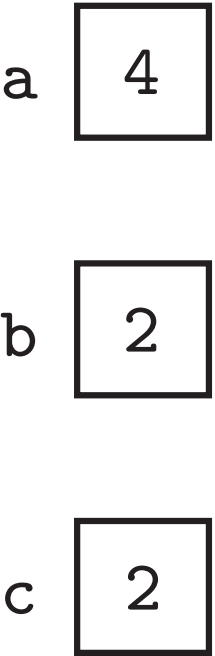
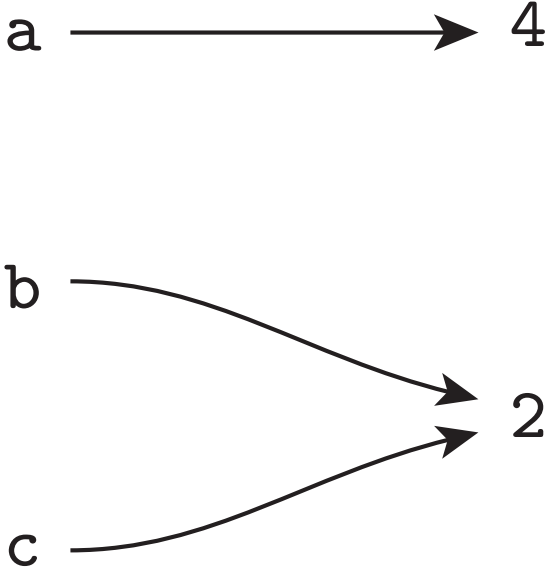
Fortran	Pascal	C	Ada
		++, -- (post-inc., dec.)	
**	not	++, -- (pre-inc., dec.), +, - (unary), &, * (address, contents of), !, ~ (logical, bit-wise not)	abs (absolute value), not, **
*, /	*, /, div, mod, and	* (binary), /, % (modulo division)	*, /, mod, rem
+, - (unary and binary)	+, - (unary and binary), or	+, - (binary)	+, - (unary)
		<<, >> (left and right bit shift)	+, - (binary), & (concatenation)
.eq., .ne., .lt., .le., .gt., .ge. (comparisons)	<, <=, >, >=, =, <>, IN	<, <=, >, >= (inequality tests)	=, /= , <, <=, >, >=
.not.		==, != (equality tests)	
		& (bit-wise and)	
		^ (bit-wise exclusive or)	
		| (bit-wise inclusive or)	
.and.		&& (logical and)	and, or, xor (logical operators)
.or.		|| (logical or)	
.eqv., .neqv. (logical comparisons)		?: (if...then...else)	
		=, +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |= (assignment)	
		, (sequencing)	


a
4
b
2
c
2


a
4
b
2
c

```
      goto L6          -- jump to code to compute address
L1:   clause_A
      goto L7
L2:   clause_B
      goto L7
L3:   clause_C
      goto L7
      ...
L4:   clause_D
      goto L7
L5:   clause_E
      goto L7

L6:   r1 := ...        -- computed target of branch
      goto *r1
L7:
```

```
T:    &L1                  -- controlling expression = 1
      &L2
      &L3
      &L3
      &L3
      &L5
      &L2
      &L5
      &L5
      &L4                  -- controlling expression = 10
L6:   r1 := . . .          -- calculate controlling expression
      if r1 < 1 goto L5
      if r1 > 10 goto L5         -- L5 is the "else" arm
      r1 -:= 1                   -- subtract off lower bound
      r1 := T[r1]
      goto *r1
L7:
```

```
class BinTree:
    def __init__(self):     # constructor
        self.data = self.lchild = self.rchild = None
    ...
    # other methods: insert, delete, lookup, ...

    def preorder(self):
        if self.data != None:
            yield self.data
        if self.lchild != None:
            for d in self.lchild.preorder():
                yield d
        if self.rchild != None:
            for d in self.rchild.preorder():
                yield d
```

```
class BinTree<T> implements Iterable<T> {
    BinTree<T> left;
    BinTree<T> right;
    T val;
    ...
    // other methods: insert, delete, lookup, ...

    public Iterator<T> iterator() {
        return new TreeIterator(this);
    }
    private class TreeIterator implements Iterator<T> {
        private Stack<BinTree<T>> s = new Stack<BinTree<T>>();
        TreeIterator(BinTree<T> n) {
            if (n.val != null) s.push(n);
        }
        public boolean hasNext() {
            return !s.empty();
        }
        public T next() {
            if (!hasNext()) throw new NoSuchElementException();
            BinTree<T> n = s.pop();
            if (n.right != null) s.push(n.right);
            if (n.left != null) s.push(n.left);
            return n.val;
        }
        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
```