TRABAJO PRÁCTICO COMPILADOR

CONSIDERACIONES GENERALES

Es necesario cumplir con las siguientes consideraciones para evaluar el TP.

1. Cada grupo deberá tener una carpeta con hojas formato A4 con la siguiente estructura:

   Carátula (punto 2)
   BNF del lenguaje
   Autómata finito del AL
   Tablas de estados y eventos del AL.
   Todas y cada una de las hojas sueltas en las que el docente escriba alguna explicación

   No es necesario traer el código impreso a menos que el docente lo pida. Las correcciones y diferentes versiones deben figurar todas en la carpeta

2. 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

3. 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.

4. Se fijarán puntos de control y fechas de reexaminación para cada grupo(Ver fechas de puntos de control). El grupo que no se presente en la fecha indicada pierde su oportunidad de ser evaluado y orientado en el desarrollo del TP. Futuras correcciones y orientaciones estarán sujetas a la disponibilidad de docentes..

5. El uso del laboratorio es prioritario para los grupos que son evaluados.

PRESENTACIÓN

NOTACIÓN INTERMEDIA

1. Forma de mostrar la notación intermedia:

   Tercetos: Los tercetos se mostrarán en pantalla en orden de creación, NUMERADOS. Aquellos tercetos que tengan como operando el resultado de otro terceto debe mostrarlo con el nro. de terceto entre corchetes.
   Ej:

   *2 (23, var1, *)*
   3 (var2, [2], :=)

   Árbol: Para mostrar el árbol es necesario que cada nodo tengo un nro. de identificador UNICO. Los nodos se mostrarán recorriendo el árbol IN-ORDEN con el siguiente formato **nodo X: hijo izq, valor, hijo derecho.** En los campos hijo izq. & derecho debe ir el nro. de nodo de dicho hijo, si no existiesen hijos deberá figurar "-". El campo valor posee el nombre de la variable o cte con nombre, el valor del número o la operación que alberga dicho nodo.
   Ej:

   *nodo 2: 3, *, 4*
   nodo 3: -, 23, -
   nodo 4: -, var1, -

   Polaca: Se deberá mostrar el array que contiene la traducción del archivo fuente a polaca inversa con todos los operando y operadores, saltos y etiquetas. Cada elemento del array deberá presentarse separado por comas.
   Ej:

   *var2, 23, var1, *, :=, ….*

***NOTA 1:**
Todo el código en notación intermedia deberá estar en un archivo TXT llamado ***Intermedia.TXT.***
El sector de declaraciones no deberá ser traducido a notación intermedia

Cualquier información que el grupo juzgue conveniente indicar, para facilitar la traducción a assembler debe mostrarse también.

GENERACIÓN DE CÓDIGO

Para la generación de código assembler se deberá:

- Usar el coprocesador matemático con sus respectivas instrucciones para todas las operaciones
- Realizar la traducción a Assembler tomando como INPUT el archivo ***Intermedia.TXT*** que contiene la notación intermedia y producir como OUTPUT un archivo llamado ***FINAL.ASM*** que contenga el código Assembler completo.

- Compilar el archivo Assembler ***FINAL.ASM*** con cualquier compilador Assembler que reconozca sus instrucciones (TASM, NASM, DASM, etc.).

- Efectuar la traducción a Assembler sólo en la regla del Start Symbol. No deberá existir ninguna otra función que traduzca a Assembler en el resto de las reglas.

***NOTA 2 :**
Todos los grupos deberán tener varios lotes de pruebas para la entrega final.

TEMAS COMUNES A TODOS LOS GRUPOS

- Iteraciones: **for** (delimitado con las palabras reservadas FOR y ROF)
- Decisiones **If** (delimitado con las palabras reservadas IF y FI, IF ELSE FI)
- Asignaciones **simples y múltiples** (de la forma a=b=c=d=2000)
- Tipos de datos numéricos: Reales 32 bits y Enteros 16 bits
  Ejemplos de Formato para reales:

```
99999.99
99.
.9999
```

  Deberá utilizarse obligatoriamente el punto "." como separador decimal.

- Constantes String
  Constantes de 30 caracteres alfanuméricos como máximo, limitada por comillas (" ") ,de la forma "xxxxx"

  Ejemplos:
```
"asldkfhsjf "
"@sdADaSjfla%dfg"
```

- Variables String
  Las variables tipo string pueden recibir en una asignación una *cte string (como la detallada más arriba)*, *una variable string* o *la concatenación de 2 (máximo dos) strings (cte o variable)*
  El operador de concatenación será el símbolo "++"

- Comentarios
  Deberán estar delimitados por los caracteres "-/" y "/-" en forma de espejo.

  Ejemplos:
```
-/ Realizo una selección /-
IF (a <= 30)
      b = "correcto" -/ asignación string /-
ENDIF
```

- Output
  WPRINT("ewr") , WPRINT(345.98) donde pueden imprimirse constantes strings o reales.
  WPRINT(x) para la variable *x* de cualquier tipo.

- Validaciones
  Las operaciones y asignaciones deben hacerse entre elementos del mismo tipo; en caso que sean entre tipos diferentes debe mostrar un error.

- 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
  [Lista de variables] : [Lista de tipos de datos]
  ENDVAR

  La *Lista de variables* debe ser una lista de identificadores separados por comas y *Lista de tipos de datos* debe ser una lista de tipos separados 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 ignorarán el o los tipos sobrantes.
Si existiesen menos tipos que variables, las variables sobrantes quedarán no declaradas.
Podrán existir varias líneas de declaración de tipos pero **solo un bloque de declaración al comienzo del programa.**

Se debe tener en cuenta que será válido aquel programa que no posea bloque de declaración (Ejemplo: un programa que posea solo sentencias WPRINT de constantes string)

Ejemplos de formato:

```
VAR
      [a, b, c] : [Integer, Float, String]
      -/ … otras líneas de declaración /-
ENDVAR
```

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

TEMAS ESPECIALES

1 – CTES. OTRAS BASES - CONVERSIONES	4 – DO WHILE – CONVERSIONES
2 – ASG.EN LÍNEA - FILTERC	5 – CTES. EN OTRAS BASES – ASG. EN LINEA
3 – DO WHILE - FILTERC	

Descripción de temas:

1. CONSTANTES EN 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.

   Las constantes en otras bases serán reconocidas en el **analizador sintáctico**

2. ASIGNACIÓN EN LÍNEA
   Se debe permitir realizar una asignación dentro de una expresión, en donde el valor a tomar será el del identificador a la izquierda de la asignación luego de realizar la misma.

```
var1= 13
var5= {n=29} + var1  -/n recibe el valor de 29 y es sumado a var1 /-
var4= {n= var4 +1}
```

   La asignación debe estar entre llaves, y permitir una expresión del lado derecho.

3. FILTERC
   Esta función del lenguaje tomará como entrada una condición especial más una lista de expresiones y devolverá **la cantidad** de expresiones que cumplen con la condición especificada

```
FILTERC  (condición, [lista de expresiones] )
```

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 expresiones, y su lado derecho una expresión.
Lista de expresiones no posee límite.

```
FILTERC  ( _>4 and _<=6.5 , [a+c,b,c-a,d])
```

4. CONVERSIONES

Se podrán utilizar funciones especiales para la conversión de valores de temperatura.

FtoC(g) g está expresada en Farenheit y devuelve Celsius
CtoF(g) g está expresada en Celsius y devuelve Farenheit
KtoC(g) g está expresada en Kelvin y devuelve Celsius
CtoK(g) g está expresada en Celcius devuelve Kelvin

Dónde:

Temperatura en ºC = (ºF -32)/1,8
Temperatura en ºF = 1,8 ºC + 32
Temperatura en ºK = ºC + 273,14

Se podrá operar con funciones de conversión dentro de expresiones.

5. DO WHILE

Sera una iteración cuya sentencia se cumple por lo menos una vez.

Ejemplo de implementación de tabla de símbolos en el análisis léxico.

NOMBRE	TIPO	VALOR	LONGITUD
var1	-	-	
var3	-	-	
_variable1	cteString	variable1	**9**
_30.5	cte	30.5	
s1	-	-	
_20	cte	20	
F			
_16	cte	16	
_110	cte	110	
_2	cte	2	

Ejemplo de implementación de tabla de símbolos en el análisis sintáctico.

NOMBRE	TIPO	VALOR	LONGITUD
var1	**real**	-	
var3	**int**	-	
_variable1	cteString	variable1	**9**
_30.5	cte	30.5	
s1	**string**	-	
_20	cte	20	
F			
_16	cte	16	
_110	cte	110	
_2	cte	2	
_15	cteH	15	
_6	cteB	6	