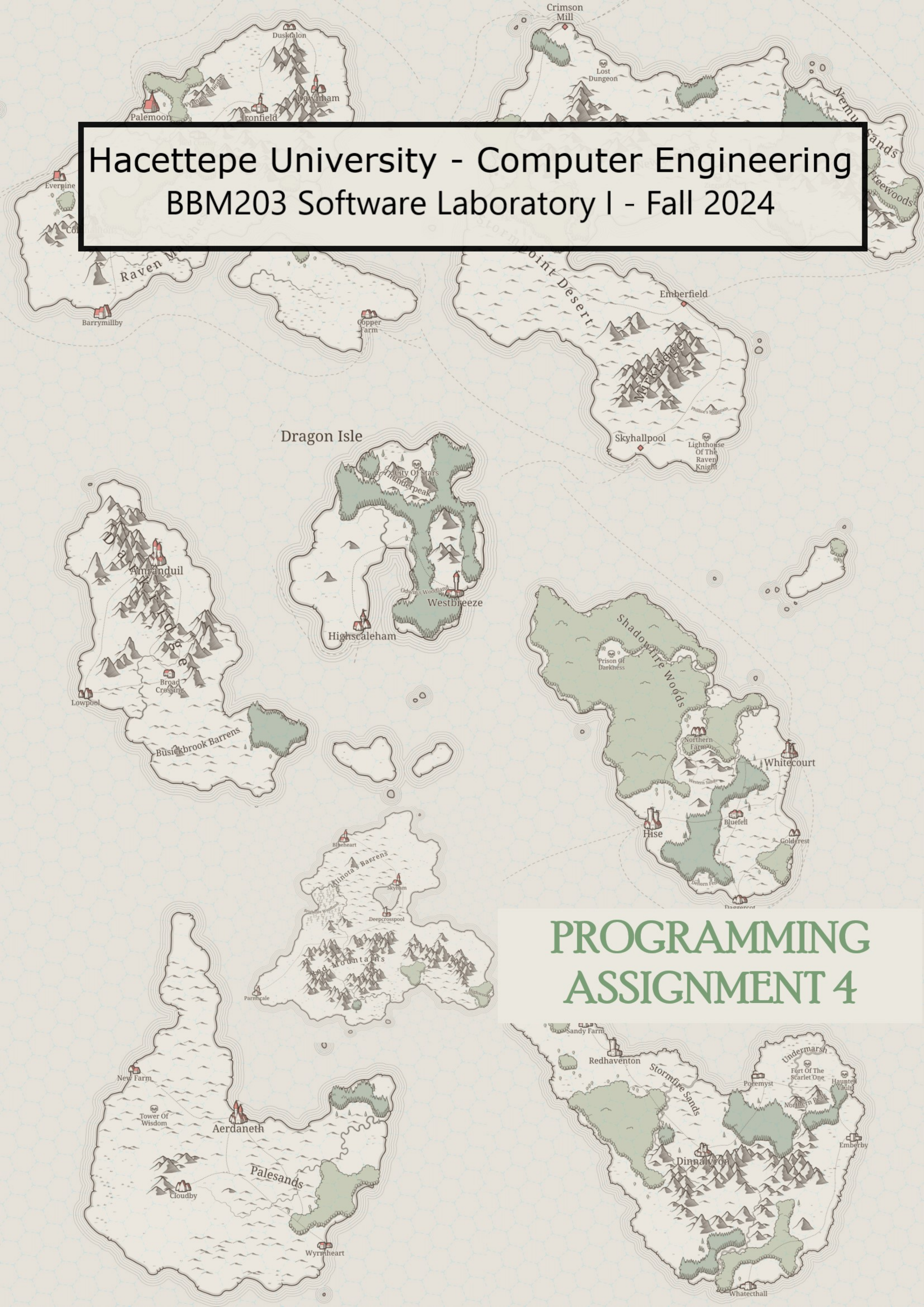

Hacettepe University - Computer Engineering

BBM203 Software Laboratory I - Fall 2024

PROGRAMMING ASSIGNMENT 4

PROGRAMMING ASSIGNMENT 4

Topics: Binary Trees, Self-Balancing Trees, Dynamic Memory Allocation, Operator Overloading, File I/O

Course Instructors: Assoc. Prof. Dr. Adnan ÖZSOY, Asst. Prof. Dr. Engin DEMİR, Assoc. Prof. Dr. Hacer YALIM KELEŞ
TAs: **M. Aslı TAŞGETİREN***, S. Meryem TAŞYÜREK, Asst. Prof. Dr. Selma DİLEK
Programming Language: C++11 - **You MUST USE this starter code**
Due Date: **Sunday, 15/12/2024 (23:59:59)** over **https://submit.cs.hacettepe.edu.tr**

Introduction

The M.A.T Game Studios (a subsidiary of H.U. Game Studios) has exciting news – they have decided to add a new player class to their popular RPG: H.U.SLAND! However, before the new version of the game can be developed, company executives wants to see a proof of concept prototype for this new idea. This task was given to you reliable interns from Hacettepe University by your supervisors.

With the new version, H.U.SLAND game now will have a special class: **Realm Shapers**. Realm Shapers are a special class of adventurers capable of crafting and reshaping the Isles of the game world. Players that were able to reach this class will have the ability to change the game world according to their whims and challenge other players for even higher ranks. However, everything comes with a price and Realm Shaper is a hard to earn but easy to lose title, making strategic gameplay essential.

There are already rumors floating around about this new game dynamic and die-hard fans are waiting for it with great enthusiasm. Interns' supervisors have given them only two weeks, it is a race to see who will be first to completely implement the new features.

The workings of the new game is confidential, however trusty interns from the Hacettepe Univesity have been let in to the secret. What is the secret behind this new game mechanism? Game world will now be working with a self-balancing tree. Allowing both ease in access to Isles in the backend code and also exciting changes to game world UI that would be unpredictable to those in unknown.

1 The New Character Class: Realm Shapers

1.1 First of Many: Shaper Tree

Shaper Tree is a custom binary tree that is implemented using an array (or a vector). In this tree newly inserted nodes are always added level-by-level, from left to right, ensuring a complete binary tree structure.

Deletion is done by deleting the node and shifting rest of the nodes while keeping their order to fill the gap.

The ranking system is derived from **level-order traversal** of the tree.

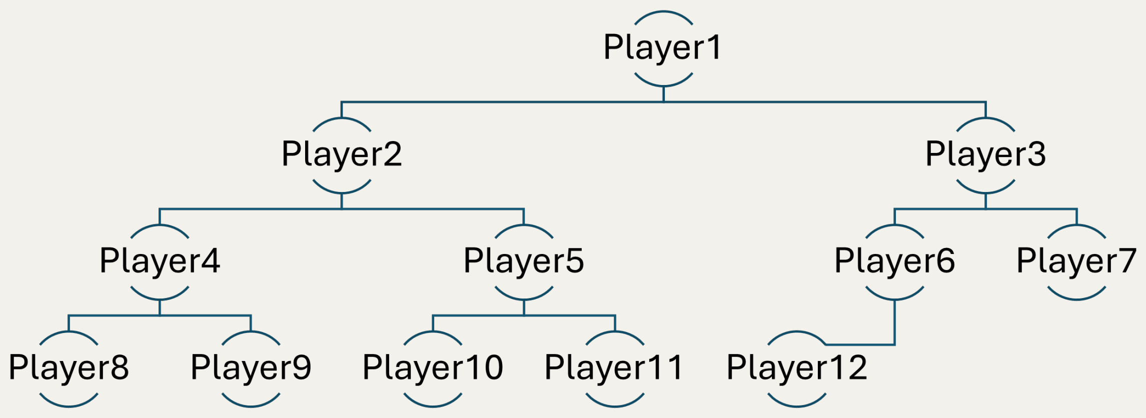

Figure 1: Tree after adding Player1 to 12, **in that order**

1.2 Not all Realm Shapers are same: Grade

Realm Shapers have a hierarchy between themselves. Grade of Realm Shapers, which is determined by their position in the Shaper Tree, influences parts of the world they can visit while their title of Realm Shapers is active. Being able to visit as many Isles as they can is necessary for Shapers, as this is the only way they can collect items necessary for crafting new realms.

Grade of Realm Shapers are determined by their depth in the Shaper Tree. Players closer to the root have higher Grades and have access to more Isles, allowing them to collect valuable items required for crafting new realms. More about access in Section 1.5.

1.3 Opportunities to Level Up: Duels

Players can improve their Realm Shaper ranking by challenging their connected superior player (parent node) to a duel. If a duel is won, players swap places with each other.

Duels are also good opportunities to win points in special attribute **Honour**. Each Realm Shapers gets 1000 Honour Points when they earn their title. Each duel won adds more Honour Points to the player. However, players should be careful while they are inviting others to a duel since losing one losts player Honour Points. A player with zero Honour Points is eliminated from the Shaper Tree and loses their Realm Shaper title.

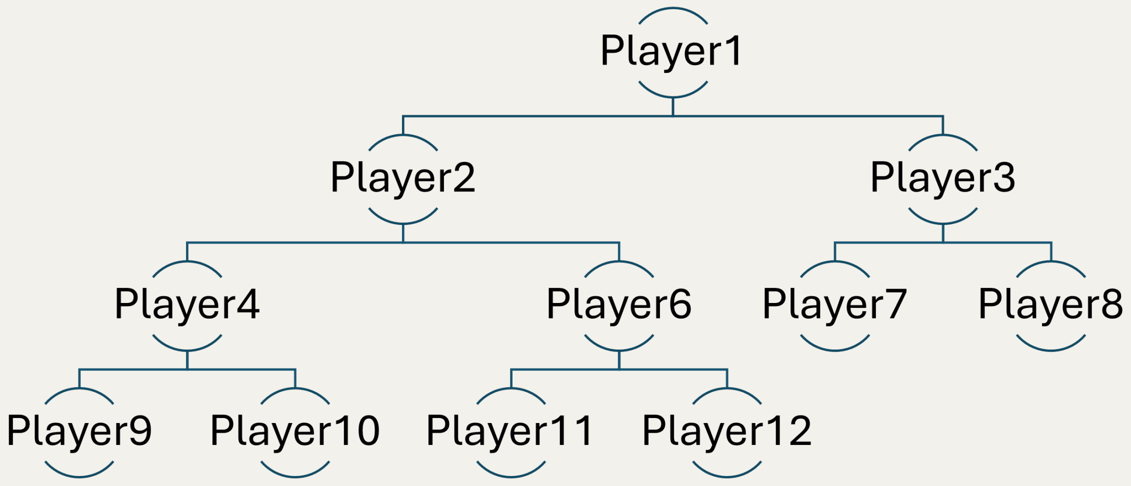

Figure 2: Tree after Player5 was deleted (Asume he lost too many duels to Player2)

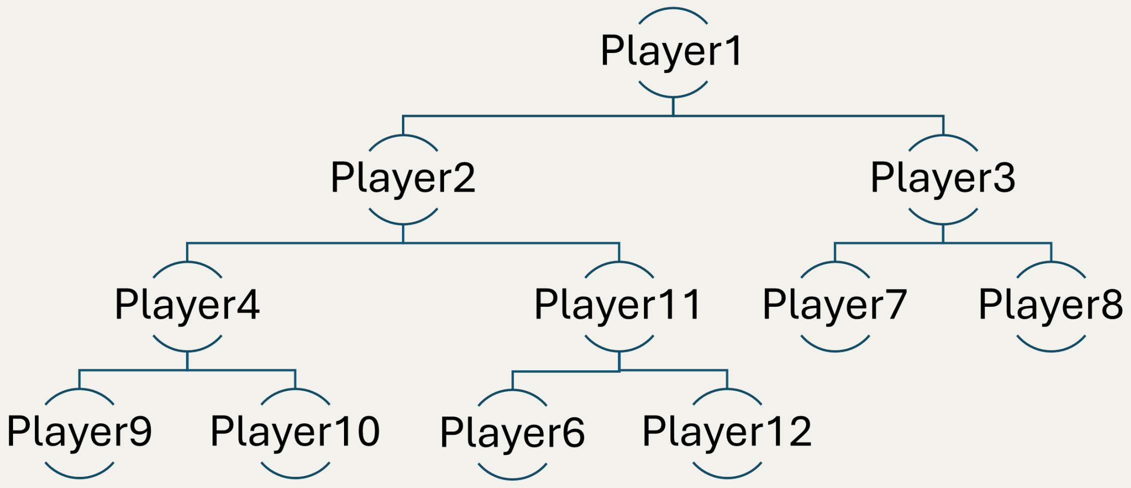

Figure 3: Tree after Player11 won a duel.

1.4 Everything Has a Price: Items

While all Realm Shapers have the technical ability to craft new realms, they cannot do this without having the necessary Shaping Energy Points. To get these necessary energy points, Shapers first need to collect necessary materials. All collected items are processed to become Shaping Energy for Shapers to use. Items that provide the energy are scattered across the world map and are only visible to players with active Realm Shaper titles. More about items at Section 2.2.

1.5 Powerful but Not OP: Access

What players without active Realm Shaper title do is not topic of this assignment. When a Realm Shaper does not have necessary access to an Isle they will be counted as not being there, even though they may have gone there anyway after de-activating their title. As said in Section 1.2. access is determined by Grade, as such depth of players in the Shaper Tree.

Three rules for access are:

1. The player at the root can access everything.
2. Players at the bottom can only access outer Isles in the world.
3. Intermediate players' access are proportional to their position in the hierarchy. Player's access level (minMapDepthAccess) in the World Tree is calculated as:

$$\text{minMapDepthAccess} = \text{totalMapDepth} \times \frac{\text{playerDepth}}{\text{totalHeight}}$$

- Where:
  - playerDepth: Distance of the player node from the root (number of edges to the root).
  - totalHeight: Total height of the Shaper Tree.
  - totalMapDepth: Maximum depth of the World Tree (map).
  - minMapDepthAccess: Minimum depth in the World Tree that a player can access, e.g. if it is 0 player can access nodes with depth bigger than and equal to 0 i.e. everywhere.

1.6 What Goes Around Comes Around: FileIO

Players that were able to become Realm Shapers are kept in a text file. Input file for this is initial_realm_shapers.txt and output file will be current_realm_shapers.txt. Both of these files should include players listed according to their rankings. More information about this at Section 4.

1.7 Ready To Use: Terminal Output

Players are written to terminal in this format. Starter code includes this code, you will not implement it.

`Shaper Tree in Terminal:`

```
Name: Player1
   ---- Name: Player2
         ---- Name: Player4
               ---- Name: Player8
               ---- Name: Player9
         ---- Name: Player5
               ---- Name: Player10
               ---- Name: Player11
   ---- Name: Player3
         ---- Name: Player6
         ---- Name: Player7
```

2 New World Order: AVL

Taking advantage of the new character class release, developers have also decided to roll out a new game mechanism for their World that has both more consistent and efficient access to the Isles in the map. For this they have decided to use a self-balancing tree, particularly AVL tree. With this they were also able to add new game mechanisms. Such as mysterious and unexpected world changes to ordinary players and a chance to change the game world to their advantage for Realm Shapers.

In the backend, AVL is implemented using pointers. Each node in the tree holds an Isle, with attributes such as its name and any associated item. The pointer-based implementation provides flexibility in managing dynamic memory allocation, enabling efficient insertion, deletion, and traversal operations. AVL ensures that the tree maintains its balance through rotations, preserving an optimal height for fast access to Isles, even as the world dynamically changes.

2.1 Initializing the World: Map

World Tree (Map) is initialized from initial_world.txt input file. Nodes (Isles) are inserted one by one while maintaining AVL balance.

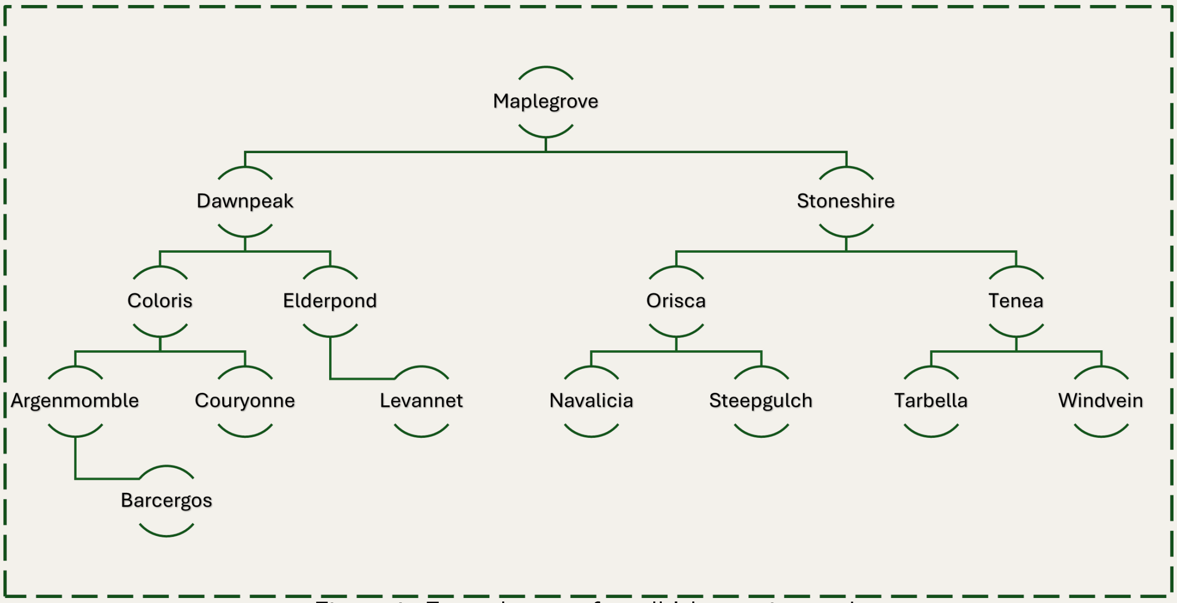

Figure 4: Example tree after all Isles are inserted

2.2 Traverse and Enrich: Items

Once World Tree (Map) is initialized it is populated with special items that Realm Shapers will use in crafting. Items are hard coded in Items enum, each with unique values. Each Island can only hold one type of Item at a time. Enriching will be done once after initialization for Goldium and Einsteinium and then every 3rd time world re-balancing is needed after that for all items.

2.2.1 Item Mechanics

Post-Order Traversal

Every third Isle (according to Post-Order Traversal) is populated with ITEM::GOLDIUM.

Pre-Order Traversal

Every fifth Isle (according to Pre-Order Traversal) is populated with ITEM::EINSTEINIUM.

BFS Item Drop

Every 3rd re-balancing of the AVL tree causes the rare and most valuable `ITEM::AMAZONITE` to drop to first Isle that has no existing item. This helps more valuable items to stay in the lower depths.

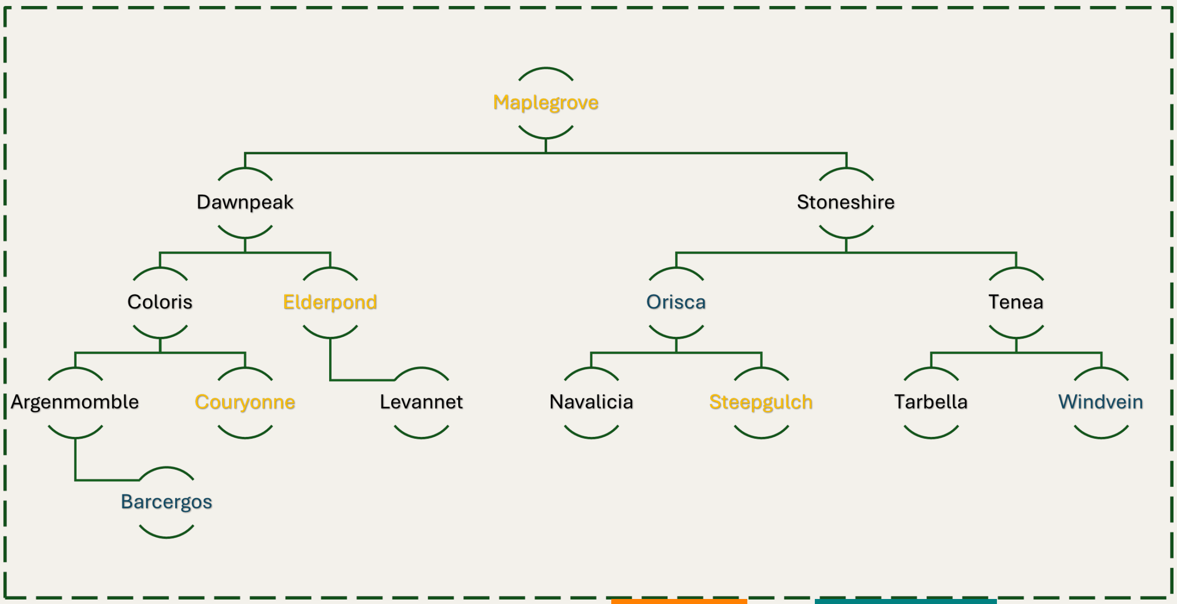

Figure 5: Tree after enriching with first GOLDIUM than EINSTEINIUM.

2.3 More is Less: Overcrowding

Because Realm Shapers are so powerful, having too many of them gather at one place causes catastrophic events. Every Isle has a maximum capacity of Shapers it can bare, if more than this number of Shapers access the Isle, Isle unfortunately collapses and self-destructs. Destroyed Isles should be removed from the Map, and if necessary, world should re-balance itself by following AVL rules.

2.4 Saving to the File: Isles and Map

World Tree (Map) is saved two different output files: current_isles.txt and current_map.txt. More about this at Section 4.

3 Game Mechanics: 5 to 1

3.1 Step by Step: Gameplay

The game simulation is driven by the following logs:

1. **Access Logs (access.log)**: Records of players attempting to visit or craft Isles.
2. **Duel Logs (duels.log)**: Records of duels initiated by players and their outcomes.

Gameplay proceeds as follows:

- For each record in `access.log`:
  - If the Isle exists and the player has access, they visit it and collect the available item.
  - If the Isle does not exist but the player has enough Energy Points, a new Isle is crafted and added to the World Tree.
- For each record in `duels.log`:
  - Duel happens with specified result and the Shaper Tree is updated accordingly.
- One duel happens for every five access record. If there are still duel records after all the access records has been processed, those duels should be done one after one-other.

After every action, updated Shaper Tree and World Tree are displayed in the terminal.

3.2 Fun Extra: Single Player Demo

This part of the assignment does not require you to implement anything extra

As a fun addition to the assignment, you will have **single-player demo mode** that will allow you to interact with the game world and rankings directly from the terminal. In this mode you will be able to add your own player to the game and win duels by answering questions. Questions are varied in topic, such as general culture ("What year Hacettepe University was established in?"), trees ("What is index of left child in array based binary tree?") and humor ("Why do python programmers wear glasses?"). To allow for this, you will be given Questions.h, Questions.cpp, and Questions I/O files. All functions in these files are fully implemented for you to run.

Features of the Single-Player Demo:

1. **Interactive Menu**: A menu-driven system that lets the player perform actions such as dueling, exploring, and crafting new Isles in real-time.
2. **Dynamic Updates**: After each action, the program displays the updated Shaper Tree and World Tree in the terminal using an ASCII-based tree visualization.
3. **Dueling**: The player can select their character and challenge others. Outcome of the duels are determined with questions instead of being pre-determined.
4. **Exploration and Crafting**: Players can attempt to explore existing Isles. If they don't have access, they can craft a new Isle using their points.
5. **Progress Tracking**: Honour Points and collected items are displayed after each turn.

4 Files: Input/Output

Below you can find details and examples from files that is given to you or is required from you.

4.1 Initial Players: initial_realm_shapers.txt

Players that will beta test Realm Shaper class have been given to you in this file. Players are ranked in the file and should be added to Shaper Tree in same order as the file. File contains Player name and their recorded honour points.

Format:

```
#PlayerName[tab]HonourPoints
Player1 1800
Player2 1400
Player3 1450
Player4 1450
```

4.2 Initial Isles: initial_world.txt

Isles that exist in the world have been given to you in this file. Isle names in the file are not ordered, but should be inserted to the map one by one following the file. File only contains Isle names.

Format:

```
#IsleName
Maplegrove
Dawnpeak
Stoneshire
```

4.3 Access Records: access.log

This file includes logs from a game that was played before, you will use these records to recreate that gameplay and test your implementation. The file includes player name and name of the isle player tried to assess. If isle already exist and player has access rights, they should visit the isle and collect any item there is. If isle does not exist and player has enough Shaping Energy, a new isle with the specified name should be crafted and inserted to the map.

Format:

```
#PlayerName[tab]IsleName
#(if place exist player visits, if not new place is added (if player has access))
Player1 Couryonne
Player2 Coloris
Player3 Orisca
```

4.4 Duel Records: duels.log

This file includes logs from a game that was played before, you will use these records to recreate that gameplay and test your implementation. The file includes name of the player that have initiated the duel and result of the duel.

Format:

```
#ChallangerPlayerName result(1: won 0: lost)
Player10    1
Player4    0
```

4.5 Current Isles: current_isles.txt

This is an output file you should provide, that should include all the isles that still exists in the map, in-order. They will be ordered based on their names.

Format:

```
Argenmomble
Barcergos
Coloris
```

4.6 Current Map: current_map.txt

This is an output file you should provide, that should include structure of the current map. Nodes of the map will written out level by level. Nodes in same level will have a [space] between.

Format:

```
Maplegrove
Dawnpeak Stoneshire
Coloris Elderpond Orisca Tenea
Argenmomble Couryonne NULL Levannet Navalicia Steepgulch Tarbella Windvein
NULL Barcergos NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL
```

4.7 Current Players: current_realm_shapers.txt

This output file should include list of all the Realm Shapers that still exist in the game. In the file, name of the players should be written based on their rankings.

Format (Player4 won a duel against its parent (Player2)):

```
Player1
Player4
Player3
Player2
```

4.8 Current Players: current_shaper_tree.txt

This output file you should provide structure of the current Shaper Tree. In the file, name of the players should be written as a list based on pre-order traversal.

Format (Players 1-11 added one after the other):

```
Player1
Player2
Player4
Player8
```

5 Assignment Implementation Tasks and Requirements: Code

In this section, **some** of the classes and functions you are required to implement are outlined. Do not alter the names or signatures of functions provided in the starter code template files. Likewise, refrain from renaming or changing the types of the specified member variables. Other than that, you will need to introduce additional functions and can add variables variables as needed.

5.1 `RealmShaper Class`

This class represents a Realm Shaper, a special class of players who can craft new Isles.

- **Attributes**:
  - `string name`: The name of the player.
  - `int honourPoints`: The current Honour Points of the player, determining their eligibility for crafting and dueling.
  - `int collectedEnergyPoints`: Points that they were able to get from Items.
- **Constructors**:
  - `RealmShaper(const string& playerName, int initialHonour)` Initializes a Realm Shaper with a given name and starting Honour Points.
  - `RealmShaper()` Default constructor initializing a blank Realm Shaper with default attributes.
- **Copy Constructor**: `RealmShaper(const RealmShaper& other)` Creates a copy of another Realm Shaper object.
- **Operator Overloading**:
  - `'=='` operator compares two players by their name.
  - `'<<'` operator helps to output Realm Shapers by their name.
- **Functions**:
  - `void gainHonour(int points)`
    Increases the Honour Points of the player by the specified amount.
  - `void loseHonour(int points)`
    Decreases the Honour Points of the player by the specified amount. If Honour Points reach zero, the player is removed from the game.
  - `void hasEnoughEnergy()`
    Checks if player has enough energy to craft an new Isle.
  - `void collectItem(Item item)`
    Collects item and adds its value to player's energy points.
  - `static vector<RealmShaper *> readFromFile(const std::string &filename)`
    Reads Player information from input file.

5.2 `Isle Class`

This class represents an Isle, a node in the AVL tree that forms the dynamic game world.

- **Attributes**:
  - `string name`
    The name of the Isle.
  - `ITEM item`
    The item located on the Isle (e.g., GOLDIUM, EINSTEINIUM, AMAZONITE).
  - `int capacity`
    Max number of Realm Shapers that can visit the Isle at a time.
  - `int playerCount`
    Current number of Realm Shapers at the Isle.
- **Constructors**:
  - `Isle(const string& isleName)`

    Creates an Isle with the given name and no associated item.
  - `Isle(const string& isleName, ITEM itemType)`

    Creates an Isle with the given name and assigns it a specific item.
- **Copy Constructor**:

  `Isle(const Isle& other)`

  Creates a copy of another Isle object.
- **Operator Overloading**:
  - `'=='` operator compares two Isles by their names.
  - `'<'` and `'>'` operators compares two Isles for ordering in the AVL tree (alphabetically by name).
  - `'<<'` operator helps to output Isles by their name.
- **Functions**:
  - `void assignItem(ITEM newItem)`

    Assigns a new item to the Isle.
  - `static vector<Isle*> readFromFile(const std::string &filename)`

    Reads and parses Isles from input file.
  - `bool increasePlayerCount()`

    Increases player count if capacity is not full. Returns isFull value.

5.3 `Map Class`

This class represents the AVL Tree structure used to manage the game world map.

5.3.1 MapNode Struct

This struct represents a node in the map structure, containing an `Isle` object and pointers to child nodes.

- **Attributes**:
  - `Isle* isle`: The data stored in the node.
  - `MapNode* left`: Pointer to the left child.
  - `MapNode* right`: Pointer to the right child.
  - int height: Height of the node.
- **Constructors**:

  `MapNode(Isle* isle)` Initializes a `MapNode` with the given `Isle` object.
- **Destructor**:

  `~MapNode()` Cleans up dynamically allocated memory for the `Isle` object if ownership is managed here.

5.3.2 Map Class

- **Attributes**:
  - `Isle* root`: Pointer to the root node of the AVL tree.
- **Constructors**: `Map()` Default constructor that initializes an empty AVL tree.
- **Functions**:
  - `void insert(Isle *isle)`
    Inserts a new Isle into the tree. Balances the tree if needed.
  - `void remove(Isle *isle)`
    Removes an Isle from the tree, rebalancing it afterward.
  - `Isle* findIsle(const string& isleName)`
    Searches for an Isle by its name and returns a pointer to the Isle if found.
  - `MapNode* findNode(Isle isle)`
    Searches for a node by the Isle it holds and returns a pointer to the node if found.
  - `int getNodeDepth(Isle *isle)`
    Determines the depth level of a Isle within the tree.
  - `int getDepth()`
    Determines the depth level of the full tree.

  - `void inOrderTraversal()`
    Travels the tree in alphabetical order.
  - `void preOrderTraversal()`
    Travels the tree in pre-order.
  - `void postOrderTraversal()`
    Travels the tree in post-order.
  - `void populateWithItems()`
    Enriches the world with items by the rules defined.
  - `void writeToFile(const std::string &filename)`
    Write the tree structure to a file as specified in Section 4.6.
  - `void writeIslesToFile(const std::string &filename)`
    Write the Isles to a file as specified in Section 4.5.
- **Destructor**: `~Map()` Cleans up all nodes in the AVL tree.

5.4 `ShaperTree Class`

This class represents the tree used to rank the players.

Attributes: `vector<RealmShaper*> players`: A dynamic array of pointers to Realm Shaper objects.

Constructors: `ShaperTree()` Default constructor that initializes an empty player ranking tree.

Functions:

- `void insert(RealmShaper* newPlayer)`
  Adds a new player to the tree in the next available position.
- `void remove(RealmShaper* player)`
  Removes the player at the given rank and maintains the completeness of the tree.
- `void duel(RealmShaper *challenger, bool result)`
  Simulates a duel between two players and swaps their positions if the challenger wins.
- `int getDepth(RealmShaper* player)`
  Determines the depth level of a player within the tree.
- `int getDepth()`
  Determines the depth level of the full tree.
- `RealmShaper* findPlayer(const string& playerName)`
  Searches for a player by its name and returns a pointer to the player if found.
- `int findPosition(RealmShaper* player)`
  Searches for a player in the tree array, returns the position index if found.
- `void inOrderTraversal()`
  Travels the tree in-order.

- `void preOrderTraversal()`
  Travels the tree in pre-order.
- `void postOrderTraversal()`
  Travels the tree in post-order.
- `void levelOrderTraversal()`
  Travels the tree in grade by grade.
- `void writeToFile(const std::string &filename)`
  Writes the shaper tree players to a file, see Section 4.8.
- `void writePlayersToFile(const std::string &filename)`
  Write the player rankings as a list to a file, see Section 4.7.

Destructor: `~ShaperTree()` Cleans up dynamically allocated Realm Shaper objects and clears the vector.

5.5 `GameWorld Class`

This class represents the overarching game world, managing both the Shaper Tree and the World Tree, and facilitating interactions between them.

Attributes:

- `Map worldTree`: The AVL tree representing the dynamic game world map.
- `ShaperTree shaperTree`: The tree representing the player ranking system.

Constructors: `GameWorld()` Initializes an empty game world with default trees.

Functions:

- `void initializeGame(vector<Isle*> isles, vector<RealmShaper*> players)`
  Initializes the game world by populating the Shaper Tree and World Tree with the provided Isles and players.
- `void processGameEvents(const string &accessLogs, const string &duelLogs)`
  Processes game by applying actions from access and duel logs, updating both trees dynamically.
- `void displayGameState()`
  Displays the current state of the World Tree and Shaper Tree in the terminal using ASCII-based visualization.
- `void saveGameState(args*);`
  Saves the current state of the World Tree and Shaper Tree to output files.
- `bool hasAccess(RealmShaper* player, Isle* isle)`
  Determines if the given player has access to the specified Isle based on their rank.
- `void exploreArea(RealmShaper* playerID, Isle* isle)`
  Allows a player to explore a specified Isle. If the Isle exists, the player interacts with it. If it does not, a new Isle may be crafted depending on the player's points.

Must-Use Starter Code

You MUST use **this starter code**. All classes should be placed directly inside your **zip** archive.

Build and Run Configuration

Here is an example of how your code will be compiled (note that instead of `main.cpp` we will use our test files):

```
$ g++ -std=c++11  *.cpp, *.h -o HUSLAND
```

Or, you can use the provided `Makefile` or `CMakeLists.txt` within the sample input to compile your code:

```
$ make
```

or

```
$ mkdir HUSLAND_build
$ cmake -S . -B HUSLAND_build/
$ make -C HUSLAND_build/
```

After compilation, you can run the program as follows:

```
$ ./HUSLAND initial_world.txt initial_realm_shapers.txt access.log duels.log
↪  current_isles.txt current_map.txt current_realm_shapers.txt
↪  current_shaper_tree.txt
```

Grading Policy

- **No memory leaks and errors: 10%**
  - No memory leaks: 5%
  - No memory errors: 5%
- **Correct implementation of the Realm Shapers Tree: 25%**
  - Proper tree initialization from the input files 5%
  - Implementation of duels 5%
  - Implementation of removing Players 5%
  - Implementation of access calculation 5%
  - Implementation of traversals 5%
- **Correct implementation of the World Tree (Map): 45%**
  - Proper tree initialization from the input files 5%
  - Implementation of Isle insertion from Player 5%
  - Implementation of Isle deletion after overcrowding 10%
  - Implementation of item distribution 10%
  - Implementation of Isle access 10%
  - Implementation of saving tree to file in-order 5%
- **Correct implementation of the game mechanics: 10%**
  - Proper game play initialization from the log files 5%
  - Implementation of game play 5%
- **Output tests: 10%**

Note that you need to get a NON-ZERO grade from the assignment in order to get the submission accepted. Submissions with grade 0 will be counted as NO SUBMISSION!

Important Notes

- Do not miss the deadline: **Sunday, 15/12/2024 (23:59:59)**.
- Save all your work until the assignment is graded.
- The assignment solution you submit must be your original, individual work. Duplicate or similar assignments are both going to be considered as cheating.
- You can ask your questions via Piazza (`https://piazza.com/hacettepe.edu.tr/fall2024/bbm203`), and you are supposed to be aware of everything discussed on Piazza.
- You must test your code via **Tur^{3}Bo Grader** `https://test-grader.cs.hacettepe.edu.tr/` (**does not count as submission!**).
- You must submit your work via `https://submit.cs.hacettepe.edu.tr/` with the file hierarchy given below:
  - **b<studentID>.zip**
    - GameWorld.cpp <FILE>
    - GameWorld.h <FILE>
    - Isle.cpp <FILE>
    - Isle.h <FILE>
    - Map.cpp <FILE>
    - Map.h <FILE>
    - RealmShaper.cpp <FILE>
    - RealmShaper.h <FILE>
    - RealmShapers.cpp <FILE>
    - RealmShapers.h <FILE>
- **You MUST use this starter code**. All classes should be placed directly in your **zip** archive.
- This file hierarchy must be zipped before submitted (not .rar, only .zip files are supported).
- Do not submit any object or executable files. Only header and C++ files are allowed.

Academic Integrity Policy

All work on assignments **must be done individually**. You are encouraged to discuss the given assignments with your classmates, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in pseudocode) **will not be tolerated**. In short, turning in someone else's work (including work available on the internet), in whole or in part, as your own will be considered as **a violation of academic integrity**. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.

The submissions will be subjected to a similarity check. Any submissions that fail the similarity check will not be graded and will be reported to the ethics committee as a case of academic integrity violation, which may result in the suspension of the involved students.