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

- Spec and Framework posted
- Extra Credit
  - PL460 program that uses all 93 grammar rules
  - Thursday, 16 November 2023, 6:59 am
  - Draw from Exercise 3

- Suggestions
  - Create First and Follow sets
  - Start with sets for the “Short Grammar”
  - Add in the remaining grammar rules
  - Testing
First and Follow Sets

- **Firsts**
  - A terminal symbol $T_i$ is a member of the First Set of non-terminal symbol $\langle nt_j \rangle$ if $T_i$ can become the first terminal symbol in a complete expansion of $\langle nt_j \rangle$.

- **Follows**
  - A terminal symbol $T_i$ is a member of the Follow Set of non-terminal symbol $\langle nt_j \rangle$ if $T_i$ can become the first terminal symbol immediately following a complete expansion of $\langle nt_j \rangle$. 
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

```c
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;
    }
}
```
# Parse Table

<table>
<thead>
<tr>
<th>T</th>
<th>BEGIN</th>
<th>END</th>
<th>SEMI</th>
<th>EQUAL</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>PLUS</th>
<th>MULT</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;nt&gt;</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-TOK</td>
<td>-TOK</td>
<td>-TOK</td>
<td>-TOK</td>
<td>-TOK</td>
<td>-TOK</td>
<td>-TOK</td>
<td>-TOK</td>
<td>-TOK</td>
</tr>
<tr>
<td><code>&lt;program&gt;</code></td>
<td>1</td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
<td>Error</td>
</tr>
<tr>
<td><code>&lt;stmt_list&gt;</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;stmt_tail&gt;</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;stmt&gt;</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;var&gt;</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;expr&gt;</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>9</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;expr_tail&gt;</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>12</td>
<td></td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
# First and Follow Sets

<table>
<thead>
<tr>
<th></th>
<th>First Set</th>
<th>Follow Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;program&gt;</td>
<td>BEGIN_TOK (1)</td>
<td></td>
</tr>
<tr>
<td>&lt;stmt_list&gt;</td>
<td>A_TOK (2), B_TOK (2), C_TOK(2)</td>
<td>END_TOK</td>
</tr>
<tr>
<td>&lt;stmt_tail&gt;</td>
<td>SEMI_TOK (3), END_TOK (4)</td>
<td>END_TOK</td>
</tr>
<tr>
<td>&lt;stmt&gt;</td>
<td>A_TOK (5), B_TOK (5), C_TOK (5)</td>
<td>END_TOK, SEMI_TOK</td>
</tr>
<tr>
<td>&lt;var&gt;</td>
<td>A_TOK (6), B_TOK (7), C_TOK (8)</td>
<td>END_TOK, SEMI_TOK, EQUAL_TOK, PLUS_TOK, MULT_TOK</td>
</tr>
<tr>
<td>&lt;expr&gt;</td>
<td>A_TOK (9), B_TOK (9), C_TOK (9)</td>
<td>END_TOK, SEMI_TOK</td>
</tr>
<tr>
<td>&lt;expr_tail&gt;</td>
<td>PLUS_TOK (10), MULT_TOK (11), END_TOK (12), SEMI_TOK (12)</td>
<td>END_TOK, SEMI_TOK</td>
</tr>
</tbody>
</table>
Character Sets
\(\alpha\) = upper or lower alphabetic characters
\(\eta\) = digits 0 to 9
\(\Theta\) = all typeable characters

Lexeme Regular Expression

\[
\text{define} \mid ( \mid ) \mid \alpha(\alpha|\eta|\_)* \mid (+|-|\lambda)(\eta^+ \mid \eta^\ast \cdot \eta^+ \mid \eta^+ \cdot \eta^* \mid \eta^+ / \eta^+) \mid \"\Theta\"^* \mid \#f \mid \#t \mid \text{display} \mid \text{newline}
\]

\(T = \{\text{DEFINE\_T, LPAREN\_T, RPAREN\_T, IDENT\_T, NUMLIT\_T, STRLIT\_T, FALSE\_T, TRUE\_T, DISPLAY\_T, NEWLINE\_T, EOF\_T}\};\)

\(NT = \{<\text{program}> , <\text{more\_defines}> , <\text{define}> , <\text{stmt\_list}> , <\text{stmt}> , <\text{literal}> , <\text{logical\_lit}> , <\text{param\_list}> , <\text{action}> \}\)

\(S = <\text{program}>\)
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))
Calculating First and Follow Sets – Procedure A

- For each rule of the form
  a. \(<nt_i> \rightarrow T_k \ldots\>
  b. \(T_k\) is included in the first set of \(<nt_i>\)

- Which rules in the short grammar include the pattern needed for Procedure A?
Calculating First and Follow Sets – Procedure B

- For each rule of the form
  a. \( <nt_i> \rightarrow <nt_j> \ldots \)
  b. if \( T_k \) is a member of the first set of \( <nt_j> \) then \( T_k \) is included in the first set of \( <nt_i> \)

- Which rules in the short grammar include the pattern needed for Procedure B?
Calculating First and Follow Sets – Procedure C

- For each rule of the form
  a. \(<nt_i> \rightarrow \lambda\>
  b. if \(T_k\) is a member of the follow set of \(<nt_i>\) then \(T_k\) is included in the first set of \(<nt_i>\)

- Which rules in the short grammar include the pattern needed for Procedure C?
Calculating First and Follow Sets – Procedure D

- For each rule of the form
  a. \(< > \rightarrow \ldots <nt_i> T_k \ldots\)
  b. \(T_k\) is included in the follow set of \(<nt_i>\)

- Which rules in the short grammar include the pattern needed for Procedure D?
Calculating First and Follow Sets – Procedure E

- For each rule of the form
  a. $< \to \ ... <nt_i> <nt_j> \ ...$
  b. if $T_k$ is a member of the first set of $<nt_j>$ then $T_k$ is included in the follow set of $<nt_i>$

- Which rules in the short grammar include the pattern needed for Procedure E?
Calculating First and Follow Sets – Procedure F

- For each rule of the form
  a. \(<nt_i> \rightarrow \ldots <nt_j>\)
  b. if \(T_k\) is a member of the follow set of \(<nt_i>\) then \(T_k\) is included in the follow set of \(<nt_j>\)

- Which rules in the short grammar include the pattern needed for Procedure F?
Subprogram Terminology

- A **subprogram definition** describes the interface to and the actions of the subprogram abstraction.
- A **subprogram call** is the explicit request that a specific subprogram be executed.
- A subprogram is said to be **active** if, after having been called, it has begun execution but has not yet completed that execution.
- Two fundamental kinds of subprograms: **procedures and functions**.
- A **subprogram header**, which is the first part of the definition,
  - specifies that the following syntactic unit is a subprogram definition of some particular kind.
  - provides a name for the subprogram.
  - may specify a list of parameters.
- The **parameter profile** of a subprogram contains the number, order, and types of its formal parameters.
- The **protocol** of a subprogram is its parameter profile plus, if it is a function, its return type.
- **Formal parameters** are defined in the subprogram header.
- **Actual parameters** are provided in the subprogram call.
Procedures vs Functions

- Procedures do not have return values
- Procedures change the calling environment via call by reference and modification of shared variables (side effects)
- Functions have return values
- Functions should not create side effects
- Many languages use a hybrid approach
Local Referencing Environments

- Stack Frames
- Local variables
- Parameter passing methods
  - Pass by value
  - Pass by reference
  - Constant pass by reference
  - Pass by name
  - Implementing passing methods
Examples
Recursion in PL460

- Rules of Recursion - Determine
  - the unit of work
  - the base case (how to make it stop)
  - the external call (how to make it start)
  - the internal call(s) (how to keep it going)

- PL460 Examples
Recursive display example

(define (tailRec value)
    (display value) (display " ")
    (if (> value 1)
        (tailRec (/ value 2))
    )
)

(define (headRec value)
    (if (> value 1)
        (headRec (/ value 2))
    )
    (display value) (display " ")
)

(define (main)
    (display "((tailRec 32) --> ")
    (tailRec 32)
    (newline)
    (display "((headRec 32) --> ")
    (headRec 32)
    (newline)
)

(main)