**EXAMEN DE COMPILADORES (2° Grado en Informática, final junio-2012)**

FACULTAD DE INFORMÁTICA

**Apellidos, nombre:** **DNI:**
**GRUPO:**

**Instrucciones:** Este enunciado y todos los folios usados deben entregarse al salir

## Parte I: PREGUNTAS TIPO TEST. 30 %.

*Cada dos respuestas incorrectas anulan una correcta.*

1. Podemos asegurar que un lenguaje de programación en el que se requiera la declaración de variables previa a su uso es:
   *a*) *libre de contexto.*
   *b*) *sensible al contexto.*
   *c*) de ambos tipos.
2. Una *máquina abstracta*:
   *a*) traduce el código intermedio a código máquina, que posteriormente será ejecutado.
   *b*) es un intérprete para un lenguaje de alto nivel.
   *c*) puede considerarse como la implementación software de una máquina.
3. Podemos afirmar que un *compilador interpretado*:
   *a*) tarda más tiempo en generar la salida que un compilador normal.
   *b*) genera una salida que se ejecutará a más velocidad que la generada por un compilador normal.
   *c*) genera una salida más portable que la generada por un compilador normal.
4. El análisis de léxico:
   *a*) consiste en la implementación de un autómata de pila.
   *b*) consiste en la implementación de un autómata finito.
   *c*) consiste en la implementación de un autómata linealmente acotado.
5. Elige la frase correcta acerca del análisis de léxico:
   a) Puede crear entradas de identificadores en la tabla de símbolos, aunque se suele dejar esta tarea al *analizador sintáctico.*
   b) Su única misión es agrupar los caracteres del programa fuente en *lexemas* y pasar la secuencia de tokens al *analizador sintáctico.* El *analizador sintáctico* se encargará de otras tareas adicionales como eliminar comentarios y caracteres de espaciado.
   c) Nunca detecta errores de compilación. Los errores se detectan a partir de la fase de *análisis sintáctico.*
6. En la siguiente gramática libre de contexto que genera un lenguaje compuesto por números:

$$\begin{aligned} N &\rightarrow D'\ N \mid D \\ D &\rightarrow 0 \mid 1 \mid ... \mid 9 \\ D' &\rightarrow 1 \mid ... \mid 9 \end{aligned}$$

   *a*) los tokens del lenguaje serían los diez dígitos y, por tanto, no necesitarían atributo para identificar su lexema.
   *b*) el único token asociado a la gramática sería el token NUM, que necesitaría un atributo para almacenar su lexema, en caso de necesitarlo.
   *c*) no hay tokens definidos.

7. Dada la siguiente gramática:

$$\begin{array}{rcl} sent & \rightarrow & \textbf{if } expr \textbf{ then } sent \\ & | & \textbf{if } expr \textbf{ then } sent \textbf{ else } sent \\ & | & S \\ expr & \rightarrow & E \end{array}$$

   *a*) es posible encontrar una gramática equivalente no ambigua y LR.

   *b*) es posible encontrar una gramática equivalente no ambigua y LL.

   *c*) no es posible encontrar una gramática equivalente no ambigua.

8. Dada la gramática G siguiente:

$$\begin{array}{rcl} Lista & \rightarrow & [\,] \mid [\,Termino\,] \\ Termino & \rightarrow & Termino\ ,\ Termino \mid ID \mid Lista \end{array}$$

   y la cadena de entrada w = $[[a, b, c], [e, f]]$, decidir cuales serían las tres primeras reducciones que realizaría un analizador ascendente para reconocer w, suponiendo que el operador ',' es asociativo por la izquierda:

   *a*) $Lista \rightarrow [\ Termino\ ]$
   $Termino \rightarrow Termino\ ,\ Termino$
   $Termino \rightarrow Lista$

   *b*) $Termino \rightarrow ID$
   $Termino \rightarrow ID$
   $Termino \rightarrow Termino\ ,\ Termino$

   *c*) $Termino \rightarrow ID$
   $Termino \rightarrow ID$
   $Termino \rightarrow ID$

9. Supongamos que hemos calculado la colección $LR(1)$ para la gramática:

$$\begin{array}{lcl} S & \rightarrow & aAd \mid bBd \mid aBe \mid bAe \\ A & \rightarrow & c \\ B & \rightarrow & c \end{array}$$

   de modo que los conjuntos $I_6$ e $I_9$ contienen los siguientes items:

$$I_6 = \{[A \rightarrow c\ \bullet, d], [B \rightarrow c\ \bullet, e]\}$$

$$I_9 = \{[A \rightarrow c\ \bullet, e], [B \rightarrow c\ \bullet, d]\}$$

   Sabiendo que la gramática es *LR-canónica*, indica la respuesta correcta:

   a) La gramática es $LALR$ y $SLR$.

   b) La gramática no es $LALR$ ni $SLR$.

   c) La gramática no es $LALR$ pero si es $SLR$.

10. Si una gramática contiene (entre otras) las siguientes reglas:

$$\begin{array}{rcl} A & \rightarrow & a\ x \\ & | & \lambda \\ B & \rightarrow & A\ a\ y \end{array}$$

   *a*) puede ser LL(1).

   *b*) puede ser SLR(1).

   *c*) no puede ser LL(1).

11. Dada la siguiente gramática:

$$\begin{array}{lcl} S & \rightarrow & L = R \mid R \\ L & \rightarrow & *R \mid id \\ R & \rightarrow & L \end{array}$$

el pivote de la forma sentencial derecha $*L = *id$ es:

a) $*L = *\underline{id}$

b) $*\underline{L} = *id$

c) $\underline{*L} = *id$

12. Indica cual es la afirmación **falsa**:

a) Las gramáticas atribuidas que pueden evaluarse con un analizador ascendente incluyen a todas las gramáticas *S-atribuidas*.

b) Las gramáticas atribuidas que pueden evaluarse con un analizador descendente incluyen a todas las gramáticas *S-atribuidas*.

c) Las gramáticas *S-atribuidas* incluyen a todas las gramáticas atribuidas que pueden evaluarse con un analizador descendente.

13. El siguiente esquema de traducción

$$\begin{array}{lcl} S & \rightarrow & A\ \{B.c = A.c\}\ B \\ A & \rightarrow & a\ A_1\ \{A.c = A_1.c + 1\} \\ A & \rightarrow & a\ \{A.c = 1\} \\ B & \rightarrow & b\ \{B_1.c = B.c - 1\}\ B_1 \\ B & \rightarrow & b\ \{If\ (B.c - 1 = 0)\ printf(True); \\ & & \qquad\qquad else\ printf(False);\} \end{array}$$

*a*) sólo usa atributos sintetizados.

*b*) va calculando la cantidad de a's y de b's, dando un mensaje de error en caso de que sólo haya una b.

*c*) comprueba si la cantidad de a's es igual que la de b's.

14. La regla

`if` $f$ tiene el tipo $s \rightarrow t$ y $x$ tiene el tipo $s$,
`then` la expresión $f(x)$ tiene el tipo $t$.

*a*) expresa la *inferencia de tipos* en funciones con un argumento.

*b*) expresa la *síntesis de tipos* en funciones con un argumento.

*c*) convierte el tipo del dominio de una función al de su rango.

15. Elige la opción correcta:

a) Un árbol sintáctico *abstracto* es una simplificación de un GDA.

b) Un GDA se puede construir utilizando las mismas técncias que las usadas para construir árboles sintácticos *abstractos* (p.e. mediante una DDS).

c) Un árbol sintáctico *abstracto* es una simplificación de un árbol de análisis sintáctico con la particularidad de que un nodo puede tener más de un padre.

## Parte II: PREGUNTAS CORTAS. 10 %.

1. Dado el siguiente programa en C que implementa un analizador, dar la gramática que genera el lenguaje reconocido por dicho analizador:

```
/* Análisis sintáctico */
#include <stdio.h>
char token, cadena[80];
int i=0;
void main(void)
{
    printf("Introduce la cadena a reconocer \n");
    printf("=>");
    scanf("%s",cadena);
    token= cadena[0];
    if (a()) printf("\nCADENA RECONOCIDA");
        else printf("\nCADENA NO RECONOCIDA");
}
/*******************************/
int a(void)
{
if (token== 'x') { i+=1; token= cadena[i]; return(1); }
else if (token== '(')
        if (b())
            {
                if (token== ')') return(1);
                else return(0);
            }
        else return(0);
      else return(0);
}
/*******************************/
int b(void)
{
    i+=1;
    token= cadena[i];
    if (a())
        if (c()) return(1);
        else return(0);
    else return(0);
}
/*******************************/
int c(void)
{
    while (token== '+')
    {
        i+=1;
        token= cadena[i];
        if (!a()) return(0);
    }
    return(1);
}
```

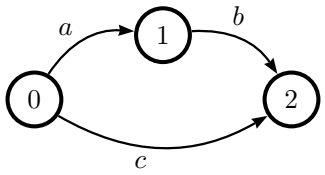
Dada cualquier gramática G, ¿existe alguna propiedad que pueda cumplir dicha gramática que haga imposible la implementación de un analizador descendente recursivo para reconocer L(G)? Justifica la respuesta.

**Parte III: PROBLEMA.** 60 %

La siguiente gramática $G$, con $V_T = \{\textbf{id}\ ,\ \textbf{num}\ ,\ \&\ ,\ ;\ ,\ :\ ,\ >\}$, $V_N = \{A, S, D, L, B\}$, símbolo inicial $A$ y el siguiente conjunto $P$ de producciones:

$$
\begin{array}{lcl}
A & \rightarrow & A\ S \\
 & | & S \\
S & \rightarrow & \textbf{num}\ :\ D\ ; \\
D & \rightarrow & L \\
 & | & \lambda \\
L & \rightarrow & L\ \&\ B \\
 & | & B \\
B & \rightarrow & \textbf{id} > \textbf{num}
\end{array}
$$

permite representar autómatas finitos textualmente. Por ejemplo, el siguiente autómata:




se representa de la siguiente forma:

```
0 : a > 1 & c > 2 ;
1 : b > 2 ;
2 : ;
```

Responder a las siguientes cuestiones:

1. *(1 punto)* Decir, justificando la respuesta, y sin construir ninguna tabla de análisis, si G es LL(1). En caso de que no lo sea, realizar las transformaciones necesarias en la gramática que puedan conducir a que lo sea.

2. *(2 puntos)* Construir la colección LR(1) para G, y la tabla LR-canónica. Indicar si G es una gramática LR-canónica justificando la respuesta.

3. *(0.5 puntos)* Simular el reconocimiento de la cadena $w \equiv$ `0 : a > ;`

4. *(1 puntos)* Indicar si G es una gramática LALR y/o SLR, justificando la respuesta, y sin calcular ninguna colección de ítems adicional.

5. *(1.5 puntos)* Dar una *definición dirigida por la sintaxis* que permita verificar si los estados destino indicados en las transiciones están definidos. El autómata del ejemplo anterior cumple esta condición, ya que los estados 1 y 2 usados en las transiciones están definidos. Sin embargo, el siguiente autómata es sintácticamente correcto pero semánticamente erróneo:

   ```
   0 : a > 1 & b > 2 ;
   1 : b > 1 ;
   ```

   ya que el estado 2, usado en una transición de salida del estado 0, no está definido. Para resolver este apartado se debe:

   *a*) indicar el número y tipo de atributos asociado a cada símbolo de G.

   *b*) describir cualquier función auxiliar que se requiera.

   *c*) asociar a cada regla de producción de G las acciones semánticas necesarias.

   *d*) decorar el árbol sintáctico proporcionado a continuación, que es el correspondiente al ejemplo del primer autómata del enunciado.

   *e*) indicar si G es S-atribuida y/o L-atribuida, justificando la respuesta.

El autómata del enunciado genera el siguiente árbol de análisis: