Apuntes
Clase 1-Analizador Léxico y Expresiones Regulares
Clase 2-Analizador Sintáctico y Gramáticas
Clase 3-Parsing

Descargas
Instalador Flex-Bison

Expresiones Regulares – Analizador Léxico - FLEX

1. Escribir expresiones regulares para los siguientes elementos devolviendo un par (token/lexema) por cada una de ellas. ¿Existen tokens que generen lexemas diferentes?

   a) Códigos postales. (Por ej.: 1234, 8366, etc., pero no 422 o 0027).
   b) Número de Patentes mercosur
   c) Comentarios acotados por /* y */. Σ = {letras, *, /}
   d) Identificadores de cualquier longitud que comiencen con una letra y contengan letras, dígitos o guiones. No pueden terminar con guión.
   e) Idem anterior pero que no contengan dos guiones seguidos
   f) Constantes en otras bases como las del lenguaje C
   g) Constantes aritméticas enteras. Controlar el rango permitido.
   h) Constantes reales con formato xx.xx Controlar el rango permitido.
   i) Constantes string de la forma “texto “
   j) Palabras reservadas (IF-WHILE-DECVAR-ENDDEC-INTEGER-FLOAT-WRITE)
   k) Operadores lógicos y operadores aritméticos básicos
   l) Constantes en otras bases como en el lenguaje C

2. Dado el siguiente fragmento de código, determinar los elementos léxicos y escribir una expresión regular para cada uno de éstos

```
DECVAR
        contador: Integer;
        promedio: Float;
        actual, suma: Float;
ENDDEC

write "Hola mundo!";
contador: 0;

actual: 999;
suma: 02;
contador: contador+1;
while (contador <= 92) {
        contador: contador + 1;
        actual: (contador/0.342) + (contador*contador);
        suma: suma + actual;
        }

write "La suma es: ";
write suma;

if (actual > 2){
```

```
        write "2 > 3";
}

if (actual < 3){
        if(actual >= 3){
                write "soy true";
        }
        if(actual <= 3){
                write "soy true";
        }
        if(actual != 3){
                write "soy true";
        }
        if(actual == 3){
                write "soy true";
        }

}else{
        actual:333.3333;
}
```

3. Sobre el fragmento anterior, escribir una tabla de símbolos para los lexemas que correspondan

4. Generar un archivo FLEX para probar el fragmento de código del ejercicio anterior.

Gramáticas - Analizador Sintáctico - BISON

5. Desarrollar una gramática en formato BNF para reconocer expresiones aritméticas simples (sumas, restas, multiplicaciones y divisiones) que operan con constantes enteras e identificadores (Utilizar tokens para los elementos léxicos)

6. Desarrollar una gramática igual a la anterior donde la multiplicación y división tenga prioridad sobre la suma y la resta.

7. Agregar a la gramática anterior la posibilidad de incorporar expresiones entre paréntesis.

8.
   a) Desarrollar una BNF/GLC para reconocer asignaciones simples. Estas deben tener un Left Side y un Right Side. El Left Side debe ser un identificador y el Right Side una expresión aritmética simple que soporte sumas, restas, multiplicaciones, divisiones y expresiones entre paréntesis (Utilizar tokens para los elementos léxicos)

   b) Hacer un árbol de parsing para el siguiente programa :
   *actual:= (contador/342) + (contador*contador);*

9. En algunos lenguajes de programación se permite la asignación múltiple, en la que varias variables pueden recibir el mismo valor. Los que siguen son algunos ejemplos de sentencias en los que se hace uso de esta forma de escritura:

a:=inc:=minimo:=***expresion***;
total:=precio:=***expresión***;

a) Escriba en BNF la sintaxis de este tipo de asignaciones con las siguientes condiciones: la sentencia debe terminar con ";" y el valor de la derecha debe ser una ***expresión*** como la desarrollada en el ejercicio 7

b) Hacer el árbol de parsing para el siguiente programa:

*actual:=promedio:=contador:= promedio/ 342 + (contador*contador);*

10. Definir una BNF/GLC que soporte programas que tengan asignaciones simples como las del ejercicio 8 y asignaciones múltiples como las del ejercicio 9. Estos programas pueden tener una sentencia o muchas sentencias. Cada sentencias finaliza con ;
Los programas de este tipo deben comenzar con la palabra reservada INICIO y finalizar con FIN.

11. Utilizar BISON y FLEX para probar el siguiente programa con la gramática del ejercicio anterior

INICIO
contador:= 0;
actual:=contador:= 999;
contador: contador+1;
*contador: contador + 1/ (actual *otra);*
*actual: (contador/342) + (contador*contador);*
actual:=promedio:=contador:=342;
FIN

Gramáticas – Análisis Sintáctico - BISON

12. Dada la siguiente gramática con elementos terminales ***fin , := y exp***

P -> L E fin
L -> L , id
L -> id
E -> := exp

a) Hacer el parsing ascendente
b) Hacer el árbol de parsing y devolver la lista de reglas para la hilera : *id,id,id,id:=exp;*
c) Probar con FLEX y BISON y comparar la lista de reglas obtenida

13. Dada la siguiente gramática

R1: S → (L)
R2: S-→ ( )
R3: L → (L)
R4: L → L ( )
R5: L → ( )

a) Hacer el parsing ascendente
b) Hacer el árbol de parsing y devolver la lista de reglas para la hilera : (((( ))) ( ) ( ))
c) Probar con FLEX y BISON y comparar la lista de reglas obtenida

14. Dada la gramática del ejercicio 8

a) Hacer el parsing ascendente
b) Hacer el árbol de parsing y devolver la lista de reglas para la hilera

*actual:= (contador/342) + (contador*contador);*

c) Probar con FLEX y BISON y comparar la lista de reglas obtenida

15. Se tiene un lenguaje que soporta 2 tipo de variables: enteras y strings, y soporta también constantes enteras y string. El mismo posee las siguientes características:
- Las variables enteras deben terminar con el carácter %
- las variables string deben terminar con $
- las constantes enteras de mas de un digito no pueden comenzar con 0
- las constantes string deben ir entre comillas ["]
- Las variables y constantes enteras se pueden sumar, restar, multiplicar y dividir libremente entre si.
- Las variables y constantes strings pueden concatenarse libremente entre si con el símbolo "++"
- No esta permitido mezclar los tipos en las operaciones
- Las asignaciones deben ser entre elementos del mismo tipo

Escribir una BNF/GLC que describa dicho lenguaje

16. Genere una BNF/GLC para un lenguaje que soporta:

- Asignaciones simples de expresiones
- Sentencia IF
- Sentencia WHILE
- Sentencia WRITE (constante string)
- Comienza y Termine con las palabras INICIO y FIN
- Separador de sentencias ;
- Separador de bloques { }
- Operador de asignación :
- Condiciones : dos expresiones con un solo operador

17. Probar con FLEX y BISON el siguiente programa. Agregar, si es necesario nuevas expresiones regulares.

```
INICIO
        contador: 0;

        actual: 999;
        suma: 02;
        contador: contador+1;
        while (contador <= 92) {
                contador: contador + 1;
                actual: (contador/0.342) + (contador*contador);
                suma: suma + actual;
                }


        if (actual > 2){
```

```
        write "2 > 3";
}

if (actual < 3){
        if(actual >= 3){
                write "soy true";
        }
        if(actual <= 3){
                write "soy true";
        }
        if(actual != 3){
                write "soy true";
        }
        if(actual == 3){
                write "soy true";
        }

}else{
        actual:333.3333;
}
```

FIN

18. Agregar al lenguaje del ejercicio anterior la posibilidad de definir sentencias en las que se pueda incorporar la función standard AVERAGE
Formato de AVERAGE : AVG(Lista de expresiones) donde Lista de expresiones es una lista de expresiones aritméticas separdas por comas dentro de dos corchetes
AVERAGE debe operar dentro de una expresión .

19. Probar con FLEX y BISON el siguiente programa

```
INICIO

write "Hola mundo!";
contador: =0;

actual: 99999999999999;
suma: 02;
contador: contador+1;
while (contador <= 92) {
        contador: contador + 1;
        actual: (contador/0.342) + (contador*contador);
        suma: suma + actual *  avg([3,3*4,12,actual,actual*2, actual*actual]);

}

write "La suma es: ";
write suma;

if (actual > 2){
        write "2 > 3";
}
```

```
if (actual < 3){
        if(actual >= 3){
                write "soy true";
        }
        if(actual <= 3){
                write "soy true";
        }
        if(actual != 3){
                write "soy true";
        }
        if(actual == 3){
                write "soy true";
        }

}else{
        actual:333.3333;
}


FIN
```

20. Agregar al lenguaje anterior la posibilidad de tener un sector de declaraciones delimitado por las palabras reservadas DECVAR y END. Dentro de este bloque las variables se declaran de la forma:
    Id: Tipo; o id,id,…..,id: Tipo

21. Probar con FLEX y BISON el siguiente programa

```
INICIO
DECVAR
        contador: Integer;
        promedio: Float;
        actual, suma: Float;
ENDDEC

write "Hola mundo!";
contador: =0;

actual: 99999999999999;
suma: 02;
contador: contador+1;
while (contador <= 92) {
        contador: contador + 1;
        actual: (contador/0.342) + (contador*contador);
        suma: suma + actual *  avg([3,3*4,12,actual,actual*2, actual*actual]);

}

write "La suma es: ";
write suma;

if (actual > 2){
        write "2 > 3";
}

if (actual < 3){
```

```
        if(actual >= 3){
                write "soy true";
        }
        if(actual <= 3){
                write "soy true";
        }
        if(actual != 3){
                write "soy true";
        }
        if(actual == 3){
                write "soy true";
        }

}else{
        actual:333.3333;
}


FIN
```

22. Generar una tabla de símbolos en el sector de declaraciones del ejercicio anterior. Indique claramente cómo quedan conformadas las columnas de la Tabla

23. Indique como sería el error si alguna variable dentro del programa no estuviese declarada. Probar.