RACE

Rapid Advanced Compiler Engine

Team 23

1] Abhishek Kumar (akuma168)

2] Aman Maheshwari (amahes10)

3] Karansher Bhangal (kbhangal)

4] Saheb Johar (ssjohar)

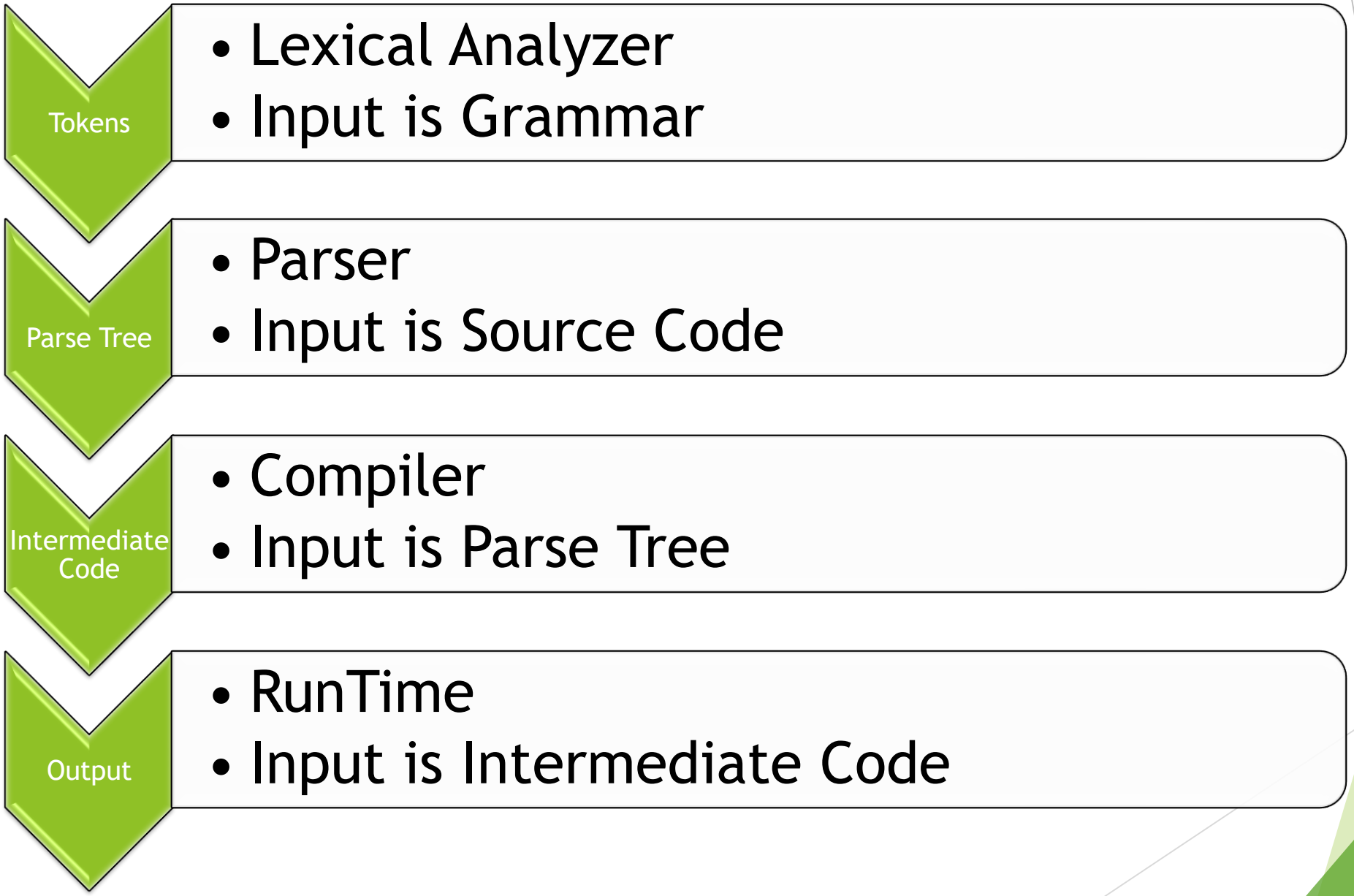
RACE *supports….*

- FLOAT Type
- BOOLEAN Type
- If-else-if loop
- WHILE Loop
- Basic Arithmetic Operators such as +,-,*,/

Features implemented(Extra Credits)

- FOR Loop
- NESTED Loops
- FUNCTION call
- RETURN statement
- Operators such as

- ✓ INCREMENT : '++'
- ✓ DECREMENT : '--'
- ✓ AND : '&&'
- ✓ OR : '||'
- ✓ NOT : '~'
- ✓ LESSER : '<'
- ✓ GREATER : '>'
- ✓ LESS or EQUAL : '<='
- ✓ MORE or EQUAL : '>='
- ✓ Not Equals : '!='
- ✓ Is Equals : '=='

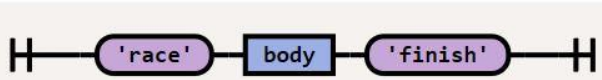

Tokens
• Lexical Analyzer
• Input is Grammar
Parse Tree
• Parser
• Input is Source Code
Intermediate Code
• Compiler
• Input is Parse Tree
Output
• RunTime
• Input is Intermediate Code

RACE Grammar Rules >>>>>

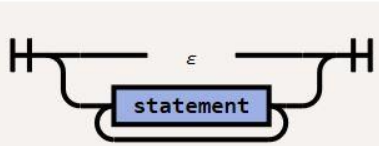
```
grammar Race;


program
: 'race' body 'finish'
;

body
:
| statement+
;

statement
:
(funcStmt|ifExpressn|whileExpressn|forExp
ressn|disp|expression|func|returnStmt)
;
```

Race.g4

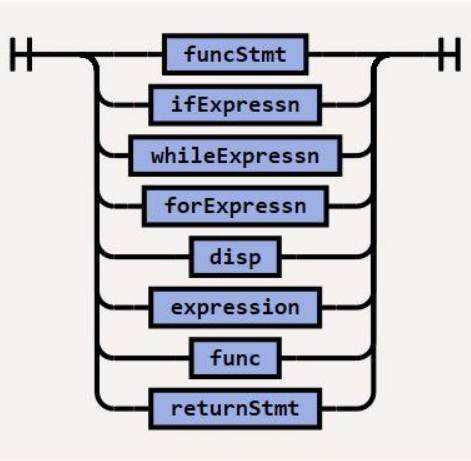
program



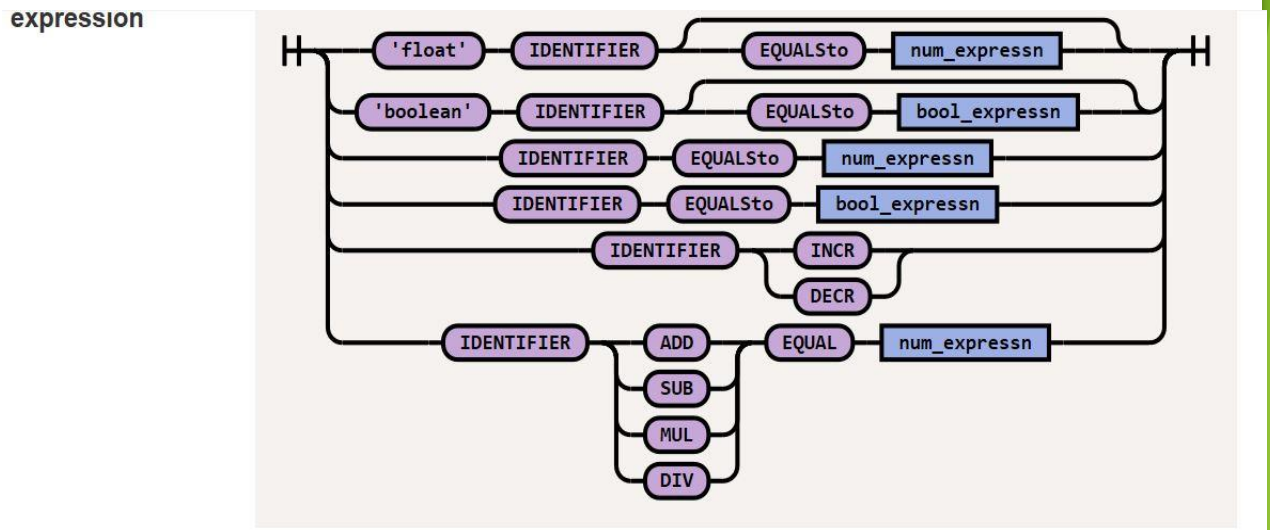
body



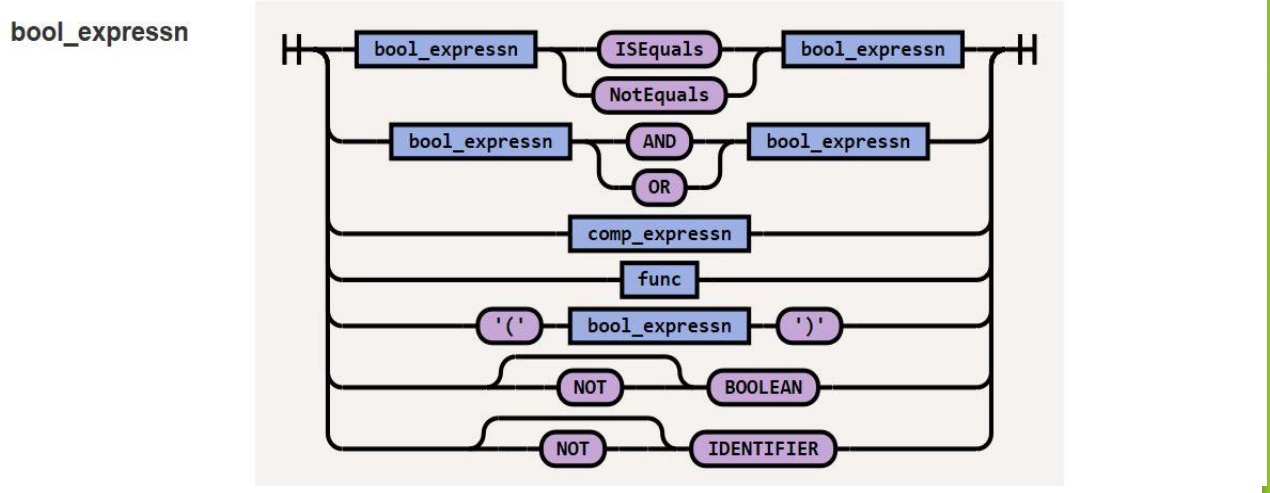
statement



```
expression
: 'float' IDENTIFIER (EQUALSto num_expressn
| 'boolean' IDENTIFIER (EQUALSto bool_expressn)?
| IDENTIFIER EQUALSto num_expressn
| IDENTIFIER EQUALSto bool_expressn
| IDENTIFIER op=(INCR|DECR)
    | IDENTIFIER op=(ADD|SUB|MUL|DIV) EQUAL
num_expressn
    ;
```

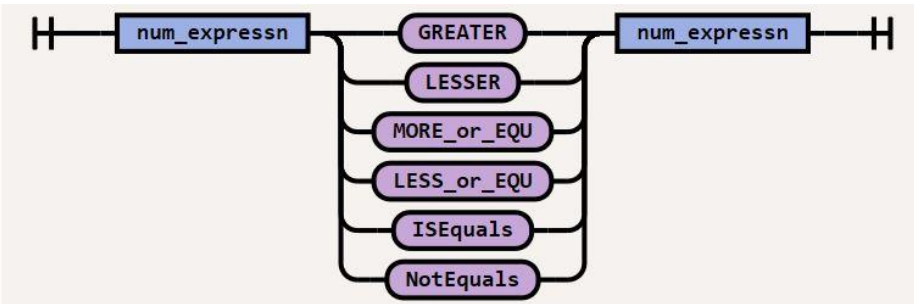



```
bool_expressn
    : bool_expressn op=(ISEquals|NotEquals) bool_expressn
    | bool_expressn op=(AND|OR) bool_expressn
    | comp_expressn
    | func
    | '(' bool_expressn ')'
    | (NOT)? BOOLEAN
    | (NOT)? IDENTIFIER
    ;
```



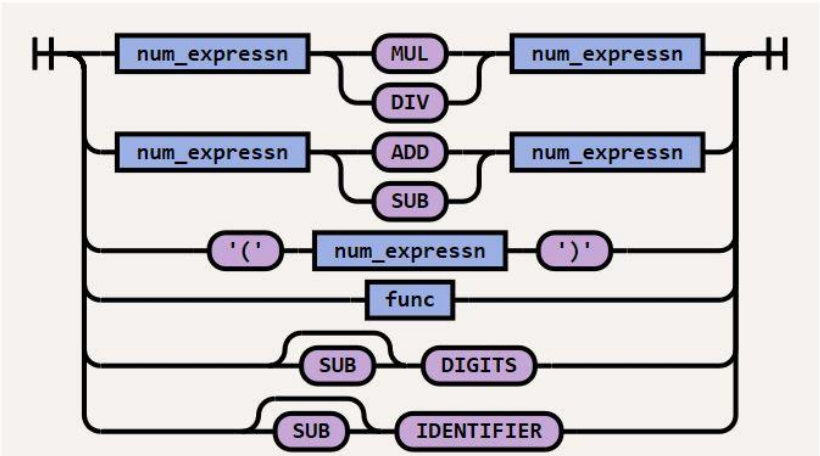
```
comp_expressn
    : num_expressn
op=(GREATER|LESSER|MORE_or_EQU|LESS_or_EQU|ISEquals|
NotEquals) num_expressn
    ;
```

comp_expressn



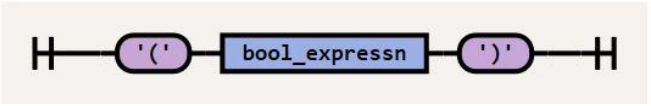

```
num_expressn
    : num_expressn op=(MUL|DIV) num_expressn
    | num_expressn op=(ADD|SUB) num_expressn
    | '(' num_expressn ')'
    | func
    | SUB? DIGITS
    | SUB? IDENTIFIER
    ;
```

num_expressn



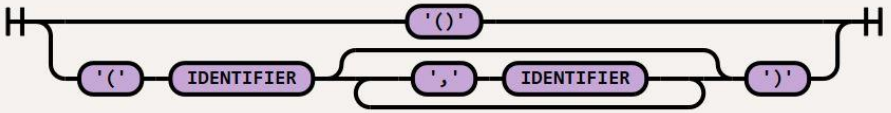

```
cond_expressn
    : '(' bool_expressn ')'
    ;
```

cond_expressn

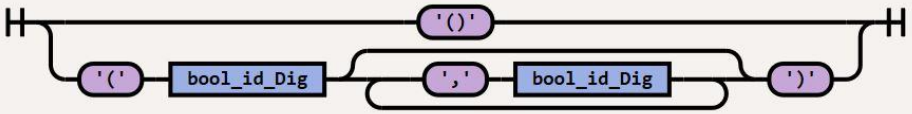


```
inputParams
    : '()'
    | '(' IDENTIFIER (',' IDENTIFIER)* ')'
    ;

inputArgs
    : '()'
    | '(' bool_id_Dig (',' bool_id_Dig)* ')'
    ;

funcStmt
    : 'function' IDENTIFIER inputParams '{' body '}'
    ;

func
    : IDENTIFIER inputArgs
    ;

returnStmt
    : 'return' (bool_id_Dig|num_expressn|bool_expressn|func)
    ;

ifExpressn
    : 'if' cond_expressn '{' body '}' (elseIfExpressn)* (elseExpressn)?
    ;
```

inputParams



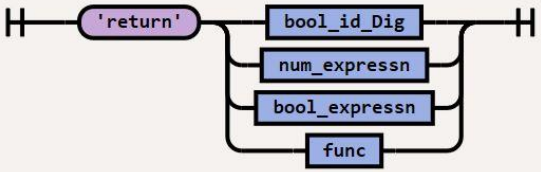
inputArgs



funcStmt

func

returnStmt



ifExpressn

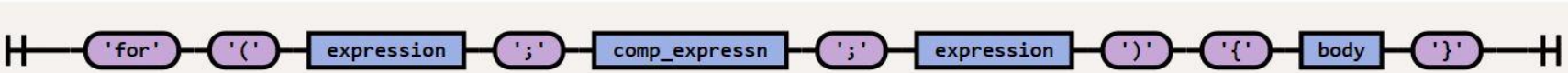
```
elseIfExpressn
    : 'else if' cond_expressn '{' body '}'
    ;

elseExpressn
    : 'else' '{' body '}'
    ;

whileExpressn
    : 'while' cond_expressn '{' body '}'
    ;

forExpressn
    : 'for' '(' expression ';' comp_expressn ';' expression ')' '{' body '}'
    ;

disp
    : 'WRITE' '(' (DIGITS|BOOLEAN|IDENTIFIER|num_expressn|bool_expressn|func) ')'
    ;
```

elseIfExpressn

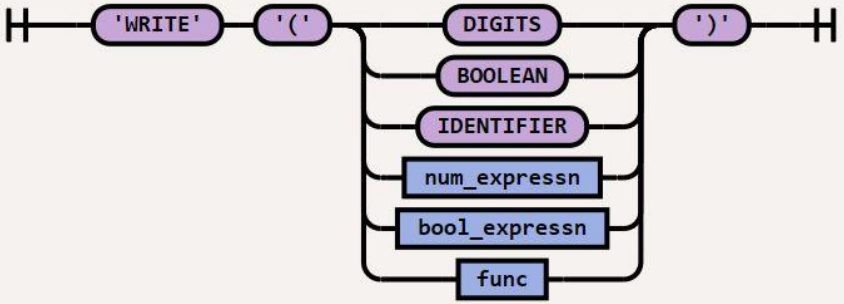
elseExpressn

whileExpressn

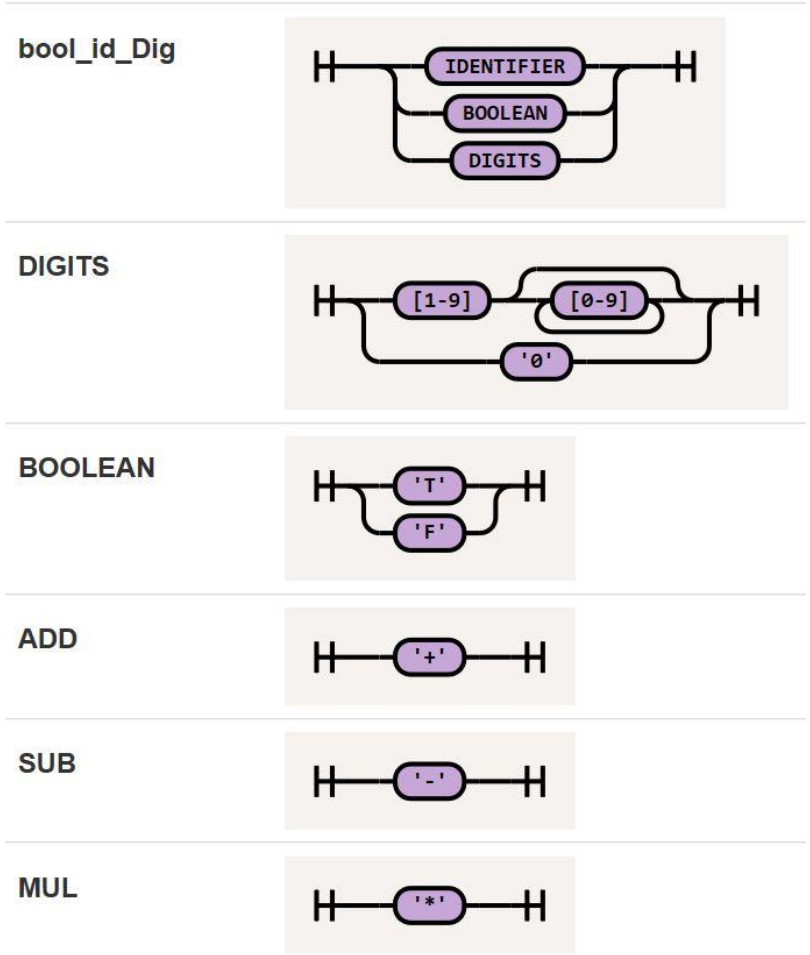
forExpressn



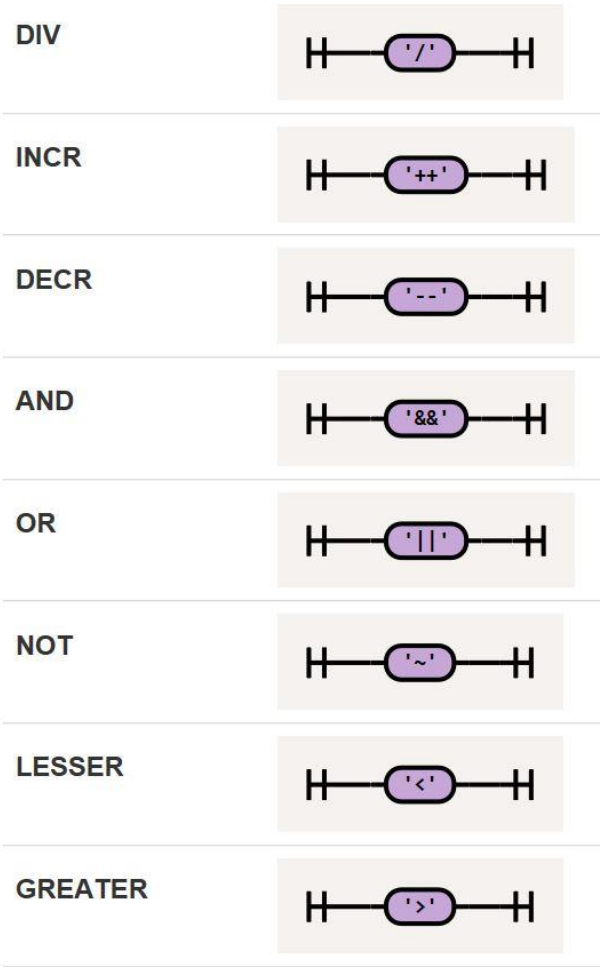
disp



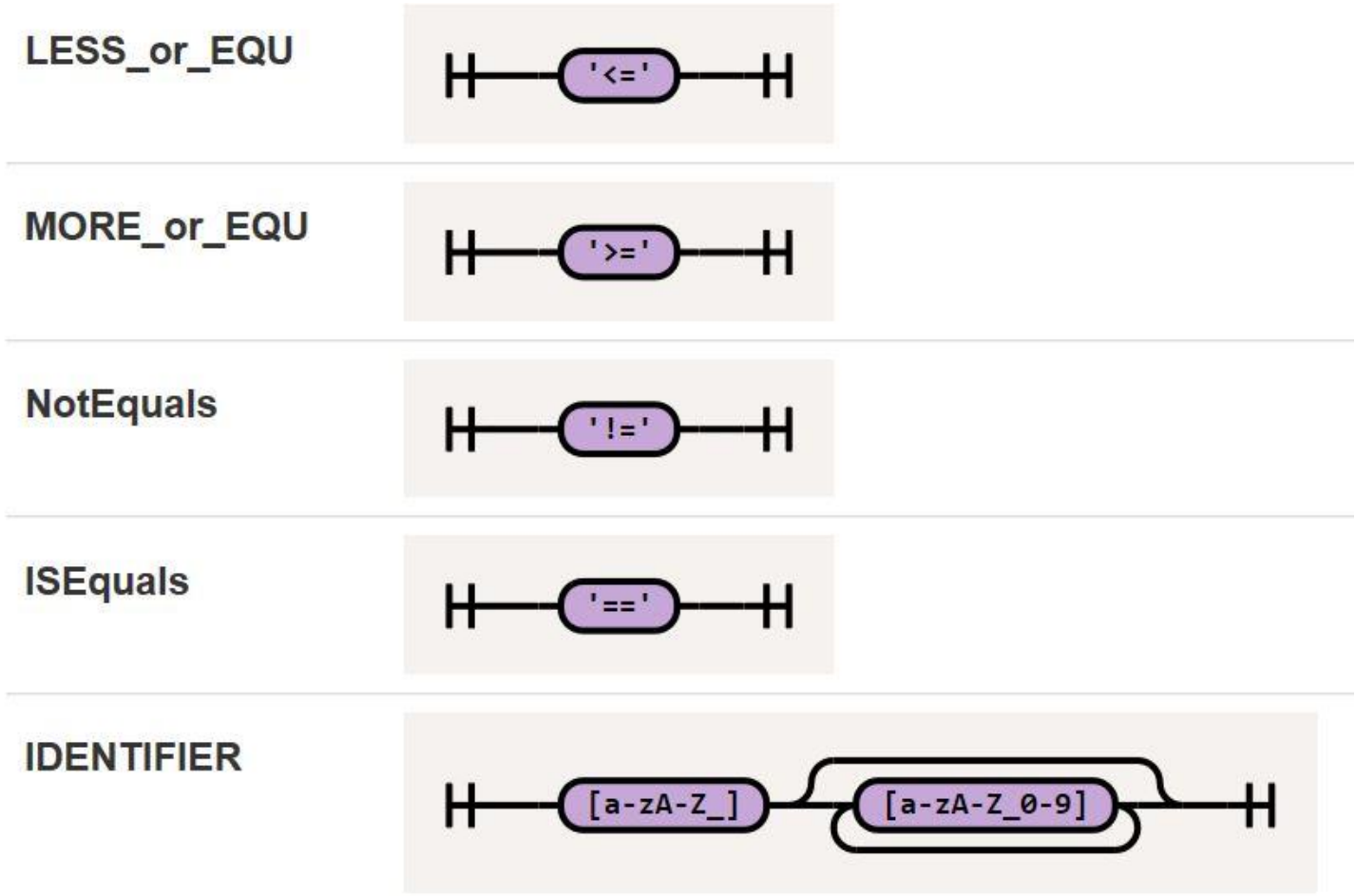
```
bool_id_Dig
    : (IDENTIFIER|BOOLEAN|DIGITS)
    ;

// accepts numbers from 0-9 and more than that.
DIGITS
: [1-9] [0-9]*
| '0'
;

// Takes Boolean value as True or false.
BOOLEAN
: 'T'
| 'F'
;

ADD         : '+';
SUB         : '-';
MUL         : '*';
```



```
DIV     : '/';
INCR    : '++';
DECR    : '--';
AND     : '&&';
OR      : '||';

NOT     : '~';
LESSER  : '<';
GREATER : '>';
```



```
LESS_or_EQU    : '<=';

MORE_or_EQU    : '>=';

NotEquals      : '!=';

ISEquals       : '==';

// Accepts lower case and upper case letters.
IDENTIFIER
: [a-zA-Z_] [a-zA-Z_0-9]*
;
```

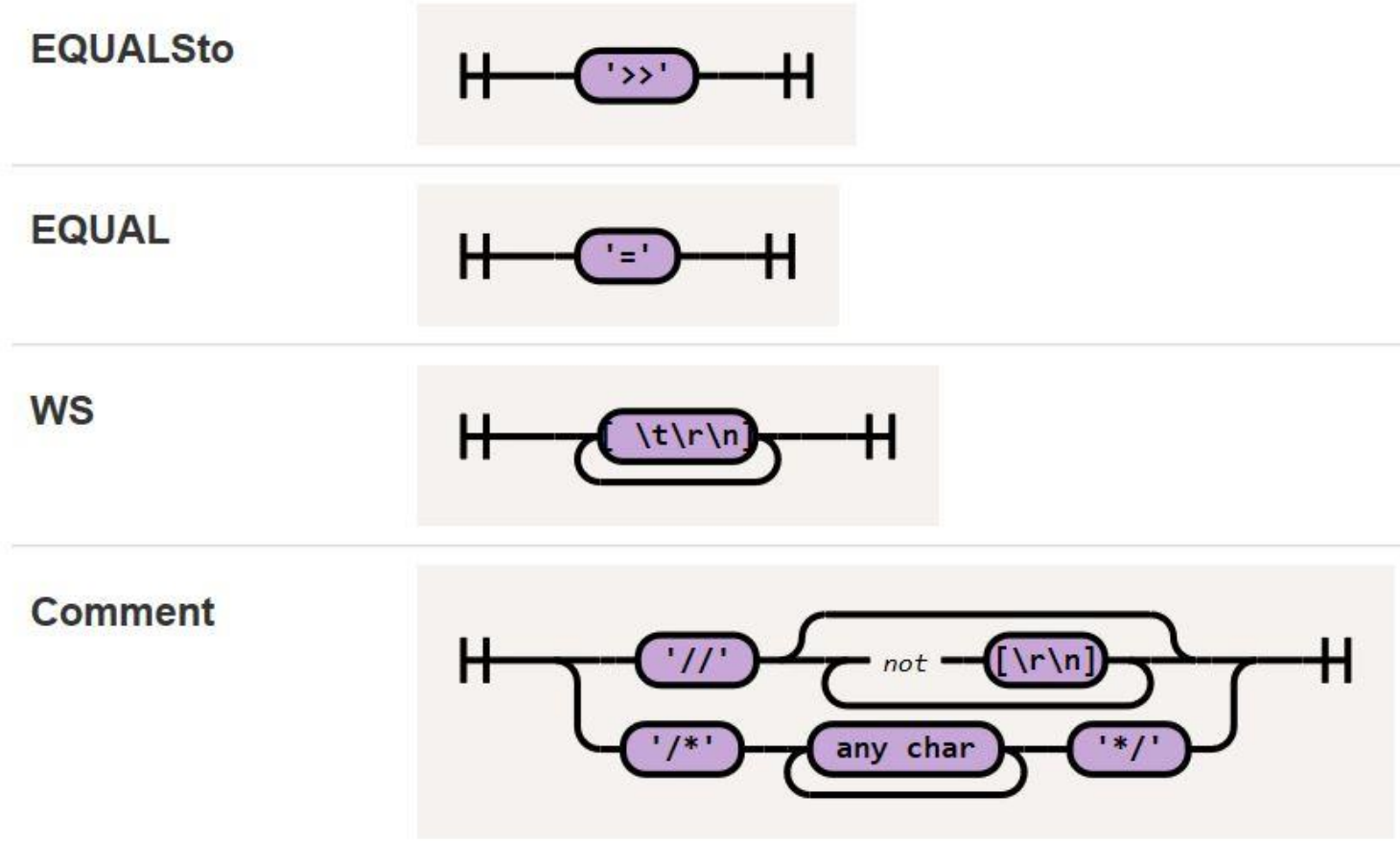


EQUALSto : '>>';

EQUAL : '=';

WS : [\t\r\n]+ -> skip; // skip spaces, tabs, newlines

Comment : ('//' ~[\r\n]* | '/*' .*? '*/') -> skip;



INTERMEDIATE CODE DEFINITION

- FOR_LOOP_START = "FOR_LOOP_START";
- FOR_LOOP_END = "FOR_LOOP_END";
- COMPARISION_START = "COMPARISION_START";
- COMPARISION_END = "COMPARISION_END";

- ACC_REGISTER = "REG";
- C_REGISTER = "C";
- B_REGISTER = "B";

- STORE_INSTRUCTION = "SAVE";
- READ_INSTRUCTION = "READ";
- WRITE_INSTRUCTION = "DISPLAY";

- ADD_INSTRUCTION = "ADD";
- SUBTRACT_INSTRUCTION = "SUB";
- MULTIPLY_INSTRUCTION = "MUL";
- DIVIDE_INSTRUCTION = "DIV";

- UNARY_MINUS = "UNARY";

- GREATER_THAN = "GREATER";
- GREATER_THAN_EQUAL = "GREATER_OR_EQUAL";
- LESS_THAN = "LESSER";
- LESS_THAN_EQUAL= "LESSER_OR_EQUAL";
- EQUAL_EQUAL = "EQUAL_TO";
- NOT_EQUAL = "NOT_EQUAL";

- OR = "OR";
- AND = "AND";
- NOT = "NOT";

- BOOLEAN_EQUAL_EQUAL = "BOOL_ISEQUALS";
- BOOLEAN_NOT_EQUAL = "BOOL_IS_NOT_EQUALS";

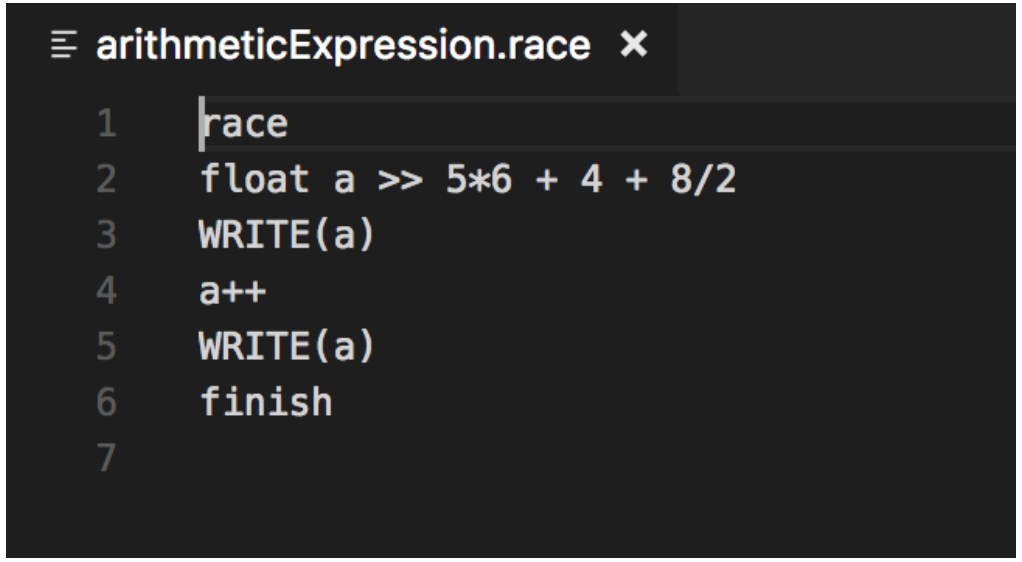
- IF_ELSE_BLOCK_START = "IF_ELSE_BLOCK_START";
- IF_ELSE_BLOCK_END = "IF_ELSE_BLOCK_END";
- IF_BLOCK_START = "IF_BLOCK_START";
- IF_BLOCK_END = "IF_BLOCK_END";
- ELSE_IF_BLOCK_START = "ELSE_IF_BLOCK_START";
- ELSE_IF_BLOCK_END = "ELSE_IF_BLOCK_END";
- ELSE_BLOCK_START = "ELSE_BLOCK_START";
- ELSE_BLOCK_END = "ELSE_BLOCK_END";

- CONDITION_START = "CONDITION_START";
- CONDITION_END = "CONDITION_END";

- WHILE_BLOCK_START = "WHILE_BLOCK_START";
- WHILE_BLOCK_END = "WHILE_BLOCK_END";

- FUNCTION_START = "FUNCTION_START";
- FUNCTION_END = "FUNCTION_END";
- FUNCTION_NAME = "FUNCTION_NAME";
- FUNCTION_PARAMS = "FUNCTION_PARAMS";
- FUNCTION_RETURN_START = "FUNCTION_RETURN_START";
- FUNCTION_RETURN_END = "FUNCTION_RETURN_END";
- FUNCTION_CALL = "FUNCTION_CALL";

arithmeticExpression.race



```
1   race
2   float a >> 5*6 + 4 + 8/2
3   WRITE(a)
4   a++
5   WRITE(a)
6   finish
7
```

Output of the program

38.0
39.0

INTERMEDIATE CODE

SAVE REG 5.0
SAVE A REG
SAVE REG 6.0
SAVE B REG
MUL REG A B
SAVE C REG
SAVE REG 4.0
SAVE B REG
ADD REG C B
SAVE C REG
SAVE REG 8.0
SAVE A REG
SAVE REG 2.0
SAVE B REG
DIV REG A B
SAVE B REG
ADD REG C B
SAVE a REG
DISPLAY a
SAVE REG 1
ADD a a REG
DISPLAY a

fibonacci.race

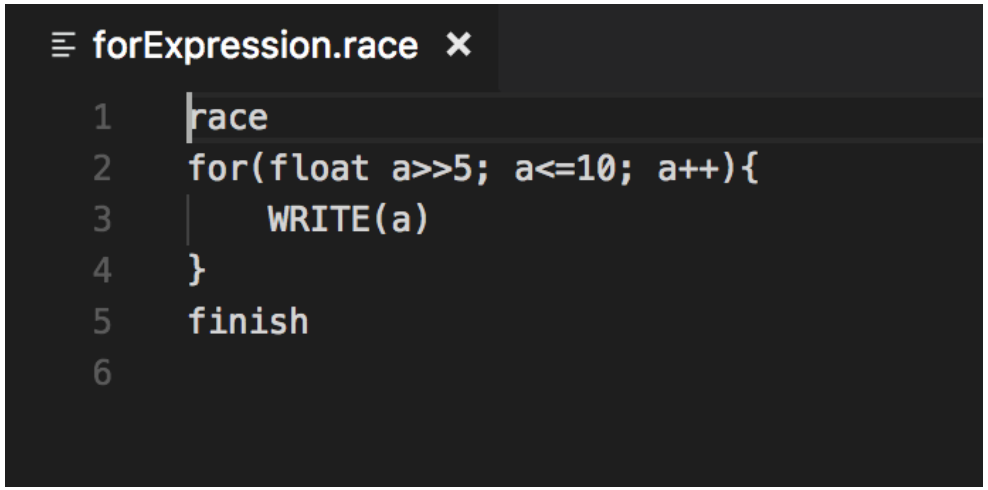
```
race
function fibonacci(x) {
    if (x == 0) {
        return 0
    }
    if (x == 1) {
        return 1
    }
    float x1 >> x - 1
    float x2 >> x - 2
    return fibonacci(x1) + fibonacci(x2)
}
WRITE(fibonacci(7))
finish
```

Output of the program
13.0

INTERMEDIATE CODE GENERATION

```
FUNCTION_START
FUNCTION_NAME fibonacci
FUNCTION_PARAMS x
IF_ELSE_BLOCK_START
IF_BLOCK_START
CONDITION_START
SAVE REG x
SAVE A REG
SAVE REG 0.0
SAVE B REG
EQUAL_TO REG A B
CONDITION_END
FUNCTION_RETURN_START
SAVE REG 0
FUNCTION_RETURN_END
IF_BLOCK_END
IF_ELSE_BLOCK_END
IF_ELSE_BLOCK_START
IF_BLOCK_START
CONDITION_START
SAVE REG x
SAVE A REG
SAVE REG 1.0
SAVE B REG
EQUAL_TO REG A B
CONDITION_END
FUNCTION_RETURN_START
SAVE REG 1
FUNCTION_RETURN_END
IF_BLOCK_END
IF_ELSE_BLOCK_END
SAVE REG x
SAVE C REG
SAVE REG 1.0
SAVE B REG
SUB REG C B
SAVE x1 REG
SAVE REG x
SAVE C REG
SAVE REG 2.0
SAVE B REG
SUB REG C B
SAVE x2 REG
FUNCTION_RETURN_START
FUNCTION_CALL fibonacci x1
SAVE C REG
FUNCTION_CALL fibonacci x2
SAVE B REG
ADD REG C B
FUNCTION_RETURN_END
FUNCTION_END
FUNCTION_CALL fibonacci 7
DISPLAY REG
```

forExpression.race



```
1   race
2   for(float a>>5; a<=10; a++){
3       WRITE(a)
4   }
5   finish
6
```

Output of the program

5.0
6.0
7.0
8.0
9.0
10.0

INTERMEDIATE CODE GENERATION

SAVE REG 5.0
SAVE a REG
FOR_LOOP_START
COMPARISION_START
SAVE REG a
SAVE A REG
SAVE REG 10.0
SAVE B REG
LESSER_OR_EQUAL REG A B
COMPARISION_END
DISPLAY a
SAVE REG 1
ADD a a REG
FOR_LOOP_END

nested_if_else.race

```
race
float a >> 5
if (a < 5){
    a >> 4
}else if (a ==5){
    a >> a + 1
}else {
    a++
}
WRITE(a)
finish
```

Output of the program

6.0

INTERMEDIATE CODE

```
SAVE REG 5.0
SAVE a REG
IF_ELSE_BLOCK_START
IF_BLOCK_START
CONDITION_START
SAVE REG a
SAVE A REG
SAVE REG 5.0
SAVE B REG
LESSER REG A B
CONDITION_END
SAVE REG 4.0
SAVE a REG
IF_BLOCK_END
ELSE_IF_BLOCK_START
CONDITION_START
SAVE REG a
SAVE A REG
SAVE REG 5.0
SAVE B REG
EQUAL_TO REG A B
CONDITION_END
SAVE REG a
SAVE C REG
SAVE REG 1.0
SAVE B REG
ADD REG C B
SAVE a REG
ELSE_IF_BLOCK_END
ELSE_BLOCK_START
SAVE REG 1
ADD a a REG
ELSE_BLOCK_END
IF_ELSE_BLOCK_END
DISPLAY a
```

nested_loop.race

```
1   race
2   float a >> 1
3   for(float index >> 0; index < 5; index++) {
4       a >> 0
5       while ( a < 10 ) {
6           WRITE(a)
7           a >> a + 1
8       }
9   }
10  finish
11
```

Output

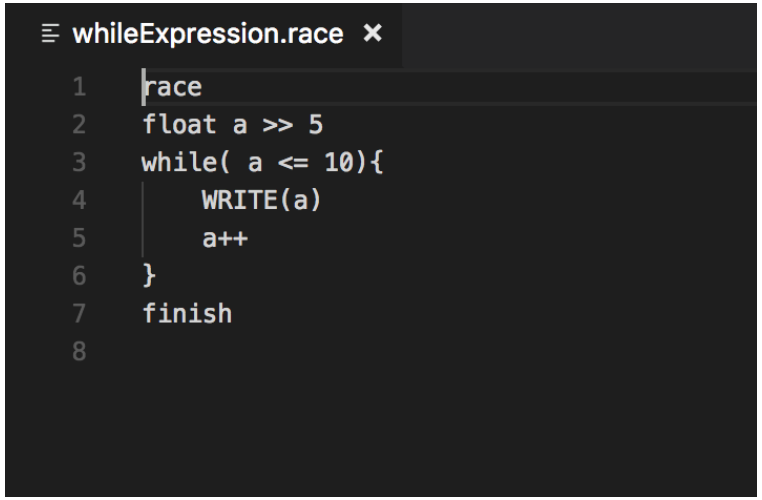
0.0			0.0	0.0
1.0	1.0	0.0	1.0	1.0
2.0	2.0	1.0	2.0	2.0
3.0	3.0	2.0	3.0	3.0
4.0	4.0	3.0	4.0	4.0
5.0	5.0	4.0	5.0	5.0
6.0	6.0	5.0	6.0	6.0
7.0	7.0	6.0	7.0	7.0
8.0	8.0	7.0	8.0	8.0
9.0	9.0	8.0	9.0	9.0
0.0		9.0		

INTERMEDIATE CODE

SAVE REG 1.0
SAVE a REG
SAVE REG 0.0
SAVE index REG
FOR_LOOP_START
COMPARISION_START
SAVE REG index
SAVE A REG
SAVE REG 5.0
SAVE B REG
LESSER REG A B
COMPARISION_END
SAVE REG 0.0
SAVE a REG
WHILE_BLOCK_START

CONDITION_START
SAVE REG a
SAVE A REG
SAVE REG 10.0
SAVE B REG
LESSER REG A B
CONDITION_END
DISPLAY a
SAVE REG a
SAVE C REG
SAVE REG 1.0
SAVE B REG
ADD REG C B
SAVE a REG
WHILE_BLOCK_END
SAVE REG 1
ADD index index REG
FOR_LOOP_END

whileExpression.race



```
race
float a >> 5
while( a <= 10){
    WRITE(a)
    a++
}
finish
```

Output of the program

5.0
6.0
7.0
8.0
9.0
10.0

INTERMEDIATE CODE

SAVE REG 5.0
SAVE a REG
WHILE_BLOCK_START
CONDITION_START
SAVE REG a
SAVE A REG
SAVE REG 10.0
SAVE B REG
LESSER_OR_EQUAL REG A B
CONDITION_END
DISPLAY a
SAVE REG 1
ADD a a REG
WHILE_BLOCK_END

Future Work

- New Datatype handling can be added in the grammar in order to have more flexibility in the language such as character, Strings etc.
- Complex datatypes such as Array, Lists, Sets can be added for higher order logic implementation.
- Object oriented programming can be added for to incorporate concepts such as inheritance, polymorphism etc.