# PRÁCTICA : ANALIZADOR LÉXICO, SINTÁCTICO Y TRADUCCIÓN DIRIGIDA POR LA SINTAXIS

Alejandro Molina López y David López Vidales

PROCESADORES DE LENGUAJES Grupo 22

# ANALIZADOR LÉXICO

Para realizar esta parte hemos utilizado la herramienta jflex. Generando desde practicaPL.flex el analizador léxico llamado AnalizadorLexico.java. En este hemos reconocido mediante diferentes expresiones regulares los tokens del lenguaje.
Debido a que los comentarios y los espacios, tabulaciones y saltos de línea no necesitan un token, hemos puesto que solo los reconozcan.
Todos los caracteres y/o cadenas de caracteres no recogidas en las expresiones regulares serán identificados como un error y se notificará especificando la fila y columna en las que se encuentra.

## Dificultades encontradas

Hemos tenido problemas a la hora de reconocer los tokens de las palabras reservadas, ya que nosotros los llamábamos literalmente como la palabra a la que representan y al hacer esto, java los confundía con sus palabras reservadas. Por ello posteriormente añadimos a los nombre el prefijo p_. Aun así nos daba problemas el token p_void así que lo nombramos finalmente p_v.

Los ejemplos que hemos empleado para comprobar que nuestro analizador léxico reconoce las distintas expresiones son:

### Ejemplos de caracteres reconocidos

1-

```
int variable = 3 ;
```

2-

```
if ( sumando >= 5 ) {
```

3-

```
// añadir nombre del usuario
nombre = 'Mario'
```

4-

```
x = z || x ;
```

**Ejemplos de caracteres no reconocidos**

1- No reconoce el caracter @ fuera de comentarios

```
@holaMundo
```

2- Falta un * tras /

```
/esto es un comentario mal hecho*/
```

3- No reconoce el caracter “_ “ fuera de comentarios

```
_Program
```

4- No reconoce ^ fuera de comentarios

```
^loadError
```

# ANALIZADOR SINTÁCTICO

Para realizar este apartado hemos tenido que importar la librería java_cup.runtime al .flex realizado en el apartado anterior. El archivo sobre el cual hemos realizado el programa CUP es practicaPL.cup. En él hemos introducido la gramática del enunciado. Debido a la sintaxis de CUP, hemos separado los no terminales y los terminales, para indicarle a CUP cuales son y a continuación la gramática. Debido a que CUP es un analizador LALR, hemos tenido que añadir encima de la gramática precedence left seguido de los no terminales que contenían OP y OPL. Esto es porque no era posible resolver una ambigüedad, ya que los distintos terminales no tenían un orden de prioridad y no sabe cuál elegir. Esta expresión por tanto prioriza los elementos de izquierda a derecha en orden de importancia. Tras acabar el .cup hemos creado el Analizador, y generado los ficheros sym.java y parser.java utilizando cup.

## Dificultades encontradas

El primer problema que nos apareció fue: Warning : *** Shift/Reduce conflict found
Este problema es debido a que nuestra gramática tiene recursividad izquierda y derecha, y cup es un analizador LALR. Para solucionarlo utilizamos la expresión anteriormente explicada precedence left.
Pero el mayor problema fue con las palabras reservadas, siendo el mismo que encontramos en el .flex. Debido a ello tuvimos que modificar el nombre de los terminales en el .cup también. Tuvimos un caso especial con la palabra reservada void, y tuvimos que cambiarla otra vez porque seguía reconociendo la palabra reservada el fichero java.

Los ejemplos utilizados son los siguientes:

## Ejemplos reconocidos

1- Entrada 1 correcta

```
void main(void){
   int v;
   funcion();
   v = 5 + 3 ;
   int e;
   e = v + PI ;
}

float funcion(float a){
   int v;
   v = 5 + 3 ;
   int e;
   e = v + PI ;
}
```

2- Entrada 2 correcta

```
void main(void){
   int v;
   v = 7;
   if(v==7){
         v =v/5;
   }else{
         v = 0;
   }
   funcion();
   v = 5 + 3 ;
   int e;
   e = v + PI ;
}
```

3- Entrada 3 correcta

```
void main(void){
   int v;
   v = 7;
   v =v/5;
   int a;
   do{
         a=0;
   }until(a==0)

}
```

4- Entrada 4 correcta

```
void main(void){
   int v;
   v = 7;
   v =v/5;
   int a;
   do{
         if(a==8){
               a=0;
         }else{
               a=8;
         }
   }until(a==0)
}
```

## Ejemplos no reconocidos

1- Entrada 1 incorrecta

Falta un ";" al final de la línea "int e".

```
void main(void){
   int v;
   funcion();
   v = 5 + 3 ;
   int e
   e = v + PI ;
}

float funcion(float a){
   int v;
   v = 5 + 3 ;
   int e;
   e = v + PI ;
}
```

2- Entrada 2 incorrecta

Falta un "{" después del else que indique que se inicia la condición.

```
void main(void){
   int v;
   v = 7;
   if(v==7){
         v =v/5;
   }else
         v = 0;
   }
   funcion();
   v = 5 + 3 ;
   int e;
   e = v + PI ;
}
```

3- Entrada 3 incorrecta

Falta un "(" después del main que indique que a continuación se van a pasar los parámetros del método.

```
void main void){
   int v;
   v = 7;
```

```
    v =v/5;
    int a;
    do{
          a=0;
    }until(a==0)
}
```

4- Entrada 4 incorrecta

Falta el "do" justo antes del segundo "{" que indica qué tipo de bucle se va a realizar.

```
void main(void){
    int v;
    v = 7;
    v =v/5;
    int a;
    {
          if(a==8){
                a=0;
          }else{
                a=8;
          }
    }until(a==0)
}
```

# TRADUCCIÓN DIRIGIDA POR LA SINTAXIS

Para la realización de este apartado, hemos tenido que añadir a cada elemento que lo necesitara un atributo sintetizado. Se ha seguido la gramática proporcionada por el enunciado para realizar la traducción. Debido a que la gramática de C y la de Pascal se diferencian, sobretodo en ciertos puntos esenciales como puede ser la declaración de variables, constantes, procedimientos y funciones, hemos tenido que realizar ciertos esquemas para poder aplicar una gramática dentro de otra. Hemos hecho uso de librerías como ArrayList, para organizar la traducción, además de ciertas clases que explicaremos a continuación:

-Token: esta clase se crea en el analizador léxico, y tiene el valor de los tokens, la línea y la columna donde se ha encontrado, para poder devolver un error si fuera necesario.
-Elemento: esta clase es el nivel más bajo de nuestra jerarquía de clases. Cada objeto de nuestro traductor es de la clase elemento. Puede ser desde un carácter, por ejemplo (, hasta el valor de un token recogido del léxico.
-Línea: esta clase está formada por un ArrayList<Elemento>, además representa una línea de código, excepto en casos especiales donde no debe haber salto de línea.
-Cabecera: esta clase se almacena la cabecera de una función.
-Cuerpo: en esta clase se almacena el cuerpo de una función, con todas las líneas y sus saltos de línea correspondientes.
-Variables: esta clase está formada por todas las variables declaradas de un Cuerpo.
-Función: esta clase es muy importante ya que un objeto de la clase función es un procedimiento o función, con su cabecera, su declaración de variables y su cuerpo.
-Constantes: esta clase es la encargada de las constantes del programa.
-Programa: la clase principal, está formada por un ArrayList<Funcion>, por una Función que será el programa principal, si lo tuviera, y por las constantes del programa. Es además la que guarda todo el string con la información en un fichero.

En la clase Analizador.java se ha añadido el código necesario para cambiar el flujo, así en vez de mostrar por consola, lo añade al fichero de salida que se llamará igual que el de entrada pero con formato .pas.

## Dificultades encontradas

La mayor dificultad ha sido reconocer las partes diferentes de ambas gramáticas y unirlas. Ya que hay ciertas partes que no han sido fáciles de traducir. Esto es debido a que la gramática de pascal es muy diferente a la de C. Por ejemplo la declaración de constantes en C se realiza al principio en DEFINES, pero en Pascal se realiza dentro de DCL en DEFCTE.
A parte de esto no hemos tenido mayores problemas, solo hemos tenido que aplicar la gramática de pascal y añadir los elementos cuando era necesario. En parte ha resultado más sencillo debido a las clases auxiliares utilizadas, ya que por ejemplo para el paso del nombre de la función al cuerpo del return, si no lo hubiéramos pasado a la clase nos hubiera costado mucho, ya que debería ser un atributo heredado y en CUP no hemos encontrado la forma de pasarlo como tal.

Un problema que nos surgió, fue que descubrimos que nuestro analizador léxico reconocía mal las constantes literales, así que tuvimos que arreglarlo ya que en la declaración de constantes:
#define cons1 ‘verdad’
#define cons2 ‘mentira’

decía que desde ‘verdad’ hasta ‘mentira’ todo era una constante literal, incluyendo el #define. Por ello tuvimos que volver a crear el analizador léxico con este problema arreglado.

Los ejemplos empleados para comprobar que el traductor funciona correctamente están contenidos en la carpeta entradas. Tras ejecutarlo aparecerá en la misma ruta de la entrada un fichero con el mismo nombre pero con extensión .pas. Por tanto al ejecutar el Analizador desde consola se debe poner:

java –jar Analizador.jar .\entradas\(nombre del fichero de entrada)

Los ejemplos utilizados se muestran en las páginas siguientes.

## Ejemplos reconocidos

1- Entrada1Correcta.c

```
#define operando1 700
#define operando2 33

int sumaLenta (int variable) {
	int contador;
	int solucion;
	solucion = variable;
	for(contador = 0 ; contador <= operando1 ; contador = contador + 1){
		solucion = solucion + 1;
	}
	return solucion;
}

int restaLenta (int minuendo, int sustraendo) {
	int contador;
	int solucion;
	solucion = minuendo;
	for(contador = 0 ; contador <= sustraendo ; contador = contador - 1) {
		solucion = solucion - 1;
	}
	return solucion;
}

int restaRara (int minuendo, int sustraendo, float real, int entero) {
	int contador;
	int solucion;
	solucion = minuendo;
	for(contador = 0 ; contador <= sustraendo ; contador = contador + contador * (2
- 1)) {
		solucion = solucion - 1;
	}
	return solucion;
}

void main (void){
	float numero;
	numero = 55.7;
	int entero;
	if (!operando2 == 37) {
		numero = 7 + 3 * numero;
	} else {
		entero = sumaLenta(operando1, 55);
	}
	entero = entero % numero;
}
```

2- Entrada2Correcta.c

```
#define centena 100
#define decena 10
#define falso 0
#define verdadero 1

void cambioValor (float uno, float dos) {
	float aux;
	if (uno > dos){
		uno = dos;
		dos = aux;
	} else {
		uno = dos;
	}
}

int acceso(void) {
	int sePuede;
	int edad;
	if (edad < 18) {
		sePuede = falso;
	}
	else{
		sePuede = verdadero;
	}
	return sePuede;
}

void main (void){
	int personas;
	personas = 5 * decena + centena;
	int mayoresPares;
	do {
		int aux;
		aux = falso;
		aux = aux + acceso(15);
		personas = personas - 1;
	} until (personas == 0)
	mayoresPares = aux;
}
```

3- Entrada3Correcta.c

```
#define falso 0
#define verdadero 1

void operacion (float uno, float dos) {
        float op1;
        float op2;
        int op3;
        int op4;
        uno = op1 + op3 / op4;
        dos = op4 * op4 * op2 - op3;
}

int esPar (int numero) {
        int aux;
        aux = numero % 2;
   if (aux == 0) {
     solucion = verdadero;
   }
   else
   {
     solucion = falso;
   }
   return solucion;
}

void main (void){
        float n1;
        float n2;
        int n3;
        int n4;
        n1 = 10;
        n2 = 29;
        n3 = 99;
        n4 = 13;
        int solucion;
        do {
                operacion(n1, n2);
                while ( 9 == 2 || n1 < n2) {
                        solucion = esPar(n3);
                        n3 = n3 + 1;
                        n1 = n1 + 1;
                }
        } until (n3 >= n4)
}
```

4- Entrada4Correcta.c

```
#define cien 100
#define cero 0

int calendario (void){
	int enero;
	int febrero;
	int julio;
	int septiembre;

	enero = 31;
	febrero = 28;
	julio = enero;
	septiembre = 30;

	if(anio == cien){
		enero = marzo * septiempre;
		febrero = febrero + 8;
		julio = 88.9;
	} else {
		septiembre = septiembre * 2.5;
	}
	return febrero;
}

float notas (int solucion) {
	int ais;
	int ia;
	int pl;
	int emp;
	int sd;
	do {
		ais = ia + emp / sd;
		pl = pl + 1;
	} until (pl == 10)
	solucion = pl;
	return solucion;
}
```

## Ejemplos no reconocidos

1- Entrada1Incorrecta.c

En la función “sumaLenta”no tenemos ningún return.

```
#define operando1 700
#define operando2 33

int sumaLenta (int variable) {
	int contador;
	int solucion;
	solucion = variable;
	for(contador = 0 ; contador <= operando1 ; contador = contador + 1){
		solucion = solucion + 1;
	}
	// no tiene return
}

int restaLenta (int minuendo, int sustraendo) {
	int contador;
	int solucion;
	solucion = minuendo;
	for(contador = 0 ; contador <= sustraendo ; contador = contador + 1) {
		solucion = solucion - 1;
	}
	return solucion;
}

void main (void){
	float numero;
	numero = 55.7;
	int entero;
	if (operando2 == 37) {
		numero = 7 + 3 * numero;
	} else {
		entero = sumaLenta(operando1, 55);
	}
	entero = entero % numero;
}
```

2- Entrada2Incorrecta.c

En el segundo método al pasar los parámetros hacemos una suma.

```
#define centena 100
#define decena 10
#define falso 0
#define verdadero 1

void cambioValor (float uno, float dos) {

   if (uno > dos){
      uno = dos;
      dos = aux;
   } else {
      uno = dos;
   }
}

int acceso(int edad + 55) {     //se hace una suma al pasarle los parámetros
   int sePuede;
   if (edad < 18) {
      sePuede = falso;
   }
   else{
      sePuede = verdadero;
   }
   return sePuede;
}

void main (void){
   int personas;
   personas = 5 * decena + centena;
   int mayoresPares;
   do {
      int aux;
      aux = falso;
      aux = aux + acceso(15);
      personas = personas - 1;
   } until (personas = 0)
   mayoresPares = aux;
}
```

3- Entrada3Incorrecta.c

Nos falta el “until” del último bucle “do”.

```
#define falso 0
#define verdadero 1

int esPar (int numero) {
   int aux;
   aux = numero % 2;
   if (aux == 0) {
     solucion = verdadero;
   }
   else
   {
     solucion = falso;
   }
   return aux;
}

void operacion (float uno, float dos) {
   float op1;
   float op2;
   int op3;
   int op4;
   uno = op1 + op3 / op4;
   dos = op4 * op4 * op2 - op3;
}

void main (void){
   float n1;
   float n2;
   int n3;
   int n4;
   n1 = 10;
   n2 = 29;
   n3 = 99;
   n4 = 13;
   int solucion;
   do {
      operacion(n1, n2);
      while ( 9 == 2 || n1 < n2) {
         solucion = esPar(n3);
         n3 = n3 + 1;
         n1 = n1 + 1;
      }
   }      // no hay until
}
```

4- Entrada4Incorrecta.c

Tenemos 2 métodos “main”.

```
#define cien 100
#define cero 0

void main (void){ // primer main
   float numero;
   numero = 55.7;
}

int calendario (void){
   int enero;
   int febrero;
   int julio;
   int septiembre;
   enero = 31;
   febrero = 28;
   julio = enero;
   septiembre = 30;

   if(anio == cien){
      enero = marzo * septiempre;
      febrero = febrero + 8;
      julio = 88.9;
   } else {
      septiembre = septiembre * 2.5;
   }
   return febrero;
}

float notas (int solucion) {
   int ais;
   int ia;
   int pl;
   int emp;
   int sd;
   do{
      ais = ia + emp / sd;
      pl = pl + 1;
   } until (pl == 10)
   solucion = pl;
   return solucion;
}

void main (void){       //segundo main
   int numero;
   numero = 106.8;
}
```