```
    1: import java.util.ArrayList;
    2: import java.util.Arrays;
    3: import java.util.Scanner;
    4: import java.util.Stack;
    5:
    6:
    7: public class Screen {
    8:         public static char[][] formula;
    9:
   10:         public static String buildString(int row, int start, int end) {
   11:                 String result = "";
   12:                 for (int i = start; i < end && i < formula[row].length; i++) {
   13:                         result += formula[row][i];
   14:                 }
   15:                 return result.trim();
   16:         }
   17:
   18:         public static int toInt(int row, int start, int end) {
   19:                 int result = 0;
   20:                 for (int i = start; i < end; i++) {
   21:                         result *= 10;
   22:                         result += formula[row][i] - '0';
   23:                 }
   24:                 return result;
   25:         }
   26:
   27:         public static int resolveSimple(int row, int start, int end) {
   28:                 String[] parts = buildString(row, start, end).split(" ");
   29:                 Stack<String> newParts = new Stack<String>();
   30:
   31:                 for (int i = 0; i < parts.length; i++) {
   32:                         if (parts[i].equals("*")) {
   33:                                 newParts.push(String.valueOf(Integer.valueOf(newParts.pop()) *
Integer.valueOf(parts[i+1])));
   34:                                 i++;
   35:                         } else {
   36:                                 newParts.push(parts[i]);
   37:                         }
   38:                 }
   39:
   40:                 int result = Integer.valueOf(newParts.get(0));
   41:                 int i = 1;
   42:                 while (i < newParts.size()) {
   43:                         if (newParts.get(i).equals("+"))
   44:                                 result += Integer.valueOf(newParts.get(i+1));
   45:                         else if (newParts.get(i).equals("-"))
   46:                                 result -= Integer.valueOf(newParts.get(i+1));
   47:
```

```
48:                         i += 2;
49:                 }
50:                 return result;
51:         }
52:
53:         public static int resolveSimple(int row, int col) {
54:                 int length = complexLength(row, col);
55:                 return resolveSimple(row, col, col+length);
56:         }
57:
58:         public static int resolveSqrt(int row, int col) {
59:                 int length = complexLength(row, col);
60:                 return (int) Math.sqrt(resolveSimple(row, col + 2, col + length));
61:         }
62:
63:         public static int resolveFraction(int row, int col) {
64:                 int length = complexLength(row, col);
65:                 return (int) (resolveSimple(row - 1, col, col + length) / resolveSimple(row + 1, col, col + length));
66:         }
67:
68:         public static int resolveComplex(int row, int col) {
69:                 switch(formula[row][col]) {
70:                 case '\\':
71:                         return resolveSqrt(row, col);
72:                 case '=':
73:                         return resolveFraction(row, col);
74:                 default:
75:                         return resolveSimple(row, col);
76:                 }
77:         }
78:
79:         public static int complexLength(int row, int col) {
80:                 int length = 0;
81:                 switch(formula[row][col]) {
82:                 case '\\':
83:                         while (col + 2 + length < formula[row - 1].length && formula[row - 1][col + 2 + length] == '_')
84:                                 length++;
85:                         length += 2;
86:                         break;
87:                 case '=':
88:                         while (col + length < formula[row].length && formula[row][col + length] == '=')
89:                                 length++;
90:                         break;
91:                 default:
92:                         while (col + length < formula[row].length && "+-*0123456789 "
.contains(String.valueOf(formula[row][col + length])))
93:                                 length++;
94:                         while ("+-* ".contains(String.valueOf(formula[row][col + length - 1])))
```

```
95:                                 length--;
96:                         break;
97:                 }
98:                 return length;
99:         }
100:
101:        public static void main(String[] args) {
102:                Scanner s = new Scanner(System.in);
103:                while (true) {
104:                        int r = s.nextInt();
105:                        int c = s.nextInt();
106:                        if (r == 0 && c == 0)
107:                                break;
108:
109:                        s.nextLine();
110:
111:                        formula = new char[r][c];
112:                        for (int i = 0; i < r; i++)
113:                                formula[i] = s.nextLine().toCharArray();
114:
115:                        int result = 0;
116:
117:                        if (r == 1)
118:                                result = resolveSimple(0, 0);
119:                        else {
120:                                int i = 0;
121:                                while (i < formula[1].length) {
122:                                        if (i == 0) {
123:                                                result = resolveComplex(1, 0);
124:                                                i += complexLength(1, 0);
125:                                        } else if (formula[1][i + 1] == '+') {
126:                                                result += resolveComplex(1, i+3);
127:                                                i += complexLength(1, i+3) + 3;
128:                                        } else if (formula[1][i + 1] == '-') {
129:                                                result -= resolveComplex(1, i+3);
130:                                                i += complexLength(1, i+3) + 3;
131:                                        }
132:
133:                                        //System.out.println(i);
134:
135:                                }
136:                        }
137:
138:                        System.out.println(result);
139:                }
140:
141:                s.close();
142:        }
```

```
143: }
```