BILKENT UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CS 315
PROJECT 2
REPORT

Project Group No: 04

Mehmet Ali Altuntaş - 21401004

Damla Eda Bıçakcı - 21402130

İrem Yurdakul - 21400299

Revised Language Design in BNF form:

Our language requires, the code must be written between “BEG” and “ENDBEG” code block, in other words this is our main function. We added our code “SPACE” feature to increase writability and readability, so user can define functions and statements without thinking about the space characters among variables or assignments. We also added array data type into our language.

Changes in BNF:

- We changed name of program *begin* to *BEG and end to BEGEND* because while using begin and end words we got an error.
- We have added left and right square and curly brackets as terminals.
- We have added a new terminal as ADD for array describing. It is now working as the following:
type arrName arr[size] = {3,4}; (values for array elements)
- We have added a new line token for assigning expressions between BEG and ENDBEG.
- We can also declare our statements with semicolon at the end of it or not to increase flexibility.
- We have print “LANGUAGE IS CORRECT” if the written code is proper for language design. If not it gives and syntax error output with the line number.

Revised BNF:

```
<prog> :BEG NL <stat_lis>t NL ENDBEG
<stat_list>: <stmt> | <stat_list> NL stmt
;
<stmt >:<if_stmt>
|<stmt_e>
;
<stmt_e> : |<assign> SEMICOLON| <loop> | <fnc> | <in_out_stmt> SEMICOLON| <expr>
SEMICOLON
;
<if_stmt >: <matched> | <unmatched>
;
<matched>: IF LEFT_PARA <bool_expr> RIGHT_PARA THEN <matched> ELSE <matched>
| <stmt_e>
| L_CURL SPACE <matched> SPACE R_CURL
;
<unmatched>: IF LEFT_PARA <bool_expr> RIGHT_PARA THEN L_CURL <stmt> R_CURL
| IF LEFT_PARA <expr> RIGHT_PARA THEN <matched> ELSE <unmatched>
;

<expr>: <bool_expr> | <math_expr>
;
<bool_expr> : <log_expr> | <equ_expr>
;

<log_expr>: <log_expr> SPACE AND SPACE <log_term>|<log_expr> SPACE AND
<log_term>|<log_expr> AND <log_term>|<log_expr> AND SPACE <log_term>
| <log_expr> SPACE OR SPACE <log_term>|<log_expr> SPACE OR <log_term>|<log_expr>
OR <log_term>|<log_expr> OR SPACE <log_term>
| <log_expr> SPACE IMPLICATION SPACE <log_term>|<log_expr> SPACE IMPLICATION
```

```
<log_term>|<log_expr> IMPLICATION <log_term>|<log_expr> IMPLICATION SPACE
<log_term>
| <log_expr> SPACE EQUIVALENCE SPACE <log_term>|<log_expr> SPACE
EQUIVALENCE <log_term>|<log_expr> EQUIVALENCE <log_term>|<log_expr>
EQUIVALENCE SPACE <log_term>
|< log_expr> SPACE IS_EQUAL SPACE <log_term>|<log_expr> SPACE IS_EQUAL
<log_term>| <log_expr> IS_EQUAL SPACE <log_term> |< log_expr> IS_EQUAL <log_term>
|< log_term>
| NEGATION <log_term>
;
<log_term >: LEFT_PARA <log_expr>  RIGHT_PARA | ID| LEFT_PARA SPACE <log_expr>
RIGHT_PARA| LEFT_PARA <log_expr> SPACE RIGHT_PARA|LEFT_PARA SPACE
<log_expr> SPACE RIGHT_PARA
| ID
;

<equ_expr>: ID SPACE GREATER_OP SPACE ID| ID SPACE SMALLER_OP SPACE ID
|NUMBER SPACE GREATER_OP SPACE NUMBER|NUMBER SPACE SMALLER_OP
SPACE NUMBER
|NUMBER SPACE GREATER_OP SPACE ID|NUMBER SPACE SMALLER_OP SPACE ID
|ID SPACE GREATER_OP SPACE NUMBER|ID SPACE SMALLER_OP SPACE NUMBER
;

<math_expr>:<math_expr> SPACE PLUS SPACE <math_term>| <math_expr> SPACE PLUS
<math_term>|<math_expr> PLUS <math_term>|<math_expr> PLUS SPACE <math_term>
| <math_expr> SPACE MINUS SPACE <math_term> |<math_expr> SPACE MINUS
<math_term>| <math_expr> MINUS <math_term>|<math_expr> MINUS SPACE< math_term>
| <math_term>
;
<math_term>: <math_term> SPACE MULTIPLIER SPACE <math_factor>| <math_term>
MULTIPLIER <math_factor> | <math_term> SPACE MULTIPLIER <math_factor>|
<math_term> MULTIPLIER SPACE <math_factor>
| <math_term> SPACE DIVISION SPACE <math_factor> | <math_term> DIVISION
<math_factor> | <math_term> SPACE DIVISION <math_factor>| <math_term> DIVISION
SPACE <math_factor>
| <math_factor>
;
<math_factor>:LEFT_PARA <math_expr>  RIGHT_PARA
| LEFT_PARA SPACE <math_expr> RIGHT_PARA
| LEFT_PARA <math_expr> SPACE RIGHT_PARA
| LEFT_PARA SPACE <math_expr> SPACE RIGHT_PARA
| ID
| NUMBER
;

<fnc>: <predicate> L_CURL <stmt> R_CURL | <predicate_instantiations>
;
<predicate>: PREDIFIER_FUNC | PREDIFIER_FUNC NL | PREDIFIER_FUNC SPACE
;
<predicate_instantiations>: PREDIFIER_INST_FCALL SEMICOLON
;
<assign>: <assi>|<assiSpace>
;
<assi>:<var> ASSIGN_OP NUMBER
|<var> ASSIGN_OP BOOLEAN
|<var> ASSIGN_OP <expr>
|<var>ASSIGN_OP ID
|<var> ASSIGN_OP L_CURL <arr> R_CURL
;
<assiSpace>:<var> SPACE ASSIGN_OP SPACE NUMBER
```

```
|<var> SPACE ASSIGN_OP SPACE BOOLEAN
|<var> SPACE ASSIGN_OP SPACE <expr>
|<var> SPACE ASSIGN_OP SPACE ID
|<var> SPACE ASSIGN_OP SPACE L_CURL <arr> R_CURL
;

<var>: VARIABLE | VARIABLE SPACE ARR L_SQR NUMBER R_SQR | ID
;

<arr>:NUMBER|<arr> COMMA NUMBER
;

<in_out_stmt>: <inp> |<inp> SPACE | <out> | <out> SPACE
;
<inp>: INP_STRM SPACE ID
;
<out>: OUT_STRM SPACE ID
;

<loop>:< while_stat> | <for_stat>
;
<while_stat>: WHILE LEFT_PARA SPACE <expr> SPACE RIGHT_PARA SPACE L_CURL
SPACE <stmt_e> SPACE R_CURL
;
<for_stat>: FOR LEFT_PARA SPACE <assign> SEMICOLON SPACE <bool_expr>
SEMICOLON SPACE <stmt_e> SPACE RIGHT_PARA SPACE L_CURL SPACE <stmt_e>
SPACE R_CURL
;
```

Test File:

After the following code inside a test file we get the "LANGUAGE IS CORRECT" output.

```
BEG
if(x < y)then{ x=x++5; }else{ a=TRUE; }
if(x < y)then{x=x**4;}
while( x < y ) { x++y; }
int abc1 arr[3]={3,4,1};
for( int i=4; x < y; x=x++2; ) { int abc1 arr[4]={4,3,56,1}; }
x&&y;
x||y;
x<->y;
x->y;
~x;
x==y;
x < y;
x > 5;
3 < 4;
x**y;
x**3;
a=x**3;
c=x--4;
d=x++4;
int b=x//5;
int i=4;
int abc1 arr[4]={4,3,56,1};
inp abc1;
out abc2;
```

```
f_foo( asd, asd21 , wqe2  );
bool f_name(int a, int b){bool a=TRUE;}
ENDBEG
```

If the user enter a syntax which is invalid for the language the user get the error output with the error line like in the following:

```
**** ERROR AT LINE NO 3 ****
```