```
                             CS 61B:  Lecture 40 Code
```

Here's code that performs one operation. For brevity, the division operator and all error checking are omitted.

```
  public static void operate(Stack numStack, char operator) throws
                                                          StackEmptyException {
    // Retrieve the operands from the stack.
    int operand2 = ((Integer) numStack.pop()).intValue();
    int operand1 = ((Integer) numStack.pop()).intValue();
    System.out.print(operator + " ");
    // Perform the operation and push the result on the number stack.
    switch (operator) {
    case '+':                                                     // Addition.
      numStack.push(new Integer(operand1 + operand2));
      break;
    case '-':                                                  // Subtraction.
      numStack.push(new Integer(operand1 - operand2));
      break;
    case '*':                                               // Multiplication.
      numStack.push(new Integer(operand1 * operand2));
      break;
    case '^':                                               // Exponentiation.
      int result = 1;
      for (int i = 0; i < operand2; i++) {
        result = result * operand1;
      }
      numStack.push(new Integer(result));
      break;
    }
  }
```

Here's code that reads a postfix expression (written on the command line), prints it, evaluates it, and prints the result. For instance, if you type

```
  Java ParsePostfix '73*2+'
```

The program responds

```
  7 3 * 2 +    result:23
```

```
public class ParsePostfix {

  public static void main(String[] args) throws StackEmptyException {
    LinkedStack numStack = new LinkedStack();         // Create a number stack.
    // Process each character of the first command-line argument.
    for (int i = 0; i < args[0].length(); i++) {
      char c = args[0].charAt(i);
      if (Character.isDigit(c)) {
        // Print each digit; push each digit on number stack.
        System.out.print(c + " ");
        numStack.push(new Integer(Character.digit(c, 36)));
      } else if ((c == '+') || (c == '-') || (c == '*') || (c == '^')) {
        operate(numStack, c);
      }
    }
    System.out.println("   result:" + numStack.top());
  }

  // Put the operate() method code here.
}
```

The following method converts an infix expression to postfix and simultaneously evaluates the postfix expression. It prints both the converted postfix expression and the computed result. (Error checking is omitted. The reading of input is also poor; only one-digit numbers can be input.)

```
  public static void main(String[] args) throws StackEmptyException {
    LinkedStack numStack = new LinkedStack();         // Create a number stack.
    LinkedStack opStack = new LinkedStack();        // Create an operand stack.
    // Process each character of the first command-line argument.
    for (int i = 0; i < args[0].length(); i++) {
      char c = args[0].charAt(i);
      if (Character.isDigit(c)) {
        // Print each digit; push each digit on number stack.
        System.out.print(c + " ");
        numStack.push(new Integer(Character.digit(c, 36)));
      } else if ((c == '+') || (c == '-') || (c == '*') || (c == '^')) {
        // Before pushing an operator onto the operand stack, check if
        //   operators currently atop the stack have greater precedence.
        while (!opStack.isEmpty() && (c != '^') &&
               (precedence(((Character) opStack.top()).charValue()) >=
                precedence(c))) {
          // Perform the operation atop the operand stack.
          char operator = ((Character) opStack.pop()).charValue();
          operate(numStack, operator);
        }
        opStack.push(new Character(c)); // Push new operation on operand stack.
      }
    }
    while (!opStack.isEmpty()) {
      char operator = ((Character) opStack.pop()).charValue();
      operate(numStack, operator);
    }
    System.out.println("   result:" + numStack.top());
  }
```

The algorithm requires us to assign each operator a number that indicates its relative precedence. We could easily add more operators later.

```
  public static int precedence(char operator) {
    switch (operator) {
    case '^':
      return 3;
    case '*':
    case '/':
      return 2;
    case '+':
    case '-':
      return 1;
    default:
      return 0;
    }
  }
```