TRABAJO PRÁCTICO COMPILADOR

CONSIDERACIONES GENERALES

Es necesario cumplir con las siguientes consideraciones:

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. 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.

3. Se fijarán puntos de control con fechas y consignas determinadas.

PRIMERA ENTREGA COMPILADOR

Se deberá entregar una carpeta con nombre: **GrupoXX** que incluirá:

- El archivo ejecutable del analizador lexicográfico que se llamará **Grupoxx.exe**
- Un archivo de pruebas generales que se llamará **pruebagral.txt** y que dispondrá de un lote de pruebas generales que abarcará todos los temas especiales y comunes.
- El compilador **Grupoxx.exe** deberá compilar en todos los casos un programa que se llamará **prueba.txt** y será igual o más pequeño que **pruebagral.txt**
- De no ser posible el envío de un archivo ejecutable deberán renombrarse de la siguiente manera: Grupoxx.exe como Grupoxx.e

El ejecutable deberá devolver la lista de tokens reconocidos.

Todo el material deberá ser enviado a: lenguajesycompiladores@gmail.com
Asunto: NOMBREDELDOCENTE-ENTREGA1-GRUPOXX (Nombre de Pila del Docente)
Fecha de entrega: 20/04/2015

SEGUNDA ENTREGA COMPILADOR

Se deberá entregar una carpeta con nombre: **GrupoXX** que incluirá:

- El archivo yacc que se llamará **Grupoxx.y**
- El archivo C (o archivo fuente generado por yacc, cualquiera sea el lenguaje) que se llamará **Grupoxx.c**
- El ejecutable que se llamará **Grupoxx.exe**
- Un archivo de pruebas generales que se llamará **pruebagral.txt** y que dispondrá de un lote de pruebas generales que abarcará todos los temas especiales y comunes.
- De no ser posible el envío de un archivo ejecutable deberán renombrarse de la siguiente manera: Grupoxx.exe como Grupoxx.e

Todo el material deberá ser enviado a: lenguajesycompiladores@gmail.com
Asunto: NOMBREDELDOCENTE-ENTREGA2-GRUPOXX (Nombre de Pila del Docente)
Fecha de entrega: 18/05/2015

ENTREGA FINAL COMPILADOR

Se deberá entregar una carpeta con nombre: **GrupoXX** que incluirá:

El archivo yacc que se llamará **Grupoxx.y**
El archivo C (o archivo fuente generado por yacc, cualquiera sea el lenguaje) que se llamará **Grupoxx.c**
El ejecutable que se llamará **Grupoxx.exe**
Un archivo de pruebas generales que se llamará **pruebagral.txt** y que dispondrá de un lote de pruebas generales que abarcará todos los temas especiales y comunes.

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

El compilador **Grupoxx.exe** deberá compilar en todos los casos un programa que se llamará **prueba.txt** y será igual o más pequeño que **pruebagral.txt**

De no ser posible el envío de un archivo ejecutable 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**

Si en la compilación de assembler se incluyen rutinas de input/output deberán ser especificadas en el archivo por lotes **Grupoxx.bat**

Todo el material deberá ser enviado a: lenguajesycompiladores@gmail.com
Asunto: NOMBREDELDOCENTE-ENTREGAFINAL-GRUPOXX (Nombre de Pila del Docente)
Fecha de entrega: 06/07/2015

PRESENTACIÓN

NOTACIÓN INTERMEDIA

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, -

(Deberá utilizarse un programa graficador para obtener una lectura más clara del árbol, como por ejemplo graphviz-2.34)

Polaca inversa: 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.

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 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.

CONSTANTES CON NOMBRE
Las constantes con nombre podrán ser reales, enteras o string. El nombre de la constante no debe existir previamente. Se definen de la forma *CONST tipo 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:

```
--/ Realizo una selección /--
```

```
CONST pivot=30
CONST str=”Ingrese cantidad de días”
```

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

COMENTARIOS
Deberán estar delimitados por “- -” y “- -“ y podrán estar anidados con un solo nivel de anidamiento

Ejemplo1:

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

Ejemplo2:

```
--/ Así son los comentarios en el 1°Cuat-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:

```
PUT “ewr”    --/ donde “ewr” debe ser una cte string /--
GET pivot    --/ donde pivot es una variable /--
PUT 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 deben ser declaradas.

Ejemplo:

DECLARE

(<id> : <type>,...,<id> : <type>)

ENDDDECLARE

La lista de variables debe separarse por comas, y pueden existir varias líneas de declaración, pero la zona de declaración será única, y al comienzo del programa.
Podrá existir un programa que no posea zona de declaración.
Se debe arrojar un error si se utiliza una variable que no se encuentra declarada previamente.

TEMAS ESPECIALES

1. **LET**

   LET id1: exp1, id2,..., idn : expn DEFAULT expr

   La sentencia LET asigna expresiones a identificadores y opera asignando cada id_i a cada $expr_i$ cuando estas últimas existan. Si la expresión no existiese se tomará la expresión por default (a continuación de la palabra reservada DEFAULT)

2. **CASE**

   La sentencia CASE tiene el siguiente formato

```
case expr0 of
      id1 => expr1;
      . . .
      idn => exprn;
esac
```

3. **UNARY IF**

   Un UNARY IF tendrá el siguiente formato

   ID = (<condicion> ? <exp_verdadera> ; <exp_falsa>)

   Y operará de la siguiente manera:
   Se evaluará la condición <condicion>
   Si la condición es verdadera, ID = <exp_verdadera>
   Si la condición es falsa, ID = <exp_falsa>

4. **QEqual**

   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 elementos de la lista es indefinida.
   Ej.

   `QEqual( a+w/b, [(d-3)*2,e,f] )` será 2 si (a+w/b = (d-3)*2) y (a+w/b =f) y (a+w/b ≠ e)

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 y funciones. A modo de ejemplo, se grafica una tabla de símbolos:

NOMBRE	TIPO	VALOR	LONGITUD
a1	Real	—	
b1	Enteral	—	
_variable1	CteString	variable1	9
_30.5	CteReal	30.5	
a	Real	—	
y	Const_R	—	
b	String	—	
z	Const_S		

Tabla de símbolos