EXAMEN DE COMPILADORES (2° Grado en Informática, final junio-2015)

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. Un *autómata finito* reconoce lenguajes regulares, un *autómata de pila* lenguajes libres de contexto, una *máquina de Turing* lenguajes con estructura de frase. Entonces, el análisis sintáctico sistemático de los lenguajes de programación:
   a) Sería posible con *autómatas finitos*.
   b) Sería posible con *autómatas linealmente acotados*.
   c) No sería posible con *máquinas de Turing*.

2. Un *compilador cruzado* es aquel en el que:
   a) El lenguaje destino es el mismo que el lenguaje de implementación.
   b) El código máquina generado no es el de la máquina sobre la que se está ejecutando el compilador.
   c) Se genera código intermedio, que luego es interpretado por una máquina virtual.

3. Dado el siguiente fragmento de un fichero *flex*:

```
%x           comentario
espacios [ \n\t]
%%
"/*"                                    BEGIN(comentario);
<comentario>(.|{espacios})              ;
<comentario>"*/"                        BEGIN(0);printf("Fin de comentario\n");
[*]*\/                                  printf("Reconocimiento erróneo\n");
.*                                      ;
```

   Ante la entrada:

   `/** Esto es un comentario **/`

   el programa en C generado:

   a) Dará como salida `Fin de comentario`.
   b) Dará como salida `Reconocimiento erróneo`.
   c) No dará salida.

4. La afirmación "existen métodos de análisis sintáctico para analizar cualquier lenguaje libre de contexto":
   a) Es cierta.
   b) Es falsa.
   c) Sólo para lenguajes generados por gramáticas propias.

5. Las siguientes reglas corresponden a una gramática G que permite expresar parcialmente las sentencias de un lenguaje de programación. Elige la opción correcta:

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

   a) G no es LL ni LR, puesto que es ambigua.
   b) G es una gramática LL, puesto que $predict(S \rightarrow \texttt{if}\ C\ \texttt{then}\ S\ E) \cap predict(S \rightarrow \texttt{print str}) = \emptyset$, pero no es SLR, puesto que es ambigua.
   c) G no es ambigua.

6. Dada la gramática G anterior, ante la entrada

   `if num then print str else print str`

   el método de análisis descendente predictivo comenzaría realizando los siguientes pasos:

Pila	Entrada	Salida
$ S	if num then print str else print str $	
$ E S then C if	if num then print str else print str $	S → if C then S E
$ E S then C	num then print str else print str $	

   Indicar cuál sería la configuración siguiente, suponiendo que se realiza una recuperación de errores en modo pánico:

   a)

Pila	Entrada	Salida
$ E S then C	print str else print str $	

   b)

Pila	Entrada	Salida
$ E S	print str else print str $	

   c)

Pila	Entrada	Salida
$ E S then	then print str else print str $	

7. Dada la gramática G anterior, para reconocer la sentencia

   `if x == 7 then print ‘‘Sí es 7’’ else print ‘‘No es 7’’`

   en un análisis descendente predictivo no recursivo, la segunda regla con la que se derivaría sería:

   a) $S \rightarrow$ `if` C `then` $S\ E$

   b) $S \rightarrow$ `print str`

   c) $C \rightarrow$ `id == num`

8. Dada la gramática G anterior, para reconocer la sentencia

   `if x == 7 then print ‘‘Sí es 7’’ else print ‘‘No es 7’’`

   en un análisis descendente predictivo recursivo, señalar el número de llamadas que se realizarían al método asociado con el no terminal `S`:

   a) 1

   b) 2

   c) 3

9. Dada la gramática G anterior, el pivote de la forma sentencial

   `if C then print str else print str`

   es:

   a) `if C then` <u>`print str`</u> `else print str`

   b) `if C then print str else` <u>`print str`</u>

   c) `if C then print str` <u>`else print str`</u>

10. Dada la siguiente gramática:

$$
\begin{array}{lcl}
P & \rightarrow & D\ L \\
D & \rightarrow & \lambda \mid D\ T\ id\ ; \\
T & \rightarrow & int \mid float \\
L & \rightarrow & L\ S \mid \lambda \\
S & \rightarrow & print\ id\ ; \mid id\ =\ num\ ;
\end{array}
$$

    ¿cuál sería la primera reducción si se realiza un análisis ascendente de la entrada `int id ; id = num;`?

    a) $D \rightarrow \lambda$

    b) $S \rightarrow id = num$

    c) $T \rightarrow int$

11. La siguiente gramática:

$$\begin{array}{lcl} A & \rightarrow & B\ C\ A \mid a \\ B & \rightarrow & C\ A\ B \mid b \mid \lambda \\ C & \rightarrow & A\ B\ C \mid c \mid \lambda \end{array}$$

   a) Es propia.

   b) No está factorizada.

   c) No es LL ni LR.

12. El conjunto I_0 de la colección LR(1) en la gramática siguiente es:

$$\begin{array}{ll} E & \rightarrow E \texttt{ and } E \mid \texttt{mo (} L \texttt{)} \mid \texttt{true} \mid \texttt{false} \\ L & \rightarrow E \texttt{ , } L \mid E \end{array}$$

   a) $\{[E' \rightarrow \cdot E, \$], [E \rightarrow \cdot E\ and\ E, \$], [E \rightarrow \cdot mo(L), \$], [E \rightarrow \cdot true, \$], [E \rightarrow \cdot false, \$]\}$

   b) $\{[E' \rightarrow \cdot E, \$], [E \rightarrow \cdot E\ and\ E, \$/and], [E \rightarrow \cdot mo(L), \$/and], [E \rightarrow \cdot true, \$/and], [E \rightarrow \cdot false, \$/and]\}$

   c) $\{[E' \rightarrow \cdot E, \$], [E \rightarrow \cdot E\ and\ E, \$], [E \rightarrow \cdot mo(L), \$/mo], [E \rightarrow \cdot true, \$/true], [E \rightarrow \cdot false, \$/false]\}$

13. Dado el siguiente fragmento de un fichero con formato *bison*:

```
%{
int cs;
%}
%token  INICIO FIN CAB COL
%%
inicial : { cs = 0; } INICIO cabecera cola FIN {if (cs) printf("%d\n",$3);else printf("%d\n",$4);};
cabecera : CAB    {$$=2012;cs=0;}
           |      {$$=2013;cs=1;};
cola : COL         {$$=2014;cs=1;}
           |       {$$=2015;cs=0;};
```

   La salida del programa en C generado a partir de él, ante la entrada `INICIO FIN` será:

   a) 2012 o 2014

   b) 2013

   c) 2015

14. Suponiendo que se hace uso de un compilador de C con equivalencia de nombres durante la comprobación de tipos, ¿qué identificadores son equivalentes en estas declaraciones?

```
struct nodo {
        int valor;
        struct nodo *sig;
};
typedef struct nodo celda1;
typedef struct nodo celda2;
celda1 *p;
celda2 *q;
```

   a) `p` y `sig`.

   b) `p` y `q`.

   c) Ningún identificador es equivalente a otro.

15. ¿De qué tipo es la siguiente gramática atribuida?

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

   a) S-atribuida.

   b) L-atribuida y S-atribuida.

   c) Ninguna de las anteriores.

Parte II: PREGUNTAS CORTAS. 10 %.
Dada la siguiente gramática correspondiente a la versión 2.0 del procesador de informes de la agencia internacional de espías[1]:

$$\begin{array}{lcl} S & \rightarrow & pa\ S \mid INF\ S \mid \lambda \\ INF & \rightarrow & es\ COO\ zo\ COL \\ COO & \rightarrow & lat\ lon \\ COL & \rightarrow & es\ COL \mid \lambda \end{array}$$

donde los símbolos tienen el siguiente significado:
pa=país; es=espía-código; zo=zona; lat=latitud; lon=longitud INF=informe; COO=coordenadas; COL=colaboradores.

Se pide identificar y explicar las erratas existentes en la siguiente tabla correspondiente al método de Análisis Sintáctico Descendente LL:

NO TERM	**TERMINAL**						
	pa	**es**	**zo**	**lat**	**lon**	**COO**	$
S	$S \rightarrow pa\ S$						$S \rightarrow \lambda$
INF		$INF \rightarrow es\ COO\ zo\ COL$		$COO \rightarrow lat\ lon$			
COO				$COO \rightarrow lat\ lon$			
COL	$COL \rightarrow \lambda$	$COL \rightarrow es\ COL$					

Parte III: PROBLEMA. 60 %.

Dada la siguiente gramática[1] G con $V_T = \{\texttt{and}, \texttt{mo}, \texttt{true}, \texttt{false}, \texttt{,}, \texttt{(}, \texttt{)}\}$ y $V_N = \{E, L\}$, siendo P:

$$\begin{array}{lcl} E & \rightarrow & E\ \texttt{and}\ E \mid \texttt{mo (} L \texttt{)} \mid \texttt{true} \mid \texttt{false} \\ L & \rightarrow & E\ \texttt{,}\ L \mid E \end{array}$$

donde el operador `mo` funciona como un or múltiple (recibe una lista de operandos y devuelve verdadero si al menos uno de ellos es verdadero), y **el operador `and` es asociativo por la izquierda**, se pide:

1. *(1 punto)* Calcular los conjuntos PRIMERO y SIGUIENTE para cada no terminal de G. Decir, justificando la respuesta, si G es una gramática LL(1). Dar todos los argumentos que consideres que impiden que G sea LL.
2. *(1.5 puntos)* Modificar la gramática G para intentar conseguir una equivalente que sea LL(1). Comprobar si la nueva gramática es LL calculando los conjuntos *predict* para cada regla.
3. *(1.5 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 `true and false true`, haciendo recuperación en modo pánico en caso de error.
5. *(1.5 puntos)*
   a) Realizar una definición dirigida por la sintaxis (DDS) que permita asignar al símbolo inicial de la gramática el resultado de evaluar una expresión derivada de G.
   b) Decorar el árbol de análisis para la entrada `true and mo ( true , false , true ) and true`.
   c) ¿La gramática G es L-Atribuida? ¿Y S-Atribuida? Justifica las respuestas.

[1] Ejercicio basado en un problema de un examen de la asignatura *Compiladores* de la Facultad de Informática de la UPM.