**Functional programming language (Haskell)**

For my implementation of a binary tree using a functional programming language. I decided to use Haskell, as I already had experience in Haskell doing Binary search trees. From my knowledge, Haskell has data structures implemented into the coding language. As commented in my Haskell program I used the sites for research.

<https://www101.dcu.ie/registry/module_contents.php?function=2&subcode=CA320>

<https://hackage.haskell.org/package/type-indexed-queues-0.2.0.0/docs/Data-BinaryTree.html>

This helped me implement a constructor into my program “Node”. My original idea was to use the “Leaf” constructor too, but I found it was over complex and unnecessarily. As a leaf is basically a node without any child nodes, as I had base cases and conditions that would detect when a node had no left or right subchild , I didn’t see the need of using a “Leaf” constructor . From an overview perspective, I actually re wrote the “Leaf” constructor just in a more basic manor .I made the binary tree generic with the “Ord” typeclass that allows your function to work for any type binary tree (see code below) I had already used “Ord” in other Haskell programs , even though binary trees are done mainly with Integers , I still think having generic modification on the program is more practical . From a Haskell point of view generics isn’t a major concern , but it is a highly recommended method when understood as generics originally comes from the Java programming language and for any user wanting to test your code , I think its critical that they aren’t restricted to one type of data input.

insertNodes :: (Ord tree) => tree -> BinaryTree tree -> BinaryTree tree

searchNodes :: (Ord tree) => BinaryTree tree -> tree -> Bool

I used the following links below to get a better understanding of tree traversal algorithms. Once understood, I implemented my ideas in the Haskell binary tree to traverse them. Preorder is useful for creating a copy of the tree. Inorder allows to create a non-decreasing tree order and postorder is useful for deleting the tree. Also, postorder is a useful way to remove a node from a tree First search to sort a list of elements, which wasn’t quite hard to implement as this is sorting algorithm I have used before

<https://stackoverflow.com/questions/26228816/haskell-pre-order-traversal-tree-to-list>

<http://hackage.haskell.org/package/tree-traversals-0.1.1.0/docs/Data-Traversable-TreeLike.html>

The beauty of using Haskell is that I can use recursive functions and given the fact that binary trees are done reclusively. It was more simple to do than the prolog version and my implementation is concise. If I wanted to implement a **remove** function , I could simply add a built-in function “delete” or make my own recursive function that searches for a element in a list and removes it by remove the head each time and recursively calling the tail of the list

**Logic programming language (Prolog)**

For my implementation of a binary tree using a Logic programming language. I decided to use Prolog. I had previously completed a Logic module using the prolog language therefore having experience with the programming language. Unlike Haskell I couldn’t use data structures or functions, instead I used predicates which is form of propositional logic. In more simpler terms my binary tree had to be built on facts, causes and hence “logic”. As mentioned in my Prolog program, I got the ideas from CA208 Logic module notes, swi-prolog docs and for the tree traversal (inorder, preorder , postorder) I used a prolog tutorial for structures and paths.

<https://www.computing.dcu.ie/~davids/CA208_Prolog_2p.pdf>

<https://www.swi-prolog.org/pldoc/index.html>

[**https://www3.cs.stonybrook.edu/~pfodor/courses/CSE595/L09\_Prolog.pdf**](https://www3.cs.stonybrook.edu/~pfodor/courses/CSE595/L09_Prolog.pdf)

For the Search and Insert Predicates, I implemented causes and facts. Having a base case for each of them, using “\_” underscore variable or in other words “I don’t care variable” this is built in method to allow for when a “atom” isn’t necessarily. Basically, after the base case I check the right and left side of the binary tree using logic, to either search or insert a specific node. Prolog is a pretty intuitive language and you can’t return a list as you are writing predicates, making it quite hard to have an abstract implementation. I noticed that I was more restricted than Haskell to achieve my goal of a binary tree. In my program the preorder traversal was quite simple to implement as it uses breadth first search algorithm and I was able to understand how to correctly write logical facts for it. Although my implantation for postorder and inorder has mixed outputs as mentioned in the program, as unlike Haskell I couldn’t use any functions or change the flow state giving me issues for consistent output.

Overall doing binary tree using a logic-based programming language can give more issues as it’s a more “black and white” ideology.

**Functional programming language (Haskell)  VS Logic programming language (Prolog)**

From my two implementations it’s clear that Functional programming has its benefits over Logic programming language. Being able to create function and change state, unlike Prolog it uses recursion and is more expensive to use. Haskell is easier to define your concept, coming from a Computer course that is more enthusiastic about programming than maths, it’s easier to understand the language at more rapid pace. I believe if you come from a maths course, prolog would be easier to understand as it is more of a mathematical relation. Haskell allows to use data structures library’s, create constructors which then will allow the user to create objects for reusability and unlike in prolog where your more restricted to what you can do . Haskell programs are usually shorter and more to point as shown in my two programs. Although there is still advantages to prolog as it easier to read when building structures, it does have built in list handling which I did notice when programming it. This is useful for sequences and also Prolog will give the best performance and reliability as the engine will have better development effort and overall greater design. I noticed when doing Haskell binary tree, I had to implement some sort of “pattern matching” unlike prolog where it assisted of using “backtracking” which helps for order and more possible solutions.

Overall both languages of course has its advantages and disadvantages , but Functional programming will always be preferred from a computer scientist perspective as its simply more robust and abstract than logic programming language.