Overview

A schematic and testing program for a binary tree structure.

Introduction

This document holds the technical design of the CSCI312\_A4\_Schmidt program and serves as a pseudo lessons learned for the changes made to the initial design. The program holds a basic binary tree structure and a testing program for it.

## Scope

Holds a binary tree structure for storing any datatype.

### PROCESSING

The program creates an instance of a BineryTreeStructure and adds seven characters to it. Then it displays the PreOrder, Inorder, and Postorder searches of the BineryTreeStructure. The user is then given the option to search the BineryTreeStructure and the program prints every node it looks at. Lastly, it prints the size of the BineryTreeStructure.

.

### DATA

The logical and physical data structure of files should be defined in detail.

Data structure definitions must include the:

description of each element, e.g. name, type, dimension;

relationships between the elements, i.e. the structure;

range of possible values of each element;

initial values of each element.

BineryTreeStructure

Holds Node, rootNode.

Holds integer, counter.

Node

Holds the Key, rightNode, leftNode, and any object.

### COMPONENTS

BineryTreeTester

* Holds the static main and runs the program through the process.

BineryTreeStructure

* Data class that defines the BineryTreeStructure and the Node.

|  |
| --- |
| **BineryTreeStructure** |
| -int: m\_Size  -Node: m\_RootNode |
| +getSize() : int  +addNode(int, Object): void  +addNode(Node, int, Object): void, private  +findNode(int): Object  +findNode(Node, int): Node, private  +printPreorder(): void  + printPreorder(Node): void  +printInorder(Node): void  +printInorder(): void  +printPostOrder(): void  +pritnPostOrder(Node): void |

|  |
| --- |
| **Node** |
| -int: m\_Key  -Object: m\_Obj  -Node: rightNode  -Node: leftNode  -int: m\_size |
| +getObj(): Object  +getKey(): int  +getLeftNode(): Node  +getRightNode(): Node  +setLeftNode(Node): void  +setRightNode(Node): void |

### TESTING

Present one or more named scenarios that will be utilized to test the application.

The testing plan should be repeatable.

Describe the scenario in detail, the steps required to execute the test, the input data, the output data, and the success criteria.

Present a summary of the testing scenarios before the details of each scenario.

Scenario #1- mystuff.txt test

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1. | Run testing data | Input: String provided by teacher |
| 2. | Enter character to search | Output: Printed text. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | Printed text with the algorithms running correctly. |
| ACTUAL OUTPUT | | The algorithm codes were completely off, didn’t work. |
| RESULTS – The desired output of a .txt file with the findings was produced. | | Fail |

Scenario #2- wap.txt test

##### 

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1. | Run testing data | Input: String provided by teacher |
| 2. | Enter character to search | Output: Printed text. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | Printed text with the algorithms running correctly. |
| ACTUAL OUTPUT | | Everything printed correctly. |
| RESULTS – The desired output of a .txt file with the findings was produced. | | Pass |