Proyecto Diseño de compiladores

Curso: Diseño de compiladores

Sesiones: miércoles (3 horas)

Modalidad del entregable: Aplicación móvil, web o escritorio

Objetivo del proyecto

Construir, de forma incremental, un mini entorno de compilación para un lenguaje didáctico **LL(1).** El equipo deberá:

1. Identificar símbolos terminales y no terminales de una gramática dada.
2. Implementar un analizador léxico (tokenizador).
    a. Deberá procesar un código fuente.
    b. En caso de no existir error alguno, mostrar la tabla de tokens generada.
    c. En caso de existir un error, detener la ejecución del compilador, mostrar en que línea del código fuente se encuentra el error y opcionalmente mostrar que token(s) esperaba el analizador.
3. Obtener el árbol de derivación de la gramática.
4. Implementar un analizador sintáctico.
5. Integrar una interfaz gráfica que permita lo siguiente:
    a. Escribir, guardar y editar un código fuente.
    b. Mostrar tabla de tokens.
    c. Mostrar errores léxicos.
    d. Mostrar si el análisis sintáctico tuvo algún error o si no, mostrar una alerta de que el código analizado paso ambos analizadores de forma exitosa.
6. Subir el proyecto a un repositorio de Git en donde se tienen que ver los hashes de los commits.

Notas:

- Los analizadores deberán ignorar los comentarios del código, los cuales estarán dados por la siguiente expresión regular: //[A-Za-z0-9]+//.
- Los Id's estarán deberán comenzar por una letra (minúscula o mayúscula).
- Los string_lit estarán dados de la siguiente forma: "Texto en string lit".
- Los num serán enteros, con punto decimal o notación científica ej. 12.2, 123 12e21, 13e-12.

- Los comentarios deberán estar de la siguiente forma: //comentario//.

Gramática

Program → DeclFunList EOF

DeclFunList → DeclFun DeclFunList

DeclFunList → ε

DeclFun → Decl

DeclFun → FunDef

Decl → Type ID ArrOpt DeclTail

DeclTail → '=' Expr ';'

DeclTail → ';'

ArrOpt → '[' NUM ']' ArrOpt

ArrOpt → ε

FunDef → Type ID '(' ParamListOpt ')' Block

ParamListOpt → ParamList

ParamListOpt → ε

ParamList → Param ParamListTail

ParamListTail → ',' Param ParamListTail

ParamListTail → ε

Param → Type ID ArrOpt

Type → int

Type → bool

Type → void

Block → '{' StmtList '}'

StmtList → Stmt StmtList

StmtList → ε

Stmt → Block

Stmt → Decl

Stmt → ExprStmt

Stmt → IfStmt

Stmt → WhileStmt

Stmt → ForStmt

Stmt → ReturnStmt

Stmt → BreakStmt

Stmt → ContinueStmt

ExprStmt → Expr ';'

ExprStmt → ';'

IfStmt → if '(' Expr ')' Stmt ElseOpt

ElseOpt → else Stmt

ElseOpt → ε

WhileStmt → while '(' Expr ')' Stmt

ForStmt → for '(' ForInit ';' ForCond ';' ForIter ')' Stmt

ForInit → Expr

ForInit → ε

ForCond → Expr

ForCond → ε

ForIter → Expr

ForIter → ε

ReturnStmt → return Expr ';'

ReturnStmt → return ';'

BreakStmt → break ';'

ContinueStmt → continue ';'

Bloque común:

Expr → Assign

Assign → Or AssignTail

AssignTail → '=' Assign

AssignTail → ε

Or → And OrTail

OrTail → '||' And OrTail

OrTail → ε

And → Eq AndTail

AndTail → '&&' Eq AndTail

AndTail → ε

Eq → Rel EqTail

EqTail → '==' Rel EqTail

EqTail → '!=' Rel EqTail

EqTail → ε

Rel → Add RelTail

RelTail → '<' Add RelTail

RelTail → '<=' Add RelTail

RelTail → '>' Add RelTail

RelTail → '>=' Add RelTail

RelTail → ε

Add → Mul AddTail

AddTail → '+' Mul AddTail

AddTail → '-' Mul AddTail

AddTail → ε

Mul → Unary MulTail

MulTail → '*' Unary MulTail

MulTail → '/' Unary MulTail

MulTail → '%' Unary MulTail

MulTail → ε

Unary → '!' Unary

Unary → '-' Unary

Unary → Postfix

Postfix → Primary PostfixTail

PostfixTail → '(' ArgListOpt ')' PostfixTail

PostfixTail → '[' Expr ']' PostfixTail

PostfixTail → '.' ID PostfixTail

PostfixTail → ε

Primary → ID

Primary → NUM

Primary → STRING

Primary → 'true'

Primary → 'false'

Primary → '(' Expr ')'

ArgListOpt → ArgList

ArgListOpt → ε

ArgList → Expr ArgListTail

ArgListTail → ',' Expr ArgListTail

ArgListTail → ε