EXAMEN DE COMPILADORES (2° Grado en Informática, final enero-2017)

Apellidos, nombre: **DNI:**

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. ¿Cuál de las siguientes NO es una regla de ejecución de un programa traductor?
   a) El programa objeto generado debe ser semánticamente equivalente al programa fuente.
   b) El programa traductor puede ejecutarse si y sólo si está escrito en código de una máquina real.
   c) El programa objeto generado estará escrito en el lenguaje destino del traductor.
2. Indica la respuesta correcta sobre los compiladores:
   a) La fase de optimización de código intermedio es opcional, pero no lo es la fase de optimización de código dependiente de la máquina.
   b) La etapa de análisis léxico de un compilador no puede emplear la tabla de símbolos.
   c) Los tokens devueltos por el analizador léxico corresponden a los símbolos terminales de la gramática manejada por el analizador sintáctico.
3. ¿Cuál de los siguientes NO es un ejemplo de comprobación de sintaxis sensible al contexto de un lenguaje de programación?
   a) La verificación de que una instrucción break sólo puede aparecer en el cuerpo de bucles o instrucciones switch.
   b) La verificación de que el tipo de dos operandos es compatible.
   c) La verificación de que los paréntesis de una expresión aritmética están balanceados.
4. Señala en qué situación nos interesa tener un intérprete
   a) cuando el programa se va usar muchas veces y se va a ejecutar en producción.
   b) cuando se trabaja de forma interactiva y se quiere ver el resultado de una instrucción antes de ejecutar la siguiente.
   c) cuando las instrucciones tienen un formato complejo.
5. Dada la siguiente especificación Flex, ¿qué retorna el analizador léxico si la entrada es `print id;`?

```
D [0-9]
L [a-z]
P [^0-9a-z;_\n\t\r]
%%
[ \n\t\r]+                  { }
({L}|"_")({L}|"_"|{D})*     { return 1; }
"print"                     { return 2; }
";"                         { return 3; }
{P}+                        { return 4; }
```

   a) La secuencia 1 1 3
   b) La secuencia 2 1 3
   c) Únicamente 4
6. La siguiente gramática, que genera la declaración de tipos de funciones en el lenguaje Haskell:

$$\begin{aligned} D &\rightarrow id :: T \\ T &\rightarrow [\ T\ ] \mid T -> T \mid (\ listaT\ ) \mid Integer \mid Float \mid Bool \\ listaT &\rightarrow T\ ,\ listaT \mid \lambda \end{aligned}$$

   a) No es LL, puesto que es recursiva por la izquierda, pero sí puede ser LR.
   b) No es LL ni LR porque una gramática recursiva por la izquierda no puede ser ni LL ni LR.
   c) No es LL ni LR.

7. Supongamos que hemos calculado la colección LR(0) y la tabla SLR para la gramática anterior:
   de modo que el conjunto I_{14} contiene los siguientes items:

$$I_{14} = \{T \to T- > T\bullet, T \to T \bullet - > T\}$$

   produciéndose una tabla SLR con conflictos. Elegir la acción adecuada para la casilla del estado 14 y el símbolo de entrada $- >$, teniendo en cuenta que este operador es **asociativo por la derecha**:

   a) d10.
   b) r3.
   c) r1.

8. Dada la gramática anterior, la forma sentencial `id :: [ T ->T ->T]`:

   a) Es forma sentencial izquierda.
   b) Es forma sentencial derecha.
   c) Es forma sentencial izquierda y derecha.

9. Dada la gramática anterior, en la forma sentencial `id :: [ T ->T ]`:

   a) El pivote es `T ->T` .
   b) El pivote es `[T->T]`.
   c) Tiene dos pivotes, puesto que la gramática es ambigua.

10. Si en un análisis sintáctico descendente nos encontramos con un error tal que en la tabla de análisis no existe ninguna regla a aplicar, es decir, $T[A, t] = \emptyset$, siendo A el símbolo de la cima de la pila y t el siguiente token en la entrada, aplicando la gestión de error en modo pánico debemos:

    a) Descartar A y t
    b) Descartar símbolos en la entrada hasta encontrar un a tal que
       $\to a \in PRIMERO(A) \Rightarrow$ sacamos A de la pila y continuamos, o bien
       $\to a \in SIGUIENTE(A) \Rightarrow$ descartamos a y continuamos.
    c) Descartar símbolos en la entrada hasta encontrar un a tal que
       $\to a \in PRIMERO(A) \Rightarrow$ continuamos, o bien
       $\to a \in SIGUIENTE(A) \Rightarrow$ sacamos A de la pila y continuamos.

11. Dada la siguiente gramática

$$\begin{array}{lcl} declarations & \to & declarations\ var\ identifierL\ ;\ |\ declarations\ let\ identifierL\ ;\ |\ \lambda \\ identifierL & \to & asig\ |\ identifierL\ ,\ asig \\ asig & \to & id \end{array}$$

    el conjunto I_0 de la colección LR(1) es:

    a) $\{[declarations' \to \cdot declarations, \$], [declarations \to \cdot declarations\ var\ identifierL\ ;, \$/var/let], [declarations \to \cdot declarations\ let\ identifierL\ ;, \$/var/let], [declarations \to \lambda\cdot, \$/var/let]\}$
    b) $\{[declarations' \to \cdot declarations, \$], [declarations \to \cdot declarations\ var\ identifierL\ ;, \$/var/let], [declarations \to \cdot declarations\ let\ identifierL\ ;, \$/var/let]\}$
    c) $\{[declarations' \to \cdot declarations, \$], [declarations \to \cdot declarations\ var\ identifierL\ ;, \$/var], [declarations \to \cdot declarations\ let\ identifierL\ ;, \$/let], [declarations \to \lambda\cdot, \$]\}$

12. Con respecto a la siguiente gramática podemos afirmar

$$\begin{array}{lcl} declarations & \to & declarations\ var\ identifierL\ ;\ |\ declarations\ let\ identifierL\ ;\ |\ \lambda \\ identifierL & \to & asig\ |\ identifierL\ ,\ asig \\ asig & \to & id \end{array}$$

    que dada la siguiente forma sentencial *var a,b,c; let k, l;*

    a) existe sólo un árbol de derivación para la anterior forma sentencial .
    b) existen dos árboles de derivación para la anterior forma sentencial.
    c) existen cuatro árboles de derivación para la anterior forma sentencial.

13. Una gramática con atributos heredados

    a) Puede no ser ni L-Atribuida ni S-atribuida.

    b) Será L-Atribuida, aunque no S-Atribuida.

    c) Será L-Atribuida y, por tanto, S-Atribuida, pues las primeras están incluidas en las segundas.

14. Dada la siguiente definición dirigida por la sintaxis:

$$\begin{array}{lll} P & \to [L](num) & \{L.x = num.v; P.v = L.v;\} \\ L & \to E & \{L.v = E.v;\} \\ L & \to L_1, E & \{L_1.x = L.x; L.v = L_1.v * L_1.x + E.v;\} \\ E & \to num & \{E.v = num.v;\} \\ E & \to P & \{E.v = P.v;\} \end{array}$$

    podemos asegurar que esta gramática atribuida:

    a) Se puede evaluar mediante un esquema de traducción durante un análisis ascendente implementado con Bison.

    b) Se puede evaluar mediante un esquema de traducción durante un análisis descendente recursivo predictivo.

    c) Se puede evaluar generando primero un árbol sintáctico durante un análisis ascendente implementado con Bison, que puede ser recorrido tras el análisis sintáctico.

15. ¿Qué tipo de representación intermedia facilita el movimiento de instrucciones de código intermedio que puede ser necesario en compiladores optimizadores?

    a) Cuádruplas.

    b) Tripletas indirectas.

    c) Tripletas.

Parte II: PROBLEMA. 70 %.

La siguiente gramática G con $V_T = \{=, \vee, \texttt{id}, \texttt{(}, \texttt{)}, \neg, \Rightarrow\}$ y $V_N = \{S, T\}$, siendo P:

$$\begin{array}{ll} S & \to \texttt{id} \ = \ T \\ T & \to T \ \vee \ T \mid T \ \Rightarrow \ T \mid \neg \ T \mid \texttt{(} \ T \ \texttt{)} \mid \texttt{id} \end{array}$$

donde los operadores $\vee$ y $\Rightarrow$ son asociativos por la izquierda y $\vee$ tiene más precedencia que $\Rightarrow$, permite representar parcialmente fórmulas de lógica de predicados. Se pide:

1. *(1 punto)* Calcular los conjuntos PRIMERO y SIGUIENTE de S y T. Sin calcular los conjuntos *predict* ni la tabla LL, dar todos los argumentos para justificar que G no es una gramática LL(1).

2. *(1.5 puntos)* Modificar la gramática G para intentar conseguir una equivalente que sea LL(1). Comprobar si la nueva gramática es LL(1) calculando los conjuntos *predict* para cada regla.

3. *(2.5 puntos)* Indicar y justificar si la gramática G es SLR(1), LR(1) y/o LALR(1), calculando la colección LR(0) y la tabla de análisis SLR. En caso de que no sea SLR, eliminar en la tabla los conflictos de forma adecuada.

4. *(0.5 puntos)* Simular, con la tabla obtenida al eliminar los conflictos, el algoritmo ascendente con la entrada `id=id`$\Rightarrow \vee$`id` haciendo recuperación en modo pánico en caso de error.

5. *(1.5 puntos)* Realizar una definición dirigida por la sintaxis (DDS) que permita generar la expresión lógica equivalente pero eliminando el operador $\Rightarrow$[1] y dejando, además, las negaciones en el nivel más profundo posible. Por ejemplo, la salida para la cadena `a=(c` $\vee \neg$`d)` $\Rightarrow$ `(b` $\vee$ `d)` sería `a=(`$\neg$`c` $\wedge$ `d)` $\vee$ `(b` $\vee$ `d)`. Definir los atributos que sean necesarios e indicar de forma justificada si la gramática es L-atribuida y/o S-atribuida. Decorar finalmente el árbol sintáctico correspondiente a la entrada `a=(c` $\vee \neg$`d)` $\Rightarrow$ `(b` $\vee$ `d)`.

[1] Recordar para esto la equivalencia entre $A \ \Rightarrow \ B$ y $\neg A \ \vee \ B$.