Few Announcements

- Two OpenMP programs to be uploaded to the course website.
- More practice to be aimed in the laboratory session next week.

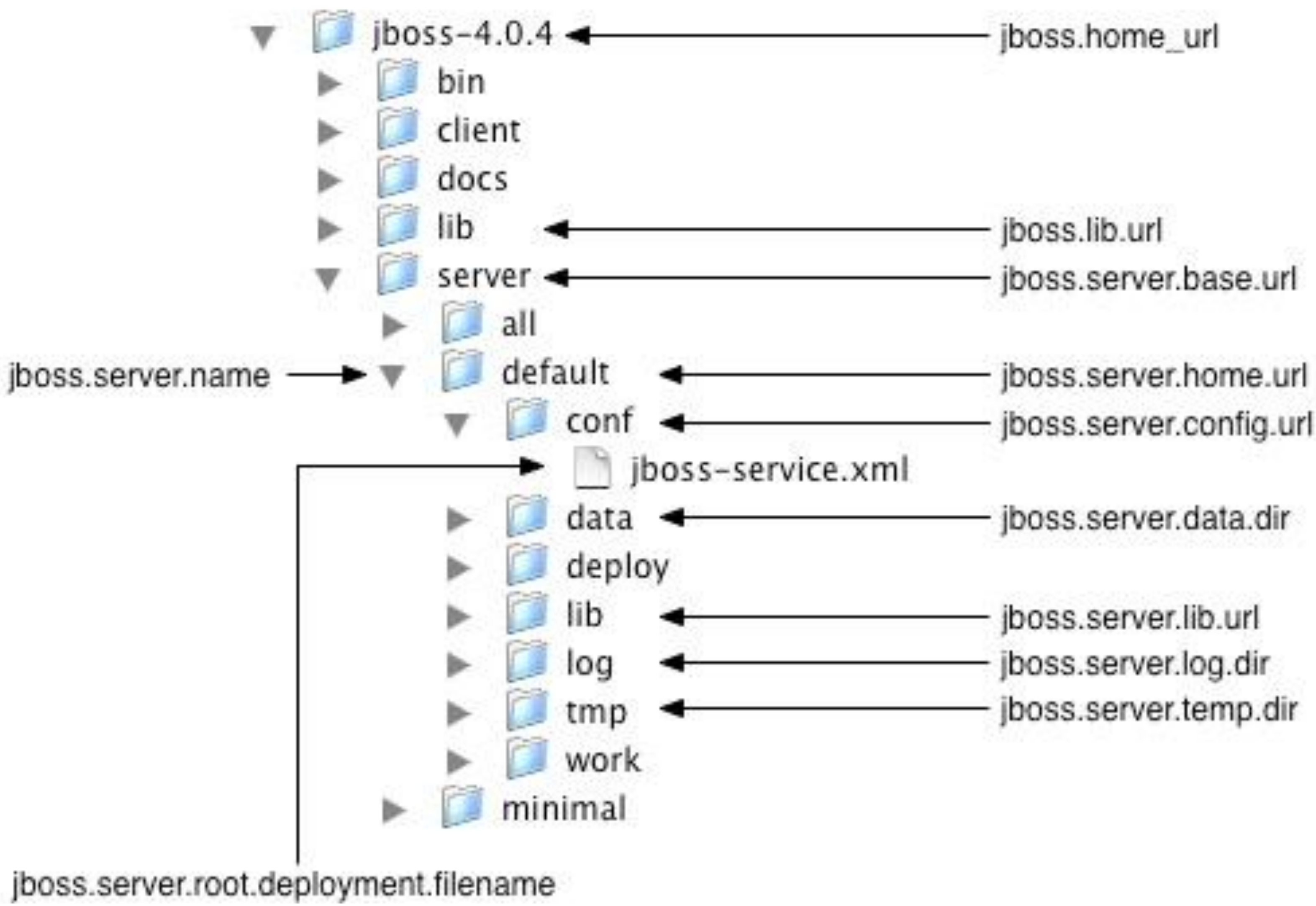
Further Data Structures

- The progress so far
  - Saw some fundamental operations as well as advanced operations on arrays, stacks, and queues
  - Saw a dynamic data structure, the linked list, and its applications.
  - Saw the hash table so that insert/delete/find can be supported efficiently.
- This topic:
  - Study data structures for hierarchical data
  - Operations on such data.
  - Leading to efficient insert/delete/find.

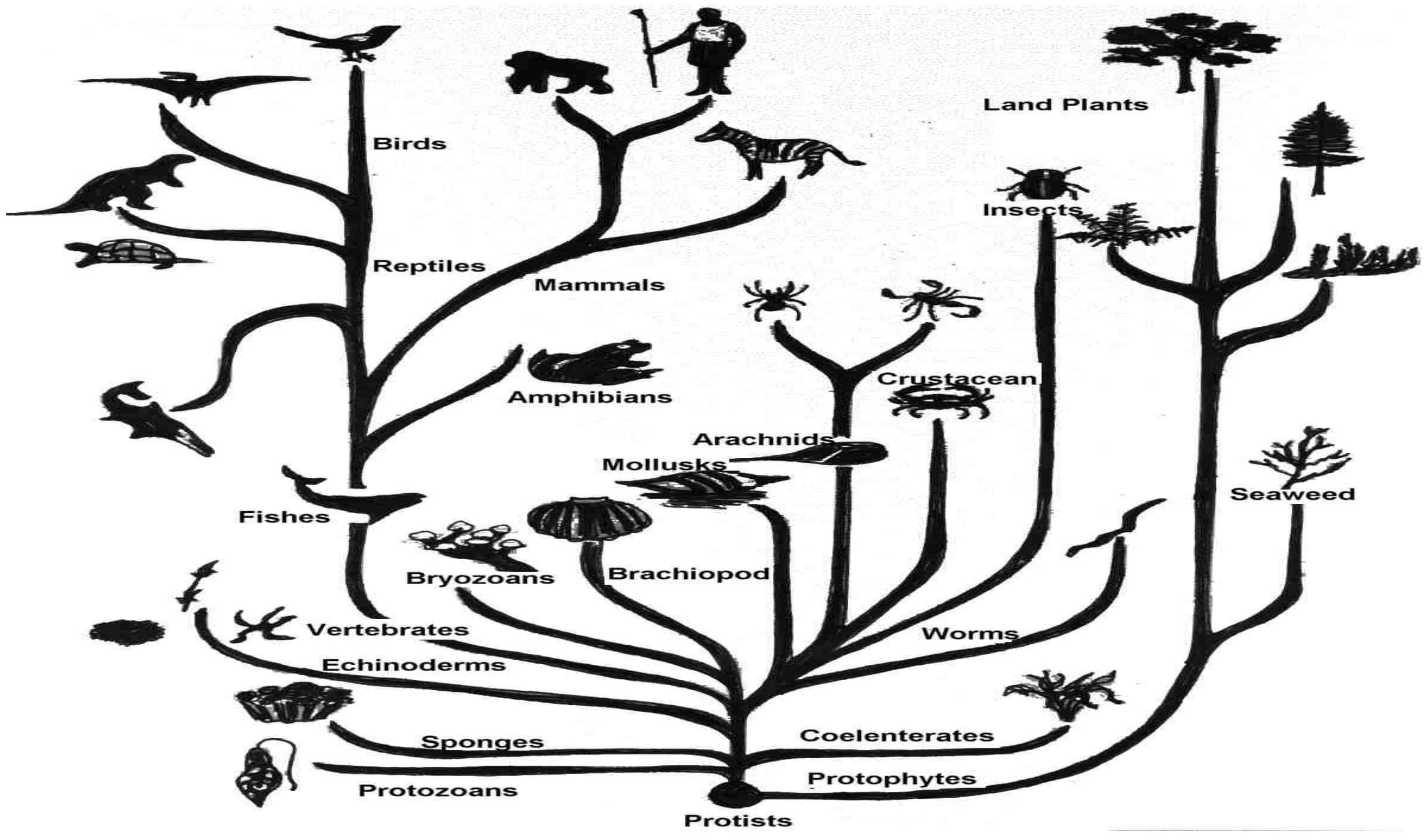
Motivation

- Consider your home directory.
- /home/user is a directory, which can contain sub-directories such as work/, misc/, songs/, and the like.
- Each of these sub-directories can contain further sub-directories such as ds/, maths/, and the like.
- An extended hierarchy is possible, until we reach a file.

Example


jboss-4.0.4
bin
client
docs
lib
server
all
default
conf
jboss-service.xml
data
deploy
lib
log
tmp
work
minimal
jboss.home_url
jboss.lib.url
jboss.server.base.url
jboss.server.name
jboss.server.home.url
jboss.server.config.url
jboss.server.data.dir
jboss.server.lib.url
jboss.server.log.dir
jboss.server.temp.dir
jboss.server.root.deployment.filename

Evolution of Species



Motivation

- Consider another example. The table of contents of a book.
- A book has chapters.
- A chapter has sections
- A section has sub-sections.
- A sub-section has sub-subsections,
- Till some point.

Motivation

- In both of the above examples, there is a natural hierarchy of data.
  - In the first example, a (sub)directory can have one or more sub-directories.
- Similarly, there are several setting where there is a natural hierarchy among data items.
  - Family trees with parents, ancestors, siblings, cousins,...
  - Hierarchy in an organization with CEO/CTO/Managers/...

Motivation

- What kind of questions arise on such hierarchical data?
  - Find the number of levels in the hierarchy between two data items?
  - Print all the data items according to their level in the hierarchy, for example like a table of contents of a book
  - Where from two members of the hierarchy trace their first common member in the hierarchy. Put differently, in the evolution process, when did two man and amphibians start to branch out?

Motivation

- As a data structure question
  - How to formalize the above notions? Plus,
  - How can more members be added to the hierarchy?
  - How can existing data items be deleted from the hierarchy?

A New Data Structure

- This week we will propose a new data structure that can handle hierarchical data.
- Study several applications of the data structure including those to:
  - expression verification and evaluation
  - searching

The Tree Data Structure

- Our new data structure will be called a tree.
- Defined as follows.
  - A tree is a collection of nodes.
  - An empty collection of nodes is a tree.
  - Otherwise a tree consists of a distinguished node r, called the root, and 0 or more non-empty (sub)trees T_1, T_2, · · · , T_k each of whose roots r_1, r_2, ..., r_k are connected by a directed edge from r.
  - r is also called as the parent of the the nodes r_1, r_2, ..., r_k.

Basic Observations

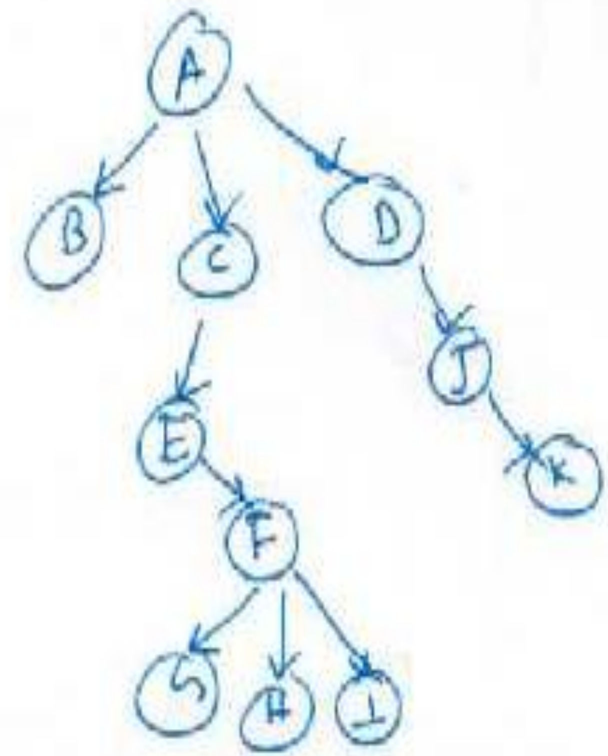
- A tree on n nodes always has n-1 edges.
- Why?

Basic Observations

- A tree on n nodes always has n-1 edges.
- Why?
  - One parent for every one, except the root.
- Before going in to how a tree can be represented, let us know more about the tree.

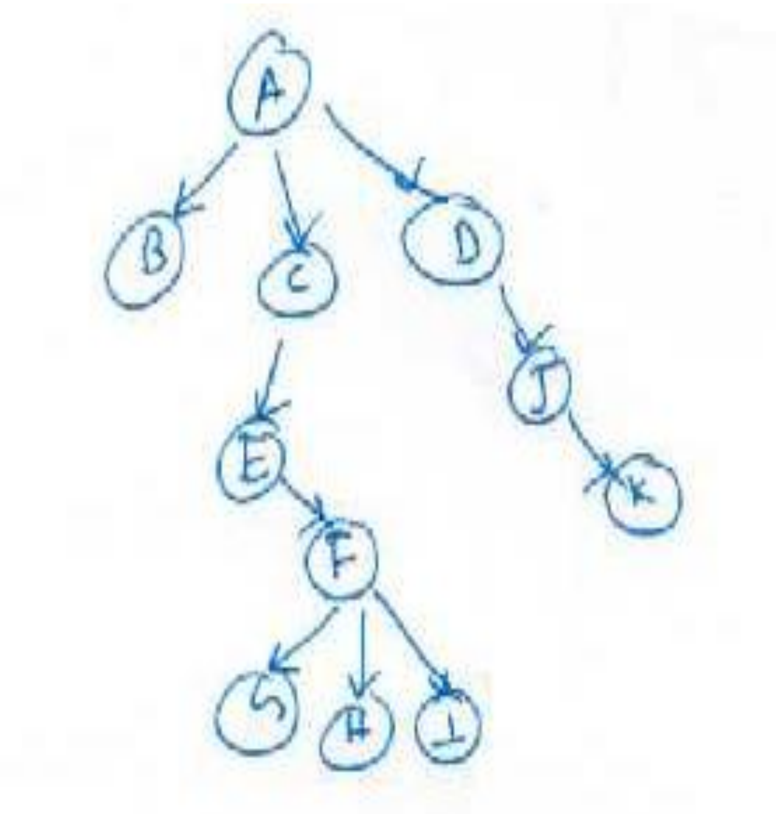
An Example

- Consider the tree shown to the right.
- The node A is the root of the tree.
- It has three subtrees whose roots are B, C, and D.
- Node C has one subtree with node E as the root.



An Example

- Nodes with the same parent are called as siblings.
- In the figure, G, H, and I are siblings.
- Nodes with no children are called leaf nodes or pendant nodes.
  - In the figure, B and K are leaf nodes.



A Few More Terms : Height, Level, and Path

- A path from a node u to a node v is a sequence of nodes $u=u_0, u_1, u_2, ..., u_k = v$ such that u_i is the parent of u_{i+1}, $i > 0$.
  - The path is said to have a length of k-1, the number of edges in the path.
  - A path from a node to itself has a length of 0.
- Example: A path from node C to F in our earlier tree is C->E->F.
- Observation: In any tree there is exactly one path from the root to any other node.

Depth

- Given a tree T, let the root node be said to be at a depth of 0.
- The depth of any other node u in T is defined as the length of the path from the root to u.
- Example: Depth of node G = 4.
- Alternatively, let the depth of the root be set to 0 and the depth of a node is one more than the depth of its parent.

Height

- Another notion defined for trees is the height.
- The height of a leaf node is set to 0. The height of a node is one plus the maximum height of its children.
- The height of a tree is defined as the height of the root.
- Example: Height of node C = 3.

Ancestors and Descendants

- Recall the parent-child relationship between nodes.
- Alike parent-children relationship, we can also define ancestor-descendant relationship as follows.
- In the path from node u to v, u is an ancestor of v and v is a descendant of u.
- If $u \neq v$, then u (v) is called a proper ancestor (descendant) respectively.

Implementing Trees

- Briefly, we also mention how to implement the tree data structure.
- The following node declaration as a structure works.

```
struct node
{
        int data;
        node *children;
}
```

Applications

- Can use this to store the earlier mentioned examples.
- Need more tools to perform the required operations.
- We'll study them via a slight specialization.

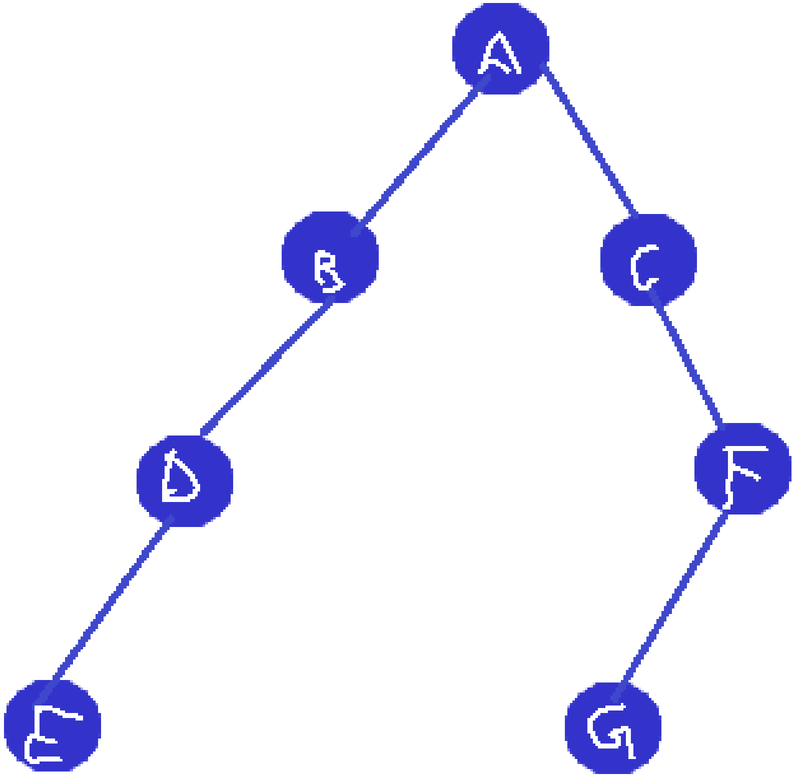
Binary Trees

- A special class of the general trees.
- Restrict each node to have at most two children.
  - These two children are called the left and the right child of the node.
  - Easy to implement and program.
  - Still, several applications.

An Example

- Figure shows a binary tree rooted at A.
- All notions such as
  - height
  - depth
  - parent/child
  - ancestor/descendant

are applicable.



Our First Operation

- To print the nodes in a (binary) tree
- This is also called as a traversal.
- Need a systematic approach
  - ensure that every node is indeed printed
  - and printed only once.

Tree Traversal

- Several methods possible. Attempt a categorization.
- Consider a tree with a root D and L, R being its left and right sub-trees respectively.
- Should we intersperse elements of L and R during the traversal?
  - OK – one kind of traversal.
  - No. -- One kind of traversal.
  - Let us study the latter first.

Tree Traversal

- When items in L and R should not be interspersed, there are six ways to traverse the tree.
  - D L R
  - D R L
  - R D L
  - R L D
  - L D R
  - L R D

Tree Traversal

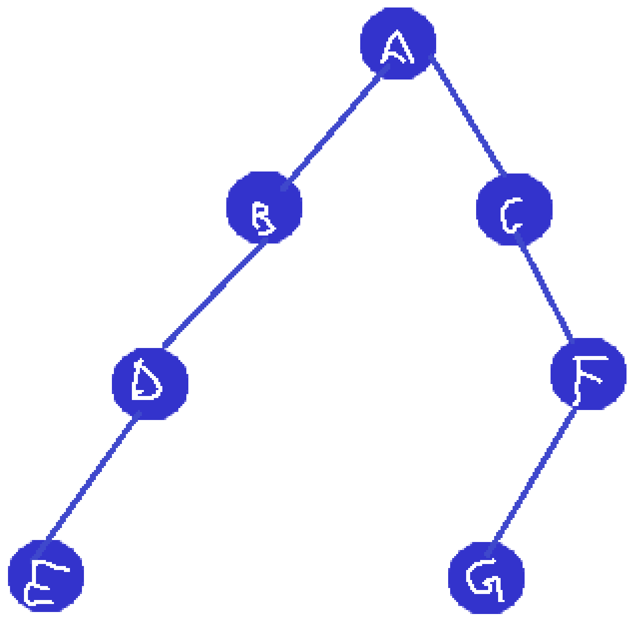
- Of these, let us make a convention that R cannot precede L in any traversal.
- We are left with three:
  - L R D
  - L D R
  - D L R
- We will study each of the three. Each has its own name.

The Inorder Traversal (LDR)

- The traversal that first completes L, then prints D, and then traverses R.
- To traverse L, use the same order.
  - First the left subtree of L, then the root of L, and then the right subtree of R.

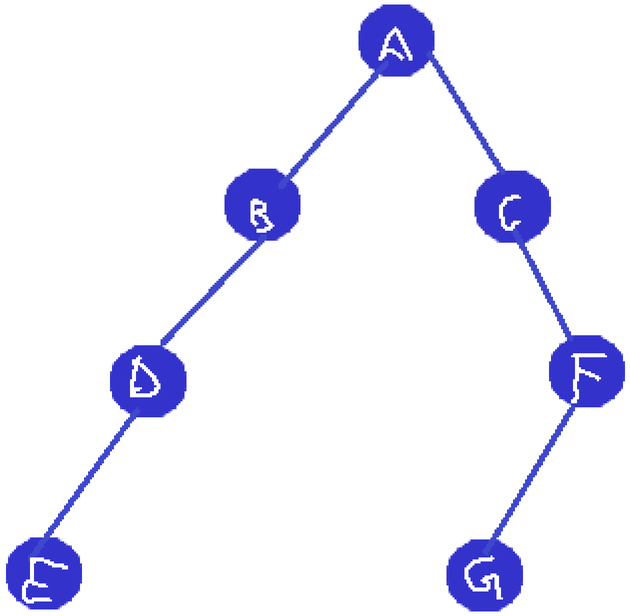
The Inorder Traversal -- Example

- Start from the root node A.
- We first should process the left subtree of A.
- Continuing further, we first should process the node E.
- Then come D and B.
- The L part of the traversal is thus E D B.



The Inorder Traversal -- Example

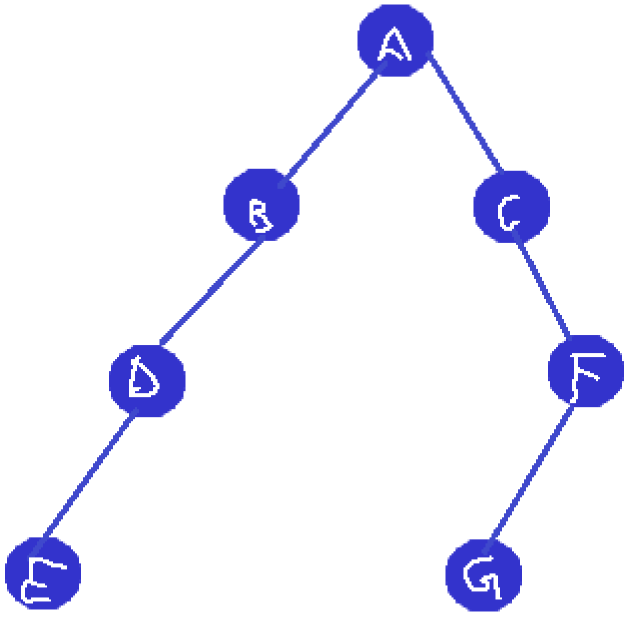
- Then comes the root node A.
- We first next process the right subtree of A.
- Continuing further, we first should process the node C.
- Then come G and F.
- The R part of the traversal is thus C G F.



Inorder: E D B A C G F

The Inorder Traversal -- Example

```
Procedure Inorder(T)
begin
    if T == NULL return;
    Inorder(T->left);
    print(T->data);
    Inorder(T->right);
end
```



Inorder: E D B A C G F

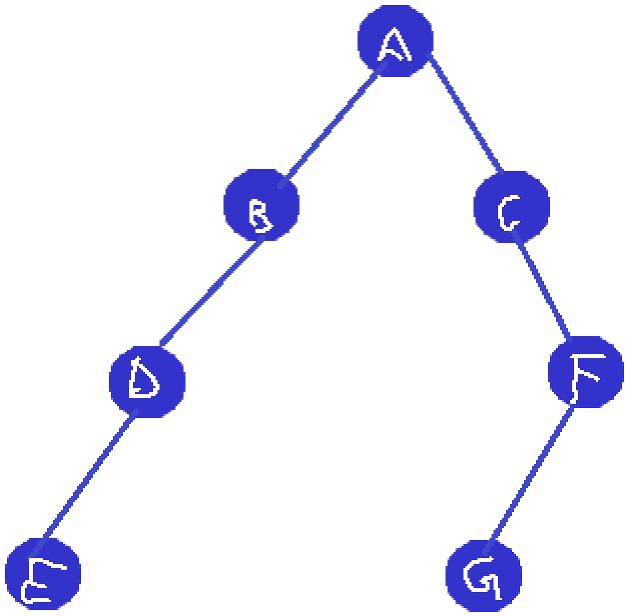
Practice Problems

The Preorder Traversal (DLR)

- The traversal that first completes D, then prints L, and then traverses R.
- To traverse L (or R), use the same order.
  - First the root of L, then left subtree of L, and then the right subtree of L.

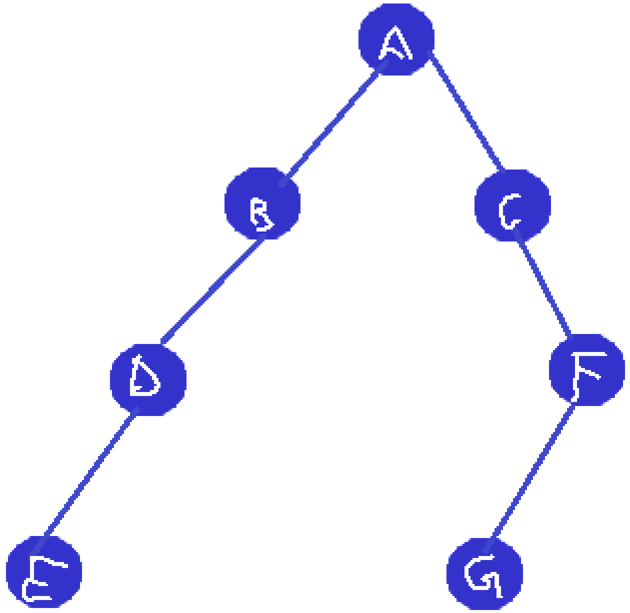
The Preorder Traversal -- Example

- Start from the root node A.
- We first should process the root node A.
- Continuing further, we should process the left subtree of A.
- This suggests that we should print B, D, and E in that order.
- The L part of the traversal is thus B D E.



The Preorder Traversal -- Example

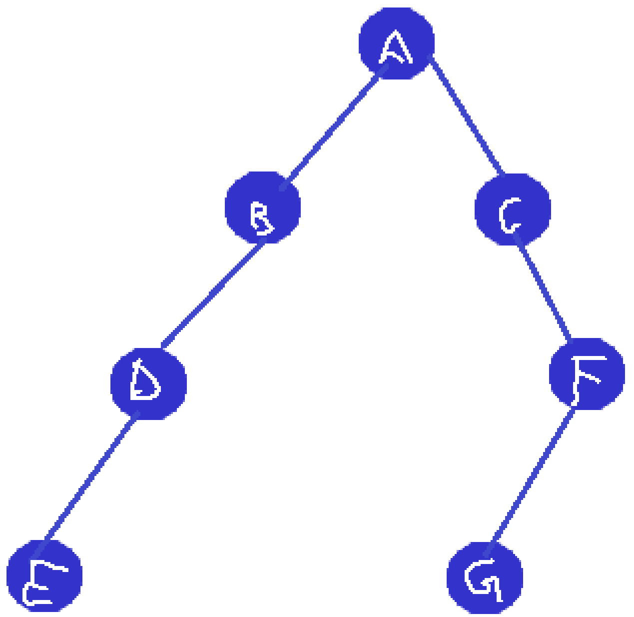
- We first next process the right subtree of A.
- Continuing further, we first should process the node C.
- Then come F and G in that order.
- The R part of the traversal is thus C F G.



Preorder: A B D E C F G

The Preorder Traversal – Example

```
Procedure Preorder(T)
begin
    if T == NULL return;
    print(T->data);
    Preorder(T->left);
    Preorder(T->right);
end
```



Preorder: A B D E C F G

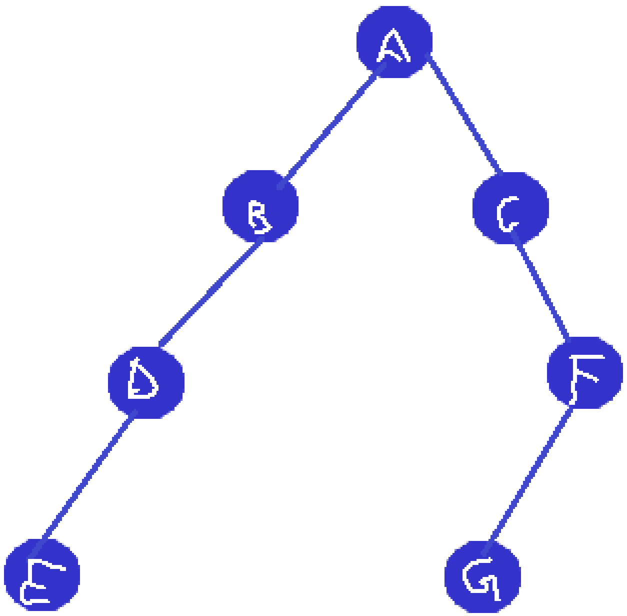
Practice Problems

The Postorder Traversal (LRD)

- The traversal that first completes L, then traverses R, and then prints D.
- To traverse L, use the same order.
  - First the left subtree of L, then the right subtree of R, and then the root of L.

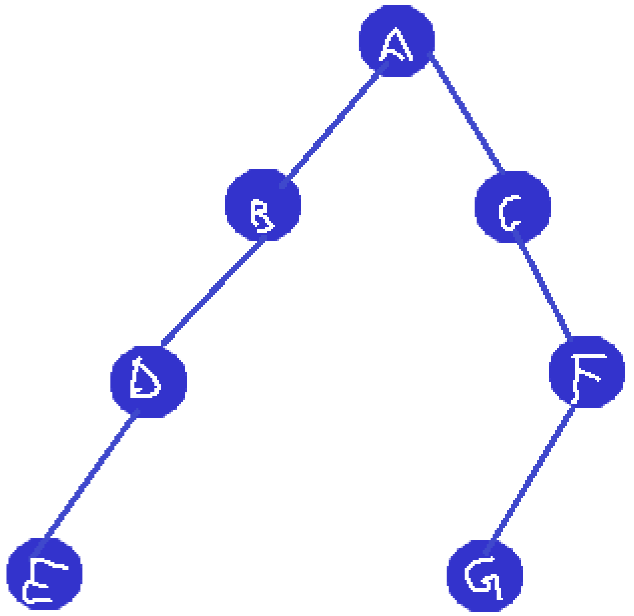
The Postorder Traversal -- Example

- Start from the root node A.
- We first should process the left subtree of A.
- Continuing further, we first should process the node E.
- Then come D and B.
- The L part of the traversal is thus E D B.



The Postorder Traversal -- Example

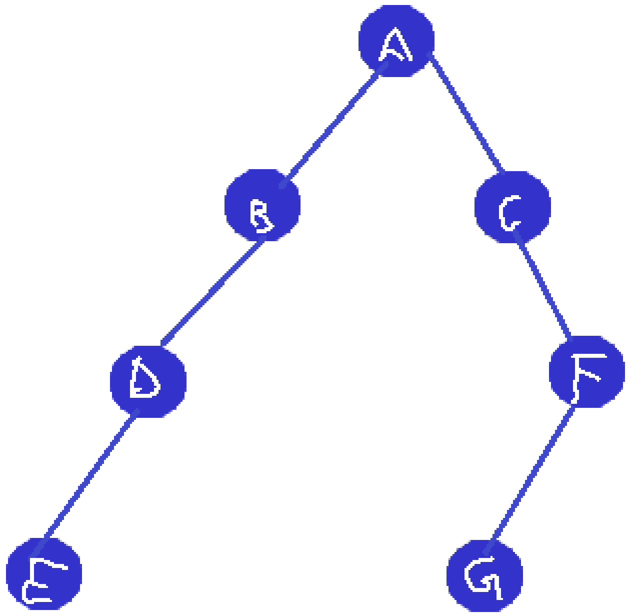
- We next process the right subtree of A.
- Continuing further, we first should process the node C.
- Then come G and F.
- The R part of the traversal is thus G F C.
- Then comes the root node A.



postorder: E D B G F C A

The Postorder Traversal -- Example

```
Procedure postorder(T)
begin
    if T == NULL return;
    Postorder(T->left);
    Postorder(T->right);
    print(T->data);
end
```



Inorder: E D B G F C A

Practice Problems

Another Kind of Traversal

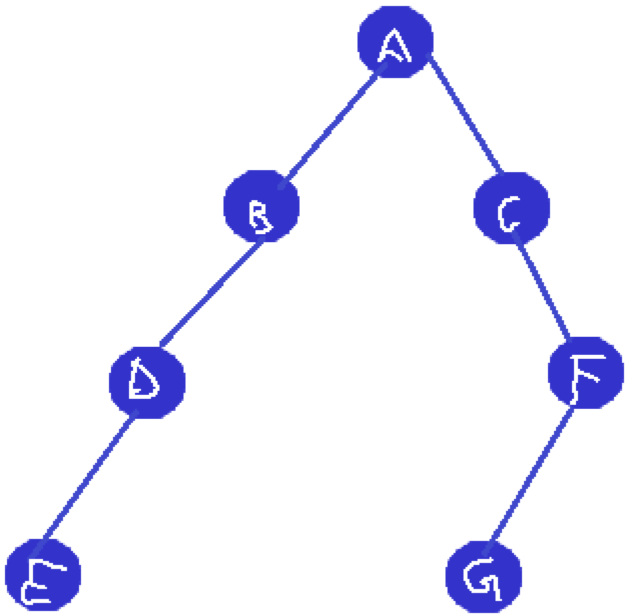
- When left and right subtree nodes can be intermixed.
- One useful traversal in this mode is the level order (or depth order) traversal.
- The idea is to print the nodes in a tree according to their lmevel starting from the root.

Another Kind of Traversal

- Why would any one want to do that?
- One example:
  - Think of printing the organization chart.
  - Start with the CEO, there are CTO, CFO, and COO, say.
  - Then, five managers under the CTO, 2 managers under the CFO, and so on,
  - Each manager has more Assistant Managers who work with a team.
  - Want to list this in that order.
- There are other such examples too
  - Game trees

How to Perform a Level Order Traversal

- Consider the same example tree.
- Starting from the root, so A is printed first.
- What should be printed next?
- Assume that we use the left before right convention.
- So, we have to print B next.
- How to remember that C follows B.
- And then D should follow C?



Level Order Traversal

- Indeed, can remember that B and C are children of A.
- But, have to get back to children of B after C is printed.
- For this, one can use a queue.
  - Queue is a first-in-first-out data structure.

Level Order Traversal

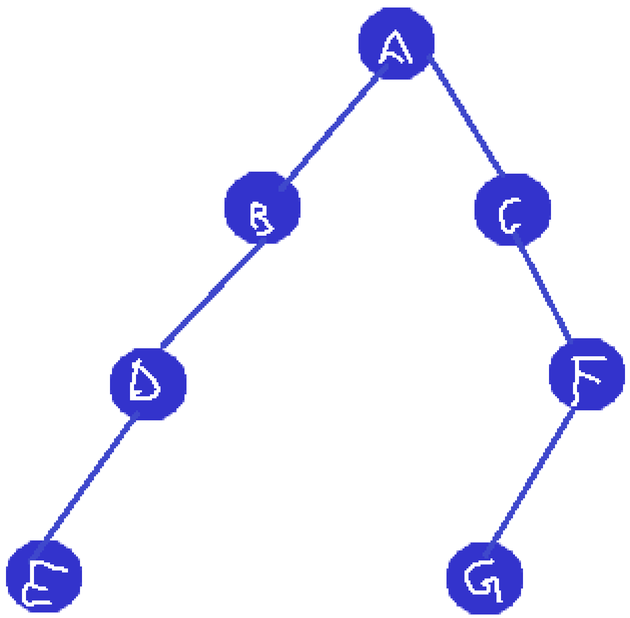
- The idea is to queue-up children of a parent node that is visited recently.
- The node to be visited recently will be the one that is at the front of the queue.
  - That node is ready to be printed.
- How to initialize the queue?
  - The root node is ready!

Level Order Traversal

```
Procedure LevelOrder(T)
Begin
    queue  Q;
    insert root into the queue Q;
    while Q is not empty do
        v = delete();
        print v->data;
        if v->left is not NULL insert v->left into Q;
        if v->right is not NULL insert v->right into Q;
    end-while
end
```

Level Order Traversal Example

Queue	Output
----------	----------
A	
B C	A
C D	B
D F	C
F E	D
E G	F
G	E
EMPTY	G



- Queue and output are shown at every stage.

Analysis of Traversal Techniques

- For inorder, preorder, and postorder traversal, let the tree have n nodes of which n_1 are in the left subtree and the rest in the right subtree.
- Recurrence relation:

$$T(n) = T(n_1) + T(n-n_1-1) + O(1)$$

- Can solve by guessing that $T(n) \leq cn$ for constant c.
- Verify.
  - $T(n) \leq cn_1 + c(n-n_1-1) + O(1) \leq cn$, provided c is large enough.

Analysis of Traversal Techniques

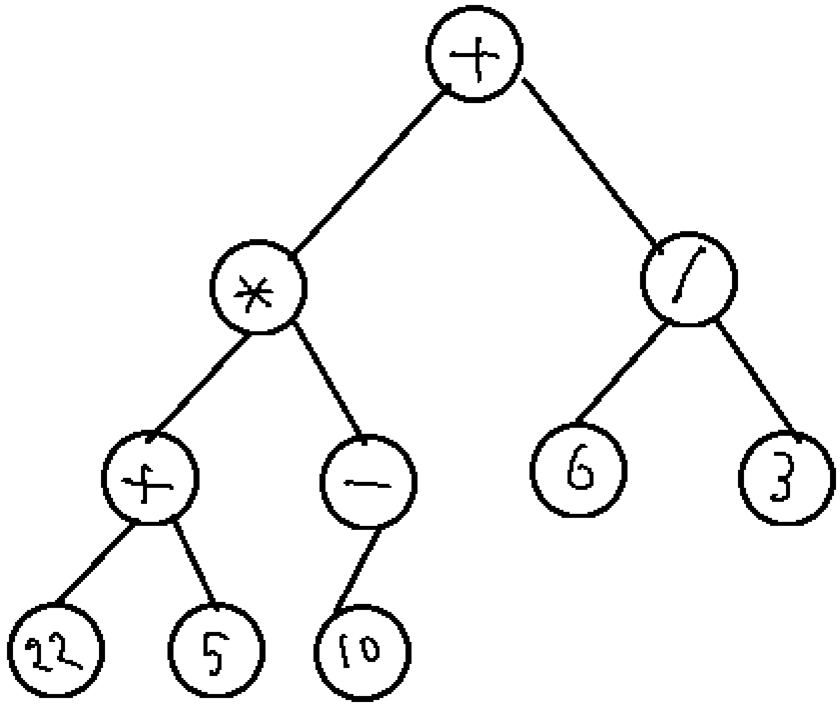
- How to analyze the level order traversal?
- Assume that the tree has n nodes.
- Each node is placed in the queue exactly once.
- The rest of the operations are all O(1) for every node.
- So the total time is O(n).
- This traversal can be seen as forming the basis for a graph traversal.

Application to Expression Evaluation

- We know what expression evaluation is.
- We deal with binary operators.
- An expression tree for a expression with only unary or binary operators is a binary tree where the leaf nodes are the operands and the internal nodes are the operators.

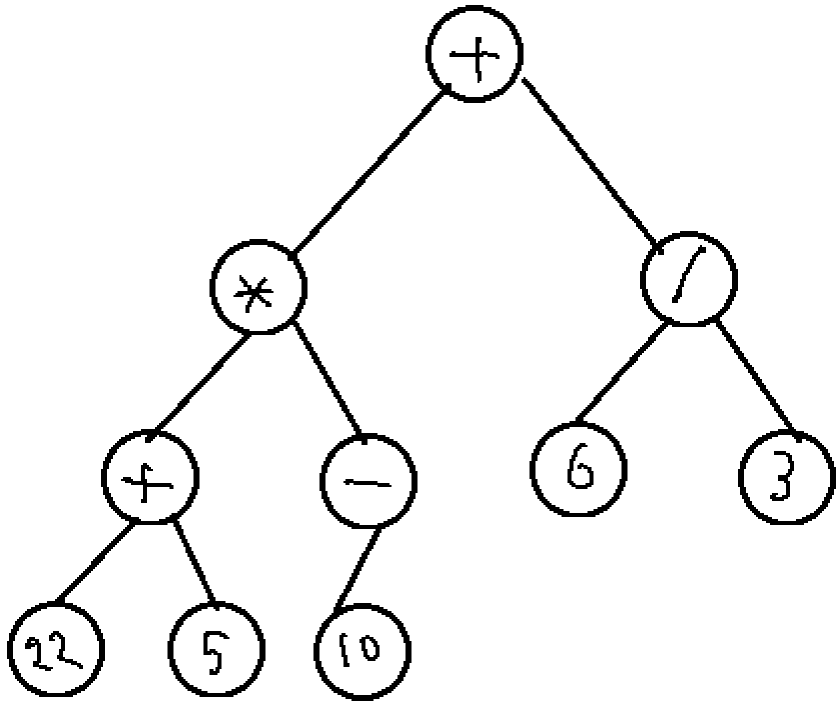
Example Expression Tree

- See the example to the right.
- The operands are 22, 5, 10, 6, and 3.
- These are also leaf nodes.



Questions wrt Expression Tree

- How to evaluate an expression tree?
  - Meaning, how to apply the operators to the right operands.
- How to build an expression tree?
  - Given an expression, how to build an equivalent expression tree?



A Few Observations

- Notice that an inorder traversal of the expression tree gives an expression in the infix notation.
  - The above tree is equivalent to the expression $((22 + 5) \times (-10)) + (6/3)$
- What does a postorder and preorder traversal of the tree give?
  - Answer: ??

Why Expression Trees?

- Useful in several settings such as
  - compliers
  - can verify if the expression is well formed.

How to Evaluate using an Expression Tree

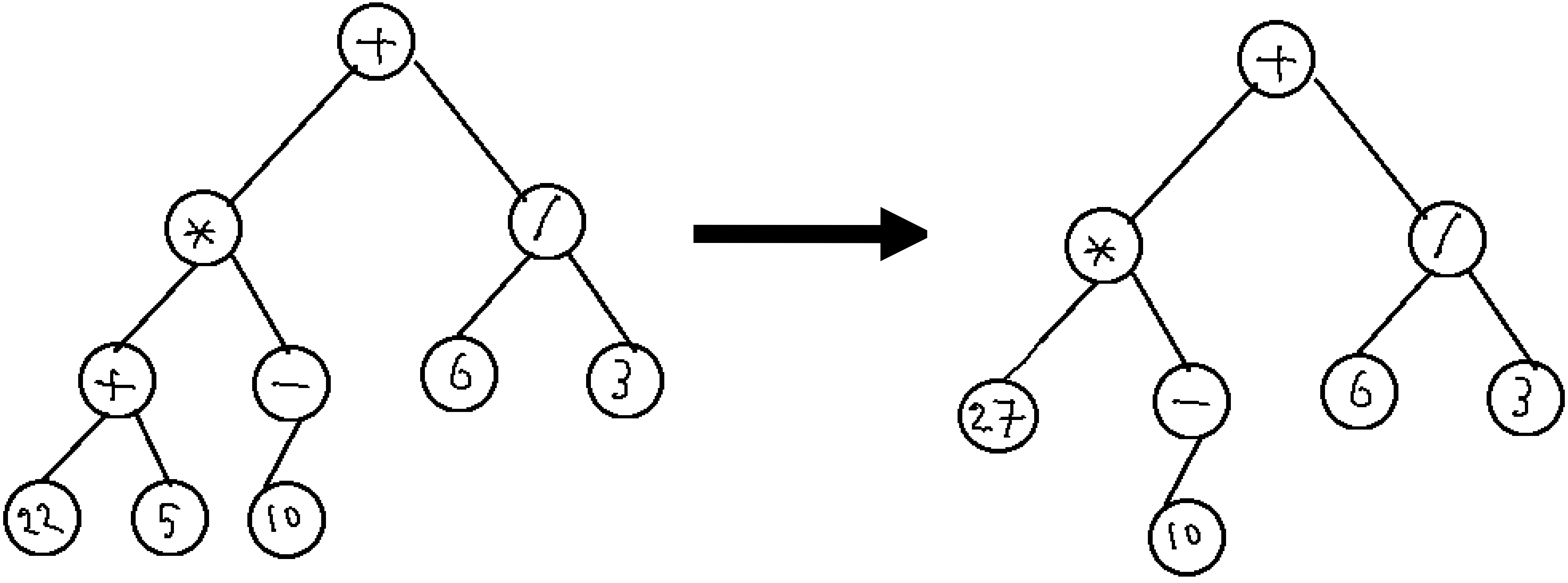
- Essentially, have to evaluate the root.
- Notice that to evaluate a node, its left subtree and its right subtree need to be operands.
- For this, may have to evaluate these subtrees first, if they are not operands.
- So, Evaluate(root) should be equivalent to:
  - Evaluate the left subtree
  - Evaluate the right subtree
  - Apply the operator at the root to the operands.

How to Evaluate using an Expression Tree

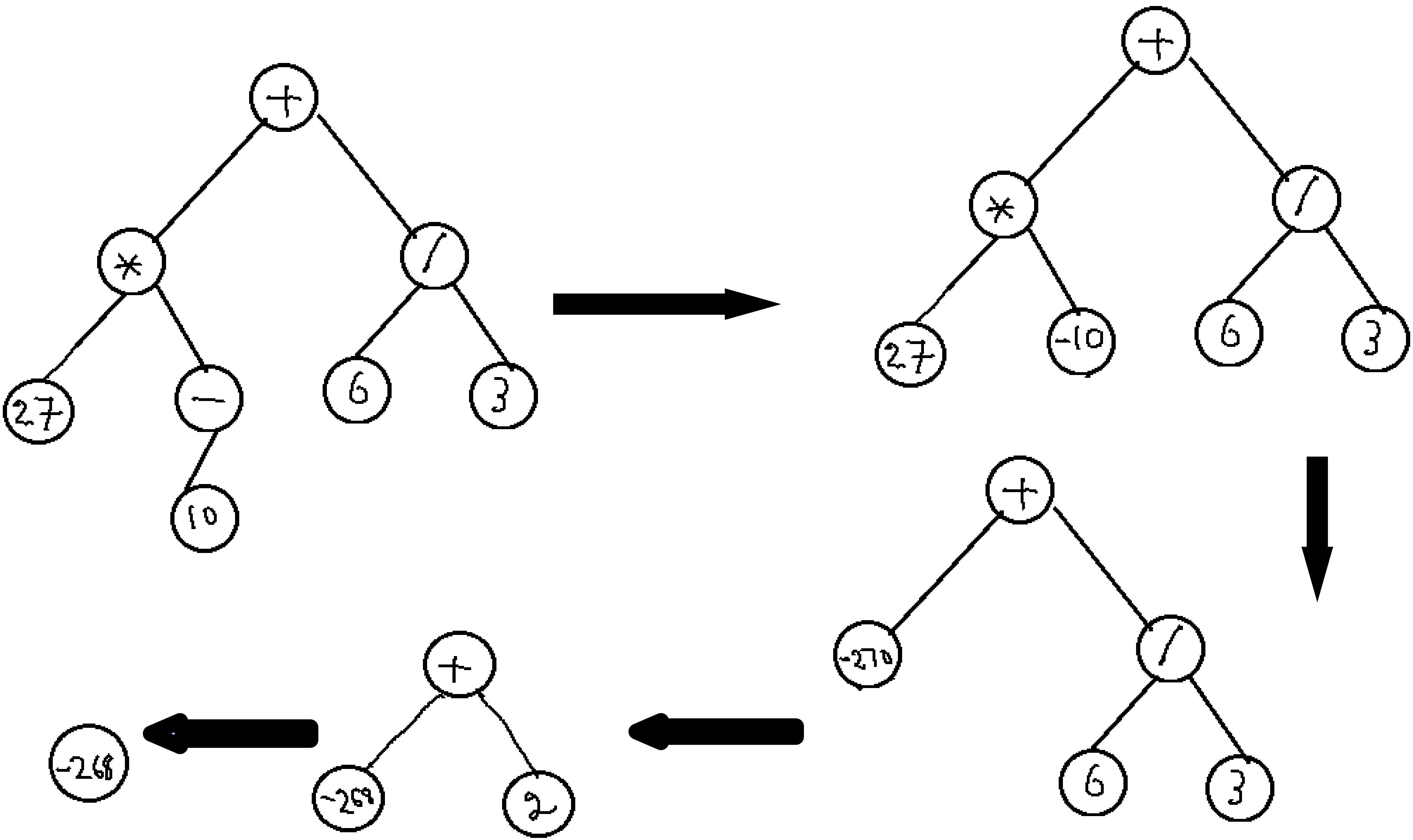
- This suggests a recursive procedure that has the above three steps.
- Recursion stops at a node if it is already an operand.

How to Evaluate using an Expression Tree

- Example



Example Contd...



Pending Question

- How to build an expression tree?
- Start with an expression in the infix notation.
- Recall how we converted an infix expression to a postfix expression.
- The idea is that operators have to wait to be sent to the output.
  - A similar approach works now.

Building an Expression Tree

- Let us start with a postfix expression.
- The question is how to link up operands as (sub)trees.
- As in the case of evaluating a postfix expression, have to remember operators seen so far.
  - need to see the correct operands.
- A stack helps again.
- But instead of evaluating subexpression, we have to grow them as trees.
  - Details follow.

Building an Expression Tree

- When we see an operand :
  - That could be a leaf node...Or a tree with no children.
  - What is its parent?
  - Some operator.
  - In our case, operands can be trees also.
- The above observations suggest that operands should wait on the stack.
  - Wait as trees.

Building an Expression Tree

- What about operators?
- Recall that in the postfix notation, the operands for an operator are available in the immediate preceding positions.
- Similar rules apply here too.
- So, pop two operands (trees) from the stack.
- Need not evaluate, but create a bigger (sub)tree.

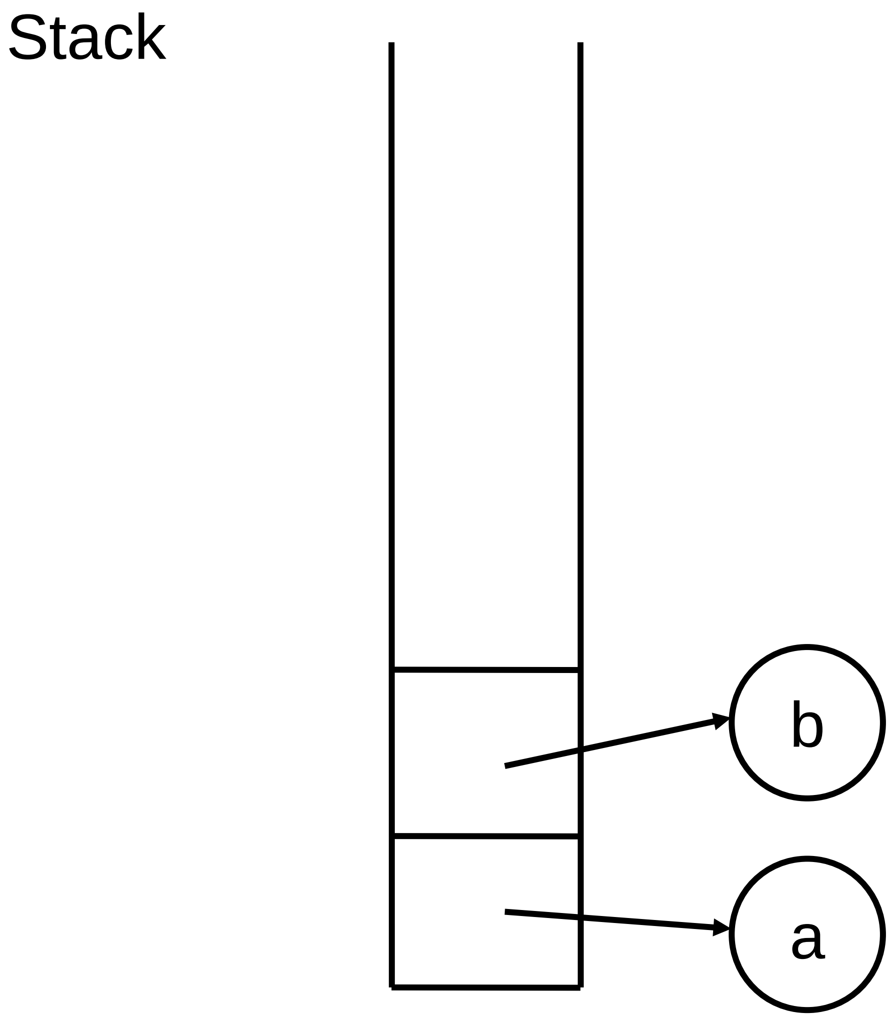
Building an Expression Tree

```
Procedure ExpressionTree(E)
//E is an expression in postfix notation.
begin
    for i=1 to |E| do
        if E[i] is an operand then
            create a tree with the operand as the only node;
            add it to the stack
        else if E[i] is an operator then
            pop two trees from the stack
            create a new tree with E[i] as the root and the two trees
              popped as its children;
            push the tree to the stack
    end-for
end
```

Example

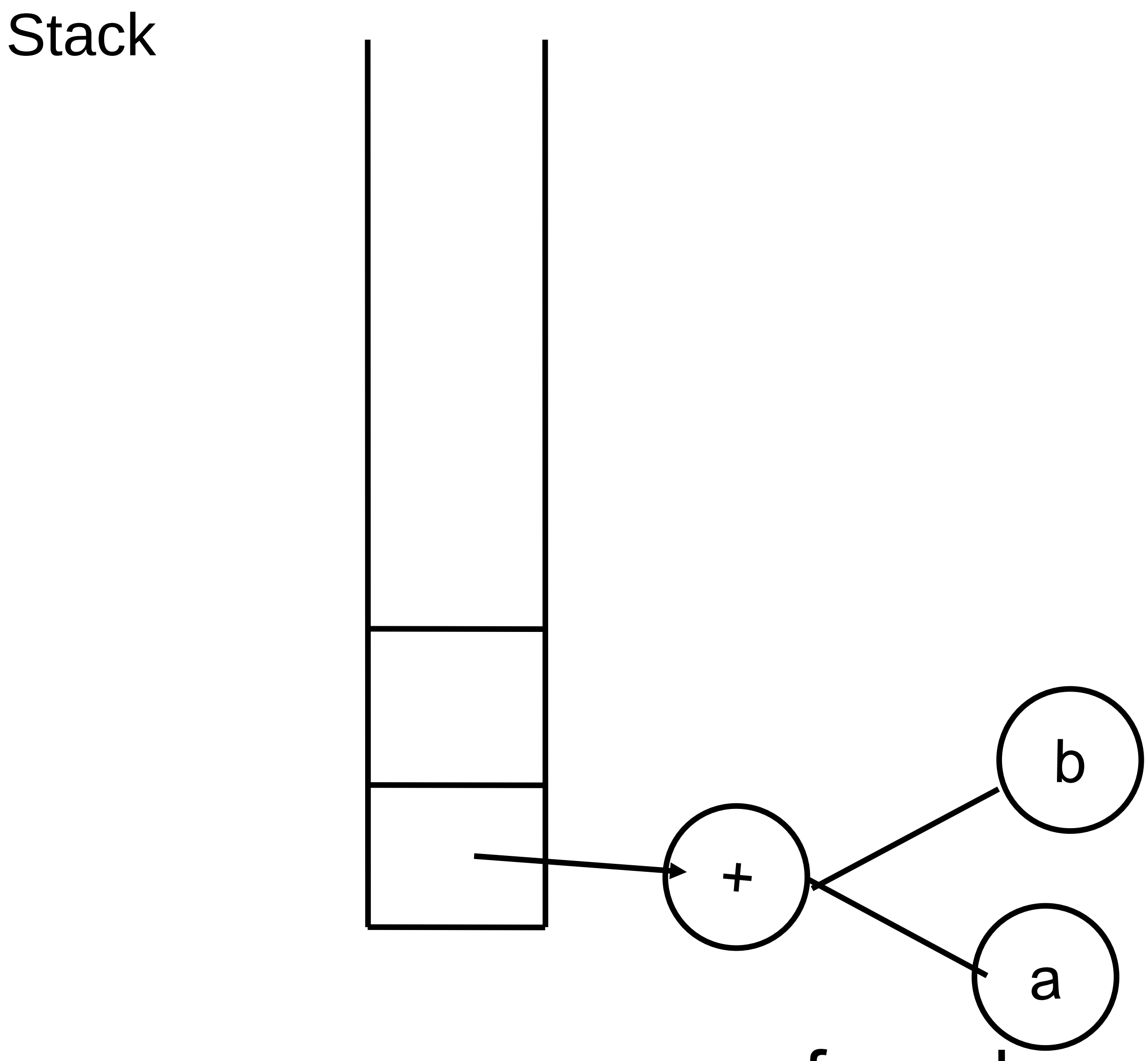
- Consider the expression
- The postfix of the expression is a b + f − c d × e + /
- Let us follow the above algorithm.

Example



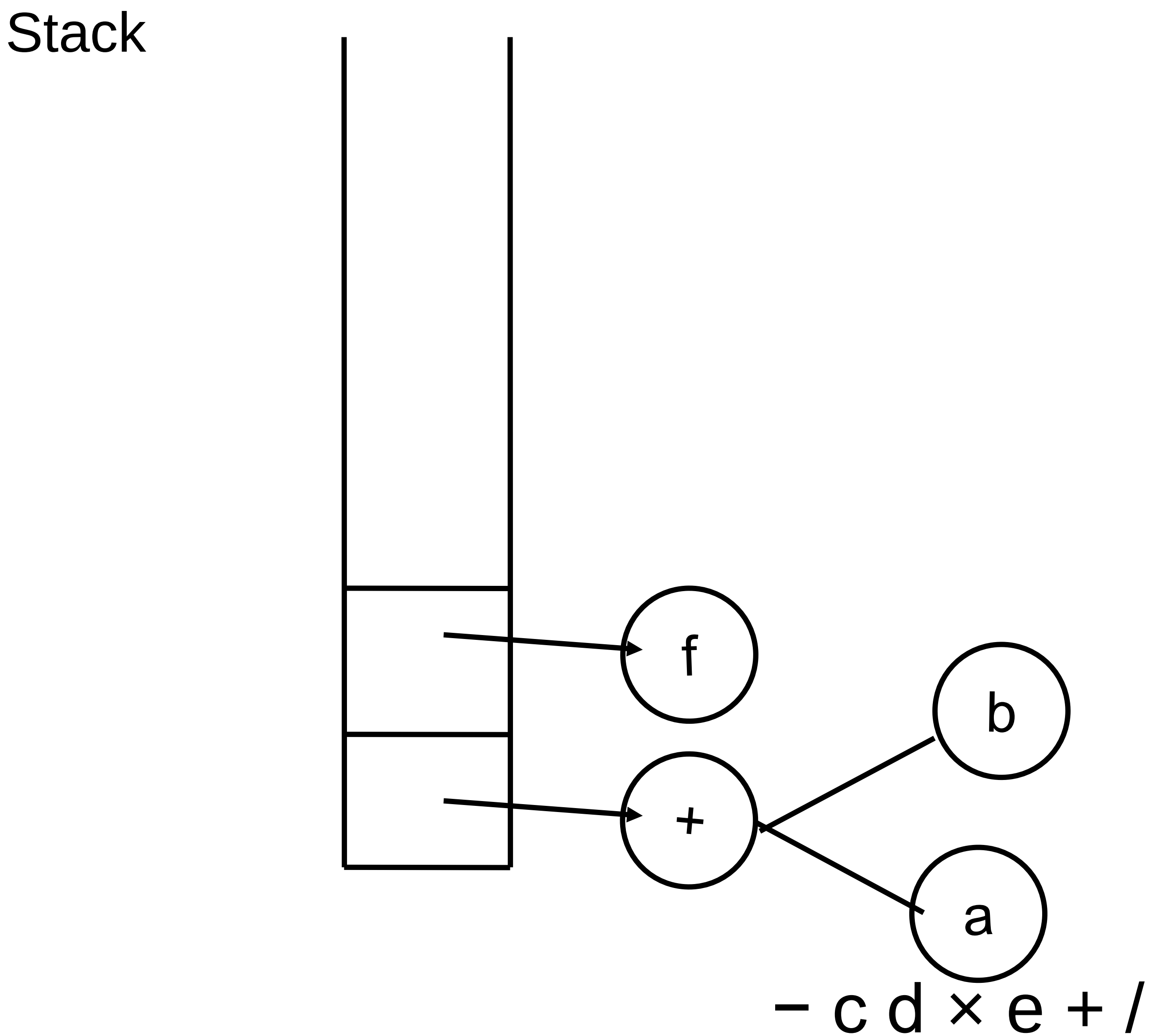
$+ \; f \; - \; c \; d \; \times \; e \; + \; /$

Example



f − c d × e + /

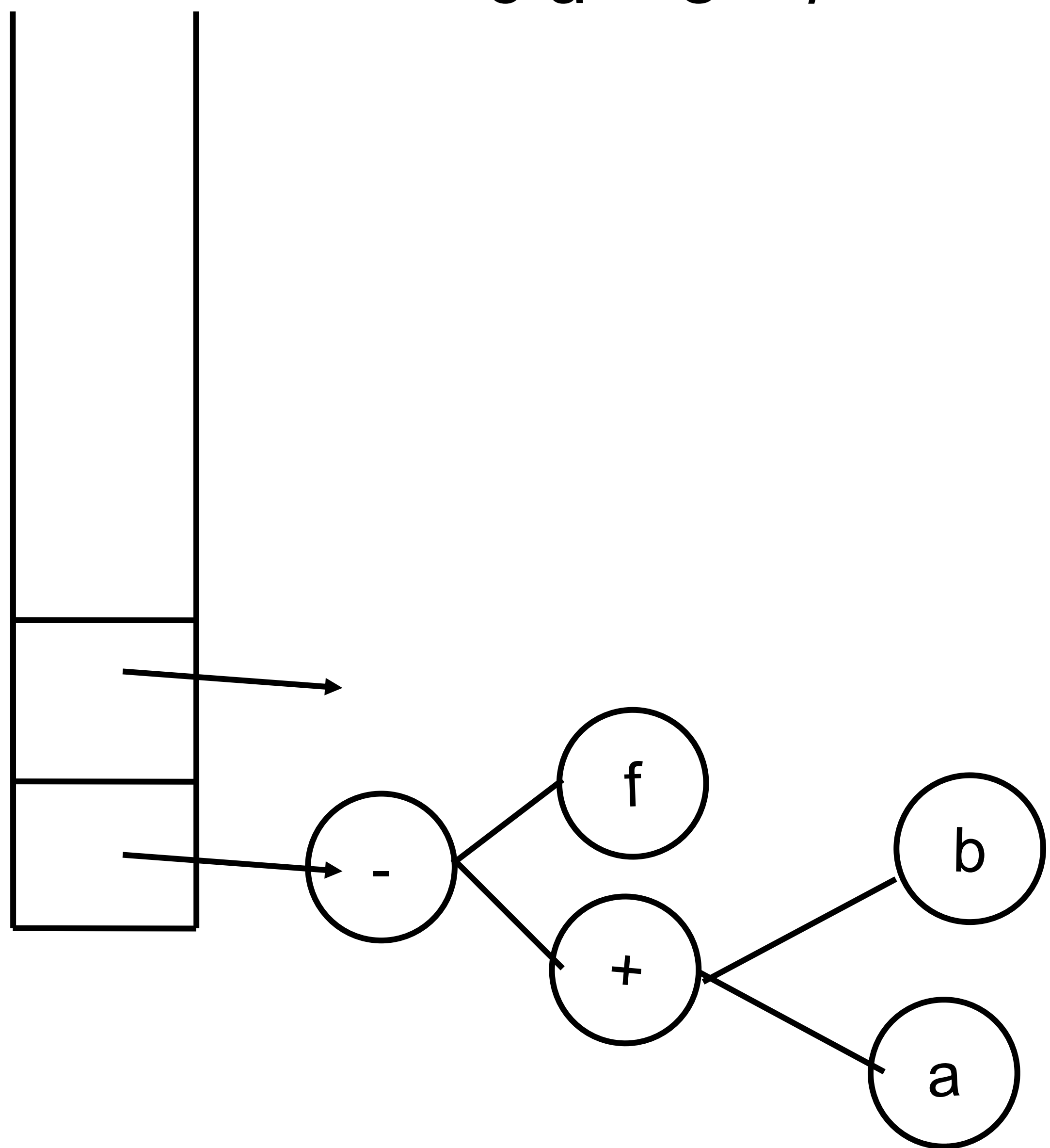
Example



Example

Stack

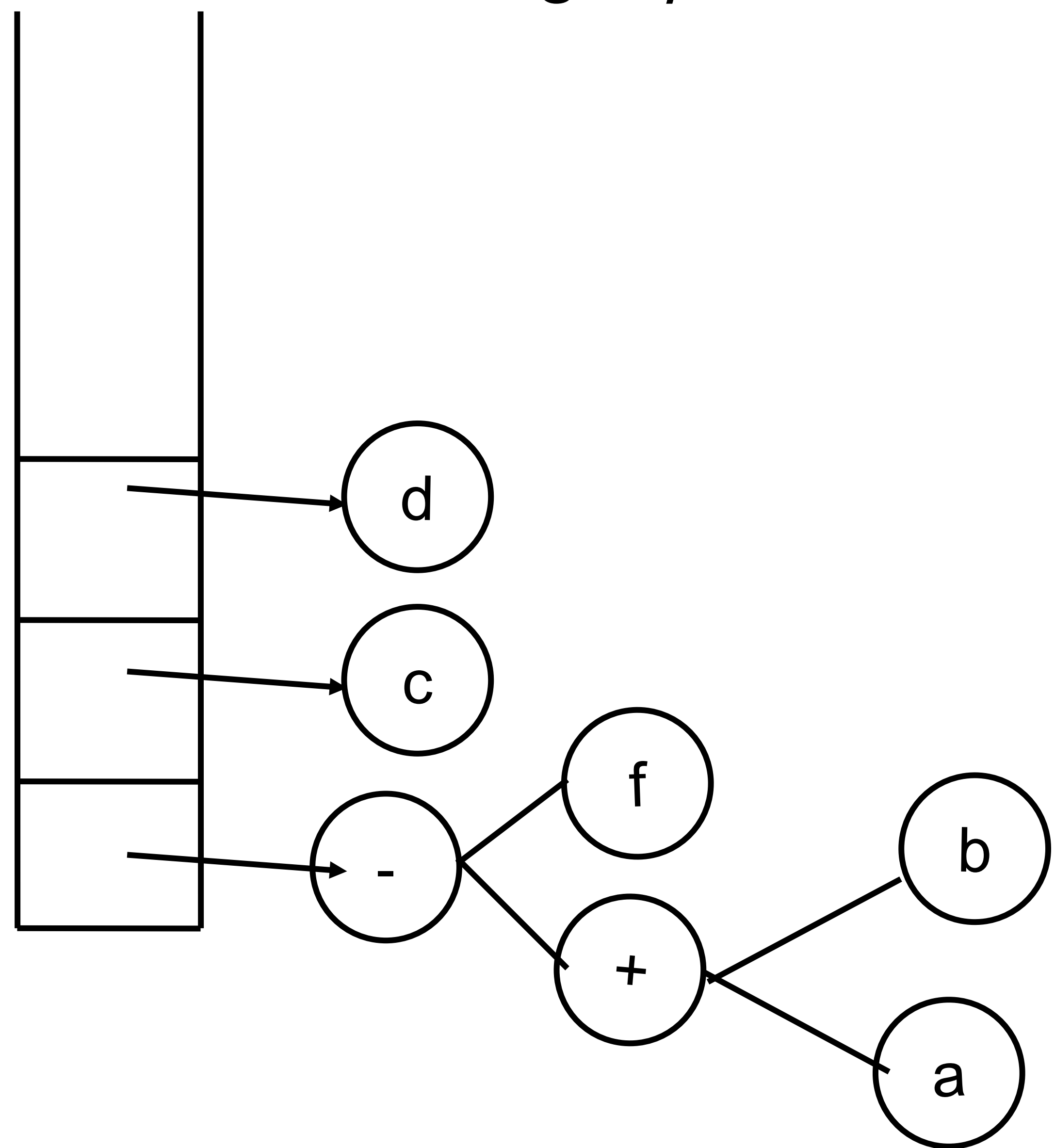
c d × e + /



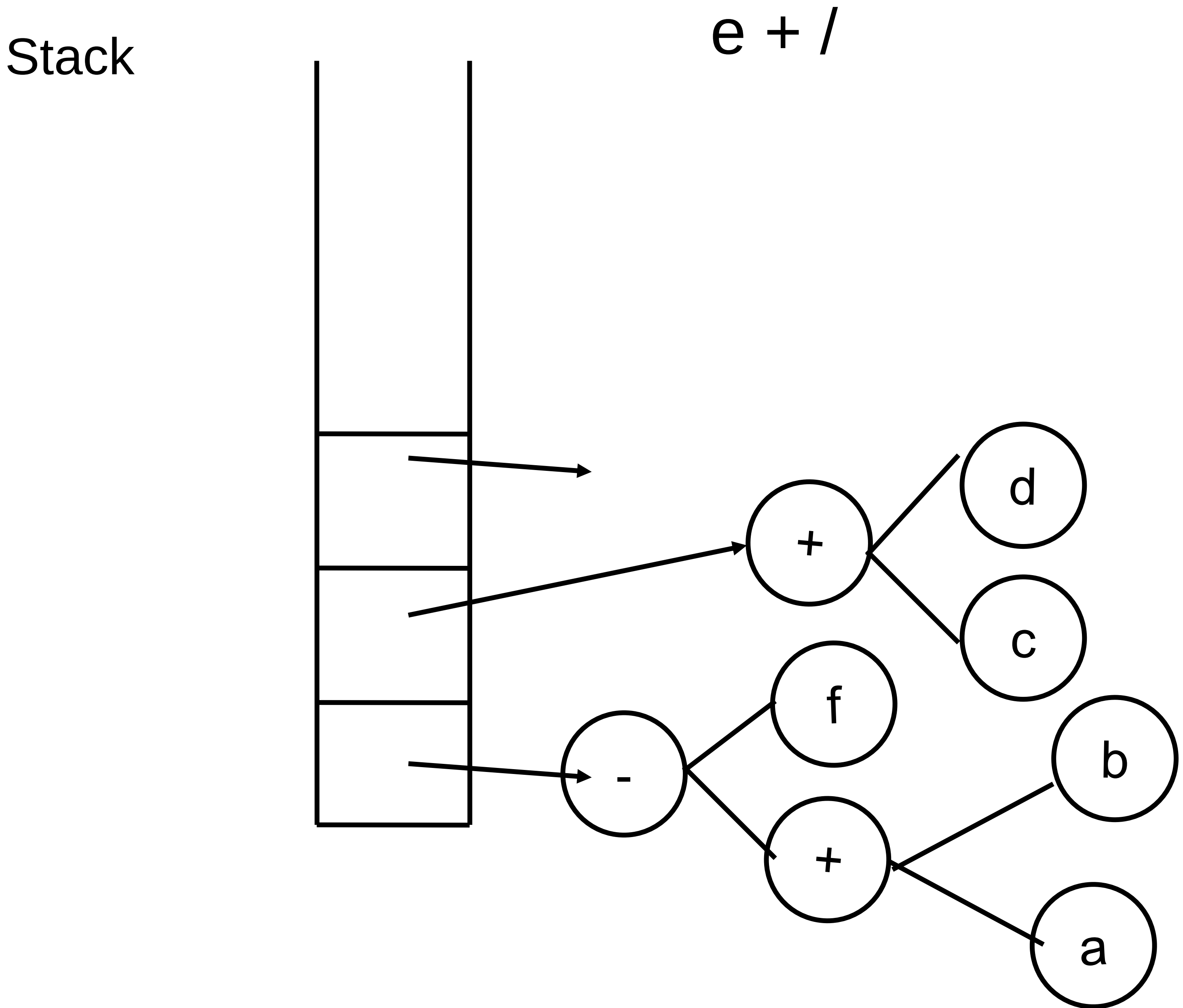
Example

Stack

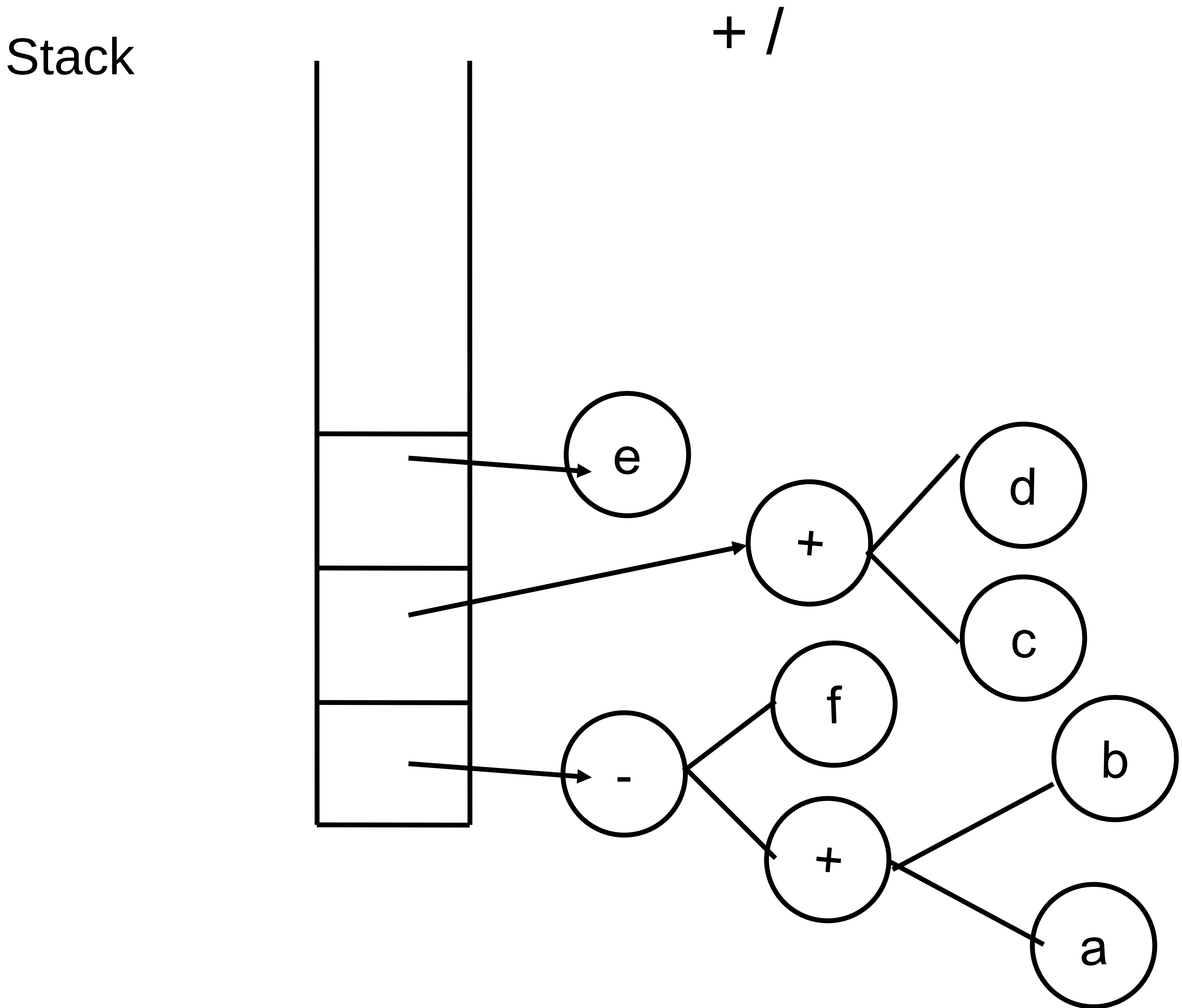
× e + /



Example

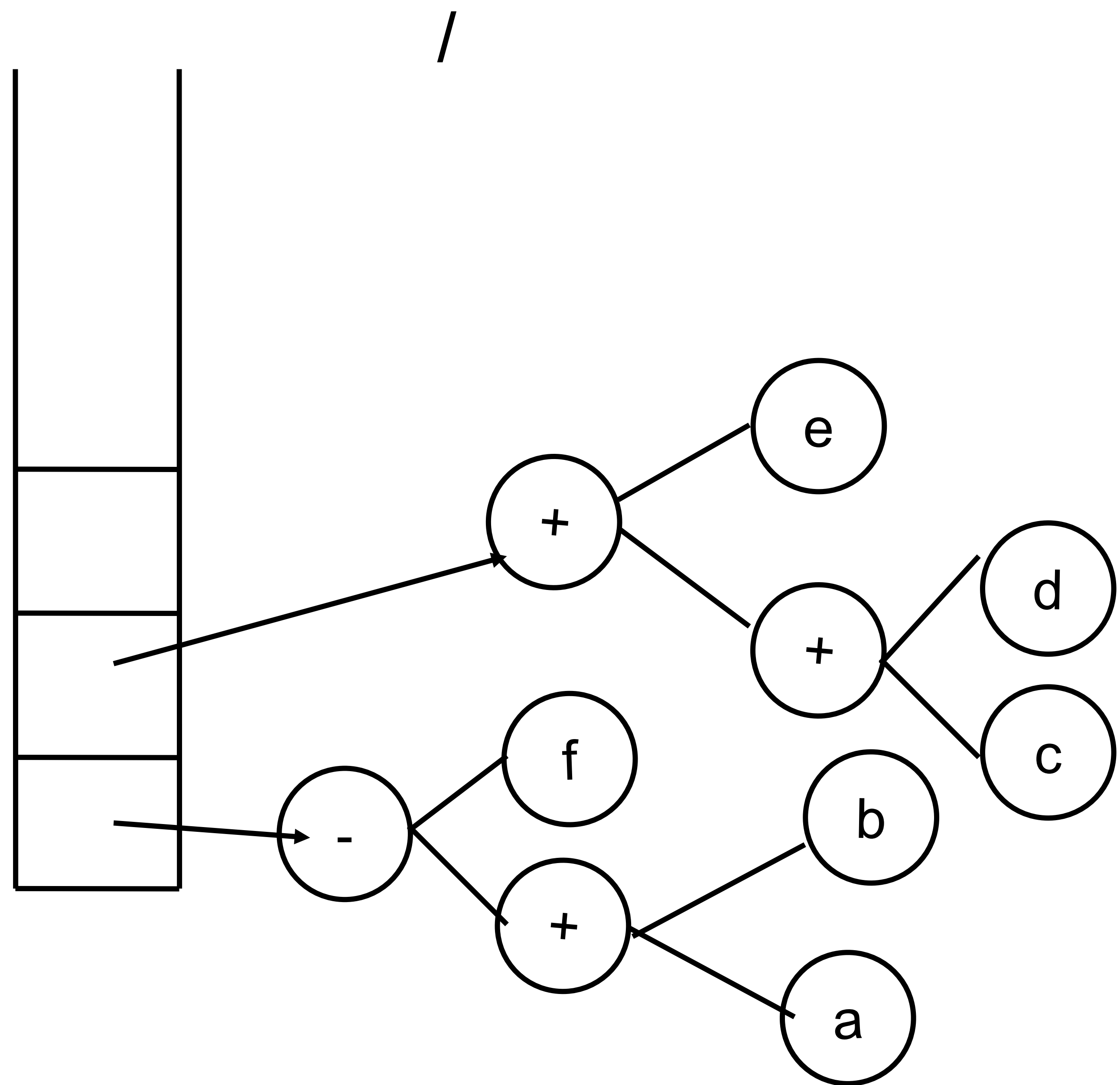


Example



Example

Stack



Example