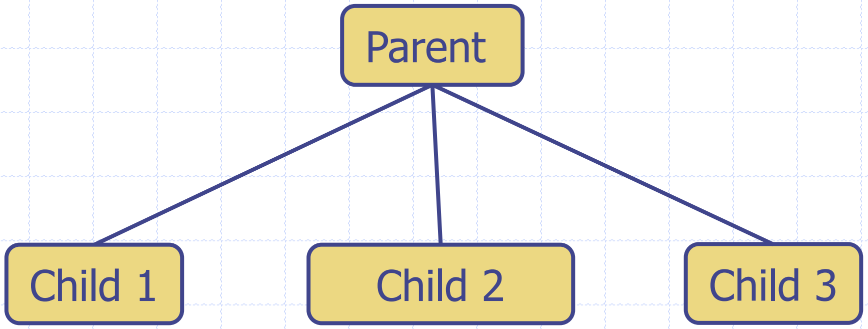
Lecture 5: Binary Trees

Sequential Unfoldment of Natural Law

Wholeness Statement

The Tree ADT is a generalization of the linked-list in which each tree node can have any number of children instead of just one; trees provide wide ranging capabilities and a highly flexible perspective on a set of element objects. *Science of Consciousness*: The whole infinite range of space and time is open to individuals with fully developed awareness. Regular transcending enlivens the qualities of the unified field in individual awareness and collective life.

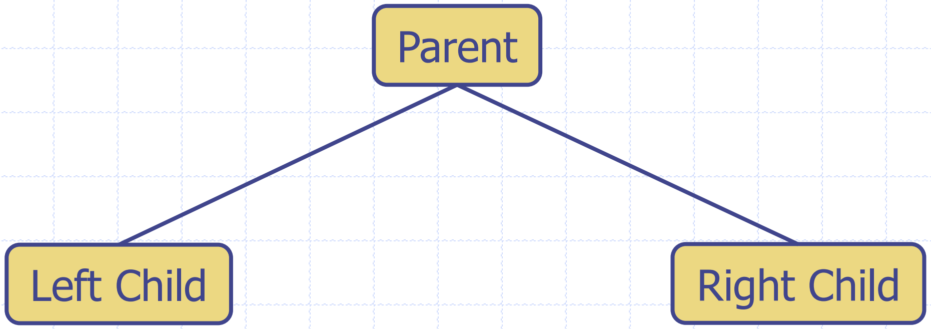
Trees



Outline

- BinaryTree ADT
- Data structures for trees
- Preorder and postorder traversals
- Inorder traversal
- Euler Tour traversal
- Template method pattern

Binary Trees

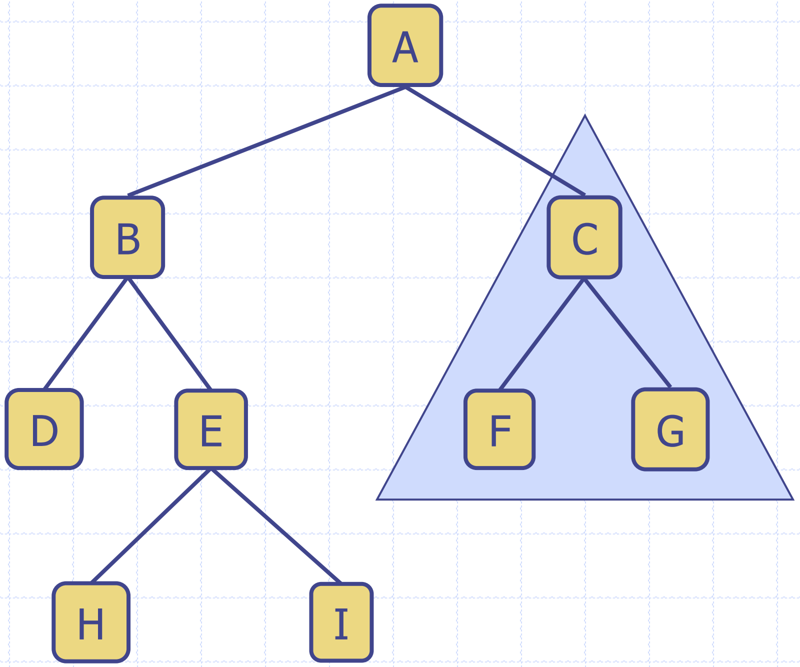


Outline

- BinaryTree ADT
- Data structures for trees
- Preorder and postorder traversals
- Inorder traversal

Tree Terminology

- ***Root***: only node without parent (A)
- ***Internal node***: node with at least one child (A, B, C, E)
- ***External node*** (a.k.a. leaf): node without children (D, H, I, F, G)
- ***Ancestors of a node***: parent, grandparent, grand-grandparent, etc.
- ***Depth of a node***: number of ancestors
- ***Height of a tree***: maximum depth of any node (3 in tree to right)
- ***Descendant of a node***: child, grandchild, grand-grandchild, etc.
- ***Subtree***: tree consisting of a node and its descendants (C, F, G)



Position and Tree ADT

- A Position or node of a tree is represented by an object storing
  - Element
  - Reference to the Parent node
  - Reference to the children nodes (a List of children for a Generic Tree)

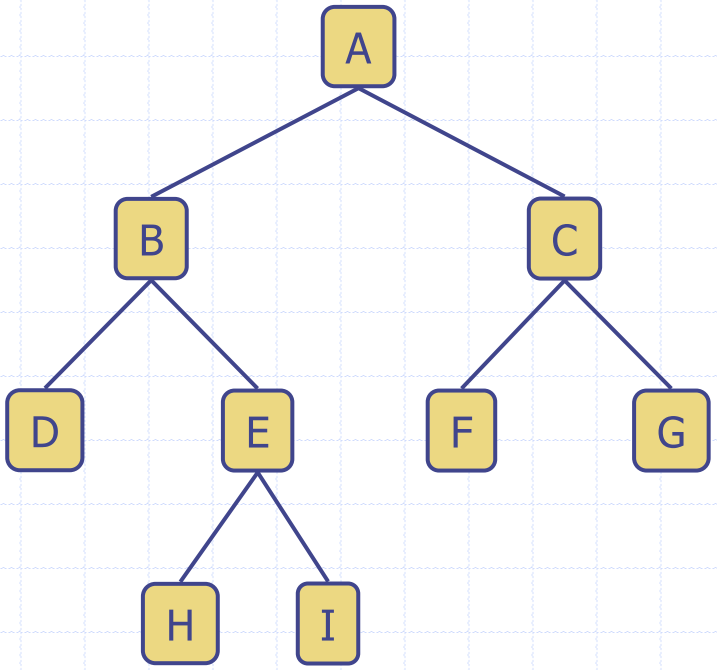
- We use positions to abstract nodes
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - objectIterator elements()
  - positionIterator positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - position leftChild(p)
  - position rightChild(p)

- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Update methods:
  - swapElements(p, q)
  - object replaceElement(p, o)
- Additional update methods would also be defined by data structures implementing the Tree ADT

Binary Tree

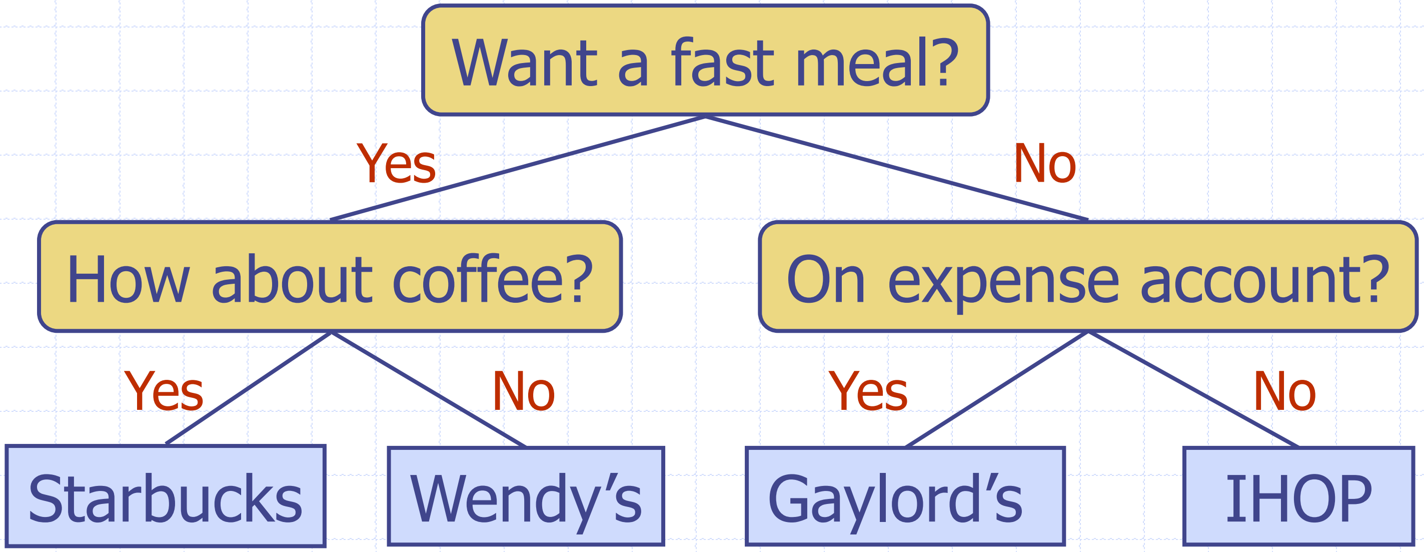
- A (proper) binary tree is a tree with the following properties:
  - Each internal node has two children
  - Each external node has no children
  - The children of a node are an ordered pair
- We assume that all binary trees are proper
- We call the children of an internal node left child and right child
- A binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree

- Applications:
  - arithmetic expressions
  - decision processes
  - searching



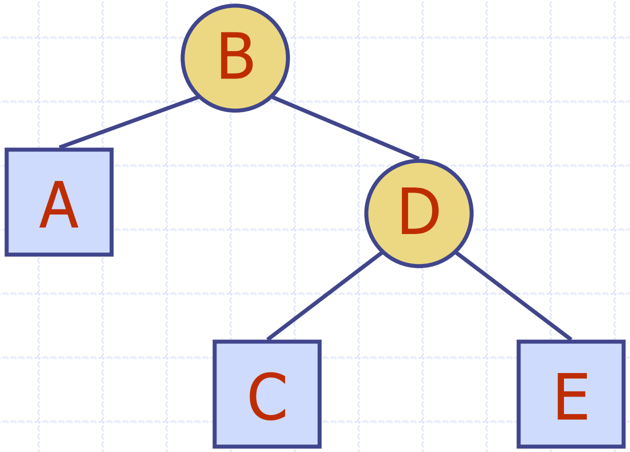
Decision Tree

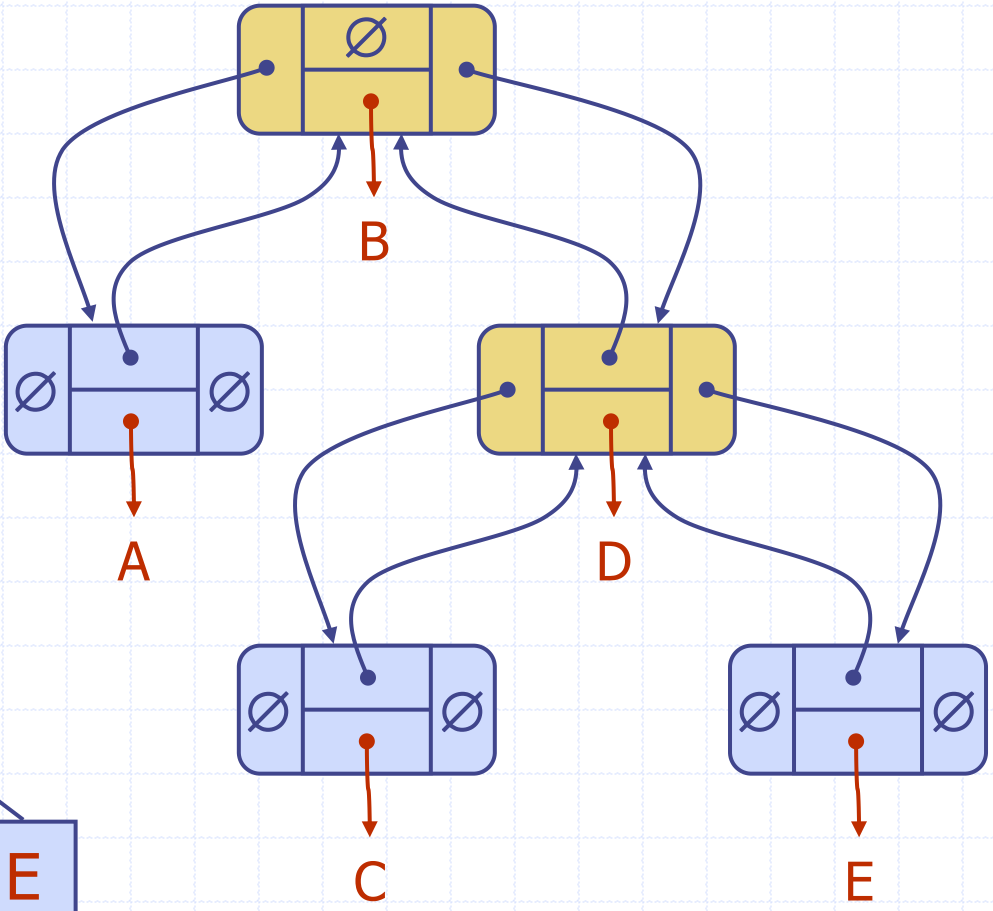
- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



Data Structure for Binary Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT





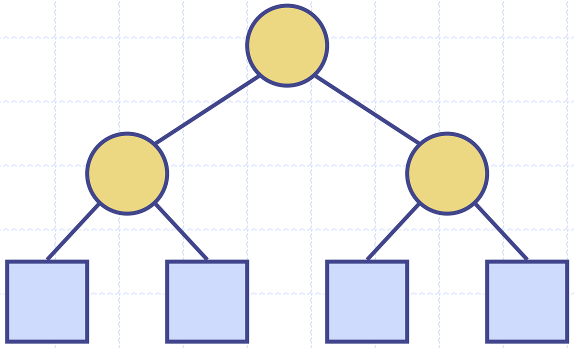
JavaScript
Position as used in Binary Trees

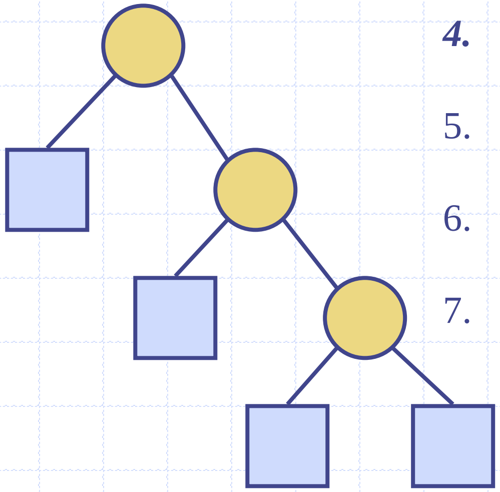
```
class TPos {
    constructor(elem, parent, left, right) {
        this._parent = parent;
        this._left = left;
        this._right = right;
        this._elem = elem;
    }
    element() {
        return this._elem;
    }
}
```

Properties of Binary Trees

- Notation

n	number of nodes
e	number of external nodes
i	number of internal nodes
h	height





- Properties:
  1. $n = i + e$
  2. $e = i + 1$
  3. $h \leq i$
  4. $e \leq 2^h$
  5. $\log_2 e \leq h$
  6. $\log_2 (i + 1) \leq h$
  7. $\log_2 (i + 1) \leq h \leq i$

Main Point

1. Each internal node of a Binary Tree has two children and each external node has no children. Thus the height, h, of a binary tree ranges as follows: $i \geq h \geq \log_2 e$, that is, $O(n) \geq h \geq O(\log_2 n)$.

   Science of Consciousness: Pure consciousness spans the full range of life, from smaller than the smallest to larger than the largest.

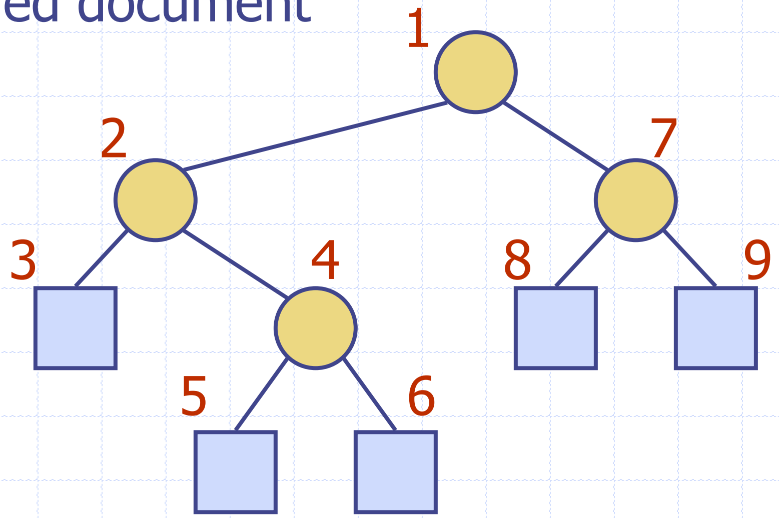
BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods for a BinaryTree:
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(p)
- Update methods would be defined by data structures implementing the BinaryTree ADT

Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

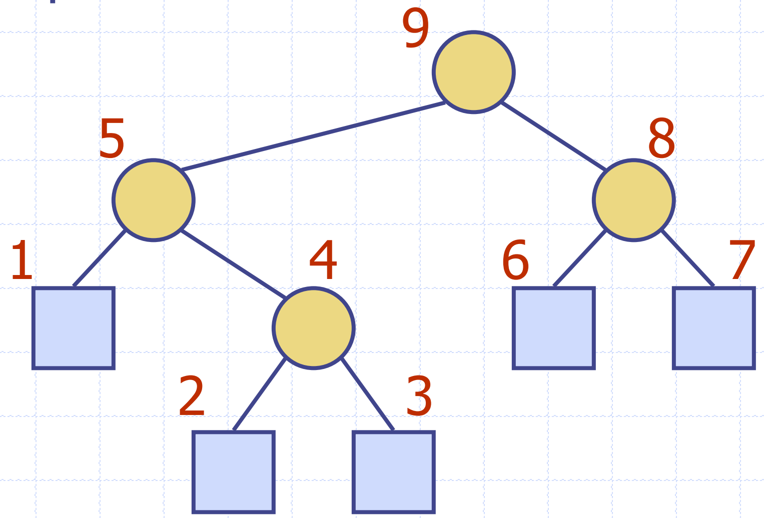
```
Algorithm preOrder(T, v)
    visit(v)
    if T.isInternal (v) then
        preOrder (T, T.leftChild (v))
        preOrder (T, T.rightChild (v))
```



Postorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: evaluate a binary expression or type check an expression and its sub-expressions

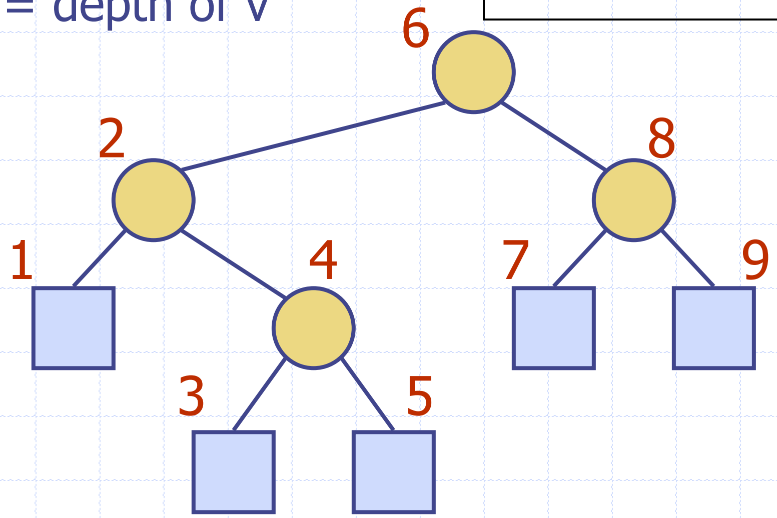
```
Algorithm postOrder(T, v)
    if T.isInternal (v) then
        postOrder (T, T.leftChild (v))
        postOrder (T, T.rightChild (v))
    visit(v)
```



Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

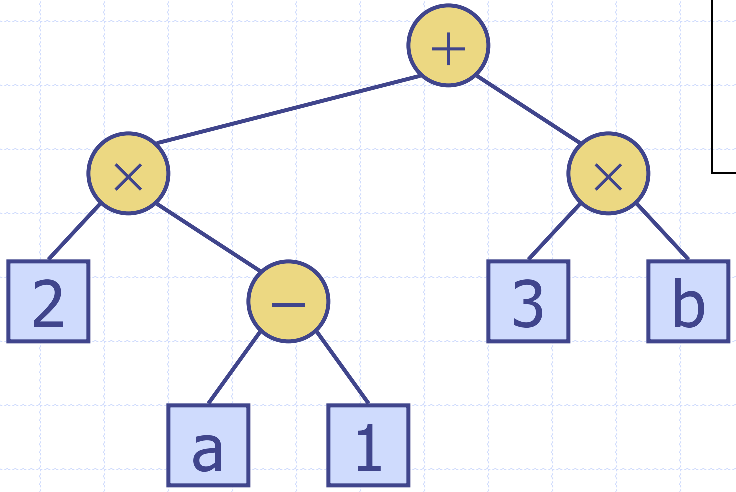
```
Algorithm inOrder(T, v)
    if T.isInternal (v) then
        inOrder (T, T.leftChild (v))
    visit(v)
    if T.isInternal (v) then
        inOrder (T, T.rightChild (v))
```



Print Arithmetic Expressions

- Specialization of an inorder traversal
  - print operand or operator when visiting node
  - print "(" before traversing left subtree
  - print ")" after traversing right subtree

```
Algorithm printExpression(T, v)
    if T.isInternal (v) then
        print("(")
        printExpression(T, T.leftChild(v))
    print(v.element ())
    if T.isInternal (v) then
        printExpression(T, T.rightChild (v))
        print (")")
```

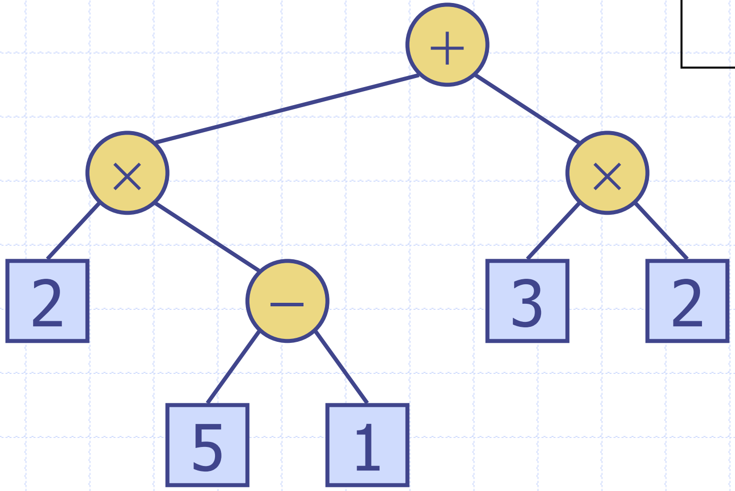


$((2 \times (a - 1)) + (3 \times b))$

Evaluate Arithmetic Expressions

- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

```
Algorithm evalExpr(T, v)
    if T.isExternal (v)  then
        return v.element ()
    else
        x ← evalExpr(T, T.leftChild(v))
        y ← evalExpr(T, T.rightChild(v))
        ◊ ← operator stored at v
        return x ◊ y
```



Exercise on Binary Trees

- Generic methods:
  - integer size()
  - boolean isEmpty()
  - objectIterator elements()
  - positionIterator positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - positionIterator children(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Update methods:
  - swapElements(p, q)
  - object replaceElement(p, o)
- Additional BinaryTree methods:
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(p)

Exercise:

- Write a method to calculate the sum of the integers in a binary tree of integers
  - Assume that an integer is stored at each internal node and **nothing** in external nodes

Algorithm sum(T)

Hint: you also need a helper function with argument Position p

Algorithm sumHelper(T, p)

Exercise on Binary Trees

- Generic methods:
  - integer size()
  - boolean isEmpty()
  - objectIterator elements()
  - positionIterator positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - positionIterator children(p)
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Update methods:
  - swapElements(p, q)
  - object replaceElement(p, o)
- Additional BinaryTree methods:
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(p)

Exercise:

- Write a method to find the maximum integer in a binary tree of integers
  - Assume that an integer is stored at each internal node and **nothing** in external nodes

Algorithm findMax(T)

Euler Tour Template (pseudo-code)

```
Algorithm EulerTour(T, v)
    if T.isExternal (v)  then
        visitExternal(T, v, result)
    else
        visitPreOrder(T, v, result)
        result[0] ← EulerTour(T, T.leftChild(v))
        visitInOrder(v, result)
        result[2] ← EulerTour(T, T.rightChild(v))
        visitPostOrder(T, v, result)

    return result[1]
```

Example of the Template Method Pattern in JavaScript

- Generic algorithm that can be specialized by redefining certain steps
- Implemented by means of an abstract JavaScript class
- Visit methods that can be redefined by subclasses
- Template method eulerTour
  - Recursively called on the left and right children
  - A result array r with elements r[0], r[1] and r[2] keeps track of the output of the recursive calls to eulerTour

```
class EulerTour {
  visitExternal(T, p, r) { }
  visitPreOrder(T, p, r) { }
  visitInOrder(T, p, r) { }
  visitPostOrder(T, p, r) { }
  eulerTour(T, p) {
    let r = new Array(3);
    if (T.isExternal(p)) { this.visitExternal(T, p, r); }
    else {
          this.visitPreOrder(T, p, r);
          r[0] = this.eulerTour(T.leftChild(p));
          this.visitInOrder(T, p, r);
          r[2] = eulerTour(T.rightChild(p));
          this.visitPostOrder(T, p, r);
    }

    return r[1];

…
```

Example of the Template Method Pattern in JavaScript

- Generic algorithm that can be specialized by redefining certain steps
- Implemented by means of an abstract JavaScript class
- Visit methods that can be redefined by subclasses
- Template method eulerTour
  - Recursively called on the left and right children
  - A result array r with elements r[0], r[2] and r[1] keeps track of the output of the recursive calls to eulerTour

```
class Sum extends EulerTour {
    visitExternal(T, p, r);
        r[1] = 0;
    }
    visitPostOrder(T, p, r);
        r[1] = r[0] + r[2] + p.element();
    }
    sum(T) {
        this. eulerTour(T, T.root());
    }
}
```

Main Point

2. The positions (nodes and elements) of a Binary Tree are visited by traversing the tree in one of three ways: pre-order, in-order, post-order. The Euler Tour allows us to traverse any given binary tree in all three ways. The Euler Tour algorithm is non-changing, but we can insert actions (change) during the traversals by overriding the default (hook) methods of the template.

   Science of Consciousness: Pure consciousness is non-changing and supports the everchanging relative creation. When we practice the TM technique, scientific research shows that mind and body are changed for the betterment of the individual and society.

Connecting the Parts of Knowledge with the Wholeness of Knowledge

1. The tree ADT is a generalization of the linked-list in which each tree node can have any number of children instead of just one. A proper binary tree is a special case of the generic tree ADT in which each node has either 0 or 2 children (a left and right child).

2. Any ADT will have a variety of implementations of its operations with varying efficiencies, e.g., the binary tree can be implemented as either a set of recursively defined nodes or as an array of elements.

3. **<u>Transcendental Consciousness</u>** is pure intelligence, the abstract substance out of which the universe is made.
4. **<u>Impulses within Transcendental Consciousness</u>**: Within this field, the laws of nature continuously organize and govern all activities and processes in creation.
5. **<u>Wholeness moving within itself</u>** : In Unity Consciousness, awareness is awake to its own value, the full value of the intelligence of nature. One's consciousness supports the knowledge that outer is the expression of inner, creation is the play and display of the Self.