Text-based Pokémon Game

1 - Introduction

In this project, you will implement a simple text-based game using C++. This hands-on experience will enhance your understanding of C++ classes and the standard string and vector libraries. Also, you will get experience developing a program based on a given specification.

This program is structured using the following four C++ classes:

- **Item** : Represents collectible items the player can obtain.
- **Enemy**: Represents Pokémon the player must battle and defeat.
- **Place** : Serves as a game stage, using vectors to contain Item and Enemy objects.
- **Player** : Represents the user, interacting with the game's world and mechanics.

2 - Tasks

Implement the four classes (remember, **it is important to use public member functions to access private member variables**). While member functions are typically implemented in a separate source file, we will define them directly within the class definitions this time. Details about the members of these classes are provided on the following pages.

The main function has already been completed. It initializes the game by creating the stages (see the table below) and the player, then repeatedly prompts the player for actions in a loop.

For reference, this program is not fully complete. For example, while the player can attack enemies, the enemies do not retaliate. The player can collect items but cannot use them. Expanding and refining the program could present an interesting challenge.

	Stage 1: Buffalo	Stage 2: Rochester	Stage 3: Syracuse
Item	Aux Power	Berry Juice Aspear Berry	None
Enemy	Arbok Weezing	None	Meowth

2.1 – Item class

- **Private** member variables

Variable name	Type	Description
`name`	`string`	Name of the item
`desc`	`string`	Description of the item

- **Public** member functions

Function name	Return type	Parameter (name, type)		Description
`Item` (constructor)	N/A	`n`	`string`	Initializes the values of `name` and `desc`
		`d`	`string`	
`getName` (getter)	`string`	N/A	`void`	Returns the current value of `name`
`getDesc` (getter)	`string`	N/A	`void`	Returns the current value of `desc`

2.2 – Enemy class

- **Private** member variables

Variable name	Type	Description
`name`	`string`	Name of the enemy
`health`	`int`	Hit points of the enemy
`attackPower`	`int`	Attack power of the enemy

- **Public** member functions

Function name	Return type	Parameter (name, type)		Description
`Enemy` (constructor)	N/A	`n`	`string`	Initializes the values of `name`, `health`, and `attackPower`
		`h`	`int`	
		`a`	`int`	
`getName` (getter)	`string`	N/A	`void`	Returns the current value of `name`
`getHealth` (getter)	`int`	N/A	`void`	Returns the current value of `health`
`getAttackPower` (getter)	`int`	N/A	`void`	Returns the current value of `attackPower`
`setHealth` (setter)	`void`	`h`	`int`	Subtracts the parameter value from `health`

2.3 – Place class

- **Private** member variable

Variable name	Type	Description
`desc`	`string`	Description of the place

- **Public** member variables (vectors)
    - Note: these vectors are empty when they are created.

Vector name	Element type	Description
`itemList`	`Item`	Contains Item objects
`enemyList`	`Enemy`	Contains Enemy objects
`placeList`	`Place*`	Contains pointers pointing to Place objects

- **Public** member functions

Function name	Return type	Parameter (name, type)		Description
`Place` (constructor)	N/A	`d`	`string`	Initializes the value of `desc`.
`getDesc` (getter)	`string`	N/A	`void`	Returns the current value of `desc`.
`addItem`	`void`	`i`	`Item`	Adds the passed Item object to `itemList`.
`addEnemy`	`void`	`e`	`Enemy`	Adds the passed Enemy object to `enemyList`.
`addPlace`	`void`	`p`	`Place*`	Adds the passed pointer to `placeList`.
`printItems`	`void`	N/A	`void`	(1-1) If `itemList` is empty, it prints the message "- No items." (1-2) Otherwise, it prints the message "- Items:" then " - [*item name*]: [*item description*]" for each item. (hint: you may need a range-based for loop) Refer to the result.txt file for output formatting.

`printEnemies`	`void`	N/A	`void`	(1-1) If `enemyList` is empty, it prints the message "- No enemies." (1-2) Otherwise, it prints the message "- Enemies:" then " - [*enemy name*]: (Health: [*enemy health value*], Attack Power: [*enemy attack power value*])" for each enemy. (hint: you may need a range-based for loop) Refer to the result.txt file for output formatting.

2.4 – Player class

- **Private** member variables

Variable name	Type	Description
`name`	`string`	Name of the player
`health`	`int`	Hit points of the player
`attackPower`	`int`	Attack power of the player

- **Private** member variable (vector)
    - Note: this vector is empty when it is created.

Vector name	Element type	Description
`inventory`	`Item`	Contains Item objects

- **Public** member variable

Variable name	Element type	Description
`currentPlace`	`Place*`	Stores the starting memory address of the Place object

- **Public** member functions

<table>
<tr><th>Function name</th><th>Return type</th><th colspan="2">Parameter (name, type)</th><th>Description</th></tr>
<tr><td rowspan="3"><code>Player</code>
(constructor)</td><td rowspan="3">N/A</td><td><code>n</code></td><td><code>string</code></td><td rowspan="3">Initializes the values of <code>name</code>, <code>health</code>, <code>attackPower</code>, and <code>currentPlace</code>. The initial value for <code>currentPlace</code> is <code>nullptr</code>.</td></tr>
<tr><td><code>h</code></td><td><code>int</code></td></tr>
<tr><td><code>a</code></td><td><code>int</code></td></tr>
<tr><td><code>pickUpItem</code></td><td><code>void</code></td><td><code>i</code></td><td><code>Item</code></td><td>(1) Adds the passed Item object to <code>inventory</code>.
(2) Prints the message “[player name] picks up [item name].”

Refer to the result.txt file for output formatting.</td></tr>
<tr><td><code>attackEnemy</code></td><td><code>void</code></td><td><code>e</code></td><td><code>Enemy&</code></td><td>(1) Prints the message “[player</td></tr>
</table>

			(think about why pass-by-reference is used here)	*name*] attacks [*enemy name*] with power [*player attack power value*]". (2) Updates the `health` value of the enemy. (3-1) If the `health` value of the enemy is less than or equal to 0, it prints the message "[*enemy name*] has been defeated!" (3-2) Otherwise, it prints the message "[*enemy name*] has [*enemy health value*] health left." Refer to the result.txt file for output formatting.
`displayInventory`	`void`	N/A	`void`	(1-1) Prints the message "Inventory is empty." if `inventory` is empty. (1-2) Prints the message "Inventory:" then "- [*item name*]: [*item description*]" for each item, otherwise. Refer to the result.txt file for output formatting.
`moveToPlace`	`void`	`p`	`Place*`	(1) Updates the value of `currentPlace` to the passed pointer value (to point to the next place). (2) Prints the message "[*player name*] moves to [(*next) place description*]". (hint: you may need to use -> to access a Place object.) Refer to the result.txt file for output formatting.

3 - Compile and Test

Type the following commands on Terminal.

```
g++ -o main *.cpp *.h
./main
```

Check if your program functions correctly with various inputs.

You can use the following automatic checking commands:

```
chmod 777 run.sh
./run.sh
```

You will see the "PASS" message if your printed output in log.txt matches exactly with the content in result.txt for the inputs provided in testInput.txt.

4 - Submit

- Please upload your **game.h** files to myCourses > Assignments > Project2.
- Do not submit *.docx, *.pdf, *.txt, or *.zip file.
- Do not change other files.
- Only one student from each team can submit.

5 - Grading

- [90 pts]
    - [15 pts] **Item** class
    - [15 pts] **Enemy** class
    - [30 pts] **Place** class
    - [30 pts] **Player** class
- [10 pts] - program execution
    - **The program execution portion of the grade is ALL or nothing.**