BLG 252E - Object Oriented Programming
Assignment #3

Due: May 24th, 2021 23:59

"War… war never changes!"

- Fallout (1997)

Introduction

For this assignment, you are expected to implement classes for a text-based strategy game called **"The Warmonger: A New Dimension"**.

In this game, players play as a war merchant trying to keep a war in balance to profit themselves the most. During the game, players can sell weapons and armors to different factions by considering their stats and attack behavior. Number of weapons and armors that players can sell is limited in every turn; therefore, they need to consider how much equipment they need to sell without disturbing the balance and losing max profit.

In each turn, players can

- see the information of faction(s);
- sell weapons and armors to all the factions at war;
- end the turn(day) after selling as much as equipment they want within the limit;
- end the current game they are playing;
- quit the game.

To better understand the game, you can watch **the gameplay video**.

For any issues regarding the assignment, please ask in **Ninova message board** or contact **Uğur Önal (onalug@itu.edu.tr)**.

Implementation Notes

The implementation details below are included in the grading:

1. **main.cpp** file is provided to you in the homework files. Do not change it since we will evaluate your homework with it.
2. Please follow a consistent coding style (indentation, variable names, etc.) with comments.
3. You are not allowed use STL containers.
4. You should implement any getter and setter methods when they are needed. You **should use the method names** used in **main.cpp**.

5. 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.
6. Make use of constants and call by reference appropriately
7. Make sure that you used proper access modifiers for attributes in terms of **public**, **protected** and **private**.
8. Print out your messages to the terminal. Do not create a .txt file to print into.
9. Since your code will be tested automatically, make sure that your outputs match with sample scenario for given inputs.

Submission Notes

1. Your program should compile and run on Linux environment using:

   "g++ -std=c++11 main.cpp Faction.cpp Orcs.cpp Dwarves.cpp Elves.cpp Merchant.cpp -o TheWarmonger.out"

   If you used any flags during compilation, please notify them. You can test your program on ITU's Linux Server using **SSH** protocol. Do not use any pre-compiled header files or STL commands. Be sure that you have included all your header files.
2. You should compress your files into an archive file named **"<your_student_number>.zip"**. Do not include any executable or project files in the archive file. You should only submit necessary files. Also, write your name and ID on the top of each document that you will upload.
3. Submissions are made through **only** the Ninova system and have a strict deadline. Assignments submitted after the deadline will not be accepted.
4. 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: **Faction, Orcs, Dwarves, Elves** and **Merchant**.

1.1 Faction

Private Attributes:

Faction class ***has***

- a **name** (e.g., "Default"),
- a Faction pointer representing **first enemy**,
- a Faction pointer representing **second enemy**,
- a **number of units** (e.g., 50),
- an **attack point** (e.g., 30),

- a **health point** (e.g., 150),
- a **unit regeneration number** (e.g., 10),
- a **total health** which is the multiplication of the number of units and health point (e.g. 8000),
- a flag controlling if the faction **is alive** (e.g., true).

Methods:

1. **Constructor:** the constructor should ***optionally*** take the name, number of units, attack point, health point, unit regeneration number attributes. It sets the total health value and makes the alive status true.
2. **AssingEnemies:** a method to assign enemies for the faction. Enemy factions are assigned to the faction to manipulate during the game.
3. **PerformAttack**: a pure virtual method to perform attack by the faction's attack behavior.
4. **ReceiveAttack**: a pure virtual method to receive attack from an enemy faction and act on it by the Faction's damage behavior.
5. **PurchaseWeapons**: a pure virtual method to receive weapon from the merchant and return the profit.
6. **PurchaseArmors** a pure virtual method to receive armor from the merchant and return the profit.
7. **Print:** a virtual method to print Faction information in the format below.

```
Faction Name:          <name>
Status:                <"Alive" or "Defeated">
Number of Units:       <numOfUnits>
Attack Point:          <attackPoint>
Health Point:          <healthPoint>
Unit Regen Number:     <unitRegen>
Total Faction Health: <totalHealth>
```

8. **EndTurn:** a method to set the Faction's information at the end of the turn. This method updates the number of units, total health and alive status of the faction.
    - If the number of units is less than 0 at the end of the turn, method should set it to 0.
    - If the number of units is 0, it should set alive status to false.

Warning: If you need any other methods for Faction class, you are expected to implement them as an inline function.

1.2 Orcs

Private Attributes:

Orcs class ***is*** a Faction class. It uses all of the private attributes of the Faction class and does not have any other private attributes.

Methods:

1. **Constructor:** the constructor behaves exactly like Faction class constructor.
2. **PerformAttack:** a method to perform attack to enemy factions.
    - If there is only one enemy faction left alive, attacks the enemy faction with all of the units with attack point.
    - If both enemy factions are alive, attacks Elves with 70% of its units with attack point and attacks Dwarves with rest of the units with attack point.
3. **ReceiveAttack:** method to receive attack from an enemy faction. Reduces number of units by total damage received divided by health point.
    - If attacking faction is Elves, incoming attack point is reduced to 75% of its original value.
    - If attacking faction is Dwarves, incoming attack point is reduced to 80% of its original value.
4. **PurchaseWeapons:** method to purchase weapons from the merchant. Increases attack point by double of weapon points bought and returns 20 gold for each weapon point bought.
5. **PurchaseArmors:** method to purchase armors from the merchant. Increases health point by triple of armor points bought and returns 1 gold for each armor point bought.
6. **Print:** method to print Faction information. Prints the battle cry of Orcs: **"Stop running, you'll only die tired!"** in quotes then prints the information as same as the Faction class.

1.3 Dwarves

Private Attributes:

Dwarves class ***is*** a Faction class. It uses all the private attributes of the Faction class and does not have any other private attributes.

Methods:

1. **Constructor:** the constructor behaves exactly like Faction class constructor.
2. **PerformAttack:** a method to perform attack to enemy factions.
    - If there is only one enemy faction left alive, attacks the enemy faction with all of the units with attack point.
    - If both enemy factions are alive, attacks each enemy faction with half of its units with attack point.
3. **ReceiveAttack:** method to receive attack from an enemy faction. Reduces number of units by total damage received divided by health point.
4. **PurchaseWeapons:** method to purchase weapons from the merchant. Increases attack point by weapon points bought and returns 10 gold for each weapon point bought.
5. **PurchaseArmors:** method to purchase armors from the merchant. Increases health point by double of armor points bought and returns 3 gold for each armor point bought.
6. **Print:** method to print Faction information. Prints the battle cry of Dwarves: **"Taste the power of our axes!"** in quotes then prints the information as same as the Faction class.

1.4 Elves

Private Attributes:

Elves class ***is*** a Faction class. It uses all the private attributes of the Faction class and does not have any other private attributes.

Methods:

1. **Constructor:** the constructor behaves exactly like Faction class constructor.
2. **PerformAttack:** a method to perform attack to enemy factions.
    - When attacking Dwarves, attack point is increased to 150%.
    - If there is only one enemy faction left alive, attacks the enemy faction with all of the units with attack point.
    - If both enemy factions are alive, attacks Orcs with 60% of its units with attack point and attacks Dwarves with rest of the units with attack point.
3. **ReceiveAttack:** method to receive attack from an enemy faction. Reduces number of units by total damage received divided by health point.
    - If attacking faction is Orcs, incoming attack point is increased to 125% of its original value.
    - If attacking faction is Dwarves, incoming attack point is reduced to 75% of its original value.
4. **PurchaseWeapons:** method to purchase weapons from the merchant. Increases attack point by double of weapon points bought and returns 15 gold for each weapon point bought.
5. **PurchaseArmors:** method to purchase armors from the merchant. Increases health point by quadruple of armor points bought and returns 5 gold for each armor point bought.
6. **Print:** method to print Faction information. Prints the motto of Elves: **"You cannot reach our elegance."** in quotes then prints the information as same as the Faction class.

1.5 Merchant

Private Attributes:

Merchant class ***has***

- a Faction pointer representing **first faction** in the battlefield,
- a Faction pointer representing **second faction** in the battlefield,
- a Faction pointer representing **third faction** in the battlefield,
- a fixed **starting weapon point** (e.g., 10),
- a fixed **starting armor point** (e.g., 10),
- a **revenue** (e.g., 360),
- an **weapon point** left for the day (e.g., 5),
- an **armor point** left for the day (e.g., 6).

Methods:

1. **Constructor:** the constructor should ***optionally*** take the starting weapon point and starting armor point attributes. It sets the weapon point and armor point and makes the revenue 0.

2. **AssingFactions:** a method to assign factions in the battlefield. Factions in the battlefield are assigned to the merchant to manipulate during the game.

3. **SellWeapons**: a method to sell weapons to a faction. Sells weapons to faction named with weapon points given, notifies players with **“Weapons sold!”** and returns true**.**
    - If faction tried to sell weapons is not alive, notifies players with **“The faction you want to sell weapons is dead!"** and returns false.
    - If weapon points tried to sell is greater than the available, notifies players with **“You try to sell more weapons than you have in possession."** and returns false.

4. **SellArmors**: a method to sell armors to a faction. Sells armors to faction named with armor points given, notifies players with **“Armors sold!”** and returns true**.**
    - If faction tried to sell armors is not alive, notifies players with **“The faction you want to sell armors is dead!"** and returns false.
    - If armor points tried to sell is greater than the available, notifies players with **“You try to sell more armors than you have in possession."** and returns false.
5. **EndTurn**: a method to set the Merchant’s information at the end of the turn. Sets the weapon and armor points to the starting values set in the constructor.

Notice: Check the gameplay **here** and implement other necessary functions accordingly, e.g., to print menus or getter and setters. Also, be aware that **main.cpp** is provided you in the homework files. Read the **main.cpp** carefully and use the function names and parameters used in **main.cpp** when implementing your classes.