

CSC 3315, Spring 2023, Assignment 1

A Grammar for the X- Programming Language BNF Grammar Description & Lexical Analyzer (Scanner)

Done by

Khalil Mouhah

Meriem Kaanoune

Ziad Elhajjame

# Grammar

* The X- programming language is defined by the following BNF grammar:

program = { declaration } { statement }.

declaration = "var" identifier ":" type.

type = "int" | "float" | "bool".

statement = assignment | if\_statement | while\_loop | for\_loop.

assignment = identifier "=" expression ";".

if\_statement = "if" condition "then" { statement } [ "else" { statement } ] "end" "if" ";".

while\_loop = "while" condition "do" { statement } "end" "while" ";".

for\_loop = "for" identifier "=" expression "to" expression "do" { statement } "end" "for" ";".

condition = expression ( "==" | "!=" | ">" | "<" | ">=" | "<=" ) expression.

expression = term { ( "+" | "-" ) term }.

term = factor { ( "\*" | "/" ) factor }.

factor = number | bool\_literal | identifier | "(" expression ")" | "-" factor.

number = digit { digit } [ "." digit { digit } ].

bool\_literal = "true" | "false".

identifier = letter { letter | digit | "\_" }.

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".

letter = "a" | "b" | "c" | ... | "z" | "A" | "B" | "C" | ... | "Z".

* Explanation of grammar:
* The program is made up of zero or more declarations followed by zero or more statements.
* A declaration starts with the keyword "var" followed by an identifier, a colon, and a type.
* The type can be "int", "float", or "bool".
* A statement can be an assignment, an if statement, a while loop, or a for loop.
* An assignment consists of an identifier followed by an equals sign, an expression, and a semicolon.
* An if statement starts with the keyword "if" followed by a condition in parentheses, the keyword "then", and a block of zero or more statements. It can also have an optional "else" block of zero or more statements, followed by the keywords "end" and "if".
* A while loop starts with the keyword "while" followed by a condition in parentheses, the keyword "do", and a block of zero or more statements. It ends with the keywords "end" and "while".
* A for loop starts with the keyword "for" followed by an identifier, an equals sign, an expression, the keyword "to", an expression, the keyword "do", and a block of zero or more statements. It ends with the keywords "end" and "for".
* A condition is an expression followed by a comparison operator ("==", "!=", ">", "<", ">=", "<=") followed by another expression.
* An expression is made up of one or more terms separated by addition or subtraction operators.
* A term is made up of one or more factors separated by multiplication or division operators.
* A factor can be a number, a boolean literal ("true" or "false"), an identifier, a parenthesized expression, or a negated factor.
* A number is made up of one or more digits, optionally followed by a decimal point and one or more digits.
* An identifier starts with a letter and can be followed by zero or more letters, digits, or underscores.
* A digit is any single digit from 0 to 9.
* A letter is any uppercase or lowercase letter in the English alphabet.

# Code Structure

* The code structure for the XmC lexer will be divided into the following modules:

Out implementation of the lexer for the given X- programming language uses JFlex, a Java-based lexical analyzer generator. The overall structure of the code is as follows:

* The X- language tokens are defined as regular expressions in a .flex file.
* The JFlex tool is used to generate a Java class file containing the lexer implementation based on the defined tokens.
* The Java class file is compiled into a bytecode file, which can be run using the Java Virtual Machine (JVM).
* The main program takes an input file name as a command-line argument or reads from standard input if no file name is provided.
* The lexer reads the input file or standard input and tokenizes it according to the X- language tokens.
* The tokens are output to the console along with their corresponding line numbers and token IDs.

# Test cases

* Example 1:

INPUT:

var x: int;

var y: bool;

if x == 5 then

y = true;

else

y = false;

end if

EXPECTED OUTPUT:

Line 1 Token #29: var

Line 1 Token #2: x

Line 1 Token #8: :

Line 1 Token #9: int

Line 2 Token #29: var

Line 2 Token #2: y

Line 2 Token #8: :

Line 2 Token #10: bool

Line 3 Token #12: if

Line 3 Token #2: x

Line 3 Token #37: ==

Line 3 Token #4: 5

Line 3 Token #13: then

Line 4 Token #2: y

Line 4 Token #37: ==

Line 4 Token #18: true

Line 5 Token #14: else

Line 6 Token #2: y

Line 6 Token #37: ==

Line 6 Token #19: false

Line 7 Token #15: end

Line 7 Token #16: if

Line 7 Token #38: ;

* Example 2:

INPUT:

var i: int;

i = 0;

while i <= 10 do

i = i + 2;

end while;

EXPECTED OUTPUT:

Line 1 Token #29: var

Line 1 Token #2: i

Line 1 Token #8: :

Line 1 Token #9: int

Line 2 Token #2: i

Line 2 Token #37: ==

Line 2 Token #3: 0

Line 2 Token #38: ;

Line 3 Token #17: while

Line 3 Token #2: i

Line 3 Token #39: <=

Line 3 Token #4: 10

Line 3 Token #18: do

Line 4 Token #2: i

Line 4 Token #37: ==

Line 4 Token #2: i

Line 4 Token #33: +

Line 4 Token #3: 2

Line 4 Token #38: ;

Line 5 Token #19: end

Line 5 Token #20: while

Line 5 Token #38: ;

* Example 3:

INPUT:

var x: bool;

for i = 1 to 5 do

x = (i != 3) && (i > 2);

end for;

OUTPUT:

Line 1 Token #29: var

Line 1 Token #2: x

Line 1 Token #8: :

Line 1 Token #10: bool

Line 2 Token #21: for

Line 2 Token #2: i

Line 2 Token #37: ==

Line 2 Token #1: 1

Line 2 Token #22: to

Line 2 Token #1: 5

Line 2 Token #23: do

Line 3 Token #2: x

Line 3 Token #37: ==

Line 3 Token #10: (

Line 3 Token #2: i

Line 3 Token #38: !=

Line 3 Token #4: 3

Line 3 Token #35: &&

Line 3 Token #10: (

Line 3 Token #2: i

Line 3 Token #39: >

Line 3 Token #1: 2

Line 3 Token #11: )

Line 3 Token #12: )

Line 3 Token #38: ;

# Feedback

* Feedback – your overall comments on the assignment: time you spent on it, the most difficult and/or more interesting part…