Recursive-Descent-Parser

Sie sollen einen Parser für die Analyse und Auswertung arithmetischer Ausdrücke mit Hilfe der Methode des rekursiven Abstiegs implementieren. Der Parser soll arithmetische Ausdrücke analysieren, die von der folgenden Grammatik generiert werden:

start	→ addexpr
addexpr	→ mulexpr (('+' | '–') mulexpr)*
mulexpr	→ bracketexpr (('*' | '/') bracketexpr)*
bracketexpr	→ '(' addexpr ')' | '−' bracketexpr | number
number	→ integer ('.' integer)?
integer	→ ('0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9')+

Die Anführungszeichen markieren hier die Terminalsymbole der Grammatik. Der Operator | steht für die Disjunktion. Die regulären Operatoren ?, *, + stehen für 0 oder 1 (?) bzw. beliebig viele (*) bzw. mindestens 1 (+) Wiederholung des vorhergehenden Ausdruckes. Leerzeichen in den arithmetischen Ausdrücken sollen ignoriert werden.

Der Parser soll aus der Eingabedatei arthmetische Ausdrücke (einer pro Zeile) lesen, parsen und gleichzeitig den Wert berechnen, und dann das Ergebnis im Format *Ausdruck = Wert* ausgeben. Hier ein Beispielaufruf:

```
> cat test.txt
2.7 * -3.5 + 1.9
3*(9-7)/4.0
> python eval.py test.txt
```

Ausgabe (Die Zahl der Nachkommastellen ist egal):

```
2.7*-3.5+1.9 = -7.55
3*(9-7)/4.0 = 1.5
```

Schreiben Sie für jedes Nichtterminal der obigen Grammatik eine Python-Funktion, welche einen Ausdruck dieses Typs erkennt, den Ergebniswert berechnet und zurückgibt. Die Funktion für das Nichtterminal auf der linken Seite einer Regel ruft die Funktionen für die Nichtterminale auf der rechten Seite der Regel auf. Außer dem Ergebniswert muss die Funktion auch noch die erreichte Stringposition zurückgeben.

Wenn ein Ausdruck ungrammatisch ist, geben Sie eine Fehlermeldung in der folgenden Form aus:

```
Fehler: schließende Klammer erwartet: 7*(3.75-1
                                              ^
```

Das Symbol ^ in der zweiten Zeile zeigt die Fehlerposition an.

Dann machen Sie mit der nächsten Eingabezeile weiter. Für die Fehlerbehandlung verwenden Sie den Exception-Mechanismus. Definieren Sie dazu eine eigene Exception-Klasse mit `def ExprError(Exception):` und fangen Sie die Exceptions im Hauptprogramm ab.

Achtung: Verarbeiten Sie die Eingabe beim Parsen Zeichen für Zeichen von links nach rechts. Versuchen Sie nicht, die Eingabe zunächst in Teile zu zerlegen und dann diese Teile zu parsen. Die Funktionen *eval*, *exec*, *find*, *index*, *split*, sowie reguläre Ausdrücke und Funktionen zur Umwandlung von Strings in Zahlen wie *int* oder *float* sind in dieser Übung nicht erlaubt. Verwenden Sie ein Dictionary, um die Ziffern 0-9 auf Integerzahlen abzubilden.

Vorüberlegungen

- Wie generiert die obige Grammatik den Ausdruck 2.75 * (-3 + 4)?
- Spielen Sie am gleichen Beispiel durch, welche Funktionsaufrufe bei der Verarbeitung des Ausdruckes erfolgen und welcher Teil der Eingabe jeweils verarbeitet wird.
- Schreiben Sie Pseudocode für die Funktion **number**.
- Wie können Sie in der Funktion **bracketexpr** entscheiden, welche der drei Alternativen vorliegt?
- Woran merken Sie bei der Verarbeitung eines Ausdrucks, dass er nicht wohlgeformt ist?
- Wie gehen Sie am besten mit Leerzeichen um?
- Wann kann der Zugriff auf eine bestimmte Position des Ausdruckes einen Indexfehler auslösen und wie können Sie den Fehler vermeiden?

Implementieren Sie das Programm und testen Sie es systematisch mit verschiedenen Eingaben.