TRABAJO PRÁCTICO COMPILADOR

CONSIDERACIONES GENERALES

Es necesario cumplir con las siguientes consideraciones para evaluar el TP.

1. Cada grupo deberá desarrollar el compilador teniendo en cuenta:
   - Todos los temas comunes (Ver ANEXO TEMAS)
   - El tema especial asignado al grupo.

2. Se fijarán puntos de control con fechas y consignas determinadas

PRIMERA ENTREGA

OBJETIVO: Realizar un analizador lexicográfico utilizando la herramienta JFLEX. La aplicación realizada debe mostrar una interfaz gráfica que pueda utilizarse como IDE del compilador, en la cual se debe poder ingresar código de nuestro programa, cargar un archivo y editarlo, compilar el contenido del programa ingresado y mostrar un texto aclaratorio, identificando los tokens reconocidos por el parser u errores encontrados en el análisis. Las impresiones deben ser claras. Las reglas que no realizan ninguna acción no deben generar salida.

El material a entregar será:

- El archivo jflex que se llamará **Lexico.flex**
- Un archivo de pruebas generales que se llamará **prueba.txt** provisto por la asignatura según el tema asignado
- Un archivo con la tabla de símbolos **ts.txt**
- Código fuente del proyecto

Todo el material deberá ser subido a algún repositorio GIT (Github, Gitlab, etc.) y su enlace enviado enviado a teoria1.unlu@gmail.com

Asunto: GrupoXX
Fecha de entrega: 26/10/20

SEGUNDA ENTREGA

OBJETIVO: Realizar un analizador sintáctico utilizando la herramienta Java CUP. La aplicación realizada debe mostrar una interfaz gráfica que pueda utilizarse como IDE del compilador, el cual en este caso deberá mostrar por pantalla un texto aclaratorio identificando las reglas sintácticas que va analizando el parser, en base a un archivo de entrada (prueba.txt). Las impresiones deben ser claras. Las reglas que no realizan ninguna acción no deben generar salida.

El material a entregar será:

- El archivo jflex que se llamará **Lexico.flex**
- El archivo jcup que se llamará **Sintactico.cup**
- Un archivo de pruebas generales que se llamará **prueba.txt** provisto por la asignatura según el tema asignado
- Un archivo con la tabla de símbolos **ts.txt**
- Código fuente del proyecto

Todo el material deberá ser subido a algún repositorio GIT (Github, Gitlab, etc.) y su enlace enviado enviado a teoria1.unlu@gmail.com

Asunto: GrupoXX
Fecha de entrega: 09/11/20

ANEXO TEMAS

TEMAS COMUNES

WHILE
Implementación de *While*

DECISIONES
Implementación de *IF*

ASIGNACIONES
Asignaciones simples *A:=B*

TIPO DE DATOS
Constantes numéricas

- enteras (16 bits)
- reales (32 bits)

El separador decimal será el punto "."

Ejemplo:

```
a = 99999.99
a = 99.
a = .9999
```

Constantes string
Constantes de 30 caracteres alfanuméricos como máximo, limitada por comillas (" ") ,de la forma *"XXXX"*

Ejemplo:

```
b = "@sdADaSjfla%dfg"
b = "asldk  fh sjf"
```

Las constantes deben ser reconocidas y validadas en el *analizador léxico*, de acuerdo a su tipo.

VARIABLES

Variables numéricas
Estas variables reciben valores numéricos tales como constantes numéricas, variables numéricas u operaciones que arrojen un valor numérico, del lado derecho de una asignación.

Variables string
Estas variables pueden recibir una constante string

Las variables no guardan su valor en tabla de símbolos.
Las asignaciones deben ser permitidas, solo en los casos en los que los tipos son compatibles, caso contrario deberá desplegarse un error.

COMENTARIOS

Deberán estar delimitados por "</" y "/>" y podrán estar anidados en un solo nivel.

Ejemplo1:

```
</
IF (a <= 30)
      b = ”correcto” </ asignación string />
ENDIF
/>
```

Ejemplo2:

```
</ Así son los comentarios />
```

Los comentarios se ignoran de manera que no generan un componente léxico o token.

SALIDA

Las salidas se implementarán como se muestra en el siguiente ejemplo:

Ejemplo:

```
PRINT "ewr"    </ donde "ewr" debe ser una cte string />
```

CONDICIONES

Las condiciones para un constructor de ciclos o de selección, deberán ser comparaciones binarias que pueden estar ligadas por un conector lógico.

```
(expresión) < (expresión)
(expresión >= expresión) && (expresión < expresión)
```

DECLARACIONES

Todas las variables deberán ser declaradas dentro de un bloque especial para ese fin, delimitado por las palabras reservadas DECLARE y ENDDECLARE, siguiendo el siguiente formato:

```
DECLARE
            Línea_de_Declaración_de_Tipos
ENDDECLARE
```

Cada *Línea_de_Declaración_de_Tipos* tendrá la forma: *[Lista de Variables] := [Lista de Tipos]*

La *Lista de Variables* debe ser una lista de variables entre corchetes separadas por comas al igual que la Lista de TIpos. Cada variable se deberá corresponder con cada tipo en la lista de tipos según su posición. No deberán existir más variables que tipos, ni más tipos que variables.
Pueden existir varias líneas de declaración de tipos.

Ejemplos de formato:

```
DECLARE
      [a1, b1] := [FLOAT, INT]
      [p1, p2, p3] := [FLOAT,FLOAT,INT]
ENDDECLARE
```

PROGRAMA

Todas las sentencias del programa deberán ser declaradas dentro de un bloque especial para ese fin, delimitado por las palabras reservadas BEGIN.PROGRAM y END.PROGRAM, siguiendo el siguiente formato:

```
BEGIN.PROGRAM
      Lista_de_Sentencias
END.PROGRAM
```

Nota: la zona de declaración de variables deberá ser previa a la sección del programa.

TABLA DE SIMBOLOS

La tabla de símbolos tiene la capacidad de guardar las variables y constantes con sus atributos.
Los atributos portan información necesaria para operar con constantes, variables.

Ejemplo 1er ENTREGA (sin agregar tipos de datos)

NOMBRE	TOKEN	TIPO	VALOR	LONG
a1	ID		—	—
b1	ID		—	—
_hola	CTE_STR	—	hola	4
_mundo	CTE_STR	—	mundo	5
_30.5	CTE_F	—	30.5	—
_55	CTE_E	—	55	—

Tabla de símbolos

Ejemplo 2da ENTREGA (agregando tipos de datos)

NOMBRE	TOKEN	TIPO	VALOR	LONG
a1	ID	Float	—	—
b1	ID	Integer	—	—
_hola	CTE_STR	—	hola	4
_mundo	CTE_STR	—	mundo	5
_30.5	CTE_F	—	30.5	—
_55	CTE_E	—	55	—

Tabla de símbolos

TEMAS ESPECIALES

- **Grupo 1 INLIST**

Esta función del lenguaje, tomará como entrada una variable numérica y una lista de constantes y devolverá verdadero o falso según la variable enunciada se encuentre o no en dicha lista. Donde lista de constantes serán constantes numéricas separadas por punto y coma (;) y delimitadas por corchetes.
Esta función será utilizada en las condiciones presentes en ciclos y selecciones.

Ejemplo:

```
INLIST(variable, [lista de constantes])
INLIST (a, [2 ; 12 ; 24 ; 48])
INLIST (z, [2.3 ; 1.22])
```

```
a1:=12
IF INLIST(a1,[11;20:12;72]) THEN /* en este caso INLIST resulta verdadera */
        d1 := 3,14
END IF
```

- **Grupo 2 TAKE**

Esta función toma como entrada un operador de suma, resta, multiplicación o división entera, una constante entera y una lista de constantes, devolviendo el valor que resulta de aplicar el operador a los primeros "n" elementos. El valor de n quedará establecido en la componente cte. El resultado de la función puede ser utilizado en otras expresiones dentro del lenguaje y admite listas vacías, en cuyo caso retorna el valor 0.

Ejemplo:

```
TAKE (Operador ;  cte ;  [lista de constantes])
```

```
DEFVAR
var1, var2, var3, var4, var5, var6 : INTEGER
cad, cad2 : STRING
ENDDEF
var1:=TAKE (*  ; 3 ;  [2 ; 12 ; 24 ; 48] )  /* Resultado: 576 */
var2:=TAKE (+  ; 2 ;  [2 ; 12 ; 24 ; 48] )  /* Resultado: 14 */
var3:=TAKE (-  ; 3 ;  [2 ; 12 ; 24 ; 48] )  /* Resultado: -34 */
var4:=TAKE (/  ; 4 ;  [2 ; 12 ; 24 ; 48] )  /* Resultado: 0 */
var6:=TAKE (/  ; 4 ;  [ ] ) /* Resultado: 0 */
```

- **Grupo 3 QEqual**

Esta función del lenguaje tomará como entrada una expresión (pivot) y una lista de expresiones. Devolverá la cantidad de elementos iguales al pivot que se encuentran en la lista. La cantidad de listas es indefinida.
Esta función será utilizada en cualquier expresión del lenguaje.

Ejemplo:

```
#Iguales(expresión, [Lista_de_expresiones1], … ,[Lista_de_expresionesN])
#Iguales(a+w/b, [(d-3)*2,e,f] )  /* =2 si  (a+w/b = (d-3)*2 ) & (a+w/b =f) & (a+w/b ≠ e) */
```

- **Grupo 4 FILTER**

Esta función del lenguaje tomará como entrada una condición especial y una lista de variables y devolverá la primera variable que cumpla con la condición especificada
Condición es una sentencia de condición simple o múltiple, cuyo lado izquierdo debe ser un guión bajo que hace referencia a cada elemento de la lista de variables, y su lado derecho una expresión.
Puede ser utilizada dentro de una expresión.

Ejemplo:

```
FILTER  (Condición, [Lista_de_variables])
FILTER  ( _<=4 , [a,b,c])
FILTER  ( _>4 and _<=6.5 , [a,b,c,d])
```

Grupo 5 MAP

Tendrá como entrada un Operador que podrá ser el operador suma o multiplicación, una Constante que podrá ser entera o float y una Lista de expresiones que estarán separadas por comas y entre corchetes. Deberá dar como resultado la sumatoria de los valores de la lista de expresiones luego de aplicado a cada elemento de la lista el Operador con la constante definida. El resultado podrá ser entero o float y podrá ser utilizada dentro de las expresiones del lenguaje.

Ejemplo:

```
MAP ( OP CTE,[expresion1,expresión2,expresion3,…,expresiónN] )
```

```
a1:=12
b1:=6
c1:=3.14
result1:=MAP(+ 2,[a1+b1,c1*(b1+3),7]) /* result1 será igual a 59.26 */
result2:= 200 + MAP(+ 2,[a1+b1,c1*(b1+3),7]) /* result2 será igual a 259.26 */
```

- **Grupo 6 LET**

La sentencia LET asigna expresiones a identificadores y opera asignando cada expr (cuando estas existan) a cada id j=[1,n]. Si la expresión no existiese se tomará la expr por default (a continuación de la palabra reservada DEFAULT)

Ejemplo:

```
LET id1 : exp1, id2,..., idn : expn DEFAULT expr
LET a1 : c3*5+2, a2, a3 : b1/5-2 DEFAULT b1*7
```

- **Grupo 7 BETWEEN**

Tomará como entrada una variable numérica y dos constantes numéricas encerradas entre corchetes y separadas por punto y coma. Devolverá verdadero o falso según la variable enunciada se encuentre dentro del rango definido por ambas constantes. Esta función será utilizada en las condiciones, presentes en ciclos y selecciones.

Ejemplo:

```
BETWEEN(variable, [constante1; constante2])
BETWEEN (a, [2 ; 12]) /* Verdadero si 2<=a<=12 Falso en caso contrario */
BETWEEN (z, [2.3 ; 11.22]) /* Verdadero si 2.3<=z<=11.22  Falso en caso contrario */
a1:=12
d1:=3,14
IF  BETWEEN(a1,[10;20]) THEN
        d1 := 3,14
END IF
```