Exercise 3 - Requirements and Guidelines

~~In this exercise you are required to create simulation program that will run several algorithms on several houses, algorithms shall now be smart as they will participate in a competition. Submissions shall include 3 different algorithms, as .so files that would be dynamically loaded by the simulation, as would be described below. The quality of the algorithm is an important part of this exercise, you are especially required that your submitted 3 algorithms:~~

(a) would actually behave differently
~~(b) be completely deterministic, i.e. would not use any random function and would have the exact same behavior each time it runs on the same house~~
(c) would take into account battery considerations
(d) would take into account dust that was revealed and not fully cleaned yet
(e) would try to "map" the house in order to get smart cleaning decisions
(f) would have a logic for returning back to the docking station on time

20 points of the exercise would be based on the algorithms competition:

- if all 3 algorithms are at the bottom 20% - you lose 20 points
- otherwise, if all 3 algorithms are at the bottom 50% - you lose 10 points

Additional bonus points based on the algorithm competition:

- 10 bonus points would be given for having at least 1 algorithm in the top 3%
- Being in the top 10% - with 2 algorithms: 5 bonus points, with all 3: 10 bonus points

(You can earn both bonuses. Grades above 100 are possible and get into the average).

Assume MaxSteps per house would allow smart algorithms to finish successfully.

~~The algorithms that you submit MUST be called after the ID of one of the team members:~~
~~<ID>_A_.so with class name for the algorithm = _<ID>_A~~
~~<ID>_B_.so with class name for the algorithm = _<ID>_B~~
~~<ID>_C_.so with class name for the algorithm = _<ID>_C~~
~~All three files must use the same team member ID.~~

~~House files: as in exercise 2. Note: submission shall include 3 different house files.~~

~~Command-line arguments and usage: as in exercise 2 plus the following addition.~~
~~(d) score_formula <score .so path>~~
~~(e) threads <num threads>~~

~~Program usage printout is:~~

```
Usage: simulator [-config <config path>] [-house_path <house path>]
[-algorithm_path <algorithm path>] [-score_formula <score .so path>]
[-threads <num threads>]
```

~~As in exercise 2, order of arguments is not defined, any order should be valid and supported. Each of the path arguments can be set with absolute or relative path. One argument can be relative and another absolute, or any other combination. The path arguments, if provided, shall point to a directory (NOT to a file) and may or may not have a trailing slash.~~
~~Each of the path arguments may be missing, in which case application should look for the files associated with this argument in the working directory - except for score formula argument which if missing shall use the default score formula.~~

Files to look for:

- ~~Config file: config.ini~~
- ~~Score formula: score_formula.so~~
- ~~House files, files with suffix: .house~~
- ~~Algorithm files: *_.so~~

Config file: as in exercise 2 **with the following changes**:

1. ~~In case config file is missing in the folder (default or the one specified by command line argument): after printing the usage an additional error line shall follow, saying:~~
   ~~`cannot find config.ini file in '<full path>'`~~
   ~~and the program shall return.~~
2. ~~In case config file parameter has a bad value (not a number, negative): error message shall appear, without a usage, saying:~~
   ~~`config.ini having bad values for <number> parameter(s): <parameter_name1>, <parameter_name2>, ...`~~
   ~~and the program shall return. Example:~~
   ~~`config.ini having bad values for 1 parameter(s): BatteryCapacity`~~
   ~~Order of bad parameters in the error message is not important.~~

~~Note: you can add these additional error messages also to ex2 if you wish.~~
~~Other error messages are the same as in ex2.~~

Score formula file

~~The score formula is the second argument to be checked on startup (after config file).~~
~~In case `-score_formula` argument is not provided in the command line, the default score formula shall be used, this is not an error and no message shall appear~~.

~~In case `-score_formula` argument is provided but "score_formula.so" cannot be found in the relevant folder in which we look for (argument doesn't lead to a valid folder, folder doesn't have such a file, etc.), usage should be printed, following with the following line:~~
~~`cannot find score_formula.so file in '<full path>'`~~
~~and the program shall return.~~

~~In case score formula file exists but cannot be opened or doesn't have valid implementation, one of the following error shall be printed accordingly:~~
~~`score_formula.so exists in '<full path>' but cannot be opened or is not a valid .so`~~ Or
~~`score_formula.so is a valid .so but it does not have a valid score_formula`~~

~~The score_formula.so shall have the following global function:~~
~~`int calc_score(const map<string, int>& score_params);`~~

~~The function calculates score according to `score_params` and returns the calculated score.~~
~~In case score cannot be calculated for some reason, the function shall return -1.~~ If this ~~happens the simulation shall ignore it and continue, but add a last line of error, after the results table and after all other errors, saying:~~
~~`Score formula could not calculate some scores, see -1 in the results table`~~

Note:

1. Your algorithms shall be optimized for the default score formula.
2. -1 result shall be used in the average calculation as is (i.e. as -1)

~~Algorithms: as in exercise 2 **with the following change**: in case the folder for the algorithm files is bad for some reason or leads to a directory with no algorithm files, or is missing~~ and ~~there are no algorithm files in the working directory: after printing the usage an additional error line shall follow, saying:~~ `cannot find algorithm files in '<full path>'` ~~and the program shall return~~.
~~Note: you can add this additional error line also to ex2 if you wish.~~
~~Other error messages are the same as in ex2.~~

~~Houses: as in exercise 2 **with the following change**: in case the folder for the house files is bad for some reason or leads to a directory with no house files, or is missing and there are no house files in the working directory: after printing the usage an additional error line shall follow, saying:~~ `cannot find house files in '<full path>'` ~~and the program shall return.~~
~~Note: you can add this additional error line also to ex2 if you wish.~~
~~Other error messages are the same as in ex2.~~

~~Results table: same as in ex2 **with the following change**: order of algorithms in table shall be printed sorted from highest average score to lowest.~~

~~Printout example:~~

```
------------------------------------------------------------
|                |001        |002        |006        |AVG        |
------------------------------------------------------------
|331332334_C_ |        200|        80|        1220|    500.00|
------------------------------------------------------------
|331332334_A_ |        600|        705|        102|    469.00|
------------------------------------------------------------
|331332334_B_ |        100|        109|        1100|    436.33|
------------------------------------------------------------
```

~~NOTE: do not assume that simulation runs only on 3 algorithms / 3 houses, there can be any number of houses / algorithms.~~
~~Order of houses (left to right) shall be based on case-sensitive order (lexical sort).~~

~~**Change in AbstractAlgorithm**:~~
~~The class AbstractAlgorithm shall **change** the signature of the method 'step' to be:~~
~~`virtual Direction step(Direction prevStep) = 0;`~~
Why this change?
In some cases, the robot (the simulator in our case) decides not to step according to the direction provided by the algorithm. Thus, for each step, the robot (the simulator in our case) provides the algorithm the previous step's direction that was actually taken. ~~In the first call to 'step' when starting cleaning the house, simulation will send STAY as prevStep.~~
~~Your simulation can always perform the recommended step~~, ~~but your algorithm shall not assume that.~~ Our simulation may occasionally decide not to follow the recommended step! When calculating path through the house (e.g. calculating way back to the docking station or getting back to previously located dust) ~~the algorithm MUST take into account the actual steps that were taken, provided as a parameter to 'step', not relying on the recommended steps returned from the calls to 'step' to be the actual steps taken~~.

Additional instructions and requirements

Threads

~~You are required in this exercise to allow the simulation to run in multiple threads, according to the *-threads* command line parameter. If this parameter is missing or is not a valid number (e.g. negative or zero) the number of threads would be 1, without any error message.~~
~~When running more than 1 thread, the simulation will run in parallel on several houses at the same time, for that, each "sub-simulation" will use its own copy of all algorithms. In case number of houses is smaller than the requested number of threads, the actual number of threads would be equal to the number of houses. In case number of houses is bigger than number of threads, when a thread finishes its "sub-simulation" on the house it was running on, it would pick the next waiting house.~~
The final report table shall still be sorted according to the house names and not according to the actual order they ran or finished.

AlgorithmRegistration

~~To allow us to run your algorithms with our registration function, we need to define a common registration process. So we provide you with the following header file for AlgorithmRegistration that you need to include in your Algorithm classes:~~

```
#include <functional>
#include <memory>
class AlgorithmRegistration {
public:
AlgorithmRegistration(std::function<std::unique_ptr<AbstractAlgorithm>()>);
};

#define REGISTER_ALGORITHM(class_name) \
AlgorithmRegistration register_me_##class_name \
      ([]{return make_unique<class_name>();} );
```

~~You are free to implement the cpp file for the AlgorithmRegistration as you wish but you are not allowed to change its header file! We may implement it differently, however the Algorithm classes are unaware of the actual implementation of the AlgorithmRegistration class. Note that the AlgorithmRegistration class is NOT the registrar itself and you have the freedom to implement the registrar itself as you wish.~~

~~Each of your algorithm cpp files will have the following line in the global scope:~~

```
REGISTER_ALGORITHM (<class_name>)
```

~~For example:~~

```
REGISTER_ALGORITHM (_331332334_C)
```

SmartPointers

A bonus of 5 points would be given for code that do not use 'new' and 'delete' but instead uses (where relevant) unique_ptr / shared_ptr and make_unique / make_shared. Note that for using shared_ptr you must have good reasons, as unique_ptr is more efficient.
To get the bonus, all parts of the exercise shall avoid using new and delete, that includes the simulation part and the algorithms part.

Examples of some printouts of possible runs:

Printout example (run with errors):

```
-------------------------------------------------------------
|              |001        |002        |006        |AVG        |
-------------------------------------------------------------
|331332334_C_ |          0|          0|       1500|     500.00|
-------------------------------------------------------------
|331332334_A_ |        600|        705|        102|     469.00|
-------------------------------------------------------------
|331332334_B_ |        100|        203|         -1|     100.67|
-------------------------------------------------------------

Errors:
003.house: missing docking station (no D in house)
004.house: too many docking stations (more than one D in house)
005.house: line number 2 in house file shall be a positive number, found: bla
006.house: cannot open file
331332334_C_.so: file cannot be loaded or is not a valid .so
331332334_E_.so: valid .so but no algorithm was registered after loading it
Algorithm 331332334_C_ when running on House 001 went on a wall in step 7
Algorithm 331332334_C_ when running on House 002 went on a wall in step 115
Score formula could not calculate some scores, see -1 in the results table
```

Printout example (config file found, but missing one value):

```
config.ini missing 1 parameter(s): BatteryCapacity
```

Printout example (config file found, but missing few values):

```
config.ini missing 2 parameter(s): BatteryCapacity, BatteryRechargeRate
```

~~Note 1: order of missing parameters in error message is not important~~.
Note 2: additional parameters in config.ini that are not defined shall be ignored, it's not an error.
Note 3: order of parameters in config.ini shall NOT be assumed! parameters may appear in any order.
Note 4: if a parameter appears in config.ini more than once, it's not an error, last value shall be taken.

Printout example (config file found, having bad values):

```
config.ini having bad values for 2 parameter(s): BatteryCapacity,
BatteryRechargeRate
```

Printout example (no algorithm file found in the relevant directory):

```
Usage: simulator [-config <config path>] [-house_path <house path>]
[-algorithm_path <algorithm path>] [-score_formula <score .so path>]
cannot find algorithm files in '/si'
```

Printout example (all algorithm files found in the relevant directory are invalid):

```
All algorithm files in target folder '/si' cannot be opened or are invalid:
331332334_A_.so: file cannot be loaded or is not a valid .so
331332334_B_.so: valid .so but no algorithm was registered after loading it
331332334_C_.so: valid .so but no algorithm was registered after loading it
```