David Vacas Miguel (GII+GIS)

Pedro Redondo Rabanal (GII+GIS)

PROCESADORES DE LENGUAJES

Analizador léxico y sintáctico

**Introducción**

En esta práctica debemos realizar un procesador de una modificación del lenguaje Pascal. En esta primera entrega hemos realizado la parte correspondiente al analizador léxico y al sintáctico. Para la realización de la práctica hemos utilizado la última versión del enunciado; en el cual, se arreglaban ciertas erratas que había en este.

La aplicación resultante se ha comprobado su funcionamiento en Windows 10 y con dos jdk distintos, uno version 1.8.0\_73 y otro 1.8.0\_60 .

**Especificación léxica**

Para la realización del analizador léxico hemos tenido en cuenta la sintaxis básica de un programa en pascal proporcionada en el enunciado:

-Palabras reservadas: begin, end, var, procedure, function, program, REAL, INTEGER, const, CHARACTER, div, mod, or, and, not, type, array, of, record, if, then, else, while, do, for, to, case.

-Símbolos: "," , ";" , "." , ".." , "=" , "+" , ":" , ":=" , "<" , ">" , "<=" , ">=" , "<>" , "-" , "\*" , "(" , ")" , "[" , "]" .

-Constantes:

* Literales: ristras de símbolos entre comillas simples. Para poder identificar las constantes literales una vez se recibe una comilla simple se pasa a un estado donde sí se reciben dos comillas simples seguidas se transforman en una y sólo se vuelve al estado inicial cuando se encuentra con una única comilla simple.
* Numéricas: este tipo de constantes puede ser tanto real o entera y decimal o hexadecimal. Para su correcta identificación se han realizado expresiones regulares para cada una de las combinaciones que son:

constanteNumericaEnteraDecimal = [+-]?[0-9]+

constanteNumericaRealDecimal = [+-]?[0-9]+"."[0-9]+

constanteNumericaEnteraHexadecimal =

"$"[+-]?[A-F0-9]+

constanteNumericaRealHexadecimal =

"$"[+-]?[A-F0-9]+"."[A-F0-9]+

-Identificadores: al igual que con las constantes literales, para la adecuada categorización de los identificadores, hemos hecho la siguiente expresión regular: [\_a-zA-Z][a-zA-Z0-9\_]\*

También reconocemos los comentarios que se escriben con “(\* \*)” o con “{ }”, sólo que dichos comentarios una vez son reconocidos no hacemos nada con ellos; ya que, su funcionalidad es ser ignorados.

A la hora de detectar un error en el léxico mostramos un mensaje por pantalla donde se podrá observar la palabra que no existe en nuestro alfabeto y su posición.

Por último, en el análisis léxico nos vimos obligados a reconocer los espacios y los saltos de línea ya que nos daban fallo cuando comenzamos a trabajar en la gramática.

**Especificación sintáctica**

Para la realización del analizador sintáctico hemos decidido realizar también parte de la gramática opcional:

-De las sentencias de control de flujo somos capaces de reconocer: while, for y case.

-De los tipos de datos somos capaces de reconocer todos los tipos dados que son: matrices y registros.

Debido a que la gramática proporcionada tenía cierto conflicto al intentar reconocer EXP, tuvimos que modificarla de la siguiente manera:

EXP ::= FACTOR EXPAUX

EXPAUX ::= OP EXP | lambda

Por último, decidimos dar un mayor detalle a la detección de errores y para ello hemos sobrescrito el método de cup llamado syntax\_error, donde ahora hemos puesto la palabra que da el error, su línea y su columna además de los posibles tokens que esperaba el analizador.

**Casos de prueba**

**Correctos**

* En este caso comprobamos el correcto funcionamiento de un programa que calcula el factorial de un número, con esto comprobamos específicamente el funcionamiento de funciones y bucles for.

program acierto1ForFun;

function factorial ( num : INTEGER ) : INTEGER;

var

id,total : INTEGER;

begin

for id:=1 to num do

total := total + id;

factorial:=total;

end;

var

input: INTEGER;

output: INTEGER;

begin

input := 6;

output:= factorial(input);

end.

* En este segundo caso se hace una simulación muy básica de un menú. Con este caso comprobamos específicamente el funcionamiento de la sentencia case, procedimientos y comentarios con paréntesis.

program acierto2Case;

procedure comenzarJuego(start: INTEGER);

begin

start:= not start;

end;

procedure reiniciarJuego(restart: INTEGER);

begin

restart:= not restart;

end;

procedure opciones(menuOpciones: INTEGER);

begin

menuOpciones:= not menuOpciones;

end;

var

start, restart, menuOpciones : INTEGER;

menu : CHARACTER;

begin

(\*se utilizan argumentos en forma de integers como booleans puesto que no estan definidos los booleans en la gramatica\*)

start:=0;

restart:=0;

menuOpciones:=0;

case menu of

play: comenzarJuego(start);

;

reset: reiniciarJuego(restart);

;

options: opciones(menuOpciones);

;

end

end.

**Erróneos**

* En este ejemplo se realiza un pequeño programa para comprobar el funcionamiento de las constantes literales y de los comentarios con llaves, además de esto se realiza un fallo con las funciones puesto que no es posible definir funciones sin argumentos con la gramática dada.

program falloArgumentos;

var

test: CHARACTER;

output: REAL;

begin

test:= 'nuevo ''char''';

{con la gramatica dada no es posible realizar funciones ni procedimientos sin argumentos}

output := argumentos();

end.

* Este caso realizamos un error común a la hora de programar como es la falta del punto y coma al final de una sentencia.

program falloPuntoComa;

function holaPersona ( persona : CHARACTER ) : CHARACTER;

begin

holaPersona := 'Hola especial'''+persona+''', te mandamos un saludo';

end;

var

holaOutput,nombre : CHARACTER;

begin

nombre:='Miguel';

{Olvido del punto y coma en una sentencia y fallo}

holaOutput:=holaPersona(nombre)

end.