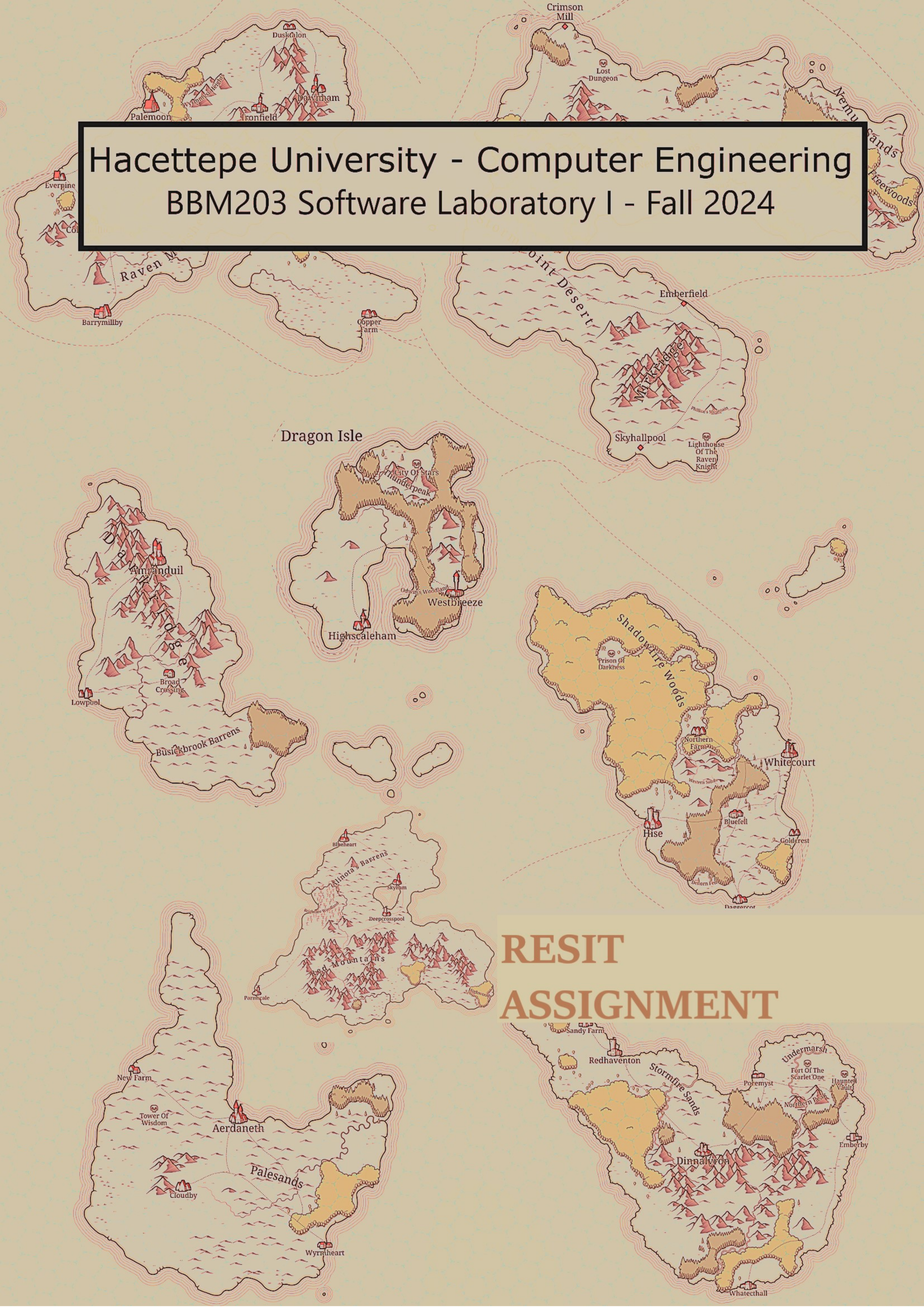

Hacettepe University - Computer Engineering

BBM203 Software Laboratory I - Fall 2024

RESIT ASSIGNMENT

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: **Saturday, 01/02/2025 (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 are now working on a revised version of their popular RPG: H.U.SLAND! Building on the foundation of the previous update, they've decided to refine the concept of the special player class, Realm Shapers, while introducing new challenges and mechanics to enhance gameplay. As reliable interns from Hacettepe University, your supervisors have once again tasked you with delivering a proof-of-concept prototype for this updated design.

In this new version, Realm Shapers continue to wield their unique power to reshape the Isles of the game world. However, the mechanics have been streamlined to focus on key aspects of their abilities: crafting new Isles, navigating the mysterious world map, and collecting artifacts imbued with magical energy. The Isles are now governed by a Left-Leaning Red-Black Tree (LLRBT), a refined self-balancing tree that ensures the game world is more stable while adding a layer of unpredictability to exploration. Additionally, the privilege to collect these artifacts once again depends on the player's rank in the Shaper hierarchy.

The new changes have sparked curiosity among die-hard fans, and anticipation for this refined mechanic is growing. However, with only six days to implement this updated version, the pressure is on for interns to deliver a polished prototype. Will you rise to the occasion and help redefine H.U.SLAND?

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.

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

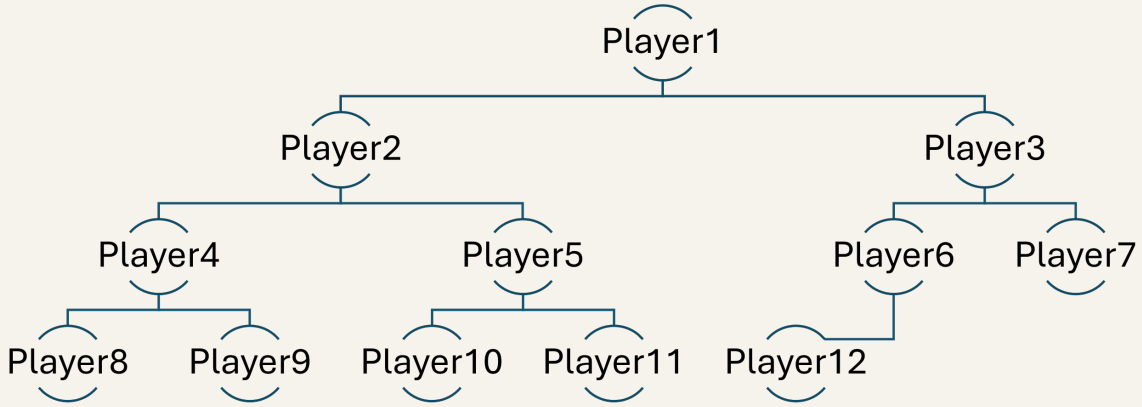

Figure 1: Tree after adding Player1 to 12, **in that order**

1.2 Not all Realm Shapers are same: Grade

Realm Shapers have a hierarchy between themselves. The grade of Realm Shapers, which is determined by their position in the Shaper Tree, influences parts of the world from which they can collect the items necessary to craft 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 ability to collect valuable items required for crafting new realms from more Isles. More about access in Section 1.4.

1.3 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.

1.4 Powerful but Not OP: Access

As said in Section 1.2. access is determined by Grade, as such depth of players in the Shaper Tree.

Rules for access are:

1. The Realm Shapers at the **lower half** of the Shaper Tree can only collect items from **black** nodes.
2. The Realm Shapers at the **upper half** of the Shaper Tree can collect items from both red and **black** nodes.

1.5 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. This file will include players listed according to their rankings. More information about this at Section 4.

1.6 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: LLRBT

With the refined design of the Realm Shapers, developers have rolled out a new game mechanism to make the World Map more dynamic while retaining efficiency and balance. To achieve this, they have replaced the existing self-balancing tree with a **Left-Leaning Red-Black Tree (LLRBT)**. This structure introduces new mechanics tied to the color of the Isles, enhancing gameplay and strategy for Realm Shapers.

In the backend, LLRBT is implemented using pointers. Each node in the tree represents an Isle, with attributes such as its name, color (red or black), and any associated items. The color of the nodes influences gameplay directly:

- Isles in **black nodes** are accessible to all players.
- Isles in **red nodes** are only open to item collection by top-ranked Realm Shapers.

The pointer-based implementation of LLRBT ensures efficient insertion and traversal operations, while also simplifying balancing through rotations and color adjustments. Unlike AVL trees, which rely on strict height balancing, LLRBT uses red and black properties to maintain a balanced structure, allowing the World Map to dynamically expand and shift without sacrificing access efficiency.

These new mechanics not only streamline the backend but also introduce strategic challenges for players, as their ability to interact with the Isles now depends on both their rank in the Shaper hierarchy and the properties of the Isles themselves. Realm Shapers can now look forward to reshaping the world with a mechanism that is both flexible and exciting.

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 LLRBT balance.

2.2 Traverse and Enrich: Items

Once the World Tree (Map) is initialized, it is populated with special items that the Realm Shapers will use in crafting. Items are hard coded in the 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 all the items i.e. Goldium, Einsteinium and Amazonite.

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.

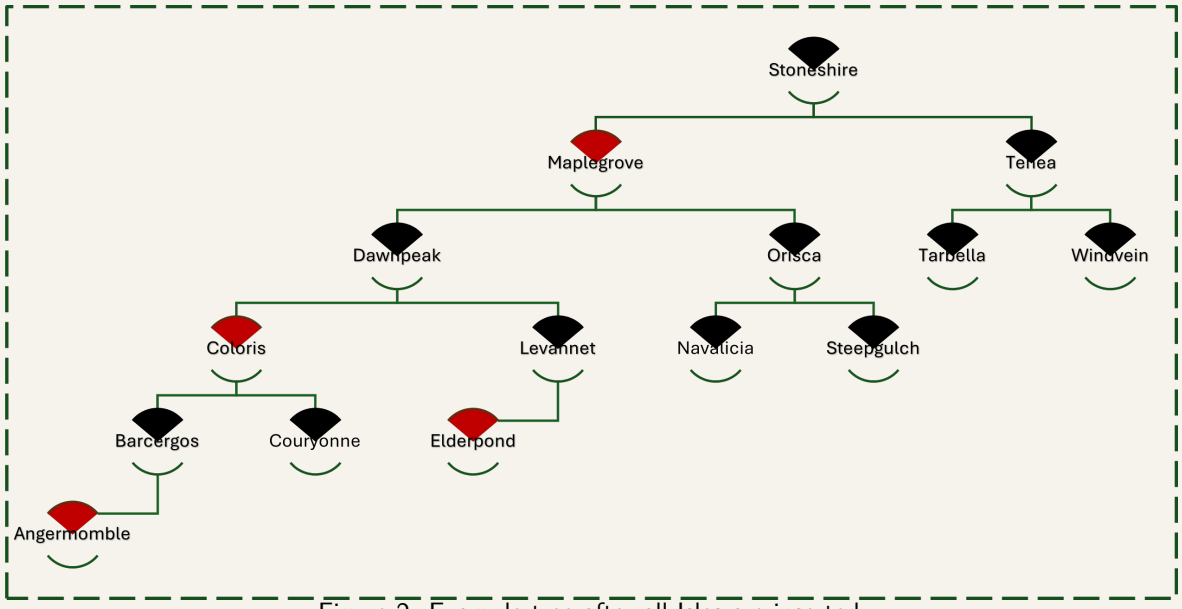

Figure 2: Example tree after all Isles are inserted

BFS Item Drop

`ITEM::AMAZONITE` is dropped to the first red Isle that has no existing item. This helps more valuable items to stay in the lower depths.

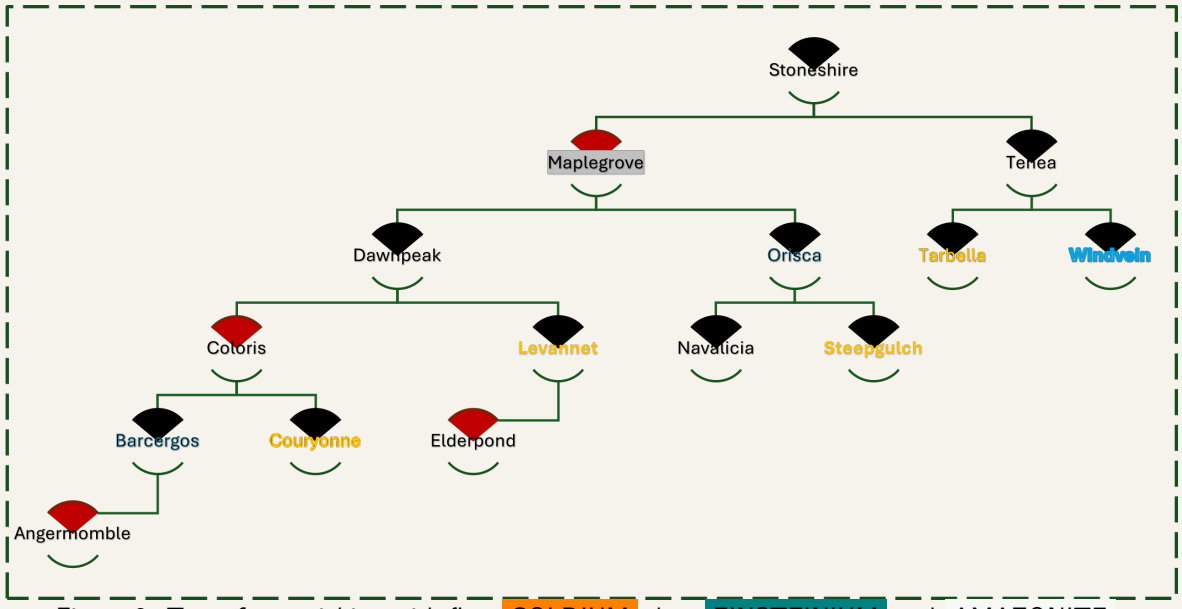

Figure 3: Tree after enriching with first GOLDIUM than EINSTEINIUM and AMAZONITE .

2.3 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

3.1 Step by Step: Gameplay

The game simulation is driven by the Access Log (access.log), Access Logs are records of players attempting to visit or craft Isles.

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 they are allowed to.
  - If the Isle does not exist but the player has enough Energy Points, a new Isle is crafted and added to the World Tree.

4 Files: Input/Output

Below you can find details and examples from files that are 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 only contains Player names.

`Format`:

```
#PlayerName
Player1
Player2
Player3
Player4
```

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 they are allowed to. 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 Current Isles: current_isles.txt

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

Format:

```
#IsleName[tab]Color
Argenmomble Red
Barcergos   Black
Coloris Black
```

4.5 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
```

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 all
```

or

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

After compilation, you can run the program as follows:

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

Grading Policy

- **No memory leaks and errors: 10%**
  - No memory leaks: 5%
  - No memory errors: 5%
- **Correct implementation of the Realm Shapers Tree: 10%**
  - Proper tree initialization from the input files 3%
  - Implementation of insertion of Players 4%
  - Implementation of traversals 3%
- **Correct implementation of the World Tree (Map): 60%**
  - Implementation of Isle insertion 25%
  - Implementation of item distribution (traversals) 25%
  - Implementation of item collection 10%
- **Correct implementation of the game mechanics: 10%**
  - Implementation of game play 10%
- **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: **Saturday, 01/02/2025 (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.