

## Gramática Corregida

S' -> Program  
Program -> Decl Program  
Program -> Decl  
Decl -> Type ident ;  
Decl -> Type ident ( Formals ) StmtBlock  
Decl -> void ident ( Formals ) StmtBlock  
Decl -> class ident Id Id' { Field' }  
Decl -> interface ident { Prototype' }  
Decl -> const ConstType ident ;  
Type -> int  
Type -> double  
Type -> bool  
Type -> string  
Type -> ident  
Type -> Type []  
ConstType -> int  
ConstType -> double  
ConstType -> bool  
ConstType -> string  
Formals -> Type ident , Formals  
Formals -> Type ident  
Id -> : ident  
Id ->  $\epsilon$   
Id' -> , ident Id'  
Id' ->  $\epsilon$   
Field' -> Field Field'  
Field' ->  $\epsilon$   
Field -> Type ident ;  
Field -> Type ( Formals ) StmtBlock  
Field -> void ( Formals ) StmtBlock  
Field -> const ConstType ident ;  
Prototype' -> Prototype Prototype'  
Prototype' ->  $\epsilon$   
Prototype -> Type ident ( Formals ) ;  
Prototype -> void ident ( Formals ) ;  
StmtBlock -> { VariableDecl' ConstDecl' Stmt' }  
ConstDecl' -> const ConstType ident ; ConstDecl'  
ConstDecl' ->  $\epsilon$   
VariableDecl' -> Type ident ; VariableDecl'  
VariableDecl' ->  $\epsilon$   
Stmt' -> Stmt Stmt'  
Stmt' ->  $\epsilon$   
Stmt -> Expr ;  
Stmt -> ;  
Stmt -> if ( Expr ) Stmt IfStmt  
Stmt -> while ( Expr ) Stmt  
Stmt -> for ( Expr ; Expr ; Expr ) Stmt  
Stmt -> break ;  
Stmt -> return Expr ;  
Stmt -> Console . Writeline ( Expr' , ) ;

Stmt -> StmtBlock  
 IfStmt -> else Stmt  
 IfStmt ->  $\epsilon$   
 Expr' -> Expr , Expr'  
 Expr' ->  $\epsilon$   
 Expr -> ident = ConditionAnd  
 Expr -> ConditionAnd  
 ConditionAnd -> Equality ConditionAnd'  
 ConditionAnd' -> && Equality ConditionAnd'  
 ConditionAnd' ->  $\epsilon$   
 Equality -> Equality == Relational  
 Equality -> Relational  
 Relational -> Relational < Additive  
 Relational -> Relational <= Additive  
 Relational -> Additive  
 Additive -> Additive + Multiplicative  
 Additive -> Multiplicative  
 Multiplicative -> Multiplicative \* Unary  
 Multiplicative -> Multiplicative % Unary  
 Multiplicative -> Unary  
 Unary -> - Primary  
 Unary -> ! Primary  
 Unary -> Primary  
 Primary -> Primary . ident  
 Primary -> Primary . ident = Expr  
 Primary -> Terminal  
 Terminal -> this  
 Terminal -> ( Expr )  
 Terminal -> New ( ident )  
 Terminal -> intConstant  
 Terminal -> doubleConstant  
 Terminal -> boolConstant  
 Terminal -> stringConstant  
 Terminal -> null  
 Terminal -> ident