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

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

Apellidos, Nombre: _____

Calificación: _____

Tipo A TEST DE TEORÍA

1.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	2.- V F a) ☐ ☐ b) ☐ ☐ c) ☐ ☐ d) ☐ ☐ e) ☐ ☐	3.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	4.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	5.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍
6.- V F a) ☐ ☐ b) ☐ ☐ c) ☐ ☐ d) ☐ ☐ e) ☐ ☐		8.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	9.- Ver al dorso	10.- V F a) ☐ ☐ b) ☐ ☐ c) ☐ ☐ d) ☐ ☐ e) ☐ ☐
11.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	12.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	13.- Ver al dorso	14.- V F a) ☐ ☐ b) ☐ ☐ c) ☐ ☐ d) ☐ ☐ e) ☐ ☐	15.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍
16.- V F a) ☐ ☐ b) ☐ ☐ c) ☐ ☐ d) ☐ ☐ e) ☐ ☐	17.- Ver al dorso	18.- V F a) ☐ ☐ b) ☐ ☐ c) ☐ ☐ d) ☐ ☐ e) ☐ ☐	19.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	20.- V F a) ☐ ☐ b) ☐ ☐ c) ☐ ☐ d) ☐ ☐ e) ☐ ☐
21.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	22.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	23.- Ver al dorso	24.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	25.- V F a) ☐ ☐ b) ☐ ☐ c) ☐ ☐ d) ☐ ☐ e) ☐ ☐

9.-
13.-
17.-
23.-

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

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

Apellidos, Nombre: _____

Calificación: _____

Tipo A TEST DE PRÁCTICA

26.- Ver al dorso	27.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	28.- Ver al dorso	29.- Ver al dorso	30.- V F a) ☐ ☐ b) ☐ ☐ c) ☐ ☐ d) ☐ ☐ e) ☐ ☐
31.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	32.- Ver al dorso	33.- V F a) ☐ ☐ b) ☐ ☐ c) ☐ ☐ d) ☐ ☐ e) ☐ ☐	34.- a) ❍ b) ❍ c) ❍ d) ❍ e) ❍	35.- Ver al dorso

26.-
28.-
29.-
32.-
35.-

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

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

TEST DE TEORÍA

1 ¿Por qué es interesante que un compilador genere módulos objeto y no el ejecutable directamente?

a) Para facilitar la compilación separada.
b) Para facilitar la construcción del *back-end* del compilador.
c) Para disminuir el tiempo de compilación de cada módulo fuente.
d) Para que el enlazador (*linker*) se encargue de la optimización de código.
e) Para facilitar la portabilidad del código.

2 Con respecto a un descompilador, ¿cuáles de las siguientes afirmaciones son ciertas?

a) Un macrodesensamblador es un tipo de descompilador.
b) Un desensamblador es un tipo de descompilador.
c) Todos los lenguajes de programación se pueden descompilar correctamente.
d) Si cualquier trozo de código máquina se pudiera descompilar a cualquier lenguaje de programación, entonces sería posible la construcción de cualquier conversor fuente-fuente.
e) No todo trozo de código máquina es susceptible de ser desensamblado.

3 Respecto a la optimización que debe realizar un compilador en el código generado:

a) Es una fase exclusiva del *front-end*.
b) Es una fase exclusiva del *back-end*.
c) Es una fase no opcional del compilador.
d) Es una fase de síntesis.
e) Ninguna de las anteriores es correcta.

4 El tratamiento de errores de un compilador:

a) Debe intentar corregir el error.
b) Es algo opcional.
c) Se considera una fase del compilador.
d) Genera errores sólo en las fases de análisis.
e) Debe proporcionar mensajes explicativos.

5 ¿Cuál de los siguientes lenguajes es admitido por Yacc (las minúsculas son los terminales)?

a)
```
W : Bad
  | Mal
  ;
B : q
  ;
M : q
  ;
```

b)
```
L : aBa
  | Mao
  ;
B : q
  ;
```

c) ε : Bad
| Bal
;
B : q
;

d) L : aLa
| Loa
| LaR
;
R : q
;

e) L : Lab
| Labab
| ε
;

6 Dada la siguiente gramática:

L : iwMF
;
F : wt
;
M : ε
| MocL
;

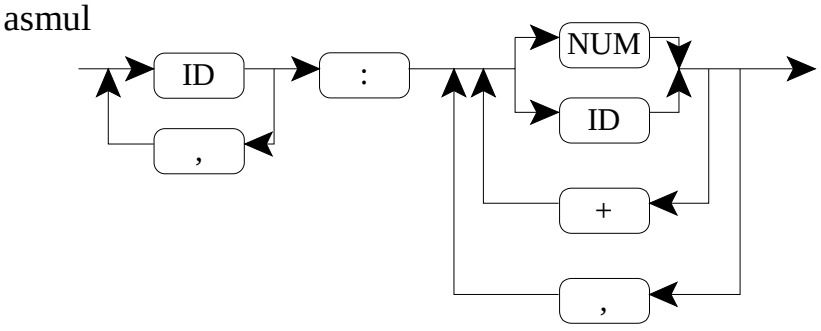
¿cuáles de las siguientes formas sentenciales posee árbol sintáctico?

a) iwocociwiwwt
b) iwiwwtwt
c) iwociwwtwt
d) iwociwwtociwwtwt
e) iwociwwtociwociwwtwtwt

8 ¿Cuál de las siguientes afirmaciones es cierta?

a)Los diagramas de Conway permiten expresar más gramáticas que las reglas de producción.
b)La notación BNF es el mecanismo que menos gramáticas permite expresar.
c)Las reglas de producción sólo permiten expresar lenguajes infinitos si poseen alguna regla recursiva (directa o indirectamente).
d) Cuando un diagrama de Conway se pasa a funciones recursivas se obtiene un analizador sintáctico ascendente.
e) Ninguna de las anteriores es cierta.

9 Dado el siguiente diagrama de Conway construir una gramática mediante reglas de producción que reconozca el mismo lenguaje:



10 ¿Cuáles de los siguientes caracteres permiten hacer uso de la sensibilidad al contexto en Lex?

a) $
b) &
c) /
d) \
e) ^

11 ¿Cuál de las siguientes variables es proporcionada por Lex?

a) input
b) yylval
c) yyinput
d) yytext
e) yystype

12 Dado el siguiente programa Lex:

```
%start UNO
%%
“hola”  { printf(“HOLA”); }
“uno”   { BEGIN UNO; }
<UNO>”hola”          { printf(“HOLA en UNO”); }
<UNO>”adios”         { BEGIN 0; }
<UNO>.               { printf(“.”); }
.|\n      {printf(“caracter”); }
```

ante la entrada: **hola uno hola alfa adios beta** la salida de Lex sería:

a) HOLAcaracterHOLA en UNO.....caractercaractercaractercaractercaracter
b) HOLAcaracter.HOLA.....caractercaractercaractercaractercaracter
c)HOLAcaracter.HOLA alfa caractercaractercaractercaractercaracter
d) HOLAcaracter.HOLA en UNO alfa caractercaractercaractercaractercaracter
e) HOLA en UNO.HOLA en UNO.....caractercaractercaractercaractercaracter

13 Escribir una expresión regular que permita escribir fechas junto con horas opcionales de la forma:

DD/MM/AA:HH:MM:SS:mmm

DD, MM y AA representan el día, el mes y el año respectivamente con uno o dos dígitos; el resto es opcional y HH, MM y SS representan las horas, minutos y segundos con dos dígitos obligatorios. Si se especifica la hora puede hacerse con o sin la parte mmm que representa los milisegundos con tres dígitos obligatoriamente.

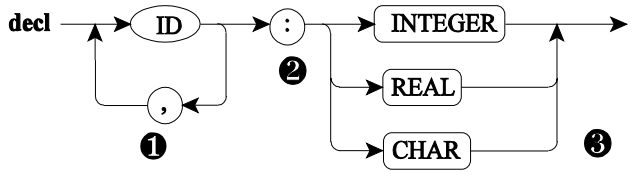
14 Con respecto a la generación de código intermedio, ¿cuáles de las siguientes afirmaciones es cierta?

a) El código de tercetos aprovecha todas las características de la plataforma final.
b) Según las necesidades, el constructor del compilador se puede inventar los tercetos que crea oportunos.
c) Los tercetos se llaman así porque permiten un máximo de tres direcciones.
d) El código intermedio siempre debe ser de tercetos.
e) La generación de código intermedio es una fase obligatorio en todo compilador.

15 Con respecto al código de tercetos, ¿cuál de las siguientes afirmaciones es cierta?
a) Inmediatamente detrás de un pseudoterceto **label** debe haber un terceto **goto**.
b) Inmediatamente detrás de un terceto **goto** debe haber un pseudoterceto **label**.
c) Inmediatamente detrás de un pseudoterceto **label** no puede haber un terceto **goto**.
d) Inmediatamente detrás de un pseudoterceto **label** no puede haber otro pseudoterceto **label**.
e) Ninguna de las anteriores es cierta.

16 En C, el código generado para las condiciones:
a) Utiliza la técnica de cortocircuito si se emplea: && y ||.
b) Una condición se considera una expresión.
c) Debe depender de en qué sentencia aparezca la condición.
d) Es el mismo para todas las condiciones.
e) Ninguna de las anteriores es cierta.

17 Sea el siguiente diagrama de Conway:



y la siguiente función recursiva:

```
decl(){
        if (token != ID) { printf("Error, esperaba un identificador."); }
        if (token == ','){
                while(token == ','){
                        get_token();
                        if (token != ID) { printf("Error, esperaba un identificador."); break; }
                        else get_token();
                }
        }else{
                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 indicar qué bloque de código se corresponde con cada bloque del diagrama de Conway (los bloques están marcados como ❶, ❷ y ❸. En caso negativo poner un contraejemplo justificativo e indicar las modificaciones necesarias que habría que hacer para corregir la disparidad.

18 ¿Qué es la tabla de símbolos?
a) Una lista enlazada para almacenar información acerca de los identificadores de usuario.
b) Una estructura de datos válida sólo en tiempo de compilación.
c) Una estructura de datos válida mientras sea necesaria, ya sea en ejecución y/o en compilación.
d) Una estructura de datos de apoyo para guardar información acerca de los identificadores de usuario.
e) Una estructura de datos de apoyo para almacenar el tipo, valor, y dirección de almacenamiento de los identificadores de usuario.

19 En general, en Yacc, cuando interesa asociar a un terminal o no terminal más de un atributo, ¿cuál es la solución?

a) Asociarle un array con los atributos.
b) Asociarle un atributo de tipo puntero.
c) Asociarle un registro con los atributos.
d) Asociarle un enumerado con los atributos.
e) Ninguna de las anteriores es correcta.

20 El **%union** de Yacc:

a) Le asocia un tipo a YYSTYPE.
b) Le asocia su atributo a cada terminal y/o no terminal.
c) Permite asignarle más un atributo a cada terminal y/o no terminal.
d) Es una estructura de tipo **union** del lenguaje C.
e) Define una estructura que acompaña a cada par (símbolo, estado) en la pila de Yacc.

21 Sea el siguiente esquema de traducción:

```
%{
#define YYSTYPE int
%}
%%
A       :       B {$$=$1;} c {printf("%d", $2); }
        ;
B       :       dd      { $$=1; }
        |       Bdd     { $$=$1+1; }
        ;
```

¿cómo se comporta ante la entrada **ddddc**?

a) No funciona, porque en una regla intermedia no se puede utilizar el atributo del antecedente.
b) Saca por pantalla un **2**.
c) Saca un valor indefinido, puesto que no se sabe el valor del atributo **c**.
d) Saca por pantalla un **4**.
e) No funciona, puesto que B tiene dos acciones semánticas que referencian a $$.

22 ¿Por qué los lenguajes actuales suelen ser fuertemente tipados?

a) Para capturar errores semánticos en tiempo de compilación.
b) Para capturar errores lógicos en tiempo de compilación.
c) Para facilitar la escritura de programas.
d) Para agilizar la fase de compilación.
e) Para facilitar la construcción de tipos definidos por el usuario.

23 La siguiente gramática no admite un análisis descendente LL(1), pero sí LR(1), explicar por qué:

```
L       :       aBc
        |       aDe
        ;
B       :       b
        ;
D       :       b
        ;
```

24 En la construcción de un lenguaje de programación que posee una zona de declaración de variables de usuario:

a) Es conveniente que sea Lex quien inserte dichos identificadores en la tabla de símbolos.
b) Es conveniente que sea Yacc quien inserte dichos identificadores en la tabla de símbolos.
c) Da igual quien inserte en la tabla de símbolos, ya sea Lex o Yacc.
d) Es conveniente insertar un identificador en la tabla de símbolos la primera vez que aparezca en una expresión o en una asignación.
e) En tal caso, el lenguaje de programación debe poseer el concepto de tipo de datos.

25 Un programa escrito en Yacc que sólo realiza el análisis sintáctico:

a) Es siempre independiente de la codificación del juego de caracteres.
b) Si utiliza *tokens* que sean caracteres entre comillas simples entonces no es independiente de la codificación del juego de caracteres.
c) Es siempre independiente de la plataforma.
d) Es independiente de la plataforma sólo si la gramática no es ambigua.
e) No tiene porqué indicar los atributos de los terminales y/o no terminales que lo pudieran necesitar.

TEST DE PRÁCTICA

Se desea hacer un lenguaje de programación con las siguientes características, y con objeto de generar código de tercetos:

a) Permitir el uso de variables de tipo entero, sin necesidad de declaraciones.
b) Controlar que la longitud de cada identificador no exceda los 19 caracteres. Si excede dicha longitud, se truncará a 20 caracteres y se sacará un mensaje por pantalla que avise del truncamiento.
c) Permitir asignaciones simples a un identificador. En una expresión sólo intervienen +, - y menos unario.
d) Optimizar las operaciones de sumar 1 y restar 1 a un identificador.
e) Permitir la operación GO TO. A este efecto, se tienen tres tipos de operaciones GOTO:
    e.1) Operación GO TO simple. Su formato será **GO TO Etiqueta**.
    e.2) Operación GO TO sobre expresión. Produce una secuencia de saltos, en función de una expresión. Es de la forma **ON expr GO TO Etq1, Etq2, Etq2, Etq3, ...**.
        Si el valor de expr es 1, se salta a Etq1. Si el valor de expr es 2 se salta a Etq2. Si es 3, a Etq3, etc. Si el valor de la expresión excede el número de etiquetas, no se salta a ningún lado, sino que se continúa el flujo de ejecución.
    e.3) Operación GO TO condicional. Produce un salto si y sólo si cierta expresión es distinta de 0. Es de la forma **IF expr GO TO Etiqueta**.
f) Las etiquetas se definen dentro del código a través de **LABEL Etiqueta:**.
g) Un número entero sólo puede intervenir en el terceto de la forma: **id := número**. A pesar de ello, y por motivos de eficiencia también se permite el terceto de la forma: **id := id +1**, y el de la forma **id := id - 1**.
h) Se pueden emplear cuantas variables temporales se considere necesario.
i) Las etiquetas que incumplan alguno de los controles siguientes, se sacarán por pantalla, al final del análisis, con el correspondiente mensaje de advertencia o error:
    i.1) Controlar que toda etiqueta a la que se salta, existe.
    i.2) Controlar que toda etiqueta definida, es usada por algún GO TO.

Este ejercicio se corresponde con el examen de Junio del 97 de la Informática de Sistemas en la UMA, y en el **se pedía** (esto **no** es lo que se pide ahora):

1) Hacer el programa LEX correspondiente, que efectúa el punto b).
2) Hacer el programa YACC correspondiente, que hace los puntos a) y del c) al h).

3) Implementar la función *ts_comprobar()*, que efectúa el punto i).

siendo la solución parcial a este ejercicio la que se muestra a continuación.

Fichero Tabsimb97.c

```
#include <stdlib.h>
#include <stdio.h>
typedef struct _simbolo
   {
    struct _simbolo * sig;
    char nombre[21];
    short int definida;
    short int usada;
   } simbolo;

simbolo * ts_crear()
{
 return NULL;
};

void ts_insertar(p_t, s)
simbolo * * p_t;
simbolo * s;
{
 s->sig = (*p_t);
 (*p_t) = s;
}

simbolo * ts_buscar(t, nombre)
simbolo * t;
char nombre[21];
{
 while( (t != NULL) && (strcmp(nombre, t->nombre)) )
     t = t->sig;
 return (t);
}

void ts_comprobar(t)
simbolo * t;
{









}
```

Fichero Jun97l.lex

```
%%

ON      { return ON; }
"GO TO" { return GOTO; }
INC     { return INC; }
DEC     { return DEC; }
IF      { return IF; }
LABEL   { return LABEL; }
":="    { return ASIG; }
[A-Za-z_]                |
```

```
[A-Za-z_][A-Za-z0-9_]{1,20} {
                        strcpy(yylval.nombre, yytext);
                        return ID;
                       }
[A-Za-z_][A-Za-z0-9_]* {
                      printf("Identificador demasiado largo.\n");
                      printf("%s truncado a ", yytext);
                      yytext[20] = 0;
                      printf("%s.\n", yytext);
                      strcpy(yylval.nombre, yytext);
                      return ID;
                     }
[0-9]+                 {
                      yylval.valor = atoi(yytext);
                      return NUM;
                     }
[ \t\n]+    { ; }
.           { return yytext[0]; }
```

Fichero Jun97y.yac

```
%{
#include <stdlib.h>
#include "tabsim97.c"

/* Declaracion de variables globales. */
char var_goto[21];
int longitud_lista;
simbolo * t;

simbolo * IOB(char * nombre) {
  simbolo * aux;
    aux = ts_buscar(t, nombre);
    if (aux == NULL) {
        aux = (simbolo *) malloc(sizeof(simbolo));
        strcpy(aux->nombre, nombre);
        aux->definida = 0;
        aux->usada = 0;
        ts_insertar(&t, aux);
    }
    return aux;
}
%}

%union {
    char nombre[21];
    int valor;
    char var_aux[21];
}




%left '+' '-'
%right MENOS_UNARIO
%start prog

%%

prog : /* Epsilon */
```

```
    | prog LABEL ID ':' {


                    }
    | prog error    ':' { yyerrok; }
    | prog sent  ';'
    | prog error ';' { yyerrok; }
    ;
sent : ID ASIG expr    {
                    printf("\t%s = %s\n", $1, $3);
                   }
    | INC '(' ID ')'  {
                    printf("\t%s = %s + 1\n", $3, $3);
                   }
    | DEC '(' ID ')'  {
                    printf("\t%s = %s - 1\n", $3, $3);
                   }
    | IF expr GOTO ID {
                    IOB($4)->usada = 1;
                    printf("\tif %s != 0 goto %s\n", $2, $4);
                   }
    | GOTO ID        {
                    IOB($2)->usada = 1;
                    printf("\tgoto %s\n", $2);
                   }
    | ON expr GOTO
                   {
                    strcpy(var_goto, $2);
                   }
         lista_etiquetas
    ;
lista_etiquetas : ID                {
                                IOB($1)->usada = 1;
                                longitud_lista = 1;
                                printf("\tif %s = %d goto %s\n", var_goto,
                                                longitud_lista,
                                                $1);
                                longitud_lista++;
                               }
            | lista_etiquetas ',' ID {
                                IOB($3)->usada = 1;
                                printf("\tif %s = %d goto %s\n", var_goto,
                                                longitud_lista,
                                                $3);
                                longitud_lista++;
                               }
            ;
expr : ID          {
                  strcpy($$, $1);
                 }
    | NUM          {
                  nueva_var($$);
                  printf("\t%s = %d\n", $$, $1);
                 }
    | expr '+' expr {
                  nueva_var($$);
                  printf("\t%s = %s + %s\n", $$, $1, $3);
                 }
    | expr '-' expr {
                  nueva_var($$);
                  printf("\t%s = %s - %s\n", $$, $1, $3);
                 }
```

```
    | '-' expr  %prec MENOS_UNARIO
                {
                  nueva_var($$);
                  printf("\t%s = - %s\n", $$, $2);
                }
    | '(' expr ')'  {
                  strcpy($$, $2);
                }
    ;

%%

#include "exa97l.c"

void main() {
  t = ts_crear();
  yyparse();
  ts_comprobar(t);
}

void yyerror(char * s) {
  printf("%s\n", s);
}

void nueva_var(char * var_aux) {
  static actual = 1;
    sprintf(var_aux, "var%d", actual++);
}
```

26 Escribir en C el contenido de la función *ts_comprobar()*.

27 Respecto a la función IOB:

a) Siempre inserta identificadores en la tabla de símbolos.
b) Devuelve un puntero a un símbolo cuyo nombre se le pasa como parámetro.
c) Si es invocada dos veces con el mismo nombre de variable como parámetro, lo meterá sólo una vez en la tabla de símbolos.
d) Puesto que esto no es un intérprete, la tabla de símbolos almacena las variables de usuario, pero no sus valores.
e) Ninguna de las anteriores es cierta.

28 Escribir los %token y %type que faltan en el programa Yacc.

29 Usando el lenguaje definido, hacer un programa que calcule el producto de dos enteros X e Y, e indicar el código de tercetos que se debe generar.

30 La solución dada para generar el código de **ON expr GOTO ...**, ¿funciona correctamente?

a) Sí, ya que no se permiten los GOTOs anidados.
b) No, ya que se utiliza una variable global.
c) No, ya que el valor de la expresión se debiera guardar como atributo de algún *token*.
d) Sí, puesto que **lista_etiquetas** es recursiva a la izquierda.
e) Sí, pero habría sido más fácil que la sintaxis de esta sentencia hubiera sido **GOTO etq1, etq2, ... ON expr**.

31 ¿Se meten en algún momento las variables en la tabla de símbolos?
a) No, lo cual constituye un error.
b) No, ya que no es necesario almacenar su valor en tiempo de compilación.
c) Sí, en el momento en que aparecen por vez primera.
d) Sí, ya que así lo exige el control de tipos.
e) Ninguna de las anteriores es correcta.

32 Escribir el código asociado a la regla:

```
prog    : prog LABEL ID ':'
        ;
```

33 ¿Por qué el atributo del token **NUM** es del tipo de **valor** y no del tipo de **nombre**?
a) Porque se trata de un número, y no de un identificador.
b) Porque un **int** ocupa menos espacio.
c) Se trata de un error. Así no funciona el programa.
d) Porque es necesario hacer cálculos aritméticos con el valor.
e) Todas las anteriores son correctas.

34 ¿Por qué hay dos reglas de control de errores?
a) Para recuperar errores tanto al final de sentencia como al final de etiqueta.
b) Es una gestión redundante que no interfiere el funcionamiento del analizador generado.
c) Porque la gramática no admite el programa vacío.
d) Una es por si se produce un error en la primera sentencia, y la otra es por si se produce en el resto de sentencias.
e) Ninguna de las anteriores es correcta.

35 ¿Cómo gestiona el programa las variables estaEsUnaVariableDeEjemplo y estaEsUnaVariableDeEjercicio?

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 responderse en los recuadros a tal efecto en el dorso de cada hoja de respuestas. Se evaluarán entre 0 y 1. En ningún caso restarán.
V) La parte teórica constituye el 50% de la nota final. Igual para la práctica.
VI) A la nota final se le sumará la calificación del test de Lex hecho en clase.