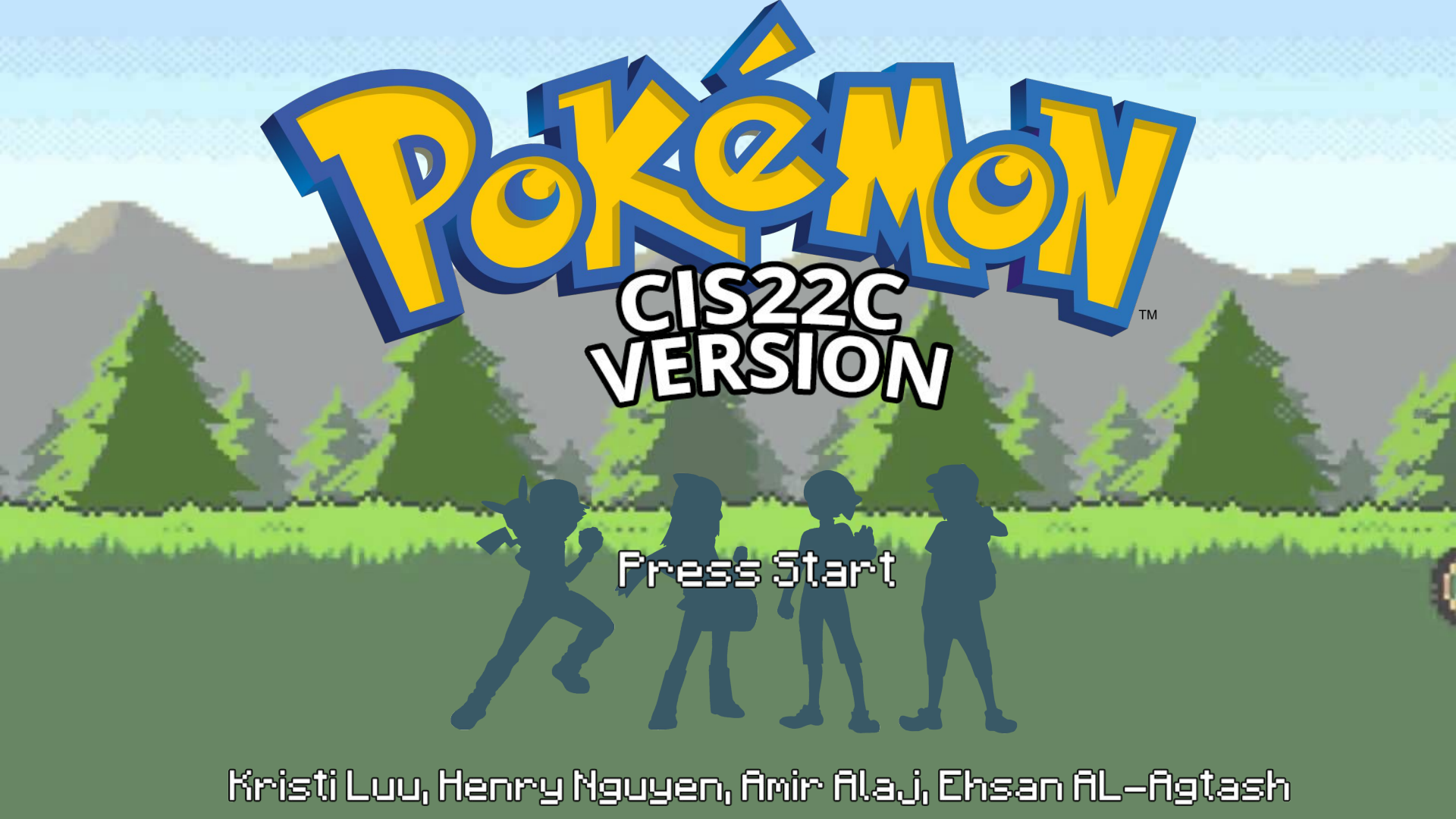
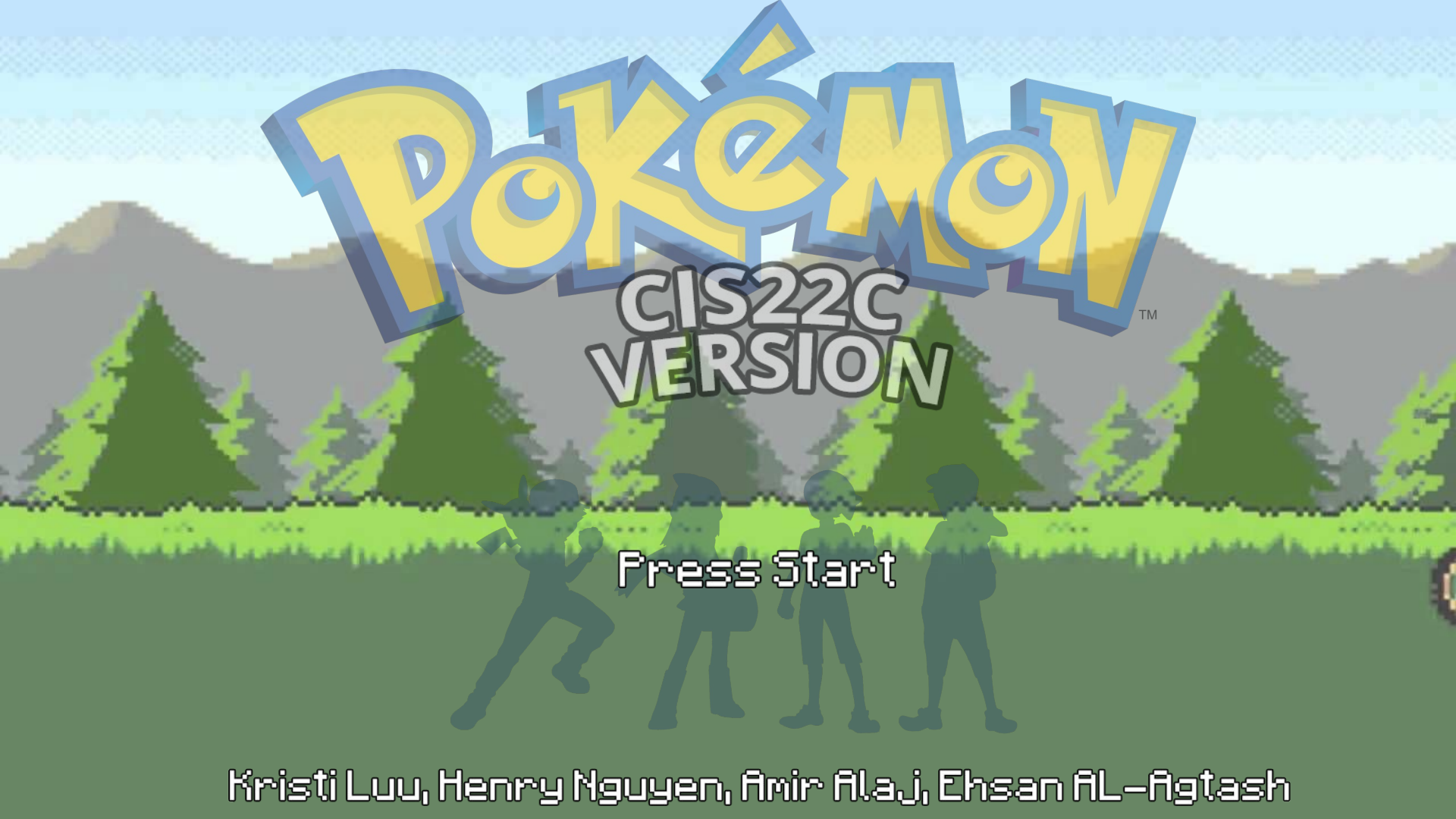
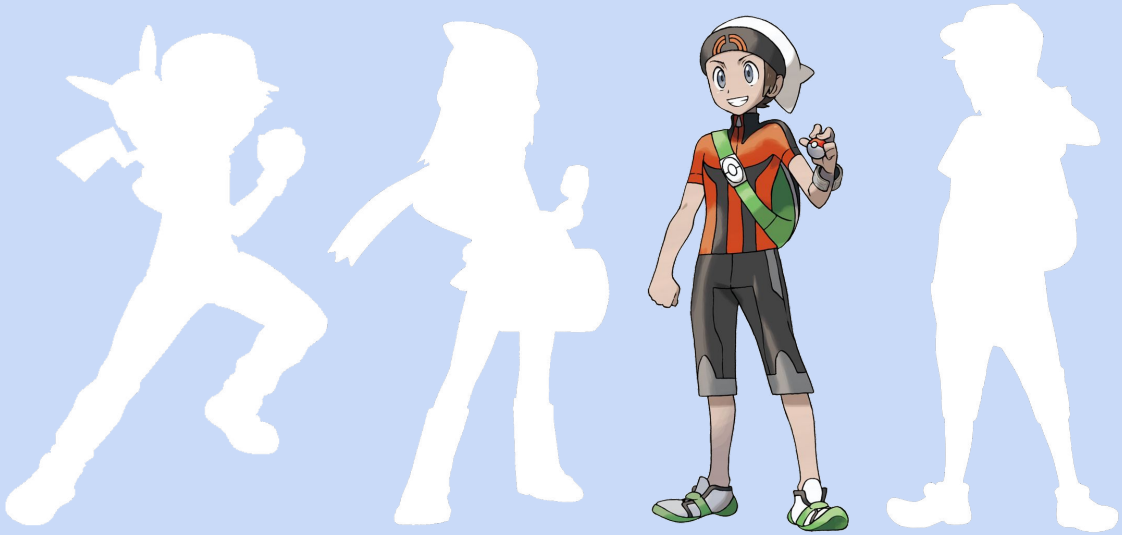
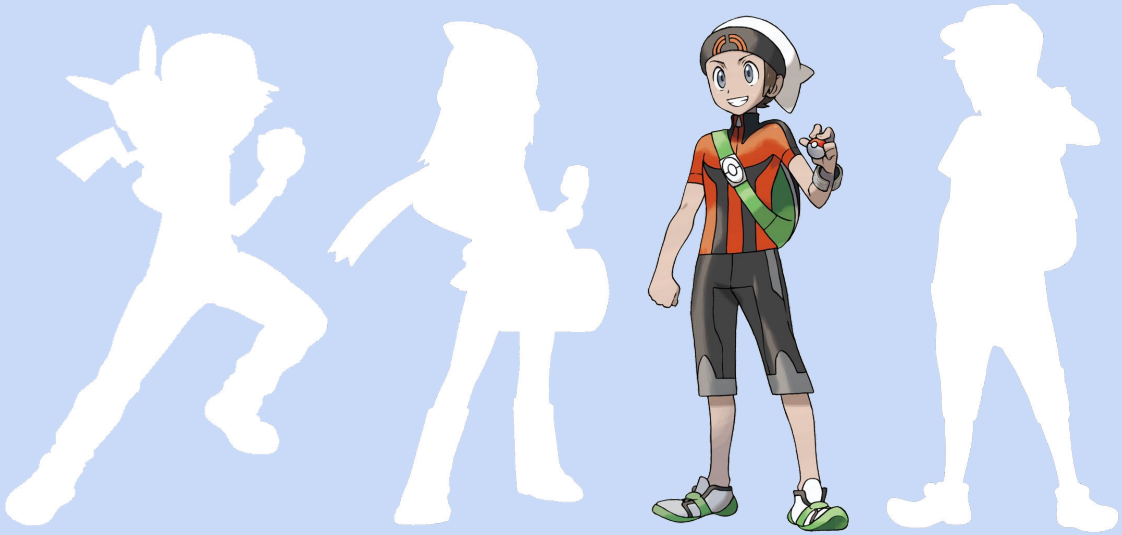

Pokémon
CIS22C
VERSION
Press Start
Kristi Luu, Henry Nguyen, Amir Alaj, Ehsan AL-Agtash


Pokémon
CIS22C
VERSION
Press Start
Kristi Luu, Henry Nguyen, Amir Alaj, Ehsan AL-Agtash

Hi. I'm Professor Oak! Welcome to Pokemon.
Let me introduce you to my trainers. ▼



Team Leader: Ehsan Al-Aqtash
Responsibilities: . . . ! ▼



Lead team and worked on Hash Functions Algorithms. ▼

Kristi Luu
Responsibilities!...! ▼

Worked on main.cpp, test plan, and screen output. ▼

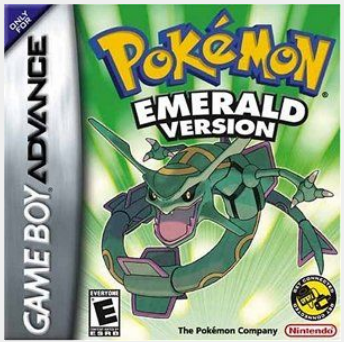
Henry Nguyen
Responsibilities!...! ▼

Worked on BST Algorithms, and Rehash Function, and Hash Function.

Amir Alaj
Responsibilities!...!

Worked on BST Algorithms, File I/O, and Undo Delete Function. ▼

WHAT IS POKÉMON?



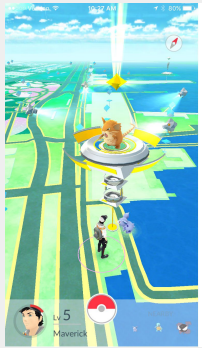
It is a series of video games designed by Nintendo.

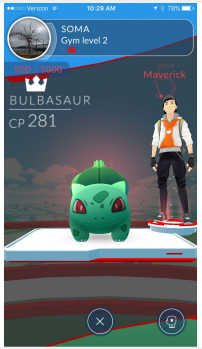
First released as a GameBoy game, and became worldwide.

Available in Wii, Nintendo DS, Nintendo Switch, and more!

WHAT IS POKÉMON?

Download on the App Store





Also comes in game-card form.
Was very popular several years ago.

Now, the phone game app called "Pokemon Go" is really popular.

WHAT IS POKÉMON?

There are also human characters, but the main population is Pokémon.

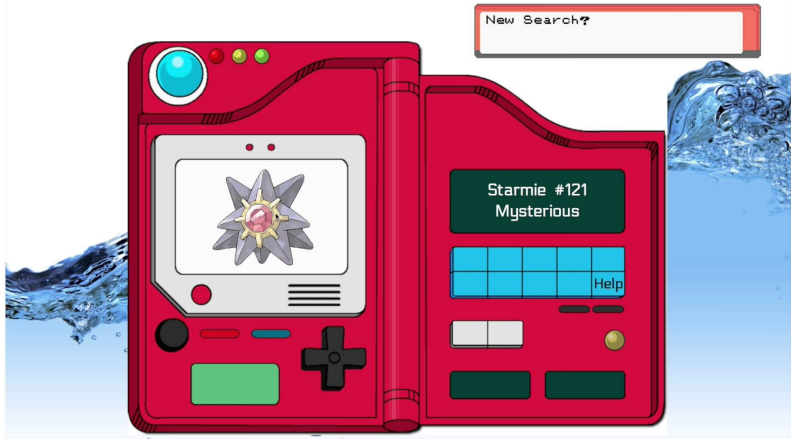
The Pokémon in this game are very detailed:

- Height
- Weight
- Type (fire type, water type, etc...)
- Species (mouse, dragon, etc.)
- And more!

AND THERE ARE HUNDREDS

OUR PROJECT

In the game, there is something called a Pokédex, which is like a dictionary of all the Pokémon you've seen or caught.



OUR PROJECT

C pokemon.h ×

```
1   #ifndef POKEMON_H_INCLUDED
2   #define POKEMON_H_INCLUDED
3
4   #include <iostream>
5   #include <string>
6
7   using namespace std;
8
9   class Pokemon
10  {
11  private:
12      string ID;
13      string name;
14      double weight;
15      string type;
16      string gender;
17      double height;
18
```

Our project is basically a program that does just that:

A dictionary of 26 pokémon in our text file.

Therefore, our text file includes:

ID -- Primary Key

Each Pokémon in the game has a unique number

We changed the numbers so that they are easier to use (5 digit number)

OUR PROJECT

pokemon.h

```
#ifndef POKEMON_H_INCLUDED
#define POKEMON_H_INCLUDED

#include <iostream>
#include <string>

using namespace std;

class Pokemon
{
private:
    string ID;
    string name;
    double weight;
    string type;
    string gender;
    double height;

```

Name of a Pokémon -- Secondary Key

This is a secondary key because it is possible to "catch" the same pokémon with different stats.

Weight

Type

Each Pokemon has a type, or an element, that they represent.

Example: Pikachu is electric

Many pokémon have 2 types, but for simplicity, we are only using one.

OUR PROJECT

pokemon.txt

```
11750 Marshtomp; 61.7 Water Male .7
10650 Jigglypuff; 12.1 Fairy Female .5
10050 Abra; 43 Psychic Male .9
11100 Slugma; 77.2 Fire Male .7
11850 Kadabra; 124.6 Psychic Male 1.3
10150 Nosepass; 213.8 Rock Female 1
11500 Groudon; 2094 Ground Genderless 3.5
12300 Igglybuff; 2.2 Normal Female .3
10100 Hariyama; 559.5 Fighting Male 2.3
12550 Raichu; 66.1 Electric Male .8
10900 Phanpy; 73.9 Ground Female .5
11000 Shelgon; 243.6 Dragon Female 1.1
12800 Combusken; 43 Fire Male .9
10850 Pikachu; 13.2 Electric Male .4
11150 Blaziken; 114.6 Fire Male 1.9
12850 Metang; 446.4 Steel Genderless 1.2
11250 Regirock; 507.1 Rock Genderless 1.7
11200 Metagross; 1212.5 Steel Genderless 1.6
11450 Kyogre; 776 Water Genderless 4.5
11300 Registeel; 451.9 Steel Genderless 1.9
11350 Latias; 88.2 Dragon Female 1.4
11550 Rayquaza; 455.2 Dragon Genderless 7
11400 Latios; 132.3 Dragon Male 2
10000 Mudkip; 16.8 Water Male .4
12650 Bagon; 92.8 Dragon Female .6
11650 Deoxys; 134 Psychic Genderless 1.7
11651 Deoxys; 134 Psychic Genderless 1.7
```

Gender

Gender varies per pokemon because sometimes there can be both male and female of that pokemon. Some are even genderless. We chose the most common.

Height

Converted to meters for simplicity

OUR PROJECT

```
Pokemon* list[50];
BinarySearchTree<Pokemon*> tree1;
BinarySearchTree<Pokemon*> tree2;
Stack<Pokemon> undo;
HashTable<string, Pokemon> hashtable(filesize);
```

```
// prints BST in breadth order
template<class ItemType>
void BinaryTree<ItemType>::_breadth(void visit(ItemType),
                                    BinaryNode<ItemType>* nodePtr)  const
{
    BinaryNode<ItemType>* newNodePtr;
    Queue<BinaryNode<ItemType>*> line;
    line.enqueue(nodePtr);

    while(!line.isEmpty())
    {
        line.dequeue(newNodePtr);
        ItemType item = newNodePtr->getItem();
        visit(item);
        if(newNodePtr->getLeftPtr() != 0)
            line.enqueue(newNodePtr->getLeftPtr());
        if(newNodePtr->getRightPtr() != 0)
            line.enqueue(newNodePtr->getRightPtr());
    }
}
```

In this program, we use two binary search trees and one hash table.

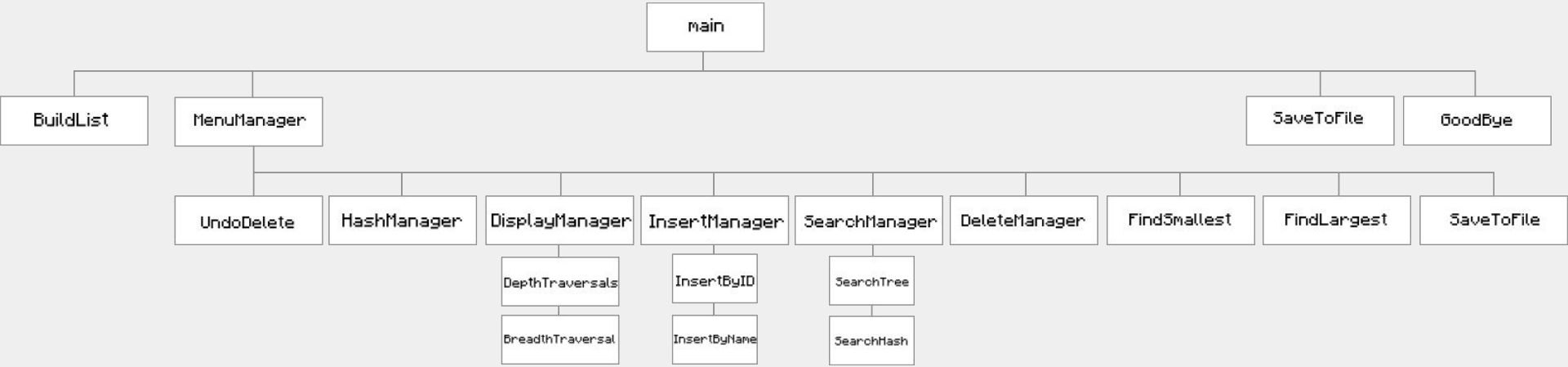
It also includes the use of stacks and queues. Queue for breadth-first display and stack for undo-delete function.

The Hash Function is Modulus Division.
The Hash Resolution is Linked List.

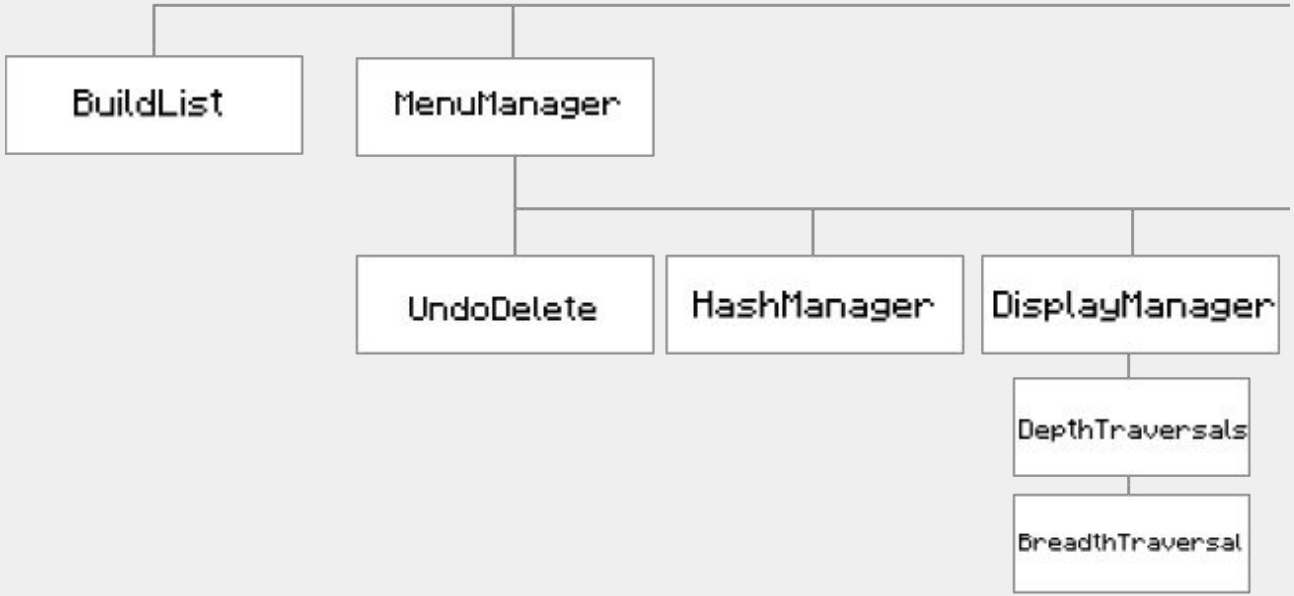
OUR PROJECT

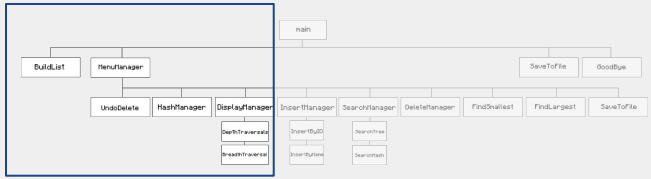
Here is a visual of all the pokémon we used.

DATA STRUCTURE CHART

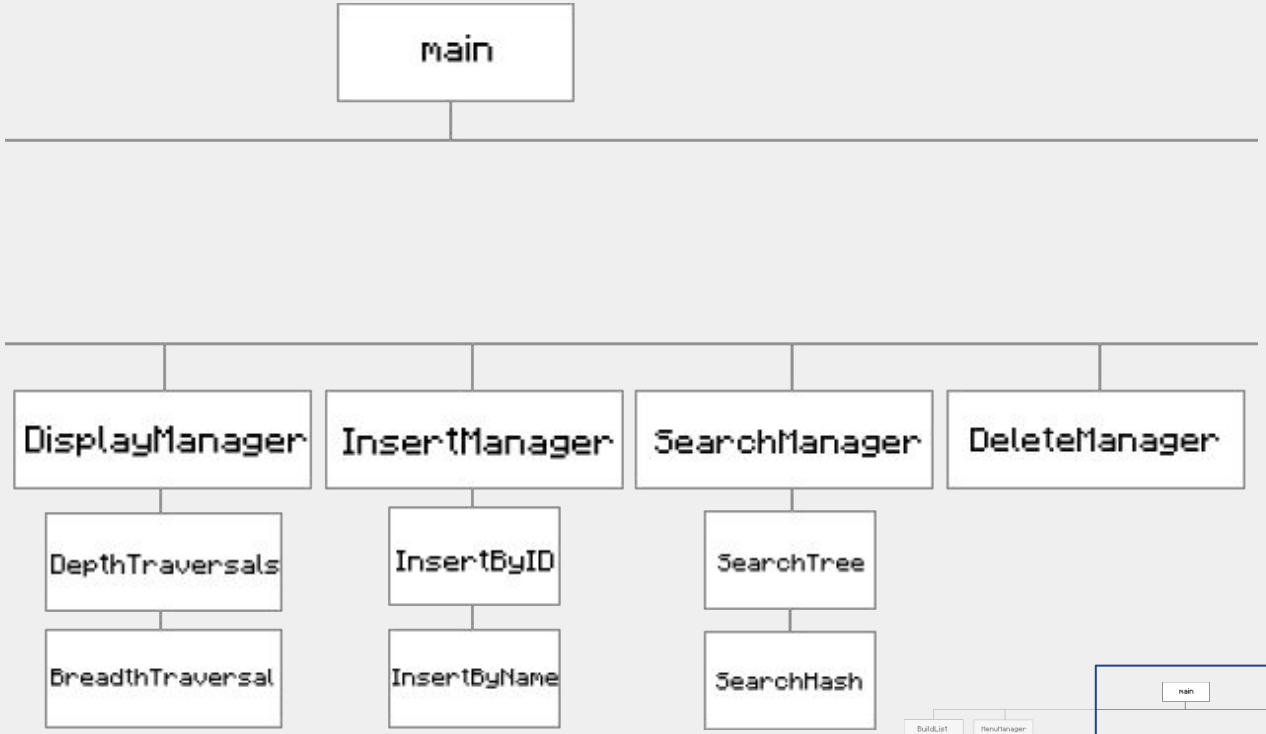


DATA STRUCTURE CHART

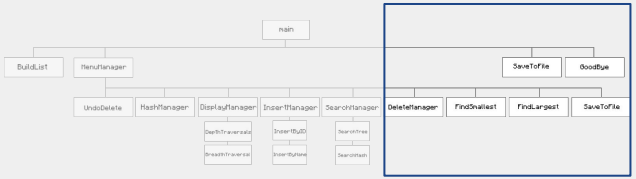




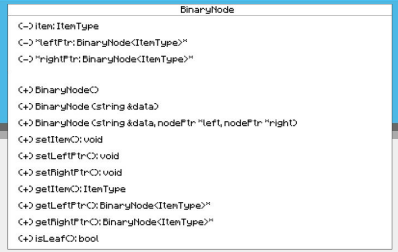
DATA STRUCTURE CHART

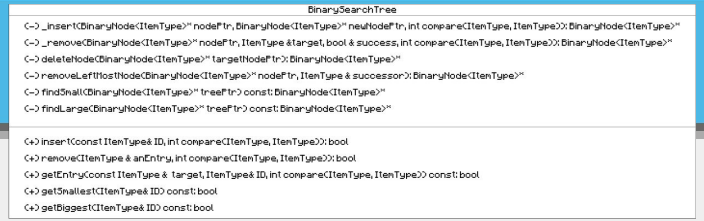


DATA STRUCTURE CHART

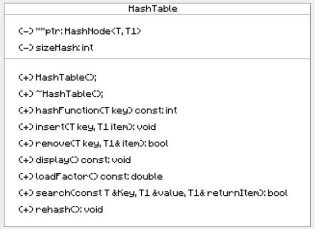


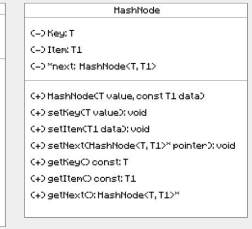
UML CHART

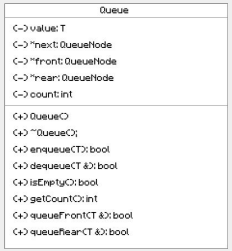

BinaryNode
(-) item: ItemType
(-) *leftPtr: BinaryNode<ItemType>*
(-) *rightPtr: BinaryNode<ItemType>*
(+) BinaryNode()
(+) BinaryNode (string &data)
(+) BinaryNode (string &data, nodePtr *left, nodePtr *right)
(+) setItem(): void
(+) setLeftPtr(): void
(+) setRightPtr(): void
(+) getItem(): ItemType
(+) getLeftPtr(): BinaryNode<ItemType>*
(+) getRightPtr(): BinaryNode<ItemType>*
(+) isLeaf(): bool


BinarySearchTree
(-) _insert(BinaryNode<ItemType>* nodePtr, BinaryNode<ItemType>* newNodePtr, int compare(ItemType, ItemType)): BinaryNode<ItemType>*
(-) _remove(BinaryNode<ItemType>* nodePtr, ItemType &target, bool & success, int compare(ItemType, ItemType)): BinaryNode<ItemType>*
(-) deleteNode(BinaryNode<ItemType>* targetNodePtr): BinaryNode<ItemType>*
(-) removeLeftMostNode(BinaryNode<ItemType>* nodePtr, ItemType & successor): BinaryNode<ItemType>*
(-) findSmall(BinaryNode<ItemType>* treePtr) const: BinaryNode<ItemType>*
(-) findLarge(BinaryNode<ItemType>* treePtr) const: BinaryNode<ItemType>*
(+) insert(const ItemType& ID, int compare(ItemType, ItemType)): bool
(+) remove(ItemType & anEntry, int compare(ItemType, ItemType)): bool
(+) getEntry(const ItemType & target, ItemType& ID, int compare(ItemType, ItemType)) const: bool
(+) getSmallest(ItemType& ID) const: bool
(+) getBiggest(ItemType& ID) const: bool

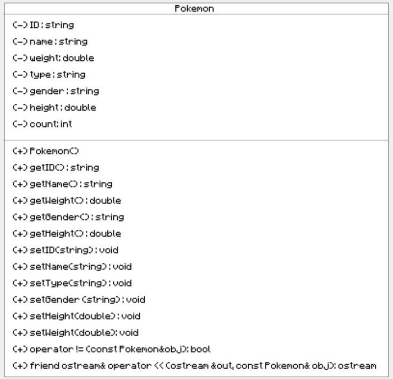

BinaryTree
(#) *ptr: BinaryNode<ItemType>*
(#) count: int
(-) destroyTree(BinaryNode<ItemType>* nodePtr): void
(-) copyTree (const BinaryNode<ItemType>* nodePtr): BinaryNode<ItemType>*
(-) _preorder(void visit(ItemType), BinaryNode<ItemType>* nodePtr) const: void
(-) _inorder(void visit(ItemType), BinaryNode<ItemType>* nodePtr) const: void
(-) _postorder(void visit(ItemType), BinaryNode<ItemType>* nodePtr) const: void
(-) _breadth(void visit(ItemType), BinaryNode<ItemType>* nodePtr) const: void
(-) printTree(BinaryNode<ItemType>* nodePtr, int level) const: void
(+) BinaryTree()
(+) BinaryTree(const BT &)
(+) virtual ~BinaryTree()
(+) isEmpty(): bool
(+) size(): int
(+) clear(): void
(+) print(): void
(+) preOrder(void visit(ItemType)): void
(+) inOrder(void visit(ItemType)): void
(+) postOrder(void visit(ItemType)): void
(+) breadthOrder(void visit(ItemType)): void
(+) insert(const ItemType &, int compare(ItemType, ItemType)): virtual bool
(+) remove(ItemType & data, int compare(ItemType, ItemType)): virtual bool
(+) getEntry(const ItemType &, ItemType& ID, int compare(ItemType, ItemType)): virtual bool
(+) BinaryTree & operator = (const BinaryTree & sourceTree);


HashTable
(-) **ptr: HashNode<T, T1>
(-) sizeHash: int
(+) HashTable();
(+) ~HashTable();
(+) hashFunction(T key) const: int
(+) insert(T key, T1 item): void
(+) remove(T key, T1& item): bool
(+) display() const: void
(+) loadFactor() const: double
(+) search(const T &Key, T1 &value, T1& returnItem): bool
(+) rehash(): void


HashNode
(-) Key: T
(-) Item: T1
(-) *next: HashNode<T, T1>
(+) HashNode(T value, const T1 data)
(+) setKey(T value): void
(+) setItem(T1 data): void
(+) setNext(HashNode<T, T1>* pointer): void
(+) getKey() const: T
(+) getItem() const: T1
(+) getNext(): HashNode<T, T1>*


Queue
(-) value: T
(-) *next: QueueNode
(-) *front: QueueNode
(-) *rear: QueueNode
(-) count: int
(+) Queue()
(+) ~Queue();
(+) enqueue(T): bool
(+) dequeue(T &): bool
(+) isEmpty(): bool
(+) getCount(): int
(+) queueFront(T &): bool
(+) queueRear(T &): bool


Stack
(-) value: T
(-) *next: StackNode
(-) *top: StackNode
(-) count: int
(+) Stack()
(+) ~Stack()
(+) push(T): bool
(+) pop(T &): bool
(+) isEmpty(): bool
(+) getCount(): int
(+) getTop(T &item): bool


Pokemon
(-) ID : string
(-) name : string
(-) weight: double
(-) type : string
(-) gender : string
(-) height : double
(-) count: int
(+) Pokemon()
(+) getID() : string
(+) getName() : string
(+) getWeight() : double
(+) getGender() : string
(+) getHeight() : double
(+) setID(string) : void
(+) setName(string) : void
(+) setType(string) : void
(+) setGender (string) : void
(+) setHeight(double) : void
(+) setWeight(double): void
(+) operator != (const Pokemon&ob_j): bool
(+) friend ostream& operator << (ostream &out, const Pokemon& ob_j): ostream

UML CHART

BinaryTree

(#) *ptr: BinaryNode<ItemType>*

(#) count: int

(–) destroyTree(BinaryNode<ItemType>* nodePtr): void

(–) copyTree (const BinaryNode<ItemType>* nodePtr): BinaryNode<ItemType>*

(–) _preorder(void visit(ItemType), BinaryNode<ItemType>* nodePtr) const: void

(–) _inorder(void visit(ItemType), BinaryNode<ItemType>* nodePtr) const: void

(–) _postorder(void visit(ItemType), BinaryNode<ItemType>* nodePtr) const: void

(–) _breadth(void visit(ItemType), BinaryNode<ItemType>* nodePtr) const: void

(–) printTree(BinaryNode<ItemType>* nodePtr, int level) const: void

(+) BinaryTree()

(+) BinaryTree(const BT &)

(+) virtual ~BinaryTree()

(+) isEmpty(): bool

(+) size(): int

(+) clear(): void

(+) print(): void

(+) preOrder(void visit(ItemType)): void

(+) inOrder(void visit(ItemType)): void

(+) postOrder(void visit(ItemType)): void

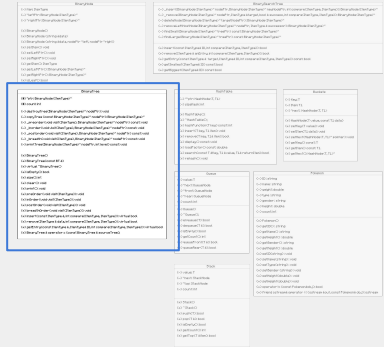
(+) breadthOrder(void visit(ItemType)): void

(+) insert(const ItemType &, int compare(ItemType, ItemType)): virtual bool

(+) remove(ItemType & data, int compare(ItemType, ItemType)): virtual bool

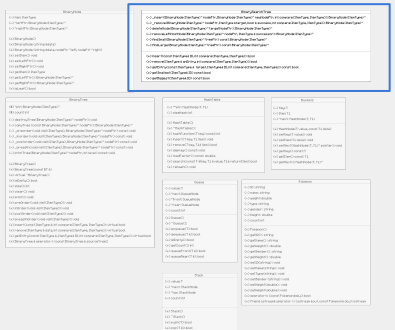
(+) getEntry(const ItemType &, ItemType& ID, int compare(ItemType, ItemType)): virtual bool

(+) BinaryTree & operator = (const BinaryTree & sourceTree);



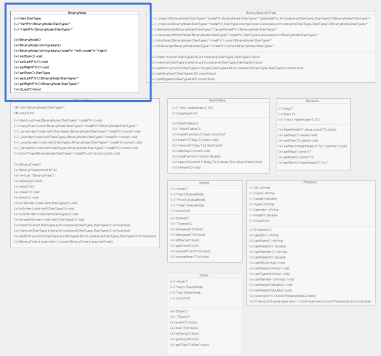
UML CHART

BinarySearchTree
(-) _insert(BinaryNode<ItemType>* nodePtr, BinaryNode<ItemType>* newNodePtr, int compare(ItemType, ItemType)): BinaryNode<ItemType>*
(-) _remove(BinaryNode<ItemType>* nodePtr, ItemType &target, bool & success, int compare(ItemType, ItemType)): BinaryNode<ItemType>*
(-) deleteNode(BinaryNode<ItemType>* targetNodePtr): BinaryNode<ItemType>*
(-) removeLeftMostNode(BinaryNode<ItemType>* nodePtr, ItemType & successor): BinaryNode<ItemType>*
(-) findSmall(BinaryNode<ItemType>* treePtr) const: BinaryNode<ItemType>*
(-) findLarge(BinaryNode<ItemType>* treePtr) const: BinaryNode<ItemType>*
(+) insert(const ItemType& ID, int compare(ItemType, ItemType)): bool
(+) remove(ItemType & anEntry, int compare(ItemType, ItemType)): bool
(+) getEntry(const ItemType & target, ItemType& ID, int compare(ItemType, ItemType)) const: bool
(+) getSmallest(ItemType& ID) const: bool
(+) getBiggest(ItemType& ID) const: bool



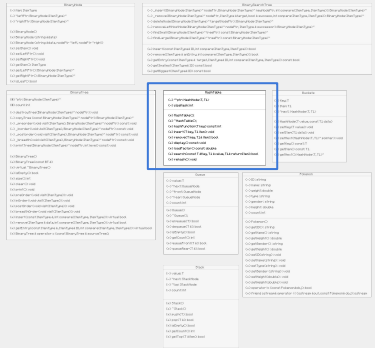
UML CHART

BinaryNode
(-) item: ItemType (-) *leftPtr: BinaryNode<ItemType>* (-) *rightPtr: BinaryNode<ItemType>*
(+) BinaryNode() (+) BinaryNode (string &data) (+) BinaryNode (string &data, nodePtr *left, nodePtr *right) (+) setItem(): void (+) setLeftPtr(): void (+) setRightPtr(): void (+) getItem(): ItemType (+) getLeftPtr(): BinaryNode<ItemType>* (+) getRightPtr(): BinaryNode<ItemType>* (+) isLeaf(): bool



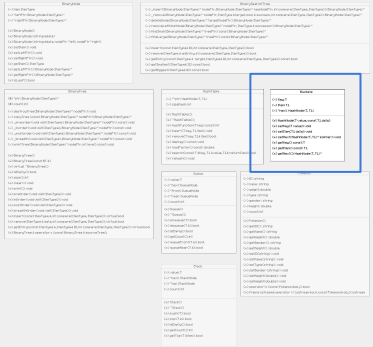
UML CHART

HashTable
(-) **ptr: HashNode<T, T1> (-) sizeHash: int
(+) HashTable(); (+) ~HashTable(); (+) hashFunction(T key) const: int (+) insert(T key, T1 item): void (+) remove(T key, T1& item): bool (+) display() const: void (+) loadFactor() const: double (+) search(const T &Key, T1 &value, T1& returnItem): bool (+) rehash(): void



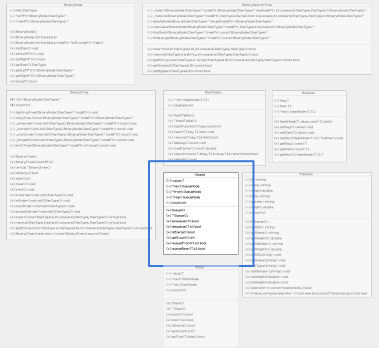
UML CHART

HashNode
(-) Key: T (-) Item: T1 (-) *next: HashNode<T, T1>
(+) HashNode(T value, const T1 data) (+) setKey(T value): void (+) setItem(T1 data): void (+) setNext(HashNode<T, T1>* pointer): void (+) getKey() const: T (+) getItem() const: T1 (+) getNext(): HashNode<T, T1>*



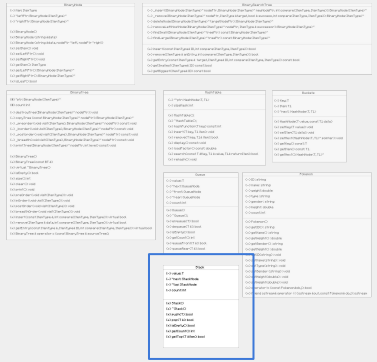
UML CHART

Queue
(–) value: T (–) *next: QueueNode (–) *front: QueueNode (–) *rear: QueueNode (–) count: int
(+) Queue() (+) ~Queue(); (+) enqueue(T): bool (+) dequeue(T &): bool (+) isEmpty(): bool (+) getCount(): int (+) queueFront(T &): bool (+) queueRear(T &): bool



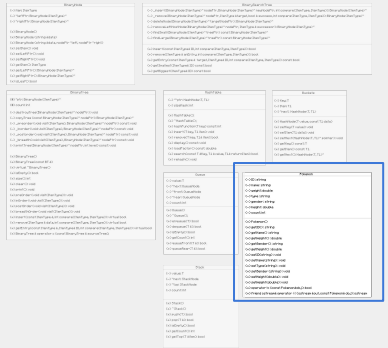
UML CHART

Stack
(–) value: T (–) *next: StackNode (–) *top: StackNode (–) count: int
(+) Stack() (+) ~Stack() (+) push(T): bool (+) pop(T &): bool (+) isEmpty(): bool (+) getCount(): int (+) getTop(T &item): bool



UML CHART

Pokemon
(–) ID : string (–) name : string (–) weight: double (–) type : string (–) gender : string (–) height : double
(+) Pokemon() (+) getID() : string (+) getName() : string (+) getWeight() : double (+) getGender() : string (+) getHeight() : double (+) setID(string) : void (+) setName(string) : void (+) setType(string) : void (+) setGender (string) : void (+) setHeight(double) : void (+) setWeight(double): void (+) operator != (const Pokemon&obj): bool (+) friend ostream& operator << (ostream &out, const Pokemon& obj): ostream



STRUCTURE CHART