Procesadores de Lenguaje. Septiembre 2008

Ejercicio 1 [3 puntos]
a) **[1 punto]** Acondiciona la siguiente gramática de atributos para permitir una implementación descendente predictiva

```
Ds ::= Ds ; D
  Ds0.ts = añade(Ds1.ts,D.e)
Ds ::= D
  Ds.ts = añade(nuevaTabla(),D.e)
D  ::= id:t
  D.e = nuevaEntrada(id.lex,t.lex)
D  ::= id
  D.e = nuevaEntrada(id.lex,'?')
```

b) **[1 punto]** Demuestra que la siguiente gramática es LL(1)

```
A ::= E D
B ::= a
B ::= b A c
C ::= d E
C ::= λ
D ::= e E D
D ::= λ
E ::= B C
```

b) **[1 punto]** Demuestra que la siguiente gramática es LR(1), pero no LALR(1). ¿Es ésta una gramática LR(0)?

```
A ::= a B c | a C d | b B d | b C c
B ::= e
C ::= e
```

Ejercicio 2 [3 puntos]
Se considera una instrucción de iteración con el siguiente formato:

hasta que *Var* **se anule hacer**
  se pasa => I_o
  no llega => I_1
  reduce *Var*
fin hasta

donde *Var* es una variable entera, e I_o e I_1 son instrucciones. Su semántica operacional informal es como sigue:

1. Si *Var* = 0, terminar la iteración
2. Si *Var* > 0, ejecutar I_o
3. Si *Var* < 0, ejecutar I_1
4. Si *Var* > 0, *Var* ← *Var* -1
5. Si *Var* < 0, *Var* ← *Var* +1
6. ir a 1

Se pide:
a) **[1 punto]** Traduce a código-p las siguientes instrucciones comentando el código generado:

```
j:= 100;
i:= -7 ;
hasta que i se anule hacer
  se pasa => j := j - i
  no llega => j := j + i
  reduce i
fin hasta;
```

Para traducir considera que las variables `i y j` han sido declaradas en el programa principal, al que también pertenece este fragmento de código, y que se utiliza un modelo de memoria que asigna direcciones absolutas a las variables. *Utiliza las instrucciones apila-dir x , desapila-dir x, donde el argumento x es una dirección absoluta de la memoria de datos.*

b) **[2 puntos]** Formaliza mediante una gramática de atributos la traducción de esta instrucción al lenguaje de la máquina P sin etiquetas simbólicas. Haz primero un esquema que muestre cómo se organiza la traducción.

(* CONTINUA EN EL REVERSO DE LA HOJA *)

Ejercicio 3 [4 puntos] Considera el siguiente programa:

```
program examen;
 type
    tArrayLista=array [5] of ^integer;
    tLista=record
               elementos: tArrayLista;
               ocupacion: integer;
           end;
 var
    lista: tLista;

    procedure rellena(nelementos: integer);
    var
      i: integer;
      procedure rellenaPosicion(posicion: integer; peso: integer);
                                                    (*1*)
      begin
        new(lista.elementos[posicion]);
        lista.elementos[posicion]^ := peso*lista.elementos[posicion-1]^ (*2*)
      end; (* de rellenaPosicion *)
    begin
      new(lista.elementos[0]);
      lista.elementos[0]^ := nelementos;
      for i:=1 to nelementos-1 do
         rellenaPosicion(i,lista.elementos[0]^); (*3*)
      lista.ocupacion := nelementos
                                                    (*4*)
    end; (* de rellena *)

begin
   rellena(3)
end.  (* de examen *)
```

Con formato: Inglés (Reino Unido)

Suponiendo que tu traductor es un traductor descendente predictivo, que no realiza ningún tipo de optimización y que el código se ejecuta en una **máquina P con *display***, se pide:

a) **[0,5 puntos]** Describe el contenido de la tabla de símbolos en el punto **(*1*)**

b) **[0,5 puntos]** Haz un esquema de la memoria de la máquina P cuando la ejecución alcanza el punto **(*4*)**. Explica la finalidad de cada segmento de memoria, y da todos los detalles que puedas sobre el contenido de cada celda.

c) **[1 punto]** Representa mediante árboles la estructura sintáctica de las sentencias **(*2*)** y **(*3*)**. Marca sobre estos árboles el recorrido que realiza el traductor.

d) **[2 puntos]** Escribe y comenta el código-p que resulta de la traducción de las dos sentencias del apartado anterior, indicando claramente el propósito de los argumentos de cada instrucción en el código-p generado (¿es el argumento una dirección? ¿Es un nivel? ¿Es el contenido de una posición de memoria?, etc.). Debes indicar, así mismo, en qué puntos del recorrido de los árboles se genera cada instrucción.