BLG 252E - Object Oriented Programming
Assignment #1

Due: March 17th 23:59

*"**Jessie**: Team Rocket blasting off at the speed of light.*
***James**: Surrender now or prepare to fight.*
***Meowth**: Meowth. That's right."*

-Pokémon (1997)

Introduction

For this assignment, you are expected to implement Satoshi Tajiri's great game "Pokémon" as a text-based strategy game. For any issues regarding the assignment, you can ask your questions on Ninova Message Board. Before writing your question, please check whether your question has been asked. You can also contact T.A. Barış Bilen (**bilenb20@itu.edu.tr**).

Implementation Notes

The implementation details below are included in the grading:

1. You are not allowed to use STL containers.
2. You may (and should) implement any getter methods when needed.
3. Make sure that there is no memory leak in your code.
4. Define required functions and object references as **const** in all cases where necessary.
5. Make sure your outputs match the sample scenario for given inputs. You will be provided with a Calico file to check your assignment's output.

1 Implementation Details

You are expected to implement four classes: **Pokemon**, **Pokedex**, **Player**, and **Enemy**.

1.1 Pokemon

Private Attributes:

Pokemon class has a **Name** (e.g., Pikachu) a **Health Point (HP)(e.g., 100)**, and a **Attack Value (Atk)(e.g., 20)**.

Methods:

1. **Constructor(s):** You are expected to write 3 constructors. The first one is a default constructor, which will initialize all values, the second one will construct a Pokemon, it will take a string for the name and an integer for the attack value of the Pokemon, and it also has to set the default Pokemon's hp to 100 and the third constructor will be a copy constructor, which will take a Pokemon class object and copy its contents.
2. **Getter(s):** You can (and should) implement getter methods for all attributes.

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

1.2 Pokedex

Private Attributes:

The Pokedex class has a **Pokedex Array** (An array that can hold Pokemon object) and an int **value** (Keeps Track of the Position)

Methods:

1. **Constructor(s):** The constructor should initialize the value as 0.
2. **updatePokedex:** Method to add new pokemons to the pokedexArray. Only new pokemons are allowed to be added. Check for duplicates!
3. **printPokedex:** Method to print Pokemon names from pokedexArray.

1.3 Player

Private Attributes:

Player class has a **name** (e.g. Ash), **Pokemon Number** (Holds Number of Pokemon's), **Pokeball Number** (Holds Number of Pokemoballs), **Badge Number** (Holds Number of Badges), Player's **Pokemon** (Object of Pokemon Class) and **Player's Pokedex** (Object with Pokedex Class).

Methods:

1. **Constructor(s):** You are expected to write two constructors. The first one is a default constructor that will initialize all values, the second one will construct a new player with a given **name** (string), and **Pokemon** (object).
2. **showPokemonNumber:** Method to return pokemonNumber.
3. **showPokeballNumber:** Method to return pokeballNumber.
4. **showBadgeNumber:** Method to return badgeNumber.

5. **getPokemon:** Method to return playerPokemon object.
6. **battleWon:** Method to update badgeNumber and pokeballNumber. Every time you win a battle your badge number goes up by 1 and your pokeBallNumber goes up by 3.
7. **catchPokemon:** Method to update pokemonNumber and pokeballNumber. Every time you catch a Pokemon your pokemonNumber goes up by 1 and your pokeballNumber goes down by 1.

1.4 Enemy

Private Attributes:

The Enemy class has a **name** (e.g., Misty) and an enemy's **Pokemon** (Object with Pokemon Class).

Methods:

1. **Constructor(s):** You are expected to write two constructors. The first one is a default constructor that will initialize all values, the second one will construct a new enemy with a given **name** (string), and **Pokemon** (object).
2. **getPokemon:** Method to return enemyPokemon object.
3. **getName:** Method to return the name of the enemy.

Notice: You have already been given a skeleton code. Please examine it before writing your own codes. Skeleton code for classes (header file) includes all necessary classes and variables. You are expected to implement the methods of these classes.

1.5 Necessary Methods You Need to Implement in Main.cpp:

1. **readEnemyNames:** Method to read enemyNames.txt file and create a dynamic string array for the enemy names.
2. **readPokemonNames:** Method to read pokemonNames.txt file and create a dynamic string array for the Pokemon names.
3. **characterCreate:** Method to create a character. You are expected to create a player with the chosen name and Pokemon. Returns a player class.
4. **fightEnemy:** Method for fighting with an enemy. You are expected to create a Pokemon and an enemy, update the pokedex if the created Pokemon is not in the pokedex and simulate a fight. The player has two options; either they can fight or runaway. Fighting will happen in turns. Every turn, the player will have this option. With each turn, both Pokemon will decrease their opponent's hp according to their attack value. The first Pokemon to reach 0 or lower than 0 health will lose. The player and enemy pokemon should start from full health(100hp) every time you enter a fight.
5. **catchPokemon:** Method for catching a pokemon. You are expected to create a new Pokemon and update pokedex if the created Pokemon is not in the Pokemon. The player has two options; either they can catch the Pokemon or runaway.

Warning: To make your life easier, set player Pokemon's atk value to 20 and hp value to 100 and set enemy Pokemon's atk value to 10 and hp value to 100. So with these values' the player should always win the fight in 5 turns.

Notice: Select a new pokemon and enemy every time fightEnemy or catchPokemon functions called.

Notice: All needed functions were given to you in skeleton code. You are expected to implement the methods inside these defined functions.

2 Gameplay

In this game you can fight with enemies to win badges and Pokeballs or catch Pokemons using Pokeballs to expand your Pokemon collection. The possible actions in the game are as follows:

1. **Fight for a badge:** Brings an enemy for you to fight and win a badge.
2. **Catch a Pokemon:** Brings a pokemon for you to catch.
3. **Number of Pokemons:** Shows how many Pokemons you have.
4. **Number of Pokeballs:** Shows how many Pokeballs you have.
5. **Number of Badges:** Shows how many badges you earned.
6. **Pokedex:** Shows the different pokemons that you see in your adventure.
7. **Exit:** Exits the game.

Notice: You can check the gameplay **here**.

3 How to compile, run, and test your code

If you want to compile and run the provided code on a terminal, you can use these commands:

g++ -Wall -Werror main.cpp pokemon.cpp pokemon.h -o assignment1
./assignment1 enemyNames.txt pokemonNames.txt

Notice: The order of the text files is important, do not change it!

You can (and should) run the Calico and Valgrind on a terminal to check your assignment with the command:

calico assignment1_test.yaml −−debug

valgrind –tool=memcheck –leak-check=full –show-leak-kinds=all ./bin/main |& tee valgrind_log.txt

Notice: Valgrind is a tool for memory debugging and memory leak detection. It is pre-installed in your given Ubuntu environment (Docker). For more information, click here. Make sure that all heap blocks are freed in your code and that no leaks are possible.

4 Submission

Submit your **main.cpp**, **pokemon.cpp** and **pokemon.h** files to Ninova. Before submitting please make sure you are passing all cases in the Calico file.