Drăghici Andreea-Maria
CR4511A

C-minus-minus Project

1. C-- grammar

1.1 Rules

```
program : program varDecl | program fnDecl | /* empty */ ;
varDecl : type id ';' | type id '[' INTLITERAL ']' ';' ;
type : INT | BOOL | VOID ;
fnDecl : type id parameters block ;
parameters : '(' ')' | '(' formalsList ')' ;
formalsList : formalDecl | formalsList ',' formalDecl ;
formalDecl : type id ;
block : '{' declList stmtList '}' ;
declList : declList varDecl | /* empty */ ;
stmtList : stmtList stmt | /* empty */ ;
stmt : CIN READ id ';'
     | CIN READ id '[' exp ']' ';'
     | COUT WRITE exp ';'
     | subscriptExpr '=' exp ';'
     | id '=' exp ';'
     | IF '(' exp ')' block
     | IF '(' exp ')' block ELSE block
     | WHILE '(' exp ')' block
     | RETURN exp ';'
     | RETURN ';'
```

| fnCallStmt ';'
;

exp: exp '+' exp
| exp '-' exp
| exp '*' exp
| exp '/' exp
| '!' exp
| exp ANDAND exp
| exp OROR exp
| exp EQEQ exp
| exp NOTEQ exp
| exp '<' exp
| exp '>' exp
| exp LESSEQ exp
| exp GREATEREQ exp
| '_' atom
| atom
;

atom: INTLITERAL
| STRINGLITERAL
| TRUE
| FALSE
| '(' exp ')'
| fnCallExpr
| subscriptExpr
| id ;

Drăghici Andreea-Maria
CR4S1A

```
fnCallExpn : id '(' ')'
           | id '(' actualList ')'
           ;

fnCallStmt: id '(' ')'
          | id '(' actualList ')'
          ;

actualList : exp | actualList ',' exp ;

ArraySubscriptExpn: id '[' exp ']' ;

id : ID ;
```

1.2 Terminal symbols

1. INT
2. BOOL
3. VOID
4. EQ => "="
5. EQEQ => "=="
6. OROR => "||"
7. ANDAND => "&&"
8. NOTEQ => "!="
9. GREATEREQ => ">="
10. LESSEQ => "<="
11. "+"
12. "-"

Drăghici Andreea - Maria.
CR 4 S 1 A

13. " * "
14. " / "
15. " ! "
16. " < "
17. " > "
18. TRUE
19. FALSE
20. WHILE
21. IF
22. ELSE
23. RETURN
24. TYPE
25. " ; "
26. " ("
27. ") "
28. " ["
29. "] "
30. " , "
31. " | "
32. ID
33. INTLITERAL
34. STRINGLITERAL
35. " { "
36. " } "
37. " . "

38. "<<"

39. ">>"

40. CIN

41. COUT

1.3. Non-terminal symbols

< program >

< varDecl >

< fnDecl >

< parameters >

< formalsList >

< formalDecl >

< block >

< declList >

< stmtList >

< stmt >

< exp >

< otherwise >

< fnCallExpr >

< fnCallStmt >

< actualList >

< subscriptExpr >

Drăghici Andreea-Maria
CR 451A

1.4 Input file example

```
/* Ordinary recursion */

export sp1;
sp1(bits32 n){
bits32 s,p;
if n == 1 {
return (1,1);
} else {
s,p = sp1(n-1);
return (s+n, p*n);
}
}
```

2. Syntax-directed translator

2.1. Eliminating left recursion

Gramatica este recursivă și nu poate fi analizată în această formă.

Neterminalele <program>, <stmtList>, <exp>, <actualList> sunt lăsate recursive.

Adăugăm patru noi simboluri neterminale <program1> <stmtList1>, <exp1>, <actualList1> și următoarele

Drăghici Andreea-Maria
CR 4 S1A

Producții:

<program> -> <program1>

<program1> -> program1 varDecl
| program1 fnDecl
| /* empty */
;

<program1> -> ε
<stmtList> -> <stmtList1>

<stmtList1> -> stmtList1 stmt
| /* empty */
;

<stmtList1> -> ε

<exp> -> <exp1>

<exp1> -> '+' <exp1> <exp1>

<exp1> -> '-' <exp1> <exp1>

<exp1> -> '*' <exp1> <exp1>

<exp1> -> '/' <exp1> <exp1>

<exp1> -> 'AND' <exp1> <exp1>

<exp1> -> 'EQEQ' <exp1> <exp1>

<exp1> -> 'NOTEQ' <exp1> <exp1>

<exp1> -> '<' <exp1> <exp1>

<exp1> -> '>' <exp1> <exp1>

<exp1> -> 'LESSEQ' <exp1> <exp1>

Bloghici Andreea - Maria
CR 151A

<exp1> -> 'GREATEREQ' <exp1> <exp1>

<exp1> -> '!' <exp1>

<exp1> -> '-' <atom>

<exp1> -> ε

<actualList> -> <actualList1>

<actualList1> -> <exp1>
| <actualList1> ',' <exp1>

<actualList1> -> ε

Noua gramatică generează același limbaj, care poate fi analizată top-down:

<program> -> <program1>

<program1> -> program1 varDecl
| program1 fnDecl
| /* empty */
;

<program1> -> ε

<varDecl> -> type id ';'
| type id '[' INT_LITERAL ']' ';'
;

<type>: INT
BOOL

Drăghici Andreea-Maria
CR 4 S1 A

```
| VOID
;

<fnDecl> -> type id parameters block;

<parameters> -> '(' ')'
             | '(' formalsList ')'
             ;

<formalsList> -> formalDecl
              | formalsList ',' formalDecl
              ;

<formalDecl> -> type id;

<block> -> '{' declList stmtList '}';

<declList> -> declList varDecl
           | /* empty */
           ;

<stmtList> -> <stmtList1>
<stmtList1> -> stmtList1 stmt
            | /* empty */
            ;

<stmtList1> -> ε
```

Drăghici Amalia-Maria
CR4S1A

```
Stmt : cin READ id ';'
     | cin READ id '[' exp ']' ';'
     | COUT WRITE exp ';'
     | subscriptExpn '=' exp ';'
     | id '=' exp ';'
     | IF '(' exp ')' block
     | IF '(' exp ')' block ELSE block
     | WHILE '(' exp ')' block
     | RETURN exp ';'
     | RETURN ';'
     | fnCall Stmt ';'
     ;
```

<exp> -> <exp1>

<exp1> -> '+' <exp1> <exp1>

<exp1> -> '-' <exp1> <exp1>

<exp1> -> '*' <exp1> <exp1>

<exp1> -> '/' <exp1> <exp1>

<exp1> -> 'ANDAND' <exp1> <exp1>

<exp1> -> 'EQEQ' <exp1> <exp1>

<exp1> -> 'NOTEQ' <exp1> <exp1>

<exp1> -> '<' <exp1> <exp1>

Dobghici Andreea-Maria
CR4S1A

<exp1> -> '>' <exp1> <exp1>

<exp1> -> 'LESSEQ' <exp1> <exp1>

<exp1> -> 'GREATEREQ' <exp1> <exp1>

<exp1> -> '!' <exp1>

<exp1> -> '_' <otom>

<exp1> -> ε

```
<otom> : INTLITERAL
       | STRINGLITERAL
       | TRUE
       | FALSE
       | '(' exp ')'
       | fnCallExpn
       | subscriptExpn
       | id
       ;

fnCallExpn : id '(' ')'
           | id '(' actualList ')'
           ;

fnCallStmt : id '(' ')'
           | id '(' actualList ')'
           ;
```

Drăghici Andreea-Maria
CR4S1A

<actualList> -> <actualList1>

<actualList1> -> <exp1>
| <actual list1> ',' <exp1>

<actualList1> -> ε

~~<subscriptExpn>: expn1~~
~~| actualList1 ',' exp1~~
~~|~~

<subscriptExpn> : id '[' exp ']'
;

<id> : iD
;

Simbolurile non-terminale sunt:

<program>
<programs1>
<varDecl>
<fnDecl>
<parameters>
<formalList>
<formal Decl>
<block>
<declList>

Drăghici Andreea-Monia
CR 4 S1 A

< stmtList >

< stmtList1 >

< stmt >

< exp >

< exp1 >

< term >

< fnCallExpr >

< fnCall Stmt >

< actualList >

< actualList1 >

< subscriptExpr >

2.2 The First sets for non-terminal symbols

Because in the recursive descent parsing the lookahead symbol is not used, it will be replaced with the terminal symbols returned by the First() function.

First(< program >) = First(< program1 >) =

= First(< varDecl >) ∪ {ε}

= First(< fnDecl >) ∪ {ε}

First(< varDecl >) = First(< type >) = {id, INITLITERAL}

Drăghici Andreea - Maria
CR 4 SIA

First(<type>) = {int, bool, void}

First(<parameters>) = First(<formalsList>) = First(<formalsDecl>)
= {id}.

First(<block>) = First(<declList>) ∪ First(<stmtList>)
=

First(<stmtList>) = First(<stmtList1>) ∪ {ε}
First(<stmt>) = {CIN, COUT, RETURN, IF, WHILE}
First(<exp>) = {ident, int-cust}
First(<exp1>) = {'+'}
First(<exp1>) = {'-'}
First(<exp>) = {'*'}
First(<exp>) = {'/'}
First(<rel-operator>) = {'<', '>', '==', '<=', '>=', '!='}