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:

1. Ingresar código de nuestro programa manualmente en un cuadro de texto adecuado a tal propósito, por ejemplo dentro del lenguaje Java JTextArea.
2. Cargar un archivo con código y poder editarlo dentro del cuadro de texto.
3. Compilar el programa ingresado (análisis léxico) y mostrar un texto aclaratorio, identificando los tokens reconocidos por el léxico u errores encontrados en el análisis. Las impresiones deben ser claras. Los elementos que no generan tokens 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** incluyendo la prueba del tema asignado al grupo
- Un archivo con la tabla de símbolos **ts.txt**
- Código fuente del proyecto
- Archivo JAR ejecutable

Todo el material deberá ser subido a algún repositorio GIT (Github, Gitlab, etc.) y su enlace enviado a teoria1.unlu@gmail.com. Indicar exactamente cuál es el archivo ejecutable jar (que se deberá abrir con botón derecho al estilo JFlap).
Asunto: GrupoXX
Fecha de entrega: 03/10/24

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, en este caso deberá permitir:

1. Cumplir mismos puntos que en la primer entrega
2. Compilar el programa ingresado (análisis sintáctico) y mostrar por pantalla un texto aclaratorio identificando las reglas sintácticas que va analizando el parser. 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** incluyendo la prueba del tema asignado al grupo
- Un archivo con la tabla de símbolos **ts.txt**
- Código fuente del proyecto
- Archivo JAR ejecutable

Todo el material deberá ser subido a algún repositorio GIT (Github, Gitlab, etc.) y su enlace enviado a teoria1.unlu@gmail.com. Indicar exactamente cuál es el archivo ejecutable jar (que se deberá abrir con botón derecho al estilo JFlap).
Asunto: GrupoXX
Fecha de entrega: 14/11/24

ANEXO TEMAS

TEMAS COMUNES

WHILE
Implementación de *While (chequeo de condición al comienzo)* utilizando el formato que el grupo desee

DECISIONES
Implementación de *IF-THEN-ELSE* utilizando el formato que el grupo desee

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:

```
WRITE "@sdADaSjfla%dfg"
var ::= "HOLA MUNDO"
```

Constantes base binaria
Constantes que comienzan con 0b y continúan con 0 y 1. Operan dentro de cualquier expresión aritmética del lenguaje

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

VARIABLES

Variables numéricas
Estas variables reciben valores numéricos tales como constantes numéricas, variables numéricas u operaciones que arrojen un valor numérico.

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. (Etapa semántica)

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:

```
write “ewr”    //* donde “ewr” debe ser una cte string *//
write 99.999 //* donde 99.999 debe ser cualquier cte numérica *//
write var   //* donde var debe ser cualquier variable numérica *//
```

CONDICIONES

Las condiciones para un constructor de ciclos o de selección, deberán ser comparaciones binarias que pueden estar ligadas por un único 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.SECTION y ENDDECLARE.SECTION, siguiendo el siguiente formato:

```
DECLARE.SECTION
            Línea_de_Declaración_de_Tipos
ENDDECLARE.SECTION
```

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

PROGRAMA

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

```
PROGRAM.SECTION
      Lista_de_Sentencias
ENDPROGRAM.SECTION
```

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 aportan información necesaria para operar con constantes y 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	—
0b011	CTE_B	—	3	—

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 ContarPrimos. (Gatica Odato Josue)

Dada una lista de expresiones (de variables y constantes), la función ContarPrimos devolverá la cantidad de números primos encontrados.

Ejemplo:

```
z::=0;
a::=2;
b::=2;
c::=53;
x = ContarPrimos([11,b+8,2,55,z*b+7,C]) // x = 4
```

Grupo 2 OPLIST (Reinoso Lucio)

El comando OPLIST, opera (sumas y multiplicaciones) sobre un primer identificador y una lista de constantes, asignando el resultado de esa operación al segundo identificador de la siguiente manera:

```
oplist [+ [a;b][2;3;5]] //si a=1 entonces b=1+2+3+5 → b = 11
oplist [* [a;b][10;3]] //si a=2 entonces b=2*10*3 → b = 60
oplist [+ [a;b][20]] //si a=3 entonces b=3+20 → b = 23
```

Grupo 3 Asignación Compleja (Nardoni Valentín)

La función asignación compleja asigna la primera posición de una segunda lista de variables a la primera variable de una primera lista de variables y asi sucesivamente. Ambas listas estan balanceadas sintácticamente.

```
Ejemplo:
AsigComp([a, b, c, d] : [1, b, 2.5, z]) // a=1, b=b, c=2.5, d=z
```

Grupo 4 Contar Distinto (!cont). (Saracho Franco)

La función especial *!cont* devuelve la cantidad de elementos cuyo valor no coincide con la expresión del primer parámetro. En el caso particular que el resultado de la expresión se encuentre en toda la lista se devuelve el numero -1. La lista del segundo parámetro solo trabaja con variables y constantes.

```
Ejemplo:
c = !cont(a+b+3/c :: [a,b,c,2,3]);
```

Grupo 5 Take (Nomico Mateo)

Calcula el valor de la función TAKE y se lo asigna a un identificador.
Esta función toma como entrada un operador de suma o multiplicación, una variable entera positiva y una lista de expresiones. Esta función devuelve el valor que resulta de aplicar el operador suma o multiplicación a los primeros “n” elementos de la lista. El valor de n quedará establecido en la componente id. Los elementos de la lista están separados por comas (,).
La lista de constantes podría ser vacía en cuyo caso se emitirá un mensaje “La lista está vacía”
El resultado será asignado al ID del lado izquierdo de la asignación

```
Formato:
id ::= TAKE(operador; id; [lista de constantes])
```

```
Ejemplos:
```

```
result::= TAKE(+; id; [3]) //* si id tomase el valor 1 devuelve 3 y se
asigna al id resul, si id resulta mayor a 1 devuelve 3 y se asigna al id
resul *//
result::= TAKE(+; id; [a+1,2*b,3]) //* si id tomase el valor 2, a y b el
valor 5,  devuelve 16, resultado de sumar las 2 primeras expresiones de la
lista y se asigna al id resul *//
result::= TAKE (*; id; [])  //* mensaje la lista esta vacía y se asigna 0
al id resul*//
```

Grupo 6 Percent(Valor,Tasa) (López María Valentina)

La función calcula el porcentaje Tasa de Valor. Valor y Tasa pueden ser expresiones numéricas. Puede ser utilizada en las expresiones del lenguaje.

```
Ejemplo:
a1::=100
c1::=10
b1::= 30+PERCENT(100+a1,2*c1)
                    /* se interpreta como 30+20 por lo tanto b1=50 */
```

Grupo 7 FILTER (Bogado Emanuel)

Esta función 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
a1::=100
b1::= 3
c1::=10
a::= FILTER  ( _<=a1*2 , [a1,b1,c1]) /* a::= a1*/
a::= FILTER  ( _<4*b1 and _>3 , [b1,c1]) /* a::= c1*/
```

Grupo 8 BETWEEN (Rodríguez Juan Cruz)

Tomará como entrada una variable numérica y dos expresiones 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 expresiones. Se deberá verificar que el resultado de la primera expresión sea menor que el resultado de la segunda expresión.
Esta función será utilizada en las condiciones, presentes en ciclos y selecciones.

```
Ejemplo:
a ::= 11
a1::= 5
b1::= 3
c1::=10

BETWEEN (a, [a1*2 ; 12]) /* Verdadero si 10<=a<=12 Falso en caso contrario */
```