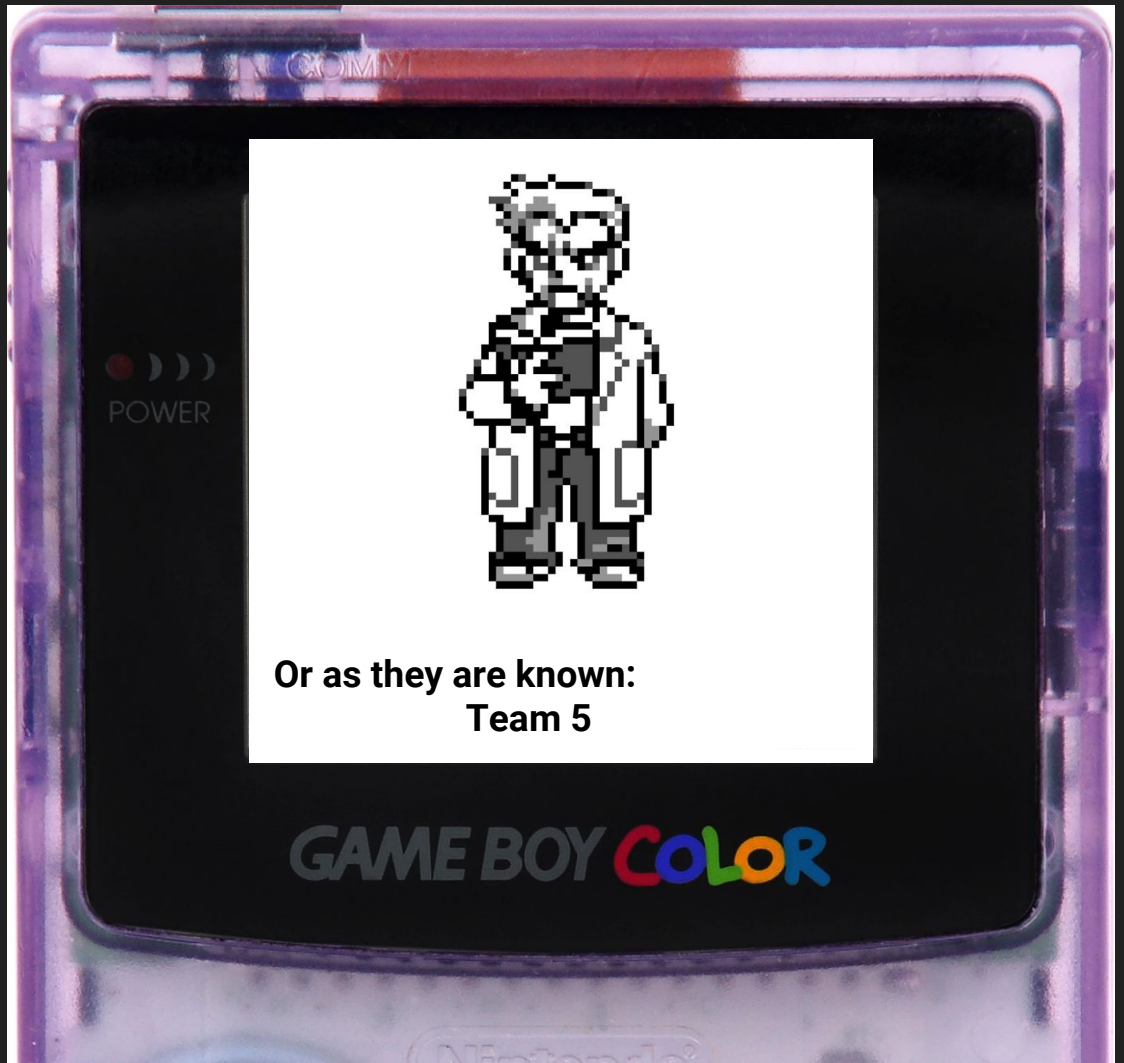
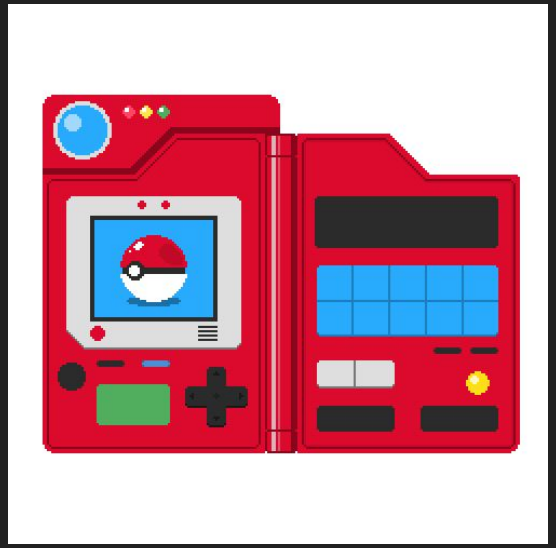

POWER
Or as they are known:
Team 5
GAME BOY COLOR

Project Goals

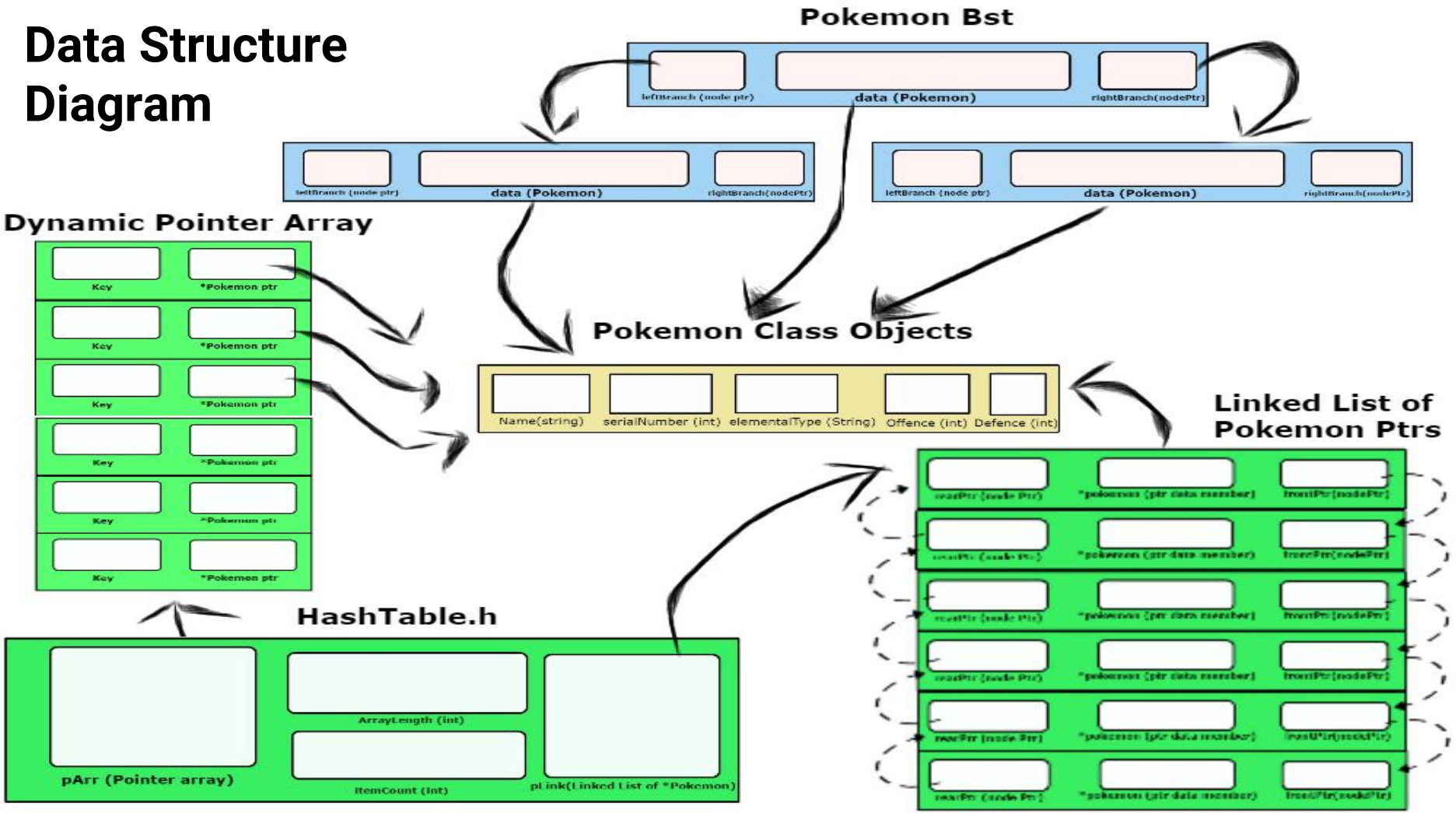
- Since its Inception in 1995, Pokemon or “Pocket Monsters” has greatly expanded in popularity. With 3.7 Million Copies of the latest games being sold at Launch Alone
- With a roster of over 802 monsters and more on the way keeping track of all these Pokemon, their type matchups and stats is a daunting task.
- Luckily in the world of Pokemon we have a device known as a Pokedex which allows us to keep track of all the Pokemon.
- This project will attempt to emulate the functionality of the Pokedex as well as include some other feature that players may find useful



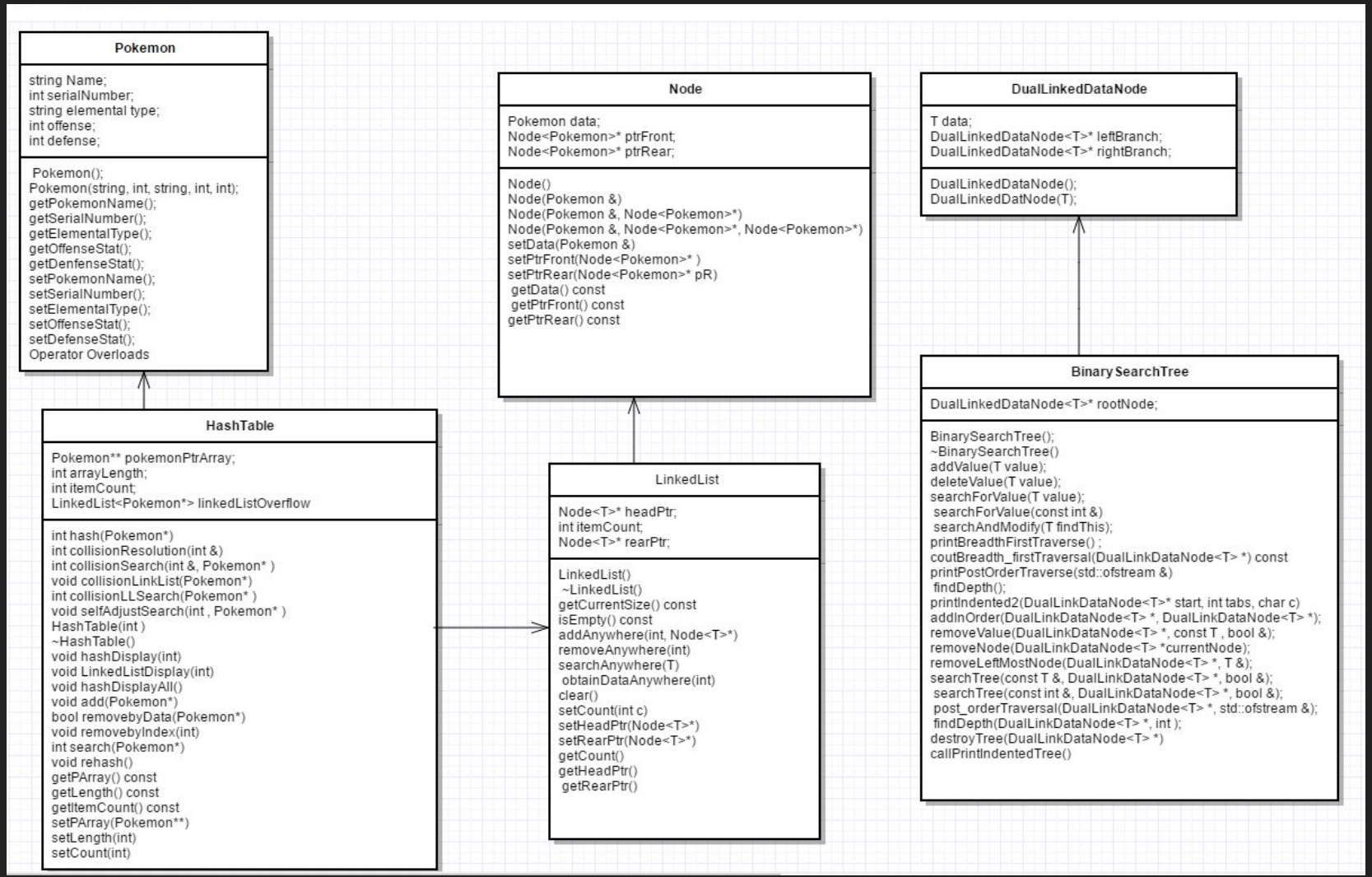
Project Application

- In addition to emulating the in Game dex we hope to make this program a tool for the more devout players of the Pokemon series which may help them build teams or even choose out their favorite Pokemon.
- This concept can be taken further by providing analysis of competitive teams and allow players to partake in the grandiose competitive scene of the Pokemon VGC tournaments held globally

Data Structure Diagram



UML


Pokemon
string Name;
int serialNumber;
string elemental type;
int offense;
int defense;
Pokemon();
Pokemon(string, int, string, int, int);
getPokemonName();
getSerialNumber();
getElementalType();
getOffenseStat();
getDenfenseStat();
setPokemonName();
setSerialNumber();
setElementalType();
setOffenseStat();
setDefenseStat();
Operator Overloads
Node
Pokemon data;
Node<Pokemon>* ptrFront;
Node<Pokemon>* ptrRear;
Node()
Node(Pokemon &)
Node(Pokemon &, Node<Pokemon>*)
Node(Pokemon &, Node<Pokemon>*, Node<Pokemon>*)
setData(Pokemon &)
setPtrFront(Node<Pokemon>*)
setPtrRear(Node<Pokemon>* pR)
getData() const
getPtrFront() const
getPtrRear() const
DualLinkedDataNode
T data;
DualLinkedDataNode<T>* leftBranch;
DualLinkedDataNode<T>* rightBranch;
DualLinkedDataNode();
DualLinkedDatNode(T);
HashTable
Pokemon** pokemonPtrArray;
int arrayLength;
int itemCount;
LinkedList<Pokemon*> linkedListOverflow
int hash(Pokemon*)
int collisionResolution(int &)
int collisionSearch(int &, Pokemon*)
void collisionLinkList(Pokemon*)
int collisionLLSearch(Pokemon*)
void selfAdjustSearch(int , Pokemon*)
HashTable(int)
~HashTable()
void hashDisplay(int)
void LinkedListDisplay(int)
void hashDisplayAll()
void add(Pokemon*)
bool removebyData(Pokemon*)
void removebyIndex(int)
int search(Pokemon*)
void rehash()
getPArray() const
getLength() const
getItemCount() const
setPArray(Pokemon**)
setLength(int)
setCount(int)
LinkedList
Node<T>* headPtr;
int itemCount;
Node<T>* rearPtr;
LinkedList()
~LinkedList()
getCurrentSize() const
isEmpty() const
addAnywhere(int, Node<T>*)
removeAnywhere(int)
searchAnywhere(T)
obtainDataAnywhere(int)
clear()
setCount(int c)
setHeadPtr(Node<T>*)
setRearPtr(Node<T>*)
getCount()
getHeadPtr()
getRearPtr()
Binary SearchTree
DualLinkedDataNode<T>* rootNode;
BinarySearchTree();
~BinarySearchTree()
addValue(T value);
deleteValue(T value);
searchForValue(T value);
searchForValue(const int &)
searchAndModify(T findThis);
printBreadthFirstTraverse() ;
coutBreadth_firstTraversal(DualLinkDataNode<T> *) const
printPostOrderTraverse(std::ofstream &)
findDepth();
printIndented2(DualLinkDataNode<T>* start, int tabs, char c)
addInOrder(DualLinkDataNode<T> *, DualLinkDataNode<T> *);
removeValue(DualLinkDataNode<T> *, const T , bool &);
removeNode(DualLinkDataNode<T> *currentNode);
removeLeftMostNode(DualLinkDataNode<T> *, T &);
searchTree(const T &, DualLinkDataNode<T> *, bool &);
searchTree(const int &, DualLinkDataNode<T> *, bool &);
post_orderTraversal(DualLinkDataNode<T> *, std::ofstream &);
findDepth(DualLinkDataNode<T> *, int);
destroyTree(DualLinkDataNode<T> *)
callPrintIndentedTree()

Data Class

Pokemon Name: Pikachu
Serial Number: 25
Elemental Type: Electric
Offense Stat: 30
Defense Stat: 30

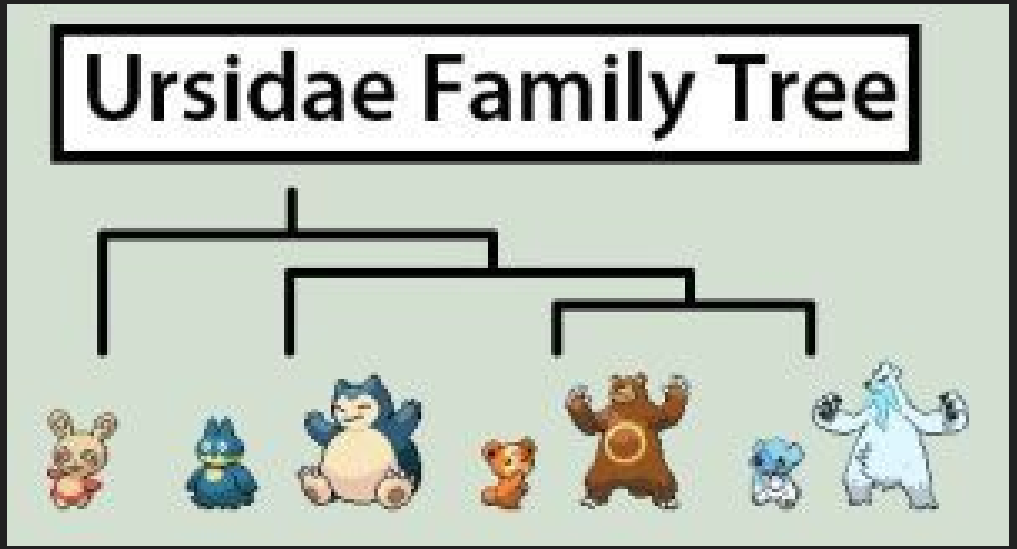
Contributors: Hammud, Clifford

Features

A. Adding Pokemon
B. Deleting Pokemon
C. Simple Retrieval of Data
D. Can rewrite the contents with a single Function call
E. Easy Comparisons of data done via Overloaded Operators
F. Overloaded >> operator allows for easy creation of Pokemon Class Objects!
G. Overloaded << allows for a clean display of the Pokemon!
H. Type Conversion to int allows easy access to serial number!

BST Features

A. Addition and Deletion of Nodes of Pokemon
B. Ability to search for Pokemon though
C. Printing out an indented list of cataloged Pokemon



The BST is Assembled via comparison of Node data which in turn uses the overloaded operators in the Pokemon class to allow for a simple assembly. The searches iterate through the tree and return values the user or developer may seek.

Contributors: Alex, Hammud & Clifford

HashTable Features

Core Features:

A. Addition and Deletion of Nodes of Pokemon
B. Ability to search for Pokemon though
C. Display the entire hash table

Unique Specification:

A. Linear Collision Resolution
B. Self-Adjusting Linked List
C. Hash Arr Resizes when 75% of indices are filled

Contributors: Leo, & Clifford

Linear Collision Vs. LinkedList Collision

```
//LINKED LIST COLLISION RESOLUTION INNER ELSE STATEMENT
else
{
	index = collisionLLSearch(data);
	isRemoved = linkedListOverflow.removeAnywhere(index);
	return isRemoved;
}
```

```
//LINEAR COLLISION RESOLUTION INNER ELSE STATEMENT
/*
else
{
	collisionSearch(index, data);
	if (index >= 0)
	{
		delete pArray[index];
		pArray[index] = nullptr;
		itemCount--;
		isRemoved = true;
		return isRemoved;
	}
}
*/
```

Wrapper class vs. Overloaded Operators for *ptrs

```
// overloaded > operator for the pokemon pointer
bool operator > (const Pokemon *&rightSide)
{
  bool status;
  If (this->serialNumber == rightSide->serialNumber)
  {
   --  More code here --
  }
  return status;
}
// overloaded > operator for the pokemon pointer
bool operator > (const Pokemon &rightSide){
  bool status;
  if (this->serialNumber == rightSide.serialNumber)
  {
   --  More code here --
  }
  return status;
}
```

```
/*
struct SortablePokemon {

Pokemon* p;

bool operator<=(SortablePokemon other) {
  return p->getSerialNumber() <= other.p->getSerialNumber();
}

friend ostream &operator <<(ostream &outStream,SortablePokemon creature)
{
  cout << creature.p->getSerialNumber()
       << " " << creature.p->getPokemonName()
       << " " << creature.p->getElementalType();
  return outStream;

}
};
*/
```