Trees From Files

In Lab 5 there is an algorithm for reading a tree from a data file. The nodes are presented in the order of a postorder traversal of the tree. Each line of the data file has the form

```
data leftbit   rightbit
```

where the first bit is 1 if the node has a left child and the second bit is 1 if the node has a right child.

For example

```
A 1 0
```

means that a node has data "A"; it has a left child but not a right child.

The algorithm for reading a file consisting of such lines and building a tree from it makes use of a stack of trees.   At each step:

1. Get the next line of the file and separate into its data, left-bit and right-bit components.
2. Build a new node for the line and insert the data into it.
3. If the **right-bit** is 1 pop the stack for the node's right child; otherwise make a new empty tree for the right child.
4. If the **left-bit** is 1 pop the stack for the node's left child; otherwise make a new empty tree for the node’s left child.
5. Push the node onto the stack

When you reach the end of the file there should be 1 item on the stack --- the entire tree.

For example:

D 0 0

E 0 0

C 1 1

B 0 1

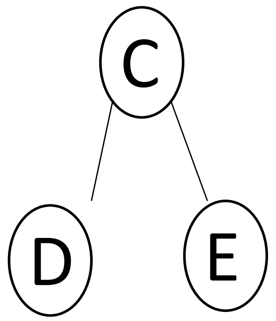
G 0 0

H 0 0

F 1 1

A 1 1

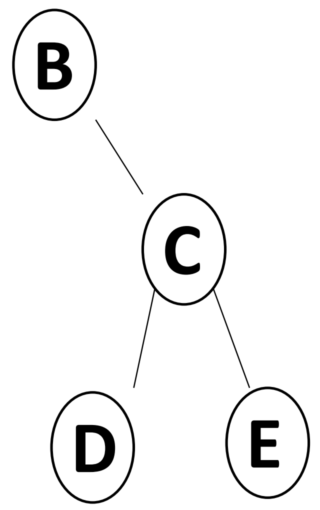
We read the first two lines: D and E have no children so the singleton nodes are pushed onto the stack with E on top of D. Node C has two children so node C pops E as its right child, D as its left:



This node is pushed onto the stack; we'll call it treeC.

D 0 0
E 0 0
C 1 1
B 0 1
G 0 0
H 0 0
F 1 1
A 1 1

We next read line B 0 1.
We make a node with data B and pop treeC off the stack as B's right child:



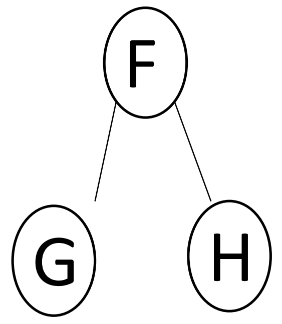
We'll call this treeB. It gets pushed onto the stack.

D 0 0
E 0 0
C 1 1
B 0 1
G 0 0
H 0 0
F 1 1
A 1 1

Next nodes G and H are made as trees with no children and are pushed onto the stack. The stack is now

node H
node G
treeB

The next line of the file builds treeF with H as its right child and G as its left:



D	0	0
E	0	0
C	1	1
B	0	1
G	0	0
H	0	0
F	1	1
A	1	1

TreeF is pushed onto the stack above treeB. The last line of the file tells us to build a new node A. We pop treeF as its right child and treeB as its left: