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. Se fijarán puntos de control con fechas y consignas determinadas

3. Todos los ejecutables deberán correr sobre Windows.

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**
- El archivo ejecutable que se llamará **Primera.exe**
- 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. (*No deberán faltar selecciones y ciclos anidados, temas especiales, verificación de cotas para las constantes, chequeo de longitud de los nombres de los identificadores, comentarios*)
- Un archivo con la tabla de símbolos **ts.txt**

Todo el material deberá ser subido a algún almacenamiento (Google drive, Dropbox, etc.) y su enlace enviado a: **lenguajesycompiladores@gmail.com**

Asunto: NombredelDocente_GrupoXX (Ej Daniel_Grupo03, Facundo_Grupo12)
Fecha de entrega: 16/09/2019

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.

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**
- El archivo ejecutable que se llamará **Segunda.exe**
- 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

Todo el material deberá ser subido a algún almacenamiento (Google drive, Dropbox, etc.) y su enlace enviado a: **lenguajesycompiladores@gmail.com**

Asunto: NombredelDocente_GrupoXX (Ej Daniel_Grupo03, Facundo_Grupo12)
Fecha de entrega: 07/10/2019

ENTREGA FINAL

OBJETIVO: Realizar un compilador utilizando el archivo generado en la segunda entrega. El programa ejecutable deberá procesar el archivo de entrada (prueba.txt), compilarlo y ejecutarlo.

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**
- El archivo ejecutable del compilador que se llamará **Grupoxx.exe** y que generará el código assembler final que se llamará **Final.asm**
- Un archivo de pruebas generales que se llamará **prueba.txt** y que dispondrá de un lote de pruebas generales que abarcará :
    - Asignaciones
    - Selecciones
    - Impresiones
    - Temas Especiales

- Un archivo por lotes (**Grupoxx.bat**) que incluirá las sentencias necesarias para compilar con TASM y TLINK el archivo **Final.asm** generado por el compilador

En todos los casos el compilador **Grupoxx.exe** deberá generar los archivos **intermedia.txt** y **Final.asm**

Todo el material deberá ser subido a algún almacenamiento (Google drive, Dropbox, etc.) y su enlace enviado a: **lenguajesycompiladores@gmail.com**

Asunto: NombredelDocente_GrupoXX (Ej Daniel_Grupo03, Facundo_Grupo12)

Fecha de entrega: 11/11/2019

ATENCION: Cada grupo deberá designar un integrante para el envío de los correos durante todo el cuatrimestre.

TEMAS COMUNES

ITERACIONES
Implementación de ciclo *REPEAT*

DECISIONES
Implementación de *IF*

ASIGNACIONES
Asignaciones simples *A:=B*

TIPO DE DATOS
Constantes numéricas

- reales (32 bits)
- enteras (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"
```

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.

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:

```
                --/ 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 implementarán como se muestra en el siguiente ejemplo:

Ejemplo:

```
PRINT "ewr" --/ donde "ewr" debe ser una cte string /--
READ base      --/ donde base es una variable /--
PRINT var1 --/ donde var1 es una variable numérica 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, Integer] : [a, b, c]
ENDVAR
```

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

TEMAS ESPECIALES

1. 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]

2. Constantes Con Nombre

Las constantes con nombre podrán ser reales, enteras, string. El nombre de la constante no debe existir previamente. Se definen de la forma *CONST variable = cte,* y tal como indica su definición, no cambiaran su valor a lo largo de todo el programa.
Las constantes pueden definirse en cualquier parte dentro del cuerpo del programa.

Ejemplo:

```
--/ Constantes con Nombre /--
CONST pivot=30
CONST str =”Ingrese cantidad de días”
```

Las constantes con nombre pueden guardar su valor en tabla de símbolos.

3. 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

FILTER (Condición, [lista de variables])

Condición es una sentencia de condición simple o múltiple, cuyo lado izquierdo debe ser un guion bajo que hace referencia a cada elemento de la lista de enteros, y su lado derecho una expresión.

Lista de variables es una lista sin límite de variables

Ej.

FILTER (_>(4 + r) and _<=6.5 , [a,b,c,d])

4. INLIST

Esta función del lenguaje, tomará como entrada una variable numérica y una lista de expresiones numéricas y devolverá *verdadero* o *falso* según la variable enunciada se encuentre o no en dicha lista. La lista no puede estar vacía.
Esta función será utilizada en las condiciones presentes en cualquier estructura que requiera una condición

INLIST(variable, [lista de expresiones])

lista de expresiones serán expresiones numéricas separadas por punto y coma (;) y delimitada por corchetes

Ejemplo

INLIST (a; [2*b+7 ; 12 ; a+b*(34+d) ; 48])
INLIST (z; [2.3 ; 1.22])

5. MOD/DIV

MOD: módulo, tendrá el siguiente formato: *expresión1 MOD expresión2*. Deberá dar como resultado el resto de la división entre expresion1 y expresion2.
DIV: división entera, tendrá el siguiente formato *expresion1 DIV expresion2*. Deberá dar como resultado la división entera entre expresion1 y expresion2.

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	TIPODATO	VALOR	LONGITUD
a1	Float	—	
b1	Int	—	
_variable1		variable1	9
_30.5		30.5	
_54		54.0	

Tabla de símbolos