**Compiladores**

Práctica 3 – Analizadores sintáctico descendente recursivo

**Objetivo:**

- Construir un analizador sintáctico descendente recursivo.

- Modelar una estructura sintáctica.

**Requisitos:**

Analizador léxico construído en la práctica 1.

Tal analizador recibe una sentencia y retorna una lista (o vector) de tokens.

**Modalidad:**

Individual o grupo de hasta 2 personas.

**Indicaciones:**

Considerar la siguiente gramática:

COM := EXPR | IGUALDAD

IGUALDAD := VAR IGUAL EXPR

EXPR := EXPR1 | EXPR2 | EXPR3

EXPR1 := EXPR OP EXPR

EXPR2 := ( EXPR )

EXPR3 := VAR | NUM

IGUAL := ‘=’

OP := ‘+’ | ‘-’ | ’\*’ | ‘/’

VAR := \_?[a-zA-Z][a-zA-Z0-9\_]\*

NUM := [0-9]

Los elementos en rojo son reconocidos por el analizador léxico

Por ejemplo la gramática consigue reproducir / reconocer:

1+1

1\*(2+1)

temp1 = 5

temp2 = 10

temp3 = Temp1+Temp2

9999 + temp1 + temp2 + temp3

**Ejercicios:**

1. Adicionar la capacidad del analizador léxico de poder reconocer los paréntesis y el símbolo de igual.
2. Crear una función **getEXPR(*tokens*)** que reciba la lista de tokens y me retorne un par ordenado **<inicio, fin>** que indican los índices en el vector de tokens. Si la operación retorna <-1,-1> significa que no pudo reconocer una **EXPR**.
3. Crear una función **getIGUALDAD(*tokens*)** con funcionamiento similar al anterior
4. Crear una función para cada EXPR del tipo **getEXPR**:

**getEXPR1**(***tokens***)

**getEXPR2**(***tokens***)

**getEXPR3**(***tokens***)

* **También es válido pasar los índices por parámetro en las funciones.**

**Hint:**

typedef pair<int,int> Coord;

Coord **getCOM**( vector<Token> tokens) {

Coord expr = **getEXPR**(tokens);

Coord igualdad = **getIGUALDAD**(tokens);

if( expr.first != -1 )

return expr;

if( igualdad.first != -1 )

return igualdad;

return make\_pair(-1,-1);

}

**Ejemplo**

***variable3 = ( (1+2) \* (3+4) ) \* ( variable1 + variable 2 )***

* CMD debe reconocer una IGUALDAD de <0,20>
* La IGUALDAD debe reconocer una VAR de <0,0>, IGUAL de <1,1> y EXPR de <2,20>
* La EXPR debe reconocer una EXPR1 desde <2,20>
* La EXPR1 debe reconocer una EXPR desde <2,14>, un OP <15,15> y un EXPR desde <16,20>
* Y así sucesivamente ...