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.
   - El tema especial según el número de tema asignado al grupo.
   - El método de generación intermedia que le sea especificado a cada grupo

2. Cada grupo deberá completar **DOS** carátulas con sus datos (caratula.doc), entregará una a la cátedra y se quedará con una copia. Tanto el número de tema como la grilla deberán figurar en la carátula.

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

PRIMERA ENTREGA

OBJETIVO: Realizar un analizador sintáctico utilizando las herramientas FLEX y BISON. El programa ejecutable deberá mostrar por pantalla 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.

Se deberá entregar una carpeta con nombre: **GrupoXX** que incluirá:

- El archivo flex que se llamará **Lexico.l**
- El archivo bison que se llamará **Sintactico.y**
- Un archivo de pruebas generales que se llamará **pruebal.txt** y que dispondrá de un lote de pruebas generales que abarcará todos los temas especiales y comunes.
- Un archivo con la tabla de símbolos **ts.txt**

Todo el material deberá ser enviado a: **lenguajesycompiladores@gmail.com**
Asunto : NombredelDocente_GrupoXX (Nombre de Pila del Docente)
Fecha de entrega:

SEGUNDA ENTREGA

OBJETIVO: Realizar un generador de código intermedio utilizando el archivo BISON generado en la primera entrega. El programa ejecutable deberá procesar el archivo de entrada (prueba.txt) y devolver el código intermedio del mismo junto con la tabla de símbolos.
Además deberá generar la cabecera del código ejecutable en assembler.

Se deberá entregar una carpeta con nombre: **GrupoXX** que incluirá:

- El archivo flex que se llamará **Lexico.l**
- El archivo bison que se llamará **Sintactico.y**
- Un archivo de pruebas generales que se llamará **prueba.txt** y que dispondrá de un lote de pruebas generales que abarcará todos los temas especiales y comunes.
- Un archivo con la tabla de símbolos **ts.txt**
- Un archivo con la notación intermedia que se llamará **intermedia.txt** y que contiene el código intermedio
- Un archivo con la cabecera de assembler que se llamará **Final.txt**

Todo el material deberá ser enviado a: **lenguajesycompiladores@gmail.com**
Asunto : NombredelDocente_GrupoXX (Nombre de Pila del Docente)
Fecha de entrega:

TEMAS COMUNES

ITERACIONES
Implementación de ciclo *WHILE*

DECISIONES
Implementación de *IF*

ASIGNACIONES
Asignaciones simples *A:=B*

TIPO DE DATOS
Constantes numéricas

- reales (32 bits)
- enteros (16 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, una variable string, o la concatenación de 2 (máximo) tipos strings (máximo) ya sean constantes o variables.
El operador de concatenación será el símbolo *"++"*

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 con un solo nivel de anidación

Ejemplo1:

```
-/ Realizo una selección /-
```

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

Ejemplo2:

```
-/ Así son los comentarios en el 2°Cuat de LyC /- Comentario -/ /-
```

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

ENTRADA Y SALIDA
Las salidas y entradas por teclado se implementaran como se muestra en el siguiente ejemplo:

Ejemplo:

```
WRITE “ewr”     --/ donde “ewr” debe ser una cte string
READ pivot      --/ donde pivot es una variable
WRITE var1      --/ donde var1 es una variable definida previamente
```

CONDICIONES
Las condiciones para un constructor de ciclos o de selección pueden ser simples *(a < b)* o múltiples.
Las condiciones múltiples pueden ser hasta **dos** condiciones simples ligadas a través del operador lógico *(AND, OR)* o una condición simple con el operador lógico *NOT*

DECLARACIONES

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

VAR
< Tipo de Dato > : < Lista de Variables>
ENDVAR

La *Lista de Variables* debe ser una lista de variables separadas por comas y *Lista de Tipos de Datos* debe ser una lista de tipos separadas por comas. Cada *Tipo* de la *Lista de Tipos* será el tipo de dato que adquiera cada *variable* en la *Lista de Variables* y esta equivalencia será posicional.
Si existiesen más tipos que variables, se ignorará el o los tipos sobrantes.
Si existiesen menos tipos que variables, las variables sobrantes quedarán no declaradas.
Pueden existir varias líneas de declaración de tipos

Ejemplos de formato:
```
VAR
      [Integer, Float, String] : [a, b, c]
ENDVAR
```

En este ejemplo, la variable ***a*** será de tipo Integer, ***b*** de tipo Float y ***c*** de tipo String

TEMAS ESPECIALES

1. **Fibonacci**
   La función especial Fibonacci (sucesión de Fibonacci) tendrá el siguiente formato:

   FIB (Expresion)

   Tomará como entrada un número natural "n", y devolverá el enésimo término de la sucesión, sabiendo que los dos primeros términos de la misma son 0 y 1, y cada uno de los demás es igual a la suma de los dos anteriores.

2. **Longitud**
   Esta función calcula la longitud de una lista

   long ([a,b,c,e]) = 4

3. **Asignaciones en línea múltiple**
   Las asignaciones en línea son múltiples asignaciones de la forma:

   [Lista de variables] := [Lista de Expresiones]

   cuya semántica será asignar cada variable del lado izquierdo de la asignaciones a cada expresión del lado derecho, uno a uno de izquierda a derecha respetando el orden. Si el lado izquierdo fuese, en cantidad, distinto al derecho, se ignorarán los sobrantes.

   Ej

   [a,b,c,d] := [10.4,cont,"hola",3]

4. **Take**

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

   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.
   En todo momento deberá validarse la cantidad definida en **cte** con la cantidad de elementos de la lista.

   Ej.

   TAKE (* ; 3 ; [2 ; 12 ; 24 ; 48]) **Resultado: 576
   TAKE (+ ; 2 ; [2 ; 12 ; 24 ; 48]) **Resultado: 14
   TAKE (- ; 3 ; [2 ; 12 ; 24 ; 48]) **Resultado: -34
   TAKE (/ ; 4 ; [2 ; 12 ; 24 ; 48]) **Resultado: 0
   TAKE (+ ; 3 ; [2 ; 12]) **Error
   TAKE (/ ; 4 ; [ ]) **Resultado: 0

5. **Ciclo Especial**
   La estructura de la sentencia será

   WHILE
   Variable IN [Lista de Expresiones]

   DO
   Sentencias

   ENDWHILE

   Lista de expresiones es una lista de expresiones separadas por comas.

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

NOMBRE	TIPO	VALOR	LONGITUD
a1	Real	—	
b1	Real	—	
_variable1	CteString	variable1	9
_30.5	CteReal	30.5	
		—	

Tabla de símbolos