EXAMEN DE COMPILADORES (2° Grado en Informática, final junio-2014)

FACULTAD DE INFORMÁTICA

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.

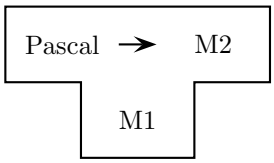
1. Se ha programado en Bison el siguiente esquema de traducción asociado a la gramática de expresiones aritméticas:

```
expresion : expresion MAS expresion {$$ = $1 + $3; }
          | expresion MENOS expresion {$$ = $1 - $3;}
          | expresion POR expresion {$$ = $1 * $3;}
          | expresion DIV expresion {$$ = $1 / $3;}
          | NUM { $$ = $1; }
```

   donde todas las variables semánticas son de tipo entero y la que corresponde al token NUM contiene el resultado de `atoi(yytext)`, calculado por un analizador léxico generado con Flex. A qué tipo de procesador de lenguaje podría pertenecer este esquema de traducción:

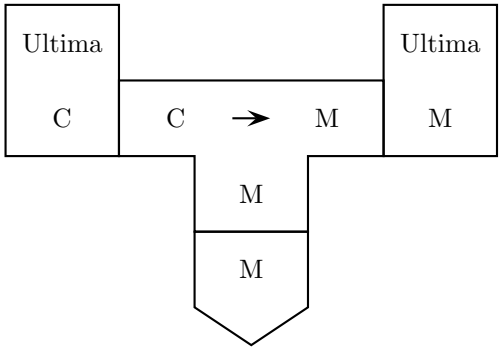
   a) A un intérprete.
   b) A un compilador.
   c) A ninguno de los dos tipos anteriores.

2. Si M1 y M2 son dos máquinas diferentes, ¿qué tipo de programa traductor es el siguiente?



   a) Un compilador cruzado.
   b) Un compilador de varias etapas.
   c) Un compilador just-in-time.

3. ¿Tiene sentido el siguiente diagrama de programas?



   a) No, porque C no es un lenguaje de bajo nivel.
   b) No, porque no se puede compilar un intérprete.
   c) Sí, incluso si Ultima representa el lenguaje de bajo nivel de una máquina.

4. Supongamos que se está realizando un análisis descendente de la entrada $id + id * id$, y la configuración actual del analizador es la siguiente:

Pila	Entrada
\$ $E'T'$	$*id$ \$

   ¿Cuál es la forma sentencial actual del análisis?

   a) $id + id\ T'\ E'$
   b) $E'\ T' * id$
   c) $T'\ E' * id$

5. Supongamos que hemos desarrollado un analizador léxico que debe reconocer tokens para representar asignaciones de números a variables. Se van a emplear las siguientes expresiones regulares:

ID	`[a-z$][a-z0-9$]*`
ASIGN	`:=`
NUM	`[0-9]+`
Blancos	`[\n\t ]+`

Indica la opción <u>incorrecta</u>:

a) Se puede realizar un control de errores en modo pánico con la expresión regular `[^a-z$:0-9]+`

b) La implementación de este analizador léxico necesita caracteres de anticipación.

c) Para tratar los blancos como delimitadores, basta con ignorarlos al reconocerlos.

6. Para analizar frases generadas por la siguiente gramática:

$$\begin{array}{lcl} X & \rightarrow & YZX \\ & | & x \\ Y & \rightarrow & ZXY \\ & | & y \\ Z & \rightarrow & XYZ \\ & | & z \end{array}$$

Elige la opción <u>incorrecta</u>:

a) Sería adecuado usar un analizador descendente predictivo recursivo puesto que la gramática es LL.

b) Un analizador descendente predictivo recursivo podría entrar en bucles infinitos con esta gramática.

c) No es posible usar un analizador descendente predictivo recursivo.

7. Dada la siguiente gramática:

$$\begin{array}{lcl} S & \rightarrow & sent \\ & | & S;S \\ & | & \lambda \end{array}$$

a) Es recursiva por la izquierda, LR y no LL.

b) Es ambigua, LL y no SLR.

c) No es LL, ni SLR, ni LALR ni LR-Canónica.

8. Los conjuntos SIGUIENTE de un no terminal A en una gramática cualquiera:

a) Siempre constituyen un subconjunto de los tokens de anticipación de cualquier item LR que contenga la regla $A \rightarrow \alpha$.

b) Contienen o bien coinciden con el conjunto de tokens de anticipación de cualquier item LR que contenga la regla $A \rightarrow \alpha$.

c) No están relacionados de ninguna forma con el conjunto de tokens de anticipación de cualquier item LR que contenga la regla $A \rightarrow \alpha$.

9. En relación con el concepto de *pivote* de una forma sentencial, indica la opción correcta:

a) Cualquier prefijo del lado derecho de una regla de producción puede actuar como pivote.

b) Es posible localizar el pivote de una forma sentencial haciendo la derivación completa más a la izquierda y viendo cuál es el cambio efectuado en el último paso de derivación.

c) Si β es el pivote de una forma sentencial con la forma $\alpha\beta\gamma$ entonces $\alpha\beta$ es un prefijo viable.

10. En relación con la función de transición del autómata de pila usado por los métodos de análisis LL(1) y LR(1), indica la opción correcta:

a) La calculada por el método LL(1) es determinista, mientras que la del método LR(1) no lo es.

b) La calculada por el método LR(1) es determinista, mientras que la del método LL(1) no lo es.

c) Ambos métodos LL(1) y LR(1) calculan una función de transición determinista.

11. Respecto al análisis ascendente, indica la opción correcta:

    a) Se puede realizar un análisis ascendente usando reducciones más a la derecha, si bien es menos práctico por requerir más memoria.

    b) No se puede realizar un análisis ascendente usando reducciones más a la derecha, ni siquiera con entradas de poca longitud.

    c) Un análisis ascendente que usase reducciones derechas requeriría que la gramática no fuese recursiva por la derecha.

12. El método LR(1) es más potente que el LL(1) porque:

    a) Puede operar sobre gramáticas no propias, mientras que el LL(1) no.

    b) Tiene más información para decidir la acción a ejecutar en cada paso, ya que dispone de todos los datos insertados en la pila más el siguiente token de la entrada, mientras que el analizador descendente sólo decide qué hacer en función del símbolo del tope de la pila y el siguiente token de la entrada.

    c) Si hay conflictos en la tabla de análisis LR(1), se pueden resolver caso por caso, mientras que los conflictos en las tablas LL(1) no se pueden resolver a mano.

13. Supongamos que se realiza un análisis SLR de la gramática siguiente

$$L \ \rightarrow L \wedge L \mid L \vee L \mid \neg L \mid L \Rightarrow L \mid L \Leftrightarrow L \mid (L) \mid \texttt{true} \mid \texttt{false}$$

    y uno de los conjuntos de items es el siguiente:

$$I_{11} = \{\ [\ L \rightarrow L\ \vee L\ \bullet\ ]\ ,\ [\ L \rightarrow L\ \bullet\ \wedge\ L\ ]\ ,\ [\ L \rightarrow L\ \bullet\ \vee\ L\ ]\ ,\ [\ L\ \rightarrow L \bullet\ \Rightarrow\ L\ ]\ ,\ [\ L \rightarrow L \bullet\ \Leftrightarrow\ L\ ]\ \}$$

    de manera que, en la tabla de análisis la fila correspondiente al estado 11 quedaría así:

Estado	Acción								Ir-a
	$\vee$	$\wedge$	$\neg$	$\Rightarrow$	$\Leftrightarrow$	`true`	`false`	\$	L
					...				
11	r2/d5	r2/d6		r2/d7	r2/d8			r2	
					...				

    Teniendo en cuenta que todos los operadores son asociativos por la izquierda y que $\Rightarrow$ y $\Leftrightarrow$ tienen menor precedencia que $\vee$ y $\wedge$ (ambos también con la misma precedencia), para eliminar los conflictos deberíamos:

    a) Elegir las reducciones en todos los casos.

    b) Elegir los desplazamientos en las casillas $[11, \vee]$ y $[11, \wedge]$ y las reducciones en $[11, \Rightarrow]$ y $[11, \Leftrightarrow]$.

    c) Elegir la reducción en la casilla $[11, \vee]$ y desplazamientos en $[11, \wedge]$, $[11, \Rightarrow]$ y $[11, \Leftrightarrow]$.

14. Supongamos que queremos evaluar un atributo heredado de un no terminal $X.h$, y dicho no terminal aparece en las siguientes reglas:

$$\begin{array}{ll} (1) & A \rightarrow \alpha\ X\ \beta \\ (2) & X \rightarrow \gamma\ X\ \delta \\ (3) & X \rightarrow \eta \end{array}$$

    donde $\alpha, \beta, \gamma, \delta, \eta \in (V_T \cup V_N - \{X\})*$. ¿Qué reglas podrían incluir una acción semántica de evaluación de $X.h$?

    a) Todas.

    b) Sólo la 1.

    c) La 1 y la 2.

15. Supongamos que de un lenguaje de programación se ha extraído el siguiente fragmento de gramática:

$$\begin{array}{lllll} (1) & S & \rightarrow & id := E \\ (2) & & \mid & if\ E = false\ then\ S \\ (3) & E & \rightarrow & id \\ (4) & & \mid & E \leq E \leq E \end{array}$$

    Suponiendo que en dicho lenguaje las variables pueden ser reales, enteras y lógicas y que se exige su declaración.

    a) En la regla (1) podríamos almacenar en la tabla de símbolos el tipo de la variable correspondiente a *id*, y en la regla (3) podríamos consultarlo.

    b) En la regla (3) podríamos almacenar en la tabla de símbolos el tipo de la variable correspondiente a *id*, y en la regla (1) podríamos consultarlo.

    c) En las reglas (1) y (3) tendríamos que consultar el tipo de las variables en la tabla de símbolos para ir haciendo las comprobaciones en el resto de las reglas.

Parte II: PROBLEMAS. 70 %.

La siguiente gramática G_1 permite expresar parcialmente las sentencias `if-then` e `if-then-else` de un lenguaje de programación ($V_T = \{\texttt{if}, \texttt{then}, \texttt{else}, \texttt{print}, \texttt{id}, \texttt{==}, \texttt{num}, \texttt{str}, \texttt{;}\}$ y $V_N = \{S, C, P, E\}$):

$$\begin{array}{lll} S & \rightarrow & \texttt{if}\ C\ \texttt{then}\ P\ E \\ C & \rightarrow & \texttt{id == num} \\ P & \rightarrow & \texttt{print str ;} \\ E & \rightarrow & \texttt{else}\ P \mid \lambda \end{array}$$

1. *(1.5 puntos)* Demuestra que la gramática es LALR(1).
2. *(1 puntos)* Con la tabla obtenida en el apartado anterior, analiza la entrada `if id == ; then print str` usando tratamiento de errores en modo pánico.
3. *(0.5 puntos)* Indica de forma justificada si la gramática es LR(1) y/o SLR(1), sin calcular ninguna colección de ítems adicional.
4. *(0.5 puntos)* Explica por qué razón esta gramática no es ambigua, a pesar de emplear `if-then` e `if-then-else`.

Supongamos que se pretende ampliar la gramática anterior para que las condiciones de las sentencias `if` puedan expresar fórmulas de la lógica proposicional. La parte de la gramática que expresa este tipo de condiciones lógicas es $G_2 = \{V_T, V_N, L, P\}$, con $V_T = \{\vee, \wedge, \Rightarrow, \Leftrightarrow, \texttt{true}, \texttt{false}\}$, $V_N = \{L\}$, y P el siguiente conjunto de producciones:

$$\begin{array}{lll} L & \rightarrow & L \wedge L \\ & \mid & L \vee L \\ & \mid & \neg L \\ & \mid & L \Rightarrow L \\ & \mid & L \Leftrightarrow L \\ & \mid & (L) \\ & \mid & \texttt{true} \\ & \mid & \texttt{false} \end{array}$$

5. *(1.5 punto)* Realiza una definición dirigida por la sintaxis (DDS) para evaluar el valor lógico de una sentencia de lógica proposicional[2] usando la gramática G_2. Para ello, se puede emplear un pseudocódigo que emplee únicamente los operadores lógicos del lenguaje C `&&` = AND, `||` = OR y `!` = NOT[1]. Muestra el árbol anotado para la sentencia $\neg\texttt{false} \Rightarrow \texttt{true} \wedge \texttt{false} \Leftrightarrow \texttt{true}$. Indica si la gramática evaluada por la DDS es S-atribuida y/o L-atribuida. Justifica si se puede evaluar al mismo tiempo que el análisis sintáctico ascendente.
6. *(1 punto)* Teniendo en cuenta las precedencias de operadores de la lógica proposicional[2], indica alguna entrada que ejemplifique que la gramática G_2 es ambigua y sugiere una gramática equivalente que no sea ambigua.
7. *(1 punto)* En caso de que sea necesario, haz las transformaciones adecuadas para que la gramática no ambigua obtenida en el apartado anterior pueda ser tratada con el método LL(1). Si no has resuelto el apartado anterior, aplica las transformaciones necesarias a la gramática G_2 inicial (aunque sea ambigua).

[1] Recuerda que $a \Rightarrow b \equiv \neg a \vee b$ y $a \Leftrightarrow b \equiv (\neg a \vee b) \wedge (\neg b \vee a)$.

[2] El orden de precedencia de los operadores, de mayor a menor (en la misma fila se indican operadores de igual precedencia), es:

1. $\neg$
2. $\vee, \wedge$
3. $\Rightarrow, \Leftrightarrow$