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**
- 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.(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 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:

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 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**

  De no ser posible el envío de un archivo ejecutable deberán renombrarse de la siguiente manera: Grupoxx_Lexico.exe como Grupoxx_Lexico.e
- Un archivo de pruebas generales que se llamará **prueba.txt** y que dispondrá de un lote de pruebas generales que abarcará :
  - Asignaciones
  - Selecciones
  - Impresiones
  - Un tema especial que se asignará por grupo

- 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

De no ser posible el envío de los archivos ejecutables deberán renombrarse de la siguiente manera:
Grupoxx.exe como Grupoxx.e
Grupoxx.bat como Grupoxx.b

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

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 base       --/ donde base 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 de la siguiente manera:

DIM < Lista de Variables> AS < Tipo de Dato >

La Lista de variables debe separarse por comas y delimitada con [], y pueden existir varias líneas DIM.
La lista de variables y la lista de tipos deben coincidir en cantidad de elementos.

Ejemplos de formato:

DIM [a, b, beta] AS [integer, real, string]
DIM [c] AS [real]

TEMAS ESPECIALES

1. AllEqual

Esta función del lenguaje tomará como entrada dos a más listas de expresiones y devolverá verdadero si cada elemento en orden posicional, es igual a su par correspondiente en cada una de las listas. Caso contrario : Falso. La cantidad de listas es indefinida.

Ejemplo:

AllEqual ([a+w,b,c], [(d-3)*2,e,f], [g,h,i]) es True si (a+w = (d-3)*2 = g) & (b = e = h) & (c=f=i)

2. #Iguales

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.

Ejemplo:

#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)

3. Operaciones con números en otras bases (Otras Bases)

Los números naturales podrán ser representados también en base 2 y 16 (binario y hexadecimal) cuya sintaxis será mediante un par ordenado donde el segundo elemento (escrito en base 10) indicará la base y el primer elemento será la representación del número en esa base.
Se deberá permitir la resolución de operaciones aritméticas entren números naturales de distintas bases.
Los números en base 10 no serán representados mediante un par ordenado.

Ejemplo:

La expresión: 20 + &F_16 – &110_ 2
debe ser interpretada como 20+15−6 = 29.

Se debe considerar error aquellos pares en los que no se represente un número válido. Por ejemplo, las expresiones: &124_2 , &G_16 , &201_2 deben ser rechazadas.

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

FILTER (Condición, [lista de variables])

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 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])

5. Punteros

Una variable de tipo puntero almacena la dirección de ***cualquier*** variable sin importar el tipo.
Los punteros deben ser declarados.
La asignación a una variable puntero, debe realizarse utilizando el operador ***ref*** que devuelve la dirección de una variable.
La asignación entre 2 variables de tipo puntero está permitida.
No existen constantes de tipo puntero o dirección, es decir que el lenguaje no permite que se pueda asignar una constante a una variable tipo puntero.

El operador ***ref*** aplicado a una constante da error.

Ej.
p1 = ref variable1;

Ejemplos de formato:

p1= ref a
p2= ref b
p1 = p2
p2 = ref 30 / ¡¡¡ERROR!!! /

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	
_&110_2	Cte	6.0	
_&F_16	Cte	15.0	

Tabla de símbolos