UNIVERSIDAD SAN JORGE

Escuela de Ingeniería Grado en Ingeniería Informática

ANÁLISIS LÉXICO Y SINTÁCTICO

Examen final

Curso/grupo:	2º INF	**Curso académico:**	2010/2011
Fecha examen:	15/06/2010	**Duración examen:**	2h 30 minutos
Convocatoria:	Julio	**Material permitido:**	-
Instrucciones:	Las preguntas no requieren respuestas especialmente extensas, se valorará la claridad en las explicaciones y las respuestas concretas El examen se debe realizar en las hojas correspondientes.		

Nombre:_____ DNI:_____

Pregunta 1 (1 punto)

Define los siguientes conceptos:

a) Compilador (0.25 ptos.)
b) JavaCC (0.25 ptos.)
c) Lookahead (0.25 ptos.)
d) Intérprete (0.25 ptos.)

Pregunta 2 (1.5 puntos)

Dibuja una figura en la que se indiquen las distintas fases de en las que se divide el proceso de compilación y otros elementos importantes que influyan en el mismo. Describe qué funciones se realizan en cada una de las fases, sus inputs y outputs y en qué consisten los elementos nombrados.

Pregunta 3 (0.5 puntos)

¿Qué formas conoces de implementar Analizadores Léxicos? ¿En qué se diferencian? ¿Cuál crees que produce scanners más eficientes? ¿Por qué?

Pregunta 4 (1 punto)

¿Qué reglas de desambiguación se aplican a la hora de reconocer tokens? (0.5 ptos.) Pon un ejemplo de cada una. (0.5 ptos.)

Pregunta 5 (2 puntos)

Las siguientes producciones forman una gramática:

L → L **a** S | **c**
S → **b** | **ε**

¿Es LL1? ¿Por qué? (0.25 ptos.)
En caso negativo, obtened su gramática LL1 equivalente. (0.5 ptos.)
Obtened su tabla de parseo LL(1) (0.5 ptos.).
Indicad si se puede reconocer o no la siguiente cadena "**c a a b a b $**" (0.5 ptos), si se puede, dibujad el árbol de parseo LL(1) (0.25 ptos.).

Pregunta 6 **(1.5 puntos)**

Dada la siguiente gramática LR(1):

1) S → **a** L
2) S → **c**
3) L → L X **b**
4) L → **ε**
5) X → **g** X
6) X → **ε**

Calcular la colección canónica LR(1), y el diagrama de transiciones asociado (DFA). (0.5 ptos.)
A partir de estos elementos, obtener la tabla para el parser LR(1). (0.5 ptos.)
Con la tabla calculada, detalla el proceso de reconocimiento de la cadena de entrada "**agbgb**". (0.5 ptos.)

Pregunta 7 **(2.5 puntos)**

La LFP ofrece en su web el resumen de las jornadas de fútbol de las categorías del fútbol español (1ª, 2ª, 2ªB, 3ª) La información que se muestra se recibe automáticamente de un servidor de una empresa externa (DatosDeFútbol S.L.) con una estructura muy concreta (descrita debajo), pero en ocasiones, los becarios de esta empresa no ponen demasiado empeño en revisar la información que suben a su servidor, lo que ocasiona que la web de la LFP no muestre ningún tipo de información hasta que el responsable de la misma se da cuenta y tiene que revisar línea por línea el texto, hasta que da con el error y lo corrige.

Un ejemplo de un posible fichero esperado válido es:

```
LIGA 1 - JORNADA 34

30/05/2011 18:00 RMA 2 : 3 ZAR
Ramos 62', Benzema 85' : Lafita 41', Gabi 54', Lafita 77'
67000 espectadores

30/05/2011 22:00 DEP 0 : 1 ATM
: Agüero 80'
34000 espectadores

30/05/2011 20:00 LEV 0 : 0 SPO
20000 espectadores

01/06/2011 17:00 RSO : FCB
Sin Comenzar

...
```

Así, describiendo el fichero, tenemos un encabezado, compuesto del campeonato ("LIGA 1"), un guión ("-") y la jornada ("JORNADA 34").

- Campeonato: Se compone de la palabra LIGA en mayúsculas y después una de las siguientes opciones: 1, 2, 2B o 3.
- Jornada: Se compone de la palabra JORNADA, seguido de una cifra de máximo 2 dígitos (1 a 38).

A continuación del encabezado, y separado por al menos 1 salto de línea tendremos una serie de partidos (de 1 a 10), con varios campos, unos obligatorios y otros opcionales:

- Fecha: Por ej. 30/05/2011, sería válido 1/1/2011 y 01/01/2011, pero no 001/001/2011.
- Hora: Por ej. 18:00, al igual que en la fecha, sería válido 1:15, y 01:15.

- Identificativo de equipo 1: Será siempre una combinación de 3 letras mayúsculas.
- Marcador equipo 1: Es opcional, ya que sólo aparece si se ha disputado el partido.
- Dos puntos: Separa los nombres de ambos equipos.
- Marcador equipo 2: Es opcional, ya que sólo aparece si se ha disputado el partido.
- Identificativo de equipo 2.

Por último, en caso de que el partido se haya disputado, tendremos una línea que indica el público asistente al estadio:

- Cifra entre 0 y 100000.
- Palabra "espectadores".

O en caso de que no se haya disputado, aparecerá una línea con el texto "Sin comenzar".

En caso de que el resultado sea distinto de 0:0, es decir, haya habido goles y el partido se haya disputado, tendremos una línea indicando los goleadores:

- Nombre de goleador: Es una combinación de letras (al menos una letra), que empieza por letra mayúscula y le siguen minúsculas.
- Minuto del gol: Cifra de 0 a 95.
- Apóstrofe o signo de minuto.
- Coma: Para separar goleadores en caso de que haya más de uno.
- Dos puntos: Para separar goles de uno u otro equipo (siempre aparece en caso de que uno de los dos equipos haya marcado).

* En la práctica no se pide saber si se ha disputado el partido o no, o controlar si una fecha es posterior o anterior, o la ordenación de los partidos, etc.

* Tampoco se pide controlar que el número de goles o goleadores sea igual al marcador de ese equipo en concreto.

* Ante cualquier duda, preguntar primero al profesor.

La LFP ha encargado a nuestra empresa que creemos un módulo que analice léxica y sintácticamente los datos recibidos, para que automáticamente acepte la información o la rechace, de forma que el servidor de DatosDeFútbol haga saltar una alarma y los becarios corrijan la información enviada.

En la siguiente página se incluye código en .jj que puede resultar útil. No hace falta copiar el código superior (options, PARSER_BEGIN … PARSER_END, etc.) a no ser que se considere necesario.

Se valorará positivamente las soluciones más óptimas, y que el scanner y parser generados, sean restrictivos con las cadenas de entrada que no cumplan las reglas (es decir, que no haya falsos positivos).

```
/**
 * JavaCC template file created by SF JavaCC plugin 1.5.17+ wizard for
JavaCC 1.5.0+
 */
options
{
  JDK_VERSION = "1.5";
  static = true;
}

PARSER_BEGIN(JSU_parser)
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;

public class JSU_parser {
      public static void main (String args [])
            throws ParseException {
            JSU_parser anLexSint = new JSU_parser (System.in);
            anLexSint.Input();
            System.out.println("Analisis terminado:");
            System.out.println ("no se han hallado errores léxico-
sintácticos");
      }
}
PARSER_END(JSU_parser)

TOKEN_MGR_DECLS : {
}

SKIP :
{
}

TOKEN :
{
}

void Input() :
{}
{
}
```