TIE-20106 Data structures and algorithms, spring 2017

version 30.1.2017

Programming assignment 1: Salary statistics

Topic of the assignment

In this assignment you write a program to produce salary statistics for a set of emplyees. The program is given as input information about employees (name and salary), and it can produce a list of employees in alphabetical or salary order. In addition to this the program can produce other salary-related statistics, which are used in real life as well: minimum and maximum salaries, median salary and salary quartiles (definitions of these can be found later in this document).

(Side note: medians and quartiles tell more about salaries than average and standard deviation, because salaries are usually really unevenly distributed. This means that a small number of billionaires can make the average unrealistically high compared to "normal" salaries).

Since this is a Data structures and algorithms assignment, performance of the program is an important grading criteria. Minimum requirement is that for each operation the average performance should be at least $\Theta(n \log n)$. Getting better performance helps to increase the grade. Especially note the following:

- In performance the essential thing is how the execution time changes with more data, not just the measured seconds.
- The required minimum performance ($\Theta(n \log n)$) includes the following: first n elemented are added to an empty data structure, and then the actual operation is performed. I.e., it is not possible to "cheat" better performance by doing heavy operations while adding element (of course in certain situations it may improve performance if some things are done while adding an element).
- More points are given if operations are implemented with better performance than the minimum requirement.
- Likewise more points will be given for better real performance in seconds (if the required minimum asymptotic performance is met). **But** points are only given for performance that comes from algorithmic choices and design of the program. (For example setting compiler optimization flags, using concurrency or hacker optimizing individual C++ lines don't give extra points).
- If the implementation is bad enough, the assignment might not pass even if the minimum performance criteria is met.

- (Added 16.2.) Examples of questions, which may help in improving the performance: Is somewhere the same thing re-done several times? Can you sometimes avoid doing something completely? Does some part of the code do more work than is absolutely necessary? Can some things be done "almost free" while doing something else?

Definitions of the statistics used in the assignment

In this assignment the following statistics are used to get information about salaries. To make the assignment easier, these definitions are a little simplified compared to the "real" mathematical definitions. Below "salary order" means salaries ordered from smallest to largest.

- *Minimum / maksimum:* Person with the smallest/largest salary. If there are more than one such person, any one of them can be chosen.
- *Median:* The middle person in salary order. In this assignment the person with index $\lfloor \frac{n}{2} \rfloor$. ($\lfloor x \rfloor$ is rounding down to nearest integer, i.e. the normal C++ rounding in integer division).
- *First quartile:* The person in the 1/4 position in salary order. In this assigment the person with index $\lfloor \frac{n}{4} \rfloor$.
- *Third quartile:* The person in the 3/4 position in salary order. In this assigment the person with index $\lfloor \frac{(3n)}{4} \rfloor$.

On sorting

While sorting employees in salary order it's possible that several persons have the same salary (or the same name when sorting alphabetically). The mutual ordering of such persons doesn't matter.

The main program only accepts names consisting of letters A-Z, a-z and spaces. The alphabetical sorting can be done either with the regular < comparison of C++ string class (in which case space comes first, then upper case letters followed by lower case letters) or the "correct way", where upper and lower case letters are equal, but space still comes before letters.

Retrictions in doing the assignment

The programming language in this assignment is C++11. Algorithms provided by the programming language or libraries (`std::sort`, `std::nth_element`, `std::list::sort`, etc.) are not allowed, but algorithms related to sorting have to be written by students (also, direct copying of code from other sources is not allowed). Allowed data structures in this assignment are sequence containers in C++11 (`std::array`, `std::vector`, `std::deque`, `std::list`), associative containers are not allowed (`std::map`, `std::set`, etc.).

Structure and functionality of the program.

Part of the program code is provided by the course, part has to be implemented by students.

Parts provided by the course

File *main.cpp* (you are **NOT ALLOWED TO MAKE ANY CHANGES TO THIS FILE)**

- Main routine, which takes care of reading input, interpreting commands and printing out results. The main routine contains also commands for testing.

File *datastructure.hpp*

- `struct Person`: Used to pass information about employees
- `class Datastructure`: The implementation of this class is where students write their code. The public interface of the class is provided by the course. You are **NOT ALLOWED TO CHANGE THE PUBLIC INTERFACE** (i.e. change names, return type or parameters of the given public member functions).

Parts of the program to be implemented as the assignment

Files *datastructure.hpp* and *datastructure.cpp*

- `class Datastructure`: The given public member functions of the class have to be implemented. You can add your own stuff into the class (new data members, new member functions, etc.)

Note! The code implemented by students should not do any output related to the expected functionality, the main program does that. If you want to do debug-output while testing, use the `cerr` stream instead of `cout`, so that debug output does not interfere with the tests.

Commands recognized by the program and the public interface of the Datastructure class

When the program is run, it shows prompt ">" and waits for commands explained below. The commands where the explanation mentions a member function, call that member function of the Datastructure class (implemented by students). Other commands are completely implemented by the code provided by the course.

If the program is given a file as a command line parameter when it is run, the program reads commands from that file and then quits.

Command **`Public member function`**	**Explanation**
add 'nimi' palkka `void add_person(string name, int salary);`	Adds an employee to the data structure with given name and salary.
size `unsigned int size();`	Returns the number of employees currently in the data structure.
clear `void clear();`	Clears out the data structure (after this size returns 0).
alphalist `vector<Person*> personnel_alphabetically();`	Returns a list (vector) of the employees in alphabetical order.

Command **`Public member function`**	**Explanation**

salarylist
`vector<Person*>`
`personnel_salary_order();`
	Returns a list (vector) of the employees in salary order (smallest first).

min `Person* min_salary();`	Returns the employee with the smallest salary (definition of minimum is elsewhere in this document).
max `Person* max_salary();`	Returns the employee with the largest salary (definition of maximum is elsewhere in this document).
median `Person* median_salary();`	Returns the employee with the median salary (definition of median is elsewhere in this document).
1stquartile `Person* first_quartile_salary();`	Returns the employee with the 1st quartile salary (definition of 1st quartile is elsewhere in this document).
3rdquartile `Person* third_quartile_salary();`	Returns the employee with the 3rd quartile salary (definition of 3rd quartile is elsewhere in this document).
random_add n (pääohjelman toteuttama)	Add n new employees with random name and salary (for testing). Note! The values really are random, so they can be different for each run.
read 'filename' (pääohjelman toteuttama)	Reads more commands from the given file (This can be used to read a list of employees from a file, run tests, etc.)
stopwatch on / off / next (pääohjelman toteuttama)	Switch time measurement on or off. When program starts, measurement is "off". When it is turned "on", the time it takes to execute each command is printed after the command. Option "next" switches the measurement on only for the next command (handy with command "read" to measure the total time of a command file).
perftest timeout n n1;n2;n3... (pääohjelman toteuttama)	Run performance tests. Clears out the data structure and add n1 random employees. Then a random command (random_add 1, min, max, median, 1stquartile, 3rdquartile, alphalist, or salarylist) is performed n times. The time for all this is measured and printed out. Then the same is repeated for n2 employees, etc. If any test round takes more than timeout seconds, the test is interrupted (this is not necessarily a failure, just arbitrary time limit).
testread 'infilename' 'outfilename' (pääohjelman toteuttama)	Runs a correctness test and compares results. Reads command from file infilename and shows the output of the commands next to the exptected output in file outfilename. Each line with differences is marked with a question mark. Finally the last line tells whether there are any differences.
help (pääohjelman toteuttama)	Prints out a list of known commands.
quit (pääohjelman toteuttama)	Quit the program. (If this is read from a file, stops processing that file.)

"Data files"

The easiest way to test the program is to create "data files", which using "add "commands enter a list of employees into the data structure. Those files can them be read in using the "read" command, after which other commands can be tested without having to enter those employees every time by hand.

Below is an example of a data file, which can be found as *example-data.txt*:

```
add 'Meikkis Matti' 2000
add 'Teikkis Terttu' 4000
add 'Miljoona Miikka' 1000000
add 'Sorrettu Sami' 1
add 'Keskiverto Keijo' 3000
add 'Kukalie Kirsi' 2500
add 'Olematon Oskari' 6000
```

Example run

Below is an example output from the program. The example's commands can be found in file *example-in.txt* and the output in file *example-out.txt*. This means you can use this example also as a test by running the program and giving command *testread 'example-in.txt' 'example-out.txt'*.

```
> clear
Cleared all persons
> size
Number of employees: 0
> read 'example-data.txt'
** Commands from 'example-data.txt'
> add 'Meikkis Matti' 2000
> add 'Teikkis Terttu' 4000
> add 'Miljoona Miikka' 1000000
> add 'Sorrettu Sami' 1
> add 'Keskiverto Keijo' 3000
> add 'Kukalie Kirsi' 2500
> add 'Olematon Oskari' 6000
>
** End of commands from 'example-data.txt'
> size
Number of employees: 7
> min
Sorrettu Sami, salary 1
> max
Miljoona Miikka, salary 1000000
> median
Keskiverto Keijo, salary 3000
> alphalist
Keskiverto Keijo, salary 3000
Kukalie Kirsi, salary 2500
Meikkis Matti, salary 2000
Miljoona Miikka, salary 1000000
Olematon Oskari, salary 6000
Sorrettu Sami, salary 1
Teikkis Terttu, salary 4000
> salarylist
Sorrettu Sami, salary 1
Meikkis Matti, salary 2000
Kukalie Kirsi, salary 2500
Keskiverto Keijo, salary 3000
Teikkis Terttu, salary 4000
Olematon Oskari, salary 6000
Miljoona Miikka, salary 1000000
> 1stquartile
Meikkis Matti, salary 2000
> 3rdquartile
Olematon Oskari, salary 6000
> quit
```