Examen de Traductores, Intérpretes y Compiladores.
Convocatoria ordinaria de Junio de 2003
3er Curso de I.T. Informática de Sistemas.

Universidad de Málaga
Departamento de Lenguajes y Ciencias de la Computación
Campus de Teatinos, 29071 MÁLAGA

TEST DE TEORÍA

1. Sean los siguientes dos tipos de datos:

```
TYPE
    UNO =   RECORD
                a : INTEGER;
                b : ARRAY [1..3] OF CHAR;
                c : ARRAY [4..5] OF CHAR;
            END;
    DOS =   RECORD
                k : INTEGER;
                w : ARRAY [1..5] OF CHAR;
            END;
```

a) Existe equivalencia estructural entre ellos.
b) Existe equivalencia nominal entre ellos.
c) Una variable de tipo UNO ocupa la misma cantidad de memoria que una de tipo DOS.
d) Si **a** se define de tipo UNO, y **b** de tipo DOS, es posible hacer la asignación **a:=b**, pero no al revés debido a la incompatibilidad de tipos.
e) Existe una equivalencia estructural parcial entre ellos.

2. ¿Cuáles de las siguientes características de un lenguaje de programación obligan a que sólo se pueda implementar mediante un intérprete?

a) Permite la recursividad infinita.
b) Permite definir funciones anidadas y evaluarlas.
c) Permite evaluar código fuente en tiempo de ejecución.
d) Permite construir dinámicamente programas que deben ser evaluados.
e) Permite evaluar y depurar las funciones en tiempo de ejecución.

3. ¿Qué circunstancias impiden hacer un análisis sintáctico descendente con retroceso?

a) El que la gramática sea recursiva a la derecha.
b) El que la gramática sea recursiva a la izquierda.
c) El que la gramática posea reglas ε.
d) El que la gramática posea un lenguaje de cardinalidad infinita.
e) El no establecer condiciones de p oda convenientes sobre las ramas infinitas.

4. A la hora de construir un traductor mediante Lex/Yacc que traduzca de un lenguaje de programación inventado por nosotros a lenguaje C, ¿cuáles de las siguientes afirmaciones son falsas?

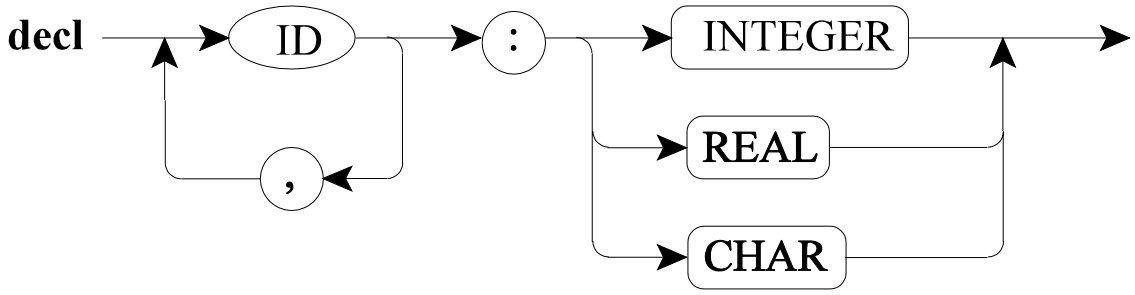
a) Vale cualquier gramática que reconozca el lenguaje de partida.
b) No es posible si el lenguaje de partida permite anidar funciones.
c) El lenguaje de partida no tiene porqué obligar a que exista una función **main()**.
d) No es necesario generar código intermedio.
e) No todo lenguaje de programación puede traducirse de esta manera.

5. Sea la siguiente gramática:

```
L       :       if L then L else L
        |       halt
        ;
```

indicar cuáles de las siguientes afirmaciones es cierta:
a) La gramática es ambigua.
b) Es recursiva a la derecha
c) Genera un número finito de sentencias
d) La gramática no está bien construida y el lenguaje que reconoce es vacío.
e) Esta gramática permite un análisis sintáctico ascendente mediante Lex/Yacc.

6. Sea el siguiente diagrama de Conway:



y la siguiente función:

```
decl(){
        if (token != ID) { printf("Error, esperaba un identificador."); }
        while(token == ID){
                get_token();
                if (token != ',') { printf("Error, esperaba una coma."); break; }
                else get_token();
        }
        if (token != ':') { printf("Error, esperaba dos puntos."); }
        else get_token();
        if (token == INTEGER || token == REAL || token == CHAR)
                get_token();
        else printf("Error, esperaba un tipo: INTEGER, REAL o CHAR);
}
```

responder a la siguiente pregunta: ¿La función recursiva reconoce el mismo lenguaje que el diagrama de Conway? En caso afirmativo justificar la respuesta. En caso negativo poner un contraejemplo justificativo.

7. En base al diagrama de Conway del apartado anterior, indicar cuáles de las siguientes gramáticas aceptan el mismo lenguaje:

a)
```
L       :       ID
        |       L ',' ID
        |       L ':' INTEGER
        |       L ':' REAL
        |       L ':' CHAR
        ;
```

b)
```
L       :       ID ':' INTEGER
        |       ID ':' REAL
        |       ID ':' CHAR
        |       ID ',' L
        ;
```

```
c)  L   :   D ':' T
            ;
    D   :   ID
        |   D ', 'ID
            ;
    T   :   INTEGER
        |   CHAR
        |   REAL
            ;
d)  L   :   D T
            ;
    D   :   ID ':'
        |   ID ', 'D
            ;
    T   :   INTEGER
        |   CHAR
        |   REAL
            ;
e)  L   :   ID T
        |   ID ', 'L
            ;
    T   :   ':' INTEGER
        |   ':' CHAR
        |   ':' REAL
            ;
```

8. En el lenguaje COBOL, las sentencias que forman parte del código de un programa (a partir de la frase reservada **PROCEDURE DIVISION**) deben escribirse a partir de la columna nº 12 y pueden extenderse a lo largo de varias líneas, dejando las 11 columnas anteriores vacías. ¿Cuáles de los siguientes métodos controlaría efectivamente esta restricción del lenguaje emitiendo algún mensaje de error?

a) 11 espacios en blanco al principio de una línea constituye un token que debe aparecer al principio de cada palabra reservada en la gramática del lenguaje.
b) En la parte de código en COBOL, cada vez que nos encontramos con un retorno de carro, a continuación debe haber 11 espacios en blanco como mínimo, o la línea debe estar vacía. En caso contrario se produce un error.
c) Con una expresión en LEX tal como
`[^\␣]{11, 11} ; /* donde el carácter ␣ representa un espacio en blanco */`
d) Con un preprocesador que, una vez llegados a la zona de código extraiga once espacios del principio de cada línea no vacía. Si no los hay, pone un lexema especial indicativo del error y que será reconocido más adelante por el analizador léxico que emitirá el correspondiente mensaje.
e) Con un preprocesador que, una vez llegados a la zona de código, elimine los once primeros caracteres de cada línea. El fichero resultante se le pasaría al analizador léxico.

9. Respecto a las tablas que se utilizan para reconocer gramáticas LALR, cuáles de las siguientes afirmaciones son ciertas:

a) Si una gramática es ambigua, entonces es posible que una misma casilla indique que hay que reducir y desplazar, habiendo de decidirse arbitrariamente por una de las dos.
b) Toda tabla debe tener una columna etiquetada con **$** que representa al EOF.
c) El número de columnas de la tabla fusionada es |T|+|N|+1, donde |T| representa al cardinal del conjunto de terminales y |N| representa al cardinal del conjunto de no terminales.

d) Si una regla es innacesible (nunca se utiliza en la derivación de ninguna de las cadenas que pertenecen al lenguaje), entonces no existe una tabla LALR que reconozca dicho lenguaje.
e) Las reglas ε qe tenga una gramática no se referencian en ninguna casilla de su tabla asociada.

10. Dibujar un único diagrama de Conway correspondiente a la siguiente gramática:

```
w       :       ε
        |       w NUM sent
        ;
sent    :       INPUT ID
        |       GOTO NUM
        |       GOTO k CORRESPONDING ON ID
        ;
k       :       NUM
        |       k ‘,’ NUM
        ;
```

11. La siguiente gramática no es ambigua pero tampoco es LALR(1). Explicar por qué.

```
w       :       ε
        |       w NUM sent
        ;
sent    :       INPUT ID
        |       GOTO NUM
        |       ON ID GOTO k
        ;
k       :       NUM
        |       k NUM
        ;
```

12. Para realizar un adecuado control de tipos en un lenguaje de programación tipado, indicar cuáles de las siguientes respuestas son ciertas:

a) Es necesaria una tabla de símbolos sólo si se desea construir un intérprete.
b) Es necesaria una tabla de símbolos sólo si se desea construir un compilador.
c) Sólo si el lenguaje posee área de definiciones, es necesaria una tabla de símbolos.
d) La tabla de símbolos debe contener el valor de cada variable.
e) El valor de cada identificador en la tabla de símbolos se establece en la primera asignación.

13. Proponer una gramática que no sea LL(1) y justificarlo convenientemente explicado mediante un ejemplo.

14. Con respecto al siguiente esquema de traducción:

```
d       :       ID ‘:’ TIPO     { $$ = $1 = $3; }
        |       ID ‘,’ d        { $$ = $1 = $3; }
        ;
```

¿cuáles de las siguientes afirmaciones son ciertas?

a) En Yacc, ante una entrada **ID, ID : TIPO** el valor del atributo de los dos **ID** es el mismo que el de **TIPO**.
b) Las acciones semánticas no pueden evaluarse en Yacc.
c) En un análisis descendente y ante la entrada **ID, ID : TIPO**, las acciones semánticas pueden ejecutarse a la vez que se realizan las derivaciones.
d) La gramática que se da no es LALR(1).
e) Conceptualmente, el valor del atributo de todo no terminal **d** será siempre el mismo que el

del terminal **TIPO**.

15. Con la tecnología actual, en una compilación incremental de un lenguaje de programación imperativo, ¿cuál suele ser la unidad de recompilación?

a) La sentencia.
b) La línea de código.
c) El procedimiento o función.
d) La clase en un lenguaje orientado a objetos.
e) El fichero fuente.

16. ¿Cuáles de las siguientes afirmaciones son válidas?

a) Un compilador se divide en *back-end* y *front-end* para facilitar la construcción de varios compiladores del mismo lenguaje para varias plataformas.
b) Un compilador cruzado se utiliza para generar código reubicable.
c) Un macroensamblador se diferencia de un ensamblador en que permite crear plantillas en ensamblador mediante un juego de directivas.
d) Un enlazador (*linker*) se utiliza para generar un fichero ejecutable a partir de varios ficheros objeto.
e) Las referencias cruzadas existentes entre ficheros objeto se resuelven en tiempo de ejecución.

17. En cualquier lenguaje de programación, qué sentencia o pseudosentencia debe seguir siempre a un **goto** incondicional:

a) Otro **goto** incondicional.
b) Un **goto** condicional.
c) Un destino de salto (**label**).
d) Cualquier sentencia que no sea otro **goto** incondicional.
e) Ninguna de las anteriores es correcta.

18. ¿Cuáles de las siguientes gramática aceptan la siguiente forma sentencial?

IF cond THEN IF cond THEN sent ELSE sent

```
a) sent :   ID ASIG NUM
        |   IF cond THEN sent ELSE sent
        ;
cond    :   ID OPREL ID
        ;
b) sent :   IF cond THEN sent
        |   IF cond THEN sent ELSE sent
        ;
cond    :   ID OPREL ID
        ;
c) sent :   ID ASIG ID
        |   IF cond THEN sent
        |   IF cond THEN sent ELSE sent
        ;
cond    :   ID OPREL ID
        ;
d) sent :   ID ASIG ID
        |   IF cond THEN sent opcional
        ;
opcional :  ELSE sent
        ;
```

```
cond    :       ID OPREL ID
        ;
e) sent :       IF ID OPREL ID THEN sent
        |       IF ID OPREL ID THEN sent ELSE sent
        |       ID ASIG NUM
        ;
```

19. Respecto a los registros de activación, cuáles de las siguientes afirmaciones son ciertas:

a) Su tamaño se calcula en tiempo de compilación.
b) Utilizan el *stack* para almacenarse.
c) Su principal utilidad es la de permitir la recursividad directa e indirecta.
d) El código para construirlo se genera en tiempo de ejecución.
e) En todos los lenguajes de programación deben contener un puntero al padre.

20. Cuáles de las siguientes afirmaciones suponen diferencias entre un compilador y un intérprete:

a) Un compilador genera código y un intérprete no.
b) Un intérprete no puede permitir la creación de funciones recursivas.
c) Un intérprete no posee fase de optimización.
d) La ejecución de un programa compilado es más eficiente que la de uno interpretado.
e) Los compiladores detectan más errores en ejecución que los intérpretes.

21. ¿Cuál es el significado especial del carácter **[** en Lex?

a) Es equivalente a al carácter especial **(**.
b) Es equivalente al carácter especial **{**.
c) Es equivalente a los caracteres especiales **(** y **{**.
d) Indica opcionalidad entre los caracteres que agrupa junto a **]**.
e) Ninguna de las anteriores es correcta.

22. ¿Cuáles de las siguientes afirmaciones es cierta con respecto a Yacc?

a) Yacc no posee mecanismo alguno para recuperarse de un error sintáctico.
b) Por defecto, cuando Yacc encuentra un error sintáctico se detiene.
c) Yacc puede corregir los errores sintácticos mediante el correcto uso de la palabra **error**.
d) La función **yyerror()** se utiliza para corregir los errores sintácticos.
e) La palabra reservada **error** siempre debería ir seguida de un *token* de seguridad.

23. El contenido de una tabla de símbolos puede variar de un lenguaje fuente a otro, incluso de una implementación a otra, pero ¿qué debe almacenarse de manera obligatoria, siempre?:

a) Las variables del programa.
b) Los tipos definidos por el usuario.
c) Las constantes utilizadas.
d) Las palabras reservadas.
e) Las variables temporales.

24. Cuando nuestro compilador encuentra la declaración de una función:

a) Debe reservar un trozo de memoria dinámica del tamaño de su registro de activación.
b) Debe meter en la tabla de símbolos su nombre, su tipo y los tipos de todos sus parámetros formales por orden.
c) Debe controlar que no haya ninguna variable global con el mismo nombre que un parámetro formal.

d) Debe controlar que los tipos de los parámetros reales coincidan con los de los formales.
e) Ninguna de las anteriores es correcta.

25. ¿Qué tipo de gramáticas no reconoce Yacc?
a) Las ascendentes.
b) Las descendentes.
c) Las recursivas por la izquierda.
d) Las recursivas por la derecha.
e) Ninguna de las anteriores es correcta.

TEST DE PRÁCTICA

A continuación se van a realizar algunas modificaciones sobre el ejemplo de generación de código de tercetos visto en clase:
I) Se añaden las siguientes reglas para permitir la utilización de la sentencia FOR:

```
sent  :  FOR ID ASIG expr
                {
①                   printf("\t%s = %s\n", $2, $4);
                }
         arribaAbajo expr opcionalFor DO
                {
                    nueva_etq($1.etq_inicio);
                    nueva_etq($1.etq_final);
                    if (! $6) /*Descendente*/{
                        printf("\t%s=-%s\n", $8, $8);
                    }
                    printf("label %s\n", $1.etq_inicio);
                    if ($6) /*Ascendente*/{
                        printf("\tif %s > %s goto %s\n", $2, $7, $1.etq_final);
                    } else {
                        printf("\tif %s < %s goto %s\n", $2, $7, $1.etq_final);
                    }
                 }
            sent ';' FIN FOR
                {
                    printf("\t%s=%s+%s\n", $2, $2, $8);
                    printf("\tgoto %s\n", $1.etq_inicio);
                    printf("label %s\n", $1.etq_final);
                 }
      ;
arribaAbajo :  TO       { $$ = 1; }
            |  DOWNTO   { $$ = 0; }
            ;
opcionalFor :  /* Epsilon */
                {
                    nueva_var($$);
                    printf("\t%s=1\n", $$);
                }
            |  STEP expr    { strcpy($$, $2); }
            ;
```

II) Supongamos que NO se quiere que las condiciones funcionen por la técnica del cortocircuito. Para ello, a continuación se muestra un ejemplo de una condición primitiva, y cómo se solucionaría la condición compuesta mediante el AND:

```
cond  :  expr '>' expr
                {
                  char etq[21];
```

```
                    nueva_etq(etq);
                    nueva_var($$);
                    printf("\t%s = 1\n", $$);
                    printf("\tif %s > %s goto %s\n", $1, $3, etq);
                    printf("\t%s = 0\n", $$);
                    printf("label %s\n", etq);
                   }
        |   NOT cond
                   {

                   }
        |   cond AND cond    {
                    nueva_var($$);
                    printf("\t%s = %s * %s\n", $$, $1, $3);
                   }
        |   cond OR cond
                  {

                   }
        |   '(' cond ')'
                   {
                    strcpy($$, $2);
                   }
        ;
```

III) Además, para que todo esto funcione, es necesario hacer unas pequeñas modificaciones en el %union y las correspondientes asignaciones de atributos, que se muestran a continuación:

```
%{
typedef struct _doble_etq
        {
            char    etq_inicio[21],
                    etq_final[21];
        } doble_etq;
typedef struct _datos_case
        {
            char    etq_final[21],
                    etq_siguiente[21];
            char    variable_expr[21];
        } datos_case;
%}

%union
{
        int numero;
        char variable_aux[21];
        char etiqueta_aux[21];
        datos_case bloque_case;
        doble_etq  bloque_for;
        short int ascendente;
}

%token <numero> NUMERO
%token <variable_aux> ID
%token <etiqueta_aux> IF WHILE REPEAT
%token <bloque_for> FOR
%token ASIG THEN ELSE FIN DO UNTIL CASE OF CASO OTHERWISE TO DOWNTO STEP
%token MAI MEI DIF

%type <variable_aux> expr cond opcionalFor
%type <bloque_case> inicio_case
%type <ascendente> arribaAbajo
```

En base a todo lo anterior, responder a las siguientes preguntas:

26. ¿Cuál es el cometido de la estructura **doble_etq** (marcar sólo una)?
a) Almacenar el final de la instrucción **FOR**.
b) Almacenar la etiqueta de inicio de la instrucción **FOR**.
c) Permitir el anidamiento de instrucciones **FOR**.
d) **doble_etq** no es una estructura sino un atributo.
e) Permitir el anidamiento de instrucciones de control de flujo cualesquiera.

27. Con respecto a las acciones semánticas de la sentencia **FOR** (marcar una, ninguna o varias):
a) La expresión que sigue a **TO** ó **DOWNTO** se evalúa cada vez que el bucle se pregunta si ejecutar un nuevo ciclo.
b) La expresión que sigue a **STEP** se evalúa cada vez que el bucle se pregunta si ejecutar un nuevo ciclo.
c) Las expresiones de la cabecera del **FOR** se evalúan una sola vez.
d) Se sabe en tiempo de compilación si el bucle es ascendente o descendente.
e) Tal y como está hecho, el cuerpo del **FOR** se ejecuta, al menos, una vez.

28. El cuerpo de un bucle de la forma:
FOR i := 1 **TO** i+7 **STEP** 1 **DO** i := i+1; **FIN FOR**;
a) No para de ejecutarse.
b) Se ejecuta 8 veces.
c) Se ejecuta 7 veces.
d) Se ejecuta 4 veces.
e) No se ejecuta nunca.

29. Si el bucle **FOR** no tiene parte **STEP expr** entonces:
a) Se asume que el incremento es +/-1 en función de si el bucle es ascendente/descendente respectivamente.
b) Se crea una expresión virtual y se evalúa durante la generación del código.
c) El bucle no funciona.
d) Se asume que el incremento va en función de si la diferencia entre las dos expresiones de la cabecera del **FOR** es positiva o negativa.
e) Se asume que el incremento es 1.

30. El bloque de código marcado como ①, ¿en cuántos otros sitios puede ponerse sin que cambie el comportamiento del **FOR**?
a) En ninguno.
b) En uno.
c) En dos.
d) En tres.
e) En cuatro o más.

31. Indicar EXACTAMENTE en qué se transformaría el siguiente bucle **FOR**:
FOR cont := aux * 2 **DOWNTO** aux **STEP** 2 **DO** beta := beta * 2; **FIN FOR**;
(Esta pregunta se valorará como 0 ó 2. En ningún caso restará puntuación).

32. Indicar las acciones semánticas que habría que incluir en la regla de producción del **IF** (sin **ELSE**) (Esta pregunta se valorará como 0 ó 2. En ningún caso restará puntuación):

sent : **IF** cond **THEN** sent ';' **FIN IF**
;

para que se comporte como se espera de ella.

33. Indicar las acciones semánticas que habría que incluir en la regla de producción de la condición compuesta por el **OR**.

34. Indicar las acciones semánticas que habría que incluir en la regla de producción de la condición compuesta por el **NOT**.

35. ¿Qué diferencia hay entre evaluar una condición con o sin cortocircuito?

a) Con cortocircuito tarda menos.
b) Con cortocircuito se facilita el control de errores. Ej.:
IF a>0 AND b/a > 5 THEN ....
c) Sin cortocircuito no tarda menos.
d) Sin cortocircuito se genera más código intermedio.
e) Sin cortocircuito puede dar un resultado diferente a con cortocircuito.

Normativa de cumplimentación:

I) Las preguntas son de respuesta única (marcadas como ❍ en la hoja de respuestas), de respuesta múltiple (marcadas como □ en la hoja de respuestas), o de respuesta libre (a responder al dorso).
II) Preguntas de respuesta única. Respondidas correctamente cuentan 1 punto. Respondidas incorrectamente restan 0,20 puntos. Las no respondidas no cuentan. Si se selecciona más de una respuesta se considerará errónea.
III) Preguntas de respuesta múltiple. Para cada opción suministrada hay que indicar si es cierta o falsa. Cada opción acertada suma 0,20 puntos. Cada opción equivocada resta 0,20 puntos. Las opciones no respondidas no cuentan.
IV) Las preguntas de respuesta libre deben respondarse en los recuadros a tal efecto en el dorso de cada hoja de respuestas. Se evaluarán entre 0 y 1; excepto las preguntas 31 y 32 que se evaluarán como 0 ó 2. En ningún caso restarán.

Normativa de evaluación:

V) Las 25 preguntas de la parte teórica suman 25 puntos, que se normalizará a un valor entre 0 y 5.
VI) Las 10 preguntas de la parte práctica suman 12 puntos (las preguntas 31 y 32 cuentan doble), que se normalizará a un valor entre 0 y 5.
VII) La suma de los valores anteriores dará lugar a una calificación normalizada entre 0 y 10.
VIII) A la calificación normalizada se le sumará la nota del test de Lex, obteniéndose la calificación final.
IX) Se considerará APROBADO todo alumno cuya calificación final supere el 4,0.