EECS268:Lab9

Contents

Due Date

This lab is due ~~one week~~ two weeks from the start of your lab.

Overview

The next two labs will use Binary Search Trees. For this lab we will only implement adding, searching, and traversing a BST. The next lab is when we'll implement other methods like removal and copy. We'll cover more BST topics in lecture next week.

Phase 1:

- Adding to a BST
- Searching a BST
- Visiting a BST in pre, in, and post orders

Phase 2:

- Coming in lab 10...

Pokemon File

You can download a sample file here (https://people.eecs.ku.edu/~jwgibbo/eecs268/2021fall/labs/lab08/pokemon.txt). Each line is organized in the following manner (white-space delimited):

```
<american pokemon name> <pokedex number> <japanese pokemon name>
```

American Name	What the pokemon is called in the U.S.A.
pokedex number	A number associated with a particular pokemon. Think of it as a pokemon's SSN
Japanese pokemon name	What the pokemon is called in Japan

Example entries:

```
Abra    63      Casey
Aerodactyl      142     Ptera
Alakazam        65      Foodin
Arbok   24      Arbok
Arcanine        59      Windie
Articuno        144     Freezer
Beedrill        15      Spear
```

```
Bellsprout      69      Madatsubomi
Blastoise       9       Kamex
Bulbasaur       1       Fushigidane
```

This lab you won't be told how many entries are in the file, but reading until the end of a file isn't hard:

Example:

```
//assume we've opened the file an have some temp variables for reading

while( myInFile >> tempUS >> tempID >> tempJP )
{
    //do something with temp variables, perhaps build a Pokemon object and add it to the BST
}
```

I know that looks weird, but essentially ifstream is overloaded to return false when it hits the end of the file.

Requirements

The next lab will involve Binary Search Trees. The table lists the functionality that you can accomplish by loading the Pokedex entries into a single Binary Search Tree. I recommend getting everything in phase 1 working before moving on to phase 2.

Notes:

- The user will provide the name of an input file formatted in the way described above from the command line
- Using that file, create and load a BST full of Pokemon
- Until the user wants to quit, let them use your pokedex in the ways listed in the table below:
  - The table list the menus labels and desired outcomes, but you will need to use the BinarySearchTree method that accomplishes the task
  - Example: If the user wants to **Search** you will need to call the **search(key)** method from the BinarySearchTree and see whether it produces a result or throws an exception to verify if the Pokemon is in the BST

Phase 1

Search	Given pokedex number (id) print all information (US name, Japanese name, pokedex number) to the user
Add	Prompt the user for a new Pokemon name (US), then new Japanese name and Pokedex number and add the entry to the tree. **Duplicates should not be allowed.** Make your add method throw an exception (std::runtime_error) if a duplicate is attempted to be added.
Print	Prompt the user for the following: ▪ Traversal order; The user can choose for the pokedex to be written in in, pre, or post order.
Quit	Exits the program

Implementation Details

- We ask that your BST be templated with two types, an ItemType and a KeyType
  - The ItemType would be the type returned from a search
  - The KeyType would be the type that you can search on
  - For example, if I search for the Pokemon with ID 25 I should get back the whole entry for that word when I search.
- The BST should only use the default comparison operators, and **not be coupled** to the methods of any specific class
  - Overload the needed comparison operators for your class
  - We will order them numerically based on their pokedex number
- If a copy is made and edits are performed, the original should remain unchanged.
- You'll notice that the traversal functions take a function as a parameter. Essentially, you're passing a function that will be called on each entry in the BST

    - The function you pass in will be a **static member** of another class (e.g. Executive)
    - A static method is a method that does NOT refer or use any instance variables (e.g. member variables) or call any non-static methods
    - A static method is described as "belonging to the class" rather than belonging to a specific instance.
    - It may look something like:

```
class Executive
{
    //other members
    static void pokemonPrinter(Pokemon p);//prints a single pokemon. Does NOT use any member variables/metho

```

WARNING: You only put the listing of **static** in the header file. DO NOT put the static keyword in your cpp file!

BST Interface

```
template <typename ItemType, typename KeyType>
class BinarySearchTreeInterface
{
    public:
    virtual ~BinarySearchTreeInterface(){}
    virtual void add(ItemType entry) = 0; //throws std::runtime_error if duplicate added
    virtual ItemType search(KeyType key) const = 0; //throws std::runtime_error if not in tree
    virtual void clear() = 0; //Empties the tree
    virtual void remove(KeyType key) = 0; //throws std::runtime_error if not in tree


    //For the following methods, each method will take a function as a parameter
    //These function then call the provided function on every entry in the tree in the appropriate order (i.
    //The function you pass in will need to a static method
    virtual void visitPreOrder(void visit(ItemType)) const = 0; //Visits each node in pre order
    virtual void visitInOrder(void visit(ItemType)) const = 0; //Visits each node in in order
    virtual void visitPostOrder(void visit(ItemType)) const = 0; //Visits each node in post order
};
```

Rubric

- 65% Pokedex Interaction
    - 20% Searching
    - 20% Adding entry
    - 25% Correct traversal orders
- 10% Terminal output
    - 10% well formatted output
- 15% Program stability
- 5% Logical user interface
- 5% Needed Operators overloaded

Retrieved from "https://wiki.ittc.ku.edu/ittc_wiki/index.php?title=EECS268:Lab9&oldid=23612"