When there are several operators, precedence and associativity determine how the operators are processed.

in which ^ is the exponentiation operator. Which subtraction and which exponentiation get evaluated first? Subtractions are processed left-to-right, meaning the result is 3. On the other hand, exponentiation is generally processed right-to-left, thereby reflecting the mathematical 2^{3^3} rather than $(2^3)^3$. Thus subtraction associates left-to-right, while exponentiation associates from right-to-left. All of these possibilities suggest that evaluating an expression such as

```
1 - 2 - 4 ^ 5 * 3 * 6 / 7 ^ 2 ^ 2
```

would be quite challenging.

If the calculations are performed in integer math (that is, rounding down on division), the answer is -8. To show this, parentheses are inserted to illustrate that the calculations are ordered:

```
( 1 - 2 ) - ( ( ( ( 4 ^ 5 ) * 3 ) * 6 ) / ( 7 ^ ( 2 ^ 2 ) ) )
```

Although the parentheses make the order of evaluations unambiguous, it is difficult to argue that they make the mechanism for evaluation any clearer. It turns out that a different expression form, called a *postfix expression*, provides a direct mechanism for evaluation. The following several sections explain how this works. The first one examines the postfix expression form and shows how postfix expressions can be evaluated in a simple left-to-right scan. The next section shows algorithmically how the previous expressions, which are presented as infix expressions, can be converted to postfix. Finally, a Java program is given that evaluates infix expressions containing additive, multiplicative, and exponentiation operators as well as overriding parentheses. An algorithm called *operator precedence parsing* is used.

11.2.1 Postfix Machines

A *postfix expression* can be evaluated as follows. Operands are pushed onto a single stack. An operator pops its operands and then pushes the result. At the end of the evaluation, the stack should contain exactly one element, which represents the result.

A *postfix expression* is a series of operators and operands. It is evaluated using a *postfix machine* as follows. When an operand is seen, it is pushed onto a stack. When an operator is seen, the appropriate number of operands are popped from the stack, the operator is evaluated, and the result is pushed back onto the stack. For binary operators, which are the most common, two operands are popped. When the complete postfix expression is evaluated, the result should be a single item on the stack that represents the answer. The postfix form represents a natural way to evaluate expressions because precedence rules are not required.

Here is a simple example. Consider the postfix expression

```
1 2 3 * +
```

The evaluation proceeds as follows: 1, then 2, and then 3 are each pushed onto the stack. To process *, we pop the top two items on the stack: 3 and then 2. Note that the first item popped becomes the rhs parameter to the binary operator, and the