**Ejercicio 1**COUNT\_DISTINCT{(a/7+b), f\*(b-1), c %% 32, 64, 128}  
COUNT\_DISTINCT{(b, (7-g), var1 %% 0}  
COUNT\_DISTINCT{a %% 1}

**Resolucion Mara**

**S -> COUNT\_DISTINCT {A}**

**S -> COUNT\_DISTINCT {B %% 0}**

**A -> E , A , cte**

**A -> E %% cte (la magia del balance que pedia el enunciado)**

**B -> B , E**

**B -> E**

**E -> E + T  
E -> E \* T  
E -> T  
T -> T \* F  
T -> T / F  
T -> F  
F -> id  
F -> cte  
F -> ( E )**

**En cuanto a los espacios no es relevante al sintáctico**

**Considerando que B %% 0 donde cero es un token cte del lexico, hay ambiguedad en su reconocimiento ya que no se puede verificar “cero” y luego en otra regex [0-9], ASI QUE EL ENUNCIADO PODIAS PONER QUE LAS REGLAS ERAN AMBIGUAS Y NO SE PODIA RESOLVER EXPRESADO ASI**

**Para ernan no nos metemos en el lexico para resoler un ejercicio del SINTACTICO.**

**Ejercicio 2**

Suponga un lenguaje que posee tipos de variables enteras y string, e introduce la posibilidad de concatenar variables tipo string. Dicha caracteristica se introduce a la nueva version sin agregar al compilador ningun simbolo no existente en la version anterior, y se realiza mediante la funcion concatena:  
  
string a:="hola"; b:=" mundo"; var1="" \*\* defincion de variables, que ya existe en la version  
var1=concatena(a, b) \*\* var1 tendra el contenido "hola mundo"  
var1=concatena(a, a, a) \*\* tendra el contenido "holaholahola"  
  
El lenguaje verifica que los tipos enunciados en la lista dentro de la funcion sean compatibles.  
Se pide indicar que etapas del compilador se veran afectadas (V o F) por la introduccion de la funcion, justificando y proponiendo un ejemplo en los casos afirmativos:  
 (a) Analizador lexicografico  
 (b) Rutinas de errores lexicos  
 (c) Analizador sintactico  
 (d) Rutinas de errores sintacticos  
 (e) Generacion de codigo intermedio  
 (f) Generacion de codigo assembler

1. **Si, tiene que recibir la palabra concatena (a , b)**
2. **No, estar mal definida como cooncatena es un id. También el ancho de la variables no cambia**
3. **Si, Función Concatena**
4. **Si, la función Concatena () y no cierro el paréntesis es un error sintáctico**
5. **Si, habra pseudocodigo para diferetes tareas dentro de la resolución de concatena**

**No, si fuera una traduccion de una funcion a otra funcion que hace lo mismo.N**

1. **Si, la resolucion en Assambler se afecta**

**No, si fuera una traduccion**

**Ejercicio 3**

¿Cuales de los siguientes tratamientos se debe aplicar a los comentarios en el analizador lexico de un compilador?  
Indique V o F y justifique.  
 (a) Se debe crear un token de tipo comentario y se lo debe enviar al analizador sintactico.  
 (b) Se debe crear el token comentario, pero el mismo se descarta.  
 (c) Si el comentario posee delimitador de inicio y cierre, se debe procesar (leer) todo el contenido pero luego ignorarlo.  
 (d) Si el comentario posee solamente delimitador de inicio y finaliza con un fin de linea, se debe descartar el resto de la lectura al detectar el caracter de inicio.  
 (e) Se debe verificar que el comentario no contenga caracteres no aceptados por el lenguaje  
 (f) Se debe verificar la existencia de palabras reservadas dentro del comentario.

1. **No, se debe crear el token, debe analizar el contenido pero no envia nada al sintactico**
2. **Si, se debe crear el token, analizar el contenido pero no enviar nada.**
3. **No, un lexico nunca ignora, siempre lee y procesa.**
4. **No, se debe descartar el contenido procesandolo hasta el fin de linea**
5. **Si, la regex de comentario aceptara caracteres permitidos por el lenguaje**
6. **No, se debe analzar el conetido para que los simbolos pertenescan al lenguaje hasta final de comentario y luego descartarlo**

**Ejercicio 4**  
  
La funcion SumSI almacena en el segundo ID la sumatoria de todas las ctes que se encuentran a la derecha del punto y coma, siempre y cuando cumplan con la condicion que respecta al primer ID.  
  
S --> SumSI ( OP ID , ID ; L )  
L --> cte  
L --> L , cte  
OP --> <> | == | < | > | <= | >=  
  
SumSI ( <> b , var1 ; 10, 20, 30 ) // b=20 -> var1 = 10 + 30  
SumSI ( <= a , var2 ; 1, 3, 5, 8) // a=7 -> var 2 = 1 + 3 + 5  
SumSI ( > p , var3 ; 25, 30, 35, 40 ) // p=25 -> var3 = 30 + 35 + 40  
  
Escribir el Ejemplo 1 en notación polaca inversa y especificar las acciones semánticas que generan la polaca inversa.

**Ejemplo: sumasi(<> b, var; 10, 20, 30)**

|  |  |
| --- | --- |
| Gramática  S-> sumasi(O id, id; L)  L-> L, cte  L-> cte  O -> < | > | <= | >= | <> | ==  **sumasi ( > b, var; 10, 20 , 30 )** | Lista de Reglas reconocidas  R03 O-><>  R09 L->cte  R08 L->L, cte  R08 L->L, cte  R01 S-> sumasi(O id, id; L) |

Polaca

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| \_sal | BLE | = | \_var | id | = | \_asg | id | = | \_ctd | 0 | = | (\_ctd)\_lst | cte | = | \_ctd | \_ctd | 1 | + | = |
| R03 O->> | | | R01 S-> sumasi(O id, id; L) (MAGIA) | | | | | | R09 L->cte | | | | | | R08 L->L, cte | | | | |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| (\_ctd)\_lst | cte | = | \_ctd | \_ctd | 1 | + | = | (\_ctd)\_lst | cte | = | \_var | (\_ctd)\_lst | CMP | **\_sal** | **42** | \_sum | \_sum | (\_ctd)\_lst | + |
| R08 L->L, cte | | | R08 L->L, cte | | | | | | | | R01 S-> sumasi(O id, id; L) | | | | | | | | |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| = | \_ctd | \_ctd | 1 | - | = | 0 | CMP | BLT | **32** | \_asg | \_sum | = |  |  |  |  |  |  |  |
| R01 S-> sumasi(O id, id; L) (MAS MAGIA) | | | | | | | | | | | | |  |  |  |  |  |  |  |

Gramática y Polaca según Mara

|  |  |
| --- | --- |
| R01 S-> sumasi(O id, id; L) | sumasi(O id,**{\_var = “id”;}** id; **{\_asg = “id”;}** L) **{i\_p (\_var); i\_p((ctd)\_lst); i\_p(“CMP”); i\_p(\_sal); i\_p (“POS\_SALTO+6”); i\_p(sum); i\_p(sum); i\_p((\_cte)\_lst); i\_p(“+”); i\_p (“=”);}** |
| R02 O-> < | O-> <  **{\_sal = “BGE”;}** |
| R03 O-> > | O-> >  **{\_sal = “BLE”;}** |
| R04 O-> <= | O-> <= **{\_sal = “BGT”;}** |
| R05 O-> >= | O-> >= **{\_sal = “BLT”;}** |
| R06 O-> <> | O-> <> **{\_sal = “BEQ”;}** |
| R07 O-> == | O-> == **{\_sal = “BNE”;}** |
| R08 L->L, cte | L->L, cte **{(\_ctd)\_lst = cte; \_ctd++;}** |
| R09 L->cte | L->cte **{\_ctd = 0;(\_ctd)\_lst = cte;}** |

Gramática y Polaca según “YO EL MEJOR” (luego de correr sumasi, para los siguientes elementos, debería decrementar el contador y hacer las comparaciones escribiendo la intermedia una y otra vez hasta que no haya más elementos en el \_ctd. Luego el sumador recien ahi se asigna).

(\_ctd)lst es una estrategia para crear variables con el nombre 0lst, 1lst, 2lst, etc para poder diferenciar cada elemento de la lst.

|  |  |
| --- | --- |
| R01 S-> sumasi(O id, id; L) | sumasi(O id,  **{ i\_p(\_var);i\_p(id);i\_p(“=”);}**  id;  **{ i\_p(\_asg);i\_p(id);i\_p(“=”);}**  L)  **{ i\_p(\_var); i\_p((ctd)\_lst); i\_p(“CMP”); i\_p(\_sal); i\_p (“POS+6”); i\_p(sum); i\_p(sum); i\_p((\_cte)\_lst); i\_p(“+”); i\_p (“=”); i\_p(\_ctd); i\_p(\_ctd); i\_p(“1”); i\_p(“-”); i\_p(”=”); i\_p(“0”); i\_p(“CMP”); i\_p(“BLT”); i\_p(POS-18); i\_p(\_asg); i\_p(\_sum); i\_p(“=”)}** |
| R02 O-> < | O-> < **{i\_p(\_sal)**; **i\_p(“BGE”); i\_p(“=”);}** |
| R03 O-> > | O-> > **{i\_p(\_sal)**; **i\_p(“BLE”); i\_p(“=”);}** |
| R04 O-> <= | O-> <= **{i\_p(\_sal)**; **i\_p(“BGT”); i\_p(“=”);}** |
| R05 O-> >= | O-> >= **{i\_p(\_sal)**; **i\_p(“BLT”); i\_p(“=”);}** |
| R06 O-> <> | O-> <> **{i\_p(\_sal)**; **i\_p(“BEQ”); i\_p(“=”);}** |
| R07 O-> == | O-> == **{i\_p(\_sal)**; **i\_p(“BNE”); i\_p(“=”);}** |
| R08 L->L, cte | L->L, **{i\_p(\_ctd)**; **i\_p(\_ctd); i\_p(“1”); i\_p(“=”); i\_p((\_ctd)\_lst); i\_p (cte); i\_p(“=”)}** |
| R09 L->cte | L->cte **{i\_p(\_ctd); i\_p(“0”); i\_p(“=”); i\_p((\_ctd)\_lst); i\_p (cte); i\_p(“=”)}** |

Pseudocódigo para entender la Polaca, acciones propias de Bison y otras que pueden ser pasadas de manera transparente a Polaca

|  |  |
| --- | --- |
| R01 S-> sumasi(O id, id; L) | sumasi(O id,  **{\_var = “id”;}**  id;  **{\_asg = “id”;}**  L)  **{do**  **if ( \_var \_sal \_(ctd)lst )**  **\_sum += \_(ctd)lst;**  **\_ctd--;**  **while (\_ctd >= 0);}**  **\_asg=\_sum;}** |
| R02 O-> < | O-> < **{\_sal = “BGE”;}** |
| R03 O-> > | O-> > **{\_sal = “BLE”;}** |
| R04 O-> <= | O-> <= **{\_sal = “BGT”;}** |
| R05 O-> >= | O-> >= **{\_sal = “BLT”;}** |
| R06 O-> <> | O-> <> **{\_sal = “BEQ”;}** |
| R07 O-> == | O-> == **{\_sal = “BNE”;}** |
| R08 L->L, cte | L->L, cte **{\_ctd++;(\_ctd)\_lst = cte;}** |
| R09 L->cte | L->cte **{\_ctd = 0;(\_ctd)\_lst = cte;}** |