lang2.ss

Creating your own Language
with Scanner/Parser

A language of your own

```
{ x := 1 ; var1 := 100 }

while (add1 x) do v := (add1 v)

sum := (sum 1 2 3)

while let x = 1; var = 2 in (sum x var) do x := sum
```

A language to call your own

Statement: **Compound** | **While** | **Assign**

  Compound: { **Expr** ; **Expr** }

  While: while **Expr** do **Statement**

  Assign: id := **Expr**

Expr: number | id | (**Expr Expr***)

Expr: let **Decl** [; **Decl**]* in **Expr**

Decl: id **= Expr**

While Statement Parse

Concrete Syntax:

```
while let x = 1; var = 2 in (sum x var) do x := sum
```

Abstract Syntax:

```
(while-statement
    (let-exp ((decl x (lit-exp 1)) (decl var (lit-exp 2)))
         (app-exp (var-exp sum)
                ((var-exp x) (var-exp var))))
       (assign-statement x (var-exp sum)))
```

Scanner

```
;; lang2.ss
;; Doug Blank

(load "petite-init.ss")
(load "sllgen.ss")

(define scanner
  '((whitespace (whitespace) skip)
    (comment ("%" (arbno (not #\newline))) skip)
    (identifier (letter (arbno (or letter digit)))
        make-symbol)
    (number (digit (arbno digit)) make-number)))
```

Grammar

```
(define grammar
  '((statement
      ("{" statement ";" statement "}")
      compound-statement)
    (statement
      ("while" expression "do" statement)
      while-statement)
    (statement
      (identifier ":=" expression)
      assign-statement)
    ...
```

Grammar (cont)

```
(expression
  (number)
  lit-exp)
(expression
  (identifier)
  var-exp)
(expression
  ("let" (separated-list declaration ";")
      "in" expression)
  let-exp)
(expression
  ("(" expression (arbno expression) ")")
  app-exp)
(declaration
  (identifier "=" expression)
  decl)))
```

Read-Eval-Print-Loop

```
(define parse
  (sllgen:make-string-parser scanner
       grammar))

;; (parse "{x := foo; y := bar}")

(define repl
  (sllgen:make-rep-loop
    "--> "
    (lambda (x) x)
    (sllgen:make-stream-parser scanner
         grammar)))

(repl)
```