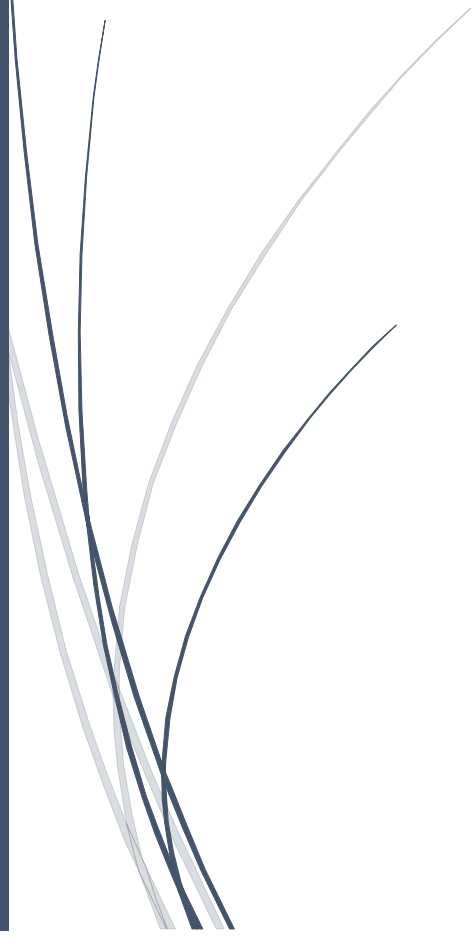
30.10.2018

RoboC

Programming Language

YUSUF SAMSUM	21501651
SAMET DEMIR	21502139
FIRAT YILDIZ	21502717



CS315 Group 1

BILKENT UNIVERSITY

BNF Description of RoboC

BNF description also can be found in BNFDescription.txt file.

<program> ::= <main_func>
| <main_func> <func_list>

<func_list> ::= <function>
| <function> <func_list>

<main_func> ::= void main () <block_stmt>

<function> ::= <return_types> <identifier> (<param_list>) { <stmt_list> <return_stmt> }

<param_list> ::= /* empty */
| <parameter> , <param_list>

<parameter> ::= <pri_Vars> <identifier>

<block_stmt> ::= { <stmt_list>? }

<stmt_list> ::= <stmt>
| <stmt> <stmt_list>

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

<basic_stmt> ::= <assign_stmt>

| <loop_stmts>

| <return_stmt>

| <declera_stmt>

| <empty_stmt>

| <block_stmt>

<empty_stmt> ::= ;

<matched> ::= if <logical_expr> <matched> other <matched>

| <basic_stmt>

<unmatched> ::= if <logical_expr> <stmt>

| if <logical_expr> <matched> other <unmatched>

<loop_stmts> ::= <while_stmt>

| <for_stmt>

| <do_while_stmt>

<while_stmt> ::= while <logical_expr> <statement>

<for_stmt> ::= for <decleration_list> ; <logical_expr> ; <for_assign_stmt_list> ; <statement>

<for_assign_stmt_list> ::= <for_assign_stmt> , <for_assign_stmt_list>

<for_assign_stmt> ::= <identifier> = <operations>

| <identifier> ++

| <identifier> --

| /* empty */

<do_while_stmt> ::= do <block_stmt> while <logical_expr> ;

<return_stmt> ::= return ;

| return <logical_expr> ;

| return <identifier> ;

| return <int> ;

<logical_expr> ::= true | false

| <logical_expr> <relations> <operations>

| <operations> <relations> <logical_expr>

| <logical_expr> <relations> <logical_expr>

<relations> ::= ==

| <

| >

| !=

| <=

| >=

| &

| |

<id_list> ::= <identifier>

| <id_list> , <identifier>

<declera_stmt>::= <pri_Vars> <identifier> = <operations> ;

| <pri_Vars> <identifier> ;

<assign_stmt> ::= <identifier> = <identifier> ;

| <identifier> = <operations> ;

<operations> ::= <term>

| <operations> + <term>

| <operations> - <term>

<term> ::= <primary>

| <term> * <primary>

| <term> / <primary>

| <term> % <primary>

<primary> ::= <constants>

| <identifier>

| <primitiveFuncCall>

| <function_call>

| (<operations>)

<function_call> ::= <identifier> (<func_call_para_list>?) ;

<func_call_para_list>::= <primary> | <primary> , <func_call_para_list>

<constants> ::= <string>

| <number>

<primitiveFuncCall> ::= move (<primary> , <primary>) ;

| grab (<primary>) ;

| turn (<primary> , <primary>) ;

| sendData (<primary> , <primary>) ;

| rcvData (<primary> , <primary>) ;

| releaseData (<primary> , <primary>) ;

| readData (<primary> , <primary>) ;

| print (<string>) ;

| println (<string>) ;

<return_types> ::= <pri_Vars> | void

<pri_Vars> ::= bool | int | char | string | double | float

The following definitions are hardcoded into the lexical scanner:

<string> ::= " <text> "

<text> ::= <str_character> | <text> <str_character>

<identifier> ::= <letter>

| <identifier> <letter>

| <identifier> <decimal digit>

| <identifier>

<str_character> ::= <tabulation> | <space> | '!' | '#' | ... | '[' | ']' | ... | 0xFF

<space> ::= 0x20

<tabulation> ::= 0x09

<int> ::= <digit> | <digit> <int>

<letter> ::= A|B|C ... |Z|a|b|c ... |z

<digit> ::= 0|1| ... |9

Explanations and Descriptions

In the first part of the project, we are supposed to implement the lexical analysis part of our language. We designed a language considering of language rules and

Chomsky Diagram. Our vision is that a good language should be readable, writable and reliable. To increase the readability, we add “;" at the end of each command and some obligations to use space characters, end line characters. For writability, we remove open and close parentheses (“(“ and “)”) after tokens (as for, while, if) which are generally limited expressions in common languages. Finally, we use mathematical rules to hinder contradictions. For example, parentheses have best priority after multiplication and division have priority.

In the language, line command is “%%“ and multiline commands will be “*/ <program>*/“ like in JAVA language. There will be no parenthesis after tokens like “ if x<5 “ however there will be “{“ “}” in order to have multilines of code in the program. What’s more, we have primitive functions for robot in order to grab something, move turn and so on. Move function has no parameter because robot just moves 1 mm when the method has called. Move will be called like “move(int mm, int side) ;”, which gets the millimeter that will moved and either robot will go back or forward, grab will be called “grab() ;”, receive data as”read(sensorID x) ;”, it gets the parameter of sensor id. The function “ turn(1,0) ;” means turn right 1 degree to right and “turn (1,1) ;” means turn 1 degree left. We have also send data, receive data to/ from master. Robot will send data to MASTER, any other robot which found by it’s id or get data from a specific actor like MASTER. Method is called like “send(DataType data, int robotID);”. “,” and “(“, “)” used in order to make language more readable.

Assignments, loops, arithmetical operations will be same with the other languages. User will assign values and numbers by using “=" like “x=5”. They are not allowed

to use “=" operator recursively. WHILE, FOR, DO, IF, OTHER are tokens that we used in the language. In the using of FOR, we separated the expressions and operations with “:” like “for int i = 0 : i<5 : i++{}”. Users will not be allowed to write single line inside the “for".

Finally, in the conditional statement, programmer use the token “OTHER" instead of “ELSE”. “OTHER" is totally same with “ELSE”. Programmers may also use “OTHER IF” statement as:

if <expression> { }

other if <expression>{ }

Similarly, we define do-while statement as:

do{<statement-list>}

while i < max

The programmers are not allowed to write single line code inside "if" statement. They can use all the tokens inside by inside recursively too.

Lex Description can be found in LexDescription.txt file.

Code example can be found in ExampleCode.txt file.