**Final Grammar**

A

# Original Language

program = A 🡪 let B in I end

decs = B 🡪 C B | ε

dec = C 🡪 D | F

var\_dec = D 🡪 var ID := K | var ID : E

type = E 🡪 int | string | void

function\_dec = F 🡪 function ID (G) : E = I end | function ID ( ) : E = I end

parameters = G 🡪 G ; H | H

parameter = H 🡪 ID : E

statements = I 🡪 I ; J | J

statement = J 🡪 ID := K | printint ( K ) | printstring ( K ) | ID := getint( ) | ID ( ) | ID ( N ) | return K | return

expr = K 🡪 K + L | K – L | L

term = L 🡪 L + M | L / M | M

factor = M 🡪 ( K ) | NUMBER | STRING\_LITERAL | ID | ID ( ) | ID ( N )

expr\_list = N 🡪 N , K | K

# Left Factoring and Eliminating Left Recursion

A 🡪 let B in I end

B 🡪 C B | ε

C 🡪 D | F

D 🡪 var ID D`

D` 🡪 := K | : E

E 🡪 int | string | void

F 🡪 function ID ( F`

F` 🡪 G ) : E = I end | ) : E = I end

G 🡪 H G`

G` 🡪 ; H G` | ε

H 🡪 ID : E

I 🡪 J I`

I` 🡪 ; J I` | ε

J 🡪 ID := J` | ID ( J`` | return J``` | printint ( K ) | printstring ( K )

J` 🡪 K | getint( )

J`` 🡪 ) | N )

J``` 🡪 K | ε

K 🡪 L K`

K` 🡪 + L K` | - L K` | ε

L 🡪 M L`

L` 🡪 + M L` | / M L` | ε

M 🡪 ( K ) | NUMBER | STRING\_LITERAL | ID M`

M` 🡪 ( ) | ( N ) | ε

N 🡪 K N`

N` 🡪 , K N` | ε