1° Parcial – 26/06/2018 - **SINTAXIS Y SEMÁNTICA DE LENGUAJES** – K2002

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1a | 1b | 1c | 1d | 2 | 3a | 3b | 4a | 4b | 4c | 4d | Notal Final |
| 0.5p | 0.5p | 0.5p | 1.5p | 3p | 1p | 1p | 0.5p | 0.5p | 0.5p | 0.5p |  |
|  |  |  |  |  |  |  |  |  |  |  |

**Ejercicio 1 - Dada la siguiente gramática formal**

1 Expresión -> Término |

2 Expresión - Término

3 Término -> Factor |

4 Factor / Termino

5 Factor -> Número |

6 ( Expresión )

7 Número -> 0|1|2|3|4|5|6|7|8|9

a) Clasificar el tipo de gramática según la jerarquía de Chomsky, justifique.

b) Indicar si existe recursividad en sus producciones, en caso afirmativo mencione si son a izquierda o a derecha.

c) Indique prioridad y asociatividad de los operadores - y /

d) Reduzca la siguiente palabra (4/1-4/2)-1-2/(1) mediante tabla de evaluación e calcule el valor de la expresión numérica según la prioridad y asociatividad de los operadores involucrados.

**Ejercicio 2 – Dado el siguiente fragmento de código**

int main (char a) {

int a = 78, b = 3.7, 0 = c, d = ‘a’, e = 5;

char main = d, contador$ = 7, g[4] = “Hola”;

if (b=3&&e=5){char d;}

}

Indique errores léxicos, sintácticos y semánticos.

**Ejercicio 3 – Flex y ERX**

a) Dadas las siguientes reglas en flex

[0-6]+ printf(“%s”,yytext);

[a-z]+ ;

[0-9]{3,5} acumulador = acumulador + atoi(yytext);

[a-fA-F] printf(“Carácter hexadecimal encontrado”);

Indique su salida yyout y las acciones que serán llevadas a cabo según las reglas indicadas previamente para la siguiente entrada yyin: 0126F127bla99AaA000

b) Dada la siguiente ERX: [\+\-][0-9]+(\.)?([0-9]+) Indique si las siguiente palabras pertenecer al lenguaje representado por dicha ERX: +0 , +1.1 , 0 , -159

**Ejercicio 4 – Indique verdadero o falso justificando adecuadamente**

a) La gramática de las expresiones es regular.

b) Las constantes enteras 0x045 y 045 son equivalentes.

c) La ERX de los identificadores es [a-zA-z][a-zA-z0-9]+

d) En ANSI C, toda sentencia sintácticamente correcta debe ser derivable de la BNF.

1 *expresión: expAsignación*

2 *expAsignación: expCondicional*

3 *expUnaria operAsignación expAsignación*

4 *operAsignación:* uno de **= +=**

5 *expCondicional: expOr*

6 *expOr* **?** *expresión* **:** *expCondicional*

7 *expOr: expAnd*

8 *expOr* **||** *expAnd*

9 *expAnd: expIgualdad*

10 *expAnd* **&&** *expIgualdad*

11 *expIgualdad: expRelacional*

12 *expIgualdad* **==** *expRelacional*

13 *expRelacional: expAditiva*

14 *expRelacional* **>=** *expAditiva*

15 *expAditiva: expMultiplicativa*

16 *expAditiva* **+** *expMultiplicativa*

17 *expMultiplicativa: expUnaria*

18 *expMultiplicativa* **\*** *expUnaria*

19 *expUnaria: expPostfijo*

20 **++** *expUnaria*

21 *operUnario expUnaria*

22 **sizeof (***nombreTipo***)**

23 *operUnario:* uno de **& \* – !**

24 *expPostfijo: expPrimaria*

25 *expPostfijo* **[** *expresión* **]**

26 *expPostijo* **(** *listaArgumentos*op **)**

27 *listaArgumentos: expAsignación*

28 *listaArgumentos* **,** *expAsignación*

29 *expPrimaria: identificador*

30 *constante*

31 *literalCadena*

32 **(** *expresión* **)**

33 *nombreTipo:* uno de **char int doublé**

*sentencia:* una de *sentCompuesta sentExpresión sentSelección sentIteración sentSalto*

*sentCompuesta:* **{** *listaDeclaraciones*op *listaSentencias*op **}**

*listaDeclaraciones: declaración*

*listaDeclaraciones declaración*

*listaSentencias: sentencia*

*listaSentencias sentencia*

*sentExpresión: expresión*op **;**

*sentSelección:* **if (** *expresión* **)** *sentencia*

**if (** *expresión* **)** *sentencia* **else** *sentencia*

**switch (** *expresión* **)** *sentencia*

*sentIteración:* **while (** *expresión* **)** *sentencia*

**do** *sentencia* **while (** *expresión* **) ;**

**for (** *expresión*op **;** *expresión*op **;** *expresión*op **)** *sentencia*

*sentSalto:* **return** *expresión*op **;**