SENG1120 – Data Structures
Trimester 2, 2024

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

Assignment	*3*	**Due Date**	*Sunday 4 August 2024 @ 2359*
Purpose	*To measure the student's ability to implement and/or use complex data structures (e.g., hash tables and binary search trees) to solve an underlying problem in C++.*		

Introduction

The Mythos Creatures are legendary creatures that have existed since the dawn of time, each embodying the elemental forces of nature. These beings maintain balance and protect the world from chaos. Each Mythos Creature is a guardian of a specific domain, from fiery mountains and frozen tundras to enchanted forests and vast oceans. They possess unique powers that reflect their elemental origins and are revered by those who understand their true nature. As the world changed, the Mythos Creature adapted, forming an intricate web of life that ensures harmony and stability across the realms. Tales of their exploits are passed down through generations, inspiring awe, and respect for the natural world they so fiercely protect.

As a seasoned creature tracker, you've dedicated your life to studying these elusive Mythos Creatures. From the fiery peaks where the Blazefox reigns to the icy expanses patrolled by the majestic Frostwing, your journey has taken you to the most remote and dangerous corners of the earth. Armed with your knowledge and keen instincts, you track these awe-inspiring beings, unravelling the mysteries of their existence and ensuring that their secrets are preserved and protected from those who would exploit them. Your encounters with the Mythos Creatures are tales of wonder and peril, each discovery bringing you closer to understanding the delicate balance they maintain in the natural world.

Overview

This assignment will test your ability to implement a hash table and binary search tree, then use their functionality to implement a creature tracker that will maintain information about the creatures you have encountered along your journey. The creature tracker program will use a binary search tree to store creature details and a hash table (with separate chaining) to store information about the different types of creatures. The creature tracker will then have basic functionality to add creatures, remove creatures, and query various information.

This assignment is worth 100 marks and accounts for 15% of your final grade. Late submissions are subject to the rules specified in the Course Outline.

The Supplied Files

This section gives an overview of the files that you are provided as part of this assignment, none of which you should modify. You are not provided with (skeleton) implementation files – you will be expected to create these files from scratch. You are recommended to take some time to understand the overall structure of the program before starting your implementation.

- `main.cpp` – contains the main function and the logic to initialise the creature tracker dictionary and perform basic operations. **This file should not be modified.**

- `empty_collection_exception.h` – contains the definition of an exception that can be thrown when an operation is attempted on an empty tree. **Note:** there is no accompanying `.cpp` file – the `.h` file contains the complete definition and implementation of the exception. **This file should not be modified.**

- `creature.h` – contains the header for the `Creature` class, which includes the instance variables and behaviours that a creature contains. **This file should not be modified.**

- `creature_type_stats.h` – contains the header for the `CreatureTypeStats` class, which includes the instance variables and behaviours for the object that maintains information about a particular type of creature. **This file should not be modified.**

- `bt_node.h` – contains the header for the `BTNode` class, which includes the instance variables and behaviours that the binary tree node contains. **This file should not be modified.**

- `bs_tree.h` – contains the header for the `BSTree` class, which includes the instance variables and behaviours that the binary search tree contains. **This file should not be modified.**

- `hash_table.h` – contains the header for the `HashTable` class, which includes the instance variables and behaviours that the hash table contains. **This file should not be modified.**

- `creature_tracker.h` – contains the header for the `CreatureTracker` class, which includes the instance variables and behaviours that the creature tracker contains. **This file should not be modified.**

- `makefile` – the `makefile` for the project, which outlines the build recipe. The `-g` flag is set for you to make debugging and/or memory leak detection more convenient, should you wish to use these tools. **This file should not be modified.**

- `creatures.txt` – a text file containing a list of 250 Mythos Creature records. Each creature has a name, type, and power level. **This file should not be modified. You may consider using a smaller version for testing if you wish.**

Running the Program

Of course, you will need to ensure the program is compiled prior to running it. You can use the supplied `makefile` for this – it will produce an executable called `CreatureTracker`. The program can be executed as normal using the command:

```
./CreatureTracker
```

This will read a file named creatures.txt to populate the creature tracker with the data of 250 creatures that you have encountered. Several operations are then performed to test its functionality. However, this testing is not exhaustive and you are encouraged to tester broader functionality. An example of the program execution is shown in Figure 2, Figure 3, and Figure 4 – it is provided in 3 separate figures given the length of output. **Note:** not all output is captured in these screenshots.

Note: the program will not compile as supplied, as you will not have the necessary implementation files. You are encouraged to write skeleton files, similar to those provided in Assignment 1, to allow compilation of an incomplete program – this will allow you to work incrementally.

The Tasks

The first (and simplest) task is to implement the `Creature` and `CreatureTypeStats` classes. These classes will require a few operators to be overloaded such that they can be used in the tree and hash table and be inserted into a stream for printing.

The next task is to implement the templated `BTNode`, `BSTree`, and `HashTable` classes. For each class, you are provided with the header file and must adhere to its definitions. These headers will be substantially similar to those discussed in lecture and lab but may differ in a few minor ways. In particular, the hash table will use `std::list` for separate chaining and a few function definitions have been modified and/or added for completeness.

Finally, you will use your `BSTree` and `HashTable` classes to provide the expected functionality for the `CreatureTracker` class.

Some additional details about each class are provided below, but you should also examine the documentation in the provided files, which also contains important information about the expected behaviour.

`Creature`

The `Creature` class is a simple data class that stores information about a creature, particularly its name, type, and power rating. For a full list of methods required and some important details, you should examine the `creature.h` file and its associated documentation.

`CreatureTypeStats`

The `CreatureTypeStats` class is a simple data class that stores information about a specific type of creature, particularly the type, a count of the number of creatures of that type, and the

total power of creatures of that type. For a full list of methods required and some important details, you should examine the `creature_type_stats.h` file and its associated documentation.

BTNode

The `BTNode` class is a templated version of a node in a binary tree and should be implemented as discussed in lecture. For a full list of methods required and some important details, you should examine the `bt_node.h` file and its associated documentation.

BSTree

The `BSTree` class is a templated version of a binary search tree and should be implemented recursively as discussed in lecture. For a full list of methods required and some important details, you should examine the `bs_tree.h` file and its associated documentation.

By default, your binary search tree should print an inorder traversal, but should provide functionality to print according to all three traversals discussed in lecture (namely preorder, postorder, and inorder).

HashTable

The `HashTable` class is a templated version of a hash table that uses a linked list for separate chaining and should be implemented as discussed in lecture. For a full list of methods required and some important details, you should examine the `hash_table.h` file and its associated documentation.

Note: you will notice that `hash_table.h` contains a few helper methods implemented for you. For example, Figure 1 shows the hash function, which you are expected to use.

```
template <typename T>
int HashTable<T>::hash_function(const std::string& key) const
{
    std::hash<std::string> hf;
    return hf(key) % capacity;
}
```

Figure 1. Hash function to be used in the hash table implementation.

Additional helper methods are provided to support operations you will need to perform with the `std::list`, such as finding, removing, and printing items. You are encouraged to review these methods, and use them where appropriate. Further details about when a helper method may be appropriate are given in `hash_table.h`.

CreatureTracker

The CreatureTracker class contains the main logic of the creature tracking process and uses your BSTree and HashTable classes to support the functionality, as further detailed in the header file. Your hash table should be initialised with 101 cells, as is the default size in the header file, but should work for any provided capacity. For a full list of methods required and some important details, you should examine the creature_tracker.h file and its associated documentation.

The add_creature method will accept the name, type, and power of creature and should then record this creature's information. This includes both inserting a Creature object into the tree and adding or updating a CreatureTypeStats object in the hash table. Conversely, the remove_creature method should remove the Creature object from the tree and update the CreatureTypeStats object accordingly. Particularly, the CreatureTypeStats object should be removed from the hash table when no creatures of its type remain.

There are a few remaining methods that will be provided to support information querying from the creature tracker.

Marking

As mentioned previously, this assignment is worth 100 marks and accounts for 15% of your final grade. Late submissions are subject to the rules specified in the Course Outline.

Your submission will be assessed on both correctness and quality. This means, in addition to providing a correct solution, you are expected to provide readable code with appropriate commenting, formatting, best practices, memory management, etc. **Code that fails to compile will result in a zero for the functionality correctness section.** Your code will be tested on functionality not explicitly shown in the supplied demo file (i.e., using a different main.cpp file). Hence, you are encouraged to test broader functionality of your program before submission. There should be no segmentation faults or memory leaks during or after the execution of the program, including for functionality not explicitly used in the demo file.

Marking criteria will accompany this assignment specification and will provide an indicative guideline on how you will be evaluated. **Note**: the marking criteria is subject to change, as necessary.

Submission

Your submission should be made using the Assignment 3 link in the Assignments section of the course Canvas site. Assignment submissions will not be accepted via email. Incorrectly submitted assignments may be penalised.

Your submission should include only the completed versions of creature.cpp, creature_type_stats.cpp, bt_node.hpp, bs_tree.hpp, hash_table.hpp, and creature_tracker.cpp. You do not need to include any other files in your

submission. Be sure that your code works with the supplied files. Do not change the files we have supplied as your submission will be expected to work with these files – when marking your code, we will add the required files to your code and compile it using the supplied `makefile`.

Compress the required files into a single `.zip` file, using your student number as the archive name. Do not use `.rar`, `.7z`, `.gz`, or any other compressed format – these will be rejected by Canvas. For example, if your student number is c9876543, you would name your submission:

c9876543.zip

If you submit multiple versions, Canvas will likely append your submission name with a version number (e.g., **c9876543-1.zip**) – this is not a concern, and you will not lose marks.

Remember that your code will conform to C++ best practices, as discussed in lecture, and should compile and run correctly using the supplied `makefile` in the standard environment (i.e., the Debian virtual machine). **If you have developed your solution using any other environment, be sure to test it with the VM prior to submission.** Please see the accompanying marking criteria for an indicative (but not final) guideline to how you will be evaluated.

Helpful Tips

A few tips that may help you along your journey.

1. Read the header files carefully – they provide further information on the specification for various methods.

2. Work incrementally – don't try to implement everything at once. Consider getting a barebones version to compile, which will enable you to test methods as you implement them.

3. Remember that you can debug your program in VS Code. See Lab3a for a brief guide. Of course, the name of the executable in your `launch.json` file will be different than the example in the lab, but this gives you a great way to inspect your data structures during the program execution. You will need to think carefully about what you would expect the data to look like.

4. You will be expected to supply a program with no memory leaks. You are encouraged to use `valgrind` to assess whether you have any leaking memory in your program.

5. Start early! The longer you wait to start, the less time you will have to complete the assignment. This is particularly important for Assignment 3, as the due date is immediately before the examination period – hence, it is in your best interest to complete the assignment ASAP to allow sufficient time to prepare for your examinations.

Good Luck!

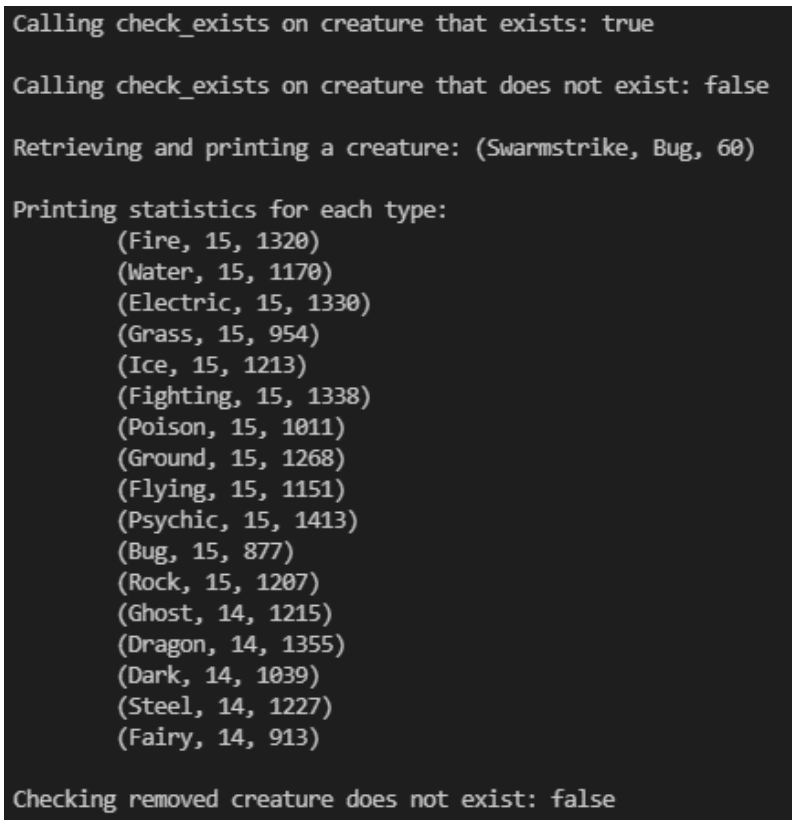

Figure 2. Example execution of the program, only showing the first few outputs.

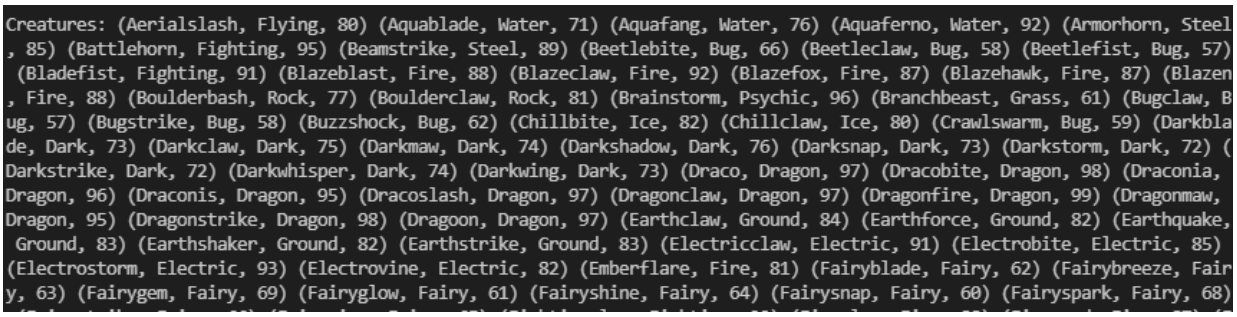

Figure 3. Example execution of the program, showing the creature tree being printed. Only some output is shown, given its length. Note that, for an inorder traversal (as is default), the creatures will be alphabetical order! Each entry indicates the creature's name, type, and power rating.

```
Type stats:
0:
1:
2:
3:
4:
5: (Poison, 15, 1011)
6:
7:
8:
9:
10:
11:
12:
13:
14:
15:
16: (Electric, 15, 1330)
17:
18:
19:
20: (Grass, 15, 954) (Dark, 14, 1039)
21:
22:
23:
24:
25:
```

Figure 4. Example execution of the program, showing the type stats hash table being printed. Lines with multiple entries indicate collisions, which are stored in a list. Only some output is shown, given its length. Each entry indicates information about the type. For example, the entry at index 5 indicates there are 15 creatures with a type of Poison. Together, these 15 creatures have a total power rating of 1011.