Assignment #7 - Pokemon Go! .... Kinda ;-) (100 points)

Due Date: Friday December 7, 11:59:00pm

Objective

To gain experience with using an array of structures, as well as with dynamic allocation of an array. This program provides further practice in file input, strings (either c-strings or string objects, programmer's choice), function building, and general code task problem solving and layout.
Filename: `poke.cpp`

Task

The year is 2007. The iphone has just been released. Seeing this opportunity, Nintendo contacts you and asks you to make a mini Pokemon catching simulator game which would eventually become the precursor to today's MEGA HIT Pokemon Go, which clearly has netted you quite a bit of $;)
Anyways, You will write a program that reads and stores Pokemon data from a file into an array of structs. You will then use this data and print it out in various formats, as well as manipulate the data, based on a user controlled menu with multiple selections.

Input File Format

- The first line of the input file will be a single integer number -- this value tells how many records are in the file. There's no guarantee how many records will be in the file, so you must create your dynamic array of structs only AFTER reading in this first integer value. The sample file I provide has a size of 151, but we will test with many different files of many different sizes.
- After this, each line represents one Pokemon entry for your Pokedex. Each line consists of the following fields (in this order), separated by some kind of whitespace:
    - Pokemon Number
    - Pokemon Name
    - Type
    - Catch % (some value between 0 and 100 - the rate at which this type of pokemon is caught).
- You may assume that a file is in the correct format, that there is no trailing whitespace in the file, and that the types and limits will be as described above. You SHOULD test on many different files that have a different number of records in the file. I've provided you a single sample input file. Modify this as you wish (keeping the same formatting. It's best to modify this .txt file right in CLion! DO NOT ever open your input file in "notepad" (windows utility). This will cause your file to have different newline chars used, and cause issues while trying to read in the file. Make sure to test your program to make sure it can handle files with different numbers of Pokemon Records. If your assignment only works with a file of size 151, you'll lose 50 points.
- A sample input file. You can right click that link, "save as" , and save it to your CS account on linprog. Make sure the text file is saved in the same directory as your .cpp file. Avoid opening this text file in other text editors like "Notepad" or "textEdit" as these can cause formatting issues.

Program Details

1. Create a structure type called `Pokemon`. A variable of type Pokemon should contain fields for the following:
    - Pokemon Number (an integer)
    - Name
    - Type
    - Catch % (an integer)
    - Number caught
    - Number Seen
    - You're welcome to add any additional fields to the struct if you wish, but the ones above are required

2. Your program should start by asking the user to type the name of the input file. Whenever a bad filename is entered, print an error message and prompt the user to re-enter the filename. The user should be ***forced*** to enter a correct filename. If the user enters an incorrect file name, ask them to enter again, over and over, until they enter a valid file. You may assume that the user entered filename will not be longer than 40 characters.

3. Setup your array of Pokemon! Read in the size (the first integer in the file), and then Read the records from the file into an array of Pokemon structures. You'll need to create this array dynamically, because you won't know how many records there are until you start reading the file itself. The data from each record in the file should be stored in one Pokemon structure (i.e. one array slot per pokemon record). You'll need to set up some other variables like your "seen" and "caught" ones on your own, as this information is not given in the file.

4. Once you're done reading from the input file, close the file.

5. Welcome the trainer! Ask for their name and print the game instructions (See sample run). You must allow people with names like "Mary Jane" or "Ash Ketchum" to play. You may assume the player's name will be no longer than 40 characters.
6. Initialize some variables that you declare to keep track of the number of each type of pokeball the user has. Start the user with 10 of each kind. The 3 kinds are Poke Balls, Great Balls, and Ultra Balls. PokeBalls give a 0% catch rate bonus , great balls give a 20% catch rate bonus, and the ultra balls give a 40% catch rate bonus.
7. Enter a menu loop that prints the following menu and allows the user to choose from the following options (they should enter the corresponding value to choose):

    ```
    1 - Hunt for a Pokemon!
    2 - Pokedex Statistics.
    3 - Print Inventory.
    4 - List Pokemon By Type
    Q - Quit
    Selection >
    ```

    Should you decide to add additional features, you're welcome to add options to this menu. If you choose to not do option 4 (extra credit), you may omit it from your menu. Note if the user enters an invalid menu option, you should handle this and allow the user to keep entering options.

Note that since the user can eiter digits or letters as a part of their menu entry, you'll need to consider the best way to read these menu choices in.

- **Menu option 1 - Hunt!:**
  When the user selects this option, you'll need to allow them to randomly encounter one of the pokemon. Generate a random number between 1 and the # of records that are present in the file. You can use this number to associate it with a Pokemon that has now appeared!
  The user will choose which ball (denoted by 1, 2, or 3, no invalid # entries should be allowed here...) to throw at the pokemon.
  You only get one shot to catch a pokemon. You either catch it or it runs away.
  Note: the user should not be able to throw a specific type of ball if they do not have any of that ball left in their inventory.
  The type of ball effects the catch rate in the following way: pokeball adds nothing to catch %, Great ball adds 20 to catch %, and ultra ball adds 40% to the catch %. Note that the catch % is DIFFERENT for each Pokemon! :)

  ```
  An example of the catch % mechanics:
  Per the input file, mew's catch % is 3%. if you use an ultra ball to try to catch it, the rate becomes 43% (3+40). If you use a great ball, th
  rate would be 23%. Finally, if you chose a regular pokeball, the rate would stay unchanged at 3%. Your random number generation to simulate th
  should be accurate as far as the odds of catching a pokemon based on its catch % and type of ball used.
  ```

- **Menu option 2 - Pokedex Stats:**
  When the user selects this option, show the pokedex stats like this:

  ```
  no 1     Bulbasaur      Seen: 0 Caught: 0
  no 2     Ivysaur        Seen: 0 Caught: 0
  no 3     Venusaur       Seen: 0 Caught: 0
  no 4     Charmander     Seen: 0 Caught: 0
  no 5     Charmeleon     Seen: 0 Caught: 0
  no 6     Charizard      Seen: 0 Caught: 0
  //...
  //... so on and so forth.... with an overview at the end:
  //...
  no 149   Dragonite      Seen: 0 Caught: 0
  no 150   Mewtwo         Seen: 0 Caught: 0
  no 151   Mew            Seen: 0 Caught: 0
  Total Pokemon Caught: 0 Total Pokemon Seen: 0
  Overall Catch Rate: 0.00%
  ```

  Note that the overall catch rate is always carried out to 2 decimal places, and this value is simply the trainer's overall caught/seen. (setw() and the left/right output stream manipulators may help you align things...).

- **Menu option 3 - Print inventory**
  When the user selects this option , print their inventory like so:

  ```
  You have:
  7 PokeBalls
  10 Great Balls
  9 Ultra Balls
  ```

  Of course, your inventory should reflect the appropriate values of each ball you have left at all times.

- **Menu option 4 - List Pokemon of Type** (Extra credit, 10 points)
  If the user selects this option, list out the pokemon of each type in the following way:

  ```
  Fire: Charmander, Chameleon, Charizard ........... Moltres
  Electric: Pikachu, Raichu, ................. Zapdos
  ```

  Make sure there's no comma after the last item in each type list. If a type list is too long, you're welcome to wrap it to the next line. You may list them in any order of types that you choose. This option must work with types of different capitializations being considered the same... for instance, "water" is the same as "WATER" which is also the same as "Water".

- **Menu option Q: Quit**
  If the user selects Q, you should quit your program. Whether your program exits due to this choice, or the user running out of Pokeballs, the final pokedex stats should be printed one last time before the program fully ends.

- Make sure to clean up any dynamically allocated space before ending the program

- Make sure to close any input or output files before ending the program

- You'll notice that there are no specified functions that your program must have. It is YOUR JOB as the PROGRAMMER to come up with useful functions for your code. **This will be a graded factor of your program, how well you utilize and build functions for yourself (up to 10 points).**

Program Requirements

- No global variables, other than constants
- You may use any of these libraries:
  - `iostream`
  - `iomanip`
  - `fstream`
  - `cctype`
  - `cstdlib`
  - `cstring`
  - `string`
  - `ctime`

- HINT: Be careful if you change the example input file. If you have trailing whitespace at the end of a text file your program attempts to read you could encounter off by one errors. You SHOULD test your program on multiple input files (all of the required format of course)
- Write your source code so that it is readable and well-documented
- Part of assessing readability and style will be how well you break your program into appropriate **functions**.

Sample Program Run

Here is one sample program run, on the sample input file linked previously. Note that this is just **one** example. You can (and should) create your own test cases to test this program more fully. The keyboard/screen interaction is shown (keyboard input is underlined), using the sample input file provided. YOUR CODE SHOULD WORK WITH SAMPLE FILES OF ALL SIZES. NOT JUST SIZE 151. SAMPLE RUN

Submit your program (Use the filename **`poke.cpp`**) in the usual way:

```
~vastola/usub/submit1 poke.cpp
```