LL grammar

xpojez00, xbudin05

November 7, 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

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

3 Non-Terminal set

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

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." : "
$< parameter\_name >$	$\rightarrow \epsilon$
$< parameters\_defined >$	$\rightarrow < parameters\_defined >$ ".",". $< parameter\_defined >$

$< parameters\_defined > \rightarrow < parameter\_defined >$
$< parameters\_defined > \rightarrow \epsilon$
$< parameter\_defined > \rightarrow id.\text{" : ".} < datatype >$

$< returning > \rightarrow \text{" : ".} < datatypes >$
$< returning > \rightarrow \epsilon$

$< scope > \rightarrow < scope > . < statement >$
$< scope > \rightarrow < statement >$

$< statement > \rightarrow < declare >$
$< statement > \rightarrow < id >$
$< statement > \rightarrow < if >$
$< statement > \rightarrow < while >$
$< statement > \rightarrow < return >$
$< statement > \rightarrow \epsilon$

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

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

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

$< lvalues > \rightarrow < lvalues > .\text{",".} < lvalue >$
$< lvalues > \rightarrow < lvalue >$
$< lvalue > \rightarrow id$
$< rvalues > \rightarrow < rvalues > .\text{",".} < rvalue >$
$< rvalues > \rightarrow < rvalue >$
$< rvalue > \rightarrow < expression >$

$< expression > \rightarrow < expression > . < binary\_operator > . < expression\_2 >$
$< expression > \rightarrow < expression\_2 >$
$< expression\_2 > \rightarrow < unary\_operator > . < expression\_3 >$
$< expression\_2 > \rightarrow < expression\_3 >$
$< expression\_3 > \rightarrow \text{"(".} < expression > .\text{")"}$

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

$< datatypes >$ $\rightarrow < datatypes >$ ”,”. $< datatype >$
$< datatypes >$ $\rightarrow < datatype >$

$< datatype >$ $\rightarrow$ **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**
$<>$ $\rightarrow$
