FIRST - SETS

	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5
Start → PROGRAM IDENT ; [VAR varDecList] CompStmt .	First(start) = {PROGRAM}				
VarDecList → identListType {; identListType}			First(VarDecList) = First(IdenListType) = { IDENT }		
IdentListType → identList : type		First(IdenListType) = First(IdentList) = { IDENT }			
IdentList → IDENT {, IDENT}	First(IdentList) = { IDENT }				
Type → simpleType		First(simpleType) = {ARRAY, INTGER, REAL, BOOLEAN}			
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 → statement {; statement}			First(StmtList) = First(Statement) = {BEGIN, READ, WRITE , IDENT, IF, WHILE, FOR}		
Statement → assignStmt		First(Statement) = {BEGIN, READ, WRITE , IDENT}			

Statement → compStmt	First(Statement) = {BEGIN}				
Statement → ifStmt		First(Statement) = {BEGIN, READ, WRITE , IDENT, IF}			
Statement → whileStmt		First(Statement) = {BEGIN, READ, WRITE , IDENT, IF, WHILE}			
Statement → forStmt		First(Statement) = {BEGIN, READ, WRITE , IDENT, IF, WHILE, FOR}			
Statement → READ '(' exprList ')'	First(Statement) = {BEGIN, READ}				
Statement → WRITE '(' exprList ')'	First(Statement) = {BEGIN, READ, WRITE}				
AssignStmt → IDENT [index] := expr	First(AssignStmt) = {IDENT}				
Index → '[' simpleExpr [.. simpleExpr] ']'	First(Index) = {[}				
IfStmt → IF expr THEN statement [ELSE statement]	First(IfStmt) = {IF}				
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}				
ExprList → expr {',' expr}					First(ExprList) = First(Expr) = { NUM, STRING, FALSE,

					TRUE, IDENT, NOT, -, (}
Expr → simpleExpr {relOp simpleExpr}				First(Expr) = First(SimpleExpr) = { NUM, STRING, FALSE, TRUE, IDENT, NOT, -, (}	
SimpleExpr → term {addOp term}			First(SimpleExpr) = First(Term) = { NUM, STRING, FALSE, TRUE, IDENT, NOT, -, (}		
Term → factor {mulOp factor}		First(Term) = First(Factor) = { NUM, STRING, FALSE, TRUE, IDENT, NOT, -, (}			
Factor → NUM | STRING | FALSE | TRUE | IDENT [index] | NOT factor | '–' factor | '(' exp ')'	First(Factor) = { NUM, STRING, FALSE, TRUE, IDENT, NOT, -, (}				
RelOp → < | <= | > | >= | = | <>	First(RelOp) = {<, <=, >, >=, =, <>}				
AddOp → + | – | OR	First(AddOp) = {+, -, OR}				
MulOp → * | / | DIV | MOD | AND	First(MulOp) = {*, /, DIV, MOD, AND}				

FOLLOW - SETS

	Iteration 1	Iteration 2
Start → PROGRAM IDENT ; [VAR varDecList] CompStmt .	Follow(Start) = {$}, Follow(varDecList) = First(CompStmt) = {BEGIN}, Follow(CompStmt) = {.}	
VarDecList → identListType {; identListType}	Follow(identListType) = {;} Follow(identListType) += Follow(VarDecList) = {; , BEGIN}	

IdentListType → identList : type	Follow(identList) = {:}, Follow(type) = Follow(IdentListType) = { ; , BEGIN}	
IdentList → IDENT {, IDENT}		
Type → simpleType	Follow(simpleType) = Follow(type) = { ; , BEGIN}	
Type → ARRAY '[' NUM .. NUM ']' OF simpleType	Follow(simpleType) = Follow(type) = { ; , BEGIN}	
SimpleType → INTEGER		
SimpleType → REAL		
SimpleType → BOOLEAN		
CompStmt → BEGIN stmtList END	Follow(StmtList) = {END}	
StmtList → statement {; statement}	Follow(Statement) = { ; } Follow(Statement) += Follow(StmtList) = { ; , END}	
Statement → assignStmt	Follow(assignStmt) = Follow(Statement) = { ; , END}	Follow(assignStmt) = Follow(statement) = { ; , END, ELSE}
Statement → compStmt	Follow(compStmt) = Follow(Statement) = { ; , END}	Follow(compStmt) = Follow(statement) = { ; , END, ELSE}
Statement → ifStmt	Follow(ifStmt) = Follow(Statement) = { ; , END}	Follow(ifStmt) = Follow(statement) = { ; , END, ELSE}
Statement → whileStmt	Follow(whileStmt) = Follow(Statement) = { ; , END}	Follow(whileStmt) = Follow(statement) = { ; , END, ELSE}
Statement → forStmt	Follow(forStmt) = Follow(Statement) = { ; , END}	Follow(forStmt) = Follow(statement) = { ; , END, ELSE}
Statement → READ '(' exprList ')'	Follow(exprList) = {) }	
Statement → WRITE '(' exprList ')'	Follow(exprList) = {) }	
AssignStmt → IDENT [index] := expr	Follow(index) = { : } Follow(expr) = Follow(assignStmt) = { ; , END}	Follow(expr) += Follow(assignStmt) = { ELSE, ; , END, THEN, DO, TO, DOWNTO, ',',) }
Index → '[' simpleExpr [.. simpleExpr] ']'	Follow(simpleExpr) = {.} Follow(simpleExpr) = {. ,]}	
IfStmt → IF expr THEN statement [ELSE statement]	Follow(expr) += THEN = { ; , END, THEN} Follow(statement) += ELSE = { ; , END, ELSE} Follow(statement) += Follow(IfStmt) = { ; , END, ELSE}	Follow(statement) += Follow(IfStmt) = { ; , END, ELSE}
WhileStmt → WHILE expr DO statement	Follow(expr) += DO = { ; , END, THEN, DO}	Follow(statement) += Follow(WhileStmt) = { ; , END, ELSE}

	Follow(statement) += Follow(WhileStmt) = { ; , END, ELSE}	
ForStmt → FOR IDENT ':=' expr toPart expr DO statement	Follow(expr) += First(ToPart) = { ; , END, THEN, DO, TO, DOWNTO} Follow(expr) += DO = { ; , END, THEN, DO, TO, DOWNTO} Follow(statement) += Follow(ForStmt) = { ; , END, ELSE}	Follow(statement) += Follow(ForStmt) = { ; , END, ELSE}
ToPart → TO		
ToPart → DOWNTO		
ExprList → expr {',' expr}	Follow(expr) += ',' = { ; , END, THEN, DO, TO, DOWNTO, ','} Follow(expr) += Follow(ExprList) = { ; , END, THEN, DO, TO, DOWNTO, ',',) }	
Expr → simpleExpr {relOp simpleExpr}	Follow(simpleExpr) += First(relOp) = {. ,], <, <=, >, >=, =, <>} Follow(simpleExpr) += Follow(expr) = {. ,], <, <=, >, >=, =, <>, ;, END, THEN, DO, TO, DOWNTO, ',',) }	
SimpleExpr → term {addOp term}	Follow(term) = First(addOp) = {+, -, OR} Follow(term) += Follow(SimpleExpr) = {+, -, OR, . ,], <, <=, >, >=, =, <>, ;, END, THEN, DO, TO, DOWNTO, ',',) }	
Term → factor {mulOp factor}	Follow(factor) = First(mulOp) = {*, /, DIV, MOD, AND} Follow(factor) += Follow(term) = {*, /, DIV, MOD, AND, +, -, OR, . ,], <, <=, >, >=, =, <>, ;, END, THEN, DO, TO, DOWNTO, ',',) }	
Factor → NUM | STRING | FALSE | TRUE | IDENT [index] | NOT factor | '–' factor | '(' expr ')'	Follow(index) +=] = { :,] } Follow(expr) +=) = { ; , END, THEN, DO, TO, DOWNTO, ',',) }	
RelOp → < | <= | > | >= | = | <>		
AddOp → + | – | OR		
MulOp → * | / | DIV | MOD | AND		