PROCESADORES DE LENGUAJES Examen (Ev. Continua)

Nombre: _____ Apellidos: _____

Ev. continua: ¿Asiste regularmente a clase? (Marcar con una X) SÍ: ___ NO: ___
Ev. continua: ¿Estudia regularmente la asignatura? (Marcar con una X) SÍ: ___ NO: ___

Análisis Sintáctico. Lenguaje ASSERTION (2 puntos)

ASSERTION es un lenguaje de programación. Sus programas pueden incluir asertos. Un aserto es una condición lógica construida con las siguientes reglas:
(1) Si f y g son expresiones enteras entonces f == g, f != g, f > g y f < g son asertos.

(2) Las constantes T y F son asertos.

(3) Si f y g son asertos también lo son su conjunción (ej. f & g), disyunción (ej. f | g) y negación (ej. –f, -g).

(4) Si f es un aserto también lo es f entre paréntesis (ej. (f)).

Sintácticamente, un programa ASSERTION está compuesto por una declaración de variables y una secuencia de instrucciones. Las instrucciones pueden ser (a) asignaciones, (b) instrucciones condicionales, (c) iteraciones y (d) lectura de valor para una variable. Un programa ASSERTION puede incluir un aserto entre llaves en cualquier punto de su secuencia de instrucciones.

Ejemplo de programa ASSERTION:

```
PROGRAMA
VARIABLES x, y, z;
LEER(x);
{ x > 0 }
z = 1;
{ z < x/2 | z == x/2 }
y = 1;
MIENTRAS (y < x/2 | y == x/2) HACER
   SI ((x/y)*y == x) ENTONCES
      z = y;
   FINSI
   y = y + 1;
{ (x/z) * z == x & (z == 1 | z > 1) & (z < x/2 | z == x/2) }
FINMIENTRAS
{ (x/z) * z == x & (z == 1 | z > 1) & (z < x/2 | z == x/2) }
```

SE PIDE: Diseño de gramática independiente de contexto para el lenguaje ASSERTION

IMPORTANTE Restricción: responda en la cara posterior de esta página (use folios para borrador y dicha cara para la versión definitiva). Sea claro y legible. Evite tachaduras.

```
// DISEÑO
// Gramática independiente de contexto para el lenguaje
// ASSERTION

programa : PROGRAMA variables instrucciones

variables: VARIABLES vars PyC

vars : VAR COMA vars
     | VAR

instrucciones : (instruccion | aserto)*

instruccion : lectura | asignacion | condicional | iteracion

lectura : LEER PA VAR PC PyC

asignacion : VAR ASIG expr PyC

condicional : SI PA cond PC ENTONCES instrucciones FINSI

iteracion : MIENTRAS PA cond PC HACER instrucciones FINMIENTRAS

aserto : LLA cond LLC

expr: expr (MAS|MENOS|POR|DIV) expr
    | NUMERO
    | VAR
    | PA expr PC

cond: cond (O|Y) cond
    | expr IGUAL expr
    | expr DISTINTO expr
    | expr MAYOR expr
    | expr MENOR expr
    | CIERTO
    | FALSO
    | MENOS cond
    | PA cond PC
```

PROCESADORES DE LENGUAJES Examen (Ev. Continua)

Nombre: _____ Apellidos: _____

Ev. continua: ¿Asiste regularmente a clase? (Marcar con una X) SÍ: ___ NO: ___
Ev. continua: ¿Estudia regularmente la asignatura? (Marcar con una X) SÍ: ___ NO: ___

Análisis Semántico. Lenguaje LEC (3 puntos)

El siguiente ejemplo muestra un programa escrito en el lenguaje de expresiones contextualizadas (LEC):

```
CONTEXTO 1
   x = 5;
   y = 7;
   z = 1;
   x = 9;

CONTEXTO 3
   x = 9;
   z = 3;

DEFECTO  2

EXPRESIONES
   (x{10} - y{2}) * z;
   (x{1} + y{1}) * z{3} + y{3};
   (x - y) * m;
```

Un programa LEC es un conjunto de expresiones aritméticas contextualizadas. Los contextos son secciones del programa en las que se inicializan las variables con constantes numéricas (ej. `z = 1`). Un programa LEC puede tener uno o más contextos. Todo programa LEC tiene un contexto por defecto (ej. `DEFECTO 2`). Las expresiones LEC se construyen con operadores aritméticos, constantes numéricas y variables. Los operadores aritméticos son la suma (+), la resta (-) y el producto (*). Las constantes numéricas son enteros positivos o cero. La variable LEC puede referenciar a un contexto (ej. `z{3}` es una variable del contexto 3) o al contexto por defecto (ej. `x`).
Suponga la siguiente gramática para LEC:

```
programa : contextos defecto expresiones

contextos : (contexto)+

contexto : CONTEXTO NUMERO asignaciones

asignaciones : (asignacion)+

asignacion : VARIABLE ASIG NUMERO PUNTOYCOMA

defecto : DEFECTO NUMERO

expresiones : EXPRESIONES (expresion PUNTOYCOMA)*

expresion : expresion1 ((MAS|MENOS) expresion)?

expresion1 : expresion2 (POR expresion1)?

expresion2 : VARIABLE LLAVEABIERTA NUMERO LLAVECERRADA
           | VARIABLE
           | NUMERO
           | PARENTESISABIERTO expresion PARENTESISCERRADO
```

SE PIDE: Decisiones de diseño y gramática atribuida para el analizador semántico del lenguaje LEC que detecte los siguientes errores:

(1) Ninguna variable puede inicializarse más de una vez en un mismo contexto (ej. `x = 9` en nuestro programa de ejemplo).
(2) No se permite seleccionar un contexto por defecto inexistente (ej. `DEFECTO 2` en nuestro programa de ejemplo).
(3) No se permiten el uso de variables con contexto inexistentes (ej. `x{10}` o `y{2}` en nuestro programa de ejemplo).
(4) No se permite el uso de variables sin inicializar (ej. `y{3}` en nuestro programa de ejemplo).

El analizador semántico dará mensajes por pantalla al detectar un error. Por ejemplo, la ejecución del analizador semántico sobre el programa de ejemplo emitirá los siguientes mensajes por pantalla:

```
Error: variable x reinicializada en contexto 1
Error: contexto por defecto 2 inexistente
Error: contexto 10 inexistente para variable x en línea 14
Error: contexto 2 inexistente para variable y en línea 14
Error: variable y{3} sin inicializar en línea 15
```

IMPORTANTE Restricción: responda en la página siguiente usando, si lo necesita, sus dos caras (use los folios para borrador y la página referida para la versión definitiva). Sea claro y legible. Evite tachaduras.

PROCESADORES DE LENGUAJES Examen (Ev. Continua)

Nombre: _____ Apellidos: _____

```
// Decisiones:

//    1:  Hay que memorizar las variables de cada contexto: memoria_contextos
            ---------------------------------------------
            contexto | variables inicializadas en contexto
            ---------------------------------------------
//    2:  Hay que memorizar el contexto por defecto: contexto_por_defecto

//    3:  Hay que comprobar las restricciones semánticas:
          (3.1) variable reinicializada en un contexto.
                Para cada asignación en un contexto se comprueba si la variable ya
                está almacenada para dicho contexto.
          (3.2) contexto por defecto inexistente. Basta comprobar si el contexto por
                defecto ocurre en memoria_contextos.
          (3.3) referencias a contextos inexistentes en una variable. Basta comprobar
                si el contexto ocurre en memoria_contextos
          (3.4) referencias a variables sin inicializar en una expresión.
                Hay dos casos, las variables del contexto por defecto, se comprueba su
                inicialización en el contexto por defecto almacenado en
                memoria_contextos. Las restantes variables se comprueban su
                inicialización en el contexto correspondiente almacenado en
                memoria_contextos

//      Ejemplo de programa LEC:
//      CONTEXTO 1
           x = 5;
           y = 7;
           z = 1;
           x = 9; // variable reinicializada

        CONTEXTO 3
           x = 9;
           z = 3;

        DEFECTO  2 //contexto por defecto inexistente

        EXPRESIONES
           (x{10} - y{2}) * z;          //contexto inexistente 10
           (x{1} + y{1}) * z{3} + y{3}; //variables sin inicializar y{3}
           (x - y) * m;


// Gramática atribuida:


// Memoria para almacenar las variables inicializadas de cada contexto:
   memoria_contextos
// Memoria para almacenar el contexto por defecto: contexto_por_defecto

programa : contextos defecto expresiones ;

contextos : (contexto)+ ;

contexto : CONTEXTO NUMERO asignaciones[NUMERO]  ;

asignaciones[CONTEXTO] : (asignacion[CONTEXTO])+; (1)

asignacion[CONTEXTO] : VARIABLE ASIG NUMERO PUNTOYCOMA
    {si VARIABLE ya está en memoria_contextos para el contexto CONTEXTO entonces
```

```
       error indicando que VARIABLE está reinicializada en CONTEXTO  (3.1)
    sino
       incluir VARIABLES en memoria_contextos para el contexto CONTEXTO (1)
    fsi};

defecto : DEFECTO NUMERO  {si existe el contexto NUMERO entonces
                              memorizarlo como contexto_por_defecto  (2)
                           sino
                              error por contexto por defecto NUMERO inexistente (3.2)
                           fsi};

expresiones : EXPRESIONES (expresion PUNTOYCOMA)*  ;

expresion : expresion1 ((MAS|MENOS) expresion)? ;

expresion1 : expresion2 (POR expresion1)? ;

expresion2 : VARIABLE LLAVEABIERTA NUMERO LLAVECERRADA
           | VARIABLE
           | NUMERO
           | PARENTESISABIERTO expresion PARENTESISCERRADO
           ;


defecto : DEFECTO NUMERO ;

expresiones : EXPRESIONES (expresion PUNTOYCOMA)*  ;

expresion : expresion1 ((MAS|MENOS) expresion)? ;

expresion1 : expresion2 (POR expresion1)? ;

expresion2 : VARIABLE LLAVEABIERTA NUMERO LLAVECERRADA
               {si no existe el contexto NUMERO en memoria_contextos entonces
                   error por contexto NUMERO inexistente para VARIABLE en la línea en
                   la que ocurre VARIABLE
                fsi  (3.3)
                si VARIABLE no inicializada en contexto NUMERO entonces
                   error VARIABLE{NUMERO} sin inicializar en la línea en la que ocurre
                   VARIABLE
                fsi}  (3.4)
           | VARIABLE
             {si VARIABLE no inicializada en contexto por defecto entonces
                error VARIABLE sin inicializar en la línea en la que ocurre VARIABLE
             fsi}  (3.4)
           | NUMERO
           | PARENTESISABIERTO expresion PARENTESISCERRADO
           ;
```