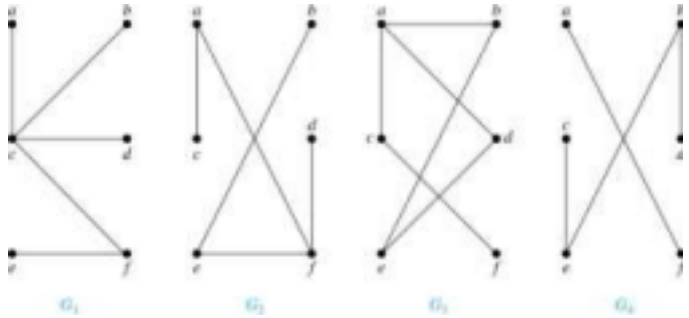
Discrete Structures Spring 2024 – Week 12

Trees

Trees

Definition: A *tree* is a connected undirected graph with no simple circuits.

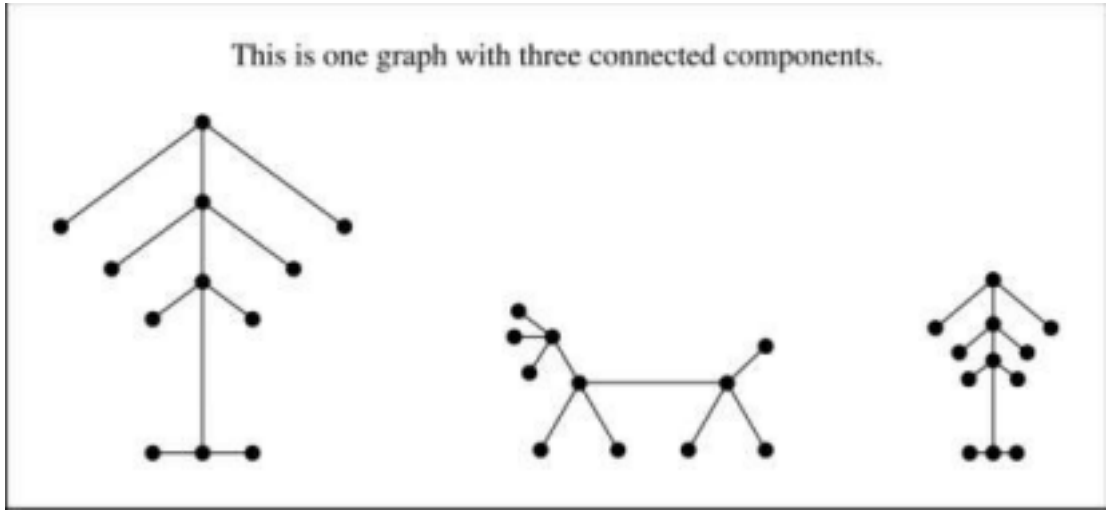
Example: Which of these graphs are trees?



Solution: G_1 and G_2 are trees - both are connected and have no simple circuits. Because e, b, a, d, e is a simple circuit, G_3 is not a tree. G_4 is not a tree because it is not connected.

Trees

Definition: A *forest* is a graph that has no simple circuit, but is not connected. Each of the connected components in a forest is a tree.

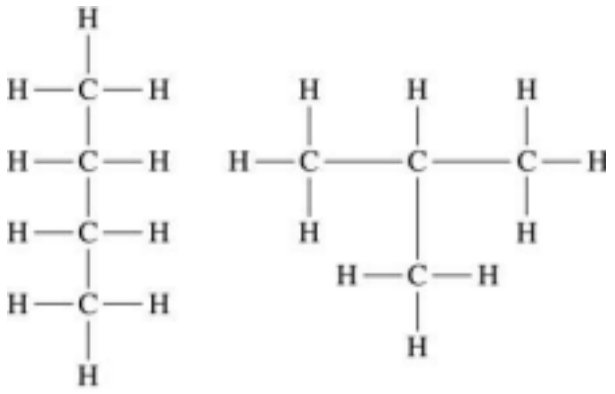


Trees as Models

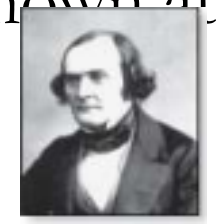
- Trees are used as models in

computer science, chemistry, geology, botany, psychology, and many other areas.

• Trees were introduced by the mathematician Cayley in 1857 in his work counting the number of isomers of saturated hydrocarbons. The two isomers of butane are shown at the right.



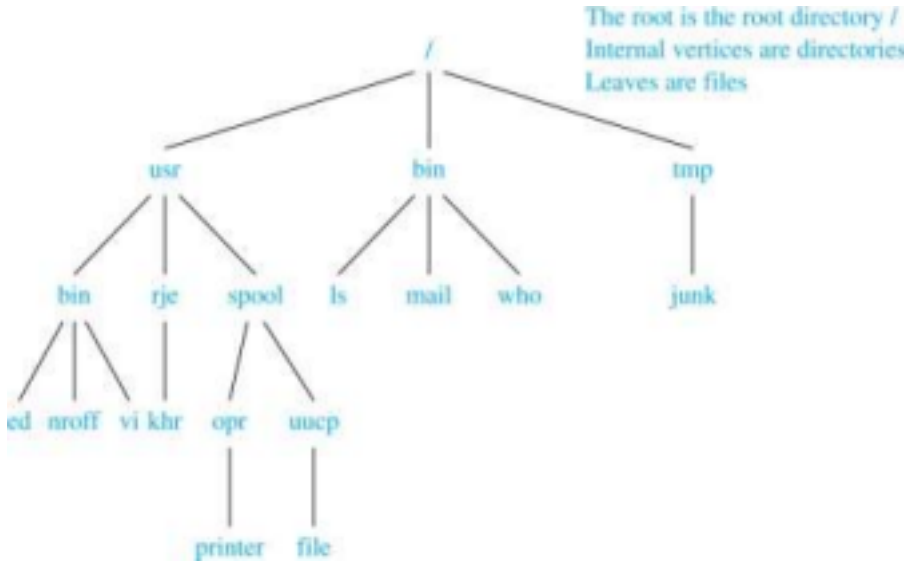
Butane Isobutane



Arthur Cayley (1821-1895)

Trees as Models

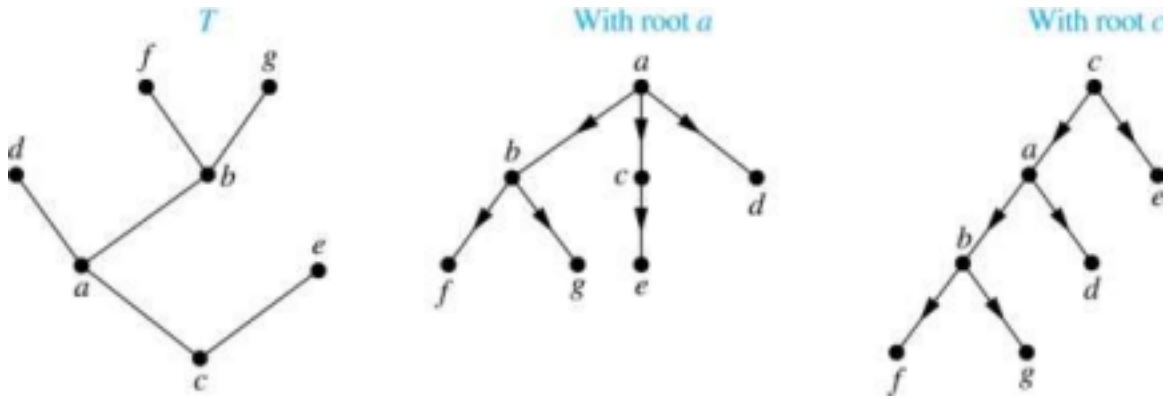
- The organization of a computer file system into directories, subdirectories, and files is naturally represented as a tree.
- Trees are used to represent the structure of organizations.



Rooted Trees

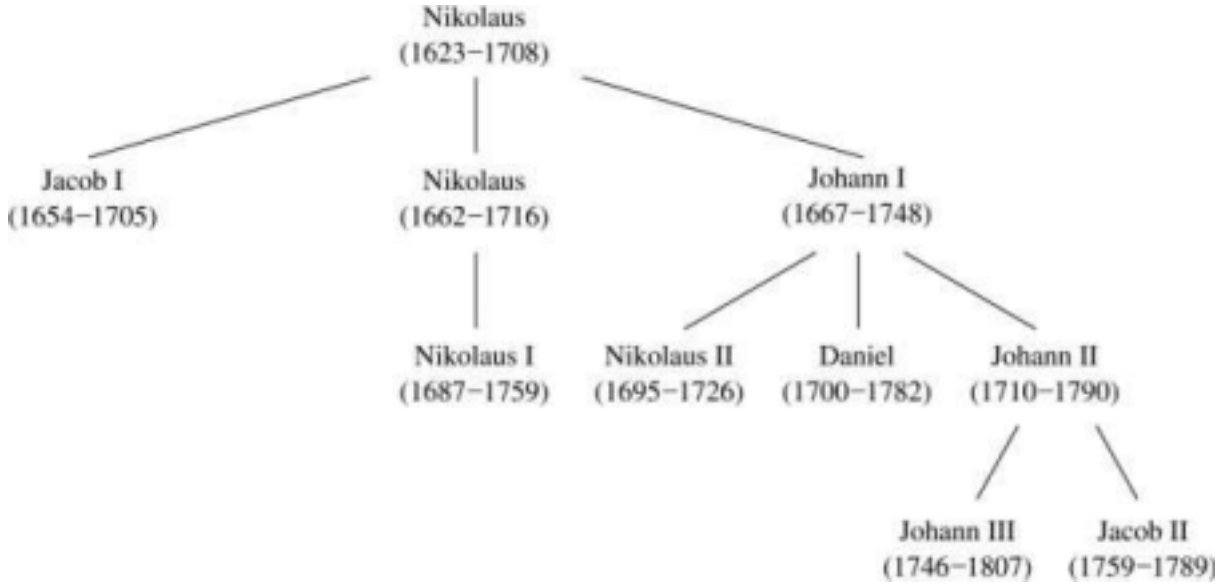
Definition: A *rooted tree* is a tree in which one vertex has been designated as the *root* and every edge is directed away from the root.

An unrooted tree is converted into different rooted trees when different vertices are chosen as the root.



Rooted Tree Terminology

- Terminology for rooted trees is a mix from botany and genealogy (such as this family tree of the Bernoulli family of mathematicians).



Rooted Tree Terminology

- If v is a vertex of a rooted tree other than the root, the *parent* of v is the unique vertex u such that there is a directed edge from u to v. When u is a parent of v, v is called a *child* of u. Vertices with the same parent are called *siblings*.
- The *ancestors* of a vertex are the vertices in the path from the root to

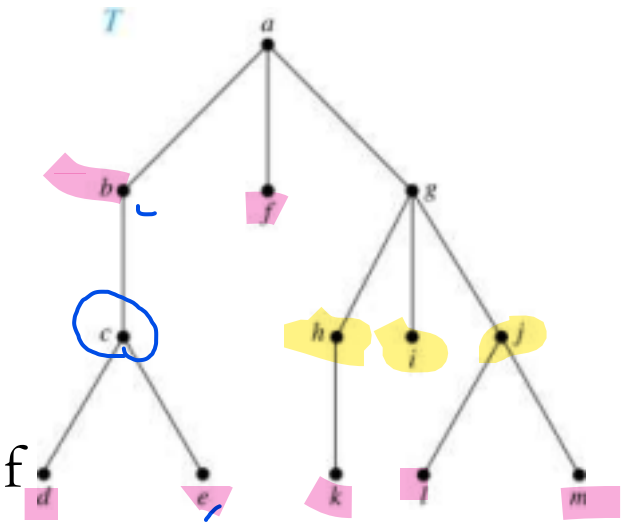
this vertex, excluding the vertex itself and including the root. The *descendants* of a vertex v are those vertices that have v as an ancestor.

- A vertex of a rooted tree with no children is called a *leaf*. Vertices that have children are called *internal vertices*.
- If a is a vertex in a tree, the *subtree* with a as its root is the subgraph of the tree consisting of a and its descendants and all edges incident to these descendants.

Terminology for Rooted Trees

Example: In the rooted tree T (with root a):

(i) Find the parent of c, the children of g, the siblings of h, the ancestors of e, and the descendants of



b.

(ii) Find all internal vertices and all leaves.

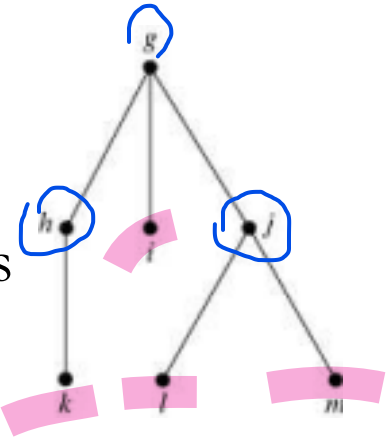
(iii) What is the subtree rooted at *G*?

Solution:

(i) The parent of *c* is *b*. The children of *g* are *h*, *i*, and *j*. The siblings of *h* are *i* and *j*. The ancestors of *e* are c, *b*, and *a*. The descendants of *b* are *c*, *d*, and *e*.

(ii) The internal vertices are *a*, *b*, *c*, *g*, *h*, and *j*. The leaves are *d*, *e*, *f*, *i*, *k*, *l*, and *m*.

(iii) We display the subtree rooted at *g*.

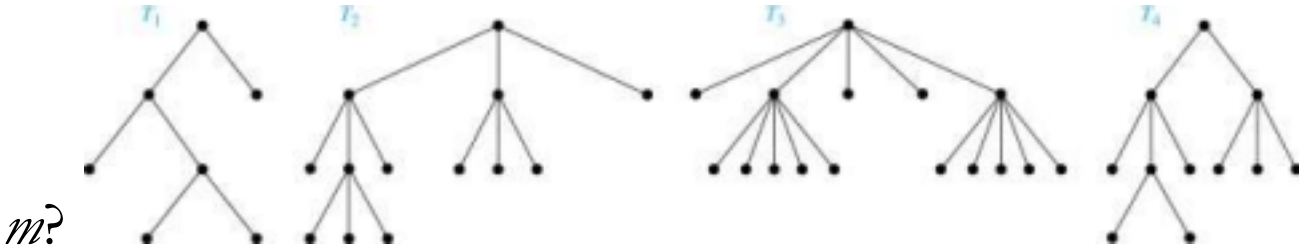


m-ary Rooted Trees

Definition: A rooted tree is called an *m-ary tree* if every internal vertex has no more than *m* children. The tree is called a *full m-ary tree* if every internal vertex has exactly *m* children.

An m-ary tree with $m = 2$ is called a *binary* tree.

Example: Are the following rooted trees full m-ary trees for some positive integer m?



Solution: T_1 is a full binary tree because each of its internal vertices has two children. T_2 is a full 3-ary tree because each of its internal vertices has three children. In T_3 each internal vertex has five children, so T_3 is a full 5-ary tree. T_4 is not a full m-ary tree for any m because some of its internal vertices have two children and others have three children.

Ordered Rooted Trees

Ordered Rooted Trees

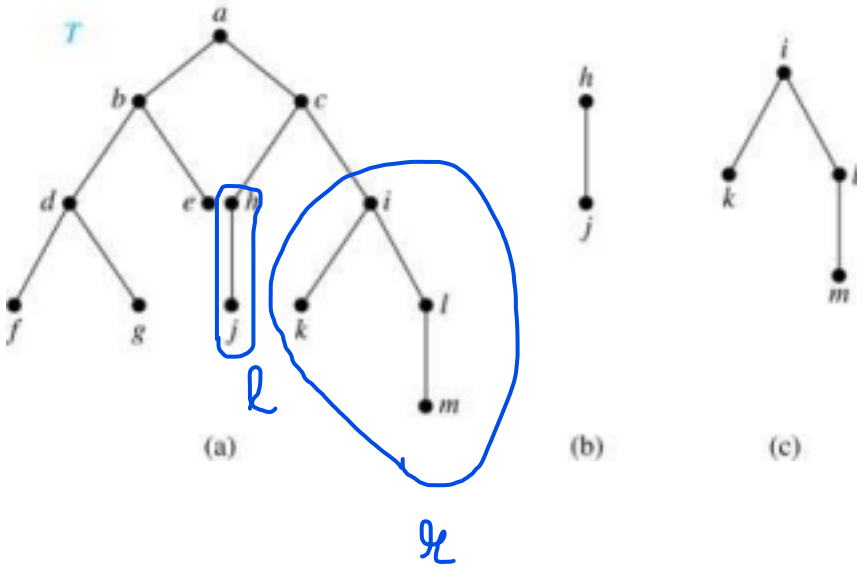
Definition: An *ordered rooted tree* is a rooted tree where the children of each internal vertex are ordered. We draw ordered rooted trees so that the children of each internal vertex are shown in order from left to right.

Definition: A *binary tree* is an ordered rooted where where each internal vertex has at most two children. If an internal vertex of a binary tree has two children, the first is called the *left child* and the second the *right child*. The tree rooted at the left child of a vertex is called the *left subtree* of this vertex, and the tree rooted at the right child of a vertex is called the *right subtree* of this vertex.

Example: Consider the binary tree T.

(*i*) What are the left and right children of d?

(*ii*) What are the left and right subtrees of c?



Solution:

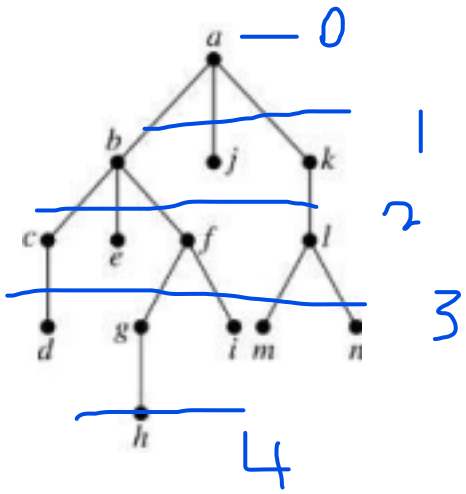
(*i*) The left child of d is f and the right child is g.

(*ii*) The left and right subtrees of c are displayed in (b) and (c).

Level of vertices and height of trees

- When working with trees, we often want to have rooted trees where the subtrees at each vertex contain paths of approximately the same length.
- To make this idea precise we need some definitions:
- The *level* of a vertex *v* in a rooted tree is the length of the unique path from the root to this vertex.
- The *height* of a rooted tree is the maximum of the levels of the vertices.

Level of vertices and height of trees



Example:

(*i*) Find the level of each vertex in the tree to the right.

(*ii*) What is the height of the tree?

Solution:

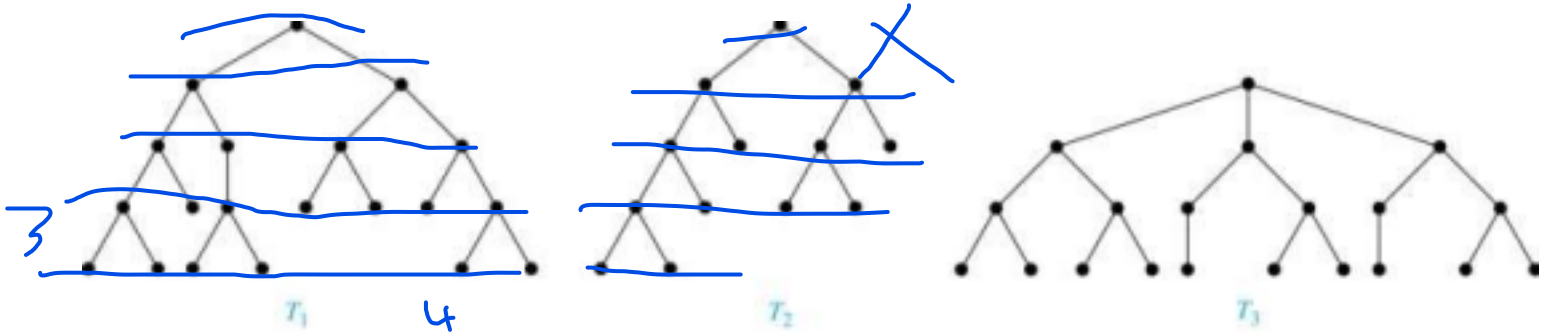
(*i*) The root *a* is at level 0. Vertices *b*, *j*, and *k* are at level 1 Vertices *c*, *e*, *f*, and *l* are at level 2. Vertices *d*, *g*, *i*, *m*, and *n* are at level 3. Vertex *h* is at level 4.

(*ii*) The height is 4, since 4 is the largest level of any vertex.

Balanced m-Ary Trees

Definition: A rooted m-ary tree of height h is *balanced* if all leaves are at levels h or $h - 1$.

Example: Which of the rooted trees shown below is balanced?



Solution: $T_1$and $T_3$are balanced, but $T_2$is not because it has leaves at levels 2, 3, and 4.

Tree Traversal

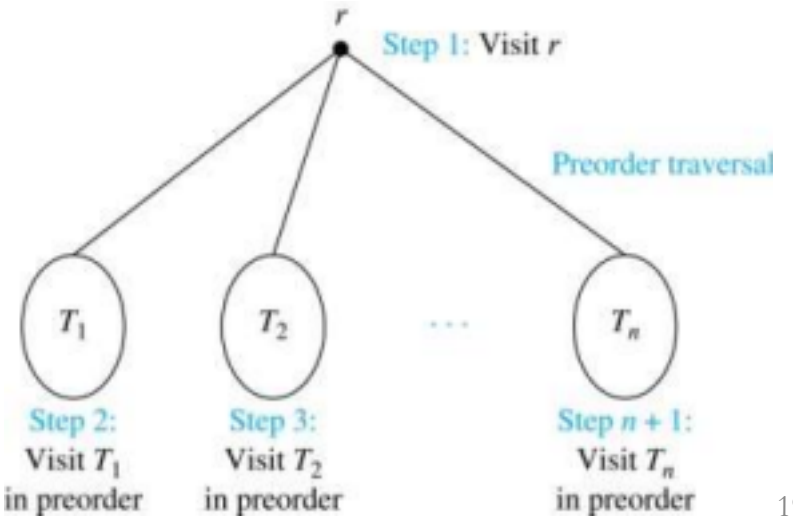
Tree Traversal

- Procedures for systematically visiting every vertex of an ordered tree are called *traversals*.
- The three most commonly used *traversals* are *preorder traversal*, *inorder traversal*, and *postorder traversal*.

Preorder Traversal

Definition: Let T be an ordered rooted tree with root r. If T consists only of r, then r is the *preorder traversal* of T. Otherwise, suppose that T_1, T_2, …, $T_n$are the subtrees of r from left to right in T. The preorder traversal begins by visiting r, and continues by traversing T_1in preorder, then T_2in preorder,

and so on, until T_n is traversed in preorder.

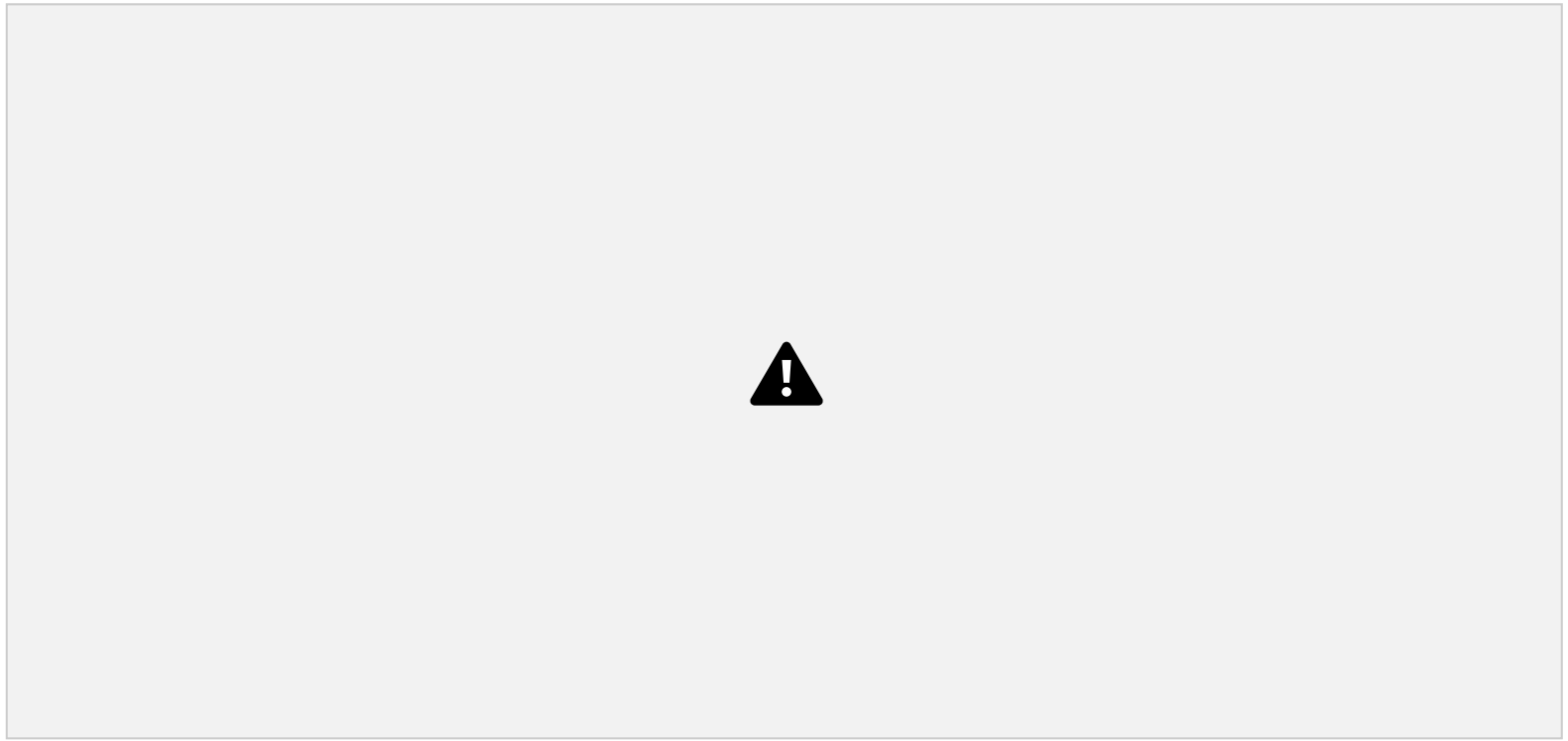


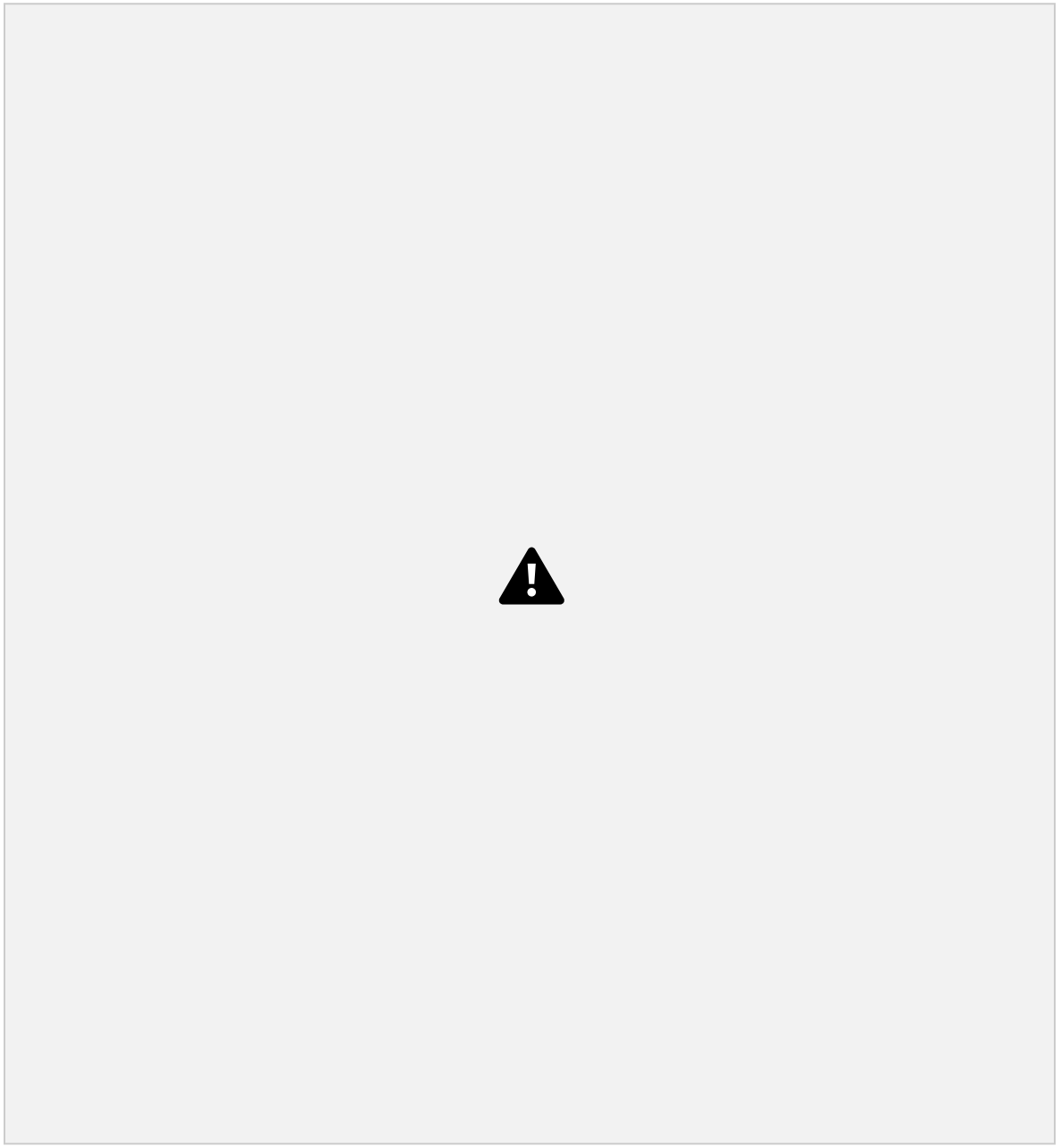
Preorder Traversal (*continued*)

- **procedure** *preorder* (*T*: ordered rooted tree)
- *r* := root of *T*
- list *r*
- **for** each child *c* of *r* from left to right

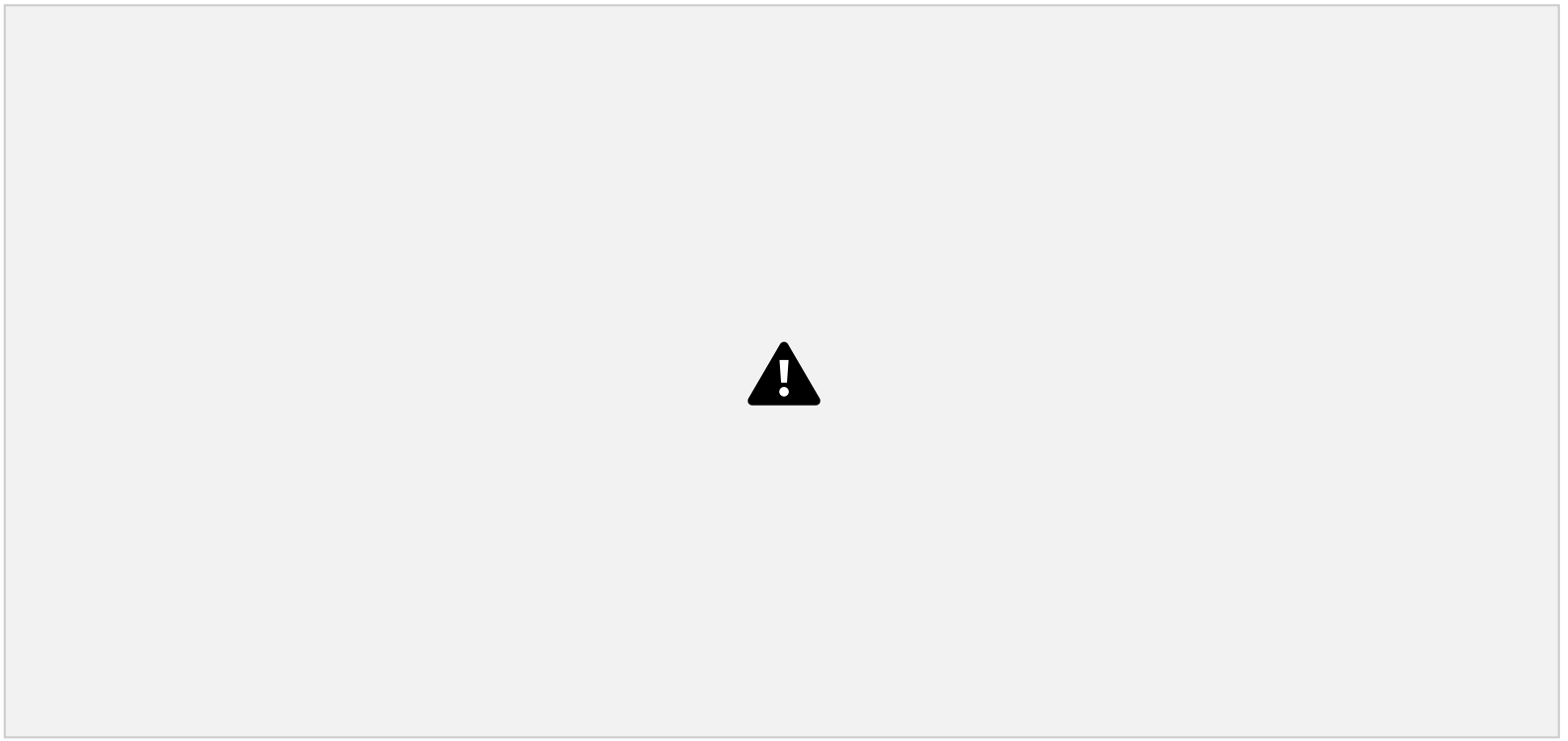
- $T(c)$:= subtree with c as root
- *preorder*$(T(c))$

Preorder Traversal (*continued*)





Preorder Traversal (*continued*)



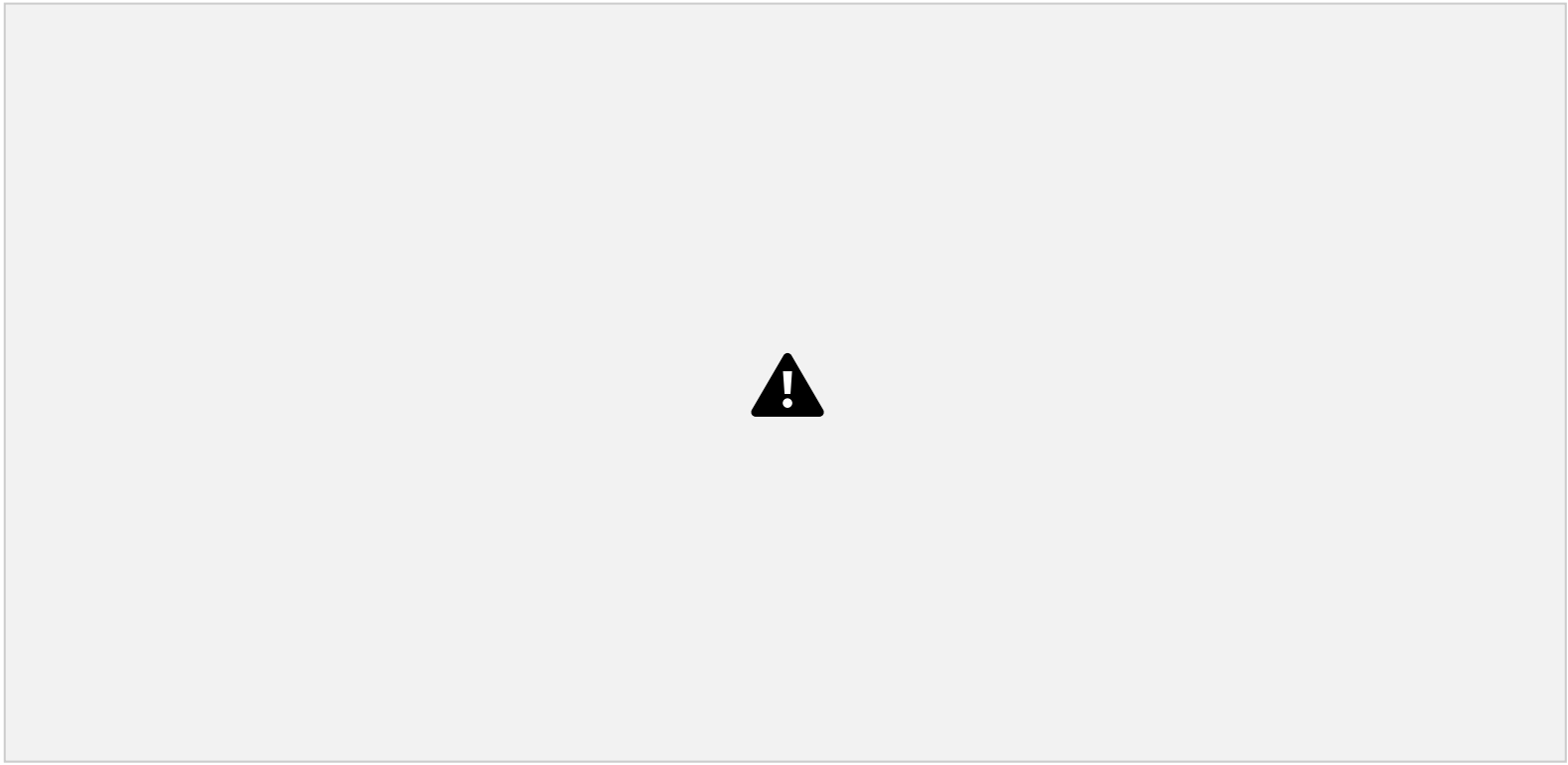
Inorder Traversal

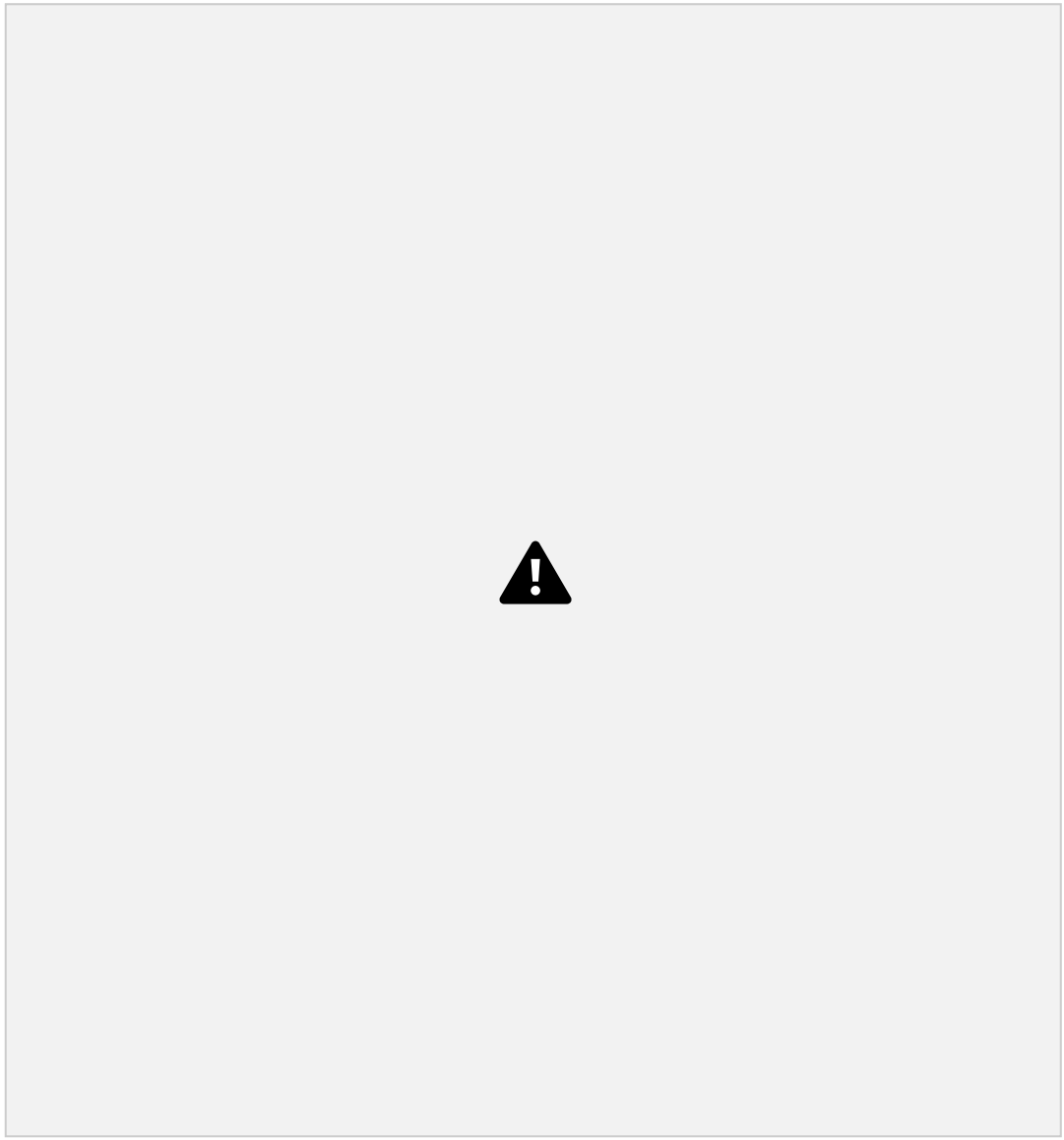
Definition: Let T be an ordered rooted tree with root r. If T consists only of r, then r is the *inorder traversal* of T. Otherwise, suppose that T_1, T_2, …, $T_n$are the subtrees of r from left to right in T. The inorder traversal begins by traversing T_1in inorder, then visiting r, and continues by traversing T_2in inorder, and so on, until $T_n$is traversed in inorder.

Inorder Traversal

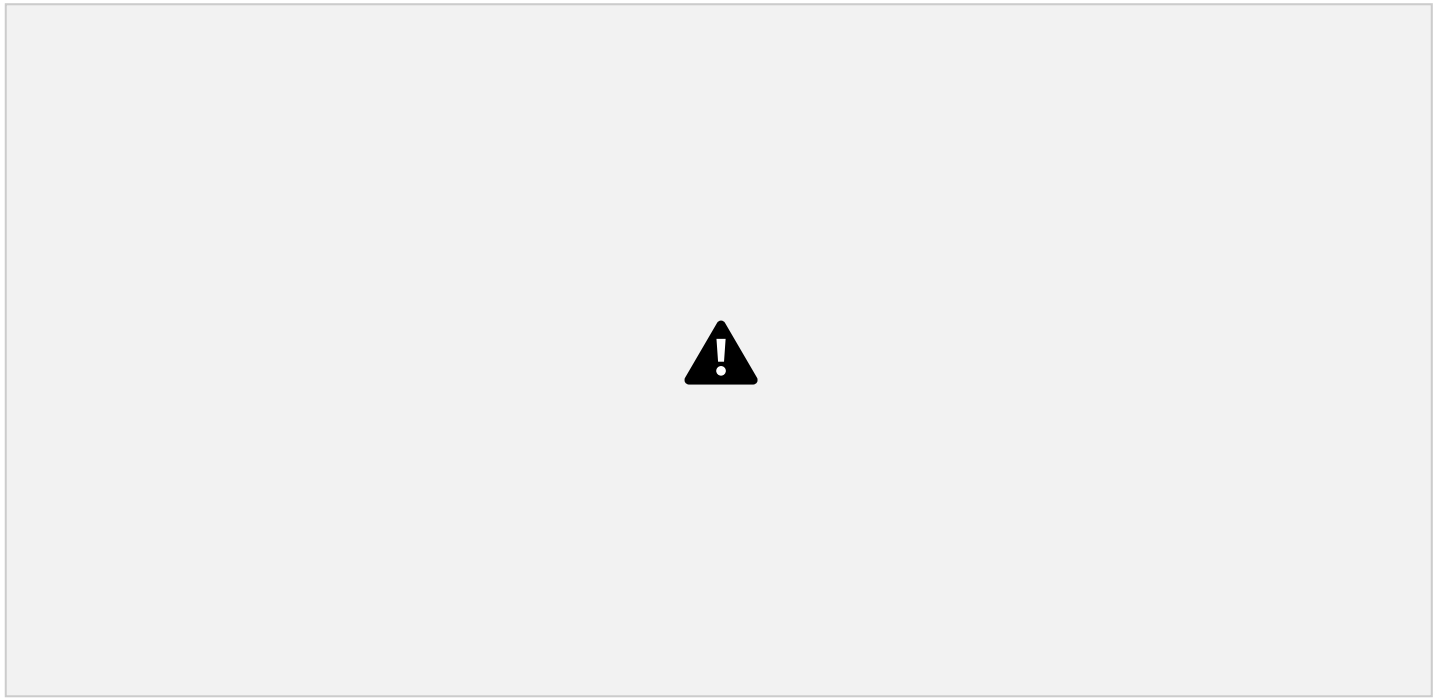
- **procedure** *inorder* (*T*: ordered rooted tree)
- *r* := root of *T*
- **if** *r* is a leaf **then** list *r*
- **else**
- *l* := first child of *r* from left to right
- *T*(*l*) := subtree with *l* as its root
- *inorder*(*T*(*l*))
- list(*r*)
- **for** each child *c* of *r* from left to right
- *T*(*c*) := subtree with *c* as root
- *inorder*(*T*(*c*))

Inorder Traversal (*continued*)



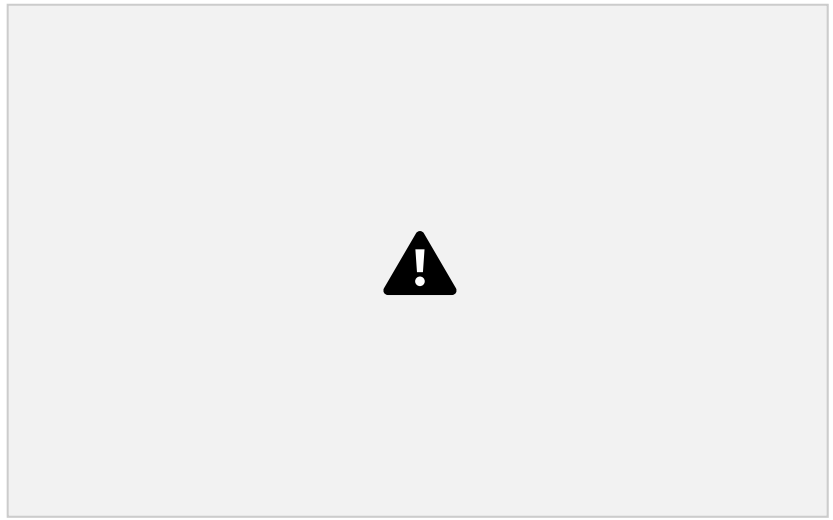


Inorder Traversal (*continued*)



Postorder Traversal

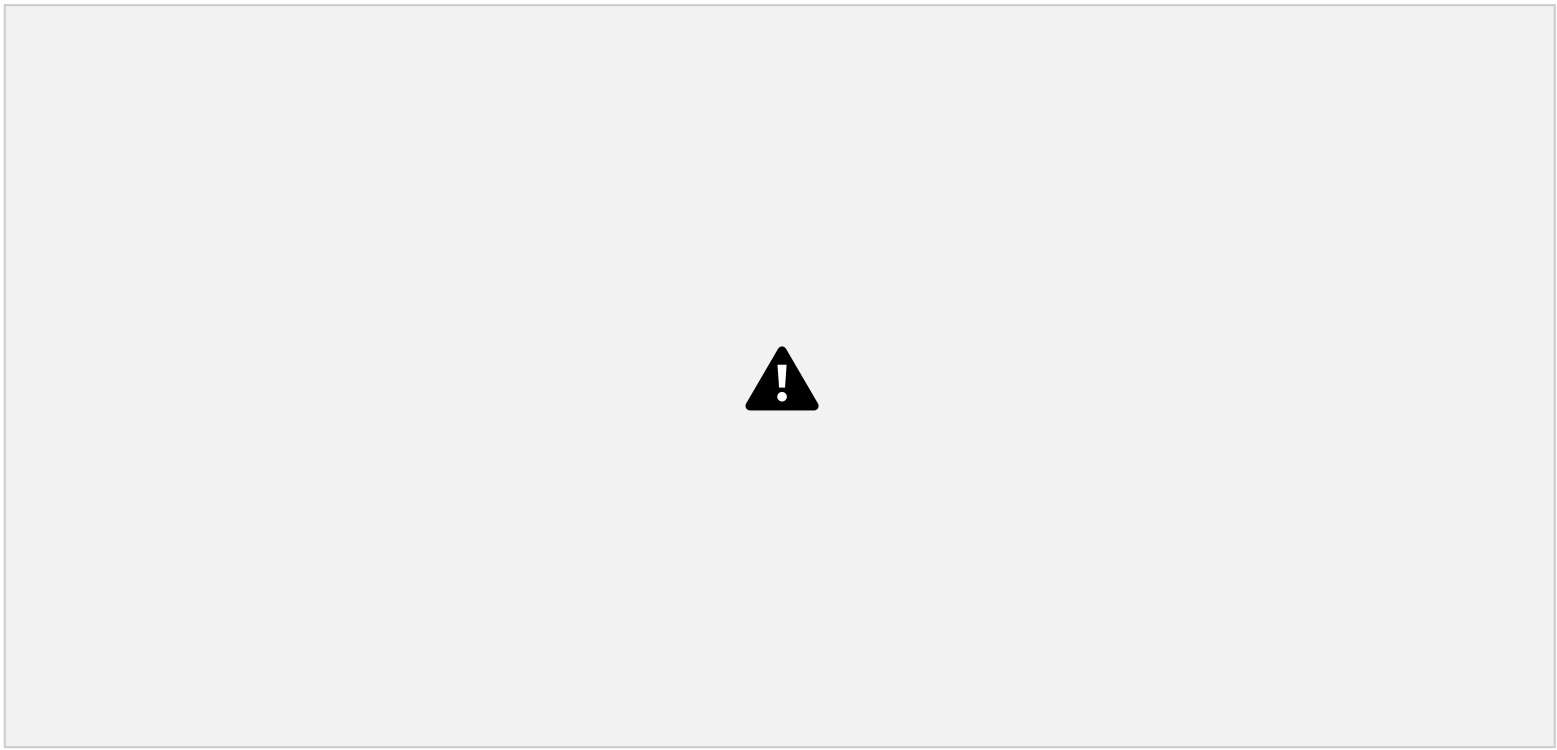
Definition: Let T be an ordered rooted tree with root r. If T consists only of r, then r is the *postorder traversal* of T. Otherwise, suppose that T_1, T_2, …, $T_n$are the subtrees of r from left to right in T. The postorder traversal begins by traversing T_1in postorder, then T_2in postorder, and so on, after $T_n$is traversed in postorder, r is visited.

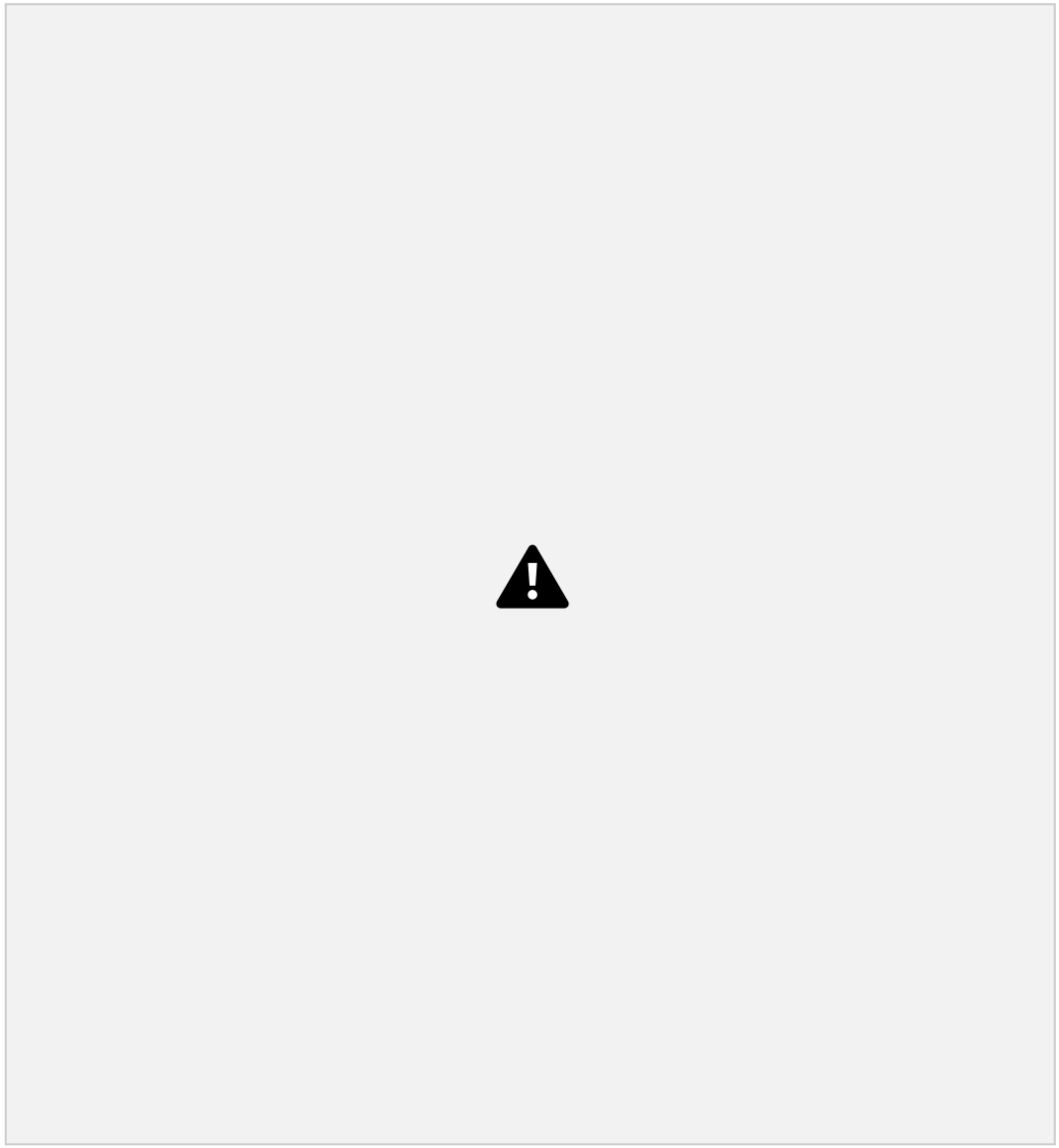


Postorder Traversal

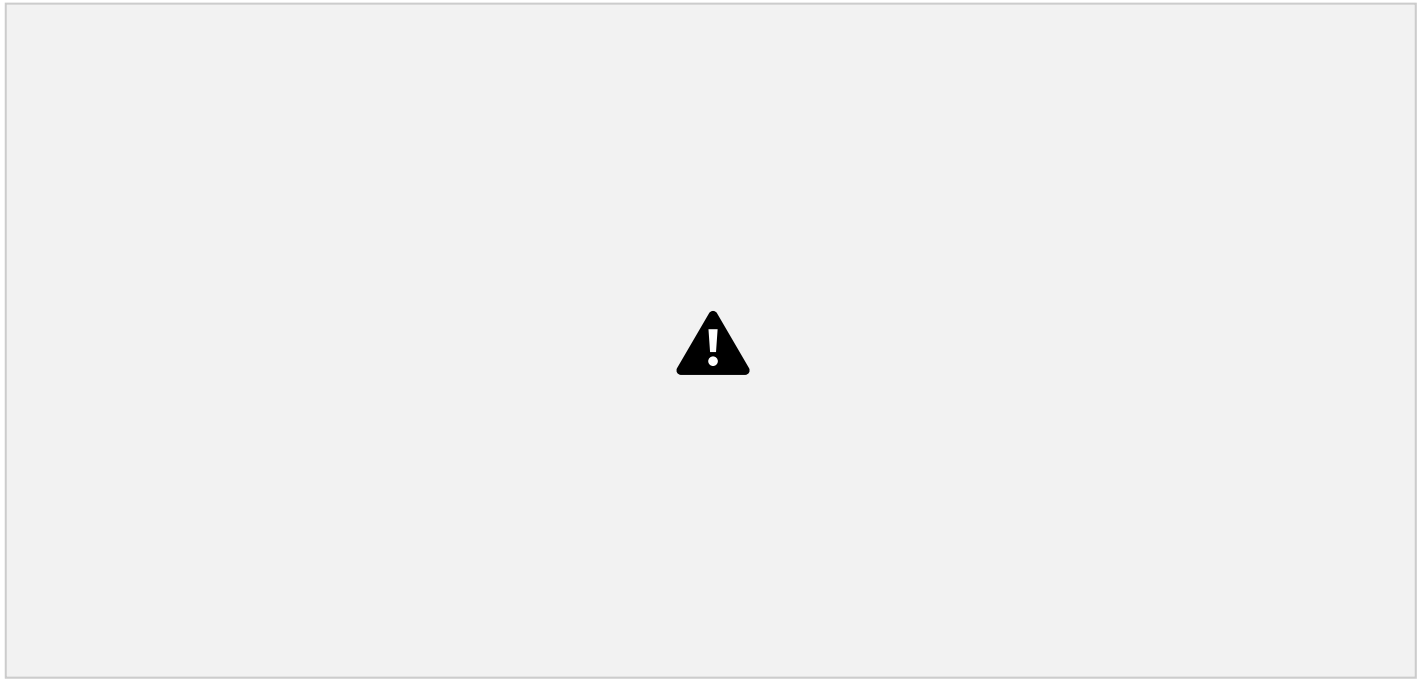
- **procedure** *postordered* (*T*: ordered rooted tree)
- *r* := root of *T*
- **for** each child *c* of *r* from left to right
- *T*(*c*) := subtree with *c* as root
- postorder(*T*(*c*))
- list *r*

Postorder Traversal (*continued*)

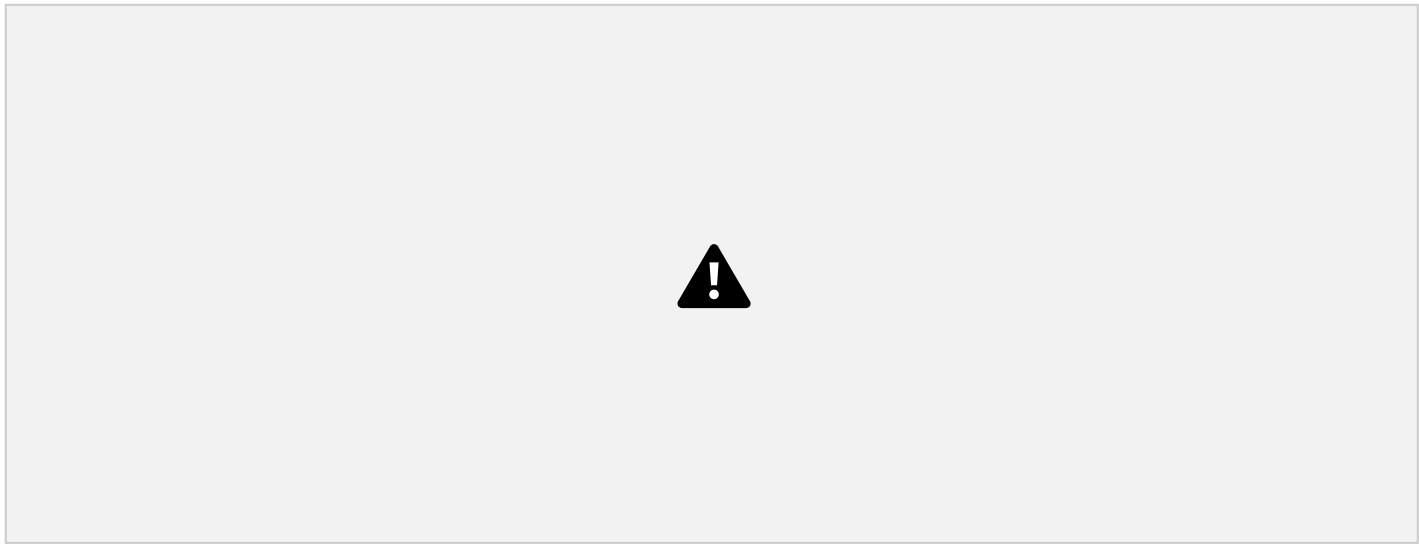




Postorder Traversal (*continued*)



Exercise!



Solution!