Prueba de Evaluación Continua de Laboratorio 2 (PECL 2)

Enunciado

En esta **segunda PECL** vamos a implementar, partiendo de la experiencia adquirida con scanner creado en la primera PECL, un ANALIZADOR SINTÁCTICO para los ficheros "**.upd**".

La especificación del fichero se ampliara en:

Los archivos UPD-XML constan obligatoriamente de las tres (3) siguientes secciones, que aparecen en el orden indicado y están delimitadas por las marcas **<upd>...</upd>**:

1. **<descr>...</descr>**: Contiene atributos que identifican el microprocesador que describe.

Estos pueden ser:

a. **<ident>... </ident>**: Nombre identificativo del procesador, formado por cualquier secuencia de caracteres(letras y/o números) que se inicie con una letra y que no incluya ningún blanco, ni tabulador, ni espacio, ni salto de línea.

b. **<date>... <\date>**: Fecha en formato día (dos dígitos)/mes (dos dígitos)/año (cuatro dígitos) en que el modelo del microprocesador se puso a la venta.
Es necesario validar la corrección de la fecha (31/01/2011 se considera válido y 15/15/2011, se considerara no válido).

El atributo **ident** es obligatorio, mientras que **date** es opcional.

2. **<storage>...</storage>**: Define los componentes de almacenamiento del microprocesador, pudiendo ser registros solitarios o agrupaciones de estos:

a. **<register>…</register>**: Define las características de un registro:

- **<name>... </name>**: Nombre identificativo de un registro, con una longitud máxima de dos caracteres, y delimitado por comillas simples.
- **<use>... </use>**: Lista, separada por comas, de cualquiera de las siguientes palabras: ***AllPurpose***, ***Accumulator***, ***ProgramPC***, ***Index***, ***FlagVector*** o ***StackPointer***, delimitadas por comillas dobles, que describe los usos posibles de un determinado registro.
- **<bitSize>...</bitSize>**: Número entero, representado por cualquier secuencia de uno o más dígitos que define el número de bits que puede contener un registro.
- **<regbitcode>...</regbitcode>**: Número binario, representado por cualquier secuencia de ceros y unos, y finalizado por el carácter 'b', que define el código que se inserta en una instrucción de direccionamiento inmediato que lo use. Este atributo es opcional.

Prueba de Evaluación Continua de Laboratorio 2 (PECL 2)

b. **<group>...</group>** : Tiene los mismos atributos que un registro y, además:

- **<registers>...</registers>:** Listado, separado por comas, de los nombres de los registros que se agrupan. Una agrupación tiene un número mínimo de *dos* registros y, en este ejercicio, no se considerará limitado en número máximo de registros.

3. **<insns>...</ insns>:** En esta sección se define el lenguaje ensamblador del microprocesador.

Cada instrucción del lenguaje se delimita por las marcas **<ins>** e **</ins>** y en su interior se definen:

a. **<opcode>...</opcode>**: Texto de la instrucción. Por ejemplo: ADD, LD, etc.

b. **<in>...</in>**: Lista de los operandos que pueden utilizarse en la operación, delimitados por corchetes y separados por comas. Los operandos pueden ser códigos de registro, o las palabras "**DATA**" o "**MEM**".

Por ejemplo, si la instrucción ADD exige que un operando sea siempre el acumulador, se tendría: <in>[A,BC], [A,B], [A,C], [A,DATA]</in>.Este atributo es opcional.

c. **<out>...</out>**: Opcionalmente, identificación del registro destino en que se almacena el resultado de la instrucción.

Por ejemplo: [A]

d. **<insbitcode>...</insbitcode>**: Lista de máscaras de bits para la definición del código binario de la instrucción, tal como se define en la PECL1.

e. **<behav>...</behav>**: Lista de textos explicativos de la instrucción, delimitados por corchetes.

Por ejemplo: [A=A+BC], [A=A+B], [A=A+C], [A=A+DATA]. Puede considerarse que el texto entre los corchetes puede ser cualquier texto, salvo saltos de línea.

Se pide

Utilizando `JLex` y `CUP`, se deberá implementar un ANALIZADOR LÉXICO y SINTÁCTICO que verifique la corrección de los ficheros `.upd` y almacene los datos relevantes del lenguaje descrito en una estructura de datos interna (insClass y storeClass)* .

Prueba de Evaluación Continua de Laboratorio 2 (PECL 2)

Para ello, deberá:

1. Crear el analizador sintáctico del lenguaje.

2. El analizador léxico se llamará ***UpdComp***.

3. Modifique el Analizador Léxico de la primera PECL para que sea compatible con el Analizador Sintáctico que se realizará. Si se precisa, modifique las clases creadas para almacenar la información analizada.

4. El analizador sintáctico deberá:

- Informar de los errores que encuentre en el código fuente y, en la medida de lo posible, recuperarse de éstos.
- Mostrar en pantalla, al finalizar su análisis, los datos almacenados internamente.

NOTA:

Respecto a esta solución, sólo cabe comentar lo siguiente:

a. El analizador léxico no introduce los símbolos en la tabla de símbolos, si no que se deja esta tarea para el analizador sintáctico.

b. Los testigos correspondientes al nombre de los registros, códigos binarios, fecha, texto y valores enteros incluyen el lexema en el símbolo (el último como un entero) para que el analizador sintáctico pueda incluir los valores en la tabla de símbolos.

c. Impresión del resultado. Se realiza con la llamada al método .toString().

d. Tratamiento de errores. Se realiza mediante la inclusión de la regla ERROR, a nivel de cada uno de los tres bloques principales de la gramática (descripción, almacenaje e instrucciones).

Prueba de Evaluación Continua de Laboratorio 2 (PECL 2)

```
class storeClass {
      private String regName;
      private boolean group;
      private Integer bitSize;
      private String bitCode;
                        // in java.util.Stack and
      public Stack use; // represents a last-in-first-out (LIFO) stack

      public Stack registers; //registers in a group

      storeClass (String r, Stack u, Integer i) {
            regName = r;
            bitSize =i;
            use = u;
      }
      storeClass (String r, Stack s, Integer i, String bc) {
            this(r,s,i);
            bitCode = bc;
      }
      storeClass (String r, Integer i, String bc, Stack s, Stack n) {
            this (r, s, i, bc);
            registers =n;
      }
      public void setRegName(String n) {
            regName = new String(n);
      }
      public void setGroup() {
            group = true;
      }
      public void setRegister() {
            group = false;
      }
      public void setBitSize(Integer i) {
            bitSize = new Integer(i);
      }
      public void setBitCode(String b) {
            bitCode = new String(b);
      }
      public String getRegName () {
            return regName;
      }

      public boolean isGroup (){
            return group;
      }

      public String toString() {
             String s = "Register: " +regName +
                  "\n\tUse: "+use.toString() +
                  "\n\tBisize: "+bitSize +
                  "\n\tBinary code: " + bitCode ;
            if (group)
                  s= s+"\n\tRegisters grouped: " +
                    registers.toString();
            s=s +"\n";
            return s;
      }
```

Prueba de Evaluación Continua de Laboratorio 2 (PECL 2)

```
class insClass {
      private String opCode;

      private Stack behaviour;
      private Stack binaryMasks;

      insClass(String r,Stack tempUseStack, Stack t)
      {
            opCode = r;
            binaryMasks  = tempUseStack;
            behaviour = t;
      }

      insClass(String r, Stack t)
      {
            opCode = r;
            behaviour = t;
      }

      public void setOpCode(String tempUseStack) {
            opCode = new String(tempUseStack);
      }

      public String getOpCode ()
      {
            return opCode;
      }

      public String toString()
      {
            return "Instruction: " + opCode +
                  "\n\tDescription: "+ behaviour.toString() +
                  "\n\tBinary masks: " + binaryMasks.toString() +
                  "\n";
      }
}
```