Diseño de Compiladores I – Cursada 2017
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 3, 4, 9, 11, 12, 13,19, 21 y 23)
  - ***Tercetos*** (grupos 1, 5, 6, 7, 10, 14, 15, 18 y 22)
  - ***Polaca Inversa*** (grupos 2, 8, 16, 17 y 20)

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

Temas particulares

Sentencias de Control

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

  El bloque se ejecutará mientras la condición sea verdadera

- ***DO UNTIL*** (tema 8 en TP1)
  DO <bloque_de_sentencias> **UNTIL (** <condicion> **) .**

  El bloque se ejecutará hasta que la condición sea verdadera

- ***FROM TO BY*** (tema 9 en TP1)
  FROM i **:=** n **TO** m **BY** j < bloque_de_sentencias > **.**

  El bloque se ejecutará hasta que se cumpla que el valor de i alcance el valor m. En cada ciclo, se actualizará la variable de control según corresponda.

  Chequeo de tipos:
  i debe ser una variable de tipo (tema 1, 2, 3 o 4).
  m, n y j pueden ser variables, constantes o expresiones aritméticas de tipo (tema 1, 2, 3 o 4).
  (1) Para los grupos que tengan asignados 2 tipos enteros, considerar el "más chico" (por ej. INT, para grupos con INT y LONG)
  (2) Para los grupos que tengan asignados 1 tipo entero, y uno de punto flotante, considerar el tipo entero
  Nota: Las restricciones de tipo serán chequeadas en esta etapa, o en la siguiente, según la asignación de representación intermedia correspondiente.

- ***SWITCH CASE*** (tema 10 en TP1) Incorporar las llaves (**{** y **}**) como caracteres válidos.
  SWITCH (variable){
  CASE valor1**:** <bloque> **.**
  CASE valor2**:** <bloque> **.**
  ...
  CASE valorN**:** <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

- **Shadowing (Temas 11 y 12 del TP1)**
  El programa podrá contener más de una variable con el mismo nombre. Cada una será visible en el programa, desde su declaración, hasta la declaración de otra variable con el mismo nombre. El compilador debe distinguir cada una de esas variables, y efectuar los chequeos necesarios para identificar cuál de las variables es utilizada en una sentencia determinada.
- **Inferencia de tipos (Tema 11 del TP1)**
  Las asignaciones de la forma:
  LET <asignación> **.**
  Implicarán inferencia de tipos. Esto significa que la variable del lado izquierdo recibirá el tipo de la expresión del lado derecho. Pero, si la variable correspondiente ya tuviera tipo, esta declaración implicará la creación de una nueva variable, que será visible de ahí en adelante.

- **Funciones MOVE (Tema 11 del TP1)**
  Entre las sentencias declarativas del programa, se podrá incluir la declaración de funciones, con el siguiente formato:
  <tipo> **FUNCTION** <nombre_de_funcion> o <tipo> **MOVE FUNCTION** <nombre_de_funcion>
  {
  <Sentencias declarativas y ejecutables de la función>
  RETURN(<expresión>) **.**
  }
  - Una función se invoca utilizando su nombre seguido de paréntesis: <nombre_de_función>**(),** y una invocación puede aparecer en cualquier lugar donde pueda aparecer una variable o constante
  - La declaración de una función usando la palabra reservada **MOVE**, implica lo siguiente:
    La función sólo podrá usar las variables declaradas localmente. Es decir, no podrá utilizar variables del ámbito global.

Incorporación de información a la Tabla de Símbolos:

- Incorporar la información que corresponda a la Tabla de símbolos, a partir de las sentencias declarativas.
  - ***Variables***
    Se deberá registrar el tipo, y si correspondiera, modificar el nombre.
  - ***Constantes***
    Se deberá registrar el tipo (esto debe ser registrado durante el Analizador Léxico).
  - ***(Tema 13) Para todos los identificadores***
    Incorporar un atributo Uso en la Tabla de Símbolos, indicando el uso del identificador en el programa (variable o nombre de función).
    Nota: Esta consigna corresponde a aquellos grupos cuyos lenguajes permitan que un identificador pueda tener diferentes usos.
  - ***(Tema 13) Funciones***
    Se deberá registrar el tipo que retornan.
- Asociar cada variable con el ámbito al que pertenece (tema 13 del TP1):
  - Dentro del ámbito definido por una función, las variables visibles serán aquellas declaradas localmente, o en los ámbitos ubicados más arriba en el árbol de anidamiento.
    Para identificar las variables 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:

```
... // sentencias declarativas y/o ejecutables del programa principal

// Declaración de una variable en el ámbito global:
a: FLOAT;  // la variable a se llamará, por ejemplo a@main

... // sentencias declarativas y/o ejecutables del programa principal

// Declaración de una función en el ámbito global:
INT FUNCTION aa  // La función aa se llamará aa@main
// Ámbito aa
{
  ... // sentencias declarativas y/o ejecutables de la función aa

  a: INT;  // la variable a se llamará a@main@aa

  ... // sentencias declarativas y/o ejecutables de la función aa
}

... // sentencias declarativas y/o ejecutables del programa principal

// Declaración de otra función en el ámbito global:
FLOAT FUNCTION bb{)// La función bb se llamará bb@main
// Ámbito bb
{
  ... // sentencias declarativas y/o ejecutables de la función bb

  x: FLOAT;   // la variable x se llamará x@main@bb

  ... // sentencias declarativas y/o ejecutables de la función bb
  }

... // sentencias declarativas y/o ejecutables del programa principal

// Declaración de una función MOVE en el ámbito global:
INT MOVE FUNCTION cc{)// La función cc se llamará cc@main
// Ámbito cc
{
  ... // sentencias declarativas y/o ejecutables de la función bb
```

```
  y: FLOAT;   // la variable y se llamará y@main@cc

  ... // sentencias declarativas y/o ejecutables de la función cc
  }

... // sentencias declarativas y/o ejecutables del programa principal


Durante la generación de código, si en el ámbito aa se usa la variable a en una
expresión, se referirá a la variable a@main@aa. En cambio, si se usa dentro del ámbito
bb, se referirá a a@main, ya que no existe declaración para a en ese ámbito. Pero, si
la variable a se usa en una expresión dentro del ámbito cc, el compilador deberá
indicar error por variable no declarada, ya que desde el ámbito cc no se pueden ver las
variables globales.
```

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).
  - Tipo incorrecto en límites de iteración **FROM TO** (cuando corresponda).

Chequeos semánticos para Funciones (Tema 13):

- Nombres de función redeclarados.
- Nombres de función como operandos de expresiones, sin los paréntesis que indican la invocación de una función.
- Función no declarada, cuando se invoque a una función no declarada previamente.

Chequeo de compatibilidad de tipos (para todos los temas):

Sólo se puede operar (en asignaciones, expresiones, comparaciones, etc.), con operandos del mismo tipo. Sólo se deben permitir operaciones con operandos de distinto tipo, si se efectúa la conversión que corresponda según asignacion al grupo (temas 14, 15 y 16).

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

Tema 14: Una conversión explícita

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 I_F, debe considerar que, cuando se indique esa conversión, I_F(expresión), el compilador deberá efectuar una conversión del tipo del argumento (en este caso INT) al tio indicado en la segunda parte de la palabra reservada. (en este caso FLOAT). Entonces, sólo se podrá operar entre un operando INT y otro FLOAT, si se indica la conversión correspondiente. En otro caso, se debe informar error.

Tema 15: Una conversión explícita e Incorporación de conversiones implícitas

Todos los grupos que tienen asignado el tema 15, deben reconocer una conversión explícita, indicada mediante una palabra reservada para tal fin.

Por ejemplo, un grupo que tiene asignada la conversión F_UI, debe considerar que, cuando se indique esa conversión, F_UI(expresión), el compilador deberá efectuar una conversión del tipo del argumento (en este caso FLOAT) al tipo indicado en la segunda parte de la palabra reservada. (en este caso UINT)

Pero, en caso que, en una expresión, comparación, asignación, etc., se combinen operandos de los dos tipos, pero no se incluya la conversión explícita, el compilador deberá generar la conversión implícita opuesta. Es decir, para el ejemplo, deberá convertir el operando de tipo UINT a FLOAT.

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 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 (variables con su nombre modificado, 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.