Simulacro 1° Parcial – **SINTAXIS Y SEMÁNTICA DE LENGUAJES**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | Nota Final |
| 2p | 1.25p | 2.25p | 2.25p | 2.25p |  |
|  |  |  |  |  |

**Ejercicio 1**

Dada la siguiente expresión de C: a+(0||2)<b>c\*0-1. Indique cual/es de las siguientes afirmaciones es correcta:

* Vale 0.
* Vale 1.
* El valor depende de los valores de a,b y c.
* La asociatividad de la operación de + y – es de derecha a izquierda.
* No es una expresión válida de C.
* La prioridad de los operadores de < y > es la misma.

**Ejercicio 2**

Indique cual/es de las siguientes ERX corresponden a sublenguajes de las constantes octales de C, justifique:

* [0-6]\*
* [0-7]{3}
* 0|[1-7]+
* [0-7]
* 0{2}[2-6][3-7]\*

**Ejercicio 3**

Dada la cadena de entrada yyin: “012911B127Bla9Ga55C0005” y las siguientes reglas de flex:

[0-7]+ printf(“%s”,yytext);

[a-z]+ ;

[0-9]{2,5} acumulador = acumulador + atoi(yytext);

[a-fA-F] acumulador += 1;

Indique cual de las siguientes afirmaciones es correcta, justifique su respuesta:

* Las reglas no poseen ERX válidas
* Su salida yyout es 9Ga
* Existe más de una salida (yyout) posible
* El valor final del acumulador es 1296
* La cantidad total de lexemas encontrados al procesar la cadena de entrada es 12

**Ejercicio 4**

Dado el siguiente fragmento de código en C

int main (int a) {

int a, b = 3-0.7, 0 = c, d = ‘a’+‘b’, e\_\_ = -5;

return 0;

}

Indique cual/es de las siguientes afirmaciones es correcta, justifique su respuesta:

* No presenta errores léxicos
* No presenta errores sintácticos
* No presenta errores semánticos
* Posee 33 lexemas

**Ejercicio 5**

Indique cual/les de las siguientes afirmaciones es correcta, justifique su respuesta:

* El lenguaje de literales cadenas de C puede ser reconocido por un AFD.
* Los AFP son utilizados para implementar gramáticas regulares (tipo 3)
* Toda gramática independiente de contexto posee una gramática regular equivalente
* La operación de intersección se encuentra definida sobre el conjunto de ER
* En C, si una expresión es sintácticamente correcta entonces es derivable de la BNF de expresiones

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 **;**