1 Makros (8 Punkte)

Die vordefinierten *special forms* push und pop dienen zur Handhabung sogenannter Kellerspeicher (*stacks*), z.B.

```
? (defparameter *stack* '(2 3 4)) → *stack*
? *stack* → (2 3 4)
? (push 1 *stack*) → (1 2 3 4)
? *stack* → (1 2 3 4)
? (pop *stack*) → 1
? *stack* → (2 3 4)
? (defparameter foo "bar") → foo
? (push foo *stack*) → ("bar" 2 3 4)
? *stack* → ("bar" 2 3 4)
```

Probiere push() und pop() als normale Funktionen zu definieren; erkläre, warum das nicht funktionieren *kann*.

Hinweis: push() und pop() haben einen sogenannten Seiteneffekt: sie verändern den Wert des Kellerspeichers dauerhaft (siehe Beispiele).

2 ADT Grammatik und Regel (12 Punkte)

Unser zweiter Schritt, um die notwendigen Grundfunktionalitäten für den Parser zu erhalten, besteht darin, einen ADT für eine Gramatikregel und einen ADT für eine Grammatik zu schreiben. Auch diesmal gibt es Freiraum für eigene Ideen. Wichtig ist nur, die Funktionaliäten (Schnittstellen) einzuhalten. Es ist z.B. notwendig für den Grammatik-ADT, dass eine Regel mit den Zugriffsfunktionen lhs (left-hand-side) bzw. rhs (right-hand-side) zerlegt werden kann.

- Schreibe einen ADT für eine kontextfreie Grammatik-Regel. Die Mindestfunktionalität soll sein:

  make-rule lhs &rest rhs kreiert eine Regel.

  lhs <rule> Selektor.

  rhs <rule> Selektor.

  copy-rule <rule>

  Freiwillige und nützliche Funktionalitäten sind: (print-rule <rule>), (is-rule <x>)

- Schreibe einen ADT für eine Grammatik. Er soll mindestens folgende Funktionalitäten besitzen:

  (make-grammar &key terminals non-terminals start-symbol) liefert eine leere Grammatik zurück (allerdings mit darin enthaltenen terminalen und nicht-terminalen Symbolen und mit dem Start-Symbol).

  (add-rule <grammar> <rule>) addiert eine Regel zu der Grammatik.

  (find-all-rules-with-lhs <grammar> <symbol>) liefert eine Liste mit allen Regeln, deren lhs eq mit dem angegebenen Symbol ist.

  (find-all-rules-with-first-rhs <grammar> <symbol>) liefert eine Liste mit all den Regeln, deren am weitesten links stehendes Symbol (leftmost :-) der right-hand-side eq mit dem angegebenen Symbol ist[1].

  Freiwillig sind folgende Funktionalitäten (die allerdings sehr nützlich beim Entwickeln einer Grammatik sein können): ({set|add|remove}-terminal(s) <grammar> {<list-of-symbols>|<symbol>}), ({set|add|remove}-non-terminal(s) <grammar> <list-of-symbols>), (print-grammar <grammar> &optional stream), (clear-grammar <grammar>) und (is-grammar <x>).

  Ein möglicher Dialog:

[1] Deutsche Sprache – schwere Sprache

```
> (setq *grammar* (make-grammar :start-symbol 's :non-terminals '(s np vp) :terminals '(noun verb))
#<grammar>   ;;; oder was immer eure Implementierung macht
> (add-rule *grammar* (make-rule 's 'np 'vp))
#<grammar>
> (find-all-rules-with-lhs *grammar* 's)
( #<rule1> )
> (find-all-rules-with-first-rhs *grammar* 'vp)
( #<rule1> )
> (second (rhs (first *)))
VP
> (first (rhs (first (find-all-rules-with-lhs *grammar* 's))))
NP
```

3 Is-Schmök (3 Punkte)

Welche Symbole sind als Makros und welche als Funktionen definiert: `dolist, cons, first, append, loop,...`? Wie kann man das erkennen? Syntaktisch? Funktionsweise?

Schreibe eine Funktion, die eine Liste von Symbolen nimmt und für jedes Symbol in einer schönen Art und Weise auf den Bildschirm schreibt, ob es als Funktion, als Makro oder gar nicht definiert ist. Die Funktion `(arglist <symbol>)` liefert die Argumentliste einer Funktion/Makro (vorausgesetzt, daß das Symbol als Funktion oder Makro definiert ist).

Abgabetermin ist der 4. Februar 2000