University of Toronto

CSC 488 / CSC2107 Compilers and Interpreters **Winter 2022**

Tutorial 2: Define Grammar Rules

Assignment 2: Define Lexer rules for Minic language
Will be released after a1 is due.

Today's Agenda:

- Introduction to ANTLR4
- Toy Example
- Syntax and .g4 files
- Applying it to MiniC

ANTLR4

- Modern Parser Generator used to build languages
- Used in A2 and A3 to define the MiniC grammar
- You should have ANTLR 4.9 and the C++ runtime installed on your machine from A1
- Reference: https://github.com/antlr/antlr4/tree/master/doc

Toy Example

- Let's define a simple grammar and save it in `expr.g4`
  Let's define a grammar to calculate add/subtract sequential operations. E.g. `1+2-3-4+5+6`

```
grammar Expr;
prog: (expr NEWLINE)* ;
expr: INT
    | expr ('+'|'-') expr
    ;
NEWLINE: [\r\n]+ ;
INT:     [0-9]+ ;
```

- How about adding multiply/divide operations and '()'? E.g. `100*(4+6)`

```
grammar Expr;
prog: (expr NEWLINE)* ;
expr: INT
    | expr ('+'|'-') expr
    | '(' expr ')'
```

```
        | expr ('*' | '/') expr
        ;
    NEWLINE: [\r\n]+ ;
    INT:     [0-9]+ ;
```

Is that correct???

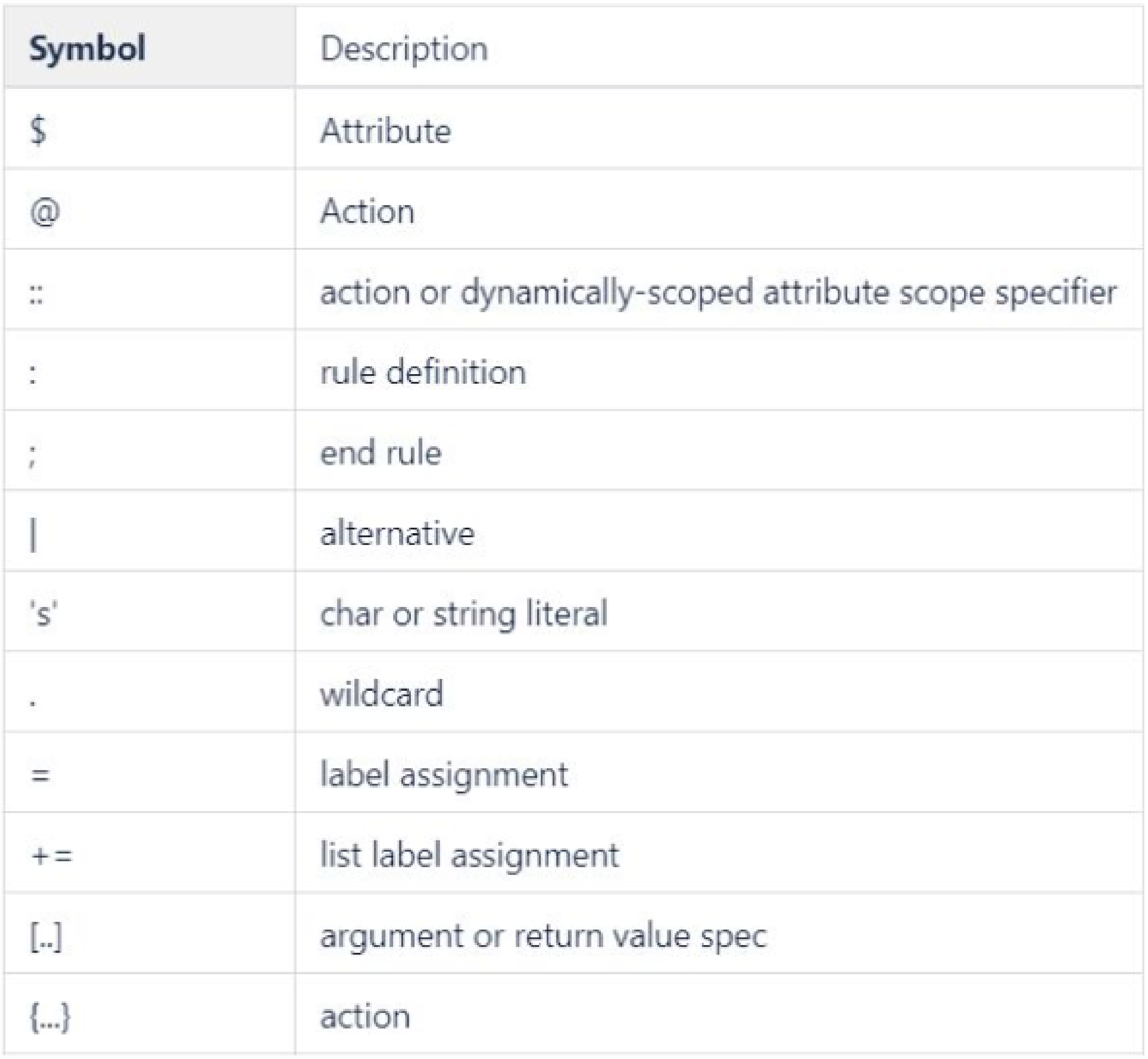
Correct version:

```
    grammar Expr;
    prog: (expr NEWLINE)* ;
    expr: INT
        | expr ('*' | '/') expr
        | expr ('+'|'-') expr
        | '(' expr ')'
        ;
    NEWLINE: [\r\n]+ ;
    INT:     [0-9]+ ;
```

- Operator precedence is important!!!
- Simple demo to run the grammar.

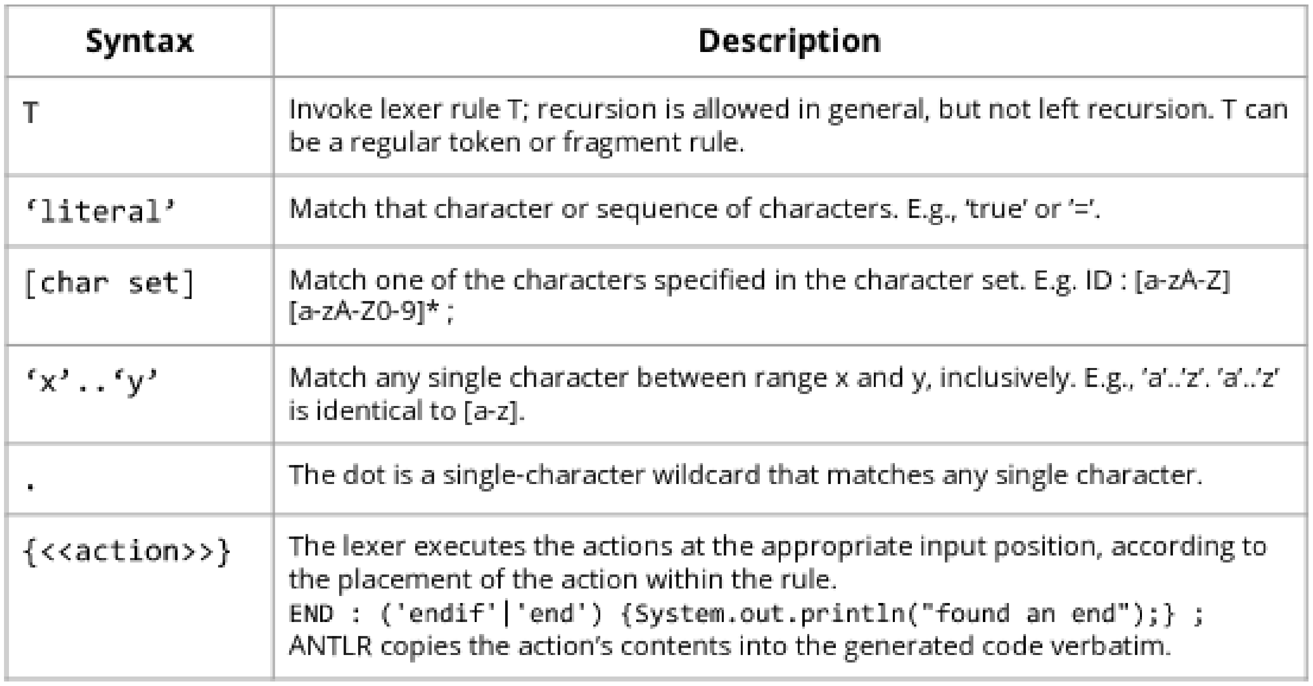
```
$ antlr4 expr.g4
$ javac Expr*.java
$ grun Expr prog -gui
100*(3+4)
^d
```

Common Symbols you maybe use



Symbol	Description
$	Attribute
@	Action
::	action or dynamically-scoped attribute scope specifier
:	rule definition
;	end rule
|	alternative
's'	char or string literal
.	wildcard
=	label assignment
+=	list label assignment
[..]	argument or return value spec
{...}	action

Lexer Rules



Syntax	Description
`T`	Invoke lexer rule T; recursion is allowed in general, but not left recursion. T can be a regular token or fragment rule.
`‘literal’`	Match that character or sequence of characters. E.g., ’true’ or ’=’.
`[char set]`	Match one of the characters specified in the character set. E.g. ID : [a-zA-Z] [a-zA-Z0-9]* ;
`‘x’..‘y’`	Match any single character between range x and y, inclusively. E.g., ’a’..’z’. ’a’..’z’ is identical to [a-z].
`.`	The dot is a single-character wildcard that matches any single character.
`{<<action>>}`	The lexer executes the actions at the appropriate input position, according to the placement of the action within the rule. `END : ('endif'|'end') {System.out.println("found an end");} ;` ANTLR copies the action’s contents into the generated code verbatim.

https://github.com/antlr/antlr4/blob/master/doc/lexer-rules.md

A2 MiniC Parsing

- Given the language specifications, fill out the .g4 file
- Don’t overthink the rules, we have simplified most of it for you
- You can use the Expr.g4 file to get started on the syntax
- You can check for correctness using the provided tester script. Your program will be evaluated based on public tests and private tests

Questions

- **How can I enforce operator precedence? Precedence is top to bottom (alternatives at top are first)**
- How do I define an epsilon? Leave it as blank (i.e. blank alternative)!
- How can I create the AST for A3? We have mostly defined the nodes for you. Specify the actions within the .g4 file to initialize the nodes
- **Try to be smart to use antlr4, javac, grun commands to test your .g4 file. Minic.g4 cannot directly be compiled by antlr4. (Comment something)**