Gramática sin problemas de factorización ni recursividad izquierda:

P→Block#
Block → ConstDecl VarDecl ProcDecl Statement
ConstDecl → const ConstAssigList;|λ
ConstAssigList→ id = num ConstAssigList`
ConstAssigList`→ , id = num ConstAssigList`|λ
VarDecl→ var IdList ;|λ
IdList → id IdList
IdList` → ,id IdList`|λ
ProcDecl→procedure id ; Block ; ProcDecl|λ
Statement → id :=Expression|call id|begin StatementList end|if Condition then Statement|while Condition do Statement|λ
StatementList→ Statement StatementList`
StatementList`→ ;Statement StatementList`|λ
Condition→Expression Relation Expression|odd Expression
Expression→ SumOperator Term Expression`|Term Expression`
Expression`→ SumOperator Term Expression`|λ
Term→ Factor Term`
Term`→MultOperator Factor Term`|λ
Relation→=|<>|<|>|<=|>=
SumOperator→+|-
MultOperator→/|*
Factor→(Expression)|id|num

Tabla de análisis sintáctico correspondiente:

	Program	Block	ConstDecl	ConstAssigList	ConstAssigList`	VarDecl	IdList
Const	P→Block #	Block → ConstDecl VarDecl ProcDecl Statement	ConstDecl → const ConstAssigList;				
#	P→Block #	Block → ConstDecl VarDecl ProcDecl Statement	ConstDecl → λ			VarDecl → λ	
var	P→Block #	Block → ConstDecl VarDecl ProcDecl Statement	ConstDecl → λ			VarDecl → var IdList ;	
procedure	P→Block #	Block → ConstDecl VarDecl ProcDecl Statement	ConstDecl → λ			VarDecl → λ	
id	P→Block	Block →	ConstDec	ConstAssi		VarDecl	IdList →

	#	ConstDecl VarDecl ProcDecl Statement	l → λ	gList→ id = num ConstAssigList`		→ λ	id IdList`
call	P→Block #	Block → ConstDecl VarDecl ProcDecl Statement	ConstDecl → λ			VarDecl → λ	
begin	P→Block #	Block → ConstDecl VarDecl ProcDecl Statement	ConstDecl → λ			VarDecl → λ	
if	P→Block #	Block → ConstDecl VarDecl ProcDecl Statement	ConstDecl → λ			VarDecl → λ	
while	P→Block #	Block → ConstDecl VarDecl ProcDecl Statement	ConstDecl → λ			VarDecl → λ	
;		Block → ConstDecl VarDecl ProcDecl Statement	ConstDecl → λ		ConstAssigList→ λ	VarDecl → λ	
,					ConstAssigList→ , id = num ConstAssigList`		

	IdList`	ProDecl	Statement	Statement List	Statement List`	Condition	Expression
Const							
#		ProDecl → λ	Statement →λ				
var							
procedure		ProcDecl →proced					

		ure id ; Block ; ProcDecl					
id		ProDecl → λ	Statement → id :=Expression	Statement List→ Statement Statement List`			
call		ProDecl → λ	Statement →call id	Statement List→ Statement Statement List`			
begin		ProDecl → λ	Statement →begin Statement List end	Statement List→ Statement Statement List`			
if		ProDecl → λ	Statement →if Condition then Statement	Statement List→ Statement Statement List`			
while		ProDecl → λ	Statement →while Condition do Statement	Statement List→ Statement Statement List`			
;	IdList → ,id IdList`	ProDecl → λ	Statement → λ		Statement List`→ ;Statement Statement List`		
end					Statement List`→ λ		
,	IdList → ,id IdList`						
+						Condition →Expression Relation Expression	Expression→ SumOperator Term Expression`
(						Condition	Expressio

						→Expression Relation Expression	n→ Term Expression`
id						Condition →Expression Relation Expression	Expression→ Term Expression`
num						Condition →Expression Relation Expression	Expression→ Term Expression`
odd						Condition → odd Expression	
-						Condition →Expression Relation Expression	Expression→ SumOperator Term Expression`

	Expression`	Term	Term`	Relation	SumOperator	MultOperator	Factor
#	Expression`→λ						
*			Term`→ MultOperator Factor Term`			MultOperator→*	
=	Expression`→λ			Relation →=			
<>	Expression`→λ			Relation →<>			
<	Expression`→λ			Relation →<			
>	Expressio			Relation			

	n`→λ			→>			
<=	Expression`→λ			Relation →<=			
>=	Expression`→λ			Relation →>=			
;	Expression`→λ						
num		Term→ Factor Term`					Factor→num
/			Term`→ MultOperator Factor Term`			MultOperator→/	
+	Expression`→ SumOperator Term Expression`		Term`→λ		SumOperator→+		
(		Term→ Factor Term`					Factor→(Expression)
id		Term→ Factor Term`					Factor→id
do	Expression`→λ						
then	Expression`→λ						
-	Expression`→ SumOperator Term Expression`		Term`→λ		SumOperator→-		
)	Expression`→λ						