Rule	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5
start -> PROGRAM IDENT ; varDec compStmt .	First(start) = { PROGRAM }				
varDec -> VAR varDecList	First(varDec) = { VAR }				
varDec -> ε	First(varDec) = { VAR, ε }				
varDecList -> identListType ; varDecList'			First(varDecList) = { IDENT }		
varDecList' -> identListType ; varDecList'			First(varDecList') = { IDENT, ε }		
varDecList' -> ε	First(varDecList') = { ε }				
identListType -> identList : type		First(identListType) = { IDENT }			
identList -> IDENT identList'	First(identList) = { IDENT }				
identList' -> , IDENT identList'	First(identList') = { "," }				
identList' -> ε	First(IdentList') = { "," , ε }				
type -> simpleType		First(type) = { INTEGER, REAL, BOOLEAN, ARRAY }			
type -> ARRAY [NUM .. NUM] OF simpleType	First(type) = { ARRAY }				
simpleType -> INTEGER	First(simpleType) = { INTEGER }				
simpleType -> REAL	First(simpleType) = { INTEGER, REAL }				
simpleType -> BOOLEAN	First(simpleType) = { INTEGER, REAL, BOOLEAN }				
compStmt -> BEGIN stmtList END	First(compStmt) = { BEGIN }				
stmtList -> statment stmtList'		First(stmtList) = { BEGIN, READ, WRITE }	First(stmtList) = { IDENT, BEGIN, IF, WHILE, FOR, READ, WRITE }		
stmtList' -> ; statement stmtList'	First(stmtList') = { ; }				
stmtList' -> ε	First(stmtList') = { ;, ε }				
statement -> assignStmt		First(statement) = { IDENT, BEGIN, READ, WRITE }			
statement -> compStmt	First(statement) = { BEGIN }				
statement -> ifStmt		First(statement) = { IDENT, BEGIN, IF, READ, WRITE }			
statement -> whileStmt		First(statement) = { IDENT, BEGIN, IF, WHILE, READ, WRITE }			
statement -> forStmt		First(statement) = { IDENT, BEGIN, IF, WHILE, FOR, READ, WRITE }			
statement -> READ (exprList)	First(statement) = { BEGIN, READ }				
statement -> WRITE (exprList)	First(statement) = { BEGIN, READ, WRITE }				
assignStmt -> IDENT assignStmt'	First(assignStmt) = { IDENT }				
assignStmt' -> := expr	First(assignStmt') = { := }				
assignStmt' -> [expr] := expr	First(assignStmt') = { :=, [}				
ifStmt -> IF expr THEN statement elsePart	First(ifStmt) = { IF }				
elsePart -> ELSE statement	First(elsePart) = { ELSE }				
elsePart -> ε	First(elsePart) = { ELSE, ε }				
whileStmt -> WHILE expr DO statement	First(whileStmt) = { WHILE }				
forStmt -> FOR IDENT := expr toPart expr DO statement	First(forStmt) = { FOR }				
toPart -> TO	First(toPart) = { TO }				
toPart -> DOWNTO	First(toPart) = { TO, DOWNTO }				

expr -> simpleExpr expr'				First(expr) = { NUM, FALSE, TRUE, IDENT, NOT, -, (, STRING }	
expr' -> ε	First(expr') = { ε }				
expr' -> relOp simpleExpr		First(expr') = { ε, <, <=, >, >=, =, <> }			
exprList -> expr exprList'					First(exprList) = { NUM, FALSE, TRUE, IDENT, NOT, -, (, STRING }
exprList' -> , expr exprList'	First(exprList') = { "," }				
exprList' -> ε	First(exprList') = { ",", ε }				
simpleExpr -> term simpleExpr'			First(simpleExpr) = { NUM, FALSE, TRUE, IDENT, NOT, -, (, STRING }		
simpleExpr' -> addOp term simpleExpr'		First(simpleExpr') = { +, -, OR, ε }			
simpleExpr' -> ε	First(simpleExpr') = { ε }				
term -> factor term'		First(term) = { NUM, FALSE, TRUE, IDENT, NOT, -, (, STRING }			
term' -> mulOp factor term'		First(term') = { *, /, DIV, MOD, AND, ε }			
term' -> ε	First(term') = { ε }				
factor -> NUM	First(factor) = { NUM }				
factor -> FALSE	First(factor) = { NUM, FALSE }				
factor -> TRUE	First(factor) = { NUM, FALSE, TRUE }				
factor -> IDENT factorIdent'	First(factor) = { NUM, FALSE, TRUE, IDENT }				
factor -> NOT factor	First(factor) = { NUM, FALSE, TRUE, IDENT, NOT }				
factor -> - factor	First(factor) = { NUM, FALSE, TRUE, IDENT, NOT, - }				
factor -> (expr)	First(factor) = { NUM, FALSE, TRUE, IDENT, NOT, -, (}				
factor -> STRING	First(factor) = { NUM, FALSE, TRUE, IDENT, NOT, -, (, STRING }				
factorIdent' -> ε	First(factorIdent') = { ε }				
factorIdent' -> [expr]	First(factorIdent') = { ε, [}				
relOp -> <	First(relOp) = { < }				
relOp -> <=	First(relOp) = { <, <= }				
relOp -> >	First(relOp) = { <, <=, > }				
relOp -> >=	First(relOp) = { <, <=, >, >= }				
relOp -> =	First(relOp) = { <, <=, >, >=, = }				
relOp -> <>	First(relOp) = { <, <=, >, >=, =, <> }				
addOp -> +	First(addOp) = { + }				
addOp -> -	First(addOp) = { +, - }				
addOp -> OR	First(addOp) = { +, -, OR }				
mulOp -> *	First(mulOp) = { * }				
mulOp -> /	First(mulOp) = { *, / }				
mulOp -> DIV	First(mulOp) = { *, /, DIV }				
mulOp -> MOD	First(mulOp) = { *, /, DIV, MOD }				
mulOp -> AND	First(mulOp) = { *, /, DIV, MOD, AND }				

Rule	Iteration 1	Iteration 2
start -> PROGRAM IDENT ; varDec compStmt .	Follow(start) = { $ } Follow(varDec) = { BEGIN } Follow(compStmt) = { . }	
varDec -> VAR varDecList	Follow(varDecList) = { BEGIN }	
varDec -> ε		
varDecList -> identListType ; varDecList'	Follow(identListType) = { ; } Follow(varDecList') = { BEGIN }	
varDecList' -> identListType ; varDecList'	Follow(identListType) = { ; } Follow(varDecList') = { BEGIN }	
varDecList' -> ε		
identListType -> identList : type	Follow(identList) = { : } Follow(type) = { ; }	
identList -> IDENT identList'	Follow(identList') = { : }	
identList' -> , IDENT identList'	Follow(identList') = { : }	
identList' -> ε		
type -> simpleType	Follow(simpleType) = { ; }	
type -> ARRAY [NUM .. NUM] OF simpleType	Follow(simpleType) = { ; }	
simpleType -> INTEGER		
simpleType -> REAL		
simpleType -> BOOLEAN		
compStmt -> BEGIN stmtList END	Follow(stmtList) = { END }	
stmtList -> statment stmtList'	Follow(statement) = { ;, END } Follow(stmtList') = { END }	
stmtList' -> ; statement stmtList'	Follow(statement) = { ;, END } Follow(stmtList') = { END }	
stmtList' -> ε		
statement -> assignStmt	Follow(assignStmt) = { ;, END }	Follow(assignStmt) = { ;, END, ELSE }
statement -> compStmt	Follow(compStmt) = { ., ;, END}	Follow(compStmt) = { ., ;, END, ELSE}
statement -> ifStmt	Follow(ifStmt) = { ;, END }	Follow(ifStmt) = { ;, END, ELSE }
statement -> whileStmt	Follow(whileStmt) = { ;, END }	Follow(whileStmt) = { ;, END, ELSE }
statement -> forStmt	Follow(forStmt) = { ;, END }	Follow(forStmt) = { ;, END, ELSE }
statement -> READ (exprList)	Follow(exprList) = {) }	
statement -> WRITE (exprList)	Follow(exprList) = {) }	
assignStmt -> IDENT assignStmt'	Follow(assignStmt') = { ;, END }	Follow(assignStmt') = { ;, END, ELSE }
assignStmt' -> := expr	Follow(expr) = { ;, END }	Follow(expr) = { ;, END,], THEN, DO, TO, DOWNTO ,",",), ELSE }
assignStmt' -> [expr] := expr	Follow(expr) = { ;, END,] } Follow(expr) = { ;, END,] }	
ifStmt -> IF expr THEN statement elsePart	Follow(expr) = { ;, END,], THEN } Follow(statement) = { ;, END, ELSE } Follow(elsePart) = { ;, END }	Follow(elsePart) = { ;, END, ELSE }
elsePart -> ELSE statement	Follow(statement) = { ;, END, ELSE }	
elsePart -> ε		
whileStmt -> WHILE expr DO statement	Follow(expr) = { ;, END,], THEN, DO } Follow(statement) = { ;, END, ELSE }	
forStmt -> FOR IDENT := expr toPart expr DO statement	Follow(expr) = { ;, END,], THEN, DO, TO, DOWNTO } Follow(toPart) = { NUM, FALSE, TRUE, IDENT, NOT, -, (, STRING } Follow(expr) = { ;, END,], THEN, DO, TO, DOWNTO } Follow(statement) = { ;, END, ELSE }	
toPart -> TO		
toPart -> DOWNTO		
expr -> simpleExpr expr'	Follow(simpleExpr) = { <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO } Follow(expr') = { ;, END,], THEN, DO, TO, DOWNTO }	Follow(simpleExpr) = { <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO, ",",), ELSE } Follow(expr') = { ;, END,], THEN, DO, TO, DOWNTO,",",), ELSE }
expr' -> ε		
expr' -> relOp simpleExpr	Follow(relOp) = { NUM, FALSE, TRUE, IDENT, NOT, -, (, STRING } Follow(simpleExpr) = { <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO }	

exprList -> expr exprList'	Follow(expr) = { ;, END,], THEN, DO, TO, DOWNTO ,",",) } Follow(exprList') = {) }	
exprList' -> , expr exprList'	Follow(expr) = { ;, END,], THEN, DO, TO, DOWNTO ,",",) } Follow(exprList') = {) }	
exprList' -> ε		
simpleExpr -> term simpleExpr'	Follow(term) = { +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO } Follow(simpleExpr') = { <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO }	Follow(term) = { <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO, ",",), ELSE } Follow(simpleExpr') = { <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO, ",",), ELSE }
simpleExpr' -> addOp term simpleExpr'	Follow(addOp) = { NUM, FALSE, TRUE, IDENT, NOT, -, (, STRING } Follow(term) = { +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO }	
simpleExpr' -> ε		
term -> factor term'	Follow(factor) = { *, /, DIV, MOD, AND, +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO } Follow(term') = { +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO }	Follow(factor) = { *, /, DIV, MOD, AND, +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO, ",",), ELSE } Follow(term') = { +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO, ",",), ELSE }
term' -> mulOp factor term'	Follow(mulOp) = { NUM, FALSE, TRUE, IDENT, NOT, -, (, STRING } Follow(factor) = { *, /, DIV, MOD, AND, +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO } Follow(term') = { +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO }	
term' -> ε		
factor -> NUM		
factor -> FALSE		
factor -> TRUE		
factor -> IDENT factorIdent'	Follow(factorIdent') = { *, /, DIV, MOD, AND, +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO }	Follow(factorIdent') = { *, /, DIV, MOD, AND, +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO, ",",), ELSE }
factor -> NOT factor	Follow(factor) = { *, /, DIV, MOD, AND, +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO }	
factor -> - factor	Follow(factor) = { *, /, DIV, MOD, AND, +, -, OR, <, <=, >, >=, =, <>, ;, END,], THEN, DO, TO, DOWNTO }	
factor -> (expr)	Follow(expr) = { ;, END,], THEN, DO, TO, DOWNTO ,",",) }	
factor -> STRING		
factorIdent' -> ε		
factorIdent' -> [expr]	Follow(expr) = { ;, END,], THEN, DO, TO, DOWNTO ,",",) }	
relOp -> <		
relOp -> <=		
relOp -> >		
relOp -> >=		
relOp -> =		
relOp -> <>		
addOp -> +		
addOp -> -		
addOp -> OR		
mulOp -> *		
mulOp -> /		
mulOp -> DIV		
mulOp -> MOD		
mulOp -> AND		