DSBA Introduction to Programming // Midterm Programming test

Spring semester 2022/23

Date: May 11, 2023

General Info

Implement the required functions and send your solution to Yandex.Contest. Write everything in a single file `solution.h` and submit that file.

Function `main()` is not needed, but you will need to check your solution locally. To do that, use the provided file `main.cpp`.

All tasks are assumed to be completed in order. Each task in the Contest checks only the methods relevant to the task, so it's possible to solve them out of order. However, if functions or methods interact with each other, the versions from your solution will be used. Follow function prototypes exactly.

Add your own functions where needed as stated in task notes.

Structure

You have two sets of records. One contains information about some workers, people who solve tasks. It is stored in the file `workers.txt`.

The other set contains records about tasks these workers solve. It is stored in `tasks.txt`.

Code that reads data from these files is already given to you, you may assume it works correctly and reads data properly.

Both tasks and workers are stored as structures.

```
struct Task {
    std::string text;
    std::string date;
    std::map<std::string, int> workloads;
    using Workload = std::map<std::string, int>::value_type;
};
```

Tasks contain *text* - the text description of what problem should be solved, *date* - a date when the task must be completed, as a string YYYY-MM-DD, and *workloads* - a map login-workload of

the type "string-int" which stores how much work is expected from different workers to solve this task. All workloads are positive integers.

For example, the first line of the file `tasks.txt` reads:

```
merge new feature,2023-05-10,daniel 5,m_gump 10
```

Here the text description *text* is "merge new feature", the task should be completed on "2023-05-11", worker "daniel" is expected to spend 5 units of work (for example, hours) on in, and worker "m_gump" is expected to spend 10 units of work on it.

Additionally, a single pair "login-workload" has a type alias `Task::Workload`. You may add more aliases for your convenience.

```
struct Worker {
    std::string name;
    std::string login;
    int maxLoad;
};
```

Worker structure contains *name* - the name of the worker, a *login* - a unique identifier of the worker and a *maxLoad* - how much work can this worker do in total (not per day).

Both *name* and *login* don't contain spaces.

Task 1: Constructors [up to 0.1]

Implement constructors for structures `Task` and `Worker` which take all fields of their structures as input.

The order for `Task` is:

1. *text*
2. *date*
3. *workloads*

The order for `Worker` is:

1. *name*
2. *login*
3. *maxLoad*

Task 2: Total workload of a task [up to 0.2]

Implement a method `int getTotalLoad()` for structure `Task`.

It should return the sum of all loads from the `workloads` field as a single integer.

Example

Consider the following task

```
merge new feature,2023-05-10,daniel 5,m_gump 10
```

There are two people assigned to it: one with workload 5 and other with workload 10. For this task the function should return **15**.

Task 3: Sort tasks [up to 0.2]

Implement a function `void sortTasks(std::vector<Task>& tasks)`.

It receives a reference to a vector of tasks and sorts that vector in ascending order by total workload. If workloads are equal, then it sorts tasks by date. If dates are equal as well - by text.

Use a custom comparator function or overload `operator<` for the structure `Task`.

Example

Consider the following list of tasks (represented as initial text inputs).

```
merge new feature,2023-05-10,daniel 5,m_gump 10
update docs,2023-05-09,daniel 11
fix issue #123,2023-05-09,a_penrose 4,johnsmith 7
water plants,2023-05-10,johnsmith 1
update dependencies,2023-05-10,a_penrose 8
```

It could be represented as a table:

text	date	total load
merge new feature	2023-05-10	15
update docs	2023-05-09	11
fix issue #123	2023-05-09	11
water plants	2023-05-10	1
update dependencies	2023-05-10	8

When sorted, this list of tasks will look like this:

text	date	total load

---

text	date	total load
water plants	2023-05-10	1
update dependencies	2023-05-10	8
fix issue #123	2023-05-09	11
update docs	2023-05-09	11
merge new feature	2023-05-10	15

Note that `"fix issue #123"` appears before `"update docs"` - their workload is the same, their date is the same, so they are compared by their text.

Task 4: Total workload of a worker [up to 0.2]

Implement the function `int getWorkerLoad(const std::vector<Task>& allTasks, const Worker& worker)`.

The function takes a vector of all tasks and a single worker as input. It should calculate the total load assigned to this worker among all the tasks.

Example

```
merge new feature,2023-05-10,daniel 5,m_gump 10
update docs,2023-05-09,daniel 11
fix issue #123,2023-05-09,a_penrose 4,johnsmith 7
water plants,2023-05-10,johnsmith 1
update dependencies,2023-05-10,a_penrose 8
```

Here the worker with the login `daniel` has two tasks assigned. One with workload 5, another with workload 11. The total is **16**.

Task 5: Adding a new task [up to 0.1]

Implement the function `void addTask(std::vector<Task>& tasks, const Task& newTask, const std::vector<Worker>& workers)`.

The function takes a vector of tasks, a single new task and a vector of workers.

It should add the task to the vector of tasks. However, if the total workload of any worker exceeds their maximum load (field `maxLoad`), the function should throw an exception `std::runtime_error` with the message saying `"Overworked"`.

Example

```
```

```
merge new feature,2023-05-10,daniel 5,m_gump 10
update docs,2023-05-09,daniel 11
fix issue #123,2023-05-09,a_penrose 4,johnsmith 7
water plants,2023-05-10,johnsmith 1
update dependencies,2023-05-10,a_penrose 8
```

Adding a new task:

```
make a prototype of a new feature,2023-05-10,johnsmith 6
```

Worker `johnsmith` has the maximum load of 10. However, the total workload for this worker is now `7 + 1 + 6 = 14`. They are considered overloaded, so the program throws an exception.

If the workload of the new task was 2, it would have been successfully added to the list because the total workload would be `7 + 1 + 2 = 10`, not exceeding the maximum workload of `johnsmith`.

Task 6: Adding a useful error message [up to 0.2]

Implement the function `void addTaskVerbose(std::vector<Task>& tasks, const Task& newTask, const std::vector<Worker>& workers)`.

It should work the same as the function from task 5, but it includes more information in its error messages.

If any workers are overworked, throw an exception `std::runtime_error` with the message saying `"Overworked: login1 by X, login2 by Y"` where `login1`, `login2` are logins of the workers and `X`, `Y` are integers showing the difference between the maximum load of a worker and their current load. Include all overworked workers in the message, sort them *by logins* and separate them by a comma and a space `, `.

Do not include a newline character at the end of the message.

Example

```
merge new feature,2023-05-10,daniel 5,m_gump 10
update docs,2023-05-09,daniel 110
fix issue #123,2023-05-09,a_penrose 4,johnsmith 7
water plants,2023-05-10,johnsmith 1
update dependencies,2023-05-10,a_penrose 8
```

Adding a new task:

```
make a prototype of a new feature,2023-05-10,johnsmith 6,a_penrose 2
```

Worker `johnsmith` becomes overworked after this task is added. Worker `daniel` is already overworked, so the error message should say

```
Overworked: daniel by 95, johnsmith by 4
```

Here 4 is **14** (total load) minus **10** (`johnsmith` 's maximum load). 95 is `5 + 110 - 20` . Login `daniel` comes first when they are sorted alphabetically.