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**

Recomendaciones:

- En caso de realizar su compilador en C, por favor evitar el uso de librerías no estándar (como por ejemplo conio.h, que solo es soportada en Windows).
- 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, ciclos anidados, temas especiales, verificación de cotas para las constantes, chequeo de longitud de los nombres de los identificadores, comentarios*, etc.
- El archivo de prueba debe ser *único* (no enviar diferentes escenarios de prueba en diferentes archivos).
- Las líneas de código que ejemplifican casos de error en tiempo de compilación deberán presentarse en el documento de manera comentadas y acompañadas de un mensaje descriptivo.
- 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, Eleazar_Grupo02, Facundo_Grupo15)

Fecha de entrega: 25/04/2022

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 repositorio GIT (GitHub, GitLab, etc) y su enlace enviado a: **lenguajesycompiladores@gmail.com**

Asunto: NombredelDocente_GrupoXX (Ej Daniel_Grupo03, Eleazar_Grupo02)
Fecha de entrega: 23/05/2022

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 repositorio GIT (GitHub, GitLab, etc) y su enlace enviado a: **lenguajesycompiladores@gmail.com**

Asunto: NombredelDocente_GrupoXX (Ej Daniel_Grupo03, Eleazar_Grupo02)

Fecha de entrega: 06/06/2022
ATENCIÓN: Cada grupo deberá designar un integrante para el envío de los correos durante todo el cuatrimestre.

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 de LyC/-
```

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:

```
WRITE "ewr" -/ donde "ewr" debe ser una cte string /-
READ base      -/ donde base es una variable /-
WRITE 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 DECVAR y ENDDEC, siguiendo el siguiente formato:

DECVAR
Línea_de_Declaración_de_Tipos
ENDDEC

Cada *Línea_de_Declaración_de_Tipos* tendrá la forma: < *Lista de Variables*> : *Tipo de Dato*

La *Lista de Variables* debe ser una lista de variables separadas por comas.
Pueden existir varias líneas de declaración de tipos, incluso utilizando más de una línea para el mismo tipo.

Ejemplos de formato:

```
DECVAR
        a1, b1 : FLOAT
        variable1 : STRING
                p1, p2, p3 : FLOAT
ENDDEC
```

TEMAS ESPECIALES

1. BETWEEN

Esta función del lenguaje, tomará como entrada una variable numérica y dos expresiones numéricas. Devolverá verdadero o falso según la variable enunciada se encuentre dentro del rango definido por ambos límites.
Esta función será utilizada en las condiciones, presentes en ciclos y selecciones.

BETWEEN(variable1, [expresion1; expresion2])

Ejemplo:
BETWEEN (a, [2 ; a*(b+4)])
BETWEEN (z, [2.3 ; 11.22])

2. AVG

Esta función del lenguaje, tomará como entrada una lista de expresiones numéricas y devolverá el promedio calculado a partir de la evaluación de los componentes de dicha lista.
Esta función será utilizada en cualquier expresión del lenguaje.

AVG([lista de expresiones])

Donde lista de expresiones será una secuencia de expresiones numéricas separadas por coma (,) y delimitada por corchetes

Ejemplo:
AVG ([2 , a+b , c*(d+e) , 48])
AVG ([2.3 , 1.22])

3. 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])

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 /*
```

TABLA DE SÍMBOLOS

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, etc.
Ejemplo

NOMBRE	TIPODATO	VALOR	LONGITUD
a1	Float	_	
b1	Int	_	
_variable1		variable1	9
_30.5		30.5	
_0b111		7	
_0xF3A1		62369	

Tabla de símbolos