CMSC 202 Spring 2020

Project 3 – Pokémon

Assignment: Project 3 – Pokémon

Due Date: Thursday, April 2nd@ 8:59pm

Value: 80 points

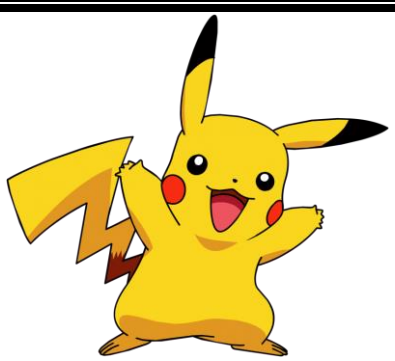
1. Overview

In this project you will:

- Implement a linked-list data structure,
- Use dynamic memory allocation to create new objects,
- Practice using C++ class syntax,
- Practice object-oriented thinking.

2. Background

For this project, we are going to be designing a simple implementation of the famous Pokémon franchise. If you are not familiar with Pokémon, it is a game where you act as a trainer of pocket monsters (Pokémon). You carefully choose the roster finding the perfect Pokémon to matchup against the computer. Finally, after you build your roster of Pokémon, you can battle them against the computer!



For this project, each node in our linked list is going to be a Pokémon. Each Pokémon has a name, an index, a type, and a type that it is strong against. The name, type, and strong are all strings and the index are an integer.

Our linked list is going to be a little bit special because it is designed to hold and manage Pokémon. We can insert Pokémon into the linked list, we can remove Pokémon from a specific location in our linked list, we can Attack Pokémon, we can Swap Pokémon, and a variety of other things.

Finally, the game itself will allow us to battle our roster of Pokémon against a randomly assigned roster of enemy Pokémon. The skill in this game is to find a preferable roster that can defeat the enemies. Each Pokémon has a strength and identifying that strength is required to defeat the enemies – especially when the enemies are stronger than we are!

3. Assignment Description

Your assignment is to build an application that can allow us to battle two linked lists of Pokémon.

1. The input file will hold some number of Pokémon with the following order: index (int), name (string), type (string), and strong (string).
2. All Pokémon should be inserted at the end of the linked list.
3. The **`m_list`** holds all of the Pokémon for the entire game. A *pocket* is a linked list representing a team. The **`m_userPocket`** holds the user's Pokémon and the **`m_enemyPocket`** holds the enemy Pokémon.
4. The data file (**`proj3_data.txt`**) is loaded via command line argument (included in the provided **`makefile`** and **`proj3.cpp`**).

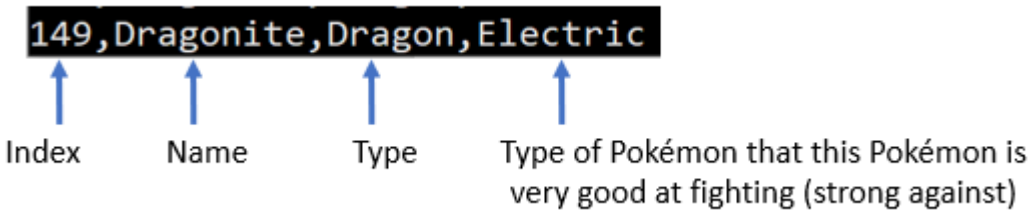


Figure 1. Input File Details

5. When a Pokémon is added to either the **`m_userPocket`** or **`m_enemyPocket`**, it is permantently removed from **`m_list`**.
6. The user's pocket will be populated by asking the user to choose Pokémon for their team. After the user has chosen their pocket, the enemy's pocket will be randomly populated from the remaining Pokémon.
7. Initially, each Pokémon team (user and enemy) will have 5 Pokémon. As a Pokémon is defeated, it is removed from the team. The battle (and game) ends when there are no Pokémon left on one of the teams.
8. All user inputs will be assumed to be the correct data type. For example, if you ask the user for an integer, they will provide an integer.
9. Regardless of the sample output below, all user input must be validated. If you ask for a number between 1 and 5 with the user entering an 8, the user should be re-prompted.
10. The general game flow:
    a. Game constructor – Initializes each of the game variables
    b. Game Start - Loads File (populates **`m_list`**), Choose user team, and populates enemy team. Calls Battle.
    c. Battle – Displays each of the teams initially. Manages each of the rounds of the battle. Allows user to attack, choose a new Pokémon to attack, or forfeit (automatically loses to enemy). Checks to see if either team still has members. The head of the **`m_userPocket`** and the **`m_enemyPocket`** always fight each other.
    d. Upon exit, nothing is saved

4. Requirements:

Initially, you will have to use the following files **`Pokemon.h`**, **`PokemonList.h`**, **`Game.h`**, **`makefile`**, **`proj3.cpp`**, and a simple test file. You can copy the files from Prof. Dixon's folder at:

`/afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj3`

To copy it into your project folder, just navigate to your project 3 folder in your home folder and use the command:

```
cp /afs/umbc.edu/users/j/d/jdixon/pub/cs202/proj3/* .
```

Notice the trailing period is required (it says copy it into this folder).

- The project must be completed in C++. You may not use any libraries or data structures that we have not learned in class. Libraries we have learned include **`<iostream>`**, **`<fstream>`**, **`<iomanip>`**, ~~**`<vector>`**~~, **`<cstdlib>`**, **`<time.h>`**, **`<cmath>`** and **`<string>`**. You should only use **`namespace std`**. You should **not** use vectors in this project.
- You must use the function prototypes as outlined in the **`Pokemon.h`**, **`PokemonList.h`**, **`and Game.h`** header file. Do not edit the header files. You need to code **`Pokemon.cpp`**, **`PokemonList.cpp`**, **`and Game.cpp`**.
- The **`PokemonList`** is the linked list class for this project. There are three PokemonLists in the game – **`m_list`** (which is the list of all Pokémon in the game), **`m_userPocket`** (which is the user's team), and **`m_enemyPocket`** (which is the computer/enemy team). Here are some general descriptions of each of the functions:
  - **`PokemonList()`** – The constructor creates a new empty linked list. **`m_head`** is **`nullptr`** and **`m_size`** is zero.
  - **`~PokemonList()`** – The destructor de-allocates any dynamically allocated memory.
  - **`InsertEnd()`** – This inserts a node into the end of the **`PokemonList`**. There is no **`m_tail`** so you must iterate over the list to the end!
  - **`Transfer(choice, Pocket)`** – Transfers a Pokémon from one linked list to another (only **`m_list`** to one of the teams in this

case). It iterates until the choice is the same as the index of the Pokémon. Creates a new Pokémon and inserts it into the destination linked list (user or enemy pocket). Removes the Pokémon from the source.

- **`Remove(choice)`** – Iterates until the choice is the same as the index of the Pokémon. Removes that Pokémon. Don't forget about the special cases when the nodes are in the beginning, middle or end of the list.
- **`GetHead()`** – Returns the head of the list.
- **`GetSize()`** – Returns the number of nodes in the list.
- **`Display()`** – Displays information about each Pokemon in the list. If the list is empty, it indicates this. Should be formatted and show index, name, type, and health in that order. Something like this:

  ```
  Index: 151 Name: Mew Type: Psychic   Health: 9
  ```

- **`Attack(PokemonList)`** – Manages each attack from one Pokémon to another. Each Pokémon does extra damage to a target type (this is the strong type).
  - For example, 147,Dratini,Dragon,Electric – in this case, a Dragon type does 5pts of damage to any Pokémon of type Electric.
- Here are the expected damages:

  `Normal Enemy Attack on User = 3pts`

  `Normal User Attack on Enemy = 2pts`

  `Attack on a vulnerable Pokémon = 5pts`

  Otherwise, updates health on the target Pokémon (user Pokémon attacks first – if user defeats enemy there is no enemy attack).
- **`SwapPokemon(PokemonList)`** – Asks the user to switch a Pokémon from somewhere in their pocket (**`m_userPocket`**) to the front of the list. Used to decide who is going to attack next. Be careful – this is kind of tricky! Don't forget to take into account if the user chooses a Pokémon that is already the head.

- **`Exist(choice)`** – Iterates through the list to see if the choice equals the index of a Pokémon. If the choice matches, return true else return false.

- The **`Game`** class manages the application and populates the various **`PokemonList`**. **`Game.cpp`** file that are prototyped in **`Game.h`**.
  - **`Game(filename)`** – The constructor that sets **`m_filename`**, dynamically allocates the **`PokemonList`** for **`m_list, m_userPocket, and m_enemyPocket.`**
  - **`~Game()`** – The destructor de-allocates any dynamically allocated memory (each list).
  - **`LoadFile()`**– The **`LoadFile`** function loads all of the Pokemon from the source file to **`m_list.`** You should not need to use stoi or stod in this function. The index should be an integer and name, type, and strong should each be strings. Default to 9 health.
  - **`Menu()`** – Allows the user to choose to attack, swap, or forfeit. Returns the user's choice.
  - **`Battle()`** – Displays both the user pocket and the enemy pocket. Then displays information about the Pokémon at the head of both the user pocket and the enemy pocket. Then posts the menu and responds accordingly. If the result of the attack is that a Pokémon is defeated, this function will swap and remove Pokémon for that list. If the **`userPocket`** is empty (user loses), return 2. If the **`enemyPocket`** is empty (computer loses), return 1.
  - **`ChooseTeam()`** – Lists all available Pokémon in **`m_list`** and allows the user to choose Pokémon for their team. Transfers each chosen Pokémon to **`m_userPocket`**.
  - **`Start()`** – Calls **`LoadFile`**, **`ChooseTeam`**, randomly populates **`m_enemyPocket`** from remaining Pokémon and returns result of **`Battle`**.

5. Sample Input and Output

5.1. Sample Run

An additional file named **`proj3_sample1.txt`** is available in

A normal run of the compiled code would look like this with user input highlighted in blue:

```
[jdixon@linux4 proj3]$ make run
./proj3 proj3_data.txt
Welcome to the game
Here is a list of Pokemon you can choose from:
------------------------------
Index: 1   Name: Bulbasaur    Type: Grass     Health: 9
Index: 2   Name: Ivysaur      Type: Grass     Health: 9
**Index 3 to 149 Removed for Space**
Index: 150 Name: Mewtwo       Type: Psychic   Health: 9
Index: 151 Name: Mew          Type: Psychic   Health: 9
------------------------------
Pick a Pokemon by enter the index (5 left):
```

Here is an example where the user is starting a battle and they swap their Pokémon for a better matchup (the real way you win this game!).

```
Pick a pokemon by enter the index (1 left): 5
------------------------------
Print player pocket
Index: 1   Name: Bulbasaur    Type: Grass     Health: 9
Index: 2   Name: Ivysaur      Type: Grass     Health: 9
Index: 3   Name: Venusaur     Type: Grass     Health: 9
Index: 4   Name: Charmander   Type: Fire      Health: 9
Index: 5   Name: Charmeleon   Type: Fire      Health: 9
------------------------------
Print cpu pocket
Index: 143 Name: Snorlax      Type: Normal    Health: 9
Index: 77  Name: Ponyta       Type: Fire      Health: 9
Index: 117 Name: Seadra       Type: Water     Health: 9
Index: 23  Name: Ekans        Type: Poison    Health: 9
```

```
------------------------------
Round 1:
CPU's Pokemon: Snorlax (Normal:9 health)
Your Pokemon: Bulbasaur (Grass:9 health)
------------------------------
Menu:
1. Attack
2. Swap
3. Forfeit
2
------------------------------
Which Pokemon would you like to choose? (Enter the index#)
Index: 1   Name: Bulbasaur    Type: Grass     Health: 9
Index: 2   Name: Ivysaur      Type: Grass     Health: 9
Index: 3   Name: Venusaur     Type: Grass     Health: 9
Index: 4   Name: Charmander   Type: Fire      Health: 9
Index: 5   Name: Charmeleon   Type: Fire      Health: 9
4
------------------------------
Round 2:
CPU's Pokemon: Snorlax (Normal:9 health)
Your Pokemon: Charmander (Fire:9 health)
------------------------------
Menu:
1. Attack
2. Swap
3. Forfeit
```

Here is an example where the user forfeits.

```
Menu:
1. Attack
2. Swap
```

```
3. Forfeit
3
------------------------------
CPU won!!!
```

Here is an example with some validation.

```
Round 1:
CPU's Pokemon: Tangela (Grass:9 health)
Your Pokemon: Bulbasaur (Grass:9 health)
------------------------------
Menu:
1. Attack
2. Swap
3. Forfeit
0
Please enter a valid choice: 4
Please enter a valid choice: 5
Please enter a valid choice: 2
------------------------------
Which Pokemon would you like to choose? (Enter the index#)
Index: 1   Name: Bulbasaur    Type: Grass     Health: 9
Index: 23  Name: Ekans        Type: Poison    Health: 9
Index: 4   Name: Charmander   Type: Fire      Health: 9
Index: 5   Name: Charmeleon   Type: Fire      Health: 9
Index: 6   Name: Charizard    Type: Fire      Health: 9
7
This is not a valid index, please try again
1
choice is head
------------------------------
Round 2:
CPU's Pokemon: Tangela (Grass:9 health)
```

```
Your Pokemon: Bulbasaur (Grass:9 health)
-----------------------------
Menu:
1. Attack
2. Swap
3. Forfeit
```

6. Compiling and Running

Because we are using a significant amount of dynamic memory for this project, you are required to manage any memory leaks that might be created. For a linked list, this is most commonly related to the dynamically allocated nodes. Remember, in general, for each item that is dynamically created, it should be deleted using a destructor.

One way to test to make sure that you have successfully removed any of the memory leaks is to use the **`valgrind`** command.

Since this project makes extensive use of dynamic memory, it is important that you test your program for memory leaks using **`valgrind`**:

`valgrind ./proj3`

Note: If you accidently use **`valgrind make run`**, you may end up with some memory that is still reachable. Do not test this – test using the command above where you include the input file. The **`makefile`** should include **`make val1`** (which is ok).

If you have no memory leaks, you should see output like the following:

```
==5606==
==5606== HEAP SUMMARY:
==5606==     in use at exit: 0 bytes in 0 blocks
==5606==   total heap usage: 87 allocs, 87 frees, 10,684 bytes
allocated
==5606==
==5606== All heap blocks were freed -- no leaks are possible
==5606==
==5606== For counts of detected and suppressed errors, rerun
with: -v
```

```
==5606== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6
from 6)
```

The important part is “in use at exit: 0 bytes 0 blocks,” which tells me all the dynamic memory was deleted before the program exited. If you see anything other than "0 bytes 0 blocks" there is probably an error in one of your destructors. We will evaluate this as part of the grading for this project.

Additional information on **`valgrind`** can be found here: http://valgrind.org/docs/manual/quick-start.html

Once you have compiled using your **`makefile`**, enter the command **`./proj3`** to run your program. You can use **`make val1`** to test each of the input files using **`valgrind`** (do NOT use **`valgrind make run`**!). They have differing sizes. If your executable is not **`proj3`**, you will lose points. It should look like the sample output provided above.

7. Completing your Project

When you have completed your project, you can copy it into the submission folder. You can copy your files into the submission folder as many times as you like (before the due date). We will only grade what is in your submission folder.

For this project, you should submit these files to the **`proj3`** subdirectory:

`proj3.cpp` — should be unchanged.

`Pokemon.h` — should be unchanged.

`Pokemon.cpp` – should include your implementations of the class functions.

`PokemonList.h` — should be unchanged.

`PokemonList.cpp` – should include your implementations of the class functions.

`Game.h` — should be unchanged.

`Game.cpp` – should include your implementations of the class functions.

As you should have already set up your symbolic link for this class, you can just copy your files listed above to the submission folder

e. cd to your project 3 folder. An example might be cd
`~/202/projects/proj3`

f. **`cp proj3.cpp Pokemon.h Pokemon.cpp Pokemon List.h PokemonList.h Game.h Game.cpp ~/cs202proj/proj3`**

You can check to make sure that your files were successfully copied over to the submission directory by entering the command

`ls ~/cs202proj/proj3`

You can check that your program compiles and runs in the **`proj3`** directory, but please clean up any **`.o`** and executable files. Again, do not develop your code in this directory and you should not have the only copy of your program here. Uploading of any **`.gch`** files will result in a severe penalty.

For additional information about project submissions, there is a more complete document available in Blackboard under "Course Materials" and "Project Submission."

IMPORTANT: If you want to submit the project late (after the due date), you will need to copy your files to the appropriate late folder. If you can no longer copy the files into the proj3 folder, it is because the due date has passed. You should be able to see your proj3 files, but you can no longer edit or copy the files in to your proj3 folder. (They will be read only)

- If it is 0-24 hours late, copy your files to **`~/cs202proj/proj3-late1`**
- If it is 24-48 hours late, copy your files to **`~/cs202proj/proj3-late2`**
- If it is after 48 hours late, it is too late to be submitted.