**Time Tracker Documentation**

**Overview**

This program is a **basic time tracker** that lets you:

1. **Create new tasks**
2. **Start/Stop tracking** how long you spend on a task
3. **Add manual hours** to a task
4. **Save** or **load** your tasks from a file

The program displays a **menu** where you can pick an action by entering a number (1–7).

**Data Storage**

Instead of using advanced data structures, the program stores task-related information in several **parallel arrays**:

* idArr[]: Holds each task’s unique ID.
* nameArr[]: Holds each task’s name (short description).
* runningArr[]: Tracks whether a task is currently timing (true = timer is running, false = not running).
* totalHoursArr[]: Holds the total number of hours spent on each task so far.
* startTimeArr[]: Records the time (time\_t) when the task’s timer was last started.

All arrays have a fixed size (MAX\_TASKS = 100). A variable taskCount tracks how many tasks are currently used/valid in these arrays.

**Key Variables**

* **taskCount**: How many tasks exist right now.
* **nextId**: What the next task’s ID should be (auto-increments each time a new task is created).
* **MAX\_TASKS**: The maximum number of tasks the program can handle (100).

**Functions and Their Purpose**

1. **main()**
   * Calls loadTasksFromFile() at the start to restore any previously saved tasks.
   * Displays a **menu** in a loop.
   * Based on user choice, calls the appropriate function.
   * If the user chooses **Exit (7)**, it optionally saves tasks and then ends the program.
2. **listTasks()**
   * Prints all existing tasks with their ID, name, whether the timer is running, and total hours logged.
3. **createTask()**
   * Lets the user enter a new task name.
   * Increases taskCount, sets idArr[taskCount] to nextId, etc.
   * Increments nextId so no two tasks have the same ID.
4. **startTimer()**
   * Asks the user which task to start timing (by ID).
   * Sets runningArr[index] = true and remembers the current time in startTimeArr[index].
5. **stopTimer()**
   * Asks the user which task to stop timing (by ID).
   * Calculates the time difference (in seconds) between now and startTimeArr[index].
   * Converts the difference to hours and adds it to totalHoursArr[index].
   * Marks the task as not running (runningArr[index] = false).
6. **addManualEntry()**
   * Lets you type in how many hours to add to an existing task (e.g., 1.5 hours).
   * Increases totalHoursArr[index] by the given amount.
7. **saveTasksToFile()**
   * Writes all tasks to a text file named tasks\_data.txt.
   * Each task’s data is saved on a separate line in this format:  
     ID|Name|Running(0 or 1)|TotalHours
8. **loadTasksFromFile()**
   * Reads tasks\_data.txt (if it exists) line by line.
   * Uses basic string functions (find(), substr()) to split each line by the | character.
   * Restores the tasks into the arrays.
   * Updates taskCount and nextId accordingly.
9. **findTaskIndexById(int searchId)**
   * Loops through all tasks, checking if idArr[i] == searchId.
   * Returns the **array index** if found, or -1 if no match.
10. **getFormattedTime(time\_t rawTime)**
    * Converts a time\_t value (representing seconds since 1970) into a human-readable date/time string (e.g., YYYY-MM-DD HH:MM:SS).

**Program Flow**

1. On start, **loadTasksFromFile()** attempts to load any previously saved tasks from tasks\_data.txt.
2. The program displays the **menu**:

mathematica

Copy

1. List All Tasks

2. Create New Task

3. Start Timer for Task

4. Stop Timer for Task

5. Add Manual Entry

6. Save Tasks to File

7. Exit

1. You enter a choice (1–7).
2. The corresponding function executes (e.g., if you choose 2, it calls createTask()).
3. When you **Exit (7)**, the program can save the tasks (optional) and then terminates.

**How To Use**

1. **Compile and Run** the program in a C++ environment (e.g., Dev-C++, Code::Blocks, etc.).
2. **Choose** any menu option by entering its number, then **press Enter**.
3. When asked for a **task ID**, type the ID shown in the task list, then **press Enter**.
4. If you make a mistake, the program will either show an error or give you a chance to re-enter.
5. When you’re done, **choose 7 (Exit)**. The program will end.