Automaten und formale Sprachen

Die Programmiersprache “list”

Aufgabe 1 list 1.0

Die Programmiersprache “list” erlaubt das Programmieren von nummerierten Listen.

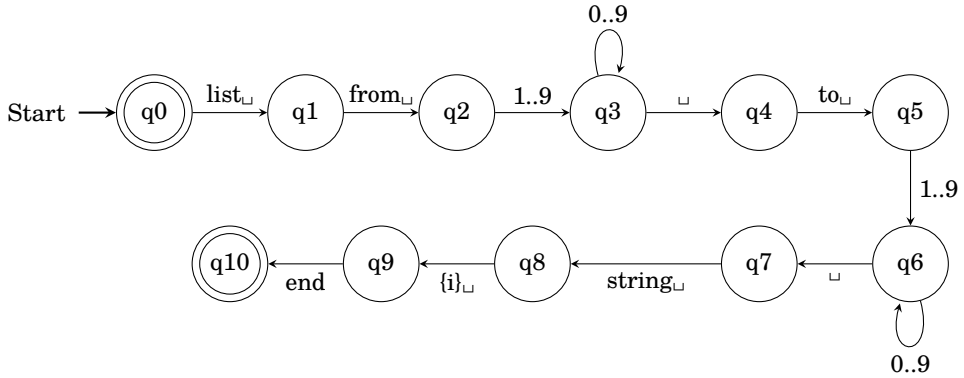
Beispielhaftes Programm in “list”:

```
list from 1 to 5
    Listeneintrag {i}
end
```

Ausgabe des Programms:

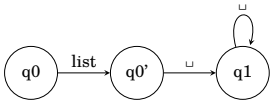
```
Listeneintrag 1
Listeneintrag 2
Listeneintrag 3
Listeneintrag 4
Listeneintrag 5
```

Ein endlicher Automat für die Analyse der Sprache könnte (vereinfacht) so aussehen.



Wobei zur besseren Übersicht folgende Vereinfachungen vereinbart werden:

- **0..9** steht für die zehn Übergänge mit jeweils einer der Ziffern von 0 bis 9 als Übergangssymbol.
- **string** steht für eine beliebige Zeichenfolge.
- Das Symbol ␣ steht für “Whitespace”, also eine beliebige Anzahl an Leerzeichen, Tabulatoren und Zeilenumbrüchen.
- Steht ␣ am Ende eines Übergangs, bedeutet dies, dass der Buchstabe (z.B. `list`) von mindestens einem “Whitespace” gefolgt werden muss. Der Übergang von `q0` nach `q1` ist also eine Kurzform von



Implementiere einen Scanner, Parser und Interpreter für “list”.

Hinweis *Die Projektvorlage enthält schon die Vorbereitung für einen Scanner, der den eingelesenen Quelltext nicht Zeichenweise durchgeht, sondern ihn jeweils an den Leerzeichen trennt und “Wort für Wort” durchläuft. Der Scanner muss also vor allem noch die Tokens aus dem Quelltext erstellen.*

Aufgabe 2 list 2.0

Für Version 2 von “list” soll die Sprache um eine `by`-Nweisung ergänzt werden. Außerdem soll es möglich sein, als Textausgabe beliebige Kombinationen aus Strings und Zählern (`{i}`) anzugeben.

Beispielhaftes Programm in “list2”:

```
list from 1 to 99 by 3
    {i} bottles of beer on the wall, {i} bottles of beer.
end
```

Ausgabe des Programms:

```
1 bottles of beer on the wall, 1 bottles of beer.
4 bottles of beer on the wall, 4 bottles of beer.
7 bottles of beer on the wall, 7 bottles of beer.
...
```

Aufgabe 3 list 3.0

"list3" soll beliebig viele `list`-Blöcke hintereinander erlauben.

Aufgabe 4 list 4.0

List ist mittlerweile schon recht umfangreich. Version 4.0 von "list" soll die Sprache daher etwas vereinfachen und `from`, `by` und die Zählvariable optional machen. Die ersten beiden bekommen als Standardwert die `1`, die Zählvariable `i`.

Ein minimales "list" Programm sieht dann so aus:

```
list to 5
    Listeneintrag {i}
end
```

Aufgabe 5 list 5.0

Version 5 wird das bisher größte Update. "list" soll nun *verschachtelte* `list`-Blöcke erlauben. Dazu muss nun die Zählervariable (bisher `i`) immer mit angegeben werden.

Beispielhaftes Programm in "list4":

```
list i from 1 to 5 by 2
    Liste {i}
    list j from 5 to 9 by 2
        - {j} Listeneintrag
    end
end
```

Ausgabe des Programms:

```
Liste 1:
- 1.5 Listeneintrag
- 1.7 Listeneintrag
- 1.9 Listeneintrag
Liste 3:
- 3.5 Listeneintrag
- 3.7 Listeneintrag
...
```

Hinweis *Version 5 ist keine reguläre Sprache mehr. Zur Analyse muss nun also ein Kellerautomat verwendet werden. Konstruiere als Hilfe zunächst einen Automaten aus dem Übergangsgraphen in Aufgabe 1. Die Kellersymbole entsprechen den Zählvariablen. Pro `list` Befehl wird die Zählvariable auf den Keller gelegt und bei `end` wieder entfernt.*

Aufgabe 6 list 6.0

Das finale Update von "list" soll nochmal einige Neuerungen bringen. Die Community diskutiert noch, welche Features als nächstes umgesetzt werden sollten. Hier sind die Vorschläge, die im Moment die meisten Stimmen haben:

- Der Aufruf der Zählvariablen in der Ausgabe könnte einfache Arithmetik unterstützen. Zum Beispiel `{i+1}` oder `{i*2}`.
- Alle Zahlen (Anfangswert, Endwert, Schrittweite) dürfen auch negativ sein. (Vorsicht: Es könnten sich semantische Fehler einschleichen.)
- Die Ausgabe erlaubt derzeit nur eingeschränkte Verwendung von Leerzeichen in der Ausgabe. Die List `1-1, 1-2, 1-3, 2-1, 2-2, 2-3` kann momentan nicht erzeugt werden. Oder Listeneinträge mit zwei Leerzeichen am Anfang. Das könnte geändert werden.
- `from` und `to` dürfen auch Buchstaben sein. `list from a to d {i} end` würde dann `a b c d` ausgeben.