1 About

This is just a basic project that I did for myself. It's nothing good. I made this just to give myself an idea of how data structures are stored in files, I also learnt about **fread()** and **fwrite()**(although I didn't use them here). It's not even close to a task application. It's just a simple project. But I am glad that I created it even if it is of little to no importance to anyone.

2 Flags

Here are the flags that will be used with the tasker's **task** command:

- **task -a** → *add a task.*
- **task -d** → *delete a task.*
- **task -chp** → *change the priority of a task.*
- **task -done** → *mark the task as done.*
- **task -l** → *list the tasks.*
- **task -clr** → *clear the .tasks file.*
- **task -h** → *help session.*
- **task -q** → *quit tasker.*

There are also color codes:

- **red** → Incomplete task.
- **green** → Completed task.

I got this idea of color codes(and also about proritizing the task) from the **devtodo** program in Linux.

Also, numbers represent the priority:

- 0 → Very important.
- 1 → Important.
- 2 → Moderately important.
- 3 → Less important.
- 4 → Optional.

Inserting a task → A new task will be initially inserted at the end of the queue. After that it will be put in the appropriate position depending on its priority.

Changing the priority of a task → When we change the priority of a task then it will get sorted in the tasklist according to its priority.

Deleting a task → Deletion will be done based on the **id**(of the task) provided by the user.

Duplicate tasks?? → There is a method called **arrange()** which takes two arguments: one is a pointer to the structure **priorq** and other is a string i.e the task name. This method will first check for duplicate task names and if this search is unsuccessful(i.e there are no duplicate tasks) then it will insert the task in its appropriate position based on the priority given to it. This function returns **true** is insertion of task was successful, otherwise it returns **false**.

Definition of the structure priorq

```
typedef struct priorq {
        int priority;
        int id;
        string taskName;
        char isDone;
        struct priorq*next;
}priorq;
```

id is the id of a task to uniquely identify it.

next basically points to the next task.

isDone indicates the status of the task i.e whether it is completed or not.

3 Changing priority of a task

3.1 The chp() function

The **chp()** function takes 3 arguments:

- **priorq*** → pointer to the structure **priorq***.
- **int** → new priority of the task.
- **int** → id of the task.

This function returns the old priority of the task. If the task with the given **id** doesnot exist then it returns -1. If the task list is empty then this function returns 2.

4 Removing whitespaces provided before or after a command

This is very important. For this task we have a function **align()** which basically removes the spaces provided before or after(if any) a command.

5 Fetching tasks

Before, the program asks the user to enter any tasks it will first read tasks(if any) from a file named **.tasks**.

For this task, we have a function called **fetchTasks()** which fetches any tasks from the **.tasks** file. This function takes **priorq**** type argument so that the changes take place in the original **priorq** pointer in the **main()**.

6 Saving tasks

Tasks are saved to the **.tasks** file at the time of quitting the program. For this we have a **writeTasks()** function. This function takes a **priorq*** as an argument.

7 Possibly a cautious step!

There can be cases when we want to run this program just for looking at its commands and not really for setting tasks. In that case, we don't want the program to create an empty **.tasks** file which contains nothing in it. If this empty **.tasks** file gets created then when we re-run this program the next time we will get a **segfault**.

8 clr()

Along with this, we also have the ability to remove the **.tasks** file. For this, we use the **clr()** or the **clear record** function. This function takes a string(the filename) as argument and removes it. It returns **true** if the deletion was successful otherwise it returns **false**.

9 Problems

If previous tasks exist, then these tasks will be read from the file, these tasks will have an id associated with them. Now, there is a variable **_id** which is set to 0 everytime we run the program afresh. Now, there may some tasks saved to **.tasks** file that may have some id, let's suppose that the id of a task is 0. Now, we have **_id** set to 0. Now, let's say that we want to add another task with some priority. Now, the **_id** will be assigned by the program and we know that **_id** is 0 initially, so our task that we just added will have an id 0. So, now we have two tasks with the same id which is a problem!

Solution: A solution to this problem is to get the largest value of id from the tasklist. That way the program will be able to assign a unique id to each new task.

10 Whitespaces between a valid command

What if we have a valid command like this:

```
(task> task                    -a
```

This command is a valid command, but it has lots of whitespaces between **task** and **-a**.

For this, we can use a character pointer to point to the string that is after those whitespaces. Now, we can insert that string character by character so that there is one whitespace between **task** and **-a**.