Algorithms of all important functions

1) Task_struct.h

Algorithm 1 Initialize a new date instance

```
// Define the Date struct
STRUCT Date
        day: integer
        month: integer
        year: integer
END STRUCT

// Define a function to initialize a new Date instance
FUNCTION initialize_date(day: int, month: int, year: int) -> Date*
        // Allocate memory for the new Date instance
        container = ALLOCATE(Date)
        // Set the Date instance's attributes
        container.day = day
        container.month = month
        container.year = year
        RETURN container
END FUNCTION
```

Algorithm 2 Get the current local date

```
FUNCTION get_current_local_date() -> Date*
        // Get the current time
        periodic_time = CURRENT_TIME()
        // Format the time as a local time
        formatted_time = LOCAL_TIME(periodic_time)
        // Initialize a new Date instance with the formatted time's attributes
        local_date = initialize_date(
                formatted_time.day,
                //The time structure stores months from the range 0-11
                formatted_time.month + 1,
                //The time structure stores years since 1900
                formatted_time.year + 1900
        )
        // Output the day of the local_date instance
        OUTPUT local_date.day
        RETURN local_date
END FUNCTION
```

Algorithm 3 Compare two dates to figure out if they are identical

```
FUNCTION compare_dates(d1: Date*, d2: Date*) -> integer
    // Compare the years of the two Date instances
    IF d1.year != d2.year THEN
        RETURN d1.year - d2.year
    // Compare the months of the two Date instances
    ELSE IF d1.month != d2.month THEN
        RETURN d1.month - d2.month
    // Compare the days of the two Date instances
    ELSE
        // The aim is to return 0 if two dates are identical
        RETURN d1.day - d2.day
    END IF
END FUNCTION
```

Algorithm 4 Initialize a new task instance

```
STRUCT task:
    Priority: integer
    text[STD_STRING_SIZE] : character
    deadline: Date
END STRUCT

SET structure task as Task using typedef

FUNCTION initialize_task(text: const char*, priority: integer, d: Date*) -> Task*
```

```
        // Allocate memory for the new task
        SET new_task = ALLOCATE memory for Task struct
        // Copy the text to the new task
        SET new_task->text = copy text
        // Set the priority of the new task
        SET new_task->priority = priority
        // Set the deadline of the new task
        SET new_task->deadline = *d
        // Return the newly created task
        RETURN new_task
END FUNCTION
```

Algorithm 5 Read all tasks information from 2 files task.txt and done.txt

```
FUNCTION read_all_tasks_from_file(filepath: const char*, num_tasks: int*) -> Task**
        SET tasks = empty list
        SET num_tasks = 0
        OPEN file at filepath
        IF unable to open file THEN
                PRINT "Error: could not open file"
                RETURN NULL
        END IF
        FOR each line IN file
                SET pr, text, deadline = extract values from line using regular expressions
                SET priority = convert pr string to integer
                SET day, month, year = convert deadline string to day, month, year
        integers
                SET new_task = initialize new Task with text, priority, and date
                ADD new_task to tasks list
                INCREMENT num_tasks by 1
        END FOR
        CLOSE file
        IF num_tasks is zero THEN
                FREE memory allocated for tasks list
                RETURN NULL
        END IF
        RESIZE tasks list to size num_tasks
        RETURN tasks
END FUNCTION
```

Algorithm 6 Get all tasks information stored in structures and return them in a task string under the designed format

```
// Get information of all tasks
FUNCTION get_printable_task(task: Task) -> char*
        SET size = calculate size of task string using snprintf
        ALLOCATE memory for task string with size
        FORMAT task string using snprintf
        RETURN task string
END FUNCTION

// Get information of completed tasks
FUNCTION get_printable_complete_task(task: Task) -> char*
        SET size = calculate size of completed task string using snprintf
        ALLOCATE memory for completed task string with size
        FORMAT completed task string using snprintf
        RETURN completed task string
END FUNCTION
```

2) Task_add.h

Algorithm 1 Add a new task to the task list file

```
FUNCTION task_add(int task_priority, char *task_text, Date *d, const char *filepath):
        SET num_tasks = 0
        task_list = read_all_tasks_from_file(filepath, num_tasks)

        // Check if the new task already exists in the file
        FOR i = 0 to num_tasks - 1 DO
            IF task_list[i]->text == task_text AND compare_dates(&task_list[i]->deadline, d) == 0 THEN
                PRINT "Task already exists:"
                PRINT get_printable_task(task_list[i])
                RETURN // Exit function if task already exists
            END IF
        END FOR

        // Initialize new task and write to file
        new_task = initialize_task(task_text, task_priority, d)
        f = OPEN filepath in append mode
        WRITE get_printable_task(new_task) to f
        CLOSE f

        // Print added task and write to file
        PRINT "Added task:"
        task_str = get_printable_task(new_task)
        PRINT task_str
END FUNCTION
```

3) Task_ls.h

Algorithm 1 Display list of incomplete tasks in order of index (base on the order you add)

```
FUNCTION task_ls()
    DECLARE num_tasks AS integer
    DECLARE task_list AS Task pointer
    SET task_list TO read_all_tasks_from_file("task.txt", &num_tasks)

    // if no tasks found in file, print message and return
    IF task_list == NULL THEN
        PRINT "No tasks to do."
        RETURN
    END IF
```

```
        // display the list of incomplete tasks in order of tasks_index
        PRINT "\nList of incomplete tasks in order of tasks_index you have added:\n"
        FOR i FROM 0 TO num_tasks-1 DO
                PRINT i+1, ". ", get_printable_task(*(task_list + i))
        END FOR
        PRINT "The number of incomplete tasks: ", num_tasks
END FUNCTION
```

Algorithm 2 Display list of incomplete tasks in order of priority

```
FUNCTION task_ls_priority()
        DECLARE num_tasks as integer = 0
        DECLARE task_list as Task pointer array
        DECLARE f as FILE pointer

        SET task_list to read_all_tasks_from_file("task.txt", num_tasks)

        IF task_list is NULL THEN
                PRINT "No tasks to do."
                RETURN
        END IF

        // sort the tasks in the list by priority using qsort()
        CALL qsort(task_list, num_tasks, sizeof(Task*), compare_tasks_priority)

        PRINT "List of incomplete tasks in priority order:"

        FOR i FROM 0 TO num_tasks-1
                PRINT i+1, ".", get_printable_task(task_list[i])
        END FOR

        PRINT "The number of incomplete tasks: ", num_tasks
 END FUNCTION
```

Algorithm 3 Display list of incomplete tasks in order of deadline

```
FUNCTION task_ls_deadline()
        DECLARE num_tasks AS Integer
        DECLARE task_list AS Array of Task*

        // Read the list of tasks from the file
        SET task_list TO read_all_tasks_from_file("task.txt", &num_tasks)
        IF task_list is NULL THEN
        // If there are no tasks, print a message and return
                PRINT "No tasks to do."
                RETURN
        END IF

        // Sort the tasks by deadline using qsort
        qsort(task_list, num_tasks, sizeof(Task*), compare_tasks_by_deadline)

        // Print the sorted tasks to the console
        PRINT "\nList of incomplete tasks in deadline order:"
        FOR i FROM 0 TO num_tasks-1 DO
                PRINT i+1, ". ", get_printable_task(*(task_list + i))
        END FOR
        PRINT "The number of incomplete tasks: ", num_tasks

END FUNCTION
```

4) Task_del.h

Algorithm 1 Delete a task in the task list

```
FUNCTION task_del(int task_index, const char *filepath)
        SET num_tasks = 0
        SET task_list = read_all_tasks_from_file(filepath, &num_tasks)

        // Check if task list is empty
```

```
IF task_list == NULL OR num_tasks == 0 THEN
    PRINT "No task to do"
    RETURN NULL
END IF

// Check if the task want to delete exists in task list
IF task_index < 0 OR task_index >= num_tasks THEN
    PRINT "Index " + task_index + " does not exist"
    RETURN NULL
END IF

SET task_to_delete = task_list[task_index-1]
// Start deleting the task at index task_index by pushing the tasks in the indexes
behind go forward 1 index
FOR i = task_index - 1 TO num_tasks - 2
    SET task_list[i] = task_list[i + 1]
END FOR
PRINT "Deleted task at index " + task_index + ":"
PRINT get_printable_task(task_to_delete)
free(task_to_delete)
SET task_list[num_tasks - 1] = NULL
SET num_tasks = num_tasks - 1
PRINT "List of incomplete task after deleting: "
FOR i = 0 TO num_tasks - 1
    PRINT i+1 + ". "
    PRINT get_printable_task(task_list[i])
END FOR
SET f = fopen(filepath, "w")
IF f == NULL THEN
    PRINT "Error: Could not open file " + filepath
ELSE
    FOR i = 0 TO num_tasks - 1
        fprintf(f, "%s", get_printable_task(task_list[i]))
        END FOR
        fclose(f)
    END IF
END FUNCTION
```

5) Task_ls_remind.h

Algorithm 1 Countdown how many days left until the deadlines

```
FUNCTION get_days_left(d: Date) -> int
    t <- current time in seconds
    local_time <- convert t to local time struct
    curr_day <- day of the month from local_time
    curr_month <- month of the year from local_time plus 1
    curr_year <- year from local_time plus 1900

    days_left <- (d.year - curr_year) * 365 + (d.month - curr_month) * 30 + (d.day - curr_day)
    RETURN days_left
END FUNCTION
```

Algorithm 2 Tasks remind for users

```
FUNCTION task_ls_remind():
    // Get current local time
    SET t = GET_CURRENT_TIME()
    SET local_time = CONVERT_TO_LOCAL_TIME(t)

    // Initialize task count variables
    SET uncompleted_today_count = 0
    SET uncompleted_future_count = 0
    SET uncompleted_past_count = 0

    // Display current local date
    PRINT "Current local date: " + FORMAT_LOCAL_TIME(local_time)
```

```
// Read tasks from file
SET num_tasks = 0
SET task_list = READ_ALL_TASKS_FROM_FILE("task.txt", num_tasks)

// If no tasks, print message and return
IF task_list == NULL THEN
    PRINT "No tasks to do."
    RETURN
END IF

// Sort tasks by priority
SORT(task_list, num_tasks, COMPARE_TASKS_PRIORITY)

// Display tasks due today
PRINT "\nToday tasks: "
FOR i FROM 0 TO num_tasks DO
    SET days_left = GET_DAYS_LEFT(task_list[i].deadline)
    IF days_left == 0 THEN
        PRINT (i+1) + ". " + GET_PRINTABLE_TASK(task_list[i])
        INCREMENT uncompleted_today_count BY 1
    END IF
END FOR
IF uncompleted_today_count == 0 THEN
    PRINT "No tasks have deadline today."
END IF
PRINT "The number of incomplete tasks today: " + uncompleted_today_count
// Display future tasks
PRINT "\nFuture tasks: "
FOR i FROM 0 TO num_tasks DO
    SET days_left = GET_DAYS_LEFT(task_list[i].deadline)
    IF days_left > 0 THEN
```

```
                PRINT GET_PRINTABLE_TASK(task_list[i]) + "Due in " +
                days_left + " day(s)."
                INCREMENT uncompleted_future_count BY 1
            END IF
        END FOR
        IF uncompleted_future_count == 0 THEN
            PRINT "No tasks have deadline in the future."
        END IF
        PRINT "The number of incomplete tasks in the future: " +
uncompleted_future_count

        // Display past tasks
        PRINT "\nPast tasks: "
        FOR i FROM 0 TO num_tasks DO
            SET days_left = GET_DAYS_LEFT(task_list[i].deadline)
            IF days_left < 0 THEN
                PRINT GET_PRINTABLE_TASK(task_list[i])
                INCREMENT uncompleted_past_count BY 1
            END IF
        END FOR
        PRINT "The number of incomplete tasks in the past: " + uncompleted_past_count

END FUNCTION
```

6) Task_done.h

Algorithm 1 User marks done tasks at chosen indexes

```
FUNCTION task_done(task_index)
        SET num_tasks TO 0
        SET task_list TO read_all_tasks_from_file("task.txt", num_tasks)
        IF task_list IS NULL OR num_tasks IS 0 THEN
            PRINT "No task to do"
            RETURN
        END IF
```

```
    IF task_index < 0 OR task_index >= num_tasks THEN
        PRINT "Index " + task_index + " does not exist"
    RETURN
    END IF
    SET done_task TO task_list[task_index-1]
    FOR i FROM task_index - 1 TO num_tasks - 2 DO
        SET task_list[i] TO task_list[i + 1]
    END FOR
    PRINT "Completed task at index " + task_index + ":\n" +
    get_printable_task(done_task)
END FUNCTION
```

7) Task_report.h

Algorithm 1 Report tasks including undone tasks and done tasks

```
FUNCTION task_report()
    DECLARE num_tasks AS INTEGER
    DECLARE task_list AS ARRAY OF Task*

    // Print list of incomplete tasks
    SET task_list TO read_all_tasks_from_file("task.txt", &num_tasks)
    IF task_list IS NULL OR num_tasks IS 0
        PRINT "No tasks to do."
    END IF
    PRINT "Pending: ", num_tasks
    FOR i FROM 0 TO num_tasks - 1
        PRINT i+1, ". ", get_printable_task(*(task_list + i))
    END FOR

    // Print list of completed tasks
    DECLARE num_tasks_completed AS INTEGER
    DECLARE task_list_completed AS ARRAY OF Task*
```

```
        SET task_list_completed TO read_all_tasks_from_file("done.txt", &num_tasks_completed)
        IF task_list_completed IS NULL
                PRINT "No tasks completed."
        END IF
        PRINT "Done: ", num_tasks_completed
        FOR i FROM 0 TO num_tasks_completed - 1
                PRINT i+1, ". ", get_printable_complete_task(*(task_list_completed + i))
        END FOR
END FUNCTION
```

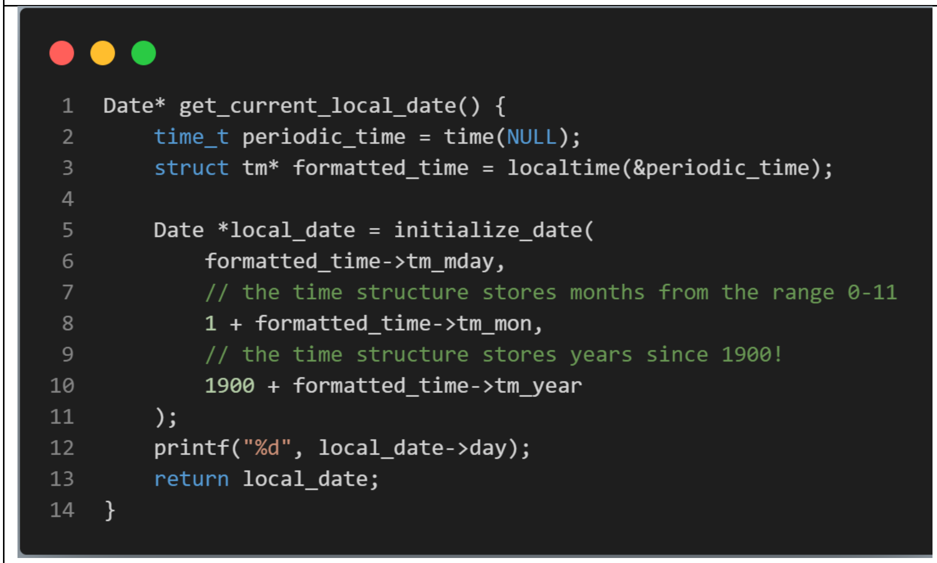
3.2 Implementation:

1) Task_struct.h

Algorithm 1 Initialize a new date instance

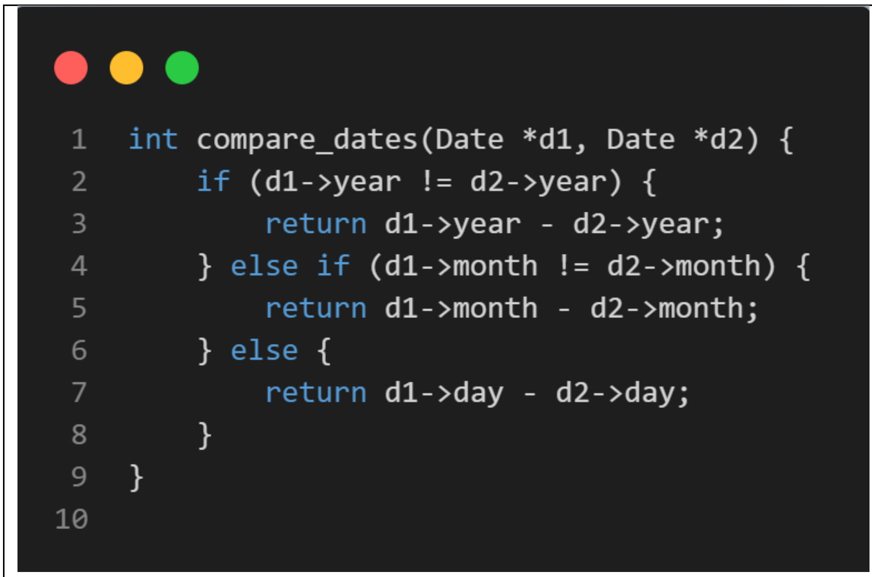
} [redacted]

Algorithm 2 Get the current local date



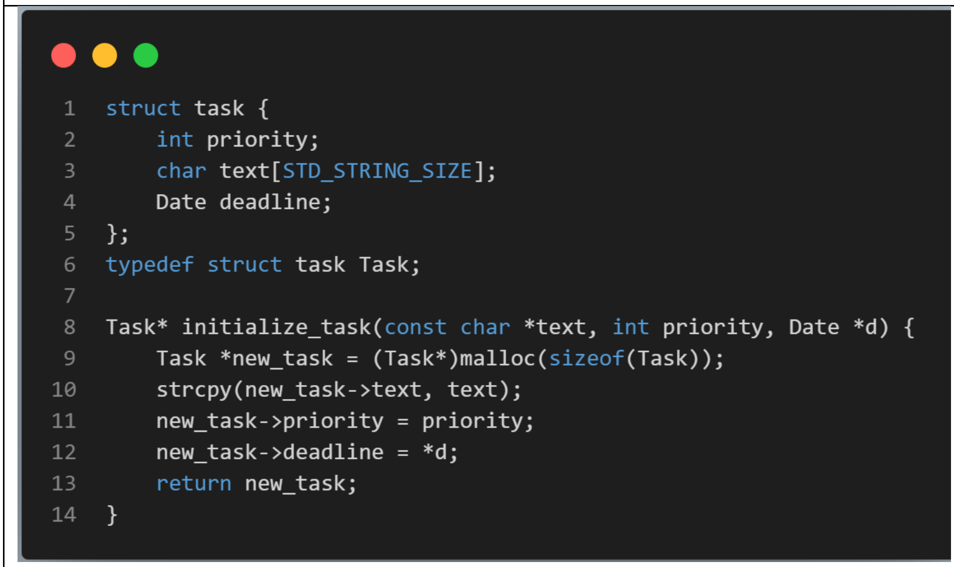
```
Date* get_current_local_date() {
    time_t periodic_time = time(NULL);
    struct tm* formatted_time = localtime(&periodic_time);

    Date *local_date = initialize_date(
        formatted_time->tm_mday,
        // the time structure stores months from the range 0-11
        1 + formatted_time->tm_mon,
        // the time structure stores years since 1900!
        1900 + formatted_time->tm_year
    );
    printf("%d", local_date->day);
    return local_date;
}
```

Algorithm 3 Compare two dates to figure out if they are identical



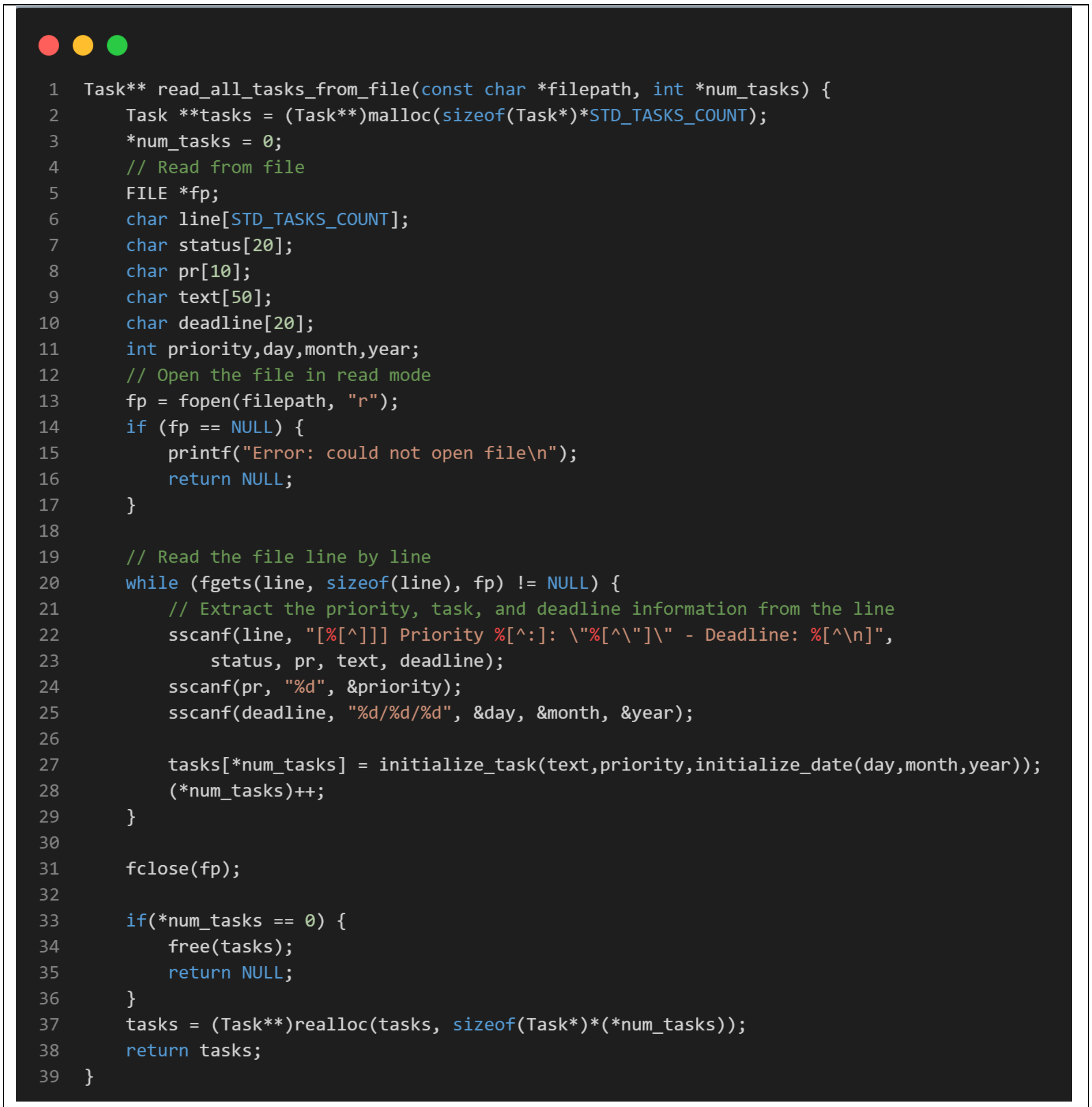
```c
int compare_dates(Date *d1, Date *d2) {
    if (d1->year != d2->year) {
        return d1->year - d2->year;
    } else if (d1->month != d2->month) {
        return d1->month - d2->month;
    } else {
        return d1->day - d2->day;
    }
}

```

Algorithm 4 Initialize a new task instance



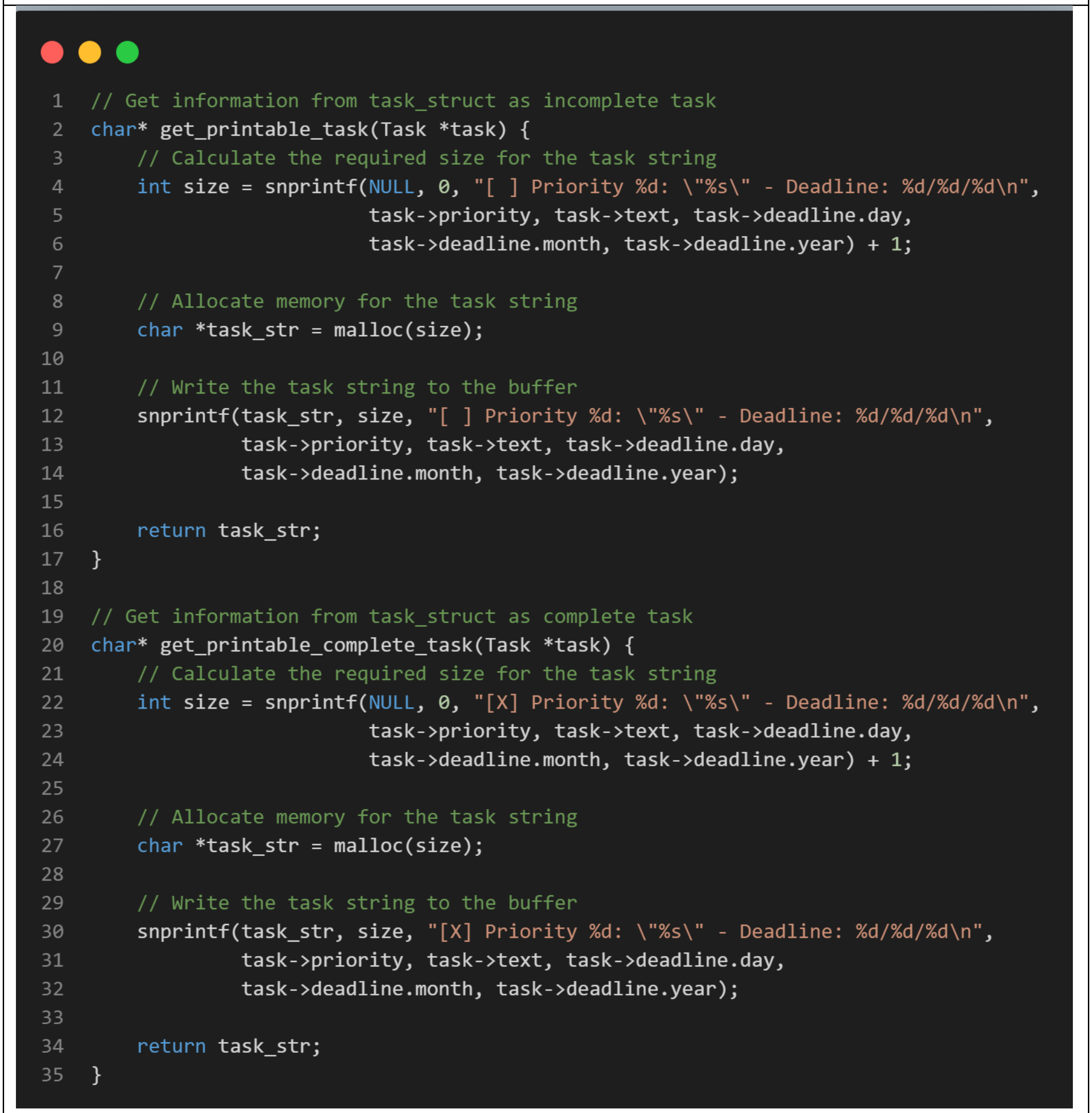
```c
struct task {
    int priority;
    char text[STD_STRING_SIZE];
    Date deadline;
};
typedef struct task Task;

Task* initialize_task(const char *text, int priority, Date *d) {
    Task *new_task = (Task*)malloc(sizeof(Task));
    strcpy(new_task->text, text);
    new_task->priority = priority;
    new_task->deadline = *d;
    return new_task;
}
```

Algorithm 5 Read all tasks information from 2 files task.txt and done.txt



```c
Task** read_all_tasks_from_file(const char *filepath, int *num_tasks) {
    Task **tasks = (Task**)malloc(sizeof(Task*)*STD_TASKS_COUNT);
    *num_tasks = 0;
    // Read from file
    FILE *fp;
    char line[STD_TASKS_COUNT];
    char status[20];
    char pr[10];
    char text[50];
    char deadline[20];
    int priority,day,month,year;
    // Open the file in read mode
    fp = fopen(filepath, "r");
    if (fp == NULL) {
        printf("Error: could not open file\n");
        return NULL;
    }

    // Read the file line by line
    while (fgets(line, sizeof(line), fp) != NULL) {
        // Extract the priority, task, and deadline information from the line
        sscanf(line, "[%[^]]] Priority %[^:]: \"%[^\"]\" - Deadline: %[^\n]",
            status, pr, text, deadline);
        sscanf(pr, "%d", &priority);
        sscanf(deadline, "%d/%d/%d", &day, &month, &year);

        tasks[*num_tasks] = initialize_task(text,priority,initialize_date(day,month,year));
        (*num_tasks)++;
    }

    fclose(fp);

    if(*num_tasks == 0) {
        free(tasks);
        return NULL;
    }
    tasks = (Task**)realloc(tasks, sizeof(Task*)*(*num_tasks));
    return tasks;
}
```

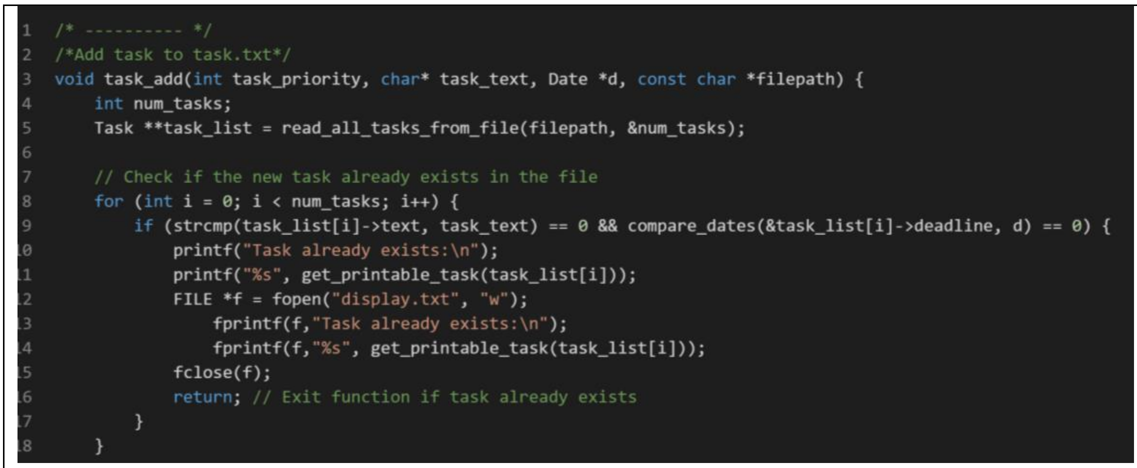
Algorithm 6 Get all tasks information stored in structures and return them in a task string under the designed format



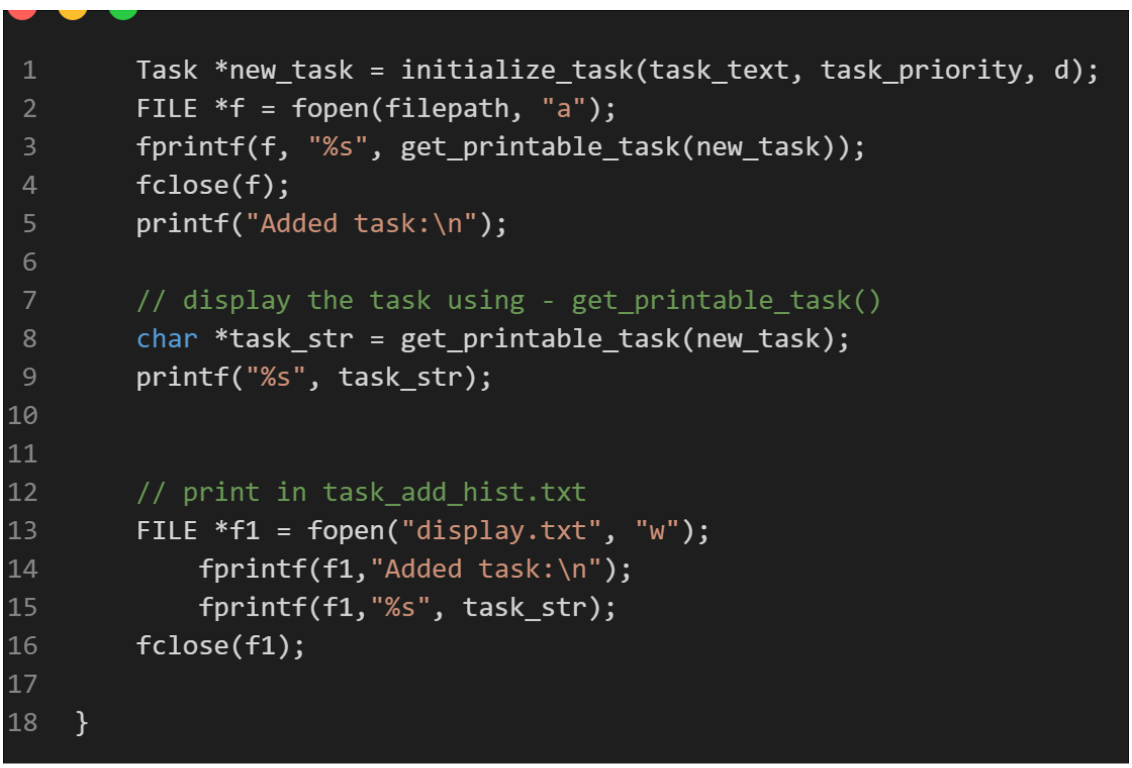
```c
// Get information from task_struct as incomplete task
char* get_printable_task(Task *task) {
    // Calculate the required size for the task string
    int size = snprintf(NULL, 0, "[ ] Priority %d: \"%s\" - Deadline: %d/%d/%d\n",
                        task->priority, task->text, task->deadline.day,
                        task->deadline.month, task->deadline.year) + 1;

    // Allocate memory for the task string
    char *task_str = malloc(size);

    // Write the task string to the buffer
    snprintf(task_str, size, "[ ] Priority %d: \"%s\" - Deadline: %d/%d/%d\n",
             task->priority, task->text, task->deadline.day,
             task->deadline.month, task->deadline.year);

    return task_str;
}

// Get information from task_struct as complete task
char* get_printable_complete_task(Task *task) {
    // Calculate the required size for the task string
    int size = snprintf(NULL, 0, "[X] Priority %d: \"%s\" - Deadline: %d/%d/%d\n",
                        task->priority, task->text, task->deadline.day,
                        task->deadline.month, task->deadline.year) + 1;

    // Allocate memory for the task string
    char *task_str = malloc(size);

    // Write the task string to the buffer
    snprintf(task_str, size, "[X] Priority %d: \"%s\" - Deadline: %d/%d/%d\n",
             task->priority, task->text, task->deadline.day,
             task->deadline.month, task->deadline.year);

    return task_str;
}
```

2) Task_add.h

Algorithm 1 Add a new task to the task list file



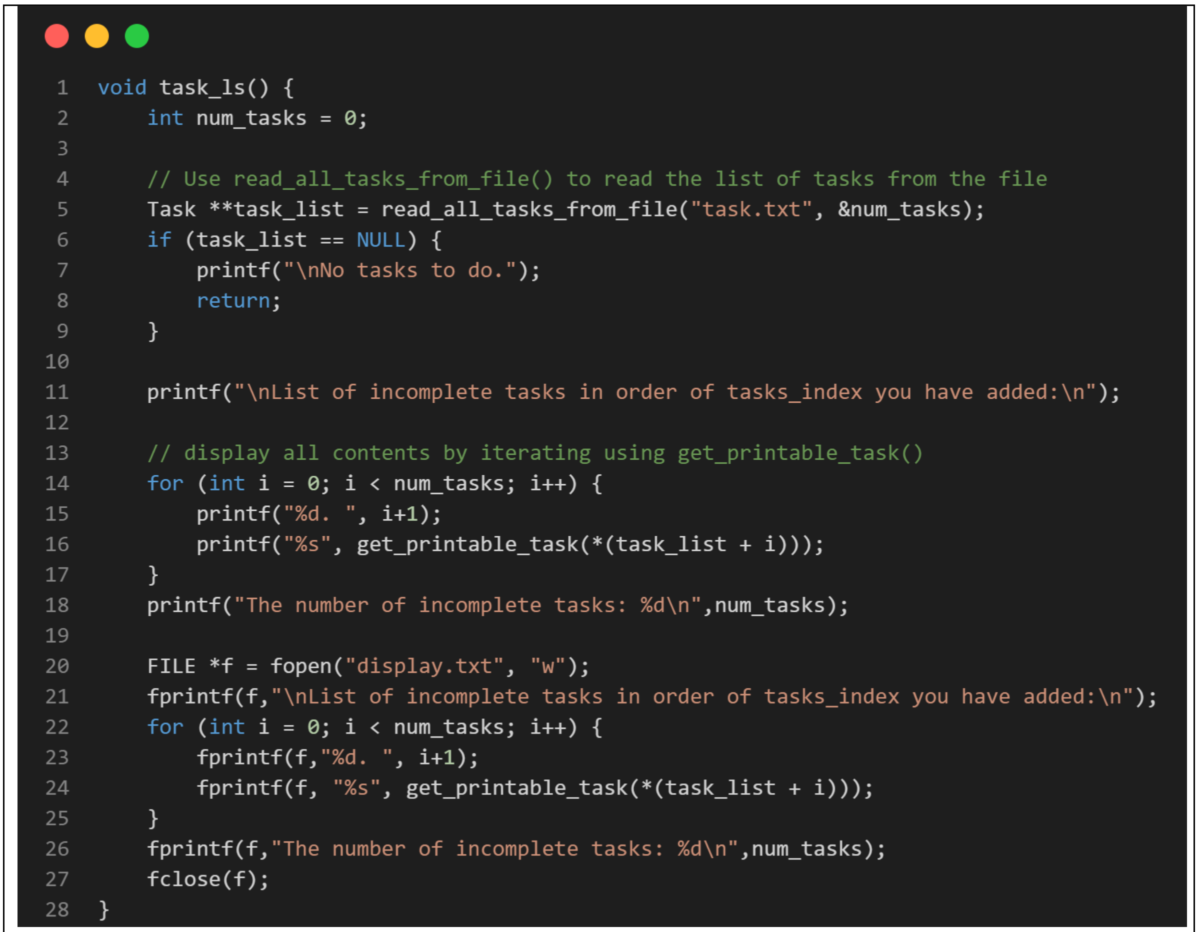
```c
/* ---------- */
/*Add task to task.txt*/
void task_add(int task_priority, char* task_text, Date *d, const char *filepath) {
    int num_tasks;
    Task **task_list = read_all_tasks_from_file(filepath, &num_tasks);

    // Check if the new task already exists in the file
    for (int i = 0; i < num_tasks; i++) {
        if (strcmp(task_list[i]->text, task_text) == 0 && compare_dates(&task_list[i]->deadline, d) == 0) {
            printf("Task already exists:\n");
            printf("%s", get_printable_task(task_list[i]));
            FILE *f = fopen("display.txt", "w");
                fprintf(f,"Task already exists:\n");
                fprintf(f,"%s", get_printable_task(task_list[i]));
            fclose(f);
            return; // Exit function if task already exists
        }
    }
```



```c
    Task *new_task = initialize_task(task_text, task_priority, d);
    FILE *f = fopen(filepath, "a");
    fprintf(f, "%s", get_printable_task(new_task));
    fclose(f);
    printf("Added task:\n");

    // display the task using - get_printable_task()
    char *task_str = get_printable_task(new_task);
    printf("%s", task_str);


    // print in task_add_hist.txt
    FILE *f1 = fopen("display.txt", "w");
        fprintf(f1,"Added task:\n");
        fprintf(f1,"%s", task_str);
    fclose(f1);

}
```

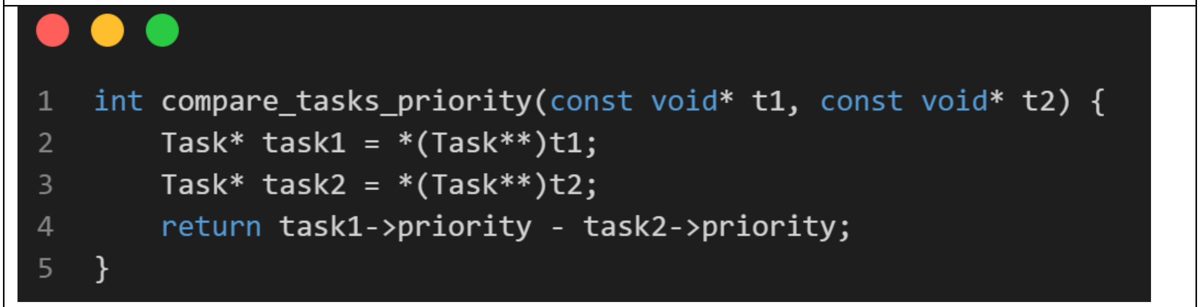
3) Task_ls.h

Algorithm 1 Display list of incomplete tasks in order of index (base on the order you add)

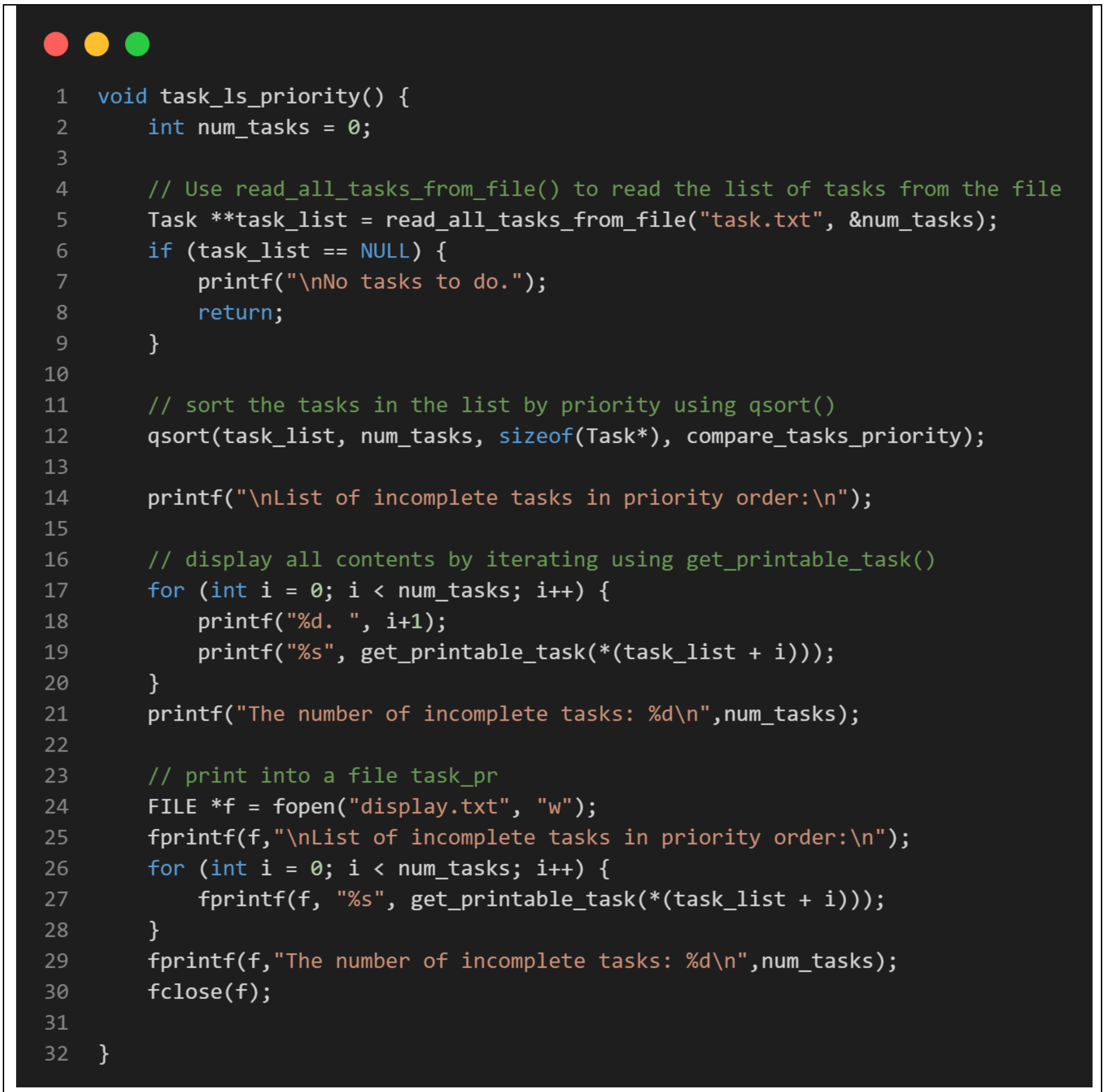


```c
void task_ls() {
    int num_tasks = 0;

    // Use read_all_tasks_from_file() to read the list of tasks from the file
    Task **task_list = read_all_tasks_from_file("task.txt", &num_tasks);
    if (task_list == NULL) {
        printf("\nNo tasks to do.");
        return;
    }

    printf("\nList of incomplete tasks in order of tasks_index you have added:\n");

    // display all contents by iterating using get_printable_task()
    for (int i = 0; i < num_tasks; i++) {
        printf("%d. ", i+1);
        printf("%s", get_printable_task(*(task_list + i)));
    }
    printf("The number of incomplete tasks: %d\n",num_tasks);

    FILE *f = fopen("display.txt", "w");
    fprintf(f,"\nList of incomplete tasks in order of tasks_index you have added:\n");
    for (int i = 0; i < num_tasks; i++) {
        fprintf(f,"%d. ", i+1);
        fprintf(f, "%s", get_printable_task(*(task_list + i)));
    }
    fprintf(f,"The number of incomplete tasks: %d\n",num_tasks);
    fclose(f);
}
```

Algorithm 2 Display list of incomplete tasks in order of priority

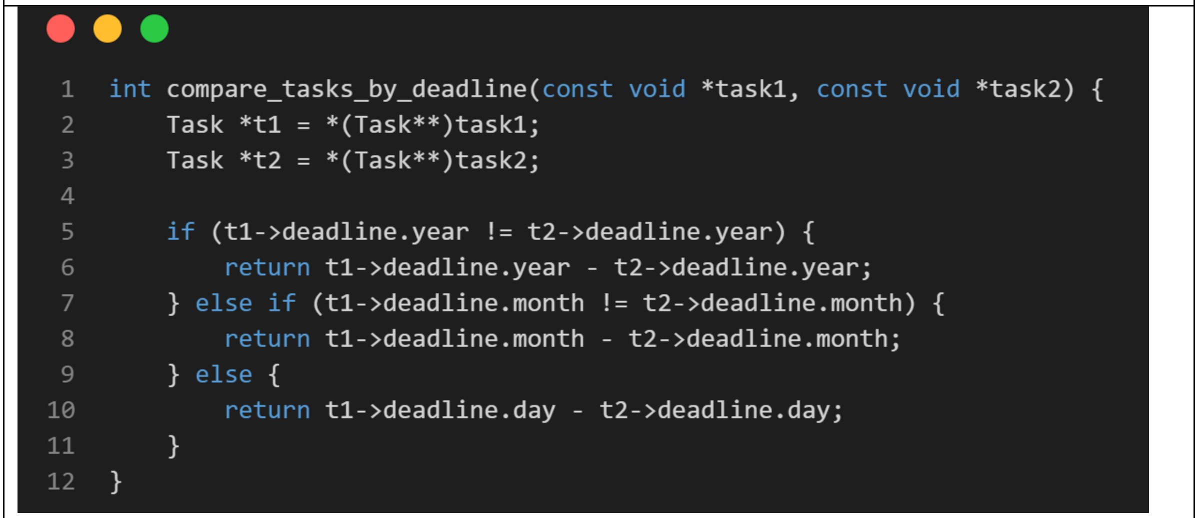


```c
int compare_tasks_priority(const void* t1, const void* t2) {
    Task* task1 = *(Task**)t1;
    Task* task2 = *(Task**)t2;
    return task1->priority - task2->priority;
}
```

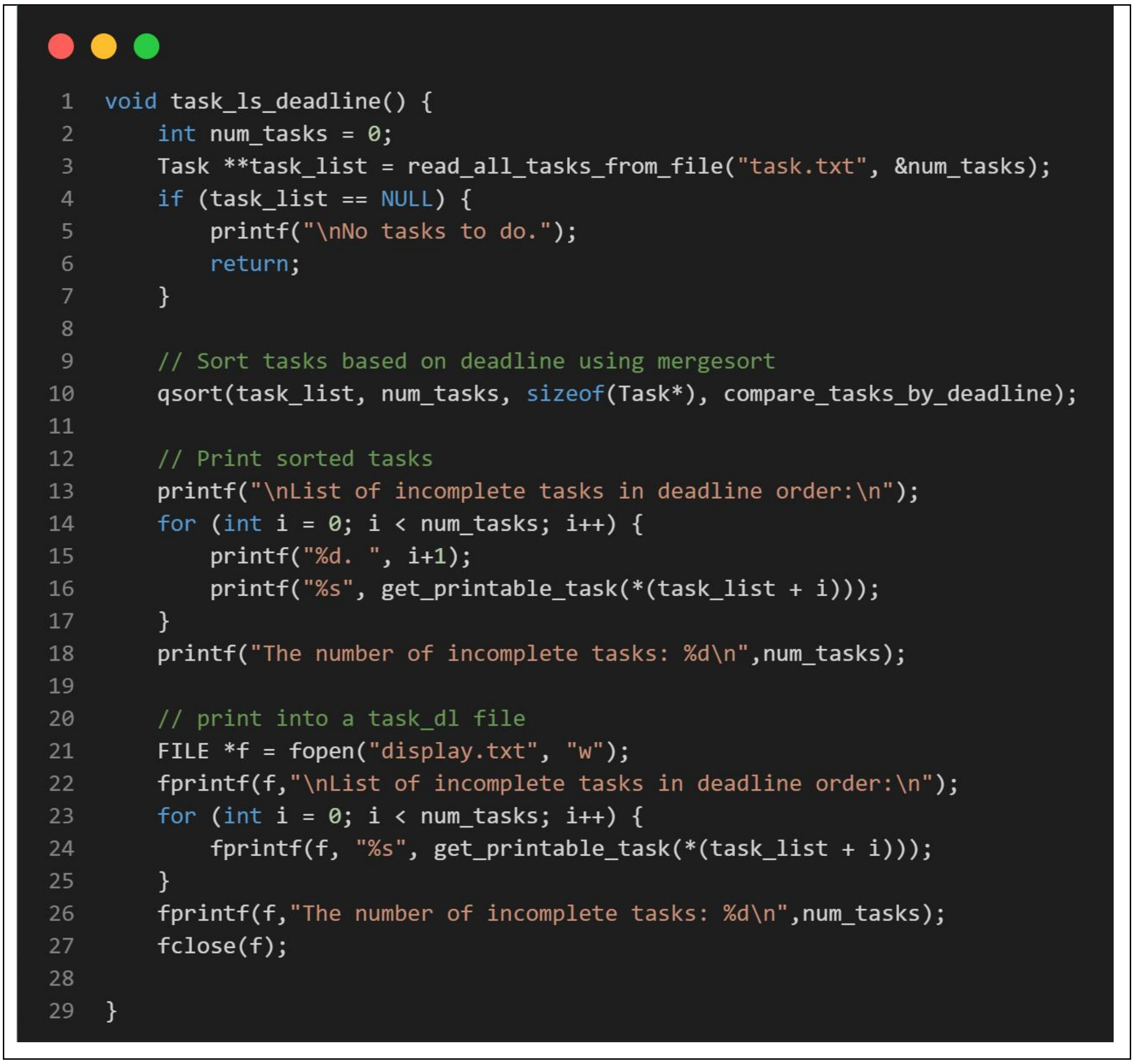


```c
void task_ls_priority() {
    int num_tasks = 0;

    // Use read_all_tasks_from_file() to read the list of tasks from the file
    Task **task_list = read_all_tasks_from_file("task.txt", &num_tasks);
    if (task_list == NULL) {
        printf("\nNo tasks to do.");
        return;
    }

    // sort the tasks in the list by priority using qsort()
    qsort(task_list, num_tasks, sizeof(Task*), compare_tasks_priority);

    printf("\nList of incomplete tasks in priority order:\n");

    // display all contents by iterating using get_printable_task()
    for (int i = 0; i < num_tasks; i++) {
        printf("%d. ", i+1);
        printf("%s", get_printable_task(*(task_list + i)));
    }
    printf("The number of incomplete tasks: %d\n",num_tasks);

    // print into a file task_pr
    FILE *f = fopen("display.txt", "w");
    fprintf(f,"\nList of incomplete tasks in priority order:\n");
    for (int i = 0; i < num_tasks; i++) {
        fprintf(f, "%s", get_printable_task(*(task_list + i)));
    }
    fprintf(f,"The number of incomplete tasks: %d\n",num_tasks);
    fclose(f);

}
```

Algorithm 3 Display list of incomplete tasks in order of deadline



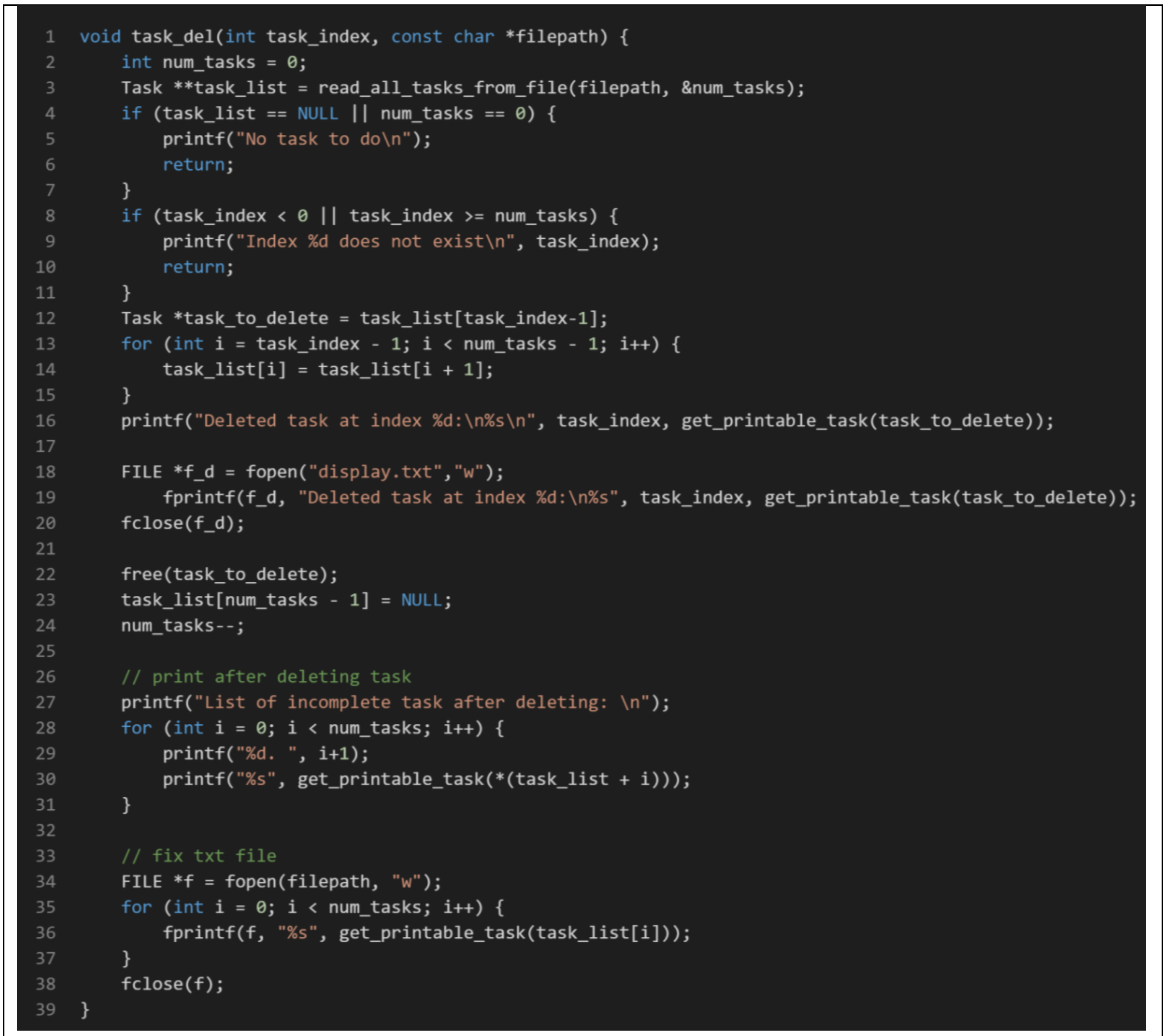
```c
int compare_tasks_by_deadline(const void *task1, const void *task2) {
    Task *t1 = *(Task**)task1;
    Task *t2 = *(Task**)task2;

    if (t1->deadline.year != t2->deadline.year) {
        return t1->deadline.year - t2->deadline.year;
    } else if (t1->deadline.month != t2->deadline.month) {
        return t1->deadline.month - t2->deadline.month;
    } else {
        return t1->deadline.day - t2->deadline.day;
    }
}
```



```c
void task_ls_deadline() {
    int num_tasks = 0;
    Task **task_list = read_all_tasks_from_file("task.txt", &num_tasks);
    if (task_list == NULL) {
        printf("\nNo tasks to do.");
        return;
    }

    // Sort tasks based on deadline using mergesort
    qsort(task_list, num_tasks, sizeof(Task*), compare_tasks_by_deadline);

    // Print sorted tasks
    printf("\nList of incomplete tasks in deadline order:\n");
    for (int i = 0; i < num_tasks; i++) {
        printf("%d. ", i+1);
        printf("%s", get_printable_task(*(task_list + i)));
    }
    printf("The number of incomplete tasks: %d\n",num_tasks);

    // print into a task_dl file
    FILE *f = fopen("display.txt", "w");
    fprintf(f,"\nList of incomplete tasks in deadline order:\n");
    for (int i = 0; i < num_tasks; i++) {
        fprintf(f, "%s", get_printable_task(*(task_list + i)));
    }
    fprintf(f,"The number of incomplete tasks: %d\n",num_tasks);
    fclose(f);

}
```

4) Task_del.h

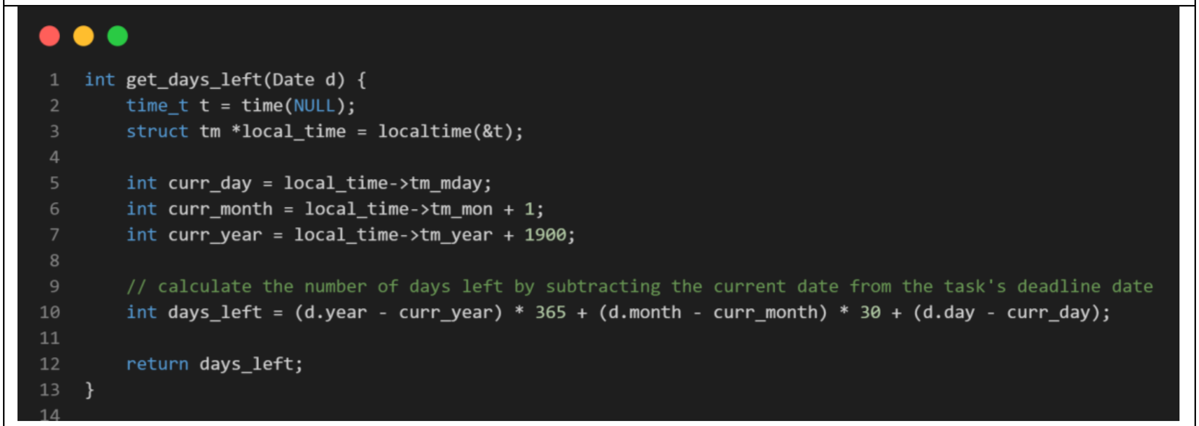
Algorithm 1 Delete a task in the task list



```c
void task_del(int task_index, const char *filepath) {
    int num_tasks = 0;
    Task **task_list = read_all_tasks_from_file(filepath, &num_tasks);
    if (task_list == NULL || num_tasks == 0) {
        printf("No task to do\n");
        return;
    }
    if (task_index < 0 || task_index >= num_tasks) {
        printf("Index %d does not exist\n", task_index);
        return;
    }
    Task *task_to_delete = task_list[task_index-1];
    for (int i = task_index - 1; i < num_tasks - 1; i++) {
        task_list[i] = task_list[i + 1];
    }
    printf("Deleted task at index %d:\n%s\n", task_index, get_printable_task(task_to_delete));

    FILE *f_d = fopen("display.txt","w");
        fprintf(f_d, "Deleted task at index %d:\n%s", task_index, get_printable_task(task_to_delete));
    fclose(f_d);

    free(task_to_delete);
    task_list[num_tasks - 1] = NULL;
    num_tasks--;

    // print after deleting task
    printf("List of incomplete task after deleting: \n");
    for (int i = 0; i < num_tasks; i++) {
        printf("%d. ", i+1);
        printf("%s", get_printable_task(*(task_list + i)));
    }

    // fix txt file
    FILE *f = fopen(filepath, "w");
    for (int i = 0; i < num_tasks; i++) {
        fprintf(f, "%s", get_printable_task(task_list[i]));
    }
    fclose(f);
}
```

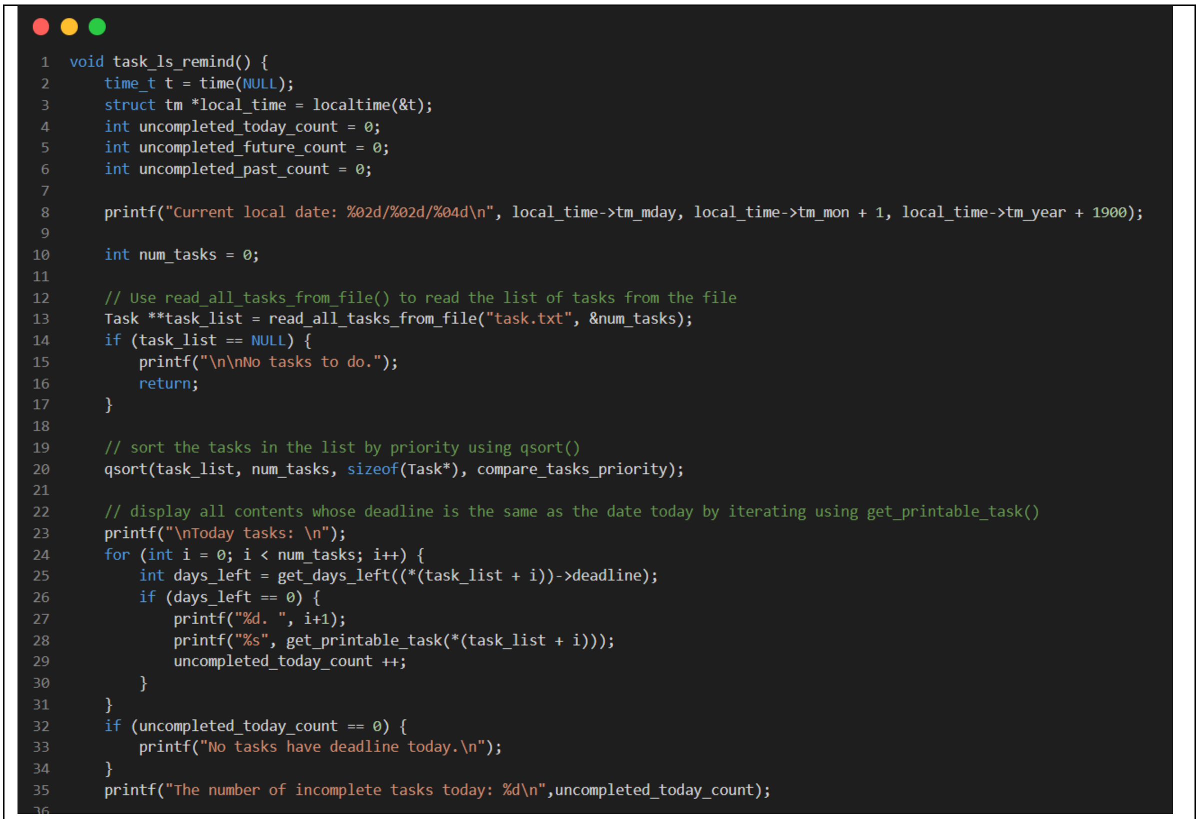
5) Task_ls_remind.h

Algorithm 1 Countdown how many days left until the deadlines

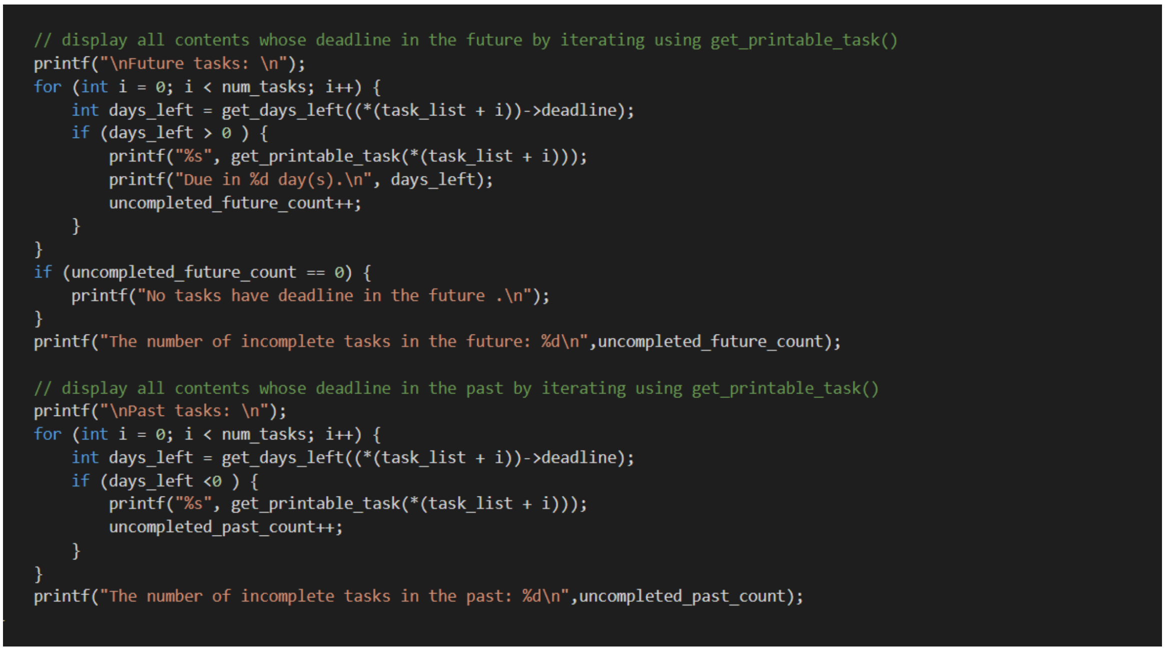


```c
int get_days_left(Date d) {
    time_t t = time(NULL);
    struct tm *local_time = localtime(&t);

    int curr_day = local_time->tm_mday;
    int curr_month = local_time->tm_mon + 1;
    int curr_year = local_time->tm_year + 1900;

    // calculate the number of days left by subtracting the current date from the task's deadline date
    int days_left = (d.year - curr_year) * 365 + (d.month - curr_month) * 30 + (d.day - curr_day);

    return days_left;
}
```

Algorithm 2 Tasks remind for users



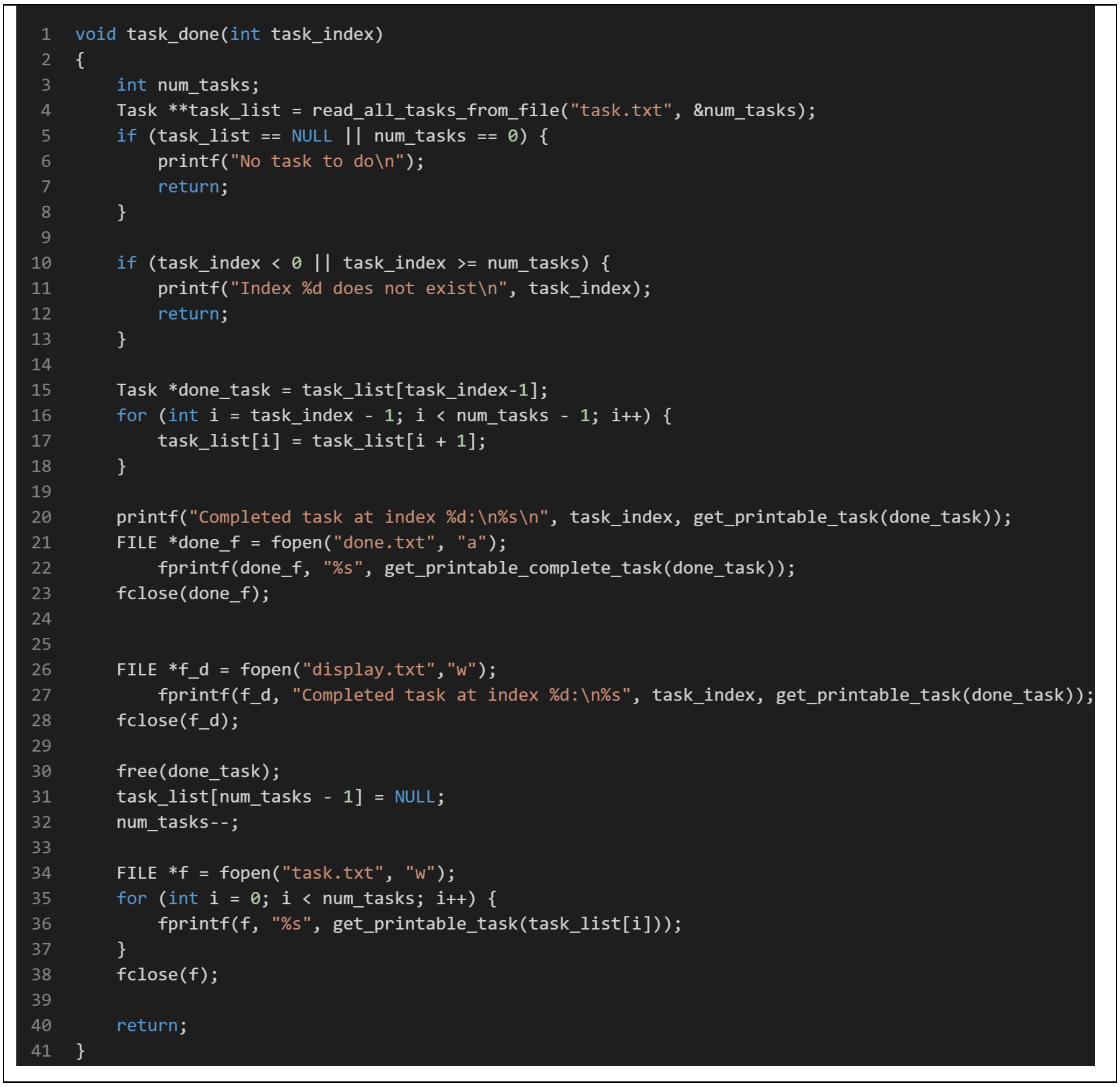
```c
void task_ls_remind() {
    time_t t = time(NULL);
    struct tm *local_time = localtime(&t);
    int uncompleted_today_count = 0;
    int uncompleted_future_count = 0;
    int uncompleted_past_count = 0;

    printf("Current local date: %02d/%02d/%04d\n", local_time->tm_mday, local_time->tm_mon + 1, local_time->tm_year + 1900);

    int num_tasks = 0;

    // Use read_all_tasks_from_file() to read the list of tasks from the file
    Task **task_list = read_all_tasks_from_file("task.txt", &num_tasks);
    if (task_list == NULL) {
        printf("\n\nNo tasks to do.");
        return;
    }

    // sort the tasks in the list by priority using qsort()
    qsort(task_list, num_tasks, sizeof(Task*), compare_tasks_priority);

    // display all contents whose deadline is the same as the date today by iterating using get_printable_task()
    printf("\nToday tasks: \n");
    for (int i = 0; i < num_tasks; i++) {
        int days_left = get_days_left((*(task_list + i))->deadline);
        if (days_left == 0) {
            printf("%d. ", i+1);
            printf("%s", get_printable_task(*(task_list + i)));
            uncompleted_today_count ++;
        }
    }
    if (uncompleted_today_count == 0) {
        printf("No tasks have deadline today.\n");
    }
    printf("The number of incomplete tasks today: %d\n",uncompleted_today_count);
```



```c
// display all contents whose deadline in the future by iterating using get_printable_task()
printf("\nFuture tasks: \n");
for (int i = 0; i < num_tasks; i++) {
    int days_left = get_days_left((*(task_list + i))->deadline);
    if (days_left > 0 ) {
        printf("%s", get_printable_task(*(task_list + i)));
        printf("Due in %d day(s).\n", days_left);
        uncompleted_future_count++;
    }
}
if (uncompleted_future_count == 0) {
    printf("No tasks have deadline in the future .\n");
}
printf("The number of incomplete tasks in the future: %d\n",uncompleted_future_count);

// display all contents whose deadline in the past by iterating using get_printable_task()
printf("\nPast tasks: \n");
for (int i = 0; i < num_tasks; i++) {
    int days_left = get_days_left((*(task_list + i))->deadline);
    if (days_left <0 ) {
        printf("%s", get_printable_task(*(task_list + i)));
        uncompleted_past_count++;
    }
}
printf("The number of incomplete tasks in the past: %d\n",uncompleted_past_count);
```

6) Task_done.h

Algorithm 1 User marks done tasks at chosen indexes



```c
void task_done(int task_index)
{
    int num_tasks;
    Task **task_list = read_all_tasks_from_file("task.txt", &num_tasks);
    if (task_list == NULL || num_tasks == 0) {
        printf("No task to do\n");
        return;
    }

    if (task_index < 0 || task_index >= num_tasks) {
        printf("Index %d does not exist\n", task_index);
        return;
    }

    Task *done_task = task_list[task_index-1];
    for (int i = task_index - 1; i < num_tasks - 1; i++) {
        task_list[i] = task_list[i + 1];
    }

    printf("Completed task at index %d:\n%s\n", task_index, get_printable_task(done_task));
    FILE *done_f = fopen("done.txt", "a");
        fprintf(done_f, "%s", get_printable_complete_task(done_task));
    fclose(done_f);


    FILE *f_d = fopen("display.txt","w");
        fprintf(f_d, "Completed task at index %d:\n%s", task_index, get_printable_task(done_task));
    fclose(f_d);

    free(done_task);
    task_list[num_tasks - 1] = NULL;
    num_tasks--;

    FILE *f = fopen("task.txt", "w");
    for (int i = 0; i < num_tasks; i++) {
        fprintf(f, "%s", get_printable_task(task_list[i]));
    }
    fclose(f);

    return;
}
```

Our program: https://github.com/Akirahai/Task_management_tool_Compsys

4. TABLE OF CONTENT

Task Management Tool Report

5. REFERENCE LIST

[1] ELEC 2030 : Computer Systems Programming. (2023). Retrieved 27 April 2023, from https://vinuni.instructure.com/courses/1327

[2] Nive927. (2023). GitHub - nive927/Short-term-Hands-on-Supplementary-Course-on-C-Programming: Retrieved 27 April 2023, from https://github.com/nive927/Short-term-Hands-on-Supplementary-Course-on-C-Programming