Proyecto Compiladores e Intérpretes

El proyecto del semestre de la materia compiladores e intérpretes consiste en modificaciones que se deben realizar al código fuente del compilador del lenguaje Tiny construido en la materia:

https://github.com/xiul/tiny

Esto se debe realizar en equipos de 4 personas máximo, no importa si las personas ven clases en secciones diferentes de la misma materia. **Se debe hacer todo en un repositorio de git**, usando para esto un fork de github del compilador entregado por mí, donde cada usuario del proyecto debe hacer un commit del trabajo realizado y por lo tanto debe tener un usuario en github.

Las modificaciones a realizar:

1. Actualmente las listas de sentencias en tiny se encuentran separadas por punto y coma, convertir dicha acción a finalizadas en punto en coma (si revisan las clases allí incluso se les enseño la solución a ese detalle)

2. Se debe añadir declaraciones de variables estilo lenguaje C: tipo_dato variable(s) separadas por coma en caso de ser más de una. Los tipos de datos aceptados serán **boolean** e **int.** Se pueden declarar variables en cualquier posición pero deben ser declaradas antes de su uso y estarán atadas a las reglas de ámbito explicadas posteriormente en este documento. No se podrán inicializar las mismas en el mismo momento de declararlas sino posteriormente.

   Ejemplos:
   - int a;
   - int a,b,c,d,e,f;

3. Modificar la sintaxis de tiny haciéndolo pasar de un lenguaje con un bloque único de ejecución a uno que posee funciones y procedimientos donde:

   - La firma de las funciones/procedimientos será:
     - Tipo_dato nombre (parámetro(s))
       - *Tipos de datos*: **boolean**, **int** y **void** (procedimiento), en caso de ser int o boolean el tipo de dato, la función debe poseer en su interior al menos una instrucción **return** variable o expresión donde se retornara el valor resultante de la función.
       - *Nombre*: sigue el mismo estándar de nombre que los identificadores en tiny
       - *Parámetros*: pueden ser cero o más parámetros separados por comas en donde cada parámetro posee la estructura tipo_dato nombre. **POR SIMPLIFICACION NO SE ACEPTARAN VECTORES COMO PARAMETROS DE LAS FUNCIONES/PROCEDIMIENTOS**
         Ejemplos:
         - void procedimiento()

        - void procedimiento(int a, boolean b)
        - boolean funcion() //debe incluir return
        - boolean funcion(int a, boolean b) //debe incluir return
    - Los procedimientos o funciones poseen una lista de sentencias limitadas entre las palabras claves **begin** y **end**.

- La llamada de una función estará compuesta por el nombre de la función más sus parámetros en cualquier contexto posible, siendo los parámetros valores de variables, constantes numéricas o expresiones u otras funciones.

  Ejemplo
    - x=función_retorna_int(p1,p2…) donde x es int;
    - procedimiento(p1,p2);
    - x= función_retorna_int(p1,p2+funcion2(a,b),4…)

- La estructura de un programa de tiny debe ser ahora:

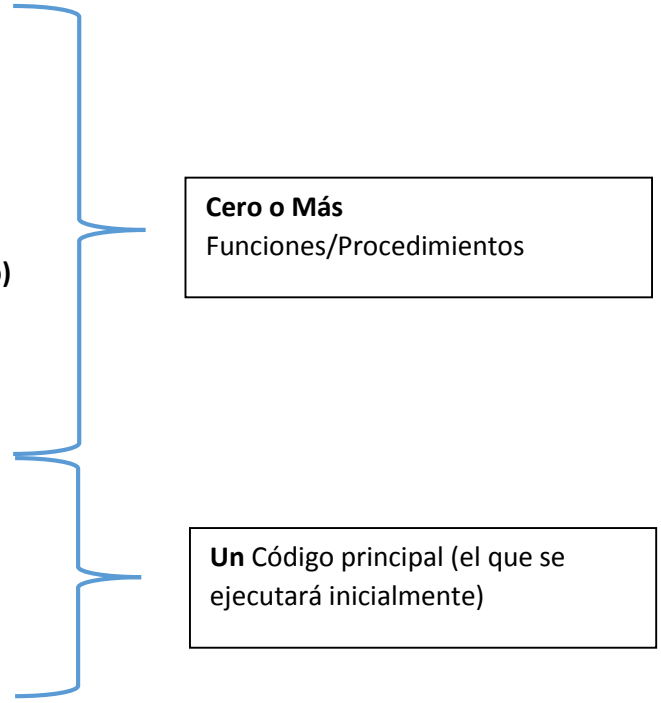
Cero o Más Funciones/Procedimientos
Un Código principal (el que se ejecutará inicialmente)

Ejemplo:

```
void procedimiento()
begin
        sentencia(s)
        …
end

boolean funcion(int a, boolean b)
begin
        sentencia(s)
        … return …;
end

begin
        sentencia(s)
end
```



- Debe tratar a las funciones como variables que almacenan en su interior la dirección de la memoria de instrucciones iMem (número de línea) donde se comienza a ejecutar el código del procedimiento. Por lo que para ejecutar dicho código podría hacer lo que dice en el libro de Louden en la página 457 punto 8 (JSUB) pasando los parámetros en la pila temporal de valores temporales (la asociada a la variable ***desplazamientoTmp*** en el código suministrado).

- El entorno de ejecución para las llamadas seguirá siendo idéntico al de tiny en cuanto al manejo de memoria, compartiendo ahora el espacio de almacenamiento del mismo entre todas sus funciones, procedimientos y el cuerpo principal. Pero cada variable será solo accesible en su ámbito de declaración (ver punto de ámbitos) por lo que no existirán registros de activación, es decir los programas no podrán ejecutarse de forma recursiva. Esto significa por ejemplo que si poseemos una declaración de variables como en este ejemplo las mismas serian distribuidas en memoria por su orden de declaración así:

```
void A()
begin
        int x;
       bolean y,z;
        …
end

boolean B(int a, boolean b)
begin
        int x,w;
        … return …;
end

begin
        int x;
        sentencia(s)
end
```

Recuerde que la tabla de símbolos guardara direcciones de dMem para las variables/parámetros y de iMem para las funciones.

	1023	
	1022	
	1021	
…		
	9	Variable x ámbito cuerpo principal
	8	Variable w ámbito función
	7	Variable x ámbito función
	6	*Parámetro a ámbito función*
	5	*Parámetro b ámbito función*
	4	Dirección iMem función B
	3	Variable z ámbito procedimiento
	2	Variable y ámbito procedimiento
	1	Variable x ámbito procedimiento
	0	Dirección iMem procedimiento A

Ver clase de generación de código para tiny si se tiene alguna duda extra.

0	1	2	3	4	5	6	7
AC	AC1	libre	libre	libre	GP	MP	PC

- En cuanto al uso de los registros será idéntico al de tiny pero ahora haciendo uso de los registros vacíos (2,3,4) para llevar a cabo el procedimiento de llamada de la función/procedimiento, retorno a la línea donde se encontraba ejecutándose y valor retornado. Esto se describirá a continuación en palabras simples sin desvelar detalles de cómo debe implementarlo ni cómo usar los registros disponibles y con valores de dirección de memoria de instrucciones aleatorios:

  - Una vez generado el código objeto del ejemplo anterior de acuerdo al modelo planteado el mismo poseerá varias partes:
  - ***iMem***

[direcciones 0-10] Preludio Estándar
[direcciones 11-20] Código procedimiento A
[direcciones 21-30] Código función B
[direcciones 31-50] Código cuerpo principal

  - Cuando un programa inicia el preludio estándar debe inicializar todo lo necesario y al finalizar debe guiar la ejecución del programa a la primera línea ejecutable del cuerpo principal es decir la **31**.
  - Una vez en ejecutando las instrucciones del cuerpo principal, cuando ocurra una llamada por ejemplo a la función B, se debe recordar en que línea se encontraba para volver allí cuando se devuelva de la función por ejemplo se hizo la llamada en la línea **32** por lo que debería retornar a ejecutar la **33**, este valor puede ser almacenado por ejemplo en un registro de los no usados.
  - Esta llamada a la función B deberá ir acompañada de parámetros, que como pueden ser diversos, lo primero que debe ocurrir es su cálculo para luego ingresarlos en la pila de valores temporales (asociada a ***desplazamientoTmp***) el valor correspondiente al mismo, por ejemplo si el parámetro es 2+3 en la pila debo introducir es el valor 5.
  - Una vez almacenada la dirección de retorno y cargados tantos parámetros como posea la función en la pila de valores temporales dichos valores deben ser colocados en la dirección de memoria correspondiente a cada parámetro en la dMem (se obtiene de la tabla de símbolos).
  - Ahora se debe ir a la línea de ejecución de dicho procedimiento/función (**21**) la cual debería estar disponible en la tabla de símbolos (es decir la tabla de símbolos guardara direcciones de dMem para las variables/parámetros y de iMem para las funciones) lo que se puede hacer con el procedimiento JSUB mencionado anteriormente.
  - Una vez allí se ejecuta el código normalmente (recuerde no hay registro de activación en este ambiente de ejecución).

        - Y cuando llegue al final de la ejecución del código de la función si se retorna un valor podría colocar este en un registro de los no usados y devolverse a la dirección de iMem que almaceno el inicio (**33**) en un uno de los registros no usado.

    - En cuanto al código generado en tiny deben tomar en cuenta que ahora el preludio al iniciar debe hacer un salto a la ubicación en la que comience la primera instrucción (iMem) a ejecutar del cuerpo principal saltando las pertenecientes a procedimientos y funciones y solo ir al código de las funciones cuando ocurra una llamada a las mismas.

En este punto es Importante, informar que esta explicación no muestra el detalle de lo que debe implementar, sino es una guía orientadora para que ud. como estudiante consiga la solución por sus propios medios mediante lo visto en clases y lo leído en la bibliografía suministrada.

4. Se debe implementar ámbitos (scopes) en las declaraciones de variables, tomando como determinantes de ámbitos solamente funciones/procedimientos y el cuerpo principal.

    - Esto significa que en un programa con una función, un procedimiento y el cuerpo principal, podría declarar la variable x en cualquiera de ellos (3 veces máximo una en cada uno) y ser variables diferentes (diferente dirección de memoria).

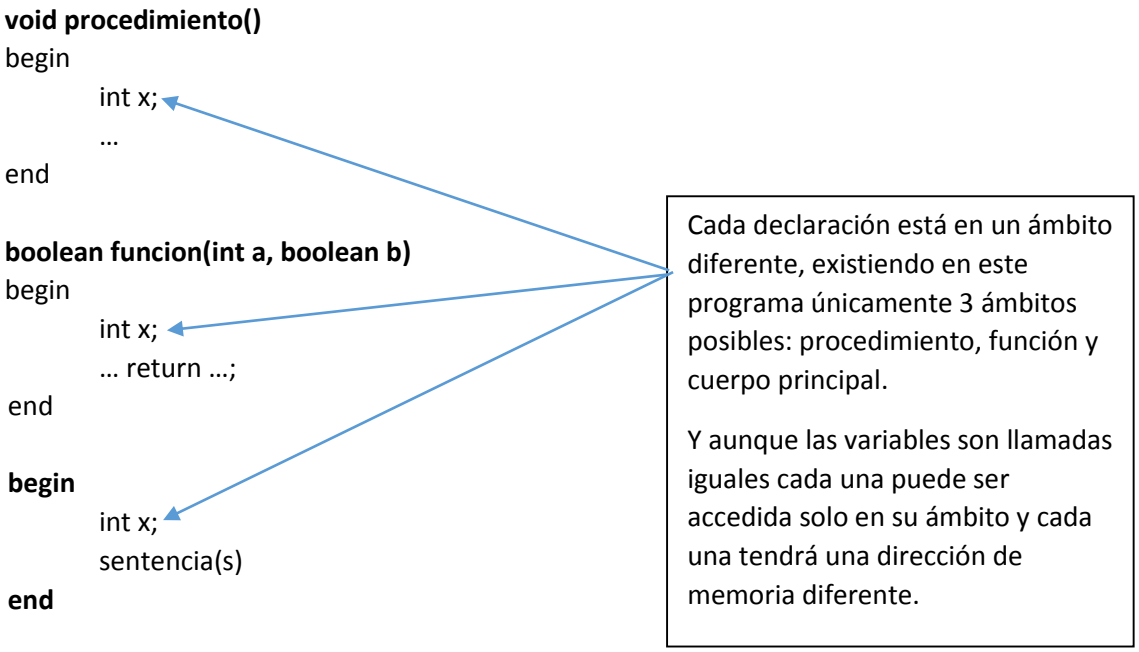
    - Cada variable será accesible por el programa para su uso solo en el ámbito declarado.

    - Esta adición de ámbitos requiere replantear la forma en que se lleva a cabo la gestión de la tabla de símbolos, siendo la manera de solucionar este problema decisión del equipo.

    - Para disminuir la complejidad del manejo de ámbitos **puede asumir** que ninguna función o procedimiento podrá llamarse como una variable declarada en el programa.

    - Los parámetros deben asumirlos como variables declaradas en la función pero que se inicializan con valor pasados en su llamada.

Ejemplo:



5. Debe añadir un analizador semántico a tiny (ver como se construye la tabla de simbolos como punto de partida) para que compruebe:

   - Tipos de datos compatibles en llamada a funciones y asignaciones.
   - Uso de variables no inicializadas en expresiones.

6. Debe añadir el manejo de vectores a tiny. Para eso la declaraciones de los mismos se hará añadiendo corchetes con una constante numérica en su interior que representa su tamaño. Se declararan tal cual y como cualquier otra variable, siendo el acceso al mismo manejado como se enseñó en clase. Su primer valor será accedido en la posición 0 (cero)

   Ejemplos:
   - int x[5]; //ocupa 5 posiciones de la 0-4
   - int y[6]; //ocupa 6 posiciones de la 0-5

7. Debe añadir el ciclo for al lenguaje, bajo la sintaxis:
    - for (inicialización ; comprobación ; paso)
      …
      end
        - Donde cada parte es una expresión valida de acuerdo a su tipo
            - Inicialización es una asignación
            - Comprobación debe ser boolean
            - Paso debe ser una asignación
            - Ejemplo
              for(x=0 ; x<10 ; x=x+1)
              …
              end

8. Debe añadir los operadores relacionales **and** y **or** con su prioridad correspondiente a las expresiones generables en tiny, así como también los operadores faltantes: != (diferente), >, >=, <=.

9. Debe permitir que el analizador léxico acepte números negativos.

10. Nunca debe mostrar al usuario un código objeto si se encontró algún error durante cualquier análisis (léxico, sintáctico o semántico) solo informando que no se emitió dicho código y si es posible sus causas.

Con entregas parciales:

- Modificaciones de la sintaxis para el lunes 13 de Abril.
- Tabla de símbolos y Analizador semántico para el Viernes 17 de Abril

Entrega final Miércoles 22 de Abril del 2015

EL NO CUMPLIMIENTO DE LAS ENTREGAS PARCIALES IMPLICARA LA PERDIDA DE LA NOTA ASOCIADA A LAS PARTES A EVALUAR ASI ESTAS SEAN COMPLETADAS POSTERIORMENTE PARA PODER FINALIZAR EL PROYECTO DE LA MATERIA SIN DERECHO A NINGUN RECLAMO POSTERIOR.