Bilkent University | Department of Computer Engineering

CS 315: Programming Languages

Lexical Analyser for a Programming Language for an Integer Language

Language: ***HazardCat***

13.10.2023

Section: 2

Group: 14

- Musa Yiğit Yayla. 22003108
- Maryam Azimli. 22101528

Instructor:

Aynur Dayanik

1. <main > =: main: {<program> }
2. <program> :: = <statements>
3. <statements>::= <statement>|< statements> +<statement>
4. < statement>:: =<cond_statement> | <loop> | <single_statement>|<statement>
5. <cond_statement>::= < If_statement >|<Else_If_statem>| <else_statement>
6. <If_statement> ::= if (<exprs>){<statements>}
7. <Else_If_statem>::= < If_statement> else_if (<exprs>){< statements>}
8. <else_statement>::= else{<statements>} can we write else without if?
9. <loop>::= <for>|<while>
10. <for>::=for (<varName>=<expr>;<expr >;<doInLoops>){<statements>};
11. <while>::= while (<expr>){statements};
12. <single_statement>::=<varDeclaration>|<return_st>|< arrDec >| <varAssign >| < constIntDecAssign>| < constStringDecAssign >| <varDecAssign>|<func_call>
13. <varDeclaration >::=let<varName>=<expr>| let <varName>=< arithmeticOperator>;
14. <varAssign >::=<varName>=<number>;
15. < varDecAssign >:= let <varAssign>;
16. < constIntDecAssign >::= const <varName>=<number>;
17. < constStringDecAssign >::= string <varName>=<string>;
18. <return_st>::=return <exprs>;
19. <arrDec>::= list <varName >;
20. <arrInit>::= <varName>={<insideOfList>}
21. <insideOfList>::=<varName>|<constant>|<varName>,<insideOfList>|<constant>,<insideOfList>;
22. <arraySizeSpecifier >::= ~

23.<func_call>::= func varName(exprs);

24.<func>::= func varName(parameters){<statements><return_st>}

25.<exprs>::= <expr>|<exprs><expr>;

26.<expr>::=(<expr>)|< arithmeticOperator >|
<boolOperator>|<varName>|<constant>|<expr><compare><expr>;
precedence

27.<varName>::=<lower_lets>|<lower_lets>+<upper_let>|<lower_lets>+<uppe
r_lets>+<nums>;

28.<parameters>::=int<varName>|int<varName>,<parameters>

29.<read>::=read < readOperator > <exprs>;

30.< readOperator>::= >>

31.<string>::=<lower_lets>|upper_lets>|< digits>|<special_buts>

32.<lower_lets>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z |<lower_lets>

33.<upper_let>::=
A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|<upper_let>

34.<special_buts>::=/|.|,|*|&|^|%|$|#|@!|~

35.<compare>::=< | > | <= | >= | == | !=

36.<DoInLoops>::=<increment>|<decrement>|<expr>

37.<increment>::=<varName>++|++<varName>|

38.<decrement>::=<varName>--|--<varName>|

39.<plusEqual><varName>+=

40.<minusEqual><varName>-=

41.<negative>::=-(number)

42.<arithmeticOperator >::= <sum>| <subtract>| <multiply>| <divide>| <mod> |
<pow>

43.<sum>::=<varName>+<varName>|<varName>+<constant>|<constant> +
<constant>

44.<subtract>::=<varName>-<varName>|<varName>-<constant>|
<constant> - <constant>

45.<multiply>::=<varName>*<varName>|<varName>*<constant>|
<constant> * <constant>

46.<divide>::=<varName>/<varName>|<varName>/<constant>|
<constant> / <constant>

47.<mod>::=<varName>%<varName>|<varName>%<constant>|
<constant> % <constant>

48.<pow>::=<varName>^<varName>|<varName>^<constant>|
<constant> ^ <constant>

49.<boolOperator>::= <not>|<or>|<and>|<xor>

50.<not>::=!

51.<or>::= ||

52.<and>::=&&

53.<xor>::= ^^

54.<number>::=<number><digits>|<digits>| <negative><digits>|

55.<digits>::=0|1|2|3|4|5|6|7|8|9

56.<comments>::= #<string>;|#<string><comments>;

57.<print>::=print(<number>|<expr>|<string>)

58.<print_line>::=print_line+<print>

A paragraph explanation for each language construct (i.e. variables and terminals) detailing their intended usage and meaning, as well as all of the associated conventions:

Terminals:

lower_lets: These terminals represent lowercase letters (a-z) and are primarily intended for naming variables and strings in code.

upper_let: These terminals denote uppercase letters (A-Z) and serve the same purpose as lowercase letters, enabling to create case-sensitive variable and string names.

special_buts: This terminal stores various special characters such as '/', '.', ',', '*', '&', '^', '%', '$', '#', '@', '!', and which can be included in strings and varNames.

digits: The digits terminals represent numerical digits (0-9) used for numeric values.

string: The string terminals signify string literals, using lowercase letters, uppercase letters, numbers, and special characters.

compare: These terminals define comparison operators like '<,' '>', '<=,' '>=,' '==,' and '!='. They are mostly used for conditional statements and loops.

increment and decrement: Incrementing or decrementing within loops mostly.

plusEqual and minusEqual: for faster reassigning or updating the current value of varName.

arithmeticOperator: These terminals uses various arithmetic operators, such as addition, subtraction, multiplication, division, modulus, and exponentiation.

boolOperator: The boolOperator terminals define Boolean operators like 'not,' 'or,' 'and,' and 'xor.' Mostly are used in loops or/and cond_statements.

Non-Trivial Token Definitions:

Comments (e.g., #<string>): Comments start with a '#' symbol, followed by a string. Efficient for writing comments to better understand it.

Identifiers (e.g., <varName>): Variables are defined using a combination of lowercase and uppercase letters, digits, and special characters. Eg: elma23

Literals (e.g., <string>, <number>): String literals encompass a wide range of characters, including lowercase and uppercase letters, digits, and special characters, to represent text data. Numeric literals are represented by digits and can include a negative sign for negative numbers.

Reserved Words (e.g., ‘if,’ ‘else,’ ‘for,’ ‘while,’ ‘let,’ ‘string,’ ‘const,’ ‘return,’ ‘list,’ ‘func,’ ‘read,’ ‘print,’ ‘print_line,’ etc.): let is for declaring integer value to varName. If, else, else_if are for conditional statements.