Bio-Informatik
Beispiele

Helge Janicke

22. März 2002

Inhaltsverzeichnis

1 Postfix Notation

Die gebräuchliche Schreibweise für Formeln ist die Infix Notation. Hierbei steht der Operator einer (binären) Operation zwischen den Operanden.

Beispiel:
`1 + 2 - ( 3 - 4 ) = 4`

Bei dieser Art der Schreibweise ist die Klammerung zwingend, denn ohne Angabe der Klammern würde die implizite Klammerung gelten.

Beispiel:
`1 + 2 - 3 - 4 = -4`

Bei der Postfix Notation wird der Operator nach den Operanden geschrieben. Diese Art der Schreibweise kommt ohne Klammern aus.

Beispiel:

```
1 + 2 - ( 3 - 4 ) = 4   // infix   notation
1 2 + 3 4 - -     = 4   // postfix notation
```

Die Auswertung eines Ausdrucks erfolgt streng von links nach rechts. Sobald ein Operator (z.B. +) gefunden wird, werden die dazugehörigen Operanden (1 und 2) durch das Ergebnis der Operation ersetzt (`1 + 2 = 3`).
Für die weitere Auswertung gilt dann:

`3 3 4 - - = 4`
usw.

1.1 Auswertung mittels eines Stacks (Stapel)

Für die Auswertung solcher Ausdrücke ist ein Stack eine geeignete Datenstruktur. Sie ist zu vergleichen mit einem Stapel Bierdeckeln. Ein Stack besitzt folgende Methoden:

push - drauflegen Legt ein Datum (Bierdeckel) auf den Stapel.

pop - herunternehmen Nimmt das oberste Datum vom Stapel.

peek - anschauen Zeigt das oberste Datum vom Stapel

Der Stack ist ein typischer LIFO (Last In First Out) Speicher. Man kann bei einem Stack nur auf das oberste Datum zugreifen.

Darstellung des Algorithmus:

Auswertung einer einfachen Formel in Postfix Notation — `Ergebnis der Berechnung für einzelne Ziffern`

<table>
<tr><td colspan="4">Lege neuen Stack an</td></tr>
<tr><td colspan="4">Für jedes Zeichen z in der Eingabe</td></tr>
<tr><td></td><td colspan="3">z</td></tr>
<tr><td></td><td>'0'...'9'</td><td>'+'</td><td>'-'</td></tr>
<tr><td></td><td rowspan="3">push c als Zahl</td><td>rv = pop()</td><td>rv = pop()</td></tr>
<tr><td></td><td>lv = pop()</td><td>lv = pop()</td></tr>
<tr><td></td><td>push(lv+rv)</td><td>push(lv-rv)</td></tr>
<tr><td colspan="4">Ergebnis ist gleich pop()</td></tr>
</table>

Grob umgesetzt in Java-Code sieht das Struktogramm dann so aus:

```
1  // Cannot compile this class.
2  // replace comments by valid code.
3
4  public class EntwurfDigit {
5
6      public static String eval(String formula) {
7          int result;
8          for (int i=0; i</* length of String formular */; i++) {
9              char c= // character at index i in formula
10             switch (c) {
11                 case '0':
12                 case '1':
13                 case '2':
14                 case '3':
15                 case '4':
16                 case '5':
17                 case '6':
18                 case '7':
19                 case '8':
20                 case '9':
21                     // push c
22                     break;
23                 case '+':
24                     // rv = pop()
25                     // lv = pop()
26                     // push(lv + rv)
27                     break;
```

```
28                  case '-':
29                      // rv = pop()
30                      // lv = pop()
31                      // push(lv - rv)
32                      break;
33                  default:
34                      // unknown symbol
35              }
36          }
37          result = // pop()
38          return result;
39      }
40
41
42      public static void main(String[] args) {
43
44      }
45
46  }
```

EntwurfDigit.java

Benötigt werden also noch folgende Methoden:

- Einen Stack `stack` erzeugen
  `java.util.Stack stack = new java.util.Stack()`
  Ein Stack ist ein Objekt, daß durch die Klasse *Stack* im Paket *java.util* definiert ist.
  - `Object pop()`
    [1] Liefert das oberste Objekt.
  - `void push(Object o)`
    Legt `o` auf den Stack.
- Länge eines Strings `s`
  `int len = s.length()`
- Zeichen (char) c an der Stelle i eines Strings `s`
  `char c = s.charAt(i)`
- Ausgabe auf der Konsole:
  `System.out.println("hello world")`

Mit Hilfe dieser Methoden kann der Code-Entwurf vervollständigt werden:

```
1  // author : hj
2  // date   : 3/19/2002
3  // purpose: demonstration of postfix notation.
4
5
```

[1] Alle Operationen sind für Objekte definiert, d.h. primitive Typen müssen in Objekte verpackt werden. Die Objekte müssen beim runternehmen wieder in der richtigen Typ gecastet werden

```
import java.lang.*;


/** Postfix evaluation for + and -.
 *  Returns the result of a simple formula (eg. "12+" -> 3 or
 *  "123++" -> 6).
 *  Digits are treated as seperate numbers. 12 is x1=1, x2=2.
 */
public class Digit {

    /** Postfix evaluation for + and -.
     *  Returns the result of a simple formula (eg. "12+" -> 3 or
     *  "123++" -> 6).
     *  Digits are treated as seperate numbers. 12 is x1=1, x2=2.
     */
    public static Object eval(String formula) {
        // stores the right operand for evaluation
        int rv;

        // stores the left operand for evaluation
        int lv;

        // the current processed character
        char c;

        // create a new stack for evaluation
        java.util.Stack stack = new java.util.Stack();

        for (int i=0; i<formula.length(); i++) {

            // character at index i in formula
            c = formula.charAt(i);

            // handle different cases
            switch (c) {
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9':
                    // for all digits push c
                    stack.push(new Integer("" + c));
                    break;
                case '+':
                    // pop operands and push sum
                    rv = ((Integer) stack.pop()).intValue();
                    lv = ((Integer) stack.pop()).intValue();
                    stack.push(new Integer(lv + rv));
                    break;
```

```
60                  case '-':
61                      // pop operands and push difference
62                      rv = ((Integer) stack.pop()).intValue();
63                      lv = ((Integer) stack.pop()).intValue();
64                      stack.push(new Integer(lv - rv));
65                      break;
66                  default:
67                      // unknown symbol
68                      System.err.println("unknown Symbol \'" + c + "\'");
69              }
70          }
71
72          // result is the last item on stack
73          return stack.pop();
74
75      }// end of eval(String formula)
76
77
78      /** Main program.
79       *  Needs one commandline argument. String containing the formula to
80       *  evaluate.
81       */
82      public static void main(String[] args) {
83          // print the result of first argument passed to Digit
84          System.out.println(args[0] + " results in " + eval(args[0]));
85      }
86
87  }// end of Digit.class
```

Digit.java