Práctica 2: Construcción de un analizador sintáctico para *adac*

Procesadores de lenguajes

Dpto. de Informática e Ingeniería de Sistemas,
Grado de Ingeniería Informática
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

1. Objetivos

Los objetivos para esta práctica son:

- Desarrollar un analizador sintáctico descendente recursivo para *adac*
- Integrar el análisis léxico con el análisis sintáctico
- Entender los requisitos para desarrollar un analizador descendente predictivo

Nota: Esta práctica está prevista para ser desarrollada durante dos sesiones de laboratorio: las sesiones 2 y 3.

2. Un ejemplo de analizador sintáctico con *javacc*

En moodle, en la sección correspondiente a la práctica 2, podéis encontrar un ejemplo de analizador sintáctico para una calculadora de enteros, que permite declarar constantes y variables. Se encuentra en el fichero `calc_enteros_sint.zip`. Para entender la forma en que se han de describir las producciones en *javacc*, deberéis leer documentación relacionada con la herramienta. El objetivo es que sirva de ejemplo para facilitar el inicio del trabajo a desarrollar, que se detalla a continuación.

3. Descripción del trabajo a realizar

En esta práctica se debe desarrollar el analizador sintáctico para *adac*. Al final del documento se muestra una posible gramática (incompleta) para el lenguaje. Se muestra completa la parte correspondiente a las expresiones, pues, usualmente, es una de las partes más complicadas de una gramática para un lenguaje de programación. Como se ve, y por simplificar la creación del compilador, la gramática hace uso de la clausula *LOOKAHEAD* en algunas de las producciones. Esto hace que el analizador sintáctico generado por *javacc* mire más de un símbolo de preanálisis antes de tomar la decisión de qué producción disparar. Nótese que, tal y como se ha escrito la gramática, solo se hace uso de dos símbolos de preanálisis, como mucho. Es importante recordar que la transformación en una gramática LL(1) es muy sencillo, si así fuera necesario. Así, las producciones

```
variable :
  LOOKAHEAD(2)
  <tID> <tACOR> <tCONST_INT> <tCCOR>
| <tID>
```

se podrían transformar en LL(1) como sigue:

```
variable : <tID> mas_variable

mas_variable :
  <tACOR> <tCONST_INT> <tCCOR>
| //corresponde a epsilon
```

Nótese también que estamos usando notación EBNF que, igualmente, puede transformarse en LL(1) si fuera necesario. Por ejemplo:

```
lista_cero_o_mas_exps: ( lista_una_o_mas_exps )?
```

se escribiría como

```
lista_cero_o_mas_exps :
   lista_una_o_mas_exps
| //corresponde a epsilon
```

4. Errores sintácticos

En caso de error sintáctico, el analizador deberá llevar a cabo una recuperación en modo pánico. Para ello:

- Informará del error, mostrando el token que lo ha generado y su posición en el fuente

- Aplicará recuperación en modo pánico, saltando tokens hasta llegar a uno de sincronización. Como mínimo, se hará recuperación a nivel de instrucción/bloque: se saltarán tokens hasta llegar a un punto y coma o un `end`.

5. Entrega de resultados de la práctica

5.1. Lo que hay que entregar

Como resultado de la práctica se deberá entregar:

- El fichero `practica_2.zip`. Este, una vez descomprimido, tendrá la misma estructura que en la práctica 1. El fichero `README.txt`, del directorio `Doc`, deberá incluir información de cuáles han sido las políticas de recuperación de errores sintácticos aplicadas.
- El fichero `tests.zip`. Este, una vez descomprimido, deberá contener los cinco programas en *adac* que se nombraban en el enunciado de la práctica 1. Los fuentes se deben llamar `test_1.adac`, `test_2.adac`, ..., `test_5.adac`

5.2. Método de entrega

La entrega se debe hacer en *hendrix* mediante la ejecución del programa `someter`, análogamente a como se hizo en la práctica anterior

5.3. Plazos de entrega

El plazo de entrega es de dos semanas, a contar desde el día de la realización de la práctica. Quien tenga la sesión en martes, deberá entregarla antes del segundo martes después de su realización. Análogamente para el resto de días de prácticas.

Gramática parcial para *adac*

Lo que sigue es una posible gramática (parcial) para *adac*, con el objetivo de que sirva de ayuda para el desarrolo de la práctica. No es necesario utilizarla. Cada pareja puede desarrollar una correcta completamente diferente, o adaptar esta según sus necesidades. Para enteder qué representa cada token o cada producción, hay que mirar los programas entregados en la batería de test suministrada.

```
1  programa :
2    <tPROCEDURE>
3    <tID>
4    <tIS>
5    declaracion_variables
6    declaracion_procs_funcs
7    bloque_sentencias
8
9  declaracion_variables :
10   ( declaracion <tPC> )*
11
12 declaracion :
13   tipo_variable lista_vars
14
15 tipo_variable :
16   <tINT> | <tCHAR> | <tBOOL>
17
18 lista_vars :
19   variable ( <tC> variable )*
20
21 variable :
22   LOOKAHEAD(2)
23   <tID> <tACOR> <tCONST_INT> <tCCOR>
24 | <tID>
25
26 declaracion_procs_funcs :
27   ( declaracion_proc_func )*
28
29 ...
30
31 declaracion_funcion :
32   cabecera_funcion
33   declaracion_variables
34   bloque_sentencias
35
36 ...
37
38 dec_parametros :
39   clase_parametros
40   tipo_variable
41   id_or_array ( <tC> id_or_array )*
```

```
42
43 id_or_array :
44   LOOKAHEAD(2)
45   <tID> <tACOR> <tCONST_INT> <tCCOR>
46 | <tID>
47
48 ...
49
50 instruccion :
51   inst_leer <tPC>
52 | inst_saltar_linea <tPC>
53 | inst_escribir <tPC>
54 | inst_escribir_linea <tPC>
55 | LOOKAHEAD(2)
56     inst_invoc_proc <tPC>
57 | inst_asignacion <tPC>
58 | inst_seleccion
59 | inst_iteracion
60 | inst_return <tPC>
61
62 inst_asignacion :
63   asignable
64   <tASSIGN>
65   expresion
66
67 ...
68
69 inst_iteracion :
70   <tWHILE>
71   expresion
72   <tDO>
73   lista_instrucciones
74   <tEND>
75
76 ...
77
78 lista_cero_o_mas_exps:
79   ( lista_una_o_mas_exps )?
80
81 ...
82
83 expresion :
84   expresion_simple ( operador_relacional expresion_simple )?
85
86 operador_relacional :
87   <tIG>
88 | <tMEN>
89 | <tMAY>
90 | <tMENI>
```

```
91  | <tMAYI>
92  | <tDIF>
93
94  expresion_simple :
95    ( <tMAS> | <tMENOS> )?
96    termino ( operador_aditivo termino )*
97
98  operador_aditivo :
99    <tMAS>
100 | <tMENOS>
101 | <tOR>
102
103 termino :
104   factor ( operador_multiplicativo factor )*
105
106 operador_multiplicativo :
107   <tPROD>
108 | <tMOD>
109 | <tDIV>
110 | <tAND>
111
112 factor :
113   <tNOT> factor
114 | <tAPAR> expresion <tCPAR>
115 | <tINT2CHAR> <tAPAR> expresion <tCPAR>
116 | <tCHAR2INT> <tAPAR> expresion <tCPAR>
117 | LOOKAHEAD(2) <tID> <tAPAR> lista_cero_o_mas_exps <tCPAR>
118 | LOOKAHEAD(2) <tID> <tACOR> expresion <tCCOR>
119 | <tID>
120 | <tCONST_INT>
121 | <tCONST_CHAR>
122 | <tCONST_STRING>
123 | <tTRUE>
124 | <tFALSE>
```