BLG 252E - Object Oriented Programming
Assignment #2

Due: April, 24th 23:59

Donovan rose to his feet in excitement. "And it strikes an equlibrium. I see. Rule Three drives him back and Rule Two drives him forward-"

- from the book: "I, Robot"

Introduction

In this assignment, you are expected to implement a turn-based game called "Explore&Mine". In this game, player's goal is to retrieve selenium mine on the planet Mercury with explorer and miner robots while preserving profit.

For any issues regarding the assignment, please contact Erhan Biçer (**bicer21@itu.edu.tr**).

Implementation Notes

The implementation details below are included in the grading:

1. Please follow a consistent coding style (indentation, variable names, etc.) with comments.
2. You are not allowed use STL containers.
3. You may (and should) implement any getter and setter methods when they are needed. You may implement helper functions if you need, however **implement them as inline functions in header**.
4. Allocate memories dynamically for arrays and implement necessary destructors where the allocated memory parts are freed. Make sure that there is no memory leak in your code.
5. Define required functions and variables as **const** in all cases where necessary otherwise you will get minus points. If a keyword is needed for a variable, you should define it.
6. Do not forget to update/set member variables when necessary.
7. Do not make any modification in main.cpp. Your code will be evaluated using main.cpp.
8. For each random operation, **use rand() function**. Seed is already set in main function as 99.
9. Since your code will be tested automatically, make sure that your outputs match with sample scenario for given inputs.You will be provided with a calico file to check your assignment output.

Submission Notes

1. Your program should compile and run on Linux environment. You should code and test your program on Docker's Virtual Environment. Do not use any pre-compiled header files or STL commands. Be sure that you have submitted all of your files.
2. Send only Robot.h, CommandCenter.h, CommandCenter.cpp and Robot.cpp
3. Do not change included libraries and path to header files.
4. Submissions are made through **only** the Ninova system and have a strict deadline. Assignments submitted after the deadline will not be accepted.
5. This is not a group assignment and getting involved in any kind of cheating is subject to disciplinary actions. Your homework should not include any copy-paste material (from the Internet or from someone else's paper/thesis/project).

1 Implementation Details

You are expected to implement 5 classes; **Robot**, **Explorer**, **Miner**, **Crew**, **CommandCenter**.

1.1 Robot

Private Attributes:

Robot class *has* a **unqID** (unique integer for each robot), **RobotName** (eg. EXP-01), **RobotType** (eg. explorer, miner), **speed** (eg. 1.8), **duration** (eg. 5), **manufactureCost** (eg. 2500), **maintenanceCost** (eg. 2500) and **canMove** (eg. false)

- **RobotName** should be in a specific format where the first three digit represent robot type, fourth digit is hyphen and the last digit(s) are **unqID** as "EXP-1" for the first robot. "exp" and "mnr" represent explorer and miner respectively.
- **speed** determines how fast can robot achieve its objective for a turn. Faster robot reduces **maintenanceCost** proportionally.
- **manufactureCost** represents the cost of manufacturing a robot, whereas **maintenanceCost** represents the cost of maintaining robot for each turn when **they take action (moving)**.
- Robot cannot take an action unless **canMove** is true. **canMove** should be initialized as true.

Warning: Robot's type and manufacture cost **must not be changed** after instantiating an object from this class.**unqID** does not depend on an instance. It can exist when there are no instance/object of the class.

Methods:

1. **Constructor(s):** The constructor should take **type**, **speed**, **durability**, **manufactureCost** and **maintenanceCost** as arguments. In the body of the constructor, remaining private attributes should be set.
2. **operator=:** Overload assign operator to copy a Robot's attributes to another Robot
3. **showInfo():** This method will print the attributes of the robot.

1.2 Explorer

Explorer class "**publicly**" inherits Robot class.

Private Attributes:

totalExploredArea is total area of the field explored by *all* explorer robots within your crew. **exploredArea** is total area of the field explored by an explorer robot within your crew. **totalNumberOfSeleniumArea** represents the total *number* of areas *where selenium is found by explorer robots* and *has not been extracted yet by miner robots* **detectedSelenium** represents whether selenium is detected or not.

Warning: totalExploredArea and **totalNumberOfSeleniumArea** member variables does not depend on an instance. They can exist when there are no instance/object of the class.

Methods:

1. **Constructor(s):** The constructor should take variables that are inherited as arguments (type, speed, durability). **manufactureCost** and **maintenanceCost** of Explorer Robot is 3000 unit and 450. **maintenanceCost** should be initialized as 450/**speed**.
2. **move:** This method returns a **boolean** variable that represents allowance of robot to be moved for current game turn. The explorer robot can move if it has not moved for the current turn already.
3. **explore:** This method is used to explore areas for finding a selenium on the ground with sensors. A random number between 0 and 100 is generated. If its higher than equal to 40, selenium signals are found and **detectedSelenium** should be set to true to represent the explored area contains selenium. Otherwise, **detectedSelenium** should be set to false. In both cases, increment **totalExploredArea** by **1500** and set **exploredArea** to **1500**. If selenium is detected, make sure that **totalNumberOfSeleniumArea** is incremented by 1. This exploration should cost you as the robot's **maintenanceCost**.
4. **operator=:** Overload assign operator to copy an Explorer Robot's attributes to another Robot.
5. **showInfo():** This method should show the values of the member variables of the explorer robot including inherited ones.

1.3 Miner

Miner "publicly" inherits Robot class.

Private Attributes:

gatheredSelenium is the total number of gathered selenium by a miner robot. **totalGatheredSelenium** is the total number of gathered selenium by *all* miner robots within your crew.

Warning: totalGatheredSelenium member variable does not depend on an instance. It can exists when there are no instance/object of the class.

Methods:

1. **Constructor(s):** The constructor should take variables that are inherited as arguments (type, speed, durability). **manufactureCost** and **maintenanceCost** of Explorer Robot is 5000 unit and 750. **maintenanceCost** should be initialized as 750/**speed**.
2. **mine:** This method is used to mine the areas with seleniums. After explorers detect areas with seleniums, miners can extract selenium mines on these areas. The amount of extracted selenium is as following steps: (1) A random **integer** between 10 and 100 **(both inclusive)** is generated. (2) This random number is multiplied by 5 to compute the extracted number of seleniums. This operation should cost you as the robot's **maintenanceCost**.
3. **move:** This method returns a boolean variable that represents allowance of robot to be moved for current game turn. The miner robot can move if there is an area with seleniums and it has not moved for this turn already. If these conditions are met, return true otherwise return false.

4. **operator=:** Override overload assign operator to copy Miner Robot's attributes to another Miner Robot.
5. **showInfo():** This method should show the values of the member variables of the miner robot including inherited ones.

Notice: Miner class can directly access to the private member variables of the Explorer class directly.

1.4 Crew

Private Attributes:

maxExplorers and **maxMiners** are the max number of "exp" robot and "mnr" robots that your Crew can haverespectively. If these are not between 1-5, set it as 2. **minerCrewSize** and **ExplorerCrewSize** are current size of the MinerCrew and ExplorerCrew respectively. These cannot be negative or higher than the maximum numbers. **MinerCrew** and **ExplorerCrew** are pointer to pointer which points to a Miner Robot Type and Explorer Type. **crewManufactureCost** and **crewMaintenanceCost** are total manufacture cost and total maintenance cost of the crew.

Methods:

1. **Constructor(s):** The constructor should take **maxExplorers**, **maxMiners**,**explorerCrewSize** and **minerCrewSize**. Fill **ExplorerCrew** and **MinerCrew** with using **manufactureRobot**.
2. **manufactureRobot:** Throughout the game, robot manufacturing will be done with this method. This method would return a pointer to **Robot** type variable. The method takes the type of the robot as string optionally. Yet, this method should also work when type is not given as argument (e.g. when player types the robot type as input). You should ensure that robotType is not different than explorer or miner. Durability should be a random number between 1 and 5. Speed should be a random float number between 1.5 and 2.5 for explorer, and between 1.0 and 1.5 for miner. Ensure that a single random operation is done for determining speed **according to type of the robot**. Avoid surplus/unnecessary random operations. It is important that **manufactured robot cannot move for that turn.**
3. **initMovement:** This method is used to initialize the allowance for the mobility of the robots as true.
4. **updateDurability** If durabilities are different than 0, decrement by 1. If a robot's durability is 0, it cannot move again and should be removed if it's not the only robot for its crew (e.g. if an explorer robot has 0 durability and it is the only one in ExplorerCrew, it should not be removed.).
5. **operator+=:** Overload += operator to add a new Robot to your crew.
6. **operator-=:** Overload -= operator to remove a Robot from the crew. This function should be able to remove a robot from its crew by takign its name as argument. This function must not remove a Robot when that Robot is the single Robot for that crew.
7. **showInfo():** This method should call the **showInfo()** method of all robots within the crew and print the **crewManufactureCost** and **crewManufactureCost**.

1.5 CommandCenter

Private Attributes:

neededSelenium is aimed *number* of seleniums. **searchArea** is the determined area of the search field within the planet. **seleniumWorth** is the worth of each selenium. **turnCount** represents the turn number. **profit** represents the profit.

Warning: neededSelenium, **searchArea** and **seleniumWorth must not be changed** after instantiating an object from this class.

Methods:

1. **Constructor(s):** Initialize member variables.
2. **isGameEnded:** returns true if game has ended, otherwise returns false. Game ends when **searchArea** is explored. Player wins if profit is positive value and **neededSelenium** is met. Do not forget that **neededSelenium** is a number of the seleniums, not worth of them. If player wins, print out "Congrats!", otherwise print "Mission Failed!".
3. **calculateProfit:** Source of the expenses are **crewMaintenanceCost** and **crewManufactureCost**.Source of the revenue is selenium. Calculate **profit** according to this.
4. **operator++:** Increment **turnCount**.
5. **showInfo():** Show, **turnCount**, **neededSelenium**, **totalGatheredSelenium**,**searchArea**, **totalExploredArea**,**totalNumberOfSeleniumArea**. If there are areas with seleniums, show player a reminder that miners should dig these areas.

Notice: CommandCenter class can directly access to the private member variables of the Explorer, Miner and Crew.

2 Gameplay

To fully understand the assignment, plase watch the gameplay video and examine calico carefully. Below menu numbers are given in paranthesis.

Firstly, you should build your explorer and miner crew. Then you can search for seleniums (1) and mine (2) if there is any. After making a move with robots, you can end the turn to make them move again (7). If you have lots of expenses, you can remove a robot to save maintenanceCost (4). If you think your crew is insuffucient in number, manufacture a robot (3). During the game you should be able to see team information (5) and statistics of the game(6).

3 How to compile, run and test your code:

With below code you can compile and run the code.
g++ -Wall -Werror src/main.cpp src/Robot.cpp src/CommandCenter.cpp -I include -o assignment2
./assignment2
.vscode folder is shared with you so that you can debug and run your code in vscode environment You can (and should) run the calico file on terminal to check your assignment with the command:
calico assignment1_test.yaml --debug

Notice: You can determine how your outputs should be by examining gameplay and calico. In calico yaml, repetitive outputs are shortened or omitted to make yaml readable. First time occurence of the outputs are written as whole, while subsequent same kind of outputs are shortened.