Ein Ausflug in den Compilerbau

Syntaktische Analyse und Berechnung arithmetischer Ausdrücke

von Torsten Brandes

Der Compilerbau ist ein (aufgrund der komplexen Materie) in der Schule vernachlässigtes Teilgebiet der praktischen Informatik, obwohl sich viele Berührungspunkte zu anderen Themen aufzeigen lassen: Zur *theoretischen Informatik* (insbesondere zu den formalen Sprachen und Grammatiken), zur *Mathematik* und schließlich zu *Datenstrukturen* (insbesondere zu Listen, Stapeln und Bäumen).

Die Unterrichtsreihe im Überblick

Ausgangspunkt der Unterrichtsreihe war ein in JAVA programmierter Taschenrechner, der im Rahmen einer Reihe zur Algorithmik an eine Mathematikbibliothek angebunden war. Die Schülerinnen und Schüler hatten darin einige Algorithmen (Quadratwurzel, trigonometrische Funktionen, Exponentialfunktion etc.) numerisch oder rekursiv implementiert. Beim Vergleich mit einem realen Taschenrechner kam dann die Frage auf, wie man die Klammerrechnung realisieren könne. Meinen Einwand, das sei gar nicht so einfach, ließen die Schüler nicht gelten, sodass ich mir diesbezüglich Gedanken machen musste.

Im Folgenden wird zunächst der Aufbau (vereinfachter) arithmetischer Ausdrücke untersucht und durch Regeln beschrieben. Es erweist sich als günstig, den Begriff der *formalen Sprache* und der (hier: kontextfreien) *Grammatik* einzuführen. Zur Syntaxprüfung wird das Verfahren des *rekursiven Abstiegs* erarbeitet und in JAVA implementiert. Zwecks Auswertung arithmetischer Ausdrücke werden diese zunächst in Postfix-Notation übersetzt, um sie klammerfrei zu machen und dann mithilfe eines Kellerspeichers berechnet.

Bezug zu Bildungsstandards

In den *Bildungsstandards Informatik* ist zunächst der Inhaltsbereich *Sprachen und Automaten* zuständig. Dort werden sehr einfache formale Sprachen (z. B. die Sprache der E-Mail-Adressen) betrachtet und (im Zusammenhang mit endlichen Automaten) Verfahren zur Syntaxprüfung erarbeitet. Die – auf die Sekundarstufe II angepassten – Kompetenzen würden lauten (vgl. AKBSI, 2008, S. 34):

Schülerinnen und Schüler

- ▷ überprüfen vorgegebene arithmetische Ausdrücke auf Korrektheit,
- ▷ nutzen formale Sprachen zur Darstellung von Ableitungsregeln.

Als weitere Inhaltsbereiche sind *Algorithmen* einerseits und *Informatiksysteme* andererseits zu nennen:

Schülerinnen und Schüler

- ▷ entwerfen und realisieren Programme mit den algorithmischen Grundbausteinen sowie Prozeduren bzw. Methoden (vgl. AKBSI, 2008, S. 30),
- ▷ verstehen die Grundlagen des Aufbaus von Informatiksystemen und deren Funktionsweise, insbesondere von Übersetzern (vgl. AKBSI, 2008, S. 37).

Die Syntax arithmetischer Ausdrücke ist den Schülern vom Mathematikunterricht her in Gestalt sogenannter „Rechenbäume“ vertraut. Aus dem Deutschunterricht kennen die Schüler einfache Verfahren zur Satzanalyse (siehe Bild 1). Das heißt, dass zusätzlich

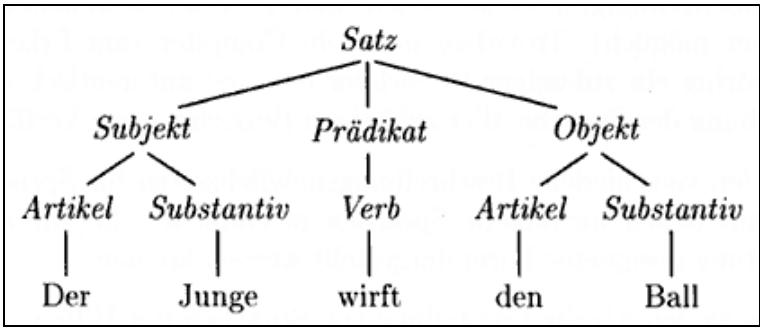

Bild 1: Syntaxbaum eines gemäß (R1) bis (R4) konstruierten Satzes.

auch die Bildungsstandards anderer Fächer betroffen sind.

Die in den Unterrichtsverlauf eingestreuten Aufgaben erfassen folgende Prozessbereiche:

- ▷ *Modellieren und Implementieren* in Aufgaben 1.4, 2.1, 2.2, 2.4 (JAVA-Programme),
- ▷ *Strukturieren und Vernetzen* in Aufgabe 1.3 (Strukturierung mittels Syntaxbaum),
- ▷ *Kommunizieren und Kooperieren* in Aufgabe 1.1 (Erstellung eines Referats),
- ▷ *Darstellen und Interpretieren* in Aufgaben 1.2, 1.4 (Darstellung von Grammatikregeln).

Syntaxprüfung als Aufgabe eines Compilers

Aufgabe 1.1: Informieren Sie sich in der Literatur (z. B. Informatik-Duden oder im Internet) zu folgenden Begriffen: Übersetzer (compiler), Quellsprache (source language), Zielsprache (object language), lexikalische Analyse (scanner), syntaktische Analyse (parser). Schreiben Sie einen kurzen Bericht (Überblicks-Referat) für Ihre Mitschüler(innen), der keine weiteren Vorkenntnisse als das aus dem Unterricht Bekannte voraussetzt.

Arithmetische Ausdrücke

Zunächst geht es darum, die Struktur eines arithmetischen Ausdrucks zu erkennen und durch Regeln zu beschreiben, um diese Regeln dann in einen Algorithmus zur Syntaxprüfung zu übersetzen. Dabei kann auf die Vorkenntnisse aus dem Mathematikunterricht zurückgegriffen werden.

Aufgabe 1.2: Ein syntaktisch korrekter arithmetischer Ausdruck ist z. B. $(2 + 7) \cdot (4 + 3)$. Nicht korrekt dagegen ist etwa $2 + + 7$ (). Die Bestandteile eines solchen Ausdrucks sind erstens Operanden (hier: einstellige Zahlen bzw. Ziffern), zweitens Operatoren (Plus- und Malzeichen), drittens Klammern.

(a) Geben Sie drei weitere korrekte arithmetische Ausdrücke an.
(b) Schreiben Sie Regeln zur Bildung korrekter arithmetischer Ausdrücke auf.

Beispiel: Die Anzahl der öffnenden Klammern muss mit der Anzahl der schließenden Klammern übereinstimmen.

Es stellt sich heraus, dass der Aufbau arithmetischer Ausdrücke einem Rekursionsmuster folgt und komplexer ist, als es zunächst aussieht. Bestrebungen der Schüler, die syntaktische Korrektheit mittels Fallunterscheidungen zu prüfen, lassen sich schnell mit etwas größeren Beispielen unterlaufen, etwa mit
$2 \cdot (3 + (2 \cdot (2 + (4 + 6))) + (4 + (4 + 7)))$.
Man benötigt also ein strukturiertes Vorgehen; dazu ist es nötig, auf den Begriff der formalen Sprache einzugehen.

Sprachen und Grammatiken

Es bietet sich an, an das Vorwissen der Schüler aus dem Deutsch- oder Fremdsprachenunterricht anzuknüpfen. Von daher sind ihnen Grammatiken als Regelwerke bekannt, die die Struktur von Sätzen definieren. Beispielsweise kennen wir in der deutschen Sprache die Regel: „Ein Satz besteht aus einem Subjekt, dem sich ein Prädikat und ein Objekt anschließen." Diese Regel lässt sich wie folgt kurz aufschreiben (siehe auch Bild 1, vorige Seite):

(R1) Satz → Subjekt Prädikat Objekt

Der Satz: „Der Junge wirft den Ball" hat genau diese Struktur, wobei „Der Junge" das Subjekt ist, „wirft" das Prädikat, und „den Ball" das Objekt.

Die Bestandteile des Satzes können ihrerseits eine Struktur aufweisen, so wird z. B. die Struktur des Subjekts durch die folgende Grammatikregel beschrieben:

(R2) Subjekt → Artikel Substantiv

Dazu muss noch festgelegt werden, was ein Artikel und was ein Substantiv ist; dies geschieht mittels folgender Regeln:

(R3) Artikel → Der
(R4) Substantiv → Junge

Bild 1 (vorige Seite) gibt eine grafische Darstellung der grammatischen Struktur unseres Beispielsatzes.

Natürlich gibt es für Sätze und Teilstrukturen noch andere zulässige Formen, die in einer vollständigen Grammatik alle durch entsprechende Regeln erfasst werden müssen. Die nicht mehr zergliederbaren Bestandteile eines Satzes heißen *Terminalsymbole*; sie stehen üblicherweise in einer großen Liste, in der der gesamte Wortschatz einer Sprache zusammengefasst ist, im Deutschen etwa der Duden.

Grammatiken beschreiben im Wesentlichen nur den syntaktischen Aufbau von Sätzen. Ob ein Satz als sinnvoll erachtet wird, ist eine andere Frage. Beispielsweise stoßen wir uns bei dem Satz: „Der Ball wirft den Jungen" nicht an seiner syntaktischen Form, sondern an seinem Inhalt, was ein semantisches Problem darstellt. Die Behandlung semantischer Aspekte werden wir hier außer Acht lassen, da dies nicht Thema der Theorie der formalen Sprachen und Automaten ist.

Eine Grammatik für arithmetische Ausdrücke

Auch die arithmetischen Ausdrücke können als Sätze einer Sprache aufgefasst werden. Ähnlich wie bei den oben angegebenen umgangssprachlichen Sätzen können wir sie durch gewisse Ersetzungsregeln analysie-

ren. Für die Sprache der arithmetischen Ausdrücke gebe ich den Schülern folgende Grammatik (System von Ersetzungsregeln) vor:

Ausdruck	→	Summand | Summand + Ausdruck
Summand	→	Faktor | Faktor · Summand
Faktor	→	Ziffer | (Ausdruck)
Ziffer	→	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Man beachte, dass die Sprache zunächst nur Ziffern und als Operatoren nur das Pluszeichen und das Malzeichen enthält.

Um zu überprüfen, ob ein gegebener arithmetischer Ausdruck syntaktisch korrekt ist, muss man ihn mithilfe der Ersetzungsregeln herleiten können. Eine Herleitung des Ausdrucks 3 + 5 · 7 sieht wie folgt aus:

$A \Rightarrow S + A \Rightarrow F + A \Rightarrow 3 + A \Rightarrow 3 + S \Rightarrow 3 + F \cdot S \Rightarrow 3 + 5 \cdot S \Rightarrow 3 + 5 \cdot F \Rightarrow 3 + 5 \cdot 7.$

A (für Ausdruck), S (für Summand), F (für Faktor), Z (für Ziffer) sind die sogenannten *Nicht-Terminalsymbole*; sie dienen als eine Art Platzhalter für die Terminalsymbole (Ziffern und Rechenzeichen). Nur diese – als die „endgültigen" Symbole – kommen in dem erfolgreich hergeleiteten arithmetischen Ausdruck vor.

Man kann die Schüler auch einen Syntaxbaum (entsprechend Bild 1, S. 59) zeichnen lassen. Für jedes Wort einer formalen Sprache lässt sich dieser eindeutig angeben. Von links nach rechts kann der der abgeleitete Ausdruck dann abgelesen werden.

Aufgabe 1.3: Leiten Sie den Ausdruck (a) (3 + 5) · 7, (b) (2 + 7) · (4 + 3) her und zeichnen Sie jeweils den Syntaxbaum (in der Mathematik „Rechenbaum" genannt).

Das Verfahren des rekursiven Abstiegs

Mithilfe der Grammatik lässt sich nun ein Parser (Programm zur Syntaxanalyse) implementieren. Zu jedem Nicht-Terminalsymbol gehört eine rekursive Prozedur, die die Wirkungsweise der Ersetzungsregel unmittelbar ausdrückt. Die Prozedur prüfeAusdruck() lautet, entsprechend der ersten Regel in obiger Grammatik, wie folgt:

```
static void prüfeAusdruck() {
  // Ausdruck -> Summand | Summand + Ausdruck
  prüfeSummand();
  if (eingabewort.charAt(i) == '+')
    {i++; prüfeAusdruck();}
  } // Ende prüfeAusdruck
```

Das zu untersuchende Eingabewort wird zeichenweise durchgegangen, und je nach Eingabezeichen weitere Prozeduren aufgerufen. Ist kein Fehler aufgetreten, wird zunächst die Prozedur prüfeSummand() aufgerufen:

```
static void prüfeSummand() {
  // Summand -> Faktor | Faktor * Summand
  prüfeFaktor();
  if (eingabewort.charAt(i) == '*')
    {i++; prüfeSummand();}
  } // Ende prüfeSummand
```

Sie ruft ihrerseits prüfeFaktor() auf:

```
static void prüfeFaktor() {
  // Faktor -> Ziffer | (Ausdruck)
  if (eingabewort.charAt(i) == '(') {
    i++; prüfeAusdruck();
    if (eingabewort.charAt(i) == ')')
      i++;
    else {
      System.out.println("
          Schliessende Klammer erwartet.");
      System.exit(1);
      } // Ende else
    } // Ende if
  else if (Character.isDigit
          (eingabewort.charAt(i)))
    i++;
  else {
    System.out.println("
           Ziffer erwartet an Stelle " + (i+1));
    System.exit(1);
    } // Ende else
  } // Ende prüfeFaktor
```

Kommt man aus der Rekursion sauber heraus und stimmen die gezählten Zeichen mit der Länge des Wortes überein, so ist das Wort syntaktisch korrekt. In JAVA lautet das Hauptprogramm wie folgt:

```
public class ArithParser {
  // Syntaxprüfung arithmetischer Ausdrücke

  static int    i, länge;
  static String eingabewort;

  static void prüfeAusdruck() {
    // Ausdruck --> Summand | Summand + Ausdruck
    prüfeSummand();
    if (eingabewort.charAt(i) == '+')
      {i++; prüfeAusdruck();}
    } // Ende prüfeAusdruck

  static void prüfeSummand() {
    // Summand --> Faktor | Faktor * Summand
    prüfeFaktor();
    if (eingabewort.charAt(i) == '*')
      {i++; prüfeSummand();}
    } // Ende prüfeSummand

  static void prüfeFaktor() {
    // Faktor --> Ziffer | (Ausdruck)
    if (eingabewort.charAt(i) == '(') {
      i++; prüfeAusdruck();
      if (eingabewort.charAt(i) == ')')
        i++;
      else {
        System.out.println("
              Schliessende Klammer erwartet.");
        System.exit(1);
        } // Ende else
      } // Ende if
    else if (Character.isDigit
            (eingabewort.charAt(i)))
      i++;
    else {
      System.out.println("
             Ziffer erwartet an Stelle " + (i+1));
      System.exit(1);
      } // Ende else
    } // Ende prüfeFaktor

  public static void main (String[] eingabe) {
    System.out.println("\n ARITHMETIK-PARSER\n");
    eingabewort = "(2+7)*(4+3) ";
    if (eingabe.length > 0)
      eingabewort = eingabe[0] + " ";
```

```
    länge = eingabewort.length() - 1;
    i = 0;
    prüfeAusdruck();
    if (i == länge)
      System.out.println(" " + eingabewort
                           + "ist korrekt.");
    else
      System.out.println(" Fehler an Stelle
                         " + (i+1));
    } // Ende main

} // Ende ArithParser
```

Aufgabe 1.4: Man erweitere Grammatik und Programm um Division und Subtraktion.

Auswertung arithmetischer Ausdrücke

Nachdem wir nun in der Lage sind, zu prüfen, ob ein arithmetischer Ausdruck korrekt ist, wollen wir ihn auswerten, d. h. seinen Wert bestimmen.

Postfixnotation

Dazu ist es zunächst notwendig, die Klammern zu entfernen, d. h. der Ausdruck wird in die sogenannte ***U**mgekehrte **P**olnische **N**otation* (kurz UPN, oder: Postfixnotation) überführt. Diese Schreibweise, die auf den polnischen Mathematiker Jan Łukasiewicz (Bild 2) zurückgeht, notiert den Operator nicht zwischen den Operanden, auf die er angewendet werden soll (Infix-Notation), sondern direkt dahinter. Damit sind Klammern entbehrlich. Ein weiterer Vorteil dieser Schreibweise ist die Möglichkeit der stapelbasierten Berechnung (siehe unten). Die Schreibweise im Vergleich:

Term	UPN
2+3	23+
2+4·4	244·+
(1+2)·(3+4)	12+24+·

UPN wurde bei frühen Taschenrechnern angewendet. Hewlett-Packard beispielsweise bietet im Netz Simulationen einiger Modelle an (Bild 3).

Wir nehmen zunächst einmal an, ein gegebener arithmetischer Ausdruck sei in Postfixnotation dargestellt. Um ihn auszuwerten, wird ein Kellerspeicher benötigt.

Auswertung mittels Kellerspeicher

Ein *Kellerspeicher* oder *Stapel* (engl.: *stack*) ist eine spezielle Liste mit folgenden Eigenschaften: Es kann immer nur auf das oberste Objekt zugegriffen werden,

Bild 2:
Der polnische Logiker Jan Łukasiewicz (1878–1956) erfand die klammerfreie „Polnische Notation" (Präfixnotation).

Quelle: LOG-IN-Archiv

und das Objekt, das zuletzt in den Stapel gebracht wurde, muss auch als erstes wieder entnommen werden (LIFO-Prinzip – ***l**ast **i**n, **f**irst **o**ut*). In JAVA werden Kellerspeicher durch die Klasse Stack realisiert, die eine Unterklasse der Klasse Vector ist. Im Unterricht empfiehlt es sich, eine Klasse Zahlstapel selbst zu implementieren (etwa mithilfe von Reihungen).

```
class Zahlstapel {
  // Implementation mittels Reihung

  int   maxLänge;
  int[] reihung;
  int   spitze;

  Zahlstapel (int m) {
    maxLänge = m;
    reihung = new int[maxLänge];
    spitze = -1;
    } // Ende Konstruktor

  boolean istLeer()
    {return spitze == -1;}

  boolean istVoll()
    {return spitze == maxLänge - 1;}

  void legeAuf (int zahl) {
    if (! istVoll())
      reihung[++spitze] = zahl;
    } // Ende legeAuf

  int holeVon() {
    if (istLeer()) return 0;
    return reihung[spitze--];
    } // Ende holeVon
```

Bild 3:
Der legendäre HP-35 aus dem Jahr 1972 – der erste wissenschaftlich-technische Taschenrechner der Welt – arbeitete mit UPN und kostete respektable 1890 DM.

Quelle: LOG-IN-Archiv

```
  int liesSpitze()
    {return reihung[spitze];}

  } // Ende Zahlstapel
```

Als Anwendung soll ein (syntaktisch korrekter) arithmetischer Ausdruck in Postfixnotation berechnet werden. Er wird wie folgt sequenziell durchlaufen:

- ▷ Ist das aktuelle Zeichen ein Operand (hier: Ziffer), so wird es auf den Stapel gelegt.
- ▷ Ein Operator (hier: Plus- oder Malzeichen) wird auf die beiden obersten Elemente des Stapels angewendet und das Ergebnis wieder auf den Stapel gelegt.
- ▷ Nach Durchlaufen des Ausdrucks liegt das Ergebnis (als einziges Element) auf dem Stapel.

Der Ausdruck 2·(3+4) entspricht dem Postfixausdruck 234+· und wird wie in der Tabelle 1 ausgewertet.

Das Programm dazu lautet:

```
public class Auswertung {
  // Postfix-Ausdruck wird ausgewertet

  public static void main (String[] eingabe) {
    String eingabewort = "23+19+*";
    if (eingabe.length > 0)
      eingabewort = eingabe[0];
    Zahlstapel stapel = new Zahlstapel(20);
    for (int i = 0; i < eingabewort.length(); i++) {
      char zeichen = eingabewort.charAt(i);
      if (Character.isDigit(zeichen))
        stapel.legeAuf(zeichen - '0');
      else { // Zeichen ist keine Ziffer
        int operand1 = stapel.holeVon();
        int operand2 = stapel.holeVon();
        if (zeichen == '+')
          stapel.legeAuf(operand1 + operand2);
        else if (zeichen == '*')
          stapel.legeAuf(operand1 * operand2);
        } // Ende else
      } // Ende for
    int wert = stapel.holeVon();
    if (stapel.istLeer())
      System.out.println("\n " + eingabewort +
                         " ergibt " + wert);
    else
      System.out.println("\n Syntaxfehler!");
    } // Ende main

  } // Ende Auswertung
```

Aufgabe 2.1: Ergänzen Sie das Programm um Subtraktion und Division.

Aufgabe 2.2: Schreiben Sie ein Programm, das Palindrome (z. B. RELIEFPFEILER) erkennt, indem es das Spiegelwort erzeugt. (a) rekursiv, (b) mithilfe eines Kellerspeichers (Stapels).

Umwandlung von Infix- in Postfixnotation

Um obiges Auswertungsverfahren anwenden zu können, muss der gegebene Ausdruck in Postfixnotation umgewandelt werden. Dafür wird wieder ein Kellerspeicher (Stapel) benötigt, der aber diesmal zur Aufnahme von Zeichen (char) geeignet sein muss. Der (syntaktisch korrekte) Ausdruck in Infixnotation wird von links nach rechts zeichenweise gelesen; dabei entsteht zugleich der Postfixausdruck, indem jeweils rechts Zeichen angehängt werden. Dies geschieht nach folgenden Regeln:

Eingabe	Operation	Stapelinhalt
2	Auf Stapel legen	2
3	Auf Stapel legen	3 2
4	Auf Stapel legen	4 3 2
+	Vom Stapel holen, ausrechnen, auf Stapel legen	7 2
·	Vom Stapel holen, ausrechnen, auf Stapel legen	1 4

Tabelle 1.

(1) Operanden (Buchstaben oder Zahlen) werden sogleich angehängt.
(2) Eine öffnende Klammer wandert auf den Stapel.
(3) Kommt eine schließende Klammer, werden solange Operatoren (Rechenzeichen) vom Stapel geholt und an den Postfixausdruck angehängt, bis eine öffnende Klammer erscheint; die Klammern werden „entsorgt".
(4) Kommt ein Operator, und der Stapel ist leer oder es liegt eine öffnende Klammer oben auf dem Stapel, so wandert er selbst auf den Stapel. Ist seine Priorität nicht höher als die des Operators oben auf dem Stapel, wird dieser vom Stapel genommen und an den Postfixausdruck angehängt; er selbst wandert auf den Stapel. Hat er dagegen höhere Priorität, wandert er selbst auf den Stapel.
(5) Ist der Infixterm beendet, wird der restliche Stapelinhalt an den Postfixterm angehängt.

Veranschaulichen wir das Verfahren an dem Ausdruck (2 + 3) · (4 + 5 · 6) z. B. in der Tabelle 2, nächste Seite.

Aufgabe 2.3: Füllen Sie die Tabelle aus für (a) 2 + 3 · 4, (b) (2 + 3) · 4, (c) 5 · 6 + 7 · (8 + 9).

Bei Balzert (1999) wird der Algorithmus anschaulich am Beispiel des von einem Bahnstellwerk geleiteten Rangierens auf einem Güterbahnhof beschrieben. Das zur Aufgabe 2.3 gehörende Programm lautet:

```
public class InPostfix {
  // Umwandlung Infix in Postfix-Ausdruck

  static int priorität (char operator) {
    if (operator == '+') return 1;
    if (operator == '*') return 2;
    return 0;
    } // Ende priorität

  public static void main (String[] eingabe) {
```

Eingabe-zeichen	Operation	Stapel-inhalt	Postfix-ausdruck
(	Auf Stapel legen	(	
2	An Postfixausdruck anhängen	(	2
+	Auf Stapel legen	(+	2
3	An Postfixausdruck anhängen	(+	2 3
)	Operator anhängen, Klammer „entsorgen“		2 3 +
•	Auf Stapel legen	•	2 3 +
(	Auf Stapel legen	• (	
4	An Postfixausdruck anhängen	• (	2 3 + 4
+	Auf Stapel legen	• (+	2 3 + 4
5	An Postfixausdruck anhängen	• (+	2 3 + 4 5
•	Auf Stapel legen	• (+ •	2 3 + 4 5
6	An Postfixausdruck anhängen	• (+ •	2 3 + 4 5 6
)	Operatoren vom Stapel holen und anhängen		2 3 + 4 5 6 • + •

Tabelle 2.

```
    String eingabewort = "(2+3)*4";
    if (eingabe.length > 0)
      eingabewort = eingabe[0];
    Zeichenstapel stapel = new Zeichenstapel(20);
    String postfix = "";
    for (int i = 0; i < eingabewort.length(); i++) {
      char zeichen = eingabewort.charAt(i);
      if (Character.isDigit(zeichen))
        postfix += zeichen;
      else if (zeichen == '(')
        stapel.legeAuf(zeichen);
      else if (zeichen == ')') {
        while (stapel.liesSpitze() != '(')
          postfix += stapel.holeVon();
        stapel.holeVon();
                    // öffnende Klammer "entsorgen"
        } // Ende else
      else {
        while (! stapel.istLeer() &&
               stapel.liesSpitze() != '(' &&
               priorität(zeichen) <=
               priorität(stapel.liesSpitze()))
          postfix += stapel.holeVon();
        stapel.legeAuf(zeichen);
        } // Ende else
      } // Ende for
    while (! stapel.istLeer())
      postfix += stapel.holeVon();
    System.out.println("\n " + postfix);
    } // Ende main

} // Ende InPostfix
```

Aufgabe 2.4: Die Programme InPostfix.java und Auswertung.java sollen zu einem einzigen Programm vereinigt werden, das einen Ausdruck in Infix-Notation auswertet.

Je nach Leistungsstand der Schüler kann man nach Besprechung des Verfahrens die Schüler das obige Parserprogramm erweitern lassen oder aber das erweiterte Programm vorgeben. Beispielsweise muss die Methode prüfeSummand() folgendermaßen erweitert werden:

```
void prüfeSummand() {
  // Summand -> Faktor | Faktor * Summand
  prüfeFaktor();
  if (eingabewort.charAt(i) == '*') {
    i++;
    stack.push("*"); // Malzeichen in den Keller
    prüfeSummand();
    } // Ende if
  } // Ende prüfeSummand
```

Das Ergebnis ist ein Compiler für arithmetische Ausdrücke, mit den beschriebenen Einschränkungen.

Zusammenfassung

Ein arithmetischer Ausdruck lässt sich berechnen, indem – wie beschrieben – folgende Teilschritte ausgeführt werden:

- ▷ Prüfung auf syntaktische Korrektheit (mittels rekursiven Abstiegs),
- ▷ Überführung des arithmetischen Ausdrucks in umgekehrte polnische Notation,
- ▷ Berechnung des vorliegenden Postfixausdrucks unter Verwendung eines Kellerspeichers.

Das Programm lässt sich um beliebige Operatoren (wie beispielsweise Potenzierung oder Sinusfunktion) erweitern, nachdem die Grammatik geeignet ergänzt worden ist.

Torsten Brandes
Rosa-Luxemburg-Oberschule
Kissingenstraße 12
13189 Berlin

E-Mail: torsten.brandes@web.de

Die JAVA-Programme stehen den Lesern über den **LOG-IN-Service** (s. S. ■) zur Verfügung.

Literatur

AKBSI – Arbeitskreis „Bildungsstandards“ der Gesellschaft für Informatik (Hrsg.): Grundsätze und Standards für die Informatik in der Schule – Bildungsstandards Informatik für die Sekundarstufe I. Empfehlungen der Gesellschaft für Informatik e. V. vom 24. Januar 2008. In: LOG IN, 28. Jg. (2008), Heft 150/151, Beilage.

Gumm, H.-P.; Sommer, M.: Einführung in die Informatik. München-Wien: Oldenbourg, [5]2002.

Balzert, H.: Lehrbuch Grundlagen der Informatik. Heidelberg: Spektrum, [2]1999.