Prof. Dr. Franz Schweiggert | Abteilung Angewandte Informationsverarbeitung | 04.05.2006
Norbert Heidenbluth | Blatt 2

Allgemeine Informatik II (SS 2006)

für
(Wirtschafts-)Mathematiker/Physiker und E-Techniker

Abgabetermin: 11. Mai 2006

Nachdem wir im ersten Übungsblatt den Datentyp "Stack" kennengelernt haben, folgen in dieser Woche Aufgaben zu dessen Verwendung, d.h. daß beide Aufgaben dieses Blattes unter Verwendung eines Stacks gelöst werden sollen. Darüberhinaus beinhaltet Aufgabe 5 nochmals das Interface-Konzept, und in Aufgabe 6 steht nochmals der algorithmische Teil im Vordergrund.

Reverse Polish Notation (RPN) – 15 Punkte

Im Elektro-Fachmarkt Ihres Vertrauens erregte unter all den Taschenrechnern *dieses Modell* Ihre besondere Aufmerksamkeit. Bei genauerer Betrachtung seiner technischen Daten fiel Ihnen auf, daß er die **RPN** beherrscht, also die "Reverse Polish Notation".

Zwar behaupten in diesem Elektro-Markt alle, nicht blöd zu sein, aber eine Erklärung, was es mit der RPN auf sich hat, blieb man Ihnen dort schuldig.

Also: Die umgekehrte polnische Notation wurde 1920 von Jan Lukasiewicz erfunden, um mathematische Ausdrücke ohne Vorrangregeln (Klammern) schreiben zu können. Einerseits erleichtert das die Eingabe in Taschenrechner bzw. Computer, andererseits bekommt man beim Ausrechnen des eingegebenen Ausdrucks "automatisch" Zwischenergebnisse geliefert und kann so überprüfen, ob die Berechnung fehlerfrei abläuft.

Wie sieht aber nun diese Notation aus? Ganz einfach: der zu berechnende Ausdruck wird so umgestellt, daß jeweils jeder Operator seinem/seinen Operanden folgt. Statt `1 + 2` (gewöhnliche Notation) heißt es in RPN: `1 2 +`. Auf diese Art werden auch komplexere Ausdrücke geschrieben: `((1+2)*3)!` wird zu `1 2 + 3 * !`, und `(1+2*3)!` wird zu `1 2 3 * + !`. Man sieht, daß also keine Klammern mehr benötigt werden.

Begriffe: Unsere gewohnte Notation wird im Sprachgebrauch als *Infix*-Notation bezeichnet, die RPN nennt man hingegen *Postfix*-Notation.

Aufgabe 5: Ein RPN-Rechner (6 Punkte)

Ein Rechner, der nach der RPN arbeitet, kann zum Beispiel unter Verwendung eines Stacks wie folgt implementiert werden:

Der eingegebene Ausdruck wird einerseits dahingegen gerprüft, daß er nur erlaubte Zeichen enthält, und gleichzeitig wird dabei wie folgt zwischen Operanden und Operatoren unterschieden:

- Wird ein *Operand* gelesen, so wird er auf den Stack gelegt.
- Wird ein *Operator* gelesen, so holt er sich die Anzahl von Operanden, die er benötigt, vom Stack[1] und führt die Berechnung durch. Das Ergebnis dieser Berechnung wird dann seinerseits wieder auf den Stack gelegt.

Und damit sind wir bereits bei der folgenden Aufgabe:

> Schreiben Sie ein Java-Programm, das (nacheinander) beliebig viele Eingabezeilen in der RPN einliest, auswertet/berechnet und am Ende jeder Eingabezeile den Inhalt des Stacks ausgibt.

Ihr Programm soll dabei die folgenden Anforderungen erfüllen:

- Es muß die vier Grundrechenarten ($+$, $-$, $\cdot$ und $/$) sowie die Fakultät (!) beherrschen.
- Die vorstehend genannten Rechenoperationen werden **nicht** im Programm selber durchgeführt, sondern jeweils einzeln von einer Klasse erledigt. Diese Klassen müssen das Interface *CalcMethod.java* implementieren.
- Ungültige Operatoren werden überlesen (und mit einer Meldung kommentiert); gleiches gilt, wenn zu wenig Operanden auf dem Stack liegen, um eine gewünschte Operation durchzuführen.
- Durch die Verwendung des Datentyps `int` erfolgt die Division stets ganzzahlig. Diese Vereinfachung nehmen wir für dieses Blatt (ausnahmsweise) in Kauf.
- Achten Sie darauf, daß mehrstellige Zahlen (z.B. 123) auch als solche verarbeitet werden und nicht wie drei einzelne Ziffern (1, 2 und 3).

Beispiel:

```
theseus$ java PostfixCalc
Calculator> 12 11 2 * 6 2 / - +
31
Calculator> 12 11 2 6 - * + 2 /
-16
Calculator> 12 +
Stack underflow!
12
```

[1] Es gibt unäre und binäre Operatoren, die also genau einen bzw. genau zwei Operanden benötigen.

Aufgabe 6: Infix → Postfix (9 Punkte)

Natürlich möchten Sie Ihren Rechner aus Aufgabe 5 umfangreich testen, andererseits ist für Sie die RPN ja doch irgendwie ungewohnt. Da wäre es doch schön, einen Konverter zu haben, gell? Und schon sind wir bei der nächsten Aufgabe:

> Schreiben Sie ein Java-Programm, das einen Ausdruck in Infix-Notation einliest und in Postfix-Notation umgewandelt wieder ausgibt.

Den folgenden Anforderungen sollte dieser Konverter genügen:

- Er soll selbständig (d.h. ohne Klammerung) die Vorrangregel "Punkt-vor-Strich" beachten,
- gleichzeitig aber auch (runde) Klammern berücksichtigen, wenn diese eingegeben wurden.
- Es genügt, wenn er die vier Grundrechenarten beherrscht.

Beispiel:

```
theseus$ java InfixToPostfix
Infix: 12 + 11 * 2 - 6 / 2
Postfix: 12 11 2 * 6 2 / - +
Infix: (12 + 11 * (2 - 6)) / 2
Postfix: 12 11 2 6 - * + 2 /
```

Tips zum Vorgehen:

- Auch bei dieser Aufgabe ist die Verwendung eines Stacks zu empfehlen.
- Von der "technischen Seite" her, ist diese Aufgabe sehr ähnlich zur Aufgabe 5, der wesentliche Unterschied liegt jedoch im zu verwendenden Algorithmus.
- Es empfiehlt sich daher, sich zunächst Gedanken über den notwendigen Algorithmus zu machen. Dazu eine (nicht vollständige) Anregung:
  1. Der eingegebene Ausdruck wird von links nach rechts zeichenweise "abgearbeitet".
  2. Operanden werden sofort ausgegeben, ebenso wird der erste gelesene Operator auf den Stack gelegt.
  3. Für jeden weiteren gelesenen Operator gilt: Wenn der oberste Operator auf dem Stack Vorrang gegenüber dem aktuell gelesenen hat, so nimm diesen obersten Operator vom Stack und gib ihn aus. Lege *andernfalls* den gelesenen Operator auf den Stack.
  4. Wiederhole Schritt 3 solange, wie der Stack nicht leer ist und der Vorrang des jeweils obersten Stack-Operatoren entsprechend höher ist als der des gerade gelesenen Operators.
  5. Sind alle Zeichen eingelesen bzw. bearbeitet worden, so gib den Rest des Stacks aus.
- Dieser Vorschlag für einen Algorithmus beachtet die "Punkt-vor-Strich"-Regel, indem Sie entsprechende Vorrang-Werte für die Operatoren definieren und beim Programmablauf überprüfen lassen. Allerdings ist die Beachtung der Klammerung noch `nicht` in diesen Vorschlag eingearbeitet. Das dürfen Sie machen ;-)
- Wie in Aufgabe 5, ist es auch hier wieder wichtig, daß die Zahl 123 nicht als drei einzelne Ziffern interpretiert wird.

Viel Erfolg!