# Diseño de Compiladores I – Cursada 2018
# Trabajo Práctico Nro. 3

La entrega se hará en forma conjunta con el trabajo práctico Nro. 4
**OBJETIVO:** Se deben incorporar al compilador, las siguientes funcionalidades:

## Generación de código intermedio

- Generación de código intermedio para las sentencias ejecutables. Es requisito para la aprobación del trabajo, que el código intermedio sea almacenado en una estructura dinámica. La representación puede ser, según la notación asignada a cada grupo:
  - ***Árbol Sintáctico*** (grupos 1, 6, 14, 18 y 20)
  - ***Tercetos*** (grupos 4, 5, 7, 8, 10, 12, 15 y 17)
  - ***Polaca Inversa*** (grupos 2, 3, 9, 11, 13, 16 y 19)

  Se deberá generar código intermedio para todas las sentencias ejecutables, incluyendo:
  - Asignaciones, Selecciones, Sentencias de control asignadas al grupo, y Sentencias **print.**

## Incorporación de información a la Tabla de Símbolos (en general):

- Incorporar la siguiente información a la Tabla de Símbolos:
  - ***Tipo:***
    - Para los Identificadores, se deberá registrar el tipo, a partir de las sentencias declarativas.
    - Para las constantes, se deberá registrar el tipo durante el Análisis Léxico.
  - ***Uso:***
    - Para los identificadores, se deberá registrar el uso del identificador en el programa, cuando un identificador puede tener diferentes usos. Por ejemplo nombre de variable, y nombre de función.
  - ***Otros Atributos:***
    - Se podrán Incorporar información adicional a las entradas de la Tabla de Símbolos, de acuerdo a los temas particulares asignados

## Chequeos semánticos:

- Se deberán detectar, informando como error:
  - Variables no declaradas (según reglas de alcance del lenguaje).
  - Variables redeclaradas (según reglas de alcance del lenguaje).
  - Funciones no declaradas (si corresponde).
  - Otros chequeos relacionados con los temas particulares asignados
- Chequeo de compatibilidad de tipos:
  - Temas 14 y 15: Sólo se permitirán operaciones con operandos de distinto tipo, si se efectúa la conversión que corresponda (implícita o explícita según asignacion al grupo).
  - Tema 16: Sólo se podrá efectuar una operación (asignación, expresión aritmética, comparación, etc.) entre operandos del mismo tipo. Otro caso debe ser informado como error.

  ***Nota:***
  - Para Tercetos y Árbol Sintáctico, este chequeo se debe efectuar durante la generación de código intermedio
  - Para Polaca Inversa, el chequeo de compatibilidad de tipos se debe efectuar en la última etapa (TP 4)

## Temas particulares

### *Sentencias de Control (Temas 7, 8, 9 y 10)*

- ***while*** (tema 7 en TP1)
  **while (**<condicion> **)** <bloque_de_sentencias>**.**

  El bloque se ejecutará mientras la condición sea verdadera

- ***loop until*** (tema 8 en TP1)
  **loop** <bloque_de_sentencias> **until (**<condicion>**).**

  El bloque se ejecutará hasta que la condición sea verdadera

- ***for*** (tema 9 en TP1)

  **for** (i **:=** n; <condición>; j) <bloque_de_sentencias>,

  El bloque se ejecutará hasta que se cumpla la condición.

  En cada ciclo, el incremento de la variable de control será igual a j.

  Se deberán efectuar los siguientes chequeos de tipo:
  - i debe ser una variable de tipo entero (temas 1-2-3-4).

- n y j serán constantes de tipo entero (temas 1-2-3-4).
- <condición> será una comparación de i con m. Por ejemplo: i < m
  Donde m puede ser una variable, constante o expresión aritmética de tipo entero (1-2-3-4).

***Notas***:

- Para los grupos que tengan asignados 2 tipos enteros, considerar el "más chico" (por ej. **integer**, para grupos con **integer** y **linteger**)
- Las restricciones de tipo serán chequeadas en esta etapa, o en la siguiente, según la representación intermedia asignada.

- ***case do*** (tema 10 en TP1)

  **case**( variable ){
  valor1**: do** <bloque> **,**
  valor2**: do** <bloque> **,**
  **...**
  valorN**: do**<bloque>**,**
  **},**

***Nota***: Los valores valor1, valor2, etc., sólo podrán ser constantes del mismo tipo que la variable. Esta restricción será chequeada en esta etapa, o en la siguiente, según la asignación de representación intermedia correspondiente.

***Temas 11, 12 y 13***

- ***Mutabilidad*** (Tema 11 del TP1)

  ***Registro de información en la Tabla de Símbolos***
  En esta etapa, se deberá registrar:
  - Mutabilidad o inmutabilidad de una variable, a partir de su declaración
  - Si una variable es de tipo puntero
  - Tipo de una variable, o tipo apuntado, en caso de punteros
  - Otros atributos que el compilador requiera para permitir esta funcionalidad en el lenguaje.

  ***Chequeos semánticos***
  En esta etapa, se deberán verificar las siguientes reglas semánticas, informando errores cuando corresponda:
  - No se permite más de una variable con el mismo nombre.
  - No se permite usar del lado izquierdo de una asignación un elemento inmutable, ya sea que se trate de una variable declarada como inmutable, o lo apuntado por un puntero si lo apuntado fuese inmutable.
  - No se permite que una variable tipo puntero apunte a otra variable de tipo puntero.

- ***Closure*** (Tema 12 del TP1)

  ***Registro de información en la Tabla de Símbolos***
  En esta etapa, se deberá registrar:
  - Si un identificador corresponde a una variable o a una función
  - Tipo de una variable, o tipo devuelto por la función
  - Otros atributos que el compilador requiera para permitir esta funcionalidad en el lenguaje.

  ***Chequeos semánticos***
  En esta etapa, se deberán verificar las siguientes reglas semánticas, informando errores cuando corresponda:
  - No se permite una variable con el mismo nombre que una función.
  - No se permiten variables ni funciones con el mismo nombre que otras variables o funciones respectivamente.
  - Las variables y funciones declaradas en el programa principal, son visibles en todo el programa, a partir de su declaración.
  - Las variables y closures declarados en una función, sólo son visibles dentro de esa función, a partir de su declaración.
  - Sólo se pueden declarar funciones dentro del programa principal.
  - Sólo se permite declarar hasta un closure dentro de una función.
  - Cuando una función que retorna tipo **fun**, retorna el nombre de una función, ésta debe tener como tipo de retorno **void.**
  - Una variable declarada de tipo **fun**, sólo puede ser asignada con otra variable de tipo **fun**, o con una función que retorne el tipo **fun**.
  - Sólo una variable que fue declarada de tipo **fun** podrá ser invocada con una sentencia ejecutable como la siguiente: **ID(),**

- ***Borrowing*** (Tema 13 del TP1)

  ***Registro de información en la Tabla de Símbolos***
  En esta etapa, se deberá registrar:
  - Si un identificador corresponde a una variable, una función, o un parámetro.
  - Tipo de una variable, tipo devuelto por la función, tipo del parámetro.
  - Información del parámetro de cada función.
  - Otros atributos que el compilador requiera para permitir esta funcionalidad en el lenguaje.

  ***Chequeos semánticos***
  En esta etapa, se deberán verificar las siguientes reglas semánticas, informando errores cuando corresponda:
  - No se permite una variable con el mismo nombre que una función, ni parámetro.

- No se permiten variables ni funciones ni parámetros con el mismo nombre que otras variables, funciones o parámetros respectivemente.
- No se permite declaración de funciones dentro de otras funciones.
- Las variables y funciones declaradas en el programa principal, son visibles en todo el programa, a partir de su declaración.
- Las variables declaradas en una función, sólo son visibles dentro de esa función, a partir de su declaración.
- Sólo se podrán invocar funciones declaradas como tales en el programa.
- Una función invocada con un parámetro con permiso **readonly**, sólo podrá utilizar el parámetro como parte de expresiones, pero no podrá asignarle nada, ni pasarlo como parámetro en la invocación de otra función.
- Una función invocada con un parámetro con permiso **write**, podrá utilizar el parámetro como parte de expresiones, usarlo del lado izquierdo en una asignación, pero no podrá pasarlo como parámetro en la invocación de otra función.
- Una función, invocada con un parámetro con permiso **pass**, podrá utilizar el parámetro como parte de expresiones, podrá pasarlo como parámetro en la invocación de otra función, pero no podrá usarlo del lado izquierdo en una asignación.
- Una función, invocada con un parámetro con permisos **write; pass**, podrá utilizar el parámetro como parte de expresiones, usarlo del lado izquierdo en una asignación, y/o pasarlo como parámetro.

***Tema 14: Conversiones explícitas***

Todos los grupos que tienen asignado el tema 14, deben reconocer una conversión explícita, indicada mediante una palabra reservada para tal fin.

Por ejemplo, un grupo que tiene asignada la conversión **linteger**, debe considerar que, cuando se indique la conversión **linteger**(expresión), el compilador debe efectuar una conversión del tipo del argumento al tipo indicado por la palabra reservada. (en este caso **linteger**). Entonces, sólo se podrá operar entre ambos operandos, si se indica la conversión correspondiente. En otro caso, se debe informar error.

***Tema 15: Conversiones implícitas***

Todos los grupos que tienen asignado el tema 15, deben incorporar las conversiones en forma implícita, cuando se intente operar entre operandos de diferentes tipos. En todos los casos, la conversión a incorporar será del tipo más chico al tipo más abarcativo. Por ejemplo, el compilador incorporará una conversión del tipo **usinteger** a **linteger**, si se intenta efectuar una operación entre dichos tipos.

En el caso de asignaciones, si el lado izquierdo es del tipo menos abarcativo, se deberá informar error por incompatibilidad de tipos.

La incorporación de las conversiones implícitas se efectuará durante la generación de código intermedio para Tercetos y Árbol Sintáctico, y durante la generación de código Assembler para Polaca Inversa.

***Tema 16: Sin conversiones***

Los grupos que tienen asignado el tema 16, deben prohibir cualquier operación (expresión, asignación, comparación, etc.) entre operandos de tipos diferentes.

**Salida del compilador**

1) Código intermedio, de alguna forma que permita visualizarlo claramente. Para Tercetos y Polaca Inversa, mostrar la dirección (número) de cada elemento, de modo que sea posible hacer un seguimiento del código generado. Para Árbol Sintáctico, elegir alguna forma de presentación que permita visualizar la estructura del árbol:

***Tercetos:***

```
20  ( + , a , b )
21  ( / , [20] , 5
)
22  ( = , z , [21]
)
```

***Polaca Inversa:***

```
10 <
11 a
12 b
13 18
14 BF
15 c
16 10
17 =
18 …
```

***Árbol Sintáctico:***

```
Por ejemplo:
Raíz
     Hijo izquierdo
          Hijo izquierdo
          Hijo derecho
     Hijo derecho
          Hijo izquierdo
               …
          Hijo derecho
```

2) Si bien el código intermedio debe contener referencias a la Tabla de Símbolos, en la visualización del código se deberán mostrar los nombres de los símbolos (identificadores, constantes, cadenas de caracteres) correspondientes.
3) Los errores generados en cada una de las etapas (Análisis Léxico, Sintáctico y durante la Generación de Código Intermedio) se deberán mostrar todos juntos, indicando la descripción de cada error y la línea en la que fue detectado.
4) Contenido de la Tabla de Símbolos.

**Nota**: No se deberán mostrar las estructuras sintácticas ni los tokens que se pidieron como salida en los trabajos prácticos 1 y 2, a menos que en la devolución de la primera entrega se solicite lo contrario.