Homework 10 — Multiple Inheritance & Exceptions

For this assignment you will build a class inheritance structure for a small portion of the game of Pokemon. Your finished program will read an input file (the *Pokedex*) with facts about different Pokemon characters and determine the specific class of Pokemon for each entry. Also, if specified on the command line, the program will output information about the breeding possibilities and *for extra credit* the potential battle matchups. We use a somewhat quirky method to determine the class for each Pokemon in the input. We pass the collection of facts (e.g., *type(s)*, *egg group(s)*, *catch rate*, etc.) to each specialized Pokemon constructor in turn, and if the constructor doesn't fail, then we know that the list of facts is indeed that specific Pokemon. Remember, the only way for a constructor to fail is to throw an exception.

Pokemon Hierarchy

First, login to the homework submission server to retrieve your assigned list of 10 specific Pokemon. In addition to these Pokemon you will also create classes for the 15 "*Egg Groups*" (e.g., `Bug`, `Flying`, `Monster`, etc.) and the `Pokemon` base class. Note that each specific Pokemon can be correctly labeled by more than one of these classes; e.g., a `Pikachu` is also a `Pichu` (the pre-evolved form), and a `Fairy` and a `Field` (its two Egg Groups) and also a `Pokemon`. Here are a few reference links to help you learn about your assigned characters:

http://pokemondb.net/mechanics/egg-groups/field
http://bulbapedia.bulbagarden.net/
http://www.serebii.net/pokedex-xy/

Prepare a detailed diagram of the class hierarchy for these classes. You may omit from the diagram the Egg Groups that are not associated with any of your assigned Pokemon (but you will need to at least make empty classes for all Egg Groups in order for the code to compile). Draw arrows indicating all of the inheritance relationships. To receive full credit, the diagram should be legible, neat, and well organized, with no messy scribbles or cross outs, a consistent (up or down) orientation to the edges, and few or no arrow crossings. You do *not* need to include any of the member variables or member functions in this diagram. We are just looking for the high level relationships between class names. You should label the virtual inheritance paths (described on the next page). *Hint: Your diagram will include 16 or 17 classes and 3, 4, or 5 of the inheritance arrows will be labeled virtual (depending on your specific list assignment).*

Since many of you will draw this using pen/pencil & paper, this part of the homework is due in hardcopy to your graduate TA ***at the end of your normal lab section on Wednesday, December 9th***. Even if you choose to draw the diagram electronically, you are still required to print it out and submit the paper in your normal lab section. You may not use late days for this portion of the assignment.

Provided Code

We provide code for the I/O for this homework assignment. *You should not modify the provided code.* When you compile the code you will need to pass the list of your assigned Pokemon. For example:

```
g++ -DMY_POKEMON_LIST=\"ListXX.txt\" main.cpp pokemon.cpp -o pokemon.out -Wall -g
```

Where `XX` is your assigned list number. To run the program you will specify the input Pokedex and output file. The provided code uses C++ preprocessor macros to enumerate the constructors for the classes of your 10 specific Pokemon + 15 Egg Groups + the base class Pokemon (26 total classes). Note that the order that these constructors are called is important. We will attempt to make a `Pikachu` before making a `Flying`, before making a `Pokemon`. Additional arguments allow printing the breeding possibilities and the attack advantages or disadvantages (for extra credit) based on the attacker and defender types:

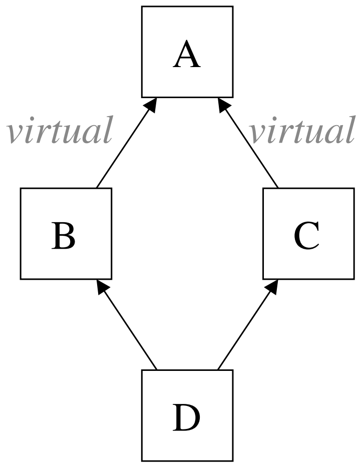
```
./pokemon.out -pokedex PokedexSmallXX.txt -output Output.txt
./pokemon.out -pokedex PokedexSmallXX.txt -output Output.txt -breeding
./pokemon.out -pokedex PokedexSmallXX.txt -output Output.txt -matchup Matchups.txt
```

The output lists each Pokemon class associated with at least one character in the input Pokedex. Specifically, how many of the input characters are members of each class, and what are the *label*s (in alphabetically sorted order) associated with those members? If `-breeding` is specified on the command line, the output will also list which Pokemon can be bred with each of the other Pokemon in the file. Two Pokemon can breed if they share a common Egg Group. *For Extra Credit:* If `-matchup` is specified on the command line, you will list which Pokemon have an attacking advantage or disadvantage over which of the other Pokemon in the input file. (Note: Battle/Training output format unspecified.)

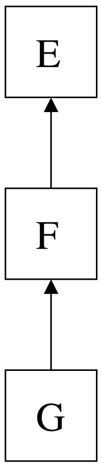
Your Implementation

Your task is to implement the 10 assigned classes + the additional 15 Egg Group classes + the `Pokemon` base class. You should split your implementation into just 2 files: `pokemon.h` and `pokemon.cpp`. Some hints about the implementation:

- In our sample solution, only one class (the base class) has member variable(s) storing character facts.
- In our sample solution, we implement one constructor for each class, taking just one argument, the STL `map` of *facts* about the Pokemon. We do not need to define the default constructor, the copy constructor, or the destructor for any of the classes. Except... to make the compiler happy, the base Pokemon class must have a *virtual* destructor (but the body of the destructor is empty).
- In organizing your code for this assignment, try to avoid unnecessarily duplicating code. For example, don't implement a function or check the value of a fact in *every* class. Instead, allow the derived class to rely on the implementation of that function in its parent class. Also, you don't need to check every fact, only enough to distinguish each character from the other Pokemon on your assigned list.
- The inheritance diagram of these characters includes *multiple inheritance.* Furthermore, the multiple inheritance is in the form of the *Diamond Problem.* That is, Class D multiply inherits from Class B and Class C, and Class B and Class C each inherit from Class A. Thus when an object of type D is created, in turn instances of B and C are created, and unfortunately both will try to make their own instance of A. If two instances of A were allowed, attempts to refer to member variables or member functions of A would be ambiguous. To solve the problem, we should specify that B *virtually* inherits from A and C *virtually* inherits from A. Furthermore, when we construct an instance of D, in addition to specifying how to call constructors for B and C, we also explicitly specify the constructor for A. Note how in the single inheritance example on the right, G only explicitly calls a constructor for F.



```
class A {
public:
  A() {}
};
class B : virtual public A {
public:
  B() : A() {}
};
class C : virtual public A {
public:
  C() : A() {}
};
class D : public B, public C {
public:
  D() : A(), B(), C() {}
};
```



```
class E {
public:
  E() {}
};
class F : public E {
public:
  F() : E() {}
};
class G : public F {
public:
  G() : F() {}
};
```

You must do this assignment on your own, as described in the "Academic Integrity for Homework" handout. If you did discuss the problem or error messages, etc. with anyone, please list their names in your `README.txt` file. When you are finished please zip up your folder exactly as instructed for the previous assignments and submit it through the course webpage.