CS 315

Project 2

,

PANTHER

Instructor: Halil Altay GÜVENİR

GROUP 4
Nursena KURUBAŞ 21602965 Sec01
İsmail Yavuzselim TAŞÇI 21602936 Sec01
Ayça Begüm TAŞÇIOĞLU 21600907 Sec01

BNF DESCRIPTION

<program> ::= begin <stmts> end \n

<stmts> ::= <stmt> \n
| <stmts> <stmt> \n

<stmt> ::= <matched>
| <unmatched>

<non_if_stmt>::= <empty>
| <comment>
| <logical_stmt>
| <method_declaration>
| <var_declaration>
| <assignment_stmt>
| <primitive_func>
| <io_stmt>
| <while_stmt>

<io_stmt> ::= <print_stmt> | <scan_stmt>

<print_stmt>::= print(<ident>)

<scan_stmt>::= scan (<ident>)

<logical_stmt>::=<and_stmt>
| <or_stmt>

<primitive_func> ::= move
| grab
| release
|<turn_func>
|<readData_func>
|<sendData_func>
|<receiveData_func>

<assignment_stmt> ::= <ident> = <nonlogical_expr>
| <ident> = <assignment_stmt>

<matched> ::= if < logical_expr >then\n <matched> else\n <matched> end_if
| <non_if_stmt>

<unmatched> ::= if < logical_expr > then\n <stmts> end_if
| if < logical_expr > then\n <matched> else\n <unmatched> end_if

<while_stmt> ::= while< logical_expr > then \n<stmts> end_while

<empty> ::= “”

<nonlogical_expr> ::= <arithmetic_expr> | <func_call>

<expr> ::= <logical_expr> | <nonlogical_expr>

<and_stmt>::= <logical_expr> and <logical_expr>

<or_stmt>::=<logical_expr> or <logical_expr>

<logical_expr> ::= <nonlogical_expr> <logical_op> <nonlogical_expr>
| <nonlogical_expr> <logical_op> boolean
| boolean

<logical_op>::= < | > | <= | >= | == | !=

<arithmetic_expr> ::= < term > + < arithmetic_expr >
| < term > - < arithmetic_expr >
| <term>

<term> ::= <factor> * <term>
| <factor> / <term>
| <factor>

<factor> ::= <ident> ** <factor>
| <num_ident>

<ident> ::= <low_letter><string>|<empty>

<string>::=<char> | <char><string>

<char> ::= <letter> | <symbol> | <digit>

<letter> ::= <low_letter> | <capital_letter>

<low_letter> ::= a|b|…|z

<capital_letter> ::= A|B|…|Z

<symbol> ::= ‘_’

<var_declaration>::= <var_type> <ident>

<digit> ::= 0|1|2|…|9

<integer> ::= <digit>|<digit><integer>

<method_declaration> ::= <return_type> <ident>(<parameter_list>)
begin \n <stmts><return_stmt> end

<return_type> ::= <var_type> | void

<return_stmt> ::= return <expr> | return

<parameter>::=<var_type> <ident>

<parameter_list>::= <parameter>|<parameter>,<parameter_list>

<var_type>::=num | string | boolean

<func_call>::=<ident> (<args>)\n

<args>::=<all_ident> | <all_ident>,<args>

<all_ident> ::= <ident> | <var_type>

<string_ident>::= <ident> | string

<num_ident> ::= <ident> | num

<turn_func> ::= turn(<num_ident>)

<readData_func> ::= readData (<num_ident>)

<sendData_func> ::= sendData(<num_ident>,<string_ident>)

<recieveData_func> ::= receiveData(<num_ident>)

<comment> ::= #<string>#

EXPLANATIONS FOR BNF DESCRIPTION

1) Program begins with reserved word "**begin"** and ends with "**end"** and a new line, between these words there will be statements to execute and ignored comments or blank lines.
<program> : non-terminal, to provide main program flow

2) <stmts>: non-terminal. Statements can be a one or more statement ends with new line.

3) <stmt>: non-terminal. A statement can be either "matched" which includes a matched if statement and all non-if statements or it can be "unmatched" which includes unmatched if statements.

4) <non_if_stmt>: non-terminal. Non-if statements contain loops, logical statements, method declarations, variable declarations, assignment statements, primitive functions, input/output statements, comments or empty statements(does nothing).

5) <while_stmt>: non-terminal**.** While statement is a loop statement that is used as below:

```
while aVariable > 0 then
        #statements come here#
end_while
```

6) <logical_stmt>: non-terminal**.** Logical statements are **and** and **or** statements.

7) <method_declaration>: non-terminal**.**
<return_type>,<func_name>, <parameter_list>, <stmts>, <return_stmt> all of them non-terminals to define a method. Methods are declared as
<return_type> <func_name>(<parameter_list>) begin\n <stmts><return_stmt> end

e.g

```
num readData ( num dimension ) begin

        #code to be executed#
        return num
end
```

return type: num
function name: readData
parameter list: num dimension
statement: code-to-be-executed
return statement: returns some num value

8) <primitive_func> : non-terminal. Primitive functions are the functions which are mentioned in the assignment: **move**, **grab**, **release**, **turn**, **readData**, **sendData**,

receiveData. **move, grab** and **release** are terminals defined in lex, <turn_func><readData_func><sendData_func><receiveData_func> are non-terminals.

9) <io_stmt> : non-terminal for input output statements

10) <print_stmt>: non-terminal to output to the user, to execute the **print** function. The function uses special **print** word takes a variable in between parenthesis to print the value of it.

11) <scan_stmt>: non-terminal to take an input from the user, to execute the **scan** function. The function uses special **scan** word that takes a variable in between parenthesis to put the input.

12) <turn_func> :non-terminal. To define the turn function.Turn function to make the robot turn rigtht wrt the given angle. **turn** is a terminal defined in lex, <num_ident> is a non-terminal.

e.g

```
boolean right_path
if  right_path == true then
        turn ( 90 )
end_if
```

13) <readData_func> :non-terminal. To define the readData function. ReadData function to read data from a given sensor ID. **readData** is terminal defined in lex, <num_ident> is a non-terminal.

```
readData( 123456789 )
```

14) <sendData_func> :non-terminal. To define the send data function that sends data to a sensor which is specified with a sensor ID. **sendData** is terminal defined in lex, <num_ident> and <string_ident> are non-terminals.

```
sendData( 123, “Hey” )
```

15) <receiveData_func> :non-terminal. To define the receiveData function that receives data from the source. **receiveData** is terminal defined in lex, <num_ident> is a non-terminal.

```
receiveData(<num_ident>)
```

16) <assignment_stmt> :non-terminal. Assignment statement can be declared as either identifier (<ident>: non-terminal) = non-logical expression(<nonlogical_expr> :non-terminal) or identifier = assignment statement.

17) <unmatched>: non-terminal to present a selection statement with **if**’s more than **else**’s.

18) <matched> : non-terminal to present a selection statement with equal number of if and else statements. It also includes all non-if statements.

19) <and_stmt> : non-terminal to compare two boolean expressions. If both of them are true the statement is **true**, otherwise **false**.

20) <or_stmt> : non-terminal to compare two boolean expressions. If at least one of them is true the statement is **true**, if both of them are false, statement is **false**.

21) <logical_op>: non terminal that contains terminal logical operators below:

x **<** y is true if y is bigger than x
x **>** y is true if x is bigger than y
x **<=** y is true if x is smaller than or equal to y
x **>=** y is true if x is bigger than or equal to y
x **==** y is true if x equals to y
x **!=** y is true if x is not equal to y

22) <arithmetic_expr> : non-terminal to define arithmetic operations; in definition of the expression this non terminal is called for recursion. Arithmetic expressions can be either addition of two arithmetic operation, subtraction of two arithmetic operation or a term.

23) <term> : non-terminal to define a term, in definition it is also used for recursion**.**Term can be multiplication of a term and a factor, division of a term to a factor or a factor.

•The reason that we divide arithmetic operation (for addition-subtraction and multiplication-division) to avoid from the **ambiguity**.

• **Precedence:**
exponents of exponential numbers >> term (multiplication and division) >> arithmetic expression (addition and subtraction)

24) <factor>: non-terminal to define factor. Factor to declare exponential numbers.
a**b is representing a^b .

25) <ident> : non-terminal. Variables' identifiers should begin with a lower-case letter however, rest can be integer, symbol or a capital letter
e.g: vAri4BLe or variaBl_e_56 or simply: aVariable

26) <var_type>: non-terminal to specify variable types. Variable types are: num, string and boolean. **num, string, boolean** are reserved words.

28) <return_type> : non-terminal to specify return type. Return type should be a variable or void.

29) <comment> : non-terminal. Comments should begin and end with **#** symbol, compiler will not execute these lines.

30) <var_declaration>: non-terminal. Starts with a variable type and followed by an identifier.

31) <return_stmt>: non-terminal. Specifies the return statement of a function, it can returns an expression or simply returns empty all by using **return** terminal.

32) <parameter_list>: non-terminal. Composed of one or more parameters.

33) <parameter>: non-terminal. Parameters start with a variable type and followed by an identifier.

34) <digit> : non-terminal to represent digits (0,1,2,3,4,5,6,7,8,9)

35) <integer>: non-terminal to represent numbers.

36) <letter> :non-terminal for <low_letter>(non terminal a-to-z) and <capital_letter>(non terminal A-to-Z)

37) <low_letter> :non terminal a-to-z

38) <capital_letter> :non terminal A-to-Z

39) <expr>: non-terminal. Can be either logical expression or nonlogical expression.

40) <logical_expr>: non-terminal. We use this expression to get a boolean value by comparing nonlogical expressions and booleans with using logical operators.

41) <nonlogical_expr>: non-terminal. Can be either an arithmetic expression or a function call.

42) <func_call>: non-terminal that represents a function call. An example function call can be like:

myFunc(7, “doSmt”)

Here, myFunc is an identifier, parenthesis are terminals, 7,”doSmt” are arguments.

43) <args>: non-terminal. Can be one or more variable type or identifier.

44) <all_ident>: non-terminal. Can be any variable type or an identifier.

45) <num_ident>: non-terminal. Can be either **num** variable type or an identifier.

46) <string_ident>: non-terminal. Can be either **string** variable type or an identifier.

47) <string>: non-terminal. Composed of one or more char.

48) <char>: non-terminal. Composed of letters symbol and digits.

49) <empty>: non-terminal. Just a representation for blanks.

Terminals from lex implementation (seperated by whitespace)**: ( ) , begin end num string boolean true false if else then while end_while return void # end_if receiveData sendData readData turn move grab release print scan and or**
(Explanations given above inside explanations of non-terminals.)