Given

`int x = 12 >> 3;`  \(\Rightarrow 0100 \Rightarrow 0010\)

`int y = x << 1;`  \(\Rightarrow 0010\)

- What is y?
  A. 1
  B. 2
  C. 10
  D. None of the above
  E. Whatever
Today’s Outline

• Tree
• Tree
• Binary Tree
Introducing Trees

- We have been studying structures with a linear organization, i.e. each node has at most 1 successor.
  - line at a final exam
  - stacks of plates

- But you may want to allow more than 1 successor!
  - family tree
  - road network → not a tree
  - directory hierarchy → tree?
A tree is a data structure where nodes can have:
- one predecessor (called parent)
- multiple successors (called children)
Tree

- Parent
- Child
- Ancestor
- Descendant
- Sibling

\[ \text{ex} \]
A \rightarrow C
C \rightarrow A
A \rightarrow E, A \rightarrow C
E \rightarrow A, C \rightarrow A
B \rightarrow C, C \rightarrow B

- Root
- Internal nodes
- Subtree: "Every node is the root of its own subtree"
- Leaves (external nodes)
Tree Logic (Natalie Jereminjenko) at Mass MoCA

Tree Features

- **Degree (of node):** number of children of node
- **Degree (of tree):** maximum degree (across all nodes)
- **Depth of node:** number of edges from root to node
- **Height of tree:** maximum depth (across all nodes)
Tree examples
House of Normandy, Battle of Hastings, 1066

William I

- Robert
- William II
- Adela
- Henry I
  - Stephen
  - William
  - Matilda
    - Henry II
The diagram illustrates the evolutionary relationships among various canids. It shows the divergence of species from the Miocene to the Pleistocene periods, with a timeline indicating millions of years before present. Key labels include:

- Black Bear
- Domestic Dog
- Gray Wolf
- Coyote
- Cape Hunting Dog
- Black-Backed Jackal
- Bush Dog
- Maned Wolf
- Hoary Fox
- Crab-Eating Fox
- Gray Fox
- Bat-Eared Fox
- Raccoon Dog
- Cape Fox
- Red Fox
- Fennec Fox
- Kit Fox
- Arctic Fox

The diagram is divided into three main groups: Wolf-like canids, South American canids, and Fox-like canids.
game the
Today’s Outline

• Tree
  • Tree
  • Binary Tree
**Binary Trees**

- Binary Tree: Tree with Degree of each node $\leq 2$
- Recursively defined. A tree can either be:
  - Empty
  - Root with left and right subtrees
Full vs. Complete

- **Full** tree – A full binary tree of height \( h \) has *leaves only* on level \( h \), and each internal node has exactly 2 children.

- **Complete** tree – A *complete* binary tree of height \( h \) is *full* to height \( h-1 \) and has all leaves at level \( h \) in leftmost locations.

All full trees are complete, but not all complete trees are full!
Example: Expression Trees

4 * 2 + 3

Build using constructor
new BinaryTree<E>(value, leftSubTree, rightSubTree)

BinaryTree<String> fourTimesTwo =
    new BinaryTree<String>("*",
    new BinaryTree<String>("4"),
    new BinaryTree<String>("2");

BinaryTree<String> fourTimesTwoPlusThree =
    new BinaryTree<String>("+",
    fourTimesTwo,
    new BinaryTree<String>("3");
Evaluating Expression Trees

- Starting at the root,
  - Evaluate left subtree
  - Evaluate right subtree
  - Perform operation (+, -, *, /) with left and right
int evaluate(BinaryTree<String> tree) {
    // base case
    if (tree.height() == 0)
        return Integer.parseInt(tree.value());

    // recursive case
    int left = evaluate(tree.left());
    int right = evaluate(tree.right());

    switch (tree.value()) {
        case "+":
            return left + right;
        case "-":
            return left - right;
        case "*":
            return left * right;
        case "/":
            return left / right;
    }
    return ERROR_CODE;
}