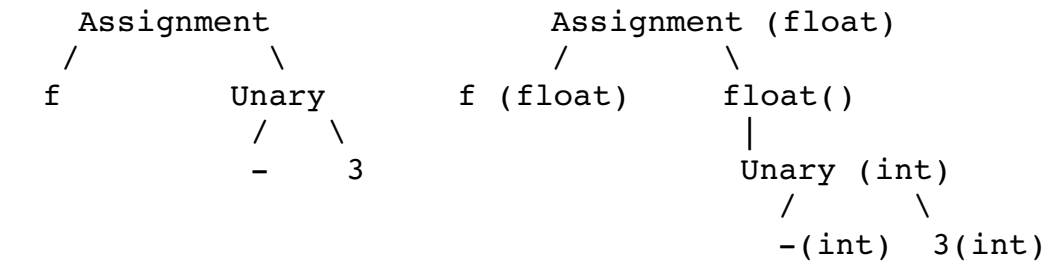
Programming Languages: Principles and Paradigms

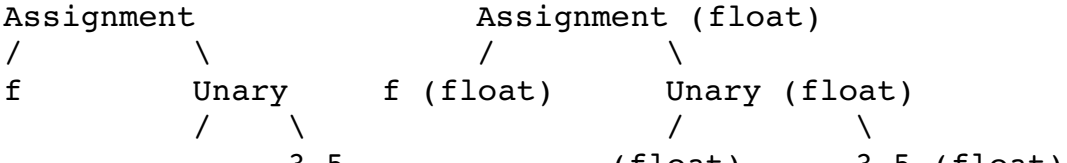
6.4, 6.6, 6.10, 6.11, 6.12

6.4

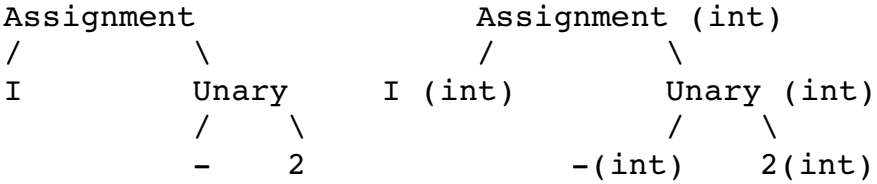
(a)



(b)

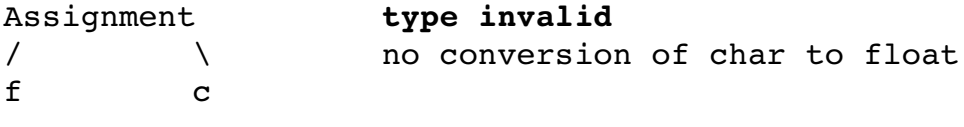


(c)

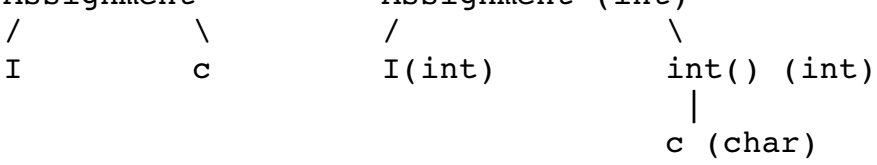


(d)

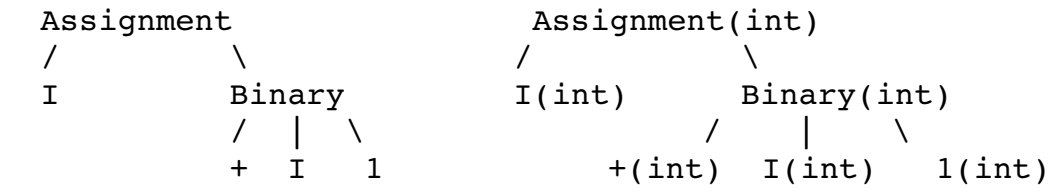
(e)



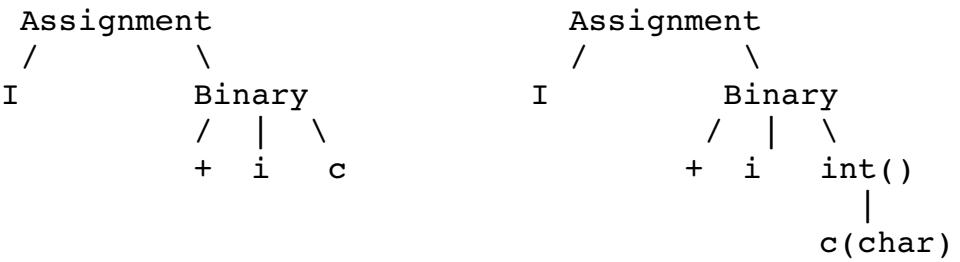
(f)



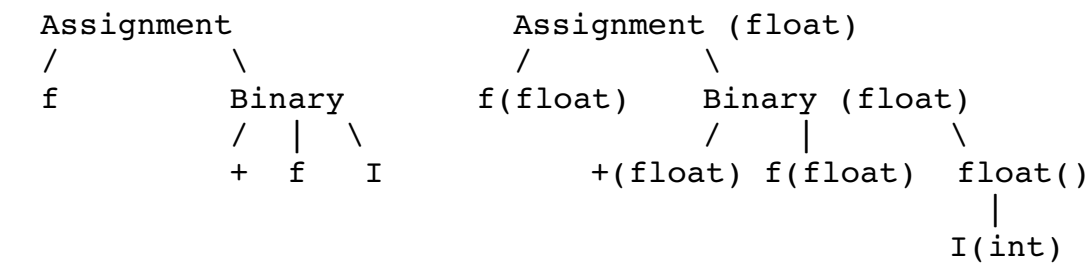
(g)



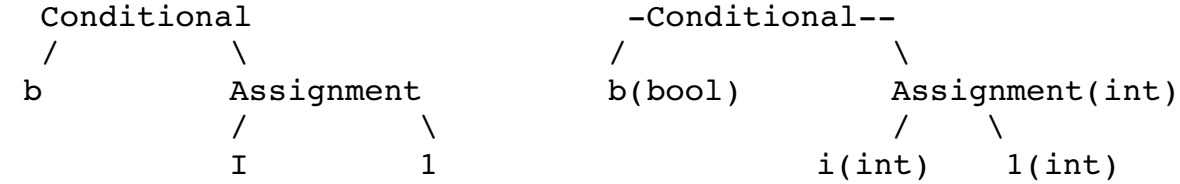
(h)



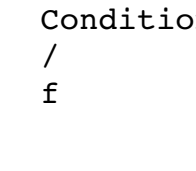
(I)



(j)

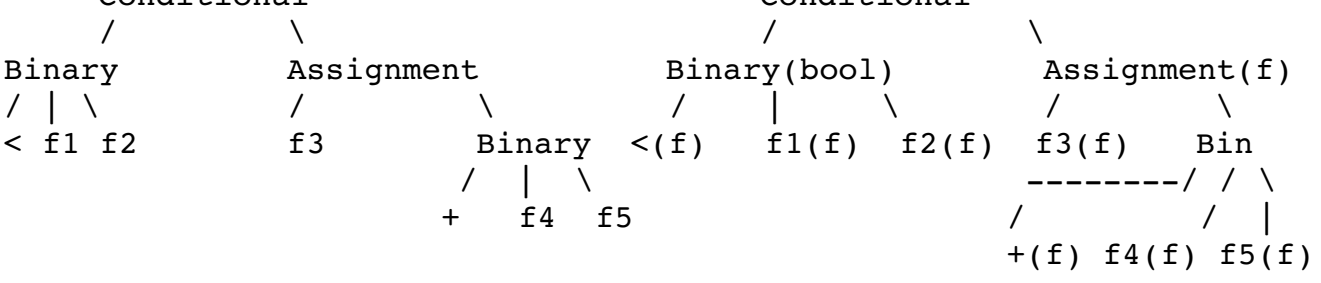


(k)



invalid type
conditional test must be bool

(l)



6.6
3

```
V: Expression x TypeMap → B
V(Expression e, TypeMap tm)
     = true                  if e is a Value
     = e `elem` tm           if e is a Variable
     = V(e.term1, tm) && V(e.term2, tm)
       && (typeOf(e.term1, tm) `elem` {float, int}
           && typeOf(e.term2, tm) `elem` {float, int})
       || (typeOf(e.term1, tm) `elem` {char, int}
           && typeOf(e.term2, tm) `elem` {char, int})
                   if e is a Binary && e.op `elem` {ArithmeticOp}
     = V(e.term1, tm) && V(e.term2, tm)
       && (typeOf(e.term1, tm) `elem` {float, int}
           && typeOf(e.term2, tm) `elem` {float, int})
       || (typeOf(e.term1, tm) `elem` {char, int}
           && typeOf(e.term2, tm) `elem` {char, int})
                   if e is a Binary && e.op `elem` {RelationalOp}

if (b.op.ArithmeticOp()) {
     check(((typ1 == Type.Int || typ1 == Type.Float)
            && (typ2 == Type.Int || typ2 == Type.Float))
           || ((typ1 == Type.Int || typ1 == Type.Char)
            && (typ2 == Type.Int || typ2 == Type.Char)),
           "type error for " + b.op);
     else if (b.op.RelationalOp()) {
          check(((typ1 == Type.Int || typ1 == Type.Float)
                 && (typ2 == Type.Int || typ2 == Type.Float))
                || ((typ1 == Type.Int || typ1 == Type.Char)
                 && (typ2 == Type.Int || typ2 == Type.Char)),
                "type error for " + b.op);
     }
}
```

6.10

I did this in 6.6 instead. I feel that if I have already done the formal modifications, the informal modifications would be busywork.

6.11

= V(e.index) if *e* is an *ArrayRef*

6.12

3 *If the Expression is an ArrayRef, then its result type is the type of that Value at the location* index *in the array*