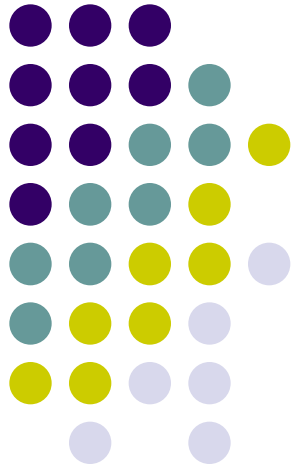
CS 460

Programming Languages

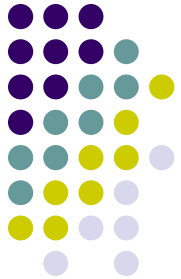
Fall 2023

Dr. Watts

(1 November 2023)



Assignments

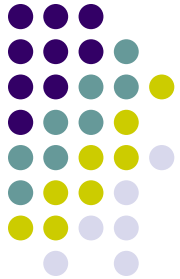


- Exercise 2
  - Script running so that your groups can improve their testing techniques
- Exercise 3
  - Posted – let me know if you see typos
  - Part 2 – comment out last function call

```
;; (main)
```

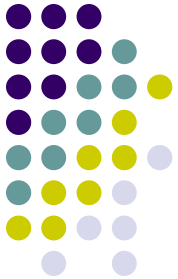
- Project 2
  - Coming soon (Wednesday)
- Exercise 5
  - Preliminary exercise will be posted this week

First and Follow Sets



- Firsts
  - A terminal symbol T_i is a member of the First Set of non-terminal symbol <nt_j> if T_i can become the first terminal symbol in a complete expansion of <nt_j>.
- Follows
  - A terminal symbol T_i is a member of the Follow Set of non-terminal symbol <nt_j> if T_i can become the first terminal symbol immediately following a complete expansion of <nt_j>.

Why do we need the First and Follow Sets?



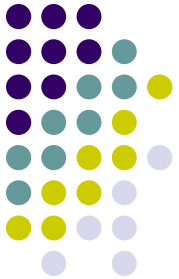
- Making decisions!

15. <non_terminal_10> → T21 …
16. <non_terminal_10> → T22 …
17. <non_terminal_10> → <non_terminal_11> …
18. <non_terminal_11> → T24 …

- Error Recovery

```
void non_terminal_10 ()
{
        if (current_token == T21)
        { // Use rule 15
        }
        else if (current_token == T22)
        { // Use rule 16
        }
        else if (current_token == T24)
        { // Use rule 17
        }
        else
        // No applicable rule
                call error_routine;
        return;
}
```

Calculating First and Follow Sets – Procedure A



1. <program> → **begin** <stmt_list> **end**
2. <stmt_list> → <stmt> <stmt_tail>
3. <stmt_tail> →; <stmt_list>
4. <stmt_tail> → λ
5. <stmt> → <var> **=** <expression>
6. <var> → **A**
7. <var> → **B**
8. <var> → **C**
9. <expression> → <var><expr_tail>
10. <expr_tail> → + <var><expr_tail>
11. <expr_tail> → * <var><expr_tail>
12. <expr_tail> → λ

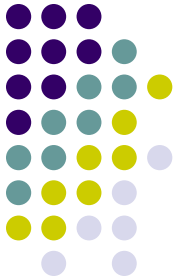
A. For each rule of the form

$<nt_i> \rightarrow T_k \ldots$

T_k is included in the first set of $<nt_i>$

<program> firsts += BEGIN_TOK (1)
{BEGIN_TOK ()}
<stmt_tail> firsts += SEMI_TOK (3)
{SEMI_TOK (3)}
<var> firsts += A_TOK (6)
<var> firsts += B_TOK (7)
<var> firsts += C_TOK (8)
{A_TOK (6), B_TOK (7), C_TOK (8)}
<expr_tail> firsts += PLUS_TOK (10)
<expr_tail> first += MULT_TOK (11)
{PLUS_TOK (10), MULT_TOK (11)}

Calculating First and Follow Sets – Procedure B



1. <program> → **begin** <stmt_list> **end**
2. <stmt_list> → <stmt> <stmt_tail>
3. <stmt_tail> →**;** <stmt_list>
4. <stmt_tail> → λ
5. <stmt> → <var> **=** <expression>
6. <var> → **A**
7. <var> → **B**
8. <var> → **C**
9. <expression> → <var><expr_tail>
10. <expr_tail> → + <var><expr_tail>
11. <expr_tail> → * <var><expr_tail>
12. <expr_tail> → λ

B. For each rule of the form

$<nt_i>$ → $<nt_j>$ …

if T_k is a member of the first set of $<nt_j>$ then Tk is included in the first set of $<nt_i>$

Firsts of nt_i += Firsts of nt_j

<stmt> firsts += <var> firsts

<stmt> firsts += {A_TOK (5), B_TOK (5), C_TOK (5)}

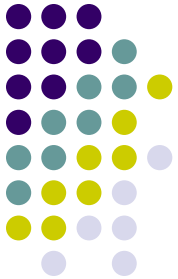
<expression> firsts += <var> firsts based on rule 9

<expression> firsts += {A_TOK (9), B_TOK (9), C_TOK (9)}

<stmt_list> firsts += <stmt> firsts

<stmt_list> firsts += {A_TOK (2), B_TOK (2), C_TOK (2)}

Calculating First and Follow Sets – Procedure C



1. <program> → **begin** <stmt_list> **end**
2. <stmt_list> → <stmt> <stmt_tail>
3. <stmt_tail> →**;** <stmt_list>
4. <stmt_tail> → λ
5. <stmt> → <var> **=** <expression>
6. <var> → **A**
7. <var> → **B**
8. <var> → **C**
9. <expression> → <var><expr_tail>
10. <expr_tail> → + <var><expr_tail>
11. <expr_tail> → * <var><expr_tail>
12. <expr_tail> → λ

C. For each rule of the form

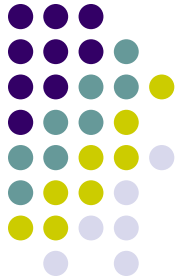
$<nt_i>$ → λ

if T_k is a member of the follow set of $<nt_i>$ then T_k is included in the first set of $<nt_i>$

<stmt_tail> firsts += <stmt_tail> follows (4)

<expr_tail> firsts += <expr_tail> follows (12)

Calculating First and Follow Sets – Procedure D



1. <program> → **begin** <stmt_list> **end**
2. <stmt_list> → <stmt> <stmt_tail>
3. <stmt_tail> →**;** <stmt_list>
4. <stmt_tail> → λ
5. <stmt> → <var> **=** <expression>
6. <var> → **A**
7. <var> → **B**
8. <var> → **C**
9. <expression> → <var><expr_tail>
10. <expr_tail> → + <var><expr_tail>
11. <expr_tail> → * <var><expr_tail>
12. <expr_tail> → λ

D. For each rule of the form

< > → … $<nt_i>$ T_k …

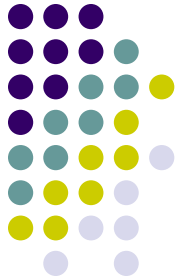
T_k is included in the follow set of $<nt_i>$

Right hand side – non-terminal immediately followed by a terminal.

<stmt_list> END_TOK
<stmt_list> follows += END_TOK (1)

<var> EQUAL_TOK
<var> follows += EQUAL_TOK (5)

Calculating First and Follow Sets – Procedure E



1. <program> → **begin** <stmt_list> **end**
2. <stmt_list> → <stmt> <stmt_tail>
3. <stmt_tail> →**;** <stmt_list>
4. <stmt_tail> → λ
5. <stmt> → <var> **=** <expression>
6. <var> → **A**
7. <var> → **B**
8. <var> → **C**
9. <expression> → <var><expr_tail>
10. <expr_tail> → + <var><expr_tail>
11. <expr_tail> → * <var><expr_tail>
12. <expr_tail> → λ

E. For each rule of the form

< > → … $<nt_i>$ $<nt_j>$ …

if T_k is a member of the first set of $<nt_j>$ then T_k is included in the follow set of $<nt_i>$

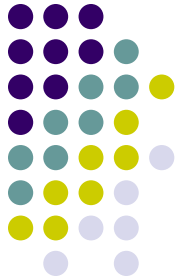
<stmt> <stmt_tail> (2)

<stmt> follows += <stmt_tail> firsts {SEMI_TOK (2)}

<var> <expr_tail> (9, 10, 11)

<var> follows += <expr_tail> firsts {PLUS_TOK (9,10,11), MULT_TOK (9,10,11)}

Calculating First and Follow Sets – Procedure F



1. <program> → **begin** <stmt_list> **end**
2. <stmt_list> → <stmt> <stmt_tail>
3. <stmt_tail> → **;** <stmt_list>
4. <stmt_tail> → λ
5. <stmt> → <var> **=** <expression>
6. <var> → **A**
7. <var> → **B**
8. <var> → **C**
9. <expression> → <var><expr_tail>
10. <expr_tail> → + <var><expr_tail>
11. <expr_tail> → * <var><expr_tail>
12. <expr_tail> → λ

F. For each rule of the form

$<nt_i>$ → … $<nt_j>$

if T_k is a member of the follow set of $<nt_i>$ then T_k is included in the follow set of $<nt_j>$

<stmt_tail> follows += <stmt_list> follows (2)

<stmt_list> follows += <stmt_tail> follows (3)

<expression> follows += <stmt> follows (5)

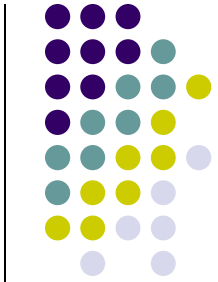
<expr_tail> follows += <expression> follows (9)

<expr_tail> follows += <expr_tail> follows (10, 11)

While sets are changing

A, B, D, E, F, C

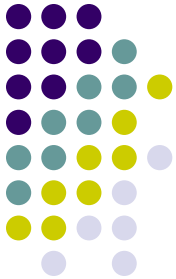
First and Follow Sets Calculation Flow



While sets are changing

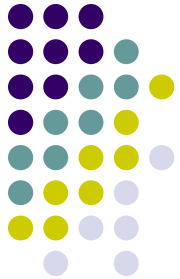
A, B, D, E, F, C

First and Follow Sets



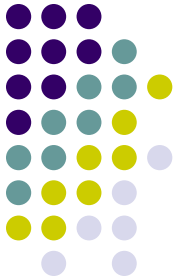
	First Set	Follow Set
<program>	BEGIN_TOK (1)	
<stmt_list>	A_TOK (2), B_TOK (2), C_TOK(2)	END_TOK
<stmt_tail>	SEMI_TOK (3), END_TOK (4)	END_TOK
<stmt>	A_TOK (5), B_TOK (5), C_TOK (5)	END_TOK, SEMI_TOK
<var>	A_TOK (6), B_TOK (7), C_TOK (8)	END_TOK, SEMI_TOK, EQUAL_TOK, PLUS_TOK, MULT_TOK
<expr>	A_TOK (9), B_TOK (9), C_TOK (9)	END_TOK, SEMI_TOK
<expr_tail>	PLUS_TOK (10), MULT_TOK (11), END_TOK (12), SEMI_TOK (12)	END_TOK, SEMI_TOK

Parse Table



T	BEGIN	END	SEMI	EQUAL	A	B	C	PLUS	MULT
<nt>	-TOK	-TOK	-TOK	-TOK	-TOK	-TOK	-TOK	-TOK	-TOK
<program>	1	Error	Error	Error	Error	Error	Error	Error	Error
<stmt_list>					2	2	2		
<stmt_tail>		4	3						
<stmt>					5	5	5		
<var>					6	7	8		
<expr>					9	9	9		
<expr_tail>		12	12					10	11

Exercise 3 Part 1 # 5



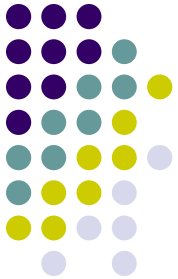
```
;;; This PL460 program contains a syntax error
;;; The display command only expects one argument

(define (main)
        (display "Hello" "World")
        (newline)
)

(main)
```

*Review, compile, and run the C++ program hw-3.cpp. How does the result differ from the PL460 result?

Exercise 3 Part 1 # 6



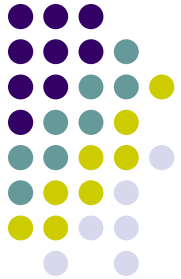
```
;;; This PL460 program contains a semantic error
;;; The variable 'Hello' has not been bound to a value

(define (main)
        (display Hello)
        (newline)
)

(main)
```

*Review, compile, and run the C++ program hw-4.cpp. How does the result differ from the PL460 result?

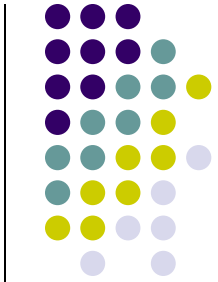
Exercise 3 Part 1 # 10



- `(+ 3 5) --> 8`
- `(+ 3 5/3) --> 14/3`
- `(+ 3 5.3) --> 8.3`
- `(+ 3/4 5) --> 23/4`
- `(+ 3.4 5) --> 8.4`

+ (addition)	integer	fraction	real
integer	integer		
fraction	fraction		
real	real		

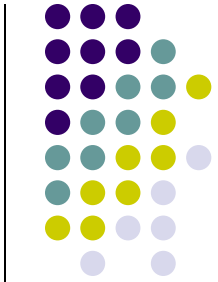
Part 1 # 11 subtraction



- `(- 3 5) -->`
- `(- 3 5/3) -->`
- `(- 3 5.3) -->`
- `(- 3/4 5) -->`
- `(- 3.4 5) -->`

- (subtraction)	integer	fraction	real
integer			
fraction			
real			

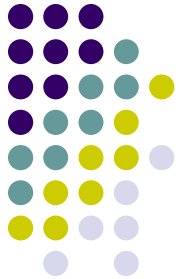
Part 1 # 11 multiplication



- `(* 3 5) -->`
- `(* 3 5/3) -->`
- `(* 3 5.3) -->`
- `(* 3/4 5) -->`
- `(* 3.4 5) -->`

- (multiplication)	integer	fraction	real
integer			
fraction			
real			

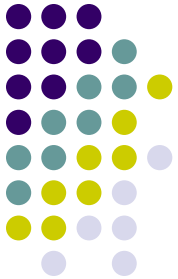
Part 1 # 11 division



- `(/ 3 5) -->`
- `(/ 3 5/3) -->`
- `(/ 3 5.3) -->`
- `(/ 3/4 5) -->`
- `(/ 3.4 5) -->`

- (division)	integer	fraction	real
integer			
fraction			
real			

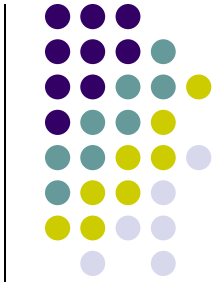
Exercise 3 Part 1 # 16



*Based on the above responses, describe the function of each of the PL460 functions represented:

a. `car:`
b. `cdr:`
c. `cons:`
d. `the ' symbol:`

Exercise 3 Part 1 # 20

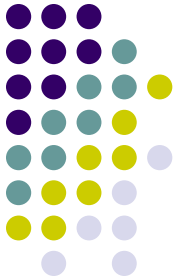


Note that the `number?` function requires a single argument.

a. `(number? 5)`
b. `(number? (- 2 5.5))`
c. `(number? 12/4)`
d. `(number? '0)`
e. `(number? 'abc)`
f. `(number? '(a b c))`
g. `(number? '())`
h. `(number? "(a b c) ")`

*What conclusions have you drawn about the functionality of the `number?` function?

Exercise 3 Part 1 # 23



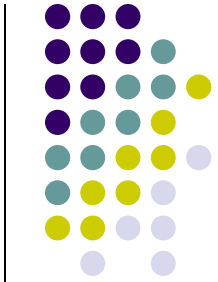
f. `(reciprocal 0)`

*Why does the last statement result in an error?

Replace that statement with a statement that displays the result of:
`(reciprocal 'abc)`

*Why does this statement result in an error?

Exercise 3 Part 1 # 24

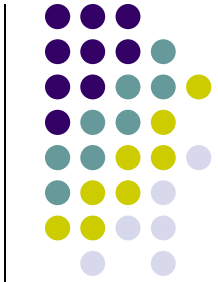


Note that the `zero?` function requires a single argument.

```
(zero? 5)
(zero? (- 2 5.5))
(zero? 12/4)
(zero? '0)
(zero? 'abc)
(zero? '(a b c))
(zero? '())
(zero? "(a b c) ")
```

*What conclusions have you drawn about the functionality of the `zero?` function?

Exercise 3 Part 1 # 25



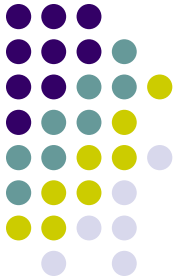
```
; This function will return the multiplicative reciprocal
; of a non-zero numeric input value
(define (reciprocal n)
        (if (and (number? n) (not (zero? n)))
                (/ 1 n)
        )
)
```

*What are the results now? Why? Modify the function to display an error message if the reciprocal cannot be calculated:

```
; This function will return the multiplicative reciprocal
; of a non-zero numeric input value
(define (reciprocal n)
        (if (and (number? n) (not (zero? n)))
                (/ 1 n)
                "invalid parameter"
        )
)
```

*What are the results now? What can you surmise about the use of an “if statement” in a PL460 function?

Exercise 3 Part 1 # 27

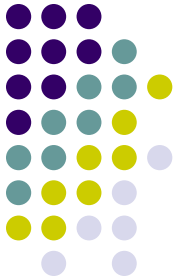


Note that the `list?` function requires a single argument.

```
(list? 5)
(list? (- 2 5.5))
(list? 12/4)
(list? '0)
(list? 'abc)
(list? '(a b c))
(list? '())
(list? "(a b c) ")
```

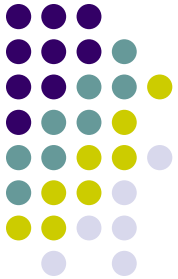
*What conclusions have you drawn about the functionality of the `list?` function?

Exercise 3 Part 1 # 28



- Ooops! Same as #24

Exercise 3 Part 1 # 29

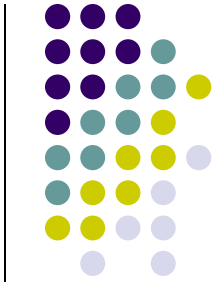


Note that the `null?` function requires a single argument.

```
(null? 5)
(null? (- 2 5.5))
(null? 12/4)
(null? '0)
(null? 'abc)
(null? '(a b c))
(null? '())
(null? "(a b c) ")
```

*What conclusions have you drawn about the functionality of the `null?` function?

Exercise 3 Part 1 # 30

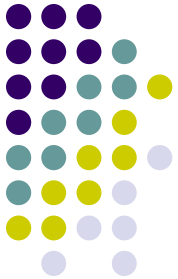


```
(define (if_ex_1 p)
        (if (= p 0)
                'equal
                (if (< p 0)
                    'negative
                    'positive
                )
        )
)

(define (if_ex_2 p)
        (if (= p 0)
                'equal
                (if (< p 0)
                    'negative
                )
        )
)
```

*How do the results produced by `if_ex_2` differ from those produced by `if_ex_1`? Why do you think these differences occur?

Exercise 3 Part 1 # 32

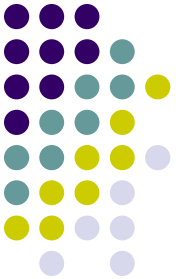


```
(define (cond_ex_1 p)
        (cond ((= p 0) 'equal)
              ((< p 0) 'negative)
              (else 'positive)
        )
)

(define (cond_ex_2 param)
        (cond ((= param 1) "The value is 1")
              ((= param 2) "The value is 2")
              ((> param 52) "The value is greater than 52")
          ((= (modulo param 5) 2) "The value ends in 2 or 7")
              (else "none of the above")
        )
)
```

*Can a `cond` function call be translated to a C++ switch statement? Why or why not?

Exercise 3 Part 1 # 36



Make a copy of the function `list_copy2` called `list_copy3`. Modify the line:

```
(cons (car ls) (list_copy2 (cdr ls)))
```

to:

```
(cons (car ls) (list_copy3 (cddr ls)))
```

Add statements to test this function.

*How does this change the functionality of `list_copy3` differ from the function of `list_copy2`?

Short Project Grammar

Character Sets

```
α = upper or lower alphabetic characters
η = digits 0 to 9
Θ = all typeable characters
```

Lexeme Regular Expression

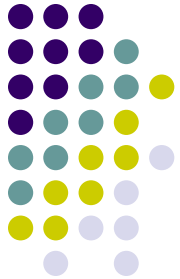
define | (|) | α(α|η|_)* | (+|-|λ)(η⁺ | η*.η⁺ | η⁺.η* | η⁺/η⁺) | "Θ*"
| **#f** | **#t** | **display** | **newline**

T = {DEFINE_T, LPAREN_T, RPAREN_T, IDENT_T, NUMLIT_T, STRLIT_T, FALSE_T, TRUE_T, DISPLAY_T, NEWLINE_T, EOF_T};

NT = {<program>, <more_defines>, <define>, <stmt_list>, <stmt>, <literal>, <logical_lit>, <param_list>, <action >}

S = <program>

Short Project Grammar

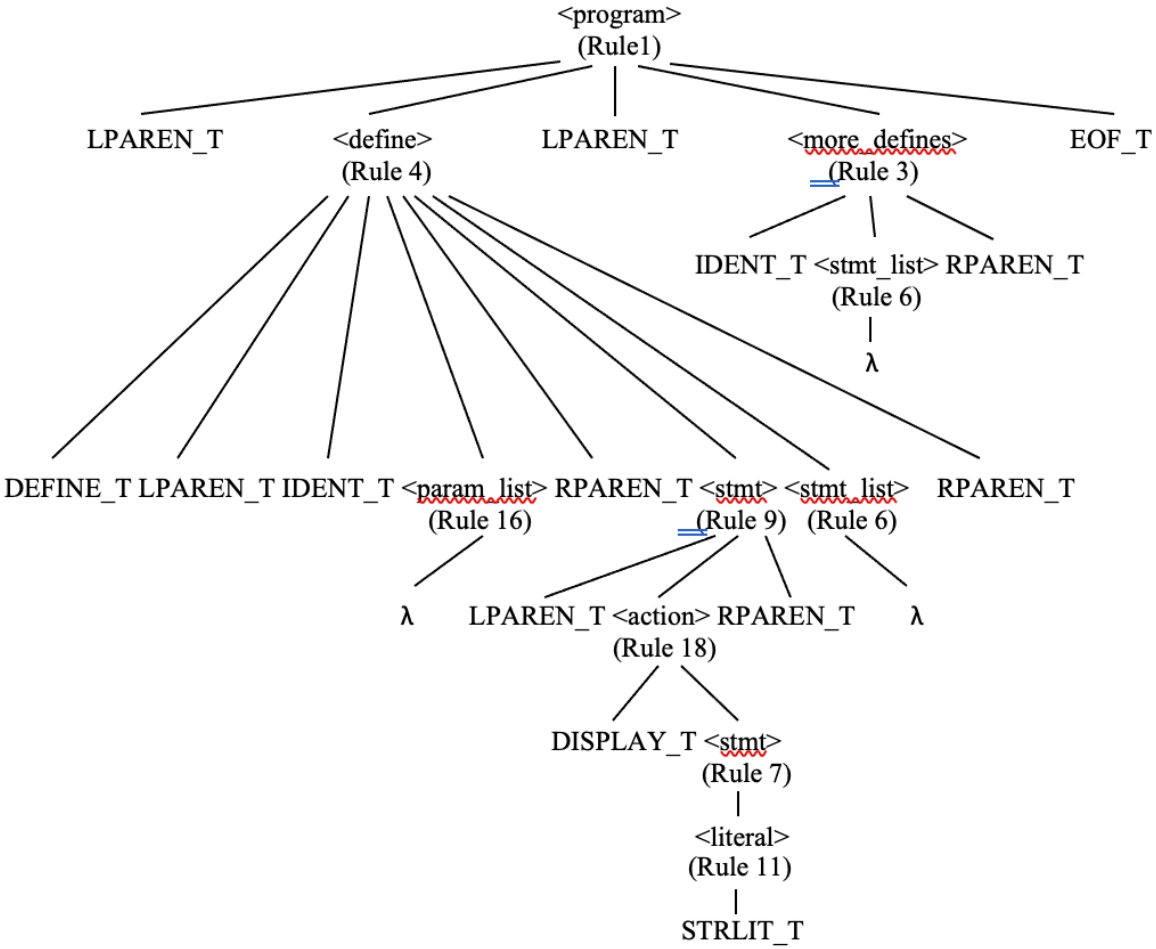


P = {

1. <program> -> LPAREN_T <define> LPAREN_T <more_defines> EOF_T
2. <more_defines> -> <define> LPAREN_T <more_defines>
3. <more_defines> -> IDENT_T <stmt_list> RPAREN_T
4. <define> -> DEFINE_T LPAREN_T IDENT_T <param_list> RPAREN_T <stmt> <stmt_list> RPAREN_T
5. <stmt_list> -> <stmt> <stmt_list>
6. <stmt_list> -> λ
7. <stmt> -> <literal>
8. <stmt> -> IDENT_T
9. <stmt> -> LPAREN_T <action> RPAREN_T
10. <literal> -> NUMLIT_T
11. <literal> -> STRLIT_T
12. <literal> -> <logical_lit>
13. <logical_lit> -> TRUE_T
14. <logical_lit> -> FALSE_T
15. <param_list> -> IDENT_T <param_list>
16. <param_list> -> λ
17. <action> -> IDENT_T <stmt_list>
18. <action> -> DISPLAY_T <stmt>
19. <action> -> NEWLINE_T

}

Short Grammar Program



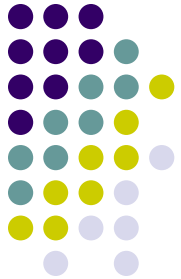
Tokens

LPAREN_T DEFINE_T LPAREN_T IDENT_T RPAREN_T LPAREN_T DISPLAY_T STRLIT_T RPAREN_T RPAREN_T LPAREN_T IDENT_T RPAREN_T EOF_T

Lexemes

```
(define (Team0) (display "Hello World")) (Team0)
```

Short Project Grammar



- Write a PL460 program that
  - Uses all possible lexemes (tokens)
  - Uses all 19 production rules
  - Is Syntactically correct

- Submit as Team[A-Z].pl460