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.
   - Los temas especiales asignados a cada 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) que será provisto por la cátedra. 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**
- El archivo **prueba.txt** provisto por la cátedra según el tema asignado
- Un archivo con la tabla de símbolos **ts.txt**

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

Asunto: NombredelDocente_GrupoXX (Ej Daniel_Grupo03, Facundo_Grupo12)

Fecha de entrega: 12/10/2020

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**
- El archivo **prueba.txt** según el tema asignado
- 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

NOTA IMPORTANTE: Para aquellos grupos que tengan como notación intermedia árbol sintáctico deberán generarlo con la aplicación GraphViz, debiendo entregar el archivo **intermedia.png**

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: 09/11/2020

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**
- El archivo **prueba.txt** según el tema asignado

En todos los casos el compilador **Grupoxx.exe** deberá generar los archivos **intermedia.txt o intermedia.png** (según corresponda) 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: 23/11/2020

IMPORTANTE

- Cada grupo deberá designar un integrante para el envío de los correos durante todo el cuatrimestre
- No cambiar los nombres de los archivos enviados

TEMAS COMUNES

ITERACIONES
Implementación de ciclo *WHILE*

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:

```
PUT "ewr" *- donde "ewr" debe ser una cte string -*
GET base      *- donde base es una variable -*
PUT var1 *- donde var1 es una vble 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 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. 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”
```

2. Constantes en base binaria y hexadecimal

Constantes enteras que responderán al formato 0bdigitosbinarios o 0xdigitoshexadecimales. Estas constantes operarán dentro de expresiones como cualquier constante entera

Ejemplo:

```
a :=  0b111 + 35 + 0xF3A1

(Resultado para a → 62411 (obtenido de sumar 7 + 35 + 62369)
```

3. CONTAR

La sentencia permite contar la cantidad de elementos que coinciden con un pivot dentro de una lista de constantes y se lo asigna a una variable. Pivot será una expresión aritmética

Ejemplo: f1=contar (2;[2,2,2,4]) dará como resultado 3
f1=contar (5;[2,2,2,4]) dará como resultado 0

4. MAXIMO

Calcula el máximo de una lista de expresiones.

Ejemplo: maximo(3*a,b+4,3,4/5+(a*b-d))

Prueba.txt- Tema 1 : Constantes en otras bases-CONTAR

DIM < contador,promedio,actual,suma> AS < Integer,Float,Float,Float >

```
PUT "Prueba.txt LyC Tema 1!";
PUT "Ingrese un valor entero para actual: ";

GET actual;
contador: 0;
suma: 02.5+0hA2B0;
while (contador <= 92) {
 contador: contador + 1;
 actual: (contador/0.342) + (contador* contar (actual*contador ; [256,0b10,52,4]));
 suma: suma + actual;
 }

PUT  "La suma es: ";
PUT  suma;

if (actual > 0b10 and actual <> 0){
 PUT  "actual es mayor que 2 y distinto de cero";
          }

Else
}
          if (actual < 0o72)
          PUT “no es mayor que 2”
}
```

.

Prueba.txt- Tema 2 : Constantes en otras bases-MAXIMO

```
DIM < contador,promedio,actual,suma> AS < Integer,Float,Float,Float >


PUT "Prueba.txt LyC Tema 2!";
PUT "Ingrese un valor entero para actual: ";

GET actual;
contador: 0;
suma: 02.5+0hA2B0;
while (contador <= 9) {
 contador: contador + 1;
actual: (contador/0.342) + (contador* maximo (actual*contador, 2,
actual*pivot,máximo (4,actual,0o250) ) );
 suma: suma + actual;
 }

PUT  "La suma es: ";
PUT  suma;

if (actual > 0b10 and actual <> 0){
 PUT  "actual es mayor que 2 y distinto de cero";
          }

Else
}
          if (actual < 0o72)
          PUT “no es mayor que 2”
}
```

Prueba.txt- Tema 3 : Constantes con nombre-MAXIMO

```
DIM < contador,promedio,actual,suma> AS < Integer,Float,Float,Float >

CONST nombre=85;
PUT "Prueba.txt LyC Tema 3!";
PUT "Ingrese un valor entero para actual: ";

GET actual;
contador: 0;
suma: 02.5+nombre;
while (contador <= 9) {
 contador: contador + 1;
actual: (contador/0.342) + (contador* maximo (actual*contador, 2,
actual*pivot,máximo (4,actual,nombre) ) );
 suma: suma + actual;
 }

PUT  "La suma es: ";
```

```
PUT  suma;

if (actual > 2 and actual <> 0){
 PUT  "actual es mayor que 2 y distinto de cero";
          }

Else
}
          if (actual < nombre)
          PUT “no es mayor que 2”
}
```

Prueba.txt- Tema 4 : Constantes con nombre-CONTAR

```
DIM < contador,promedio,actual,suma> AS < Integer,Float,Float,Float >

CONST nombre=80;

PUT  "Prueba.txt LyC Tema 4!";
PUT  "Ingrese un valor entero para actual: ";

GET actual;
contador: 0;
suma: 02.5+nombre;
while (contador <= 92) {
 contador: contador + 1;
 actual: (contador/0.342) + (contador* contar (actual*contador ;
[256,nombre*suma,52,4]));
 suma: suma + actual;
 }

PUT  "La suma es: ";
PUT  suma;

if (actual > 2 and actual <> 0){
 PUT  "actual es mayor que 2 y distinto de cero";
          }

Else
}
          if (actual < nombre)
          PUT “no es mayor que 2”
}
```

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

Tabla de símbolos

NOMBRE	TIPO	VALOR	LONGITUD
a1	Float	—	
b1	Int	—	
_variable1		variable1	9
_30.5		30.5	