1. When might you want the compiler to ignore type differences in an expression?
- Consider an expression that has an integer plus or minus a double. We would want the outcome to be a double. Thus, having the compiler ignore the type difference if this came is more bemeficial.

7. Describe a situation in which the add operator in a programming language would not be commutative.
- A case in which the add operator in a programming language is not commutative would be the concatenation of two strings. One sequence that two strings are concatnated differs than the other sequence they can be concatenate

8. Describe a situation in which the add operator in a programming language would not be associative.
- A scenario where summing integers together such that they result in an overflow or underflow would not be associative. The order in which the summation you do first, if there are more than two integers being summed, can change the output. For example, if the first two integers resulted in an overflow if summed first, would be different (if assuming the third integer is negative) rather if the second and third were summed first.

9. Assume the following rules of associativity and precedence for expressions:

B) a * (b - 1) / c mod d

$(((a * (b - 1)^{1})^{2} / c)^{3} \text{ mod } d)^{4}$

D. -a or c = d and e

$((-a)^{1} \text{ or } ((c = d)^{2} \text{and } e)^{3})^{4}$

F. -a + b

$(- (a + b)^{1})^{2}$

10. Show the order of evaluation of the expressions of Problem 9, assuming that there are no precedence rules and all operators associate right to left.

B) a * (b - 1) / c mod d

$(a * ((b - 1)^{1} / (c \text{ mod } d)^{2})^{3})^{4}$

D. -a or c = d and e

$(- (a \text{ or } (c = (d \text{ and } e)^{1})^{2})^{3})^{4}$

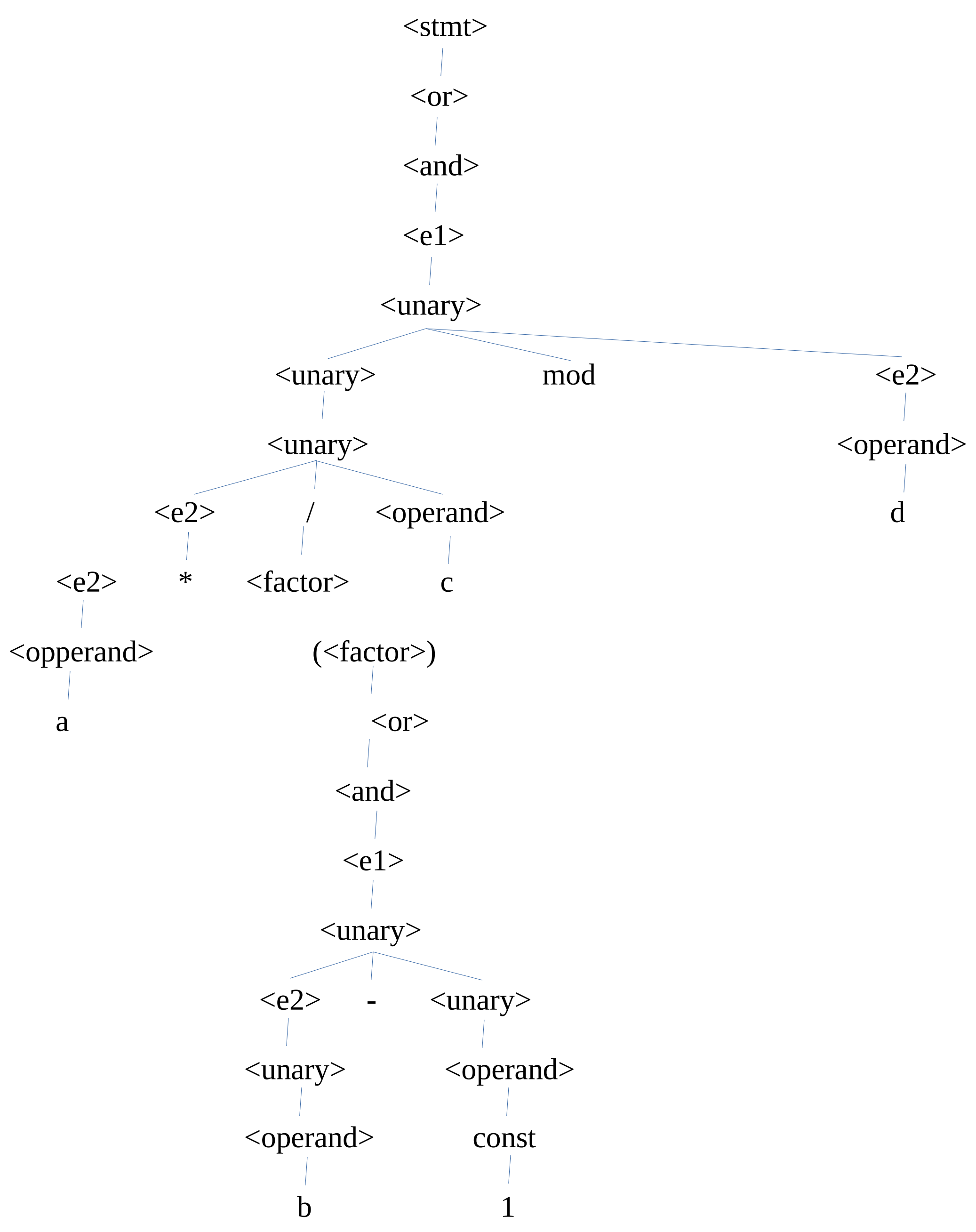
F. -a + b

$(- (a + b)^{1})^{2}$

11. Write a BNF description of the precedence and associativity rules defined for the expressions in Problem 9.
Assume the only operands are the names a,b,c,d, and e.
Lowest to Highest Precedence Operators In BDF description:

```
<stmt> :: <expression> or <or> | <expression> xor <or> | <or>
<or> :: <or> and <and> | <and>
<and> :: <and> = <e1> | <and> /= <e1> | <and> < <e1> | <and> <= <e1> | <and>  >= <e1> | <and> > e1>
              | <e1>
<e1> :: -<unary> | <unary>
<unary> :: <unary> + <e2> | <unary> - <e2> | <unary> & <e2> | <unary> mod <e2> | <e2>
<e2> :: <e2> * <factor> | <e2> / <factor> | not <factor> | <factor>
<factor> :: ( <e1> ) | <operand>
<operand> :: a | b | c | d | e | const | (<stmt>)
```

12. Using the grammar of Problem 11, draw parse trees for the expressions of Problem 9.
B) a * (b - 1) / c mod d


<stmt>
<or>
<and>
<e1>
<unary>
<unary>
mod
<e2>
<unary>
<operand>
<e2>
/
<operand>
d
<e2>
*
<factor>
c
<opperand>
(<factor>)
a
<or>
<and>
<e1>
<unary>
<e2>
-
<unary>
<unary>
<operand>
<operand>
const
b
1