



Design of Compilers

LL1 Parser User Guide

By:

Dalia Ayman Ahmed

CESS pre-junior

Presented to:

Dr. Sahar Haggag

Eng. Amr Saad

program \rightarrow stmt-seq

stmt-seq \rightarrow stmt stmt-seq`

stmt-seq` \rightarrow ; stmt-seq`

stmt-seq` $\rightarrow \epsilon$

stmt \rightarrow if-stmt

stmt \rightarrow repeat-stmt

stmt \rightarrow assign-stmt

stmt \rightarrow read-stmt

stmt \rightarrow write-stmt

if-stmt \rightarrow if exp then stmt-seq if-stmt`

if-stmt` \rightarrow end

if-stmt` \rightarrow else stmt-seq end

repeat-stmt \rightarrow repeat stmt-seq until exp

assign-stmt \rightarrow identifier := exp

read-stmt \rightarrow read identifier

write-stmt \rightarrow write exp

exp \rightarrow simple-exp exp`

exp` \rightarrow comparison-op simple-exp

exp` $\rightarrow \epsilon$

comparison-op \rightarrow <

comparison-op \rightarrow =

simple-exp \rightarrow term simple-exp`

simple-exp` \rightarrow addop term simple-exp`

simple-exp` $\rightarrow \epsilon$

addop \rightarrow +

addop \rightarrow -

term \rightarrow factor term`

term` \rightarrow mulop factor term`

term` $\rightarrow \epsilon$

mulop \rightarrow *

mulop \rightarrow /

factor \rightarrow (exp)

factor \rightarrow number

factor \rightarrow identifier

First Set of Non-Terminals:

$\text{first}(\text{program}) = \{\text{if, repeat, identifier, read, write}\}$

$\text{first}(\text{stmt-seq}) = \{\text{if, repeat, identifier, read, write}\}$

$\text{first}(\text{stmt-seq}') = \{;, \epsilon\}$

$\text{first}(\text{stmt}) = \{\text{if, repeat, identifier, read, write}\}$

$\text{first}(\text{if-stmt}) = \{\text{if}\}$

$\text{first}(\text{if-stmt}') = \{\text{end, else}\}$

$\text{first}(\text{repeat-stmt}) = \{\text{repeat}\}$

$\text{first}(\text{assign-stmt}) = \{\text{identifier}\}$

$\text{first}(\text{read-stmt}) = \{\text{read}\}$

$\text{first}(\text{write-stmt}) = \{\text{write}\}$

$\text{first}(\text{exp}) = \{(\text{, number, identifier}\}$

$\text{first}(\text{exp}') = \{<, =, \epsilon\}$

$\text{first}(\text{comparison-op}) = \{<, =\}$

$\text{first}(\text{simple-exp}) = \{(\text{, number, identifier}\}$

$\text{first}(\text{simple-exp}') = \{+, -, \epsilon\}$

$\text{first}(\text{addop}) = \{+, -\}$

$\text{first}(\text{term}) = \{(\text{, number, identifier}\}$

$\text{first}(\text{term}') = \{*, /, \epsilon\}$

$\text{first}(\text{mulop}) = \{*, /\}$

$\text{first}(\text{factor}) = \{(\text{, number, identifier}\}$

Follow Set of Non-Terminals:

$\text{follow}(\text{program}) = \{\$\}$

$\text{follow}(\text{stmt-seq}) = \{\$, \text{end, else, until}\}$

$\text{follow}(\text{stmt-seq}') = \{\$, \text{end, else, until}\}$

$\text{follow}(\text{stmt}) = \{;, \$, \text{end, else, until}\}$

$\text{follow}(\text{if-stmt}) = \{;, \$, \text{end, else, until}\}$

$\text{follow}(\text{if-stmt}') = \{;, \$, \text{end, else, until}\}$

$\text{follow}(\text{repeat-stmt}) = \{;, \$, \text{end, else, until}\}$

$\text{follow}(\text{assign-stmt}) = \{;, \$, \text{end, else, until}\}$

$\text{follow}(\text{read-stmt}) = \{;, \$, \text{end, else, until}\}$

$\text{follow}(\text{write-stmt}) = \{;, \$, \text{end, else, until}\}$

$\text{follow}(\text{exp}) = \{\text{then, }, \$, \text{end, else, until,)}\}$

$\text{follow}(\text{exp}') = \{\text{then, }, \$, \text{end, else, until,)}\}$

$\text{follow}(\text{comparison-op}) = \{(\text{, number, identifier}\}$

$\text{follow}(\text{simple-exp}) = \{<, =, \text{then, }, \$, \text{end, else, until,)}\}$

$\text{follow}(\text{simple-exp}') = \{<, =, \text{then, }, \$, \text{end, else, until,)}\}$

$\text{follow}(\text{addop}) = \{(\text{, number, identifier}\}$

$\text{follow}(\text{term}) = \{+, -, <, =, \text{then, }, \$, \text{end, else, until,)}\}$

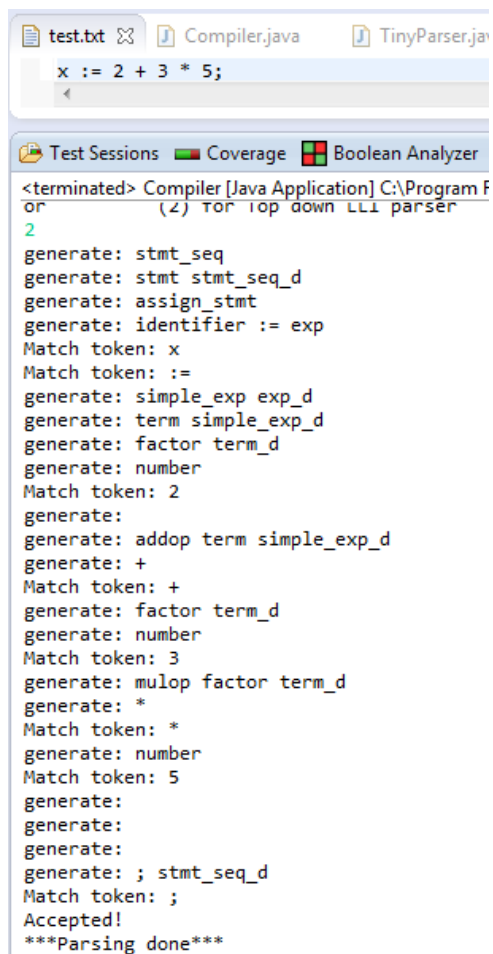
$\text{follow}(\text{term}') = \{+, -, <, =, \text{then, }, \$, \text{end, else, until,)}\}$

$\text{follow}(\text{mulop}) = \{(\text{, number, identifier}\}$

$\text{follow}(\text{factor}) = \{*, /, +, -, <, =, \text{then, }, \$, \text{end, else, until,)}\}$

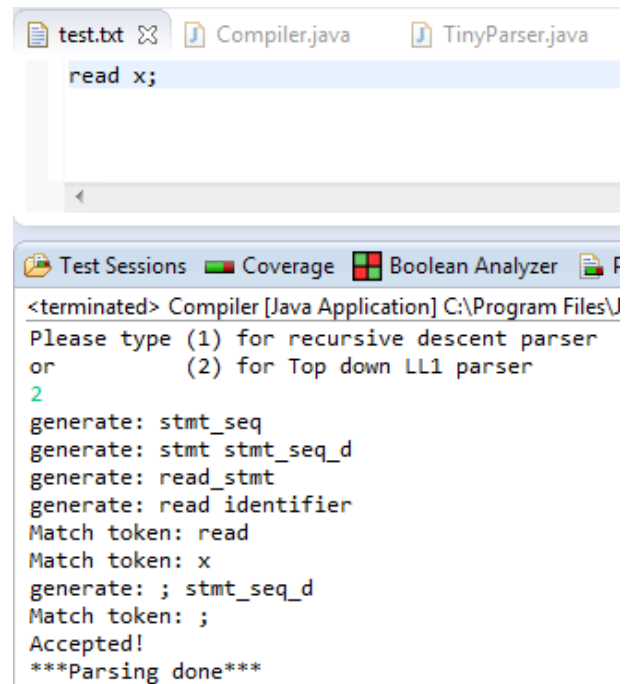
Parsing Table: attached in an Excel file.

Screenshots:



The screenshot shows a Java IDE with three tabs: test.txt, Compiler.java, and TinyParser.java. The test.txt tab contains the code `x := 2 + 3 * 5;`. Below the code editor, there is a toolbar with icons for Test Sessions, Coverage, and Boolean Analyzer. The output window displays the following text:

```
<terminated> Compiler [Java Application] C:\Program F
or (2) for Top down LL1 parser
2
generate: stmt_seq
generate: stmt stmt_seq_d
generate: assign_stmt
generate: identifier := exp
Match token: x
Match token: :=
generate: simple_exp exp_d
generate: term simple_exp_d
generate: factor term_d
generate: number
Match token: 2
generate:
generate: addop term simple_exp_d
generate: +
Match token: +
generate: factor term_d
generate: number
Match token: 3
generate: mulop factor term_d
generate: *
Match token: *
generate: number
Match token: 5
generate:
generate:
generate: ; stmt_seq_d
Match token: ;
Accepted!
***Parsing done***
```



The screenshot shows a Java IDE with three tabs: test.txt, Compiler.java, and TinyParser.java. The test.txt tab contains the code `read x;`. Below the code editor, there is a toolbar with icons for Test Sessions, Coverage, and Boolean Analyzer. The output window displays the following text:

```
<terminated> Compiler [Java Application] C:\Program Files\
Please type (1) for recursive descent parser
or (2) for Top down LL1 parser
2
generate: stmt_seq
generate: stmt stmt_seq_d
generate: read_stmt
generate: read identifier
Match token: read
Match token: x
generate: ; stmt_seq_d
Match token: ;
Accepted!
***Parsing done***
```

```
test.txt Compiler.java TinyP
if x = 2
then
write y;
end;

Test Sessions Coverage Boolean Ar
<terminated> Compiler [Java Application] C:\Pro
generate: factor term_d
generate: number
Match token: 2
generate:
generate:
Match token: then
generate: stmt stmt_seq_d
generate: write_stmt
generate: write_exp
Match token: write
generate: simple_exp exp_d
generate: term simple_exp_d
generate: factor term_d
generate: identifier
Match token: y
generate:
generate:
generate:
generate: ; stmt_seq_d
Match token: ;
generate:
generate: end
Match token: end
generate: ; stmt_seq_d
Match token: ;
Accepted!
***Parsing done***
```

```
test.txt Compiler.java T
if x = 2
then
write y;

Test Sessions Coverage Boolean
<terminated> Compiler [Java Application] C
generate: write_exp
Match token: write
generate: simple_exp exp_d
generate: term simple_exp_d
generate: factor term_d
generate: identifier
Match token: y
generate:
generate:
generate:
generate: ; stmt_seq_d
Match token: ;
Parsing Stack end error
** Parsing error **
```