Project 1
~Pokemon RPG~

CSC-17C-44049
Name: Najera, Enrique
Date Due: 17 April 2017

Introduction

Title: Pokemon RPG

In the world of Pokemon, one must level up their Pokemon in order to be the strongest trainer!
Leveling up Pokemon, however, requires battling other Pokemon and foes!
You can also catch Pokemon to expand your library of available fighters (from weak to strong).
Fighting monsters will make your Pokemon unhealthy and hurt, but using items can restore their health! Buy items at the Hospital or find them lying around!

First time players: press ‘h’ to bring up a legend and help menu!

Quitting the game or dying in-game will make you lose your progress, so be careful!

I chose this style of project because it contains a lot of logic and features. This helped me decide what concepts to use, and where, easier.

Summary

Program size: ~1160 lines (Excluded the header [file, author, date, purpose] and huge comments, then rounded up to the nearest 10^{th} place)
Number of major variables: ~44
Number of constructs: ~17

It was fairly challenging to do. I had the most problems with buffer overflows caused by linked lists, displaying linked lists the way I wanted them to, making up how the project requirements would fit into a game, and thinking up what to do for the project in general. I also had no idea what to use a queue for (feels like it would be a repeat of something already in-game).

This project took me about 16 hours to do, with about 1 hour of debugging.
I had many challenges and features I would have loved to add, but sadly, time does not allow it at the moment.
I had fun making this project (although I admit some times were frustrating) and at the same time learned so much on how to use a few of the STL algorithms, containers, and iterators.

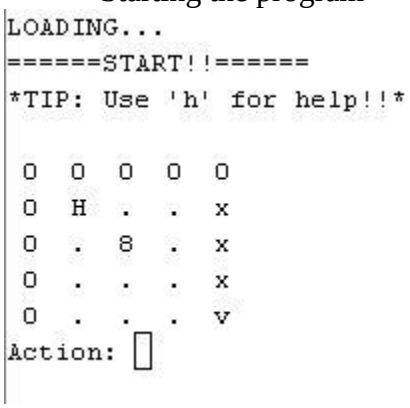
Other aspects that I used that have not been covered in this course are using allocators for sets. I used an allocator to sort the set based on string length. Everything else I haven’t really seen before was a requirement, so I don’t think it counts.

Description

At first, I tried to make a dice game of chance, but it was way too short and not enough to include all the concepts necessary. On the side I tried to make a map-panning program using pure ASCII characters and C++, which, in turn, gave me the idea to mix the two and this eventually led to a Pokemon-style game. Oddly enough, using a sheet of paper to solve problems I had helped a whole lot! Other than that, looking at documentation and redoing concepts on a separate program helped me get through my problems.

Sample Input/Output

Starting the program



Pressing ‘h’ brings up a help menu

```
Action: Legend
 '8' PLAYER
 '~' WATER (water types found here)
 'x' CONCRETE (rock types found here)
 'v' TALL GRASS (grass types found here)
 '.' NORMAL TILE (Items found here)
 'H' HOSPITAL (Buy medicine here)

Actions
 In world
    w move up
    s move down
    a move left
    d move right
    p PokeDex menu
    h brings up this menu
    q quits the game
    ~dying also stops the game
 In battle
    1 attacks rival Pokemon
    2 attempts to catch Pokemon
    3 runs from battle
    4 uses health potion for your Pokemon
Casually walking around gives chance of finding items!
```

Pressing ‘q’ quits the game (output varies)

```
SHUTTING DOWN...

Your attack log:

Sorted log to count number of times an action was used

Attacks used: 0
Catches used: 0
Times Ran:    0
Times Healed: 0
```

Pressing ‘p’ displays the “PokeDex”

```
=======PokeDex========
These are all the Pokemon found in this world
Weak Rock
Weak Grass
Weak Water
Strong Rock
Strong Grass
Strong Water
Intermediate Rock
Intermediate Grass
Intermediate Water

=======Nontindo=======
```

Pressing ‘w’ moves the player (‘8’) upward (camera also moves accordingly)

```
 0  0  0  0  0
 0  H  8  .  x
 0  .  .  .  x
 0  .  .  .  x
 0  .  .  .  v
Action:
```

Pressing ‘s’ moves the player (‘8’) downward (camera also moves accordingly)

```
 0  H  .  .  x
 0  .  8  .  x
 0  .  .  .  x
 0  .  .  .  v
 0  .  .  .  v
Action:
```

Pressing ‘d’ moves the player (‘8’) right (camera also moves accordingly)

```
 H  .  .  x  x
 .  .  8  x  x
 .  .  .  x  x
 .  .  .  v  v
 .  .  .  v  v
Action:
```

Pressing ‘a’ moves the player (‘8’) left (camera also moves accordingly)

```
 0  H  .  .  x
 0  8  .  .  x
 0  .  .  .  x
 0  .  .  .  v
 0  .  .  .  v
Action:
```

Walking on ‘H’ takes you to the hospital

```
Shop? (NOT YET IMPLEMENTED)
***Items in stock***
forwards: Full Half Small

backwards:Small Half Full
```

Walking on ‘.’ tiles has a chance of finding a heal potion (output varies)

```
 Found half heal!!

 O  .  .  .  x
 O  .  .  .  v
 O  8  .  .  v
 O  ~  ~  ~  v
 O  O  O  O  O
Action:
```

Walking on ‘~’ tiles has a chance of a Water-type Pokemon attacking (Pokemon strength varies)

```
A Weak Water Level 2 Appeared!!

Your Pokemon at hand: Weak Grass Level 3

Your Pokemon's Health: 10
Weak Water's Health: 8

What do you do?
1) Attack
2) Catch
3) Run
4) Use Heal
Action:
```

Walking on ‘v’ tiles has a chance of a Grass-type Pokemon attacking (Pokemon strength varies)

```
A Weak Grass Level 2 Appeared!!

Your Pokemon at hand: Weak Grass Level 3

Your Pokemon's Health: 9
Weak Grass's Health: 5

What do you do?
1) Attack
2) Catch
3) Run
4) Use Heal
Action:
```

Walking on ‘x’ tiles has a chance of a Rock-type Pokemon attacking (Pokemon strength varies)

```
A Intermediate Rock Level 6 Appeared!!

Your Pokemon at hand: Weak Grass Level 3

Your Pokemon's Health: 9
Intermediate Rock's Health: 14

What do you do?
1) Attack
2) Catch
3) Run
4) Use Heal
Action:
```

Selecting ‘1) Attack’ during battle

```
Your Attacks
Tackle  Power: 1

Your Weak Grass used Defend!

Weak Water used Defend!

Your Pokemon's Health: 8
Weak Water's Health: 7

What do you do?
1) Attack
2) Catch
3) Run
4) Use Heal
Action:
```

Selecting ‘1) Attack’ during battle if the attacker failed to attack

```
Your Attacks
Tackle  Power: 1

Your Weak Grass used Defend!

Intermediate Grass failed to attack!

Your Pokemon's Health: 9
Intermediate Grass's Health: 9

What do you do?
1) Attack
2) Catch
3) Run
4) Use Heal
Action:
```

Selecting ‘1) Attack’ and winning (Pokemon type and level up varies)

```
Intermediate Grass killed!
Your Weak Grass leveled up 2!
```

Selecting ‘1) Attack’ and losing. ‘CIN ERROR’ displayed because I used arrow keys for the purpose of displaying ‘CIN ERROR’ on the log (output varies greatly)

```
====Your Pokemon died!====
====Game Over====

Your Pokemon

Weak Grass
Weak Grass


Your attack log:
  MOVE: 1  USED: RUN
  MOVE: 2  USED: CIN ERROR
  MOVE: 3  USED: ATTACK
  MOVE: 4  USED: ATTACK
  MOVE: 5  USED: ATTACK
  MOVE: 6  USED: ATTACK
  MOVE: 7  USED: ATTACK
  MOVE: 8  USED: ATTACK
  MOVE: 9  USED: ATTACK
  MOVE: 10  USED: ATTACK
  MOVE: 11  USED: CIN ERROR
  MOVE: 12  USED: CIN ERROR
  MOVE: 13  USED: CIN ERROR
  MOVE: 14  USED: ATTACK
  MOVE: 15  USED: ATTACK
```

```
Sorted log to count number of times an action was used
ATTACK
ATTACK
ATTACK
ATTACK
ATTACK
ATTACK
ATTACK
ATTACK
ATTACK
ATTACK
CIN ERROR
CIN ERROR
CIN ERROR
CIN ERROR
RUN

Attacks used: 10
Catches used: 10
Times Ran:    10
Times Healed: 10


====Your Pokemon died!====
====Game Over====

Press any key to quit
```

Selecting ‘2) Catch’ success (swaps your monster)

```
Action: Successfully caught a Weak Grass!
```

Selecting ‘2) Catch’ fail (does nothing, attacker attacks or misses)

```
Action: Failed to catch Weak Grass
```

Selecting ‘3) Run’ flees from battle

```
Got Away Safely
```

Selecting ‘4) Use Heal’ uses a found or bought health potion (output varies)

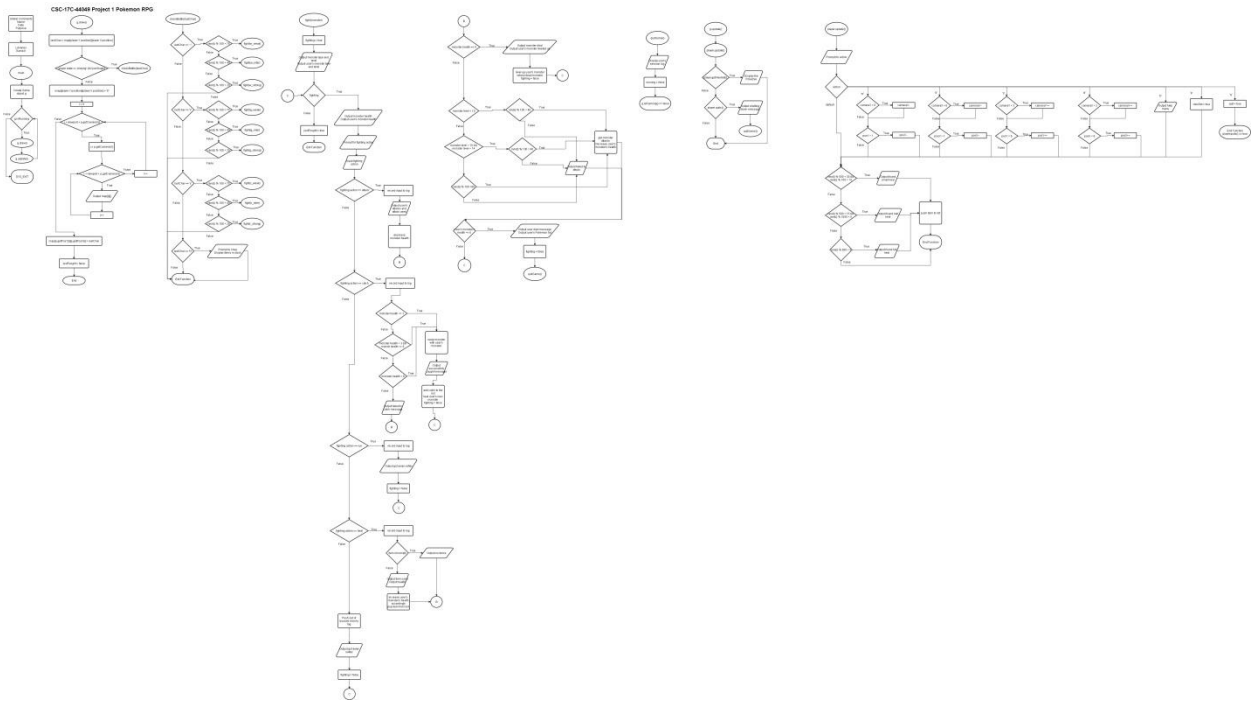
```
Used small heal!

Your Pokemon now has 10 health points!
```

Selecting ‘4) Use Heal’ without items

```
NO ITEMS!
```

Flowchart



Pseudocode

```
while game.isRunning() do
    game.draw()
```

```
        game.update()
exit

game.draw()
        lastChar = map[playerPosY][playerPosX]

        if player is playing and !justFought do
                checkBattle(lastChar)

        i = playerCameraY
        for i < viewport + playerCameraY do
                j = playerCameraX
                for j < viewport + playerCameraX do
                        output map[i][j]
                        j++
                i++
        justFought = false

checkBattle(lastChar)
        check lastChar
                if (water or grass or rock) do
                        if (chance) do
                                fight(monster)
                if (hospital) do
                        output items available to buy

fight(monster)
        fighting = true
        output who is fighting

        while (fighting) do
                output health of both monsters
                prompt for fighting action

                if (attack) do
                        log the action
                        output available attacks
                        attack the monster and decrease its health
                if (catching) do
                        log the action
                        check health of monster
                                if (chance) do
                                        swap monsters
                                        add newly caught monster to list
                                        heal the monster
                                        fighting = false
```

```
                        else do
                                output failed to catch message
                if (run from battle) do
                        log the action
                        fighting = false
                if (healing) do
                        log the action
                        if (empty items list) do
                                output no items message
                        else do
                                output item used
                                heal accordingly
                                pop item from list of items
                else do
                        log the action
                        fighting = false

                if (monster health == 0) do
                        level up user's monster
                        full heal dead monster
                        fighting = false

                check monster's level
                        if (chance) do
                                reduce user's monster's health accordingly
                        else do
                                output monster missed or failed to attack message

                if (user's monster's health <= 0) do
                        output monster died message
                        output user's  monster list
                        fighting = false
                        quitGame()

quitGame()
        display player's action log
        running = false

game.update()
        player.update()
        if (player.getViewSet()) do
                display set or "PokeDex"
        if (player.quits()) do
                quitGame()

player.update()
```

```
prompt for action
if (action == 'w' or 'a' or 's' or 'd') do
        move player and camera accordingly
if (action == 'h') do
        output a help menu
if (action == 'p') do
        viewSet = true
if (action == 'q') do
        quit = true
default do nothing

if (chance) do
        output found item
        push item to item list
else do nothing
```

Variables

Variable	Type	Location
pokeDex	set<string, SortOrder>	Game.h:74 Game.cpp:105 – 115, 140, 506
log	list<string>	Game.h:76 Game.cpp:262, 276, 299, 365, 373, 395, 520, 537, 540
mList	MonList, linked list	Game.h:79 Game.cpp: 64, 65, 143, 146 – 149, 240, 309, 328, 347, 415, 419, 483
sList_H	Shop, doubly linked list	Game.h:80 Game.cpp:76, 222
sList_T	Shop, doubly linked list	Game.h:81 Game.cpp:77, 223
mIt	map<string, int>::iterator	Game.h:100 Game.cpp:280 – 282, 289, 291, 294, 435, 437, 449, 451, 463
sIt	set<string>::iterator	Game.h:101 Game.cpp:506, 507
lIt	list<string>::iterator	Game.h:102 Game.cpp:520, 524, 540, 541, 545 - 548
items	stack<string>	Player.h:21, 26, 40, 55 Player.cpp:91, 96, 101
attacks	map<string, int>	Pokemon.h:40, 41, 45 – 47, 51 – 54, 59, 60, 64 – 66, 70 – 74, 78, 79, 83 – 85, 89 – 92, 99, 144, 153

Concepts

Variable	Type	Location
swap	Algorithm	Game.cpp:305, 324, 343,
sort	Algorithm	Game.cpp:537
mIt	Iterator	Game.h:100 Game.cpp:280 – 282, 289, 291, 294, 435, 437, 449, 451, 463
sIt	Iterator	Game.h:101 Game.cpp:506, 507
lIt	Iterator	Game.h:102 Game.cpp:520, 524, 540, 541, 545 - 548
pokeDex	Container	Game.h:74 Game.cpp:105 – 115, 140, 506
log	Container	Game.h:76 Game.cpp:262, 276, 299, 365, 373, 395, 520, 537, 540
mList	Container	Game.h:79 Game.cpp: 64, 65, 143, 146 – 149, 240, 309, 328, 347, 415, 419, 483
sList_H	Container	Game.h:80 Game.cpp:76, 222
sList_T	Container	Game.h:81 Game.cpp:77, 223
items	Container	Player.h:21, 26, 40, 55 Player.cpp:91, 96, 101
attacks	Container	Pokemon.h:40, 41, 45 – 47, 51 – 54, 59, 60, 64 – 66, 70 – 74, 78, 79, 83 – 85, 89 – 92, 99, 144, 153

References

- Previous homework
- Gaddis book
- Explains advanced C++ concepts
  https://www.youtube.com/channel/UCcDGsN3JxMavDkM9INRLGFA
  https://www.youtube.com/user/CodingMadeEasy
- Used to learn about libraries (documentation)

http://www.cplusplus.com/reference/

- NetBeans' autocorrect feature with man pages

Program

main.cpp

```
/*
 * File: main.cpp
 * Author: Najera Enrique
 * Purpose: Main game loop for Project 1 - Pokemon
 * Date Due: 17 April 2017
 */

// User Libraries
#include "Game.h"

// Start method main handles game loop
int main(int argc, char** argv) {
    // Declare Objects
    Game g;

    // After game has been initialized
    while (g.isRunning()){
        g.draw();   // Draw our game at default settings
        g.update(); // Update our game
    }

    // SYS_EXIT
    return 0;
}// End method main
```

Game.h

```
/*
 * File: Game.h
 * Author: Najera Enrique
 * Date Due: 17 April 2017
 * Purpose: The game's main logic and drawing
 */

#ifndef GAME_H
#define GAME_H

// System Libraries
```

```
#include <map>
#include <set>
#include <list>

// User Libraries
#include "Player.h"
#include "Pokemon.h"
#include "MonsterList.h"
#include "ShopList.h"

// Sorts the pokeDex set by string length
struct SortOrder {
   // Operator overload
   bool operator()(const string &first, const string &second){
      // Get the length of strings
      int length1 = first.length();
      int length2 = second.length();

      // If same length, return alphabetical order
      if (length1 == length2)
         return (first < second);
      return (length1 < length2);
   }
};

// Start class Game
class Game{
   public:
      // Constructor & Destructor
      Game();
      ~Game();

      // Function Prototypes
      bool isRunning(){return running;} // Should our game run
      void update();                 // Update logic
      void draw();                   // Draw based on updated logic
      void quitGame();               // Quits the game
      void checkBattle(char);        // Check if we are in a battle state
      void fight(Pokemon *);         // Actual fighting state
      void dispSet();                // Displays the set, or PokeDex
      void dispLog();                // Displays the list, or program log
      Shop *stockShop(char);         // Fills doubly linked list with items
      void printShop_F(Shop *);      // Print shop in a forward fashion
      void printShop_B(Shop *);      // Print shop in a backward fashion

      // -Link List Functions
```

```
    void addBefore(MonList *, string, string);
    void prntList(MonList *);

private:
    // Declare Variables
    bool running;    // is/should the game (be) running
    bool justFought; // Did the player just fight
    int fAction;     // Holds fighting action player has provided

    char map[8][8];  // Map/World container
                     // Was dynamic and read from a file, but
                     // that caused too much seg_fault errors
                     // and delimiter errors
    int viewport;    // How much of the world to display on camera

    // Declare Objects
    std::set<string, SortOrder>pokeDex; // Holds all pokemon names
                                        // in the current world
    std::list<string>log;               // Holds log of player input

    Player p;       // Player
    MonList *mList; // List of Pokemon the player has
    Shop *sList_H;  // List of item in shop (holds the Head)
    Shop *sList_T;  // List of item in shop (holds the Tail)

    // -Pokemon Objects
    Pokemon *starter;  // Starter, or default, Pokemon
    Pokemon *current;  // Holds current Pokemon our player is using

    Pokemon *w_weak;   // Weak Water
    Pokemon *w_inter;  // Intermediate Water
    Pokemon *w_strong; // Strong Water

    Pokemon *g_weak;   // Weak Grass
    Pokemon *g_inter;  // Intermediate Grass
    Pokemon *g_strong; // Strong Grass

    Pokemon *r_weak;   // Weak Rock
    Pokemon *r_inter;  // Intermediate Rock
    Pokemon *r_strong; // Strong Rock

    // Declare iterators
    std::map<string, int>::iterator mIt; // Loops through Pokemon attacks
    std::set<string>::iterator sIt;      // Loops through PokeDex
    std::list<string>::iterator lIt;     // Loops through log
```

```
};// End class Game

#endif /* GAME_H */
```

Game.cpp

```
/*
 * File: Game.cpp
 * Author: Najera Enrique
 * Date Due: 17 April 2017
 * Purpose: Main Game container
 *          Builds and draws map,
 *          Creates objects,
 *          Handles Game state: fighting, menu/pause
 */

// User Libraries
#include "Game.h"

// System Libraries
#include <iostream>
#include <iomanip>   // setw()
#include <cstdlib>   // rand()
#include <ctime>     // time()
#include <set>
#include <typeinfo>  // typeid()
#include <algorithm> // sort()

using namespace std;

// Start  constructor Game
Game::Game(){
   cout << "LOADING...\n";
   // Set random number seed
   srand(time(0));

   // Initialize variables
   viewport = 5;       // View up to 5 elements on X and Y
                       // Changing this could cause seg_fault
                       // If change is wanted, however, must also
                       // change the Player object's "if (camera_ < [int]) camera_++;"
                       // where '_' means 'X' or 'Y' and [int] means any integer value
                       // Its all trial and error and depends on the 'map' array size!!

   justFought = false; // Checks if player just fought a monster
                       // to prevent a fight loop on the same tile
```

```
// Create our Pokemon
// Format (type, health, level, power)
// -Starter
starter = new Pokemon("Weak Grass", 10, 3, 3);
current = starter; // Our current Pokemon

// -Water
w_weak = new Pokemon("Weak Water", 8, 2, 2);
w_inter = new Pokemon("Intermediate Water", 12, 5, 4);
w_strong = new Pokemon("Strong Water", 18, 9, 6);

// -Grass
g_weak = new Pokemon("Weak Grass", 5, 2, 1);
g_inter = new Pokemon("Intermediate Grass", 10, 5, 3);
g_strong = new Pokemon("Strong Grass", 14, 9, 5);

// -Rock
r_weak = new Pokemon("Weak Rock", 6, 2, 2);
r_inter = new Pokemon("Intermediate Rock", 14, 6, 6);
r_strong = new Pokemon("Strong Rock", 20, 9, 8);

// Fill our list with the starter
mList = new MonList;
MonList *prev = mList;
prev->data = starter->getType();
prev->linkPtr = NULL;
MonList *end=new MonList;
end->data=starter->getType();
end->linkPtr=NULL;
prev->linkPtr=end;
prev=end;
// End fill list

// Fill our shop list
sList_H = stockShop('h');
sList_T = stockShop('t');

// Create the map
for (int i = 0; i < 8; i++){
    for (int j = 0; j < 8; j++){
        // Create Border
        if (i == 0 || j == 0 ||
            j == 7 || i == 7)
            map[i][j] = '0';
        // Tall Grass
```

```
            else if (i >= 4 && j >= 4 && map[i][j] != '0')
                map[i][j] = 'v';
            // Concrete
            else if (i <= 4 && j >= 4 && map[i][j] != '0')
                map[i][j] = 'x';
            // Water
            else if (i == 6 && map[i][j] != 'x')
                map[i][j] = '~';
            // Hospital
            else if (i == 1 && j == 1)
                map[i][j] = 'H';
            // Floor
            else map[i][j] = '.';
        }
    }
    // End creating map

    // Create our pokedex
    pokeDex.insert("Weak Grass");
    pokeDex.insert("Weak Water");
    pokeDex.insert("Weak Rock");

    pokeDex.insert("Intermediate Grass");
    pokeDex.insert("Intermediate Water");
    pokeDex.insert("Intermediate Rock");

    pokeDex.insert("Strong Grass");
    pokeDex.insert("Strong Water");
    pokeDex.insert("Strong Rock");

    p.setViewSet(false);

    // Start our loop once elements successfully initialized
    cout << "======START!!======\n";
    cout << "*TIP: Use 'h' for help!!*\n\n";
    running = true;
}// End constructor Game

// Start destructor Game
Game::~Game(){
    // Destroy our Pokemon objects
    delete w_weak;
    delete w_inter;
    delete w_strong;
    delete g_weak;
    delete g_inter;
```

```
    delete g_strong;
    delete r_weak;
    delete r_inter;
    delete r_strong;

    // Delete the set just in case
    // No pointers so we can just clear it
    pokeDex.clear();

    // Destroy MonList if it contains elements
    if (mList==NULL) ;
    else{
        do{
            MonList *temp=mList->linkPtr;
            delete mList;
            mList = temp;
        }while (mList!=NULL);
    }
}// End destructor Game

// Start method update
void Game::update(){
    p.update(); // Update our player

    // If player wants to see PokeDex, display
    if (p.getViewSet())dispSet();

    // If player quits, end game
    if (p.quits()){
        cout << "\nSHUTTING DOWN...\n";
        quitGame();
    }
}// End method update

// Start method checkBattle
// Checks if our player will fight
void Game::checkBattle(char lastCh){
    // Check water types
    if (lastCh == '~'){
        // 50% chance of a weak pokemon
        if (rand() % 100 < 75){
            fight(w_weak);
        }
        // 40 % chance of an intermediate pokemon
        else if (rand() % 100 < 40){
            fight(w_inter);
```

```
    }
    // 20% chance of a strong pokemon
    else if (rand() % 100 < 20){
        fight(w_strong);
    }
}

// Check grass type
if (lastCh == 'v'){
    // 50% chance of a weak pokemon
    if (rand() % 100 < 50){
        fight(g_weak);
    }
    // 40 % chance of an intermediate pokemon
    else if (rand() % 100 < 30){
        fight(g_inter);
    }
    // 20% chance of a strong pokemon
    else if (rand() % 100 < 20){
        fight(g_strong);
    }
}

// Check rock type
if (lastCh == 'x'){
    // 50% chance of a weak pokemon
    if (rand() % 100 < 75){
        fight(r_weak);
    }
    // 40 % chance of an intermediate pokemon
    else if (rand() % 100 < 40){
        fight(r_inter);
    }
    // 20% chance of a strong pokemon
    else if (rand() % 100 < 20){
        fight(r_strong);
    }
}

// Check if in hospital
if (lastCh == 'H'){
    cout << "\nShop? (NOT YET IMPLEMENTED)\n";
    cout << "***Items in stock***\n";
    cout << "forwards: ";printShop_F(sList_H);
    cout << "\nbackwards:";printShop_B(sList_T);
    cout << endl;
```

```
    }
}// End method checkBattle

// Start method fight
// Puts our game into a fighting state
void Game::fight(Pokemon *monster){
    // Declare Variables
    bool fighting = true; // Loops our fighting state

    //log.push_back(monster->getType()); UGLY OTUPUT

    // Output who approached us
    cout << "\nA " << monster->getType()
         << " Level " << monster->getLevel() << " Appeared!!\n";
    // Output our Pokemon's current state
    cout << "\nYour Pokemon at hand: " << mList->data
         << " Level " << current->getLevel() << endl;

    // The fighting loop
    while (fighting){
        // Output health of both
        cout << "\nYour Pokemon's Health: " << current->getHealth() << endl;
        cout << monster->getType() << "'s Health: " << monster->getHealth() << endl;

        // Prompt for player fighting action
        cout << "\nWhat do you do?\n";
        cout << "1) Attack " << endl
             << "2) Catch  " << endl
             << "3) Run    " << endl
             << "4) Use Heal" << endl
             << "Action: ";

        cin >> fAction;

        // If bad input, just leave
        if (cin.fail()) {
            // Log cin fail as 999
            log.push_back("CIN ERROR");
            cout << "\nGot Away Safely\n";
            cin.ignore();
            cin.clear();
            fighting = false;
            return;
        }

        // Static cast to prevent wrong type errors
```

```
fAction = static_cast<int>(fAction);

// Chose to attack
if (fAction == 1){
   // Insert to log
   log.push_back("ATTACK");
   // Outputs user's available attacks
   cout << "\nYour Attacks\n";

   for (mIt = current->getAttacks().begin();
         mIt != current->getAttacks().end(); mIt++){
      cout << mIt->first << " " << " Power: " << mIt->second;
      cout << endl;
   }

   // Prompt for attack (SOON)

   // Output attack used
   mIt = current->getAttacks().begin();
   cout << "\nYour " << current->getType() << " used "
       << mIt->first << "!" << endl;

   // Decrease offending monster's attack
   monster->hit(mIt->second);
}
// Chose to catch
else if (fAction  == 2){
   // Insert to log
   log.push_back("CATCH");

   // Dying health = easier catch (80%)
   if (monster->getHealth() <= 3){
      if (rand() % 100 < 80){
         // Swap algorithm changes player's monster
         std::swap(current, monster);
         cout << "Successfully caught a " << current->getType()
               << "!" << endl;
         // Add catch to our list
         addBefore(mList, monster->getType(), current->getType());
         // Fully heal new Pokemon
         current->heal("full");
         // Leave the fighting state
         fighting = false;
         return ;
      }
      else {
```

```
            cout << "Failed to catch " << monster->getType() << endl;
        }
    }
    // Sick health = decent chance (50%)
    else if (monster->getHealth() > 3 && monster->getHealth() <=5){
        if (rand() % 100 < 50){
            // Swap algorithm changes player's monster
            std::swap(current, monster);
            cout << "Successfully caught a " << current->getType()
                 << "!" << endl;
            // Add catch to our list
            addBefore(mList, monster->getType(), current->getType());
            // Fully heal new Pokemon
            current->heal("full");
            // Leave the fighting state
            fighting = false;
            return ;
        }
        else {
            cout << "Failed to catch " << monster->getType() << endl;
        }
    }
    // Healthy = small chance (2%)
    else if (monster->getHealth() > 6){
        if (rand() % 100 < 2){
            // Swap algorithm changes player's monster
            std::swap(current, monster);
            cout << "Successfully caught a " << current->getType()
                 << "!" << endl;
            // Add catch to our list
            addBefore(mList, monster->getType(), current->getType());
            // Fully heal new Pokemon
            current->heal("full");
            // Leave the fighting state
            fighting = false;
            return ;
        }
        else {
            cout << "Failed to catch " << monster->getType() << endl;
        }
    }
    // Other, output failed to catch message
    else { cout << "Failed to catch " << monster->getType() << endl; }

}
// Chose to run
```

```
else if (fAction == 3){
    // Insert to log
    log.push_back("RUN");
    cout << "\nGot Away Safely\n";
    fighting = false;
    return;
}
// Choose to heal player's monster
else if (fAction == 4){
    // Log this event
    log.push_back("HEAL");

    // If no items, leave
    if (p.getItems().empty()){
        cout << "\nNO ITEMS!" << endl;
    }
    // If item, heal Pokemon accordingly
    else {
        cout << "\nUsed " << p.getItems().top() << " heal!" << endl;
        current->heal(p.getItems().top());

        cout << "\nYour Pokemon now has " << current->getHealth()
             << " health points!" << endl;

        // Remove item from list
        p.popItems();
    }

}
// If error, just leave
else {
    // Insert to log
    log.push_back("OUT OF BOUNDS ERROR");
    cout << "\nGot Away Safely\n";
    fighting = false;
    return;
}

        /*** Monster Action Handler ***/

// Check if monster is dead before attacking
// BUG SINCE I'M USING LINKED LIST FOR DISPLAYING POKEMON TYPE!!
if (monster->getHealth() == 0){
    cout << "\n" << monster->getType() << " killed!" << endl;
    cout << "Your " << current->getType() << " leveled up "
         << monster->getLevel() / 2 << "!" << endl;
```

```
    // Level up our monster
    current->lvlUp(monster->getLevel() / 2);

    // Change type if level is high
    if (current->getLevel() >= 5 && current->getLevel() <= 13 ){
        addBefore(mList, current->getType(), "Intermediate Grass"); // Add to list first!
        current->setType("Intermediate Grass"); // Should split 'type' & 'element'
    }
    else if (current->getLevel() >= 14 && current->getType() != "Strong Grass"){
        addBefore(mList, current->getType(), "Intermediate Grass"); // Add to list first!
        current->setType("Strong Grass");       // Should split 'type' & 'element'
    }

    // Reset attacker's health to prevent infinite level up
    monster->heal("full");

    // Leave fighting state
    fighting = false;
    return ;
}

// Low level 40% chance of attacking
if (monster->getLevel() < 10){
    if (rand() % 100 < 40){
        // Outputs attack used
        mIt = monster->getAttacks().begin();
        cout << endl << monster->getType()
             << " used "  << mIt->first << "!\n";

        current->hit(monster->getPower()); // Decrease player health
    }
    else {
        cout << endl << monster->getType() << " failed to attack!\n";
    }
}
// Intermediate 60% chance of attacking
else if (monster->getLevel() > 10 && monster->getLevel() < 14){
    if (rand() % 100 < 60){
        // Outputs attack used
        mIt = monster->getAttacks().begin();
        cout << endl << monster->getType()
             << " used "  << mIt->first << "!\n";

        current->hit(monster->getPower()); // Decrease player health
    }
```

```
            else {
                cout << endl << monster->getType() << " failed to attack!\n";
            }
        }
        // High levels 80% attack
        else {
            if (rand() % 100 < 80){
                // Outputs attack used
                mIt = monster->getAttacks().begin();
                cout << endl << monster->getType()
                    << " used "  << mIt->first << "!\n";

                current->hit(monster->getPower()); // Decrease player health
            }
            else {
                cout << endl << monster->getType() << " failed to attack!\n";
            }
        }
        // End monster attack handler

        // If our Pokemon's health has dropped to or below 0
        // End game
        if (current->getHealth() <= 0){
            // Output game over message
            cout << "\n====Your Pokemon died!====\n";
            cout << "====Game Over====\n";

            // Print the list of Pokemon the player had
            prntList(mList);

            // Leave this loop
            fighting = false;

            // Leave the game
            quitGame();
            cout << "\n====Your Pokemon died!====\n";
            cout << "====Game Over====\n";
            cout << "\nPress any key to quit\n\n";
        }

    }// End fighting loop

    // Gives 1 step delay before Pokemon appear
    justFought = true;
}// End method fight
```

```
// Start method dispSet
void Game::dispSet(){
   cout << "\n=======PokeDex========\n";
   cout << "These are all the Pokemon found in this world\n";
   // Go through PokeDex set and output
   for (sIt = pokeDex.begin(); sIt != pokeDex.end(); sIt++)
      cout << *sIt << endl;
   cout << "\n=======Nontindo=======\n";
   p.setViewSet(false);
}// End method dispSet

// Start method dispLog displays the log
// Log contains user actions
void Game::dispLog(){
   // Declare Variables
   int i = 0; // Move counter
   cout << "\nYour attack log:\n";

   // Loop through log
   for (lIt = log.begin(); lIt != log.end(); lIt++){
      i++; // Increment move counter
      // Output move number and what user used against monster
      cout << setw(8) << "MOVE: " << i
           << setw(5) << "  USED: " << *lIt << endl;
   }// End for loop

   cout << "\nSorted log to count number of times "
        << "an action was used\n";

   // Declare counter variables
   int numAttack = 0;
   int numCatch  = 0;
   int numRun    = 0;
   int numHeal   = 0;

   // Sort algorithm
   log.sort();

   // Output and count
   for (lIt = log.begin(); lIt != log.end(); lIt++){
      cout << *lIt << endl;
      // BUG: DOESN'T COUNT FOR EACH
      // ONLY COUNTS FOR ONE ("ATTACK")!!
      // DEBUGGER'S CONSOLE DOESNT COOPERATE!!
      if (*lIt == "ATTACK")     numAttack++;
      else if (*lIt == "CATCH") numCatch++;
```

```
        else if (*lIt == "RUN")   numRun++;
        else if (*lIt == "HEAL")  numHeal++;
    }

    // Output count results
    cout << endl;
    cout << "Attacks used: " << numAttack << endl;
    cout << "Catches used: " << numAttack << endl;
    cout << "Times Ran:    " << numAttack << endl;
    cout << "Times Healed: " << numAttack << endl;
    cout << endl;
}// End method dispLog

// Start method quitGame
void Game::quitGame(){
    dispLog();       // Display the log
    running = false; // Quit our game loop
}// End method quitGame

// Start method draw
// Draws our game elements
void Game::draw(){
    // Get the last tile our player stepped on to overwrite it
    char lastChar = map[p.getPosY()][p.getPosX()];

    // Check if a Pokemon has approached us
    // If our player just fought, skip this
    if (p.getState() == 'p' && !justFought)checkBattle(lastChar);

    // Place our player in the world
    map[p.getPosY()][p.getPosX()] = '8';

    // Draw our map/world
    for (int i = p.getCameraY(); i < viewport + p.getCameraY(); i++){
        for (int j = p.getCameraX(); j < viewport + p.getCameraX(); j++){
            cout << setw(2) << map[i][j] << " ";
        }
        cout << endl;
    }

    // Overwrite the last character
    map[p.getPosY()][p.getPosX()] = lastChar;

    // We did not just fight
    justFought = false;
```

```
}// End method draw


      /*** Link List Functions ***/

// Start method addBefore
// BUG ADDS MORE THAN ONE!!
void Game::addBefore(MonList *front, string before, string val){
   MonList *next = front;          // Keeps track of next node
   MonList *prev = new MonList;    // Stores previous node
   MonList *newNode  = new MonList; // Creates new node for next value

   newNode->data = val; // Store value in newNode's data

   // Go through list until it hits position wanted
   while (next->linkPtr != NULL && next->data != before){
      // Clone everything before 'before'
      prev = next;
      next = next->linkPtr;
   }

   // Store newNode into the linked list
   prev->linkPtr = newNode;
   newNode->linkPtr = next;
}// End method addBefore

// Start method printList prints our Pokemon all game
void Game::prntList(MonList *front){
   cout << "\nYour Pokemon\n";
   MonList *next=front; //Start at the front of the list
   cout<<endl;         //Put the beginning on a new line
   do{
      cout<<setw(4)<<next->data<<" "; //Print the link
      next=next->linkPtr;            //Go to the next link
      cout << endl;
   }while(next!=NULL);               //Stop when your at the end
   cout<<endl;

}// End method printList

// Start method stockShop
// Takes in char 'h' for returning the HEAD
// Takes in char 't' for returning the TAIL
// else return HEAD
Shop *Game::stockShop(char loc){
   Shop *head; // Head
```

```
    Shop *tail; // End
    Shop *n;    // Next

    // Full heal
    n = new Shop;
    n->data = "Full";
    n->prev = NULL;   // First node has no previous
    head = n;
    tail = n;

    // Half heal
    n = new Shop;
    n->data = "Half";
    n->prev = tail;
    tail->next = n;
    tail = n;

    // Small heal
    n = new Shop;
    n->data = "Small";
    n->prev = tail;
    tail->next = n;
    tail = n;

    // Close list
    tail->next = NULL;

    // Check the argument for proper return
    if (loc == 't' || loc == 'T') return tail;
    else return head;
}// End method stockShop

// Start method printShop_F
void Game::printShop_F(Shop *head){
    Shop *temp = head; // Points to front of list

    // Print while data
    do {
        cout << temp->data << " ";
        temp = temp->next; // Point to next node
    }while(temp != NULL);
    cout << endl;
}// End method printShop_F

// Start method printShop_B
void Game::printShop_B(Shop *tail){
```

```
    Shop *temp = tail; // Points to end of list

    // Print while data
    do {
        cout << temp->data << " ";
        temp = temp->prev; // Point to previous node
    }while(temp != NULL);
    cout << endl;

}// End method printShop_B
```

Player.h

```
/*
 * File: Player.h
 * Author: Najera Enrique
 * Date Due: 17 April 2017
 * Purpose: Contains our Player properties
 *          Also updates movement
 */

#ifndef PLAYER_H
#define PLAYER_H

// System Libraries
#include <stack>
#include <string>

// Start class Player
class Player{
    public:
        // Constructor & Destructor
        Player();
        ~Player(){items.empty();}

        // Function Prototypes
        void update();   // Handle user input
        void outHelp();  // Outputs a help page
        void popItems(){items.pop();} // Uses the stack's 'pop' algorithm

        // Mutator Functions
        void setState(char s){state = s;}     // Sets player's state
        void setViewSet(bool b){viewSet = b;} // Sets if PokeDex set
                                              // should be displyed

        // Accessor Functions
```

```
        int getCameraX() const {return cameraX;}   // Get our camera's x position
        int getCameraY() const {return cameraY;}   // Get our camera's y position
        int getPosX()    const {return posX;}      // Get our player's x position
        int getPosY()    const {return posY;}      // Get our player's y position
        char getState()  const {return state;}     // Get our player's state
        bool getViewSet() const { return viewSet;} // View pokeDex set?
        std::stack<std::string> getItems() const {return items; } // Get inventory
        bool quits() const {return quit;};         // Did player just quit

    private:
        // Declare Variables
        int cameraX;  // Holds camera's x position
        int cameraY;  // Holds camera's y position
        int posX;     // Holds player's x position
        int posY;     // Holds player's y position
        char action;  // Holds player input
        char state;   // Holds player's state
        bool viewSet; // Tells game to display PokeDex set
        bool quit;    // Holds if player decided to quit

        // Declare Objects
        std::stack<std::string>items; // Holds all our items
};// End class Player

#endif /* PLAYER_H */
```

Player.cpp

```
/*
 * File: Player.cpp
 * Author: Najera Enrique
 * Date Due: 17 April 2017
 * Purpose: Contains our Player properties
 *          Also updates movement
 */

// User Libraries
#include "Player.h"

// System Libraries
#include <iostream>
#include <cstdlib>  // rand()
using namespace std;

// Start constructor Player
```

```
Player::Player(){
   // INIT variables
   cameraX = 0;
   cameraY = 0;
   posX = 2;     // Place at center of camera
   posY = 2;     // Place at center of camera
   state = 'p';  // State playing
   action = ' '; // No action taking place
   quit = false;
}// End constructor Player

// Start method update
void Player::update(){
   // Prompt for action
   cout << "Action: ";
   cin >> action;

   // Action handler
   switch(action){
      // If playing in the world
      // UP
      case 'w':
         // Moves everything UP
        // if (state == 'p'){ /!\ ERROR: makes player freeze
            if (cameraY > 0)cameraY--; // If camera in bounds
            if (posY > 1)posY--;       // If player in bounds
         //}
         break;
      // LEFT
      case 'a':
         // Moves everything LEFT
        // if (state == 'p'){ /!\ ERROR: makes player freeze
         if (cameraX > 0)cameraX--; // If camera in bounds
         if (posX > 1)posX--;//}    // If player in bounds
         break;
      // DOWN
      case 's':
         // Moves everything DOWN
         //if (state == 'p'){ /!\ ERROR: makes player freeze
         if (cameraY < 3)cameraY++; // If camera in bounds
         if (posY < 6)posY++;//}    // If player in bounds
         break;
      // RIGHT
      case 'd':
         // Moves everything RIGHT
         //if (state == 'p'){ /!\ ERROR: makes player freeze
```

```
            if (cameraX < 3)cameraX++; // If camera in bounds
            if (posX < 6)posX++;//}     // If player in bounds
            break;
        // HELP
        case 'h':
            // Displays help page
            //if (state == 'p'){ /!\ ERROR: makes player freeze
            outHelp();//}
            break;
        // PAUSE (just displays PokeDex)
        case 'p':
            viewSet = true;
            break;
        // QUIT
        case'q':
            // Acknowledges game that player quit
            quit = true;
            break;
        // Else do nothing
        default:
            break;
    }// End action handler

    // After every move, check for an item
    // 6% chance to find 'small' heal
    if (rand() % 500 < 30 && rand() % 100 > 15){
        cout << "\n Found small heal!!\n";
        items.push("small");
    }
    // 3% chance to find 'half' heal
    else if (rand() % 500 < 15 && rand() % 1000 > 5){
        cout << "\n Found half heal!!\n";
        items.push("half");
    }
    // 1% chance to find 'full' heal
    else if (rand() % 500 < 5){
        cout << "\n Found full heal!!\n";
        items.push("full");
    }
    // Found nothing!
    else {}

    cout << endl;
}// End method update

// Start method outHelp
```

```
void Player::outHelp(){
   // Output help page
   cout << "Legend "<<endl
        << " '8' PLAYER\n"
        << " '~' WATER (water types found here)\n"
        << " 'x' CONCRETE (rock types found here)\n"
        << " 'v' TALL GRASS (grass types found here)\n"
        << " '.' NORMAL TILE (Items found here)\n"
        << " 'H' HOSPITAL (Buy medicine here)\n"
        << endl
        << "Actions "<<endl
        << " In world\n"
        << "    w move up\n"
        << "    s move down\n"
        << "    a move left\n"
        << "    d move right\n"
        << "    p PokeDex menu\n"
        << "    h brings up this menu\n"
        << "    q quits the game\n"
        << "    ~dying also stops the game\n"
        << " In battle\n"
        << "    1 attacks rival Pokemon\n"
        << "    2 attempts to catch Pokemon\n"
        << "    3 runs from battle\n"
        << "    4 uses health potion for your Pokemon\n"
        << "Casually walking around gives chance of finding items!\n"
        << endl;
}// End method outHelp
```

Pokemon.h

```
/*
 * File: Pokemon.h
 * Author: Najera Enrique
 * Date Due: 17 April 2017
 * Purpose: Holds Pokemon properties
 */

#ifndef POKEMON_H
#define POKEMON_H

// System Libraries
#include <map>    // Holds Pokemon's power
#include <string>
using namespace std;
```

```
// Start class Pokemon
class Pokemon{
    public:
        // Default constructor
        Pokemon(){
            // INIT everything to 1
            health = 1;
            type = 1;
            level = 1;
        };// End default constructor

        // Constructor
        // Takes in a type, total health, level, power of attacks
        Pokemon(string t, int h, int l, int p){
            health = h;
            maxHealth = h;
            type = t;
            level = l;
            power = p;

            // Check type and assign attacks
            // Map is
            if (type == "Weak Grass"){
                // Weak types only have 2 attacks
                attacks["Tackle"] = 1;
                attacks["Defend"] = 1;
            }
            else if (type == "Intermediate Grass"){
                // Intermediate types have 3 attacks
                attacks["Tackle"] = 1;
                attacks["Whip"] = 3;
                attacks["Absorb"] = 2;
            }
            else if (type == "Strong Grass"){
                // Strong types have all 4 attacks
                attacks["Whip"] = 3;
                attacks["Absorb"] = 2;
                attacks["Photosynthesis"] = 5;
                attacks["Spore"] = 4;
            }

            else if (type == "Weak Water"){
                // Weak types only have 2 attacks
                attacks["Tackle"] = 2;
                attacks["Defend"] = 1;
            }
```

```
    else if (type == "Intermediate Water"){
        // Intermediate types have 3 attacks
        attacks["Tackle"] = 2;
        attacks["Splash"] = 2;
        attacks["Wave"] = 4;
    }
    else if (type == "Strong Water"){
        // Strong types have all 4 attacks
        attacks["Splash"] = 2;
        attacks["Wave"] = 4;
        attacks["Whirlpool"] = 5;
        attacks["Tsunami"] = 6;
    }

    else if (type == "Weak Rock"){
        // Weak types only have 2 attacks
        attacks["Tackle"] = 2;
        attacks["Defend"] = 1;
    }
    else if (type == "Intermediate Rock"){
        // Intermediate types have 3 attacks
        attacks["Tackle"] = 1;
        attacks["Throw"] = 6;
        attacks["Quake"] = 5;
    }
    else if (type == "Strong Rock"){
        // Strong types have all 4 attacks
        attacks["Throw"] = 6;
        attacks["Quake"] = 8;
        attacks["Crush"] = 5;
        attacks["Sandstorm"] = 8;
    }
}; // End Constructor

// Destructor
~Pokemon(){
    // Delete our map just in case
    attacks.clear();
};

// Function Prototypes
void hit(int p){health -= p;}    // Removes health by power of attacker
void lvlUp(int lU){level += lU;} // Levels Pokemon up
void heal(string med){
    // If already at full health, leave
    if (health == maxHealth) return;
```

```
        // If given full medicine, fill health
        // to max health
        if (med == "full"){
            health = maxHealth;
        }
        // If given half medicine, fill health
        // half of max
        else if (med == "half"){
            health += maxHealth / 2;
            // If previous calculation exceeds the max health,
            // health is our maxHealth
            if (health > maxHealth)
                health = maxHealth;
        }
        // If given small dosage, add health by 2
        else if (med == "small"){
            health += 2;
            // If previous calculation exceeds the max health,
            // health is our maxHealth
            if (health > maxHealth)
                health = maxHealth;
        }
        // If some kind of arg error, return
        else { return; }

    }

    // Mutator Functions
    void setType(string t){type = t;}

    // Accessor Functions
    int getHealth() const {return health;}
    string getType() const {return type;}
    int getPower() const {return power;}
    int getLevel() const {return level;}
    map <string, int> getAttacks() const {return attacks;}
private:
    // Declare Variables
    int health;    // Pokemon's total health
    int maxHealth; // Pokemon's maxHealth for healing
    string type;   //Pokemon's type
    int level;     // Pokemon's level
    int power;     // Pokemon's attack power

    map<string, int>attacks; // Holds list of attacks
```

```
                        // Keytype (string) = name of attack
                        // Value (int)      = power of attack
};// End class Pokemon

#endif /* POKEMON_H */
```

MonsterList.h

```
/*
 * File: MonsterList.h
 * Author: Najera Enrique
 * Date Due: 17 April 2017
 * Purpose: Holds a list of monsters the player has caught
 */

#ifndef MONSTERLIST_H
#define MONSTERLIST_H

// System Libraries
#include <string>

//Start structure MonList
struct MonList {
   string data;      // Holds Pokemon's type
   MonList *linkPtr; // Pointer to next data
};// End structure MonList

#endif /* MONSTERLIST_H */
```

ShopList.h

```
/*
 * File: ShopList.h
 * Author: Najera Enrique
 * Date Due: 17 April 2017
 * Purpose: Doubly linked list used as a shop
 *          so the user could buy items
 */

#ifndef SHOPLIST_H
#define SHOPLIST_H

// System Libraries
#include <string>

struct Shop{
```

```
    std::string data;
    Shop *next;
    Shop *prev;
};

#endif /* SHOPLIST_H */
```