# Memoria de la 1º entrega: Análisis léxico y sintáctico

Práctica obligatoria Procesadores de Lenguajes

04/04/2019

Universidad Rey Juan Carlos (Móstoles)

Sergio Lira Díaz, David López Vidales y Sara Rodríguez Alarcón

Universidad Rey Juan Carlos

# Contenido

# 1. Descripción del trabajo realizado

En esta entrega hemos realizado la parte obligatoria correspondiente al análisis léxico y sintáctico de la práctica, así como el tratamiento de sentencias de control de flujo, *if*, *while-do* y *repeat-until*, como la notificación y recuperación de errores, correspondiente a la parte opcional.

Hemos hecho una aplicación que analiza un código en Pascal y lo traduce a C gracias a las herramientas *JFLEX*, *CUP* y el *IDE Intellij*.

# 2. Temas a destacar

## 1. Dificultades encontradas

Durante la realización de la práctica hemos encontrado diferentes dificultades a la hora de integrar *CUP* en el *IDE Intellij*, ya que no existe como librería interna, por lo que hemos tenido que importar el *JAR*. Otro problema encontrado es que el *JAR* generado se guarda en una carpeta sin permisos de ejecución, así que hemos tenido que montar un *script* que copiase ese *JAR* en otra carpeta que sí tuviera permisos para poder ejecutarlo. También encontramos dificultades en la parte del analizador léxico porque en algunas ocasiones pusimos palabras reservadas después de la declaración de *identifier* y esto generaba conflictos, la solución de este último problema fue tan simple como cambiar el orden en las declaraciones. Con respecto al control de errores, tuvimos problemas ya que había demasiados errores y entraban en conflicto, es decir, al tratar los errores superponíamos unos consecuentes error sobre otros. Esto lo solucionamos eliminando los que sobraban.

## 2. Anotaciones

Hemos creado un *script* llamado "*execute*" que copia en otra carpeta el *JAR* que el proyecto genera y lo ejecuta con un caso de prueba. También hemos creado otro *script* para la parte del analizador sintáctico cuya función es ejecutar el archivo ".*cup*". Ambos *scripts* los hemos hecho para facilitarnos la realización y ejecución de la práctica.

Queremos destacar que el *JAR* que se solicita en el enunciado de la práctica se encuentra dentro de la carpeta de **pruebas**.

# 3. Casos de prueba

## 1. Casos de prueba correctos

```
1   program test1ok;
2
3
4   var contador: INTEGER;
5
6
7   begin
8
9   contador:= 0;
10
11  while contador < 10 do
12      begin
13      contador:= contador + 1;
14      end
15  end.
```

```
1   program test2ok;
2
3
4   var contador, resultado: INTEGER;
5
6
7   begin
8
9   contador:= 0;
10  resultado:= 0;
11
12  repeat
13      begin
14          contador:= contador + 1;
15          resultado:= resultado + ((contador * 4) div 2);
16      end
17  until contador >= 10;
18  end.
```

```
program test3ok;

const id = 1;

var aux, aux2: INTEGER;
var tet: REAL;

procedure proceso(p,q,r: INTEGER; s: REAL);
begin
    p := 3;
end;

function funcionP(p,q,r: INTEGER; s:REAL): INTEGER;
begin
    q := 2 + funcionP(2,3,4);
end;

begin

    id := 2 + (2 * 4);
end.
```

```
program test4ok;

const id = 100;

var aux, resultado: INTEGER;


begin

aux:= 50;
resultado:= 100;

if id = aux then
    begin
        resultado:= ((resultado div 50) * 30) - 500 + 10;
    end
else
    begin
        resultado:= 10101110;
    end
end.
```

## 2. Casos de prueba incorrectos

```
1   program test1bad;
2
3
4   var contador: INTEGER;
5
6
7   begin
8
9   contador:= 0;
10
11  while (contador >.< 10) do
12      begin
13      contador:= contador + 1;
14      end;
15
16
17  end.
```

En el programa "*test1bad*" los errores se encuentran en la línea 11 (ya que el *while* no debería llevar paréntesis y la condición booleana está mal formulada) y en la línea 14 (porque sobra un punto y coma después del *end*).

```
1   program test2bad;
2
3
4   var contador, resultado: INTEGER;
5
6
7   begin
8
9   contador:= 0;
10  resultado:= 0;
11
12  repeating
13
14      contador:= contador + 1;
15      resultado:= resultado + ((contador * 4) div 2);
16
17  until (contador >= 10);
18  end.
```

En el programa "*test2bad*" los errores se encuentran en la línea 12 (ya que la sentencia del *repeat* está mal formulada), en las líneas 13 y 16 (porque falta el *begin-end* dentro del bucle) y en la línea 17 (porque sobran los paréntesis del *until*).

```
1   program test3bad;
2
3   const id = 1;
4
5   var aux, aux2: INTEGER;
6   var tet: REAL;
7
8   procedure proceso(p,q,r: INTEGER; s: REAL);
9
10      p := 3;
11  end;
12
13  function funcionP(p,q,r: INTEGER; s:REAL): INTEGER;
14  begin
15      q := 2 + funcionP(2,3,4)
16
17
18  begin
19
20      id := 2 + (2 * 4);
21  end.
```

En el programa "*test3bad*" los errores se encuentran en la línea 9 (ya que falta un *begin*), en la línea 15 (porque falta un punto y coma al final de la sentencia) y en la línea 16 (porque falta un *end*).

```
1   program test4bad;
2
3   const id = 100;
4
5   var aux, resultado: INTEGER;
6
7
8   begin
9
10  aux:= 50;
11  resultado:= 100;
12
13  if (id = aux) then
14      resultado:= ((resultado div 50) * 30) - 500 + 10;
15  else
16      resultado:= 10101110;
17  end.
```

En el programa "*test4bad*" los errores se encuentran en la línea 13 (ya que el *if* no debería tener paréntesis) y en las líneas 14 y 16 (porque faltan los *begin-end* dentro del *if-else*).