LL grammar

xpojez00, xbudin05

October 31, 2021

1 Introduction

Our approach of creating LL table was to start from the easier **and** smaller parts **and** work our way to more complex non-terminals. At first, we filled the terminal set with all valid tokens. Our non-terminal **and** rule sets were empty. Starting from the variable declaration, value assignments **and** conditions. We are planning on implementing LL grammar using Predictive parsing.

2 Terminal set

$T = \{id, integer, number, string, "-", "+", "*", "/", "//", " : ", ", ", "\#", "(", ")", " < ", " <= ", " > ",$
$" >= ", "..", " = ", " == ", " = ",$ **and**, **boolean**, **do**, **else**, **elseif**, **end**, **false**, **function**, **global**, **if**,
integer, **local**, **nil**, **not**, **number**, **or**, **require**, **return**, **string**, **then**, **true**, **while**$\}$

3 Non-Terminal set

$NT = \{< program >, < global\_scope >, < function\_declare >, < function\_define >, < function\_call >$
$, < parameters >, < parameter >, < parameter\_name >, < parameters\_defined >, < parameter\_defined >$
$, < returning >, < scope >, < declare >, < id >, < if >, < while >, < return >, < declare\_assign >$
$, < assign >, < condition >, < condition\_branch >, < lvalues >, < lvalue >, < rvalues >, <$
$rvalue >, < expression >, < expression\_2 >, < expression\_3 >, < datatypes >, < datatype >, <$
$unary\_operator >, < binary\_operator >\}$

4 Rule set

$< program >$	$\rightarrow$ **require**$.string. < global\_scope >$
$< global\_scope >$	$\rightarrow < function\_declare >$
$< global\_scope >$	$\rightarrow < function\_define >$
$< global\_scope >$	$\rightarrow < function\_call >$
$< function\_declare >$	$\rightarrow$ **global**$.id." : ".$**function**$."(". < parameters > .")". < returning >$
$< function\_define >$	$\rightarrow$ **function**$.id."(". < parameters\_defined > .")". < returning > . < scope >$ **.end**
$< function\_call >$	$\rightarrow id."(". < rvalues > .")"$
$< parameters >$	$\rightarrow < parameters > .", ". < parameter >$
$< parameters >$	$\rightarrow < parameter >$
$< parameters >$	$\rightarrow \epsilon$
$< parameter >$	$\rightarrow < parameter\_name > . < datatype >$
$< parameter\_name >$	$\rightarrow id." : "$
$< parameter\_name >$	$\rightarrow \epsilon$
$< parameters\_defined >$	$\rightarrow < parameters\_defined > .", ". < parameter\_defined >$

$$< parameters\_defined > \quad \rightarrow \; < parameter\_defined >$$
$$< parameters\_defined > \quad \rightarrow \; \epsilon$$
$$< parameter\_defined > \quad \rightarrow \; id."\,:\,".< datatype >$$

$$< returning > \quad \rightarrow \; "\,:\,".< datatypes >$$
$$< returning > \quad \rightarrow \; \epsilon$$

$$< scope > \quad \rightarrow \; < declare >$$
$$< scope > \quad \rightarrow \; < id >$$
$$< scope > \quad \rightarrow \; < if >$$
$$< scope > \quad \rightarrow \; < while >$$
$$< scope > \quad \rightarrow \; < return >$$

$$< declare > \quad \rightarrow \; \textbf{local}.< lvalues >."\,:\,".< datatypes > .< declare\_assign >$$
$$< id > \quad \rightarrow \; id."(".< rvalues > .")"$$
$$< id > \quad \rightarrow \; id.< assign >$$
$$< id > \quad \rightarrow \; id.",".< lvalues > .< assign >$$
$$< if > \quad \rightarrow \; \textbf{if}.< condition > .\textbf{end}$$
$$< while > \quad \rightarrow \; \textbf{while}.< expression > .\textbf{do}.< scope > .\textbf{end}$$
$$< return > \quad \rightarrow \; \textbf{return}.< rvalues >$$
$$< return > \quad \rightarrow \; \textbf{return}$$

$$< declare\_assign > \quad \rightarrow \; < assign >$$
$$< declare\_assign > \quad \rightarrow \; \epsilon$$
$$< assign > \quad \rightarrow \; "\,=\,".< rvalues >$$
$$< condition > \quad \rightarrow \; < expression > .\textbf{then}.< scope > .< condition\_branch >$$

$$< condition\_branch > \quad \rightarrow \; \textbf{else}.< scope >$$
$$< condition\_branch > \quad \rightarrow \; \textbf{elseif}.< condition >$$
$$< condition\_branch > \quad \rightarrow \; \epsilon$$

$$< lvalues > \quad \rightarrow \; < lvalues > .",".< lvalue >$$
$$< lvalues > \quad \rightarrow \; < lvalue >$$
$$< lvalue > \quad \rightarrow \; id$$
$$< rvalues > \quad \rightarrow \; < rvalues > .",".< rvalue >$$
$$< rvalues > \quad \rightarrow \; < rvalue >$$
$$< rvalue > \quad \rightarrow \; < expression >$$

$$< expression > \quad \rightarrow \; < expression > .< binary\_operator > .< expression\_2 >$$
$$< expression > \quad \rightarrow \; < expression\_2 >$$
$$< expression\_2 > \quad \rightarrow \; < unary\_operator > .< expression\_3 >$$
$$< expression\_2 > \quad \rightarrow \; < expression\_3 >$$
$$< expression\_3 > \quad \rightarrow \; "(".< expression > .")"$$

$$< expression\_3 > \quad \rightarrow \; string$$
$$< expression\_3 > \quad \rightarrow \; number$$
$$< expression\_3 > \quad \rightarrow \; integer$$
$$< expression\_3 > \quad \rightarrow \; id$$
$$< expression\_3 > \quad \rightarrow \; id."(".< rvalues > .")"$$
$$< expression\_3 > \quad \rightarrow \; \textbf{true}$$
$$< expression\_3 > \quad \rightarrow \; \textbf{false}$$
$$< expression\_3 > \quad \rightarrow \; \textbf{nil}$$

$$< datatypes > \quad \rightarrow \; < datatypes > .",".< datatype >$$
$$< datatypes > \quad \rightarrow \; < datatype >$$

$< datatype >$	$\rightarrow$	**integer**
$< datatype >$	$\rightarrow$	**number**
$< datatype >$	$\rightarrow$	**string**
$< datatype >$	$\rightarrow$	**boolean**
$< unary\_operator >$	$\rightarrow$	"#"
$< unary\_operator >$	$\rightarrow$	**not**
$< binary\_operator >$	$\rightarrow$	" $-$ "
$< binary\_operator >$	$\rightarrow$	" $+$ "
$< binary\_operator >$	$\rightarrow$	" $*$ "
$< binary\_operator >$	$\rightarrow$	"/"
$< binary\_operator >$	$\rightarrow$	"//"
$< binary\_operator >$	$\rightarrow$	".."
$< binary\_operator >$	$\rightarrow$	" $<$ "
$< binary\_operator >$	$\rightarrow$	" $<=$ "
$< binary\_operator >$	$\rightarrow$	" $>$ "
$< binary\_operator >$	$\rightarrow$	" $>=$ "
$< binary\_operator >$	$\rightarrow$	" $==$ "
$< binary\_operator >$	$\rightarrow$	" $=$ "
$< binary\_operator >$	$\rightarrow$	**and**
$< binary\_operator >$	$\rightarrow$	**or**
$<>$	$\rightarrow$	