Übungen zur Vorlesung
„Grundlagen der Informatik 1“
Studiengang E-Technik

WS 2001/02

AG Betriebssysteme, FB 3
Jan Bredereke

ausgegeben: 11.1.2002
Abgabe und Besprechung:
in der Übung am **17.1.2002**

Serie 5

Grammatiken

Aufgabe 1: Syntaxüberprüfung für Taschenrechner mit polnischer Notation

Die polnische Notation (PN) ist eine Schreibweise für mathematische Terme, die z.B. von manchen Taschenrechnern als Eingabeform verwendet wird. Der Verknüpfungsoperator steht dabei immer vorne, und es folgen die zu verknüpfenden Zahlen oder Ausdrücke danach. Um die Terme leichter lesbar zu machen, verwenden wir Klammern. Den mathematischen Term „$4 + 8 - 2$“ schreibt man z.B. als „`+(4,-(8,2))`“.

Schreibt ein C-Programm, welches vollständig geklammerte mathematische Terme, gegeben in PN, auf ihre syntaktische Korrektheit bezüglich der unten angegebenen Grammatik überprüft.

Euer Programm soll den zu überprüfenden Ausdruck als eine Kommandozeilenparameter-Zeichenkette übergeben bekommen, d. h. es soll später z.B. in der Form
`SynCheck "+(4,-(8,2))"`
aufgerufen werden. Der zu prüfende String befindet sich dann in dem ersten Element des `argv`-Parameters der `main()`-Funktion Eures Programms. Als Ergebnis soll auf dem Bildschirm nur ausgegeben werden, ob es sich bei einem Ausdruck um einen korrekten `TERM` handelt oder nicht; es soll dabei *nicht* auch noch das Ergebnis eines übergebenen Ausdrucks berechnet werden.

a) Lexikalische Grammatik

Die Ausdrücke sollen aus positiven Zahlen inklusive der 0, den mathematischen Grundoperationen `+,-,*,/`, runden Klammern `()` sowie Kommas zusammengesetzt sein. Daraus ergibt sich für die Token-Definitionen die folgende lexikalische Grammatik:

```
binop = + | / | *
minop = -
lb    = (
rb    = )
komma = ,
num   = 0 | 1 | 2 | 3 | ... | 10 | 11 | ...
```

`NUM` soll dabei den natürlichen Zahlen inklusive 0 entsprechen. Der Einfachheit halber sind auch Zahlen mit führenden Nullen zugelassen.

Schreibt zunächst ein Programm `lexCheck`, das prüft, ob der Eingabestring ausschließlich aus zulässigen Token besteht.

Definiert hierzu einen Aufzählungstyp („`enum`“), der Namen für alle zulässigen Token enthält, sowie die Namen `unbekannt` und `string_ende`. Schreibt dann eine Funktion `getToken()`, die zur aktuellen Stelle im Eingabestring den Namen des zugehörigen Tokens liefert, und die die aktuelle Stelle entsprechend nach hinten verschiebt. Schreibt schließlich eine Hauptfunktion `main()`, die `getToken()` so lange aufruft, bis entweder das Stringende oder ein Fehler gefunden wurde.

Testet das Programm mit einigen Eingaben.

b) Syntaktische Grammatik

Aufbauend auf der lexikalischen Grammatik besteht die PN-Grammatik aus fünf Regeln:

```
TERM    = BINTERM
        | MINTERM
        | num

BINTERM = binop ARGS

MINTERM = minop MINREST

MINREST = ARGS
        | num

ARGS    = lb TERM komma TERM rb
```

Schreibt ein Programm `synCheck`, daß seine Eingabe auf korrekte PN-Syntax überprüft. Verwendet dazu die Funktion `getToken()` aus der vorigen Teilaufgabe. Zusätzlich benötigt Ihr nun eine Funktion `putToken`, die im Eingabestring wieder zur Position vor dem letzten Aufruf von `getToken()` zurückgeht. (`putToken()` wird nie mehr als einmal aufgerufen, ohne daß ein neuer Aufruf von `getToken()` folgt.)

Schreibt für jede Regel der obigen PN-Grammatik eine eigene Funktion zur Überprüfung. Dadurch spiegelt sich die rekursive Natur der Grammatikregeln direkt in einer entsprechenden Aufrufstruktur Eurer Funktionen wieder.

Zeigt anhand geeigneter Testfälle, daß Euer Programm korrekte und fehlerhafte PN–Terme unterscheiden kann.