Reading
Read Sections 4.4 through 4.6 of our textbook *Compilers.*

Written Assignment 4

1. Exercise 4.4.1, parts a, b, and c (page 231).
2. Exercise 4.4.3.
3. Exercise 4.5.1 (page 240). For this exercise, show the rightmost derivation for both sentential forms (a) and (b) and underline the handles in each form.
4. Exercise 4.5.2. For this exercise, show the rightmost derivation for both sentential forms (a), (b), and (c) and underline the handles in each form.
5. Exercise 4.5.3.
6. Consider the following grammar:

E	→	*E* + *T*
	|	*T*
T	→	*T F*
	|	*F*
F	→	*F* *****
	|	**a**
	|	**b**

Construct the SLR parsing table for this grammar. This will require you to compute the Follow sets for the nonterminals *E, T,* and *F,* as well as the item sets.

Programming Assignment 4

To our previous FSS language we add *while*, *begin,* and *print* expressions, resulting in somewhat simple Scheme (SSS) programs. *While* expressions support iteration, and *begin* expressions support compound expressions comprising a sequence of one or more expressions.

prog	→	*expr*+
expr	→	DOUBLE
	|	BOOLEAN
	|	ID
	|	'(' RATOR *expr*∗ ')'
	|	'(' 'def' ID *expr* ')'
	|	'(' 'if' $expr_1$ $expr_2$ $expr_3$ ')'
	|	'(' 'print' *expr* ')'
	|	'(' 'while' $expr_1$ $expr_2$ ')'
	|	'(' 'begin' *expr*+ ')'
BOOLEAN	→	'true' | 'false'
RATOR	→	ARITHMETIC | RELATIONAL | LOGICAL
ARITHMETIC	→	'+' | '–' | '*' | '/'

RELATIONAL → '=' | '>' | '<'
LOGICAL → '&' | '|' | '!'

A *while* expression evaluates its test expression $expr_1$. If $expr_1$ evaluates to true, the body $expr_2$ is evaluated and then the process is repeated; if $expr_1$ evaluates to false, the most recent value of $expr_2$ gets returned (or 0 if $expr_2$ was never evaluated). Thus a *while* expression behaves like the *while* expression found in C or Java.

A *begin* expression evaluates its expressions left to right and returns the value of its rightmost expression. The expressions preceding the rightmost expression are generally performed for their side-effect which, in our current language, is either assignment or print.

A *print* expression evaluates its operand expression and prints its value and also returns this value.

Some examples:

```
> java run
(begin 2 3 (* 2 2))
^Z
4.0

> java run
(begin (def a 3) (def a (* a a)) a)
^Z
9.0

> java run
(def a (print (+ 2 3)))
(+ a 12)
^Z
5.0
17.0

> java run
(def n 4)
(while (> n 0)
  (def n (- (print n) 1)))
^Z
4.0
3.0
2.0
1.0
0.0
```

```
> java run      // factorial of 6
(def a 1)
(def n 6)
(while (> n 0)
  (begin (def a (* a n))
         (def n (- n 1))))
a
^Z
720.0

> java run     // print first 10 values of Fibonacci sequence
(def n 10)
(def a 1)
(def b 1)
(while (> n 0)
  (begin
    (print a)
    (def next (+ a b))
    (def a b)
    (def b next)
    (def n (- n 1))))
^Z
1.0
1.0
2.0
3.0
5.0
8.0
13.0
21.0
34.0
55.0
0.0
```