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

Apellidos, nombre: **GRUPO:**
D.N.I.:
Este enunciado y todos los folios usados deben entregarse al salir

Parte I: PREGUNTAS TIPO TEST. 30 %.
Cada respuesta correcta vale 0.2 puntos.
Cada dos respuestas incorrectas anulan una correcta.

1. Elegir, entre los siguientes, el tipo de *máquina abstracta* más adecuado para realizar el **análisis sintáctico** de un lenguaje de programación:
   a) *Autómata Finito*, pues siempre necesitaremos analizar las palabras del lenguaje.
   b) *Autómata de Pila*, a pesar de que la mayoría de los lenguajes de programación no son *libres de contexto* sino *sensibles al contexto*.
   c) *Autómata Linealmente Acotado*, puesto que la mayoría de los lenguajes de programación tienen restricciones contextuales (por ejemplo, la declaración de variables).
2. Elige la frase correcta acerca del **análisis de léxico**:
   a) Suele ser una función a la que llama el *analizador sintáctico* cada vez que necesita un token, aunque podría no realizarse de forma explícita y dejar el reconocimiento de palabras como parte del análisis sintáctico.
   b) Suele generar un fichero explícito de tokens que constituye la entrada del *analizador sintáctico*.
   c) Se realiza mediante la simulación de *autómatas de pila*, que proporcionan la potencia suficiente para las tareas de E/S.
3. Un compilador interpretado consiste en:
   a) Un compilador que da la opción de generar *código máquina* o *código intermedio* dependiendo de las necesidades del usuario.
   b) Un compilador que genera código escrito en *lenguaje de alto nivel* que posteriormente es interpretado por algún intérprete adecuado existente en el sistema.
   c) Un compilador que genera *código intermedio* que posteriormente es interpretado por una **máquina virtual**.
4. Con respecto al código generado por un compilador:
   a) Podemos implementarlo de manera que exista garantía de que siempre genera *código óptimo*, aunque generalmente no merece la pena.
   b) Nunca podremos implementar un compilador que genere código óptimo en ninguna situación, puesto que se trata de un problema *NP-completo*.
   c) Nunca podremos implementar un compilador que garantice la generación de código óptimo en todos los casos.
5. En el proceso de **arranque** debe darse la circunstancia de que:
   a) Coinciden el *lenguaje de implementación* y el *lenguaje destino*.
   b) Coinciden el *lenguaje de implementación* y el *lenguaje fuente*.
   c) Coinciden el *lenguaje de implementación* con el de una *máquina virtual* existente en el sistema.
6. Un **preprocesador** es:
   a) Una herramienta que toma como entrada varios ficheros escritos en lenguaje de alto nivel y permite traducirlos a código máquina.
   b) Un traductor cuyo *lenguaje fuente* está constituido por una serie de macros y su *lenguaje destino* es una forma extendida de algún lenguaje de alto nivel.
   c) Un traductor cuyo *lenguaje fuente* es una forma extendida de un lenguaje de alto nivel y su *lenguaje destino* es la forma estándar del mismo lenguaje.
7. Señalar la razón por la que se considera qua las **técnicas de recuperación de errores** son importantes:
   a) Porque permiten obtener un código objeto sin los errores introducidos por el usuario.
   b) Porque permiten informar al usuario de una forma más precisa.
   c) Porque posibilitan la detección de más de un error.

8. Una **gramática recursiva por la izquierda**:
   a) No puede ser *LL*, ni *LR*.
   b) No puede ser *LL*, aunque sí *SLR*.
   c) No puede ser *SLR*, aunque sí *LR-Canónica*.
9. Las **gramáticas LL** y **LR**:
   a) Pueden tener *λ-reglas* y ser *ambiguas*.
   b) Pueden tener *λ-reglas* pero no pueden ser *ambiguas*.
   c) No pueden tener *λ-reglas* ni ser *ambiguas*.
10. Elegir la opción correcta:
    a) El *método de análisis LL* es ascendente y predictivo.
    b) El *método de análisis LR* es ascendente y no predictivo.
    c) Los *métodos de análisis LL y LR* son ambos predictivos, de manera que el primero funciona obteniendo las derivaciones por la izquierda desde el símbolo inicial de la gramática y el segundo obteniendo las reducciones por la izquierda a partir de la cadena de entrada.
11. El **tratamiento de errores** a nivel de frase en cualquiera de los métodos estudiados:
    a) Es un método sistemático que no varía de una gramática a otra.
    b) Requiere la determinación de posibles errores que puedan producirse en el lenguaje fuente, y para ello podemos hacer una llamada a un procedimiento particular en cada casilla vacía de la tabla de análisis.
    c) Es un método que requiere la determinación previa de todas las posibles frases erróneas que puedan aparecer en un programa y que, además, permite corregirlas.
12. Supongamos que hemos calculado la colección LR(0) y la tabla SLR para la siguiente gramática:

$$E \quad \rightarrow \quad E * E \mid E + E \mid id$$

    de modo que los conjuntos I_5 e I_6 contienen los siguientes items:

$$I_5 = \{E \rightarrow E * E\bullet, E \rightarrow E \bullet * E, E \rightarrow E \bullet + E\}$$
$$I_6 = \{E \rightarrow E \bullet * E, E \rightarrow E + E\bullet, E \rightarrow E \bullet + E\}$$

    produciéndose una tabla SLR con conflictos. Elegir la acción adecuada para la casilla del estado 6 y el símbolo de entrada *:
    a) d3.
    b) r1.
    c) r2.
13. Una **gramática L-Atribuida**:
    a) Puede tener atributos sintetizados y heredados de cualquier otro símbolo de la gramática. De hecho, cualquier gramática *L-Atribuida* es también *S-Atribuida*.
    b) Puede tener atributos sintetizados y heredados sólo de hermanos izquierdos. De hecho, cualquier gramática *L-Atribuida* es también *S-Atribuida*.
    c) Puede tener atributos sintetizados. De hecho, cualquier gramática *S-Atribuida* es también *L-Atribuida*.
14. La **tabla de símbolos**:
    a) Es una estructura de datos útil para analizar la *semántica* de un lenguaje de programación, aunque en ningún caso se usa en la *comprobación de tipos*.
    b) Es una estructura de datos que resulta útil para el almacenamiento de las variables y sus atributos, de manera que suele pasar información de las *declaraciones* a los *usos*.
    c) Es una estructura de datos que sirve para almacenar información semántica, de forma que dicha información se suele añadir a lo largo de la *fase de síntesis* para usarla posteriormente en la *fase de análisis*.
15. Con respecto a la **traducción de expresiones y sentencias** de un lenguaje de programación:
    a) Si son necesarios *atributos heredados* y realizamos un *análisis LR*, sólo podremos llevarlo a cabo accediendo directamente a valores intermedios de la pila de análisis.
    b) Si son necesarios *atributos sintetizados* y realizamos un *análisis LL*, no podremos llevar a cabo la traducción en ningún caso.
    c) Si son necesarios *atributos heredados* y realizamos un *análisis LL*, no podremos llevar a cabo la traducción en ningún caso.

Parte II: PREGUNTAS CORTAS. 10 %.

1. Si una gramática G es LR-Canónica pero no es LALR, en la tabla LALR sólo pueden aparecer conflictos reducción/reducción y no desplazamiento/reducción. Proponer un ejemplo (concreto o genérico) en el que al obtener un estado I_{ij} para una tabla LALR a partir de dos estados sin conflictos I_i e I_j de una colección LR(1), en el primero (el I_{ij}) aparezca un conflicto reducción/reducción.

2. Dada la gramática G con $V_T = \{int, float, ident, , \}$, $V_N = \{DECLARACION, TIPO, LISTA\_VAR\}$, símbolo inicial $DECLARACION$ y el siguiente conjunto de producciones:

$$\begin{array}{lcl} DECLARACION & \rightarrow & TIPO\ LISTA\_VAR \\ TIPO & \rightarrow & int \\ & | & float \\ LISTA\_VAR & \rightarrow & ident\ ,\ LISTA\_VAR \\ & | & ident \end{array}$$

   ¿Es G una gramática **propia**? ¿Por qué? ¿Puede ser una gramática no-propia LR? ¿Y LL? Justificar las respuestas.

Parte III: PROBLEMAS. 60 %

La siguiente gramática G, con $V_T = \{(,), :=, \mathbf{id}, \mathbf{num}, ,\}$, $V_N = \{, L, R\}$, símbolo inicial A y el siguiente conjunto P de producciones:

$$\begin{array}{lcl} A & \rightarrow & (\ L\ ) := (\ R\ ) \\ L & \rightarrow & \mathbf{id} \\ & | & L\ ,\ \mathbf{id} \\ R & \rightarrow & \mathbf{num} \\ & | & R\ ,\ \mathbf{num} \end{array}$$

permite definir asignaciones múltiples (es una característica de algunos lenguajes de programación, como Perl). Por ejemplo, la siguiente sentencia:

```
(a, b) := (1, 2)
```

asigna simultáneamente a la variable *a* el valor *1*, y a la variable *b* el valor *2*. Responder a las siguientes cuestiones:

1. *(1 puntos)* Decir, justificando la respuesta, y sin construir ninguna tabla de análisis, si G puede ser LL(1). En caso de que no lo sean, realizar las transformaciones necesarias para llegar a una gramática equivalente que pueda serlo.

2. *(1 punto)* Verificar si G o la gramática equivalente obtenida en el apartado anterior es LL(1), calculando los conjuntos PRIMERO y SIGUIENTE, los conjuntos *predict*, y la tabla de análisis.

3. *(0.5 puntos)* Simular el comportamiento del algoritmo de análisis LL(1), usando la tabla obtenida en el apartado anterior, para la cadena $w \equiv (a, b)(1, 2)$, realizando la recuperación de errores en modo pánico en caso de error.

4. *(1.25 puntos)* Obtener la colección LR(1) para la gramática G inicial y construir la tabla LR-canónica. Indicar si G es una gramática LR-canónica justificando la respuesta.

5. *(0.75 puntos)* Indicar si G es una gramática LALR y/o SLR, justificando la respuesta, y sin calcular ninguna colección de ítems adicional.

6. *(1.5 puntos)* Dar una *definición dirigida por la sintaxis* usando la gramática G para realizar la traducción de las asignaciones múltiples. La definición debe construir dos listas, una con los identificadores de la parte izquierda de la asignación, y otra con los valores de la parte derecha. También debe comprobar que las dos listas tienen la misma longitud, y en caso contrario debe indicar un error semántico. Las funciones que se pueden usar son:

   - $longitud([e_1, e_2, \ldots, e_n]) = n$
   - $iesimo([e_1, e_2, \ldots, e_n], i) = e_i$
   - $añadir([e_1, e_2, \ldots, e_n], e) = [e_1, e_2, \ldots, e_n, e]$
   - $crealista(e) = [e]$
   - $genasig(a, b)$ genera el código de una asignación simple $a := b$.
   - $error(mensaje)$

   Para realizar este apartado es necesario:

   - indicar el número y tipo de atributos asociado a cada símbolo de G.
   - asociar a cada regla de producción de G las acciones semánticas necesarias.
   - decorar el árbol sintáctico correspondiente a la cadena $w \equiv (a, b) := (1, 2)$.
   - indicar si G es S-atribuida y/o L-atribuida, justificando la respuesta.