***Procesadores de Lenguajes Memoria Práctica 4***

**Equipo MultiProRobo:**

* **Hernán Indíbil de la Cruz Calvo**
* **Alejandro Martín Simón Sánchez**
* **Alejandro Zornoza Martínez**

***ÍNDICE***

[1. Analizador Léxico 1](#_Toc502629256)

[2. Analizador Sintáctico 2](#_Toc502629257)

[3. Analizador Semántico 3](#_Toc502629258)

# Introducción

En esta práctica se implementarán las partes de analizador léxico, analizador sintáctico y analizador semántico de un compilador.

Para describir informalmente como es el lenguaje se describe a continuación como es el programa:

El objetivo es procesar un lenguaje que comienza con la palabra PROGRAMA y un identificador, a continuación, constan las siguientes partes:

* Declaración de variables globales. Es opcional.
* Declaración de funciones. Es opcional.
* Sentencias compuestas que representa el cuerpo del programa principal: conjunto de instrucciones que van entre las palabras INICIO y FIN, terminadas en punto y coma.

Se espera que en caso de no haber fallos se indique que ha ido todo correctamente, y en caso de encontrar un fallo indicar donde se encuentra y qué se esperaba.

# Analizador Léxico

Para realizar el analizador léxico lo primero que se ha realizado ha sido identificar los diferentes tipos de componentes que pueden aparecer en el programa:

Palabras clave = {PROGRAMA, VAR, VECTOR, ENTERO, REAL, BOOLEANO, PROC, FUNCION, INICIO, FIN, SI, ENTONCES, SINO, MIENTRAS, HACER, LEE, ESCRIBE, Y, O, NO, CIERTO}

OpRel = {=, <>, <, <=, >=, >}

OpAdd = {+, -}

OpMult = {\*, /}

OpAsigna = {:=}

Así el lenguaje es la unión de: OpRel + OpRel + OpAdd + OpMult + OpAsigna + [a-zA-Z0-9:;,\[\]\(\)]

Para implementar dichos componentes sobre el analizador hay que tener en cuenta qué se debe almacenar acerca de ellos, en cuanto a los operadores almacenan el operador. Sobre los identificadores y las palabras reservadas se almacenan el valor y el número de línea. Acerca de los números se almacenan un booleano que indica si es entero o no, en caso de no serlo se trataría como real, y el valor del número. Del resto de componentes no es necesario almacenar ningún atributo.

Sobre el analizador léxico que hemos implementado cabe destacar que basa su ejecución en la lectura de todo el fichero carácter a carácter hasta que llega al fin de fichero, en función del carácter que encuentre tomará un estado u otro del MDD implementado.

Además hay que tener en cuenta que se incluye la posibilidad de insertar comentarios, así si detecta una llave de apertura “{“ ignora todo lo que haya después hasta que encuentre el cierre “}”, de no encontrarlo habría un error.

….

# Analizador Sintáctico

La gramática no es LL1, por lo que ha hay que adaptarla (¿?)

# Analizador Semántico

Para implementar la parte del analizador semántico que trata acerca de comprobar la aparición de identificadores repetidos se han añadido las siguientes reglas a la gramática:

<Programa> -> PROGRAMA id ; {<decl\_var>.ids = [id.valor]} <decl\_var> {<decl\_subprg>.ids = <decl\_var>.ids} <decl\_subprg> <instrucciones> .

<decl\_var> -> VAR {<lista\_id>.ids = <decl\_var>.ids} <lista\_id> : <tipo> ; {<decl\_v>.ids = <lista\_id>.ids} <decl\_v> {<decl\_var>.ids = <decl\_v>.ids} | λ

<decl\_v> -> {<lista\_id>.ids = <decl\_v>.ids} <lista\_id> : <tipo> ; {<decl\_v>1.ids = <lista\_id>.ids} <decl\_v>1 {<decl\_v>.ids = <decl\_v>1.ids} | λ

<lista\_id> -> id {if id.valor in <lista\_id>.ids then ERROR(identif repetido);

<lista\_id>.ids.append(id.valor);

<resto\_listaid>.ids = <lista\_id>.ids} <resto\_listaid> {<lista\_id>.ids = <resto\_listaid>.ids}

<resto\_listaid> -> , {<lista\_id>.ids = <resto\_listaid>.ids} <lista\_id> {<resto\_listaid>.ids = <lista\_id>.ids}

-> λ

<tipo> -> <tipo\_std> | VECTOR [ num ] de <tipo>

<tipo\_std> -> ENTERO | REAL | BOOLEANO

<decl\_subprg> -> {<decl\_sub>.ids = <decl\_subprg>.ids} <decl\_sub> ; {<decl\_subprg>1.ids = <decl\_sub>.ids} <decl\_subprg>1 {<decl\_subprg>.ids = <decl\_subprg>1.ids} | λ

<decl\_sub> -> PROC id {if id.valor in <decl\_sub>.ids then ERROR(identif repetido);

<decl\_sub>.ids.append(id.valor);} ; <instrucciones>

-> FUNCION id {if id.valor in <decl\_sub>.ids then ERROR(identif repetido);

<decl\_sub>.ids.append(id.valor);} : <tipo\_std> ; <instrucciones>

<instrucciones> -> INICIO <lista\_inst> FIN

<lista\_inst> -> <instruccion> ; <lista\_inst> | λ

<instruccion> -> INICIO <lista\_inst> FIN

-> <inst\_simple>

-> <inst\_e/s>

-> SI <expresion> ENTONCES <instruccion> ; SINO <instruccion>

-> MIENTRAS <expresion> HACER <instruccion>

<inst\_simple> -> id <resto\_instsimple>

<resto\_instsimple> -> opasigna <expresion> | [ <expr\_simple> ] opasigna <expresion> | λ

<variable> -> id <resto\_var>

<resto\_var> -> [ <expr\_simple> ] | λ

<inst\_e/s> -> LEE ( id ) | ESCRIBE ( <expr\_simple> )

<expresion> -> <expr\_simple> <expr\_aux>

<expr\_aux> -> oprel <expr\_simple> | λ

<expr\_simple> -> <termino> <resto\_exsimple> | <signo> <termino> <resto\_exsimple>

<resto\_exsimple> -> opsuma <termino> <resto\_exsimple> | O <termino> <resto\_exsimple> | λ

<termino> -> <factor> <resto\_term>

<resto\_term> -> opmult <factor> <resto\_term> | Y <factor> <resto\_term> | λ

<factor> -> <variable> | num | ( <expresion> ) | NO <factor> | CIERTO | FALSO

<signo> -> + | -

De esta forma se implementa el analizador semántico sobre el analizador semántico, añadiendo los identificadores encontrados a una lista, pasándolo por parámetro y devolviéndolo en las funciones requeridas.

En caso de encontrar un identificador repetido muestra un error por cada identificador que se repite.

Fragmento de código de ejemplo:

def analyzeListaId(self, \*\*kwargs):

        ids = kwargs['ids']

        if (self.component == None):

            return ids

        if (self.component.cat == "Identif"):

            v = self.component.valor

            if(v not in ids):

                ids.append(v)

            else:

                self.errorS(id = v)

            self.advance()

            ids = self.analyzeRestoListaId(ids = ids)

        else:

            self.error(msg='Identif',

                sync=set([None, "DosPtos"]))

        return ids

Corresponde a:

<lista\_id> -> id {if id.valor in <lista\_id>.ids then ERROR(identif repetido);

<lista\_id>.ids.append(id.valor);

<resto\_listaid>.ids = <lista\_id>.ids} <resto\_listaid> {<lista\_id>.ids = <resto\_listaid>.ids}

A las funciones los parámetros se les pasan por “key arguments”, utilizando la key “ids”. Como se puede observar, si se encuentra un identificador, comprueba si está el valor en la lista de identificadores, en caso de no estar añade el identificador y continúa su ejecución, posteriormente se pasa la lista de ids a analyzeRestoListaId, que a su vez devuelve la lista, siendo este último el valor de la lista que devuelve analyzeListaId. En caso de estar presente en la lista muestra que ese identificador está repetido y no se vuelve a añadir.

# Conclusiones

Al realizar el analizador sintáctico y al realizar distintas pruebas que debería pasar detectamos un error en la gramática definida inicialmente, faltando un punto y coma antes del “SINO” en la especificación sintáctica.

<instruccion> -> SI <expresion> ENTONCES <instruccion> ; SINO <instruccion>

De esta forma queda solucionado, pero evidentemente modifica el conjunto de siguientes.