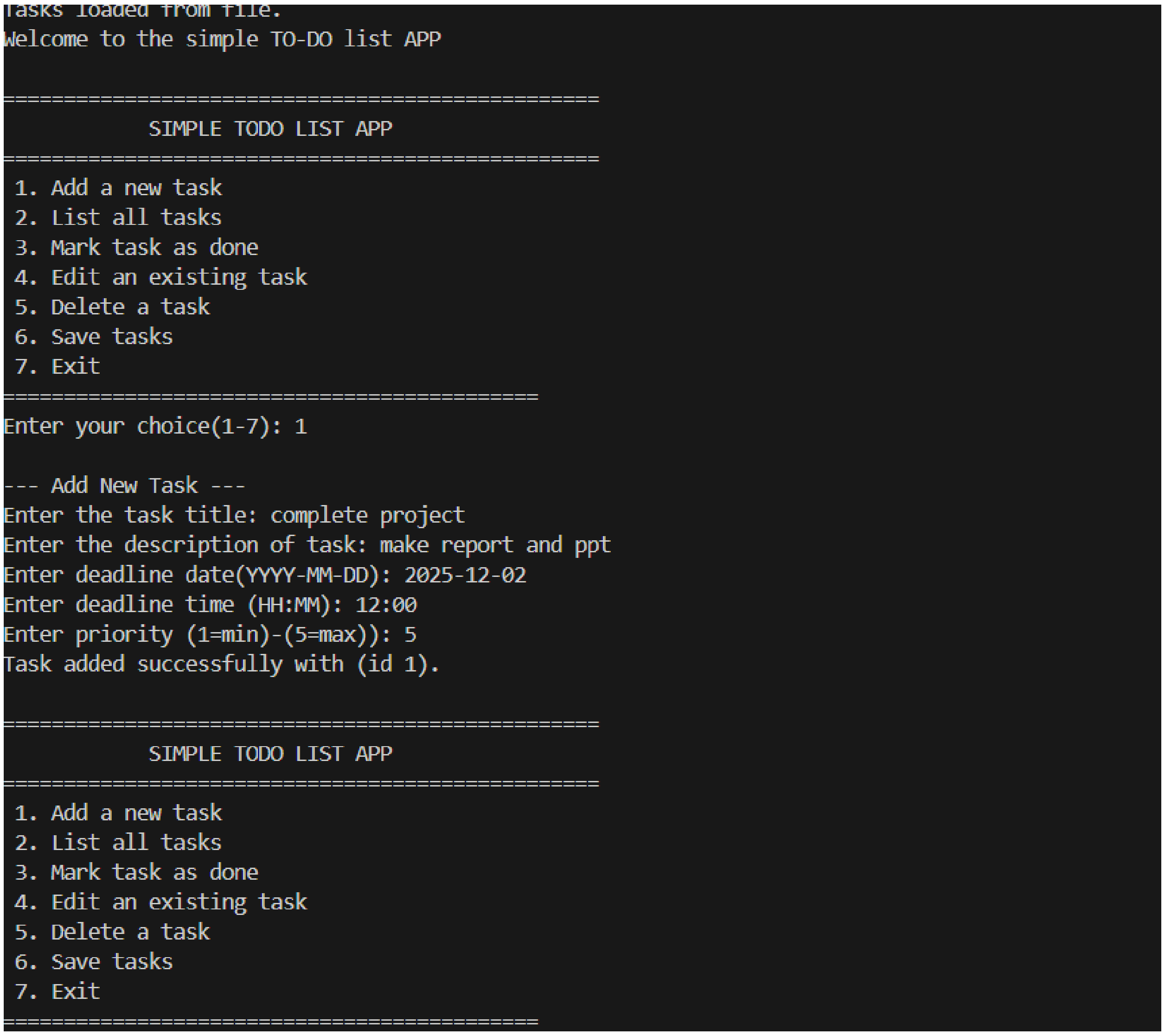


```c
// main fuction: start point of the program
int main() {
  Task tasks[MAX_TASKS];         // array to hold tasks
int taskCount, nextId, choice, Running = 1;
  initTasks(tasks, MAX_TASKS, &taskCount, &nextId );     // initialize
  loadTasksFromFile(tasks, MAX_TASKS, &taskCount, &nextId);  // load saved tasks
printf("Welcome to the simple TO-DO list APP\n");
  while (Running) {         // main loop
  printMenu();
choice = readIntInRange("Enter your choice(1-7): ", 1,7);
  switch (choice) {                                     // process user choice
      case 1: addTask(tasks, MAX_TASKS, &taskCount, &nextId); break;
      case 2: listTasks(tasks, taskCount);      break;
      case 3: markTaskAsDone(tasks, taskCount); break;
      case 4: editTask(tasks, taskCount);       break;
      case 5: deleteTask(tasks, &taskCount);    break;
      case 6: saveTasksToFile(tasks, taskCount); break;      // Manual save option
      case 7: printf("Exiting the TO-DO list app.\n"); Running = 0;    // stop loop
       break;
    }
 }
   return 0;                  //exit success
}
```

TESTING AND RESULT

THIS IS OUTPUT OF CODE WITH VALID INPUTS

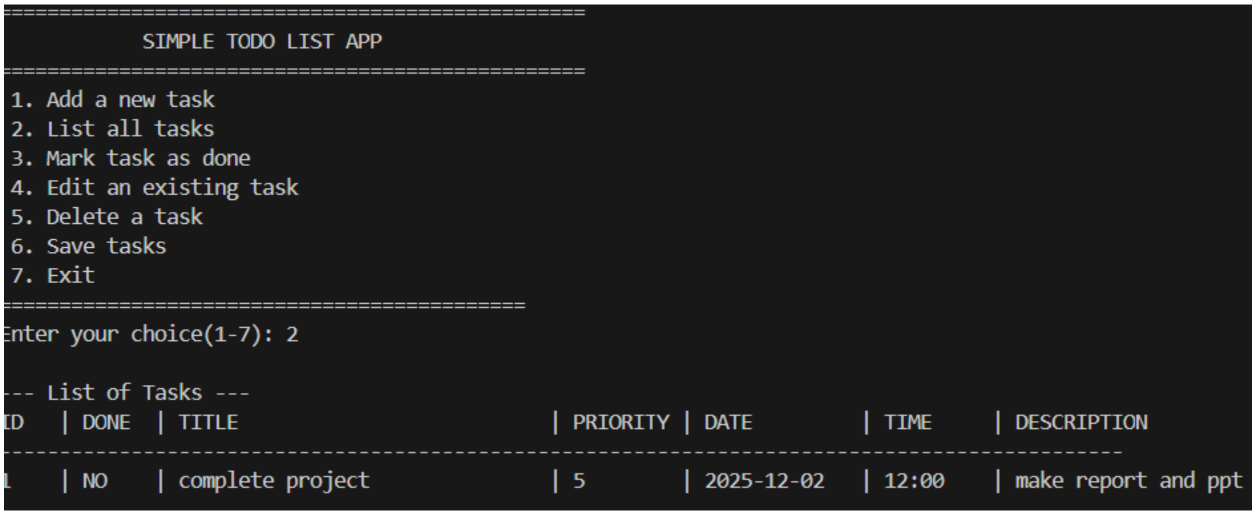
THE MENU WILL APPER AGAIN AND AGAIN UNTIL THE USER SELECT EXIT(7).



```
Tasks loaded from file.
Welcome to the simple TO-DO list APP

=================================================
           SIMPLE TODO LIST APP
=================================================
 1. Add a new task
 2. List all tasks
 3. Mark task as done
 4. Edit an existing task
 5. Delete a task
 6. Save tasks
 7. Exit
============================================
Enter your choice(1-7): 1

--- Add New Task ---
Enter the task title: complete project
Enter the description of task: make report and ppt
Enter deadline date(YYYY-MM-DD): 2025-12-02
Enter deadline time (HH:MM): 12:00
Enter priority (1=min)-(5=max)): 5
Task added successfully with (id 1).

=================================================
           SIMPLE TODO LIST APP
=================================================
 1. Add a new task
 2. List all tasks
 3. Mark task as done
 4. Edit an existing task
 5. Delete a task
 6. Save tasks
 7. Exit
============================================
```

OUTPUT WHEN THE USER SELECT LIST ALL

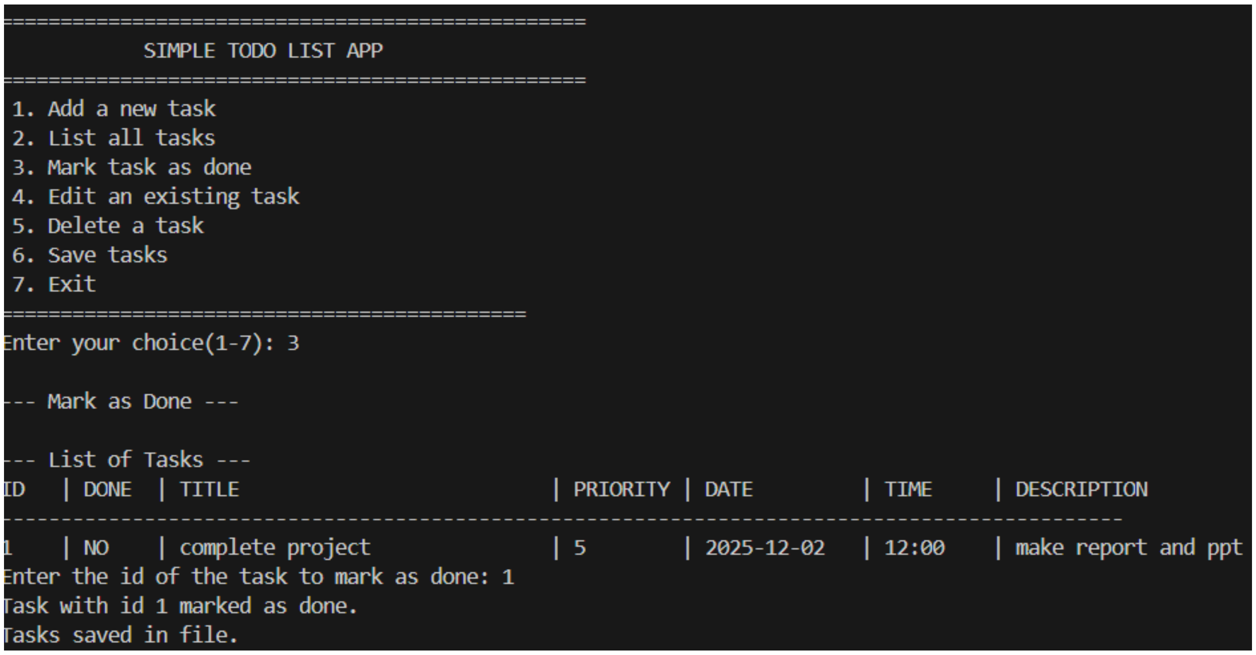
- THE FIRST ENTERED TASK WILL GET ID 1



```
================================================
          SIMPLE TODO LIST APP
================================================
 1. Add a new task
 2. List all tasks
 3. Mark task as done
 4. Edit an existing task
 5. Delete a task
 6. Save tasks
 7. Exit
===========================================
Enter your choice(1-7): 2

--- List of Tasks ---
ID   | DONE   | TITLE                          | PRIORITY | DATE         | TIME     | DESCRIPTION
---------------------------------------------------------------------------------------------
1    | NO     | complete project               | 5        | 2025-12-02   | 12:00    | make report and ppt
```

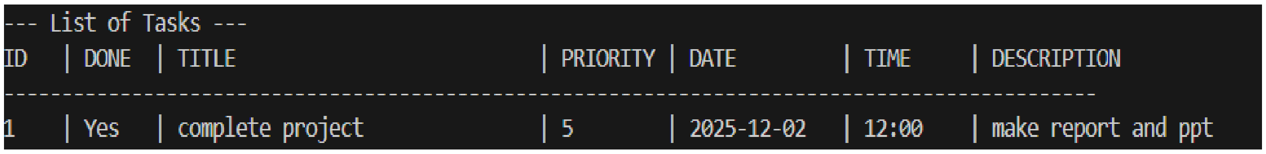
OUTPUT WHEN SELECTED MARK AS DONE

- THE UPDATE IS AUTO SAVED IN FILE



```
================================================
          SIMPLE TODO LIST APP
================================================
 1. Add a new task
 2. List all tasks
 3. Mark task as done
 4. Edit an existing task
 5. Delete a task
 6. Save tasks
 7. Exit
===========================================
Enter your choice(1-7): 3

--- Mark as Done ---

--- List of Tasks ---
ID   | DONE   | TITLE                          | PRIORITY | DATE         | TIME     | DESCRIPTION
---------------------------------------------------------------------------------------------
1    | NO     | complete project               | 5        | 2025-12-02   | 12:00    | make report and ppt
Enter the id of the task to mark as done: 1
Task with id 1 marked as done.
Tasks saved in file.
```

- AFTER UPDATING THE STATUS



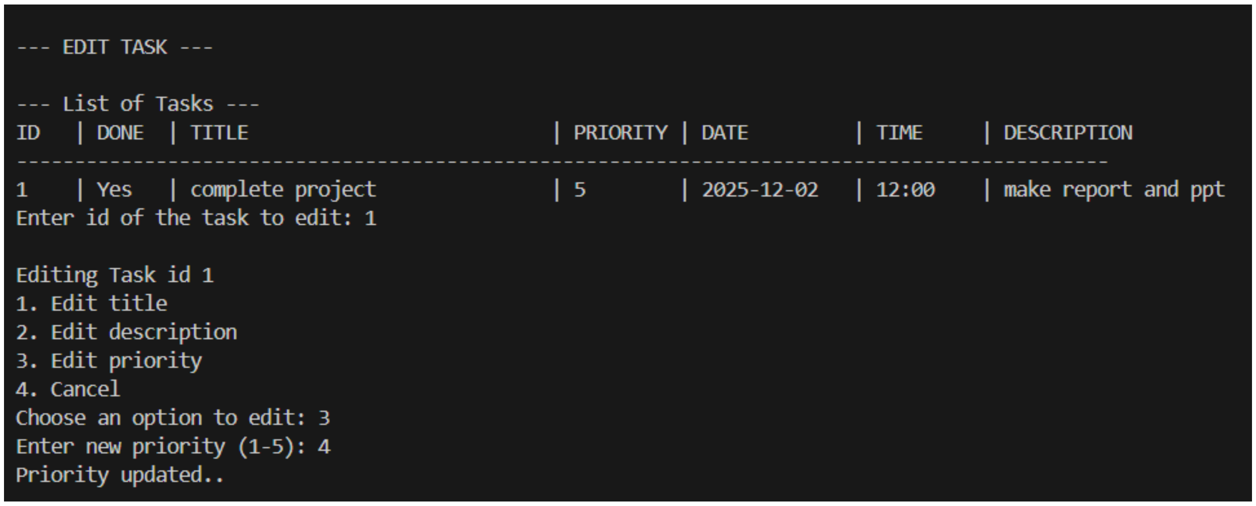
```
--- List of Tasks ---
ID    | DONE  | TITLE                          | PRIORITY | DATE        | TIME     | DESCRIPTION
----------------------------------------------------------------------------------------------
1     | Yes   | complete project               | 5        | 2025-12-02  | 12:00    | make report and ppt
```

OUTPUT OF EDIT AN EXISTING TASK

Edit menu:

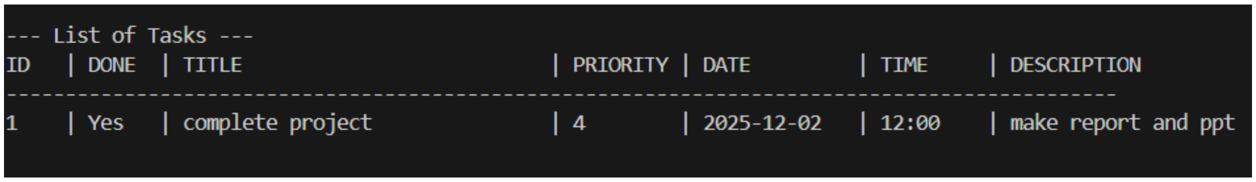
With the edit opton

(eg) user select to edit priority.



```
--- EDIT TASK ---

--- List of Tasks ---
ID    | DONE  | TITLE                          | PRIORITY | DATE        | TIME     | DESCRIPTION
----------------------------------------------------------------------------------------------
1     | Yes   | complete project               | 5        | 2025-12-02  | 12:00    | make report and ppt
Enter id of the task to edit: 1

Editing Task id 1
1. Edit title
2. Edit description
3. Edit priority
4. Cancel
Choose an option to edit: 3
Enter new priority (1-5): 4
Priority updated..
```

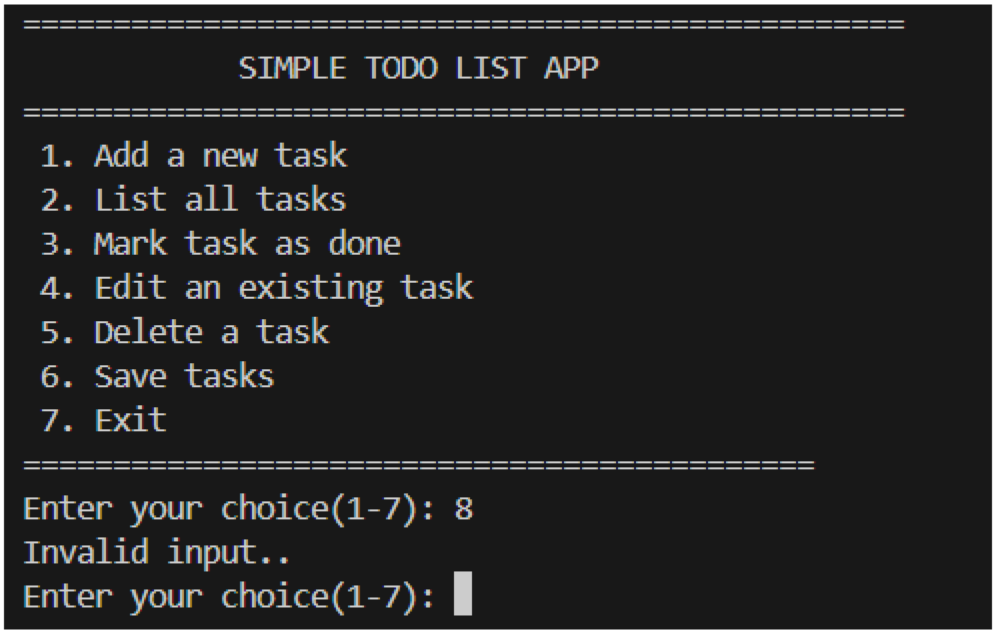
Updated priority.



```
--- List of Tasks ---
ID    | DONE  | TITLE                          | PRIORITY | DATE        | TIME     | DESCRIPTION
----------------------------------------------------------------------------------------------
1     | Yes   | complete project               | 4        | 2025-12-02  | 12:00    | make report and ppt
```

OUTPUT IF INPUT IS IN-VALID

In main fun. the menu will repeat until the user give a valid input.



```
==================================================
            SIMPLE TODO LIST APP
==================================================
 1. Add a new task
 2. List all tasks
 3. Mark task as done
 4. Edit an existing task
 5. Delete a task
 6. Save tasks
 7. Exit
=============================================
Enter your choice(1-7): 8
Invalid input..
Enter your choice(1-7): 
```

CONCLUSION:

The To-Do list application demononstrates that how using the C programming concepts a practical, user friendly task management system can create. By using structures, arrays,file handling and modular functions this project organize data efficiently and ensure the long term stroage of the tasks through saving them in file.

The program successfully performs all the essential operations such as adding, listing, editing, deleting and marking tasks as complete. It’s a User friendly application

Overall this project highlights the importance of structure programming, clean code organisation, and data persistence.

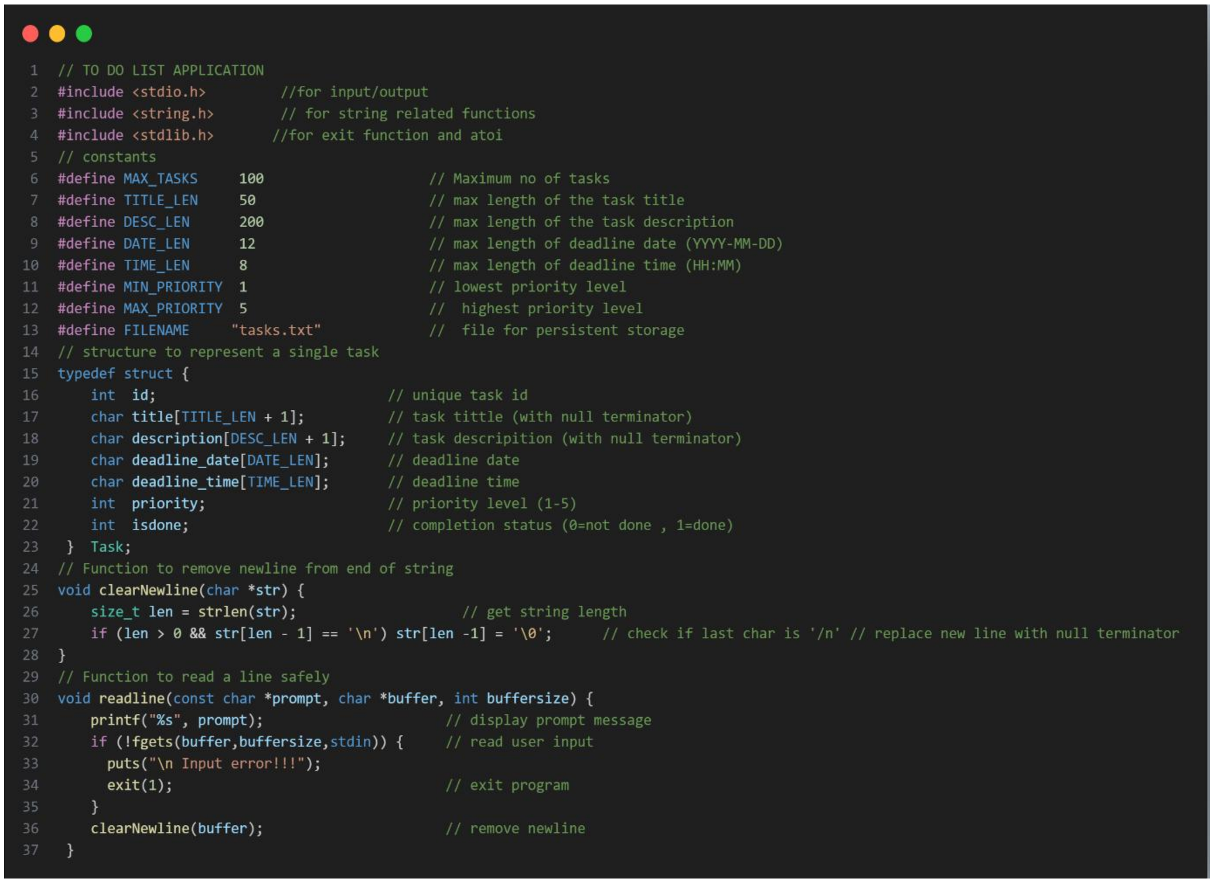
FUTURE ENHANCEMENTS:

- Add sorting Features(by priority,deadline date,status,or alphabetical order).
- Search and Filter options.
- Undo/Delete recovery feature.
- Use dynamic mem. instead of fixed-size arrays.

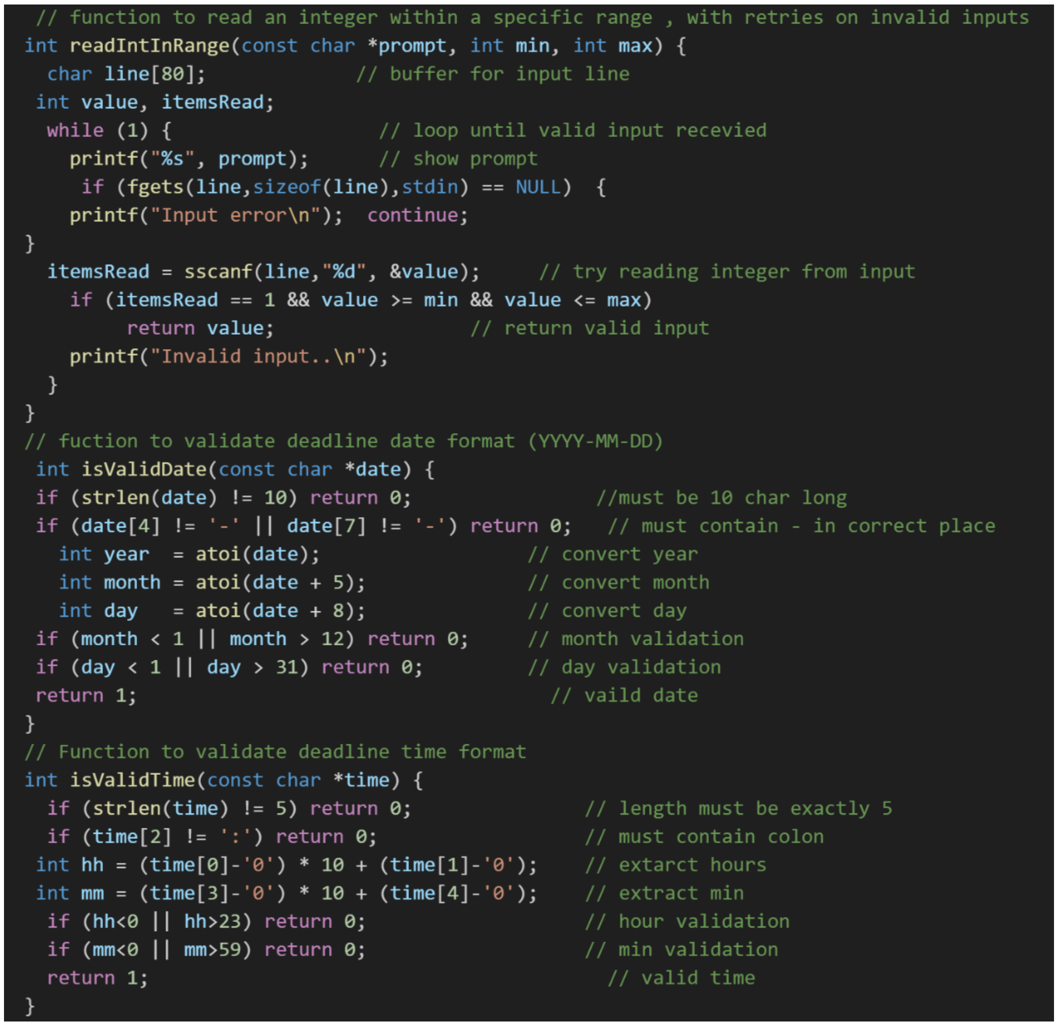
REFERENCES:

- UPES University course material and class notes
- Online c tutorials

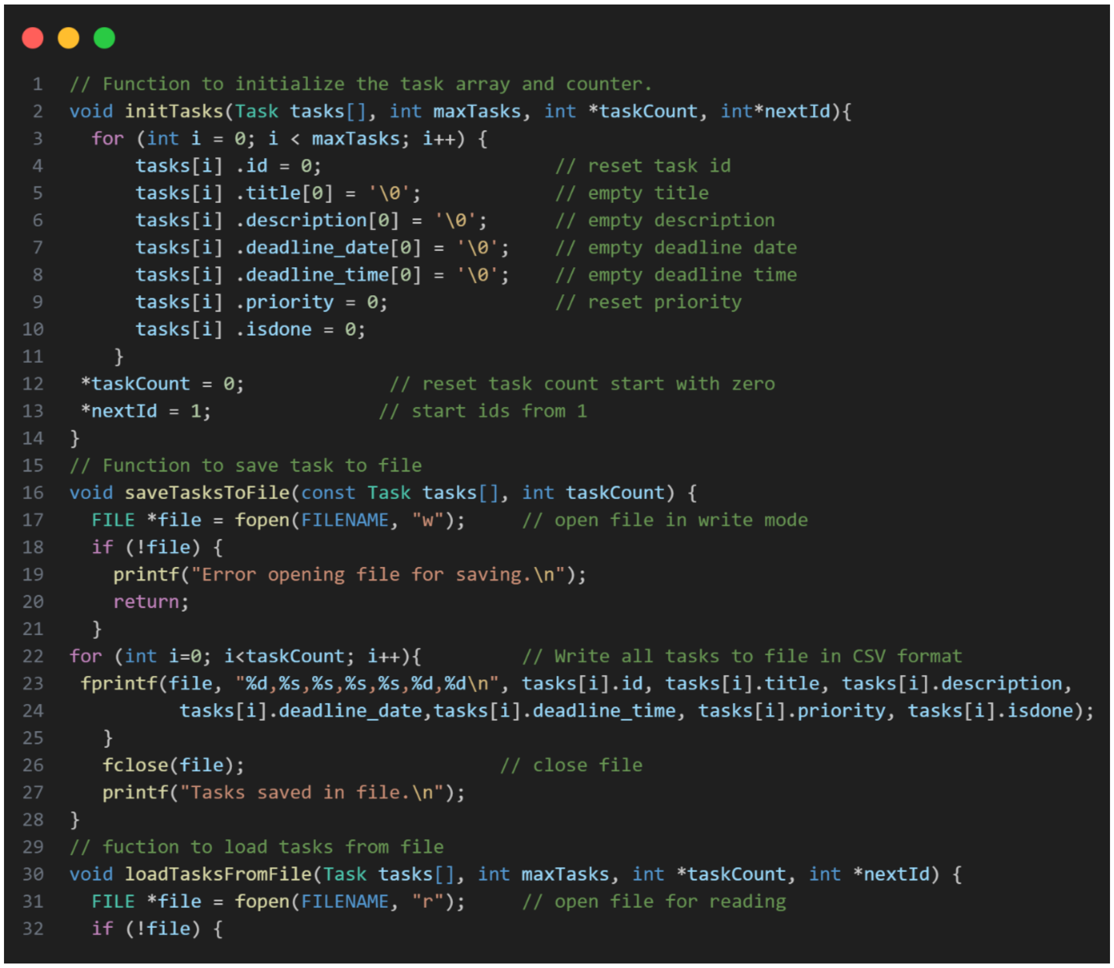
APPENDIX:



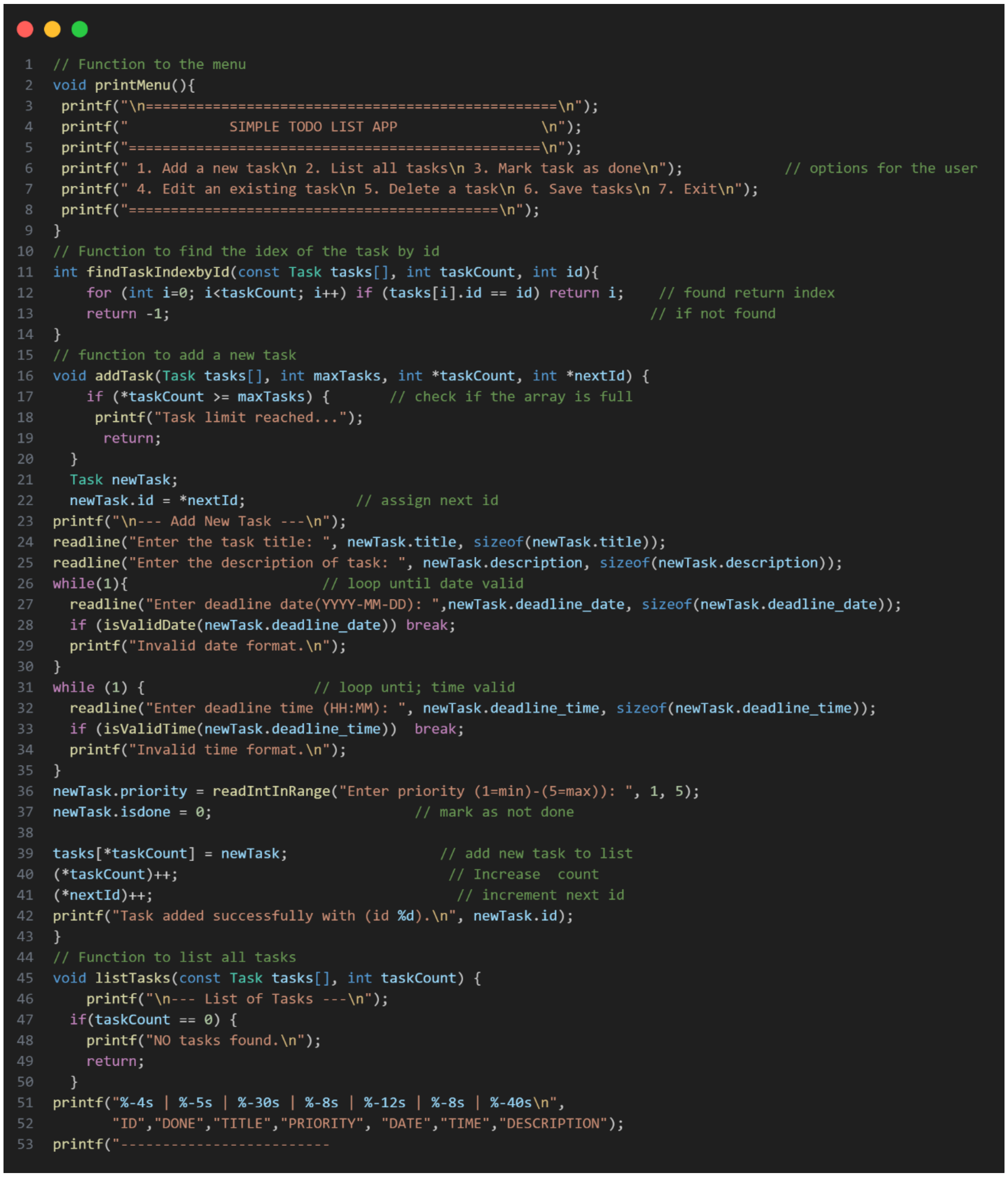
```c
// TO DO LIST APPLICATION
#include <stdio.h>         //for input/output
#include <string.h>        // for string related functions
#include <stdlib.h>       //for exit function and atoi
// constants
#define MAX_TASKS     100                  // Maximum no of tasks
#define TITLE_LEN     50                   // max length of the task title
#define DESC_LEN      200                  // max length of the task description
#define DATE_LEN      12                   // max length of deadline date (YYYY-MM-DD)
#define TIME_LEN      8                    // max length of deadline time (HH:MM)
#define MIN_PRIORITY  1                    // lowest priority level
#define MAX_PRIORITY  5                    //  highest priority level
#define FILENAME     "tasks.txt"           //  file for persistent storage
// structure to represent a single task
typedef struct {
    int  id;                          // unique task id
    char title[TITLE_LEN + 1];        // task tittle (with null terminator)
    char description[DESC_LEN + 1];   // task descripition (with null terminator)
    char deadline_date[DATE_LEN];     // deadline date
    char deadline_time[TIME_LEN];     // deadline time
    int  priority;                    // priority level (1-5)
    int  isdone;                      // completion status (0=not done , 1=done)
 }  Task;
// Function to remove newline from end of string
void clearNewline(char *str) {
    size_t len = strlen(str);                   // get string length
    if (len > 0 && str[len - 1] == '\n') str[len -1] = '\0';      // check if last char is '/n' // replace new line with null terminator
}
// Function to read a line safely
void readline(const char *prompt, char *buffer, int buffersize) {
    printf("%s", prompt);                      // display prompt message
    if (!fgets(buffer,buffersize,stdin)) {     // read user input
      puts("\n Input error!!!");
      exit(1);                                 // exit program
    }
    clearNewline(buffer);                      // remove newline
 }
```



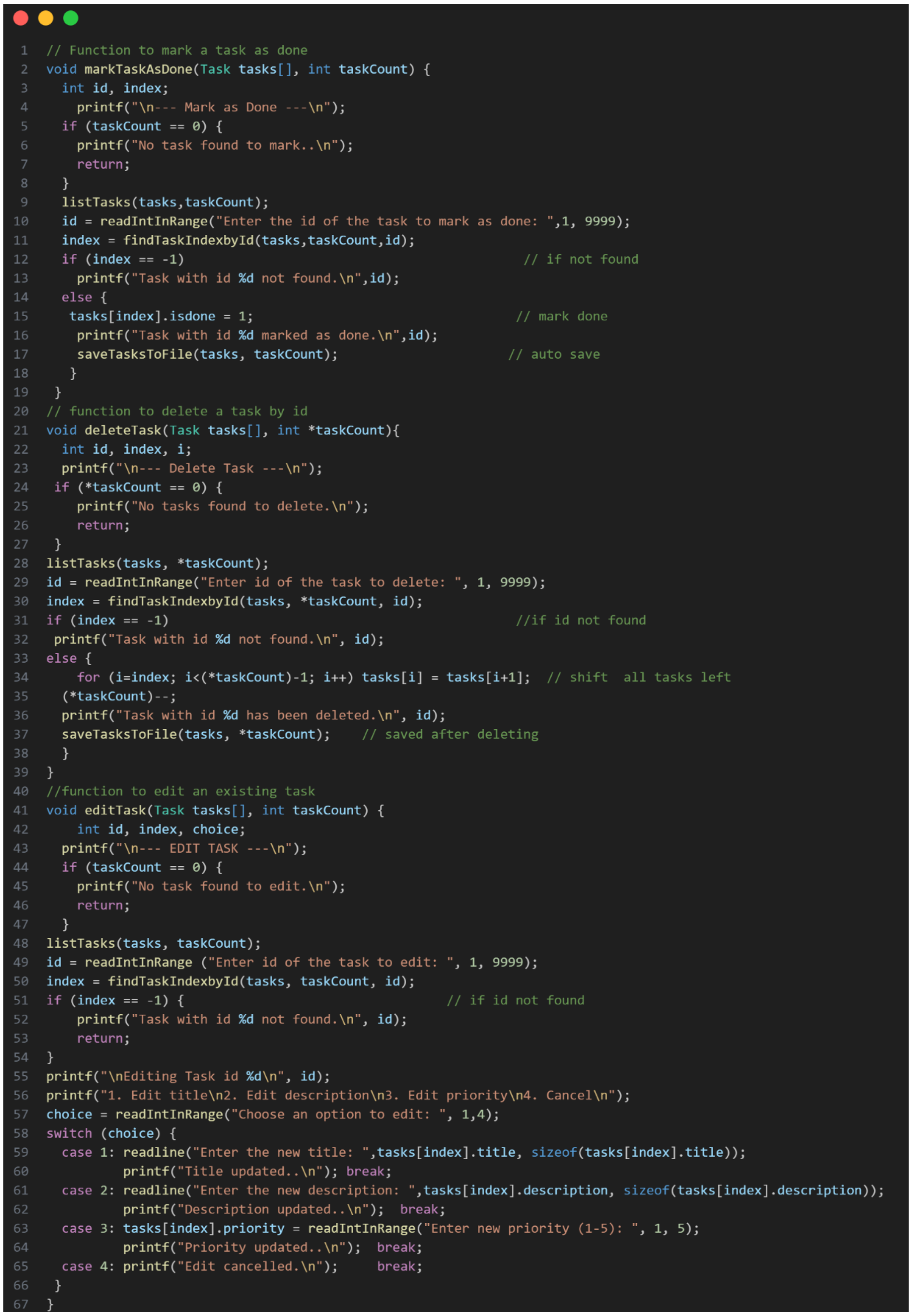
```c
 // function to read an integer within a specific range , with retries on invalid inputs
int readIntInRange(const char *prompt, int min, int max) {
  char line[80];             // buffer for input line
 int value, itemsRead;
  while (1) {                  // loop until valid input recevied
    printf("%s", prompt);      // show prompt
     if (fgets(line,sizeof(line),stdin) == NULL)  {
    printf("Input error\n");  continue;
}
  itemsRead = sscanf(line,"%d", &value);     // try reading integer from input
    if (itemsRead == 1 && value >= min && value <= max)
         return value;                 // return valid input
    printf("Invalid input..\n");
  }
}
// fuction to validate deadline date format (YYYY-MM-DD)
 int isValidDate(const char *date) {
 if (strlen(date) != 10) return 0;                //must be 10 char long
 if (date[4] != '-' || date[7] != '-') return 0;   // must contain - in correct place
   int year  = atoi(date);                  // convert year
   int month = atoi(date + 5);              // convert month
   int day   = atoi(date + 8);              // convert day
 if (month < 1 || month > 12) return 0;     // month validation
 if (day < 1 || day > 31) return 0;         // day validation
 return 1;                                    // vaild date
}
// Function to validate deadline time format
int isValidTime(const char *time) {
  if (strlen(time) != 5) return 0;               // length must be exactly 5
  if (time[2] != ':') return 0;                  // must contain colon
 int hh = (time[0]-'0') * 10 + (time[1]-'0');    // extarct hours
 int mm = (time[3]-'0') * 10 + (time[4]-'0');    // extract min
  if (hh<0 || hh>23) return 0;                   // hour validation
  if (mm<0 || mm>59) return 0;                   // min validation
  return 1;                                        // valid time
}
```



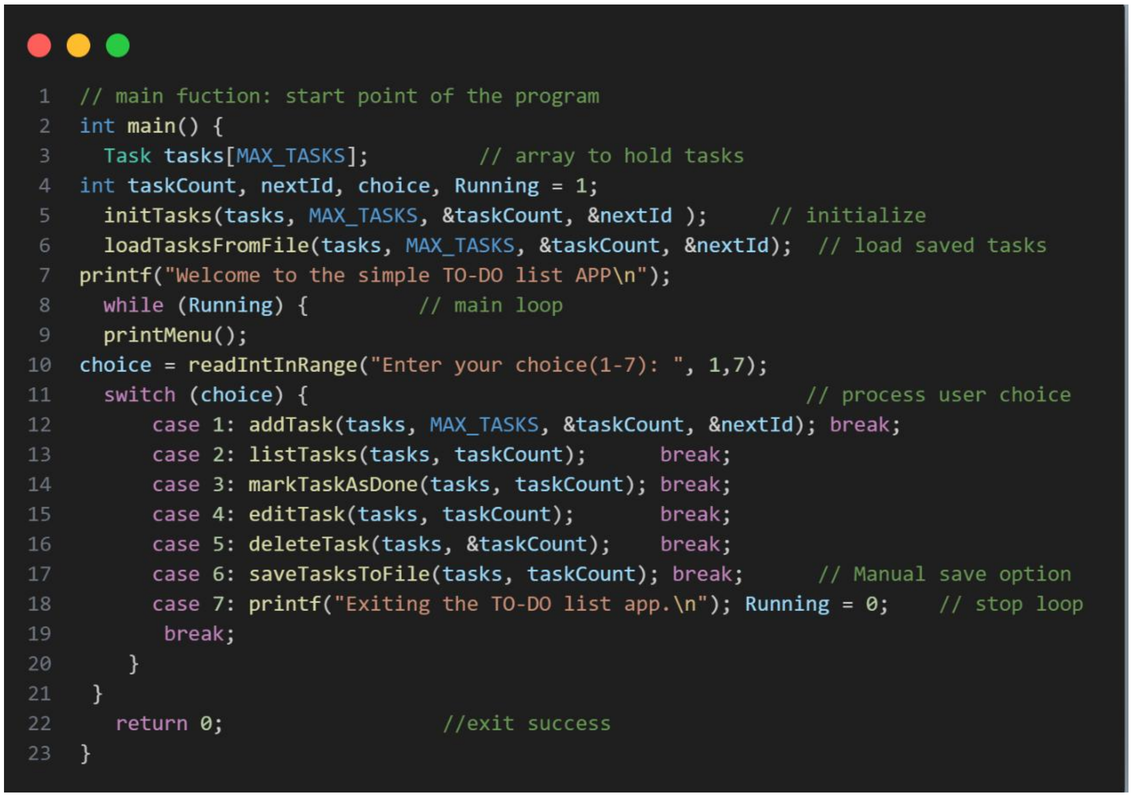
```c
// Function to initialize the task array and counter.
void initTasks(Task tasks[], int maxTasks, int *taskCount, int*nextId){
  for (int i = 0; i < maxTasks; i++) {
      tasks[i] .id = 0;                     // reset task id
      tasks[i] .title[0] = '\0';            // empty title
      tasks[i] .description[0] = '\0';      // empty description
      tasks[i] .deadline_date[0] = '\0';    // empty deadline date
      tasks[i] .deadline_time[0] = '\0';    // empty deadline time
      tasks[i] .priority = 0;               // reset priority
      tasks[i] .isdone = 0;
    }
 *taskCount = 0;             // reset task count start with zero
 *nextId = 1;               // start ids from 1
}
// Function to save task to file
void saveTasksToFile(const Task tasks[], int taskCount) {
  FILE *file = fopen(FILENAME, "w");     // open file in write mode
  if (!file) {
    printf("Error opening file for saving.\n");
    return;
  }
for (int i=0; i<taskCount; i++){         // Write all tasks to file in CSV format
 fprintf(file, "%d,%s,%s,%s,%s,%d,%d\n", tasks[i].id, tasks[i].title, tasks[i].description,
          tasks[i].deadline_date,tasks[i].deadline_time, tasks[i].priority, tasks[i].isdone);
   }
   fclose(file);                       // close file
   printf("Tasks saved in file.\n");
}
// fuction to load tasks from file
void loadTasksFromFile(Task tasks[], int maxTasks, int *taskCount, int *nextId) {
  FILE *file = fopen(FILENAME, "r");     // open file for reading
  if (!file) {
```



```
// Function to the menu
void printMenu(){
 printf("\n================================================\n");
 printf("            SIMPLE TODO LIST APP                 \n");
 printf("=================================================\n");
 printf(" 1. Add a new task\n 2. List all tasks\n 3. Mark task as done\n");            // options for the user
 printf(" 4. Edit an existing task\n 5. Delete a task\n 6. Save tasks\n 7. Exit\n");
 printf("============================================\n");
}
// Function to find the idex of the task by id
int findTaskIndexbyId(const Task tasks[], int taskCount, int id){
    for (int i=0; i<taskCount; i++) if (tasks[i].id == id) return i;    // found return index
    return -1;                                                          // if not found
}
// function to add a new task
void addTask(Task tasks[], int maxTasks, int *taskCount, int *nextId) {
    if (*taskCount >= maxTasks) {       // check if the array is full
     printf("Task limit reached...");
      return;
  }
  Task newTask;
  newTask.id = *nextId;             // assign next id
printf("\n--- Add New Task ---\n");
readline("Enter the task title: ", newTask.title, sizeof(newTask.title));
readline("Enter the description of task: ", newTask.description, sizeof(newTask.description));
while(1){                      // loop until date valid
  readline("Enter deadline date(YYYY-MM-DD): ",newTask.deadline_date, sizeof(newTask.deadline_date));
  if (isValidDate(newTask.deadline_date)) break;
  printf("Invalid date format.\n");
}
while (1) {                     // loop unti; time valid
  readline("Enter deadline time (HH:MM): ", newTask.deadline_time, sizeof(newTask.deadline_time));
  if (isValidTime(newTask.deadline_time))  break;
  printf("Invalid time format.\n");
}
newTask.priority = readIntInRange("Enter priority (1=min)-(5=max)): ", 1, 5);
newTask.isdone = 0;                         // mark as not done

tasks[*taskCount] = newTask;                   // add new task to list
(*taskCount)++;                                 // Increase  count
(*nextId)++;                                     // increment next id
printf("Task added successfully with (id %d).\n", newTask.id);
}
// Function to list all tasks
void listTasks(const Task tasks[], int taskCount) {
    printf("\n--- List of Tasks ---\n");
  if(taskCount == 0) {
    printf("NO tasks found.\n");
    return;
  }
printf("%-4s | %-5s | %-30s | %-8s | %-12s | %-8s | %-40s\n",
       "ID","DONE","TITLE","PRIORITY", "DATE","TIME","DESCRIPTION");
printf("-------------------------
```



```c
// Function to mark a task as done
void markTaskAsDone(Task tasks[], int taskCount) {
  int id, index;
    printf("\n--- Mark as Done ---\n");
  if (taskCount == 0) {
    printf("No task found to mark..\n");
    return;
  }
  listTasks(tasks,taskCount);
  id = readIntInRange("Enter the id of the task to mark as done: ",1, 9999);
  index = findTaskIndexbyId(tasks,taskCount,id);
  if (index == -1)                                            // if not found
    printf("Task with id %d not found.\n",id);
  else {
   tasks[index].isdone = 1;                                  // mark done
    printf("Task with id %d marked as done.\n",id);
    saveTasksToFile(tasks, taskCount);                      // auto save
   }
 }
// function to delete a task by id
void deleteTask(Task tasks[], int *taskCount){
  int id, index, i;
  printf("\n--- Delete Task ---\n");
 if (*taskCount == 0) {
    printf("No tasks found to delete.\n");
    return;
 }
listTasks(tasks, *taskCount);
id = readIntInRange("Enter id of the task to delete: ", 1, 9999);
index = findTaskIndexbyId(tasks, *taskCount, id);
if (index == -1)                                             //if id not found
 printf("Task with id %d not found.\n", id);
else {
    for (i=index; i<(*taskCount)-1; i++) tasks[i] = tasks[i+1];  // shift  all tasks left
  (*taskCount)--;
  printf("Task with id %d has been deleted.\n", id);
  saveTasksToFile(tasks, *taskCount);    // saved after deleting
  }
}
//function to edit an existing task
void editTask(Task tasks[], int taskCount) {
    int id, index, choice;
  printf("\n--- EDIT TASK ---\n");
  if (taskCount == 0) {
    printf("No task found to edit.\n");
    return;
  }
listTasks(tasks, taskCount);
id = readIntInRange ("Enter id of the task to edit: ", 1, 9999);
index = findTaskIndexbyId(tasks, taskCount, id);
if (index == -1) {                                  // if id not found
    printf("Task with id %d not found.\n", id);
    return;
}
printf("\nEditing Task id %d\n", id);
printf("1. Edit title\n2. Edit description\n3. Edit priority\n4. Cancel\n");
choice = readIntInRange("Choose an option to edit: ", 1,4);
switch (choice) {
  case 1: readline("Enter the new title: ",tasks[index].title, sizeof(tasks[index].title));
          printf("Title updated..\n"); break;
  case 2: readline("Enter the new description: ",tasks[index].description, sizeof(tasks[index].description));
          printf("Description updated..\n");  break;
  case 3: tasks[index].priority = readIntInRange("Enter new priority (1-5): ", 1, 5);
          printf("Priority updated..\n");  break;
  case 4: printf("Edit cancelled.\n");     break;
 }
}
```



```c
// main fuction: start point of the program
int main() {
  Task tasks[MAX_TASKS];         // array to hold tasks
int taskCount, nextId, choice, Running = 1;
  initTasks(tasks, MAX_TASKS, &taskCount, &nextId );     // initialize
  loadTasksFromFile(tasks, MAX_TASKS, &taskCount, &nextId);  // load saved tasks
printf("Welcome to the simple TO-DO list APP\n");
  while (Running) {         // main loop
  printMenu();
choice = readIntInRange("Enter your choice(1-7): ", 1,7);
  switch (choice) {                              // process user choice
      case 1: addTask(tasks, MAX_TASKS, &taskCount, &nextId); break;
      case 2: listTasks(tasks, taskCount);      break;
      case 3: markTaskAsDone(tasks, taskCount); break;
      case 4: editTask(tasks, taskCount);       break;
      case 5: deleteTask(tasks, &taskCount);    break;
      case 6: saveTasksToFile(tasks, taskCount); break;      // Manual save option
      case 7: printf("Exiting the TO-DO list app.\n"); Running = 0;    // stop loop
       break;
    }
 }
   return 0;                  //exit success
}
```

❖ This concludes the project report on

“ To-do list application ”

Thank you..

Name Rashi Banyal

Sap Id 590028073

Batch 59