parse tree hesaplayıcı

gerçekleme

```
1           def evaluate(parseTree):
2               opers = {'+':operator.add, '-':operator.sub,
3                        '*':operator.mul, '/':operator.div}
4               leftC = parseTree.getLeftChild()
5               rightC = parseTree.getRightChild()
6
7               if leftC and rightC:
8                   fn = opers[parseTree.getRootVal()]
9                   return fn(evaluate(leftC),evaluate(rightC))
10              else:
11                  return parseTree.getRootVal()
```

→ s8: özyinelidir

parse tree hesaplayıcı

gerçekleme

```
1           def evaluate(parseTree):
2               opers = {'+':operator.add, '-':operator.sub,
3                        '*':operator.mul, '/':operator.div}
4               leftC = parseTree.getLeftChild()
5               rightC = parseTree.getRightChild()
6
7               if leftC and rightC:
8                   fn = opers[parseTree.getRootVal()]
9                   return fn(evaluate(leftC),evaluate(rightC))
10              else:
11                  return parseTree.getRootVal()
```

→ s8: özyinelidir

→ s2: fonksiyon pointerları (etiketleri)

parse tree hesaplayıcı

gerçekleme

```
1           def evaluate(parseTree):
2               opers = {'+':operator.add, '-':operator.sub,
3                        '*':operator.mul, '/':operator.div}
4               leftC = parseTree.getLeftChild()
5               rightC = parseTree.getRightChild()
6
7               if leftC and rightC:
8                   fn = opers[parseTree.getRootVal()]
9                   return fn(evaluate(leftC),evaluate(rightC))
10              else:
11                  return parseTree.getRootVal()
```

- → s8: özyinelidir
- → s2: fonksiyon pointerları (etiketleri)
- → s7-8: sol ağacın değeriyle sağ ağacın değeri üzerinde işlem yapılır

parse tree hesaplayıcı

gerçekleme

```
1           def evaluate(parseTree):
2               opers = {'+':operator.add, '-':operator.sub,
3                        '*':operator.mul, '/':operator.div}
4               leftC = parseTree.getLeftChild()
5               rightC = parseTree.getRightChild()
6
7               if leftC and rightC:
8                   fn = opers[parseTree.getRootVal()]
9                   return fn(evaluate(leftC),evaluate(rightC))
10              else:
11                  return parseTree.getRootVal()
```

→ s8: özyinelidir

→ s2: fonksiyon pointerları (etiketleri)

→ s7-8: sol ağacın değeriyle sağ ağacın değeri üzerinde işlem yapılır

→ çocuk yoksa (s9), kökteki değeri döndür (s10)

→ örnek: (3 + (4 * 5)) (S9)

5.5.2 ağaçta dolaşım

preorder, inorder, postorder

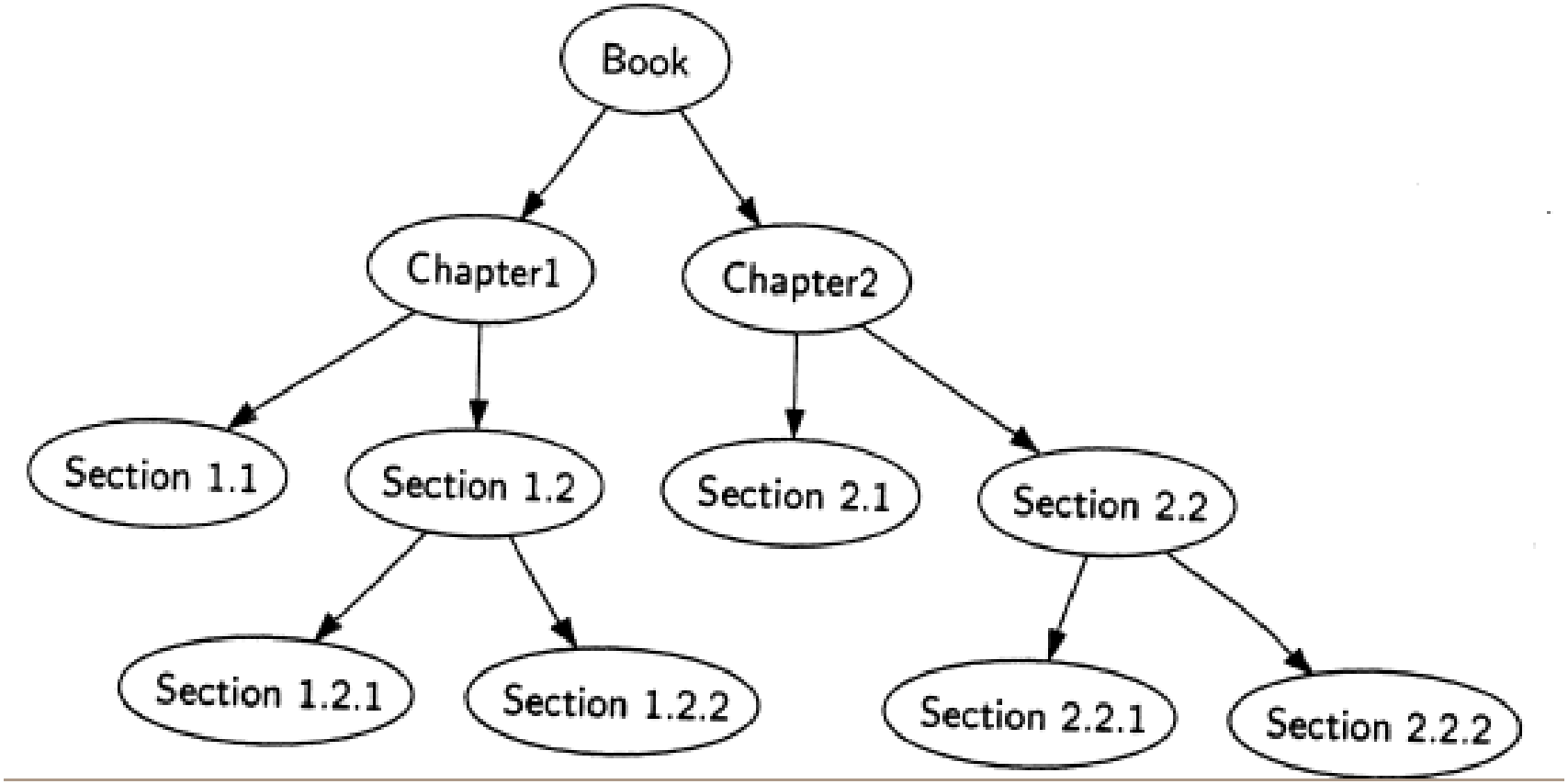
preorder: kök $\Longrightarrow$ *solaltağaç* $\Longrightarrow$ *sağaltağaç*

inorder: sol altağaç $\Longrightarrow$ *kök* $\Longrightarrow$ *sağaltağaç*

postorder: sol altağaç $\Longrightarrow$ *sağaltağaç* $\Longrightarrow$ *kök*

örnek

aşağıdaki ağacı pre|post|inorder ile tarayınca ne olur?



preorder: gerçekleme

gerçekleme (dış işlev olarak)

```
1       def preorder(tree):
2           if tree:
3               print tree.getRootVal()
4               preorder(tree.getLeftChild())
5               preorder(tree.getRightChild())
```

preorder: gerçekleme

gerçekleme (sınıfın yöntemi olarak)

```
1           def preorder(self):
2               print self.key
3               if self.left:
4                   self.left.preorder()
5               if self.right:
6                   self.right.preorder()
```

postorder: gerçekleme

gerçekleme (dış işlev olarak)

```
1       def postorder(tree):
2           if tree != None:
3               postorder(tree.getLeftChild())
4               postorder(tree.getRightChild())
5               print tree.getRootVal()
```

postorder: eval: gerçekleme

postorder ağacı hesaplamak için

```
1       def postordereval(tree):
2           opers = {'+':operator.add, '-':operator.sub,
3                    '*':operator.mul, '/':operator.div}
4           res1 = None
5           res2 = None
6           if tree:
7               res1 = postordereval(tree.getLeftChild())
8               res2 = postordereval(tree.getRightChild())
9           if res1 and res2:
10              return opers[tree.getRootVal()](res1,res2)
11          else:
12              return tree.getRootVal()
```

→ dipnot: defterden çizim (S11)

inorder: gerçekleme

gerçekleme (dış işlev olarak)

```
1       def inorder(tree):
2           if tree != None:
3               inorder(tree.getLeftChild())
4               print tree.getRootVal()
5               inorder(tree.getRightChild())
```

inorder: gerçekleme (full parantezli çıktı)

gerçekleme: full parantezli çıktı

```
1       def printexp(tree):
2           sVal = ""
3           if tree:
4               sVal = '(' + printexp(tree.getLeftChild())
5               sVal = sVal + str(tree.getRootVal())
6               sVal = sVal + printexp(tree.getRightChild())+')'
7           return sVal
```

→ dipnot: defterden çizim (S12)

örnek

örnek: PB:5220

```
>>> from listing_5_5_8 import *
>>> from listing_5_16 import *
>>> x = BinaryTree('*')
>>> x.insertLeft('+')
>>> l = x.getLeftChild()
>>> l.insertLeft(4)
>>> l.insertRight(5)
>>> x.insertRight(7)
>>>
>>> print printexp(x)
(((4)+(5))*(7))
```