b) If `myTree` is the `BinaryTree` object for the expression: 

\[(4 + 5) \times 7\]

what gets printed by a call to:

<table>
<thead>
<tr>
<th><code>myTree.inorder()</code></th>
<th><code>myTree.preorder()</code></th>
<th><code>myTree.postorder()</code></th>
<th><code>inorder(myTree)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>4 \n* \n5 \n* \n7 \n</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{c) If } \text{myTree}\text{ is the }\text{BinaryTree}\text{ object for the expression: } \((4 + 5) \times 7\), \text{ what gets printed by a call to } \text{myTree.printexp()?}
\]

\[
\text{d) If } \text{myTree}\text{ is the }\text{BinaryTree}\text{ object for the expression: } \((4 + 5) \times 7\), \text{ what gets returned by a call to } \text{myTree.postordereval()?}
\]

c) Write the `height` method for the `BinaryTree` class.

\[
\text{height (tree)} = \max(\text{height left subtree}, \text{height right subtree})
\]

4. Consider the Binary Search Tree (BST). For each node, all values in the left-subtree are < the node and all values in the right-subtree are > the node.

\[
\text{a. What is the order of node processing in a preorder traversal of the above BST?}
\]

```
50, 30, 9, 5, 18, 34, 32, 38, 47, 70, 58, 65, 80
```

\[
\text{b. What is the order of node processing in a postorder traversal of the above BST?}
\]

```
5, 18, 9, 33, 32, 47, 34, 30, 65, 58, 80, 70, 50
```

\[
\text{c. What is the order of node processing in an inorder traversal of the above BST?}
\]

```
9, 18, 32, 33, 34, 35, 47, 50, 58, 65, 70, 80
```

\[
\text{d. Starting at the root, how would you find the node containing “32”?}
\]

Right of 50, left at 30, right at 34

\[
\text{e. Starting at the root, when would you discover that “65” is not in the BST?}
\]

Walk down branch until at 58 and seeing that 58 has no right child

\[
\text{f. Starting at the root, where would be the “easiest” place to add “65”?}
\]

As right child of 58

\[
\text{g. Where would we add “5” and “33”?}
\]

See above
1. Consider the partial `TreeNode` class and partial `BinarySearchTree` class.

```python
class TreeNode:
    def __init__(self, key, val, left=None, right=None, parent=None):
        self.key = key
        self.payload = val
        self.leftChild = left
        self.rightChild = right
        self.parent = parent

    def hasLeftChild(self):
        return self.leftChild

    def hasRightChild(self):
        return self.rightChild

    def isLeftChild(self):
        return self.parent and \n            self.parent.leftChild == self

    def isRightChild(self):
        return self.parent and \n            self.parent.rightChild == self

    def isRoot(self):
        return not self.parent

    def isLeaf(self):
        return not (self.rightChild or self.leftChild)

    def hasAnyChildren(self):
        return self.rightChild or self.leftChild

    def hasBothChildren(self):
        return self.leftChild and self.rightChild

    def replaceNodeData(self, key, value, lc, rc):
        self.key = key
        self.payload = value
        self.leftChild = lc
        self.rightChild = rc
        if self.hasLeftChild():
            self.leftChild.parent = self
        if self.hasRightChild():
            self.rightChild.parent = self

    def __iter__(self):
        if self:
            if self.hasLeftChild():
                for elem in self.leftChild:
                    yield elem
            yield self.key
            if self.hasRightChild():
                for elem in self.rightChild:
                    yield elem
```

A `BinarySearchTree` object

```
self
```

- `size`: root is None if size is 0
- `root`: TreeNode objects

```python
class BinarySearchTree:
    def __init__(self):
        self.root = None
        self.size = 0

    def length(self):
        return self.size

    def __len__(self):
        return self.size

    def __iter__(self):
        return self.root.__iter__()

    def __str__(self):
        """Returns a string representation of the tree rotated 90 degrees counter-clockwise""

    def strHelper(root, level):
        resultStr = ""
        if root:
            resultStr += strHelper(root.rightChild, level+1)
            resultStr += "|" * level + "\n"
            resultStr += str(root.key) + "\n"
            resultStr += strHelper(root.leftChild, level+1)
            return resultStr

    return strHelper(self.root, 0)
```

---

a) How do the `BinarySearchTree` `__iter__` and `__str__` methods work?
Data Structures (CS 1520)  
Lecture 19  
Name:  

More partial TreeNode class and partial BinarySearchTree class.

```python
class BinarySearchTree:
    ...
    def _contains_(self, key):
        if self._get(key, self.root):
            return True
        else:
            return False
    
def get(self, key):
        if self.root:
            res = self._get(key, self.root)
            if res:
                return res.payload
            else:
                return None
        else:
            return None
    
def _get(self, key, currentNode):
        if not currentNode:
            return None
        elif currentNode.key == key:
            return currentNode
        elif key < currentNode.key:
            return self._get(key, currentNode.leftChild)
        else:
            return self._get(key, currentNode.rightChild)
    
def __getitem__(self, key):
        return self.get(key)
    
def __contains__(self, key):
        return self.get(key)
    
def put(self, key, val):
        if self.root:
            self._put(key, val, self.root)
        else:
            self.root = TreeNode(key, val)
            self.size = self.size + 1
    
def _put(self, key, val, currentNode):
        if key < currentNode.key:
            if currentNode.hasLeftChild():
                self._put(key, val, currentNode.leftChild)
            else:
                currentNode.leftChild = TreeNode(key, val, parent=currentNode)
        elif key > currentNode.key:
            if currentNode.hasRightChild():
                self._put(key, val, currentNode.rightChild)
            else:
                currentNode.rightChild = TreeNode(key, val, parent=currentNode)
        else:
            currentNode.payload = val
            self.size += 1
```

A BinarySearchTree object

![Diagram of TreeNode objects](image)

b) The _get method is the "workhorse" of BST search. It recursively walks currentNode down the tree until it finds key or becomes None.

In English, what are the base and recursive cases?

**Base Cases:**
1) Walk off branch of BST
2) Find node with key

**Recursive:**
1) Search left subtree
2) Search right subtree

c) What is the put method doing?

**Check for**

d) Complete the recursive _put method.

e) Draw the "shape" of the BST after puts of:
50, 60, 30, 70, 90, 40, 65

![BST diagram with values](image)

f) If “n” items are in the BST, what is put’s: Best-case O(1)? Worst-case O(n)? Average-case O(n)?