```cpp
#include "World.h"

/*  Task: Task 11 - Spike
*   Title: Game Data From Graphs
*   Author: Thomas Horsley - 103071494
*
*   Currently:  None It's done. I'm done. GAWWWWWWD file reading is yuck.
*
*
*   Completed:  Working on using a file reader to instantiate the location game object
*               and there-in the world object from a text file, could use JSON later.
*
*               Will probably end up making the world object more of a world "manager"
*               class. So given this, we're giving it the ability to read and write saves
*               and it will have access to the player object.
*
*               Chopped up the big main.cpp into a bunch of more managable header and
*               implementation files. Also fixed my inconsistent af naming.
*
*               Finished it all and transitioned from using dumb vector to smart maps
*
*       Formatting:
*           To make life easier with the functions that I've already written we're going
*           to split the txt file using a double delimiter tech.
*               * Delimit once to split the file by location
*                   This will return a vector of strings
*               * Iterate through each vector and split their strings into another set
*                 of vectors.
*               * Use this data to instantiate location objects
*
*
*   Summary:
*       So I don't really like this code to be honest. It works yes but instantiation of
*       locations is a really fickle fucker. Additionally the order of the location data
*       must be the same as the order of the locations in the vector. This is an issue for
*       now as the program doesn't write its own save files. Cuz i wrote it and it's stoop.
*
```

```
38  *       World has a set of location pointers. It can read from a file and assign the input
39  *       data to the locations.
40  *
41  *
42  *       Next step is to implement world traversal before chopping this up into class files
43  *       for organisation and shove it into zorkish.......... this is going to be fun and easy :)
44  */
45
46  int main() {
47      //Ahhhhhhh what a nice clean main :)
48      World world;
49      world.loadLocationData();
50
51      while (world.checkIsRunning()) {
52          world.render();
53          world.update();
54      }
55  }
```

```h
#pragma once
#include <iostream>
#include <fstream>
#include <unordered_map>

class Location {
private:
    std::string _name;
    std::string _description;
    std::unordered_map<std::string, Location*> _exits;

public:
    Location(std::string name = " ",
        std::string description = " ",
        std::unordered_map<std::string, Location*> exits = {});

    std::string getName();
    std::string getDescription();
    Location* getExit(std::string direction);
    std::unordered_map<std::string, Location*> getExits();

    void setName(std::string name);
    void setDescription(std::string description);
    void addExit(std::string direction, Location* exit);
    void setAll(std::string name, std::string description,
        std::unordered_map<std::string, Location*> exits);

    void showDetails();
    void showExits();
};


```

```cpp
#include "Location.h"
#include <algorithm>

Location::Location(std::string name,
    std::string description,
    std::unordered_map<std::string, Location*> exits) {
    _name = name;
    _description = description;
    _exits = exits;
}

std::string Location::getName() { return _name; }
std::string Location::getDescription() { return _description; }
Location* Location::getExit(std::string direction) {
    return _exits[direction];
}

std::unordered_map<std::string, Location*> Location::getExits() { return _exits; }

void Location::setName(std::string name) { _name = name; }
void Location::setDescription(std::string description) {
    std::replace(description.begin(), description.end(), '_', ' ');
    _description = description;
}

void Location::addExit(std::string direction, Location* exit) {
    _exits.insert({ direction, exit });
}

void Location::setAll(std::string name, std::string description,
    std::unordered_map<std::string, Location*> exits) {
    _name = name;
    _description = description;
    _exits = exits;
}

//  FYI if you're running anything earlier than C++ 17 this function will break.
void Location::showExits() {
    std::cout << "Exits: " << std::endl;

    for (auto& [direction, exit] : _exits) {
        std::cout << ">> Location: " << exit->getName()
            << " -- Direction: " << direction << std::endl;
    }
}

void Location::showDetails() {
```

```
48      std::cout << "Room: " << _name << std::endl
49          << "Description: " << _description << std::endl;
50      showExits(); std::cout << std::endl;
51  }
```

```h
#pragma once
#include <iostream>
#include <fstream>
#include <vector>

class WorldLoader
{
private:
    std::string _file_name;
    std::ifstream _reader;

    bool isComment(std::string string_data);
    std::vector<std::string> processLineInput(std::string string_data, char delimiter);

public:
    WorldLoader(std::string file_name = "");
    ~WorldLoader();

    std::vector<std::string> splitLine(std::string string_data, char delimiter);
    std::vector<std::string> getLinesByDelimiter(char splitter);
    std::vector<std::string> getLinesByDelimiter(char splitter, std::string string_data);

};


```

```cpp
#include "WorldLoader.h"
#include "string"

bool WorldLoader::isComment(std::string string_data) {
    remove(string_data.begin(), string_data.end(), ' ');

    if (string_data[0] == '#') {
        string_data.erase(string_data.begin(), string_data.end());
        return true;
    }
    return false;
}

std::vector<std::string> WorldLoader::processLineInput(std::string string_data, char delimiter) {
    std::string line = string_data;
    std::vector<std::string> delimited_data;

    remove(line.begin(), line.end(), ' ');  // Remove Whitespace
    line.pop_back(); // Get rid of junk value which is added by remove??? look into that

    if (!isComment(line)) {
        delimited_data = splitLine(line, delimiter);
    }

    return delimited_data;
}

WorldLoader::WorldLoader(std::string file_name) {
    _file_name = file_name;

    if (_file_name == "") { std::cout << "Error: No file name supplied."; }
    else { _reader.open(file_name); }
}

WorldLoader::~WorldLoader() { if (_reader.is_open()) { _reader.close(); } }

std::vector<std::string> WorldLoader::splitLine(std::string string_data, char delimiter) {
    std::vector<std::string> split_strings;
    int start_idx = 0, end_idx = 0;

    for (int i = 0; i <= string_data.size(); i++) {
        if ((char)string_data[i] == delimiter) {
            std::string delimited_data;
            end_idx = i;
            delimited_data.append(string_data, start_idx, end_idx -
```

```
                start_idx);
46              split_strings.push_back(delimited_data);
47
48              start_idx = end_idx + 1;
49          }
50      }
51
52      return split_strings;
53  }
54
55  std::vector<std::string> WorldLoader::getLinesByDelimiter(char splitter) {
56      char delimiter = splitter;
57      std::string line;
58      std::vector<std::string> formatted_strings;
59
60      if (!_reader.is_open()) { _reader.open(_file_name); }
61
62      while (std::getline(_reader, line)) {
63          for (auto it : processLineInput(line, delimiter)) {
64              formatted_strings.push_back(it);
65          }
66      }
67
68      return formatted_strings;
69  }
70
71  std::vector<std::string> WorldLoader::getLinesByDelimiter(char splitter,
      std::string string_data) {
72      char delimiter = splitter;
73      std::string line = string_data;
74      std::vector<std::string> formatted_strings;
75
76      if (!_reader.is_open()) { _reader.open(_file_name); }
77
78      while (std::getline(_reader, line)) {
79          for (auto it : processLineInput(line, delimiter)) {
80              formatted_strings.push_back(it);
81          }
82      }
83
84      return formatted_strings;
85  }
```

```
1  #pragma once
2  #include <iostream>
3  #include <fstream>
4  #include <vector>
5
6  #include "WorldLoader.h"
7  #include "Location.h"
8
9
10 /*  The world object will also have to act as a pseudo-interface between
     the save data
11 *   and all of the world objects. It wont have the ability to read directly
      from files
12 *   but will have the ability to take a vector of semi-formatted data from
     the FileReader,
13 *   process this more and assign its' elements to the neccessary location
     objects
14 *
15 *   The rules for the text file are as follows:
16 *       Use ';' to seperate locations
17 *       Use ':' to seperate location data
18 *       Everything after the description is an exit name
19 *       E.g. Name1:Description1:ExitAName:ExitBName:ExitCName;
20 *            Name2:Description2:ExitAName:ExitDName:ExitZName;
21 *
22 *       Use _ as replacement for spaces. I will eventually write a function
      which converts
23 *       between the two but for right now all white space is being removed
     so descriptions
24 *       will look funny but meh.
                  */
25
26 class World {
27 private:
28     bool _is_running = true;
29     std::string _save_name = "test.txt";
30     WorldLoader* _reader = new WorldLoader(_save_name);
31
32     std::vector<Location*> _locations;
33     std::vector<std::string> _valid_directions = {
34         "north", "n", "up", "forward",
35         "east", "e", "right",
36         "south", "s", "down", "back", "backwards",
37         "west", "w", "left"
38     };
39
40     Location* _current_location = nullptr;
41
42     // Any new locations are instantiated here
```

```
43      std::vector<Location*> constructLocations();
44      Location* getLocationByName(std::string location_name);
45
46      std::string processDirectionInput(std::string dir);
47      bool checkDirectionsValid(std::string dir);
48
49  public:
50      World();
51      ~World();
52      void update();
53      void render();
54
55      bool checkIsRunning();
56      void addLocation(Location* new_location);
57      void showCurrentLocation();
58      void showLocations();
59      void loadLocationData();
60
61  };
62
63
```

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>

#include "World.h"

std::vector<Location*> World::constructLocations() {
    std::vector<Location*> locations;

    //Allocate Memory for rooms
    Location* lounge_room = new Location();
    Location* courtyard = new Location();
    Location* bedroom = new Location();
    Location* backyard = new Location();

    //Push those pointers to _locations
    locations.push_back(lounge_room);
    locations.push_back(courtyard);
    locations.push_back(bedroom);
    locations.push_back(backyard);

    return locations;
}

Location* World::getLocationByName(std::string location_name) {
    for (auto location : _locations) {
        if (location->getName() == location_name) { return location; }
    }
}

World::World() {
    _locations = constructLocations();

    if (_current_location != nullptr) {
        _locations.push_back(_current_location);
    }
    else { _current_location = _locations[0]; }
}

World::~World() {
    delete _reader;
}

bool World::checkIsRunning() { return _is_running; }

void World::addLocation(Location* new_location)
    { _locations.push_back(new_location); }

```

```
50 void World::showCurrentLocation() { _current_location->showDetails(); }
51
52 void World::showLocations() {
53     for (auto location : _locations) {
54         location->showDetails();
55     }
56 }
57
58 /*  Here is a funny function. This will load everything except exit data
     into a location.
59 *   The exit data will be done on a second pass as the structure requires
     the location to
60 *   own a name in to search it in the vector.   */
61 void World::loadLocationData() {
62     //Delimiters
63     char first_pass_delim = ';';
64     char second_pass_delimiter = ':';
65     char third_pass_delimiter = ',';
66
67     std::vector<std::string> unformatted_room_data = _reader-
       >getLinesByDelimiter(first_pass_delim);
68     std::vector<std::vector<std::string>> formatted_room_data;
69
70     for (auto room_data_set : unformatted_room_data) {
71         room_data_set += second_pass_delimiter; // Add the delimiter to
           the end of the line
72         std::vector<std::string> formatted_line = _reader->splitLine
           (room_data_set, second_pass_delimiter);
73         formatted_room_data.push_back(formatted_line);
74     }
75
76     for (int room_idx = 0; room_idx < formatted_room_data.size(); room_idx
       ++) {
77         _locations[room_idx]->setName(formatted_room_data[room_idx][0]);
78         _locations[room_idx]->setDescription(formatted_room_data[room_idx]
           [1]);
79     }
80
81     // It's assumed that the locations in the load file are in the same
       order as in the
82     // _locations vector.
83     for (int room_idx = 0; room_idx < formatted_room_data.size(); room_idx
       ++) {
84         for (int exit_idx = 2; exit_idx < formatted_room_data
           [room_idx].size(); exit_idx++) {
85             std::vector<std::string> split_direction_location =
86                 _reader->splitLine(formatted_room_data[room_idx][exit_idx]
                   +=third_pass_delimiter, third_pass_delimiter);
87             _locations[room_idx]->addExit(split_direction_location[0],
```

```
                getLocationByName(split_direction_location[1]));
88          }
89      }
90  }
91
92  /*  Needs to be able to take input from the player, translate that into a direction and
93  *   change the players current location based off this input.
94  *
95  *
96  */
97  std::string World::processDirectionInput(std::string dir) {
98      std::string direction = dir;
99      std::transform(direction.begin(), direction.end(), direction.begin(),
100         [](unsigned char c) { return std::tolower(c); });   // To lower
101
102     if (direction == "north" || direction == "n" ||
103         direction == "forward" || direction == "up")
104     {
105         return "north";
106     }
107
108     else if (direction == "east" || direction == "e" || direction == "right")
109     {
110         return "east";
111     }
112
113     else if (direction == "south" || direction == "s" ||
114         direction == "back" || direction == "backwards" || direction == "down")
115     {
116         return "south";
117     }
118
119     else if (direction == "west" || direction == "w" || direction == "left") {
120         return "west";
121     }
122
123     else { return "error"; }
124 }
125
126 bool World::checkDirectionsValid(std::string dir) {
127     if (std::find(_valid_directions.begin(), _valid_directions.end(), dir) != _valid_directions.end())
128     {
129         std::string direction = processDirectionInput(dir);
130         std::unordered_map<std::string, Location*> current_exits =
```

```
            _current_location->getExits();
131         std::unordered_map<std::string, Location*>::const_iterator exit =
              current_exits.find(direction);
132
133         return exit != current_exits.end();
134     }
135
136     return false;
137 }
138
139 void World::update() {
140     std::string action;
141     std::string direction;
142
143     std::cin >> action;
144     std::cin >> direction;
145     std::cout << std::endl;
146
147     if (action == "quit") { _is_running = false;    }
148     else if (action == "go") {
149         if (checkDirectionsValid(direction)) {
150             _current_location = _current_location->getExits()
                [processDirectionInput(direction)];
151         }
152     }
153 }
154
155 void World::render() {
156     std::cout << std::endl;
157     std::cout <<
        "------------------------------------------------------------" <<
        std::endl;
158     std::cout << "Room: " << _current_location->getName() << std::endl;
159     std::cout << _current_location->getDescription() << std::endl;
160     std::cout << std::endl;
161     std::cout << "Where do you want to go?" << std::endl;
162     std::cout << ">> ";
163 }
164
165
```