# Diseño de Compiladores I – Cursada 2024
# Trabajo Práctico Nro. 3

La entrega se hará en forma conjunta con el trabajo práctico Nro. 4 (**14/11/2024**)
**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* | *Polaca Inversa* | *Tercetos* |
|---|---|---|
| 2 | 3 | 1 |
| 6 | 11 | 4 |
| 7 | 12 | 5 |
| 8 | 15 | 10 |
| 9 | 18 | 13 |
| 14 | 21 | 16 |
| 17 | 24 | 20 |
| 19 | 25 | 22 |
| 23 | 26 | 27 |

Se deberá generar código intermedio para todas las sentencias ejecutables, incluyendo:

- Asignaciones, Selecciones, Sentencias de control asignadas al grupo, Sentencias OUTF, sentencias RET y toda otra sentencia ejecutable que admita el lenguaje**.**
- Para las funciones, se deberá generar el código intermedio correspondiente a las sentencias ejecutables del cuerpo, en forma independiente para cada función. Se podrán generar estructuras independientes para cada una, o ubicarlas todas en la misma estructura que el programa principal, con algún delimitador que permita determinar claramente el inicio y fin de cada una.

## Incorporación de información Semántica y chequeos

- ***Incorporación de información a la Tabla de Símbolos:***
  - ***Tipo:***
    - Durante el Análisis Léxico se deberá registrar, en la Tabla de Símbolos, el tipo de las constantes, y de las variables cuya letra inicial permita determinarlo (***tema 12***)
    - Para el resto de los Identificadores, se deberá registrar el tipo en la Tabla de Símbolos a partir de las sentencias declarativas.
  - ***Uso:***
    - Incorporar un atributo **Uso** en la Tabla de Símbolos, indicando el uso de cada identificador en el programa (nombre de variable, nombre de función, nombre de tipo, nombre de parámetro, etc.).
  - ***Otros Atributos:***
    - Se podrán Incorporar atributos adicionales a las entradas de la Tabla de Símbolos, de acuerdo a los temas particulares asignados

## Chequeos Semánticos:

En esta etapa, se deberán efectuar los siguientes chequeos semánticos, informando errores cuando corresponda:

- 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 (según reglas de alcance del lenguaje).
  - Funciones redeclaradas (según reglas de alcance del lenguaje).
  - Tipos no definidos (según reglas de alcance del lenguaje).
  - Tipos redefinidos (según reglas de alcance del lenguaje).
  - Toda otra situación que no cumpla con las siguientes reglas:

***Reglas de alcance:***

- Cada variable y función será visible dentro del ámbito en el que fue declarada y por los ámbitos contenidos en el ámbito de la declaración.
- Cada variable, función, tipo será visible a partir de su declaración, con la restricción indicada en el ítem anterior.
- Se permiten variables con el mismo nombre, siempre que sean declaradas en diferentes ámbitos.
- Se permiten funciones con el mismo nombre, siempre que sean declarados en diferentes ámbitos.
- No se permiten variables y funciones con el mismo nombre dentro de un mismo ámbito.
- Otras reglas que correspondan a temas particulares.

➢ Chequeo de compatibilidad de tipos:

- ***Temas 25 y 26***: Sólo se permitirán operaciones con operandos de distinto tipo, si se efectúa la conversión que corresponda (implícita o explícita según asignación al grupo).
- ***Tema 27***: 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 y la incorporación de conversiones implícitas (si corresponde) se debe efectuar en la última etapa (TP 4)

➢ Chequeos de tipo del parámetro en invocaciones a funciones

- ***Temas 25 y 26***: El tipo del parámetro real debe coincidir con el tipo del parámetro formal. En otro caso, se debe reportar un error.
- ***Tema 27***: Se permitirá que el tipo del parámetro real sea diferente al del parámetro formal, siempre que se anteponga al nombre del parámetro real el tipo del parámetro formal, para convertirlo al momento de la invocación.

➢ Otros chequeos relacionados con los temas particulares asignados.

## Asociación de cada variable con el ámbito al que pertenece:

Para identificar variables, funciones, etiquetas, etc. con el ámbito al que pertenecen, se utilizará "***name mangling***". Es decir, el nombre de una variable llevará, a continuación de su nombre original, la identificación del ámbito al que pertenece.

Ejemplos:

Ámbitos

```
main
begin
      ...
      // Declaraciones en el ámbito global
      LONGINT a;                                  // la variable a se llamará a:main
                                                  // main identifica el ámbito global
      ...
      INTEGER FUN aa(LONGINT x)                   // Ámbito aa
      begin
            INTEGER a;                            // la variable a se llamará a:main:aa
              ...
            INTEGER FUN aaa(INTEGER w)            // Ámbito aaa
            begin
                  ...
                  LONGINT x;                      // la variable x se llamará
                                                  // x:main:aa:aaa
                  ...
            end                                   // Fin Ámbito aaa
      end                                         // Fin Ámbito aa
      ...

      INTEGER FUN bb(INTEGER z)                   // Ámbito bb
      begin
```

```
            ...
             INTEGER a;                          // la variable a se llamará a:main:bb
             ...
      end                                        // Fin Ámbito bb
      ...
      INTEGER FUN cc(INTEGER y)                  // Ámbito cc
      begin
            INTEGER a;                          // la variable a se llamará a:main:cc
              ...
      end                                        // Fin Ámbito cc
      ...
end                                             // Fin Ámbito main
```

## TEMAS PARTICULARES

### *Funciones (Para todos los temas)*

***Registro de información en la Tabla de Símbolos***

En esta etapa, se deberá registrar:

- Información referida a la función:
  - El tipo devuelto
  - El tipo y nombre del parámetro.
- Otros atributos que el compilador requiera para permitir esta funcionalidad en el lenguaje.

***Código intermedio***

- Se deberá generar código para cada función declarada. El código de cada función se podrá generar en una única estructura, junto con el programa principal, o bien, utilizando una estructura independiente para el programa principal y para cada función.
- Cuando se detecte una **invocación**, se deberá generar código la misma, chequeando que el tipo del parámetro real coincida con el del parámetro formal, o incluya la conversión correspondiente (***tema 27***).
- El pasaje de parámetros será por ***copia-valor***. Por lo tanto, se deberá generar el código para el pasaje de parámetros correspondiente previo a cada invocación.

### *Temas 11 y 12 -*

- **Tema 11**: **Subtipos**

Cuando el compilador detecte la declaración de un nuevo tipo, definido como subrango de un tipo básico, se deberá registrar en la entrada correspondiente de la Tabla de Símbolos:

- Información del nuevo tipo:
  - Nombre del tipo
  - Tipo base
  - Límite inferior
  - Límite superior
- Otros atributos que el compilador requiera para permitir esta funcionalidad en el lenguaje.

Por ejemplo, para:

```
typedef enterito := integer[-10,10];
```

se debe registrar que enterito es un tipo, que el tipo base es integer, y los límites inferior y superior del rango:

Cuando se detecte una declaración de variables del nuevo tipo, por ejemplo:

```
enterito a, b, c;
```

se deberá indicar, para las entradas de las variables declaradas en la Tabla de Símbolos, que el tipo es enterito.

***Semántica asociada al nuevo tipo:***

Al detectar el nuevo tipo, el compilador debe registrar cuál será el comportamiento cuando existan operaciones entre variables del nuevo tipo con variables de tipos diferentes. Si se implementan tablas de compatibilidad de tipos, se deberá

agregar una fila y una columna para el nuevo tipo, indicando qué combinaciones se permiten y qué compinaciones están prohibidas, considerando las conversiones cuando corresponda.
Por ejemplo, el compilador debería registrar que un subrango de enteros puede operar del mismo modo que su tipo base.

```
a := 1;          // si a fue declarado del tipo enterito definido antes,
                 // esta operación es válida.
x := a + 1.5;    // a de tipo enterito, y x de tipo single o double
```

El compilador deberá:
- (tema 25) Generar la conversión de enterito a single / doublé según corresponda
- (tema 26) Detectar incompatibilidad de tipos, a menos que se incluya la conversión explícita correspondiente:
  - X := **tos**(a) + 1.5; si x y 1.5 son de tipo single
  - X := **tod**(a) + 1.5; si x y 1.5 son de tipo double
- (tema 27) Detectar incompatibilidad de tipos

- **Tema 12**: **Tipos Embebidos**

  Durante el Análisis Léxico se deberá registrar, en la Tabla de Símbolos, el tipo para los identificadores cuya letra inicial permita determinarlo, como se indicó en el Trabajo Práctico 1. Estos identificadores sólo podrán ser utilizados como nombres de variables.
  Por lo tanto, se deberán efectuar los siguientes chequeos:
  - Si alguno de estos identificadores aparece en una sentencia de declaración de tipo de variables, se debe indicar que existe una redeclaración de variables.
  - Si alguno de estos identificadores aparece como nombre de función, o de parámetro formal, se deberá informar el error de redeclaración.
  - Si alguno de estos identificadores aparece como etiqueta, se deberá informar que se trata de un error.

***Temas 13 a 16: Sentencias de Control***

- **Tema 13**: ***WHILE***

  ```
  WHILE ( <condicion> ) <bloque_de_sentencias_ejecutables> ;
  ```

  El bloque de sentencias ejecutables se ejecutará mientras la condición sea verdadera.

- **Tema 14**: ***REPEAT UNTIL***

  ```
  REPEAT <bloque_de_sentencias_ejecutables> UNTIL ( <condicion> );
  ```

  El bloque de sentencias ejecutables se ejecutará hasta que la condición sea verdadera. La evaluación de la condición se efectuará luego de la ejecución del bloque.

- **Tema 15**: ***REPEAT WHILE***

  ```
  REPEAT <bloque_de_sentencias_ejecutables> WHILE ( <condicion> );
  ```

  El bloque de sentencias ejecutables se ejecutará mientras la condición sea verdadera. La evaluación de la condición se efectuará luego de la ejecución del bloque.

- **Tema 16**: ***FOR***

  **FOR** (i **:=** m ; <condición> ; **up**/**down** n ) < bloque_de_sentencias_ejecutables > ;
  i debe ser una variable entera.
  m y n serán constantes enteras

  La variable de control i, comenzará la iteración con el valor m.
  La condición controlará el final de la ejecución del bloque.
  En cada ciclo, el incremento de la variable de control será igual a n.

  Ejemplos:

  ```
  FOR (i := 1;  i<10 ; up 1)…      // La ejecución del bloque se iniciará con un valor inicial 1 para la
                                   // variable i, y se ejecutará mientras su valor sea menor que 10.
                                   // En cada ciclo se incrementará el valor de i en 1.
  FOR (j := 10;  j > 0 ; down 2)…  // La ejecución del bloque se iniciará con un valor inicial 10 para la
                                   // variable j, y se ejecutará mientras su valor sea mayor que 0.
  ```

// En cada ciclo se decrementará el valor de j en 2.

Se deberán efectuar los siguientes chequeos de tipo:
- i debe ser una variable de tipo entero (1-2-3-4).
- m y n deben ser constantes de tipo entero (1-2-3-4).

***Nota***:
- Para Tercetos y Árbol Sintáctico, los chequeos de tipos se deben efectuar durante la generación de código intermedio (TP03)
- Para Polaca Inversa, los chequeos de tipos se deben efectuar en la última etapa (TP04)

***Temas 17 al 19***

- **Tema 17**: ***Asignaciones múltiples***

  Se deberá generar código para asignar cada elemento de la derecha al correspondiente de la izquierda, en el orden en que se presenten.

  Por ejemplo, para:

  ```
  a,b,c := 1+d, e, f+5;        // Se deberán generar 3 asignaciones: a:=1+d, b:=e y c:=f+5
  ```

  Se deberá chequear que el número de elementos a la izquierda sea igual al número de elementos de la derecha. En caso contrario se informará un error.

  Se deberá chequear la compatibilidad de tipos en cada asignación individual, en forma independiente.

- **Tema 18**: ***Asignaciones múltiples***

  Se deberá generar código para asignar cada elemento de la derecha al correspondiente de la izquierda, en el orden en que se presenten.

  Por ejemplo, para:

  ```
  a,b,c := 1+d, e, f+5;    // Se deberán generar 3 asignaciones: a:=1+d, b:=e y c:=f+5
  ```

  En caso que del lado izquierdo haya menos elementos que del lado derecho, se descartarán las expresiones sobrantes del lado derecho, y se informará la situación con un Warning.

  En caso que del lado izquierdo haya más elementos que del lado derecho, se asignará a los elementos sobrantes el valor 0, y se informará la situación con un Warning.

  Por ejemplo, para:

  ```
  x,y := 1, f1(3), w;          // Se descartará el último elemento del lado derecho
  i, j , k := a*3 , b;         // Se deberán generar 3 asignaciones: i:=a*3, j:=b y k:=0
  ```

  // En ambos casos, se emitirá un Warning:

- **Tema 19**: ***Pattern Matching***

  Se deberá generar código para comparar cada elemento de la izquierda con el correspondiente de la derecha, en el orden en que se presenten. Las comparaciones se combinarán, luego, con el operador lógico AND.

  Por ejemplo, para:

  ```
  if ((a,c,...) >= (b,2.3,...)) then …
  ```

  // El código a generar deberá evaluar la siguiente condición: a >= b AND c >= 2.3 AND …

  Se deberá chequear que el número de elementos a la izquierda sea igual al número de elementos de la derecha. En caso contrario se informará un error.

  Se deberá chequear la compatibilidad de tipos en cada comparación individual, en forma independiente.

***Temas 20 al 22***

- **Tema 20**: ***struct***

  Cuando el compilador detecte la declaración de un nuevo tipo **struct**, se deberá registrar en la Tabla de Símbolos, la información del nuevo tipo:

  – Nombre del tipo

- Número de componentes
- Información de cada componente:
  - Nombre
  - Tipo
- Otros atributos que el compilador requiera para permitir esta funcionalidad en el lenguaje.
- Por ejemplo, para:

```
typedef Struct <integer,single,integer> {
    a,
    b,
    c,
} ts1;
```

se debe registrar que ts1 es un tipo, que tiene 3 componentes, y los nombres de sus componentes, indicando de qué tipo es cada uno.

Cuando se detecte una declaración de variables del nuevo tipo, por ejemplo:

```
ts1 s1,s2,s3;    // declaración de las variables s1 a s3 con el tipo ts1
```

se deberá indicar, para las entradas de las variables declaradas en la Tabla de Símbolos, que el tipo es ts1.

Las variables declaradas de un tipo **struct** sólo podrán utilizarse en asignaciones donde el lado izquierdo y derecho sean variables de este tipo. No debe permitirse su uso en otro tipo de sentencias / expresiones.

Los componentes de las variables declaradas con el nuevo tipo podrán ser utilizados en cualquier lugar donde pueda utilizarse una variable, con las mismas reglas y chequeos que se consideran para variables.

- **Tema 21**: ***pair***

Cuando el compilador detecte la declaración de un nuevo tipo **pair**, se deberá registrar en la Tabla de Símbolos, la información del nuevo tipo:

- Nombre del tipo
- Tipo de los componentes
- Otros atributos que el compilador requiera para permitir esta funcionalidad en el lenguaje.
- Por ejemplo, para:
- `typedef pair <integer> pint;` //declara un tipo par de enteros

se debe registrar que pint es un tipo, cuyos componentes son de tipo integer.

Cuando se detecte una declaración de variables del nuevo tipo, por ejemplo:

```
pint p1,p2,p3;    // declaración de las variables p1 a p3 con el tipo pint
```

se deberá indicar, para las entradas de las variables declaradas en la Tabla de Símbolos, que el tipo es pint.

Las variables declaradas de un tipo **pair** sólo podrán utilizarse en asignaciones donde el lado izquierdo y derecho sean variables de este tipo. No debe permitirse su uso en otro tipo de sentencias / expresiones.

Los componentes de las variables declaradas con el nuevo tipo podrán ser utilizados en cualquier lugar donde pueda utilizarse una variable, con las mismas reglas y chequeos que se consideran para variables.

- **Tema 22**: ***triple***

Cuando el compilador detecte la declaración de un nuevo tipo **triple**, se deberá registrar en la Tabla de Símbolos, la información del nuevo tipo:

- Nombre del tipo
- Tipo de los componentes
- Otros atributos que el compilador requiera para permitir esta funcionalidad en el lenguaje.
- Por ejemplo, para:
- `typedef triple <integer> tint;` //declara un tipo par de enteros

se debe registrar que tint es un tipo, cuyos componentes son de tipo integer.

Cuando se detecte una declaración de variables del nuevo tipo, por ejemplo:

```
tint t1,t2,t3;    // declaración de las variables t1 a t3 con el tipo pint
```

se deberá indicar, para las entradas de las variables declaradas en la Tabla de Símbolos, que el tipo es tint.

Las variables declaradas de un tipo **triple** sólo podrán utilizarse en asignaciones donde el lado izquierdo y derecho sean variables de este tipo. No debe permitirse su uso en otro tipo de sentencias / expresiones.
Los componentes de las variables declaradas con el nuevo tipo podrán ser utilizados en cualquier lugar donde pueda utilizarse una variable, con las mismas reglas y chequeos que se consideran para variables.

## *Temas 23 y 24: Flujo de Control*

- **Tema 23**: ***goto***

  Cuando el compilador detecte un goto a una etiqueta, deberá generar en el código intermedio una bifurcación incondicional a la posición donde se encuentre la etiqueta correspondiente.

  Al detectar la etiqueta, se debe incorporar la etiqueta al código intermedio, ya sea como un nodo del árbol, un terceto tipo etiqueta, o un elemento de la Polaca Inversa.

  Se debe chequear la existencia de la etiqueta a la que se pretende bifurcar.

### **Tema 24**: Iterador con condición

```
FOR (i := m ; <condicion1> ; up/down n ; <condicion2>) < bloque_de_sentencias_ejecutables >
;
```

El control de la iteración seguirá las reglas de la sentencia for indicadas para el tema 16. Pero, en cada ciclo, para decidir si el cuerpo se ejecuta o no, se deberá evaluar la segunda condición, habilitando la ejecución sólo en los casos en que su evaluación resulte verdadera. El cumplimiento o no de <condicion2> no afectará el control deteminado por los otros elementos del encabezado.

Ejemplo:

```
a := 2;
FOR (j := 1;  j <= 5 ; up 1 ; (a < 5))
Begin
   a := a + j;
   Outf ({a es menor que 5});
End;

// El cuerpo sólo se ejecutará para los 2 primeros ciclos de la iteración.
```

## *Temas 25 a 27: Conversiones*

### **Tema 25**: ***Conversiones Explícitas***

- El compilador debe reconocer el uso de conversiones explícitas en el código fuente, que serán indicadas mediante la palabra reservada asignada en el TP02.

  **TOS**(<expresión>)  // para grupos que tienen asignado el tema 5

  **TOD**(<expresión>)  // para grupos que tienen asignado el tema 6

- Se debe considerar que cuando se indique la conversión **TOD**(expresión) o **TOS**(expresión), el compilador deberá generar código para efectuar una conversión del tipo del argumento al tipo indicado por la palabra reservada. (**DOUBLE** o **SINGLE** respectivamente). Dado que los otros tipos asignados al grupo son enteros (1-2-3-4), el argumento de una conversión debe ser del tipo entero asignado.
- Sólo se podrán efectuar operaciones entre dos operandos de distinto tipo (uno entero y otro flotante), si se convierte el operando de tipo entero (1-2-3-4) al tipo de punto flotante mediante la conversión explícita que corresponda. En otro caso, se debe informar error.
- En el caso de asignaciones, si el lado izquierdo es tipo entero (1-2-3-4), y la expresión del lado derecho es de tipo diferente, se deberá informar error por incompatibilidad de tipos.

### **Tema 26**: ***Conversiones Implícitas***

- El compilador debe incorporar las conversiones en forma implícita, cuando se intente operar entre operandos de diferentes tipos (uno entero y otro flotante). En todos los casos, la conversión a incorporar será del tipo entero (1-2-3-4)

al tipo de punto flotante asignado al grupo. Por ejemplo, el compilador incorporará una conversión del tipo **INTEGER** a **SINGLE**, si se intenta efectuar una operación entre dos operandos de dichos tipos.

- En el caso de asignaciones, si el lado izquierdo es de uno de tipo entero (1-2-3-4), y la expresión del lado derecho es de tipo diferente, se deberá informar error por incompatibilidad de tipos. En cambio, si el lado izquierdo es del tipo de punto flotante asignado al grupo, y el lado derecho es de tipo entero (1-2-3-4), el compilador deberá incorporar la conversión que corresponda al lado derecho de la asignación.

**Tema 27**: ***Sin Conversiones***

El compilador debe prohibir cualquier operación (expresión, asignación, comparación, etc.) entre operandos de tipos diferentes, informando en cada caso, cuál es la combinación de tipos que está provocando la incompatibilidad.
En la invocación de funciones, si se aplica la conversión permitida para este tema para el parámetro real, el compilador deberá generar, previo a la copia del parámetro formal sobre el real, la conversión indicada.

## 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.
Ejemplos:

***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 :=
…
```

***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.