BOLETÍN PRÁCTICA 3
JAVA CUP: HERRAMIENTA DE CONSTRUCCIÓN DE UN ANALIZADOR SINTÁCTICO (FASE 1)

OBJETIVOS

Tras la realización de este boletín, y del estudio de la parte correspondiente de la asignatura, deberás ser capaz de:

- Utilizar java Cup para construir un analizador sintáctico que reconozca las estructuras del lenguaje, e informe de los errores cuando aparezcan construcciones sintácticas incorrectas.

TEMÁTICA

Recordamos que la fase de análisis sintáctico consiste en convertir el código fuente en otras estructuras (comúnmente árboles) que faciliten el análisis y construcción del código objeto. A partir de la salida producida por el analizador léxico (tokens), el analizador sintáctico comprueba que las tokens forman construcciones válidas.

En esta práctica, comenzaremos la construcción de un analizador sintáctico. En esta primera fase, simplemente comprobaremos que los símbolos que aparecen en la expresiones ya contienen un valor, e informaremos de construcciones incorrectas que pudieran aparecer en el código fuente. Para ello, utilizaremos el lenguaje de ejemplo introducido en la práctica 2, y el analizador léxico que fue construido al efecto.

TRABAJO PREVIO

- Lectura del manual de Java Cup: http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html
- Diseño de las estrategias que se utilizarán para resolver el problema.

La sesión de prácticas se dedicará a implementar el enfoque, y no a diseñar las estrategias necesarias para la resolución del ejercicio. Asimismo, para evitar pérdidas de tiempo debidas al uso de la herramienta, será necesario haber practicado la integración JFlex - Java Cup fuera del aula. Para ello, además de los contenidos vistos en clase, existen algunos enlaces interesantes, como http://crysol.org/es/node/819

ENUNCIADO

Dado el lenguaje de programación presentado en la práctica anterior y a partir del analizador léxico producido, construye un analizador sintáctico que informe sobre los siguientes errores:

- Uso de un símbolo cuyo valor no ha sido definido en el lado derecho de una expresión de asignación. Habrá que informar sobre el número de línea en el que aparece.
- Construcciones incorrectas

En esta fase, no es necesario que generes código objeto.

Puedes partir del siguiente código:

```
package MIPS;
import java_cup.runtime.*;
import java.io.*;
import java.util.*;
```

```
action code {:


:};

parser code {:

public static void main(String[] arg){
    if (arg.length!=1)
      System.out.println("Debe dar el nombre del fichero de entrada como parámetro");
    else
    {
         String fileName=arg[0];
         LexicalAnalyzerMIPS miAnalizadorLexico=null;
         parser parserObj=null;
         Symbol x=null;

         try{
           miAnalizadorLexico =
             new LexicalAnalyzerMIPS(new InputStreamReader(new FileInputStream(fileName)));

         } catch (IOException e){
           System.err.println("Fichero de entrada "+fileName+" no encontrado");
           System.exit(0);
         }
         try{
            parserObj = new parser(miAnalizadorLexico);
            x=parserObj.parse();
         } catch (Exception e) {
            System.err.println("No se pudo inicializar el compilador");
            System.exit(0);
         }
      } // end else
}
:};


terminal LEER, …;

precedence left …;
precedence left …;
precedence left …;

/* Gramática */
```

Para realizar este ejercicio, deberás definir una estructura de datos adecuada para saber qué símbolos han sido declarados y cuáles no. Considera el uso de un vector o de una tabla hash (java.util.Vector o java.util.Hashtable).