Tree

A tree is a non-linear data structure where the data are organized in a hierarchical manner.

A tree is a finite set of one or more nodes such that

* There is a specially designated node called the root.
* The remaining nodes are partitioned into n ≥ 0 disjoint sets T1, T2, . . . , Tn, where each of these sets is a tree. T1, T2, . . . , Tn are called the subtrees of the root.

Binary Search Tree (BST)

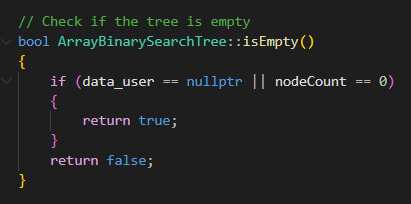
A binary search tree (BST) is a binary tree. It may be empty. If it is not empty then it satisfies the following properties:

* Each node has exactly one key and the keys in the tree are distinct.
* The keys (if any) in the left subtree are smaller than the key in the root.
* The keys (if any) in the right subtree are larger than the key in the root.
* The left and right subtrees are also binary search trees.

Functions:

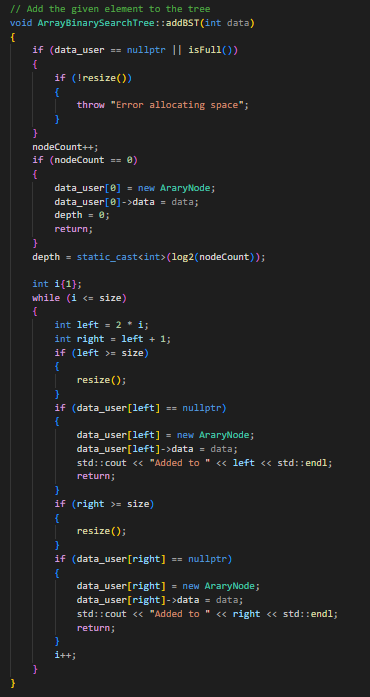
1. **isEmpty():**

* Returns true if the tree is empty, and false otherwise



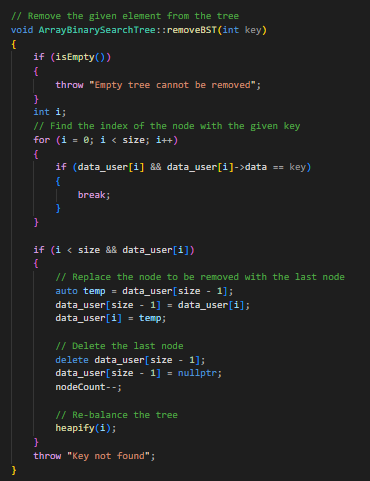
1. **addBST(data):**

* Inserts an element to the BST



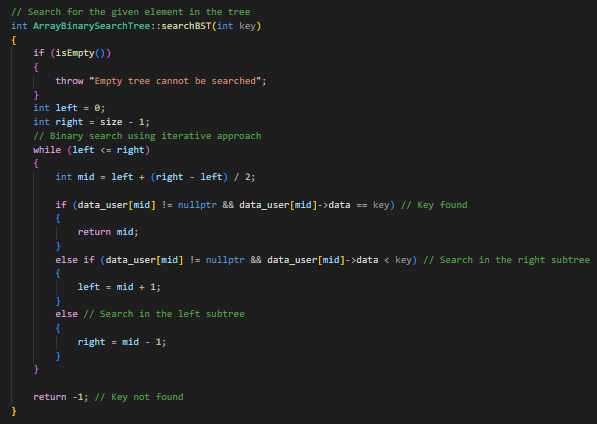
(c) removeBST(keyToDelete):

* Removes the node with the given key from the BST



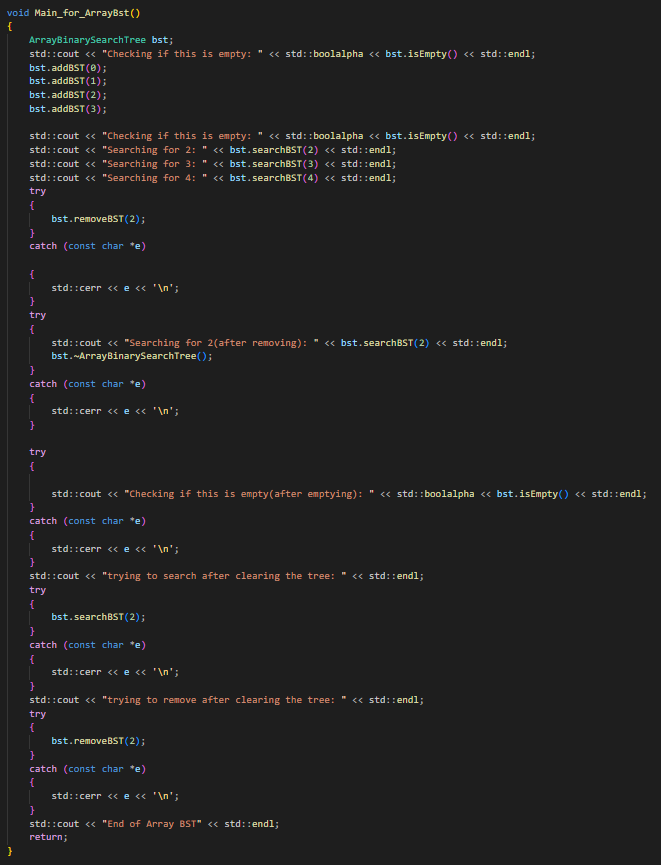
1. **searchBST(targetKey):**

* Returns true if the key exists in the tree, and false otherwise

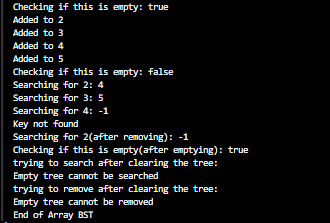


Main Functions and it’s output:

Main:

:

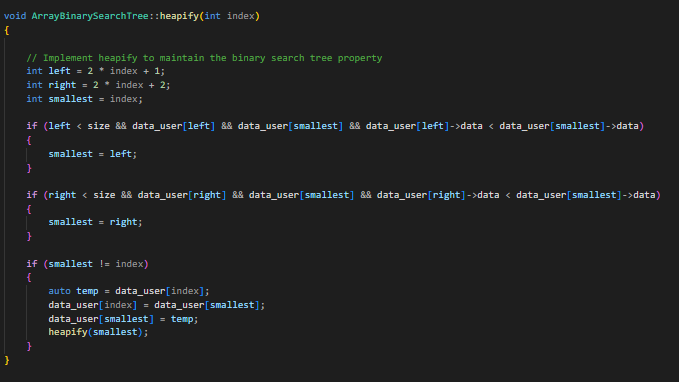
Output:



Other important functions:

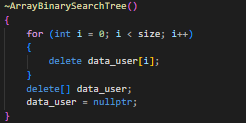
**Heapify(index):**

* This function will take an index and maintain the heap structure



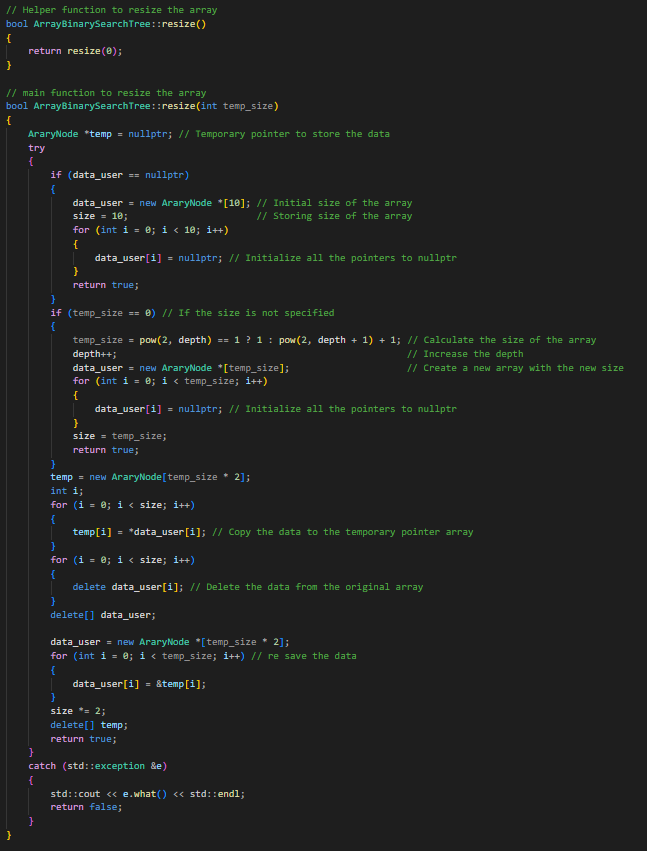
~ArrayBinarySearchTree()

* This function will remove all the elements stored in the “data\_user”



Resize():

* This function will resize the array that is storing the data.



Note:

* This is just the Array implementation of the binary search tree since my roll number is odd (27).