Program -> decl {decl}

decl -> **void ID** fun-decl’ | **int ID** decl’

decl’ -> **;** | **[num]** | fun-decl’

fun-decl’ -> **(**params**)** compound-stmt

params -> param-list | **void**

param-list -> param {**,** param}

param -> **int ID** param’

param’ -> **[ ]** | epsilon

stmt -> expr-stmt | compound-stmt | selection-stmt | iteration-stmt | return-stmt

expr-stmt -> [expr ] **;**

compound-stmt -> **{** {**int ID** [ **[num]** ] } {stmt} **}**

selection-stmt -> **if** **(**expr**)** stmt [**else** stmt]

iteration-stmt -> **while (**expr**)** stmt

return-stmt -> **return** [expr] **;**

expr -> **ID** expr’ | **num** simple-expr’ | **(**expr**)** simple-expr’

expr’ -> = expr | **[**expr**]** expr’’ | **(**args**)** simple-expr’ | simple-expr’

expr’’ -> = expr | simple-expr’

simple-expr’ -> additive-expr’ [relop additive-expr]

relop -> **<=** | **<** | **>** | **>=** | **==** | **!=**

additive-expr -> term {addop term}

additive-expr’ -> term’ {addop term}

addop -> **+** | **-**

term -> factor {mulop factor}

term’ -> {mulop factor}

mulop -> **\*** | **/**

factor -> **(**expr**)** | **num** | **ID** varcall

varcall -> **[**expr**]** | **(**args**)** | **epsilon**

args -> args-list | **epsilon**

args-list -> expr { , expr}

**First sets**

Program -> {void, int}

decl -> {void, int}

decl’ -> { ; , [ , ( }

fun-decl’ -> { ( }

params -> {void, int}

param-list -> { int }

param -> { int }

param’ -> { [ , epsilon}

stmt -> {; , ID, NUM, ( , if, while, return}

expr-stmt -> {; , ID, NUM, ( }

compound-stmt -> { { }

selection-stmt -> {If}

iteration-stmt -> {while}

return-stmt -> {return}

expr -> {ID, NUM, ( }

expr’ ->{ =, ID, NUM, ( , \* , / }

expr’’ -> { =, ID, NUM, ( , \* , / }

simple-expr’ -> {\*, /}

relop -> {**< ,** **>, =** , **!**}

additive-expr -> { ( , num , ID }

additive-expr’ -> { **\***, **/** }

addop -> {**+**, **-**}

term -> { ( , num , ID }

term’ -> { **\***, **/** }

mulop -> { **\***, **/**}

factor -> { ( , num , ID}

varcall -> { [, (, epsilon}

args -> { ID, NUM, ( , epsilon}

args-list -> { ID, NUM, ( }

**Follow Sets**

Program -> {$}

decl -> {$ }

decl’ -> {$ }

fun-decl’ -> { $ }

params -> { ) }

param-list -> { ) }

param -> { ‘,’ , ) }

param’ -> { ‘,’ , ) }

stmt -> { else, } }

expr-stmt -> { else, } }

compound-stmt -> { $, else, } }

selection-stmt -> { else, } }

iteration-stmt -> { else, } }

return-stmt -> {else, } }

expr -> { ; , ), ], ‘,’ , =, ID, NUM, ( , \* , / }

expr’ ->{ ; , ), ], ‘,’ , =, ID, NUM, ( , \* , / }

expr’’ -> ; , ), ], ‘,’ , =, ID, NUM, ( , \* , / }

simple-expr’ -> { ; , ), ], ‘,’ , =, ID, NUM, ( , \* , / }

relop -> { ( , num , ID }

additive-expr -> { ; , ), ], ‘,’ , =, ID, NUM, ( , \* , / }

additive-expr’ -> { **< ,** **>, =** , **!**, ; , ), ], ‘,’ , =, ID, NUM, ( , \* , / }

addop -> { ( , num , ID }

term -> { **< ,** **> , =** , **!** , ; , ) , ] , ‘,’ , =, ID, NUM, ( , \* , / , **+**, **-**}

term’ -> { **+**, **-** }

mulop -> { ( , num , ID }

factor -> { **< ,** **> , =** , **!** , ; , ) , ] , ‘,’ , =, ID, NUM, ( , \* , / , **+**, **-** }

varcall -> { **< ,** **> , =** , **!** , ; , ) , ] , ‘,’ , =, ID, NUM, ( , \* , / , **+**, **-** }

args -> { ) }

args-list -> { ) }