Diseño de Compiladores I – Cursada 2022
Trabajo Práctico Nro. 3

La entrega se hará en forma conjunta con el trabajo práctico Nro. 4 (17/11/2022)
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*
1	4	3
2	5	6
7	9	12
8	15	14
10	18	17
11	19	20
13	25	22
16		23
21		24

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

- Asignaciones, Selecciones, Sentencias de control asignadas al grupo, invocaciones a funciones y Sentencias **out.**

Incorporación de información Semántica y chequeos

- ***Incorporación de 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:***
    - Incorporar un atributo Uso en la Tabla de Símbolos, indicando el uso del identificador en el programa (variable, nombre de función, 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).
  - Toda otra situación que no cumpla con las siguientes reglas:

  Reglas de alcance:
  - Cada variable o función será visible dentro del ámbito en el que fue declarada/o y por los ámbitos enidoss contenidos en el ámbito de la declaración.
  - Cada variable, o función 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.
  - Se permiten etiquetas con el mismo nombre, siempre que sean declaradas en diferentes ámbitos.
  - No se permiten variables, funciones y etiquetas con el mismo nombre dentro de un mismo ámbito.
- Chequeo de compatibilidad de tipos:
  - Temas 21 y 22: Sólo se permitirán operaciones con operandos de distinto tipo, si se efectúa la conversión que corresponda (implícita o explícita según asignación al grupo).

- Tema 23: 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)

➢ Chequeos de tipo y número de parámetros en invocaciones a funciones

- El número de los parámetros reales en una invocación a una función, deben coincidir con el número de los parámetros declarados para la función.
- Para el tipo de los parámetros, se aplicará el chequeo de compatibilidad, indicado en el ítem anterior.

➢ Otros chequeos relacionados con los temas particulares asignados.

Asociación de cada variable con el ámbito al que pertenece:

Para identificar las variables y funciones 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 con Nombre

```
pp // nombre del programa
{
      ...
      // Declaraciones en el ámbito global
      i32 a;    // la variable a se llamará a.pp
                // pp identifica el ámbito global
      ...
      fun aa(i32 x) : i16        // Ámbito aa
      {
              i16 a;  // la variable a se llamará a.pp.aa
              ...
              fun aaa(i16 w) : i32                    // Ámbito aaa
            {
                  ...
                  i32 x;   // la variable x se llamará
                           // x.pp.aa.aaa
                  ...
            }                                          // Fin Ámbito aaa
      }                                                // Fin Ámbito aa
      ...
      fun bb(i16 z): i32                               // Ámbito bb
      {
             ...
              i16 a;  // la variable a se llamará a.pp.bb
              ...
      }                                                // Fin Ámbito bb
      ...
      fun cc(i16 y) : i16                              // Ámbito cc
      {
              i16 a;  // la variable a se llamará a.pp.cc
              ...
      }                                                // Fin Ámbito cc
      ...
}                                                      // Fin Ámbito pp
```

TEMAS PARTICULARES

Tema 9: discard

Ante una sentencia:

discard ID(<lista_de_parametros_reales>);

La invocación se efectuará normalmente, pero el valor de retorno de la función será descartado.

Ejemplo:

```
discard f1(2);      // el valor de retorno de la función f1 será descartado.
```

Tema 10: Cláusulas de compilación

Ante una sentencia de declaración de constantes:

const <lista_de_constantes>;

el compilador deberá registrar en la Tabla de Símbolos, la información de las constantes declaradas, registrando que se trata de constantes, el valor y el tipo que corresponda.

Ante una sentencia:

when (<condición>) **then** <bloque_de_sentencias>

Las sentencias del bloque sólo deberán ser consideradas en la compilación, cuando la condición pueda evaluarse como verdadera en tiempo de compilación.

Tema 11: While con continue

while (<condicion>) <bloque_de_sentencias_ejecutables> **;**

El bloque se ejecutará mientras la condición sea verdadera.
Si se detecta una sentencia **break**, se debe salir de la iteración en la que se encuentra.
Si se detecta una sentencia **continue**, se debe forzar el salto al próximo ciclo de la iteración.

Tema 12: While con expresión

while (<condicion>) **:** (<asignación>) <bloque_de_sentencias_ejecutables> **;**

El bloque se ejecutará mientras la condición sea verdadera.
Si se detecta una sentencia **break**, se debe salir de la iteración en la que se encuentra.
La asignación del encabezado, en caso de estar presente, se deberá ejecutar al final de cada ciclo.

Tema 13: Do until con continue

do <bloque_de_sentencias_ejecutables> **until** (<condicion>);

El bloque se ejecutará hasta que la condición sea verdadera
Si se detecta una sentencia **break**, se debe salir de la iteración en la que se encuentra.
Si se detecta una sentencia **continue**, se debe forzar el salto al próximo ciclo de la iteración.

Tema 14: Do until con expresión

do <bloque_de_sentencias_ejecutables> **until** (<condicion>) **:** (<asignación>) **;**

El bloque se ejecutará hasta que la condición sea verdadera
Si se detecta una sentencia **break**, se debe salir de la iteración en la que se encuentra.
La asignación del final de la sentencia, en caso de estar presente, se deberá ejecutar al final de cada ciclo.

Tema 15: for con continue

for (i =: n; <condición_for>; +/- j) < bloque_de_sentencias_ejecutables > **;**

- El bloque se ejecutará hasta que se cumpla la condición.
- En cada ciclo, el incremento (+) o decremento (-) 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 (1-2-3-4-5-6).
  - n y j deben ser constantes de tipo entero (1-2-3-4-5-6).
  - <condición_for> debe ser una comparación de i con m. Por ejemplo: i < m
    Donde m puede ser una variable, constante o expresión aritmética de tipo entero (1-2-3-4-5-6).

  Nota:
  - Para Tercetos y Árbol Sintáctico, los chequeos de tipos se debe efectuar durante la generación de código intermedio
  - Para Polaca Inversa, los chequeos de tipos se deben efectuar en la última etapa (TP 4)
- Si se detecta una sentencia **break**, se debe salir de la iteración en la que se encuentra.
- Si se detecta una sentencia **continue**, se debe forzar el salto al próximo ciclo de la iteración, efectuando la actualización de la variable de control previamente a dicho ciclo.

Tema 16: for

for (i =: n; <condición_for>; +/- j) < bloque_de_sentencias_ejecutables > **;**

- El bloque se ejecutará hasta que se cumpla la condición.
- En cada ciclo, el incremento (+/ ausencia de signo) o decremento (-) 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 (1-2-3-4-5-6).
  - n y j deben ser constantes de tipo entero (1-2-3-4-5-6).
  - <condición_for> debe ser una comparación de i con m. Por ejemplo: i < m
    Donde m puede ser una variable, constante o expresión aritmética de tipo entero (1-2-3-4-5-6).

  Nota:
  - Para Tercetos y Árbol Sintáctico, los chequeos de tipos se debe efectuar durante la generación de código intermedio
  - Para Polaca Inversa, los chequeos de tipos se deben efectuar en la última etapa (TP 4)
- Si se detecta una sentencia **break**, se debe salir de la iteración en la que se encuentra.

Tema 17: Break con etiquetado

Al detectar una etiqueta precediendo a la sentencia de control asignada (temas 11 al 16), y un break con una etiqueta, se deberá proceder de la siguiente forma:

- Se debe salir de la iteración etiquetada con la etiqueta indicada en el break.
- Se deberá chequear que exista tal etiqueta, y que la sentencia break se encuentre en la sentencia de control que lleva esa etiqueta, o en una sentencia anidada en la misma.

Por ejemplo:

```
outer: while (i < 10) {
        while (j < 5) {
            break :outer;  //El break provocará salir de la iteración etiquetada con outer
        };
    };
```

Tema 18: Continue con etiquetado

Al detectar una etiqueta precediendo a la sentencia de control asignada (temas 11 al 16), y un continue con una etiqueta, se deberá proceder de la siguiente forma:

- Se debe saltar al próximo ciclo de la iteración etiquetada con la etiqueta indicada en el continue.
- Se deberá chequear que exista tal etiqueta, y que la sentencia continue se encuentre en la sentencia de control que lleva esa etiqueta, o en una sentencia anidada en la misma.

Por ejemplo:

```
outer: while (i < 10) : (i =: i + 1) {
  while (j < 5) : (j := j + 1) {
      continue :outer; // provocará el salto al próx. ciclo del while etiquetado con outer.
  };
};
```

Tema 19: Diferimiento

Al detectarse la presencia de la palabra reservada **defer** al comienzo de una sentencia o bloque de selección o iteración, la ejecución de dicha sentencia o bloque deberá diferirse al final del ámbito donde se encuentra la sentencia diferida.

Tema 20: Sentencia de control como expresión

Incorporar, a la sentencia de control asignada en los temas 11 a 16, la posibilidad de devolver un valor utilizando la sentencia break, o un valor por defecto en la finalización normal de la sentencia.

Al detectarse la utilización de una sentencia de control (temas 11 a 16) como expresión, se deberá proceder de la siguiente manera:

- Si se produce la ejecución del break, el valor retornado por la sentencia de control, que se utilizará como expresión, será el indicado en la sentencia break.
- Si no se produce la ejecución del break, el valor retornado por la sentencia de control, que se utilizará como expresión, será el indicado en el else al final de la sentencia.

Ejemplos:

Ejemplo para while: (11)

```
a =: while (i < end){
     if (i = number) {
        break 1;
        }
     } else 3;
```

Ejemplo para do until: (13)

```
a =: do {
     if (i = number) {
        break 1;
        }
     }until (i> end) else 3 ;
```

Ejemplo para for: (15/16)

```
a =: for (i =: 0 ;i < end ; +1){
     if (i = number) {
        break 1;
        }
     } else 3;
```

Ejemplo para while: (12)

```
a =: while (i < end) : (i=:i+1)
{
     if (i = number) {
        break 1;
        }
     } else 3;
```

Ejemplo para do until: (14)

```
a =: do {
     if (i = number) {
        break 1;
        }
    } until (i> end) : (i=:i+1) else 3 ;
```

Para todos los ejemplos, si se ejecuta el break, el valor asignado a la variable a será 1. En caso contrario, será 3.

Dado que el valor retornado participa en una expresión aritmética, se deberá chequear que el tipo del valor devuelto sea compatible con el resto de la expresión en la que participa.

Tema 21: Conversiones Explícitas

- Todos los grupos que tienen asignado el tema 21, deben reconocer una conversión explícita, indicada mediante la palabra reservada para tal fin.
- Por ejemplo, un grupo que tiene asignada la conversión **tof64**, debe considerar que cuando se indique la conversión **tof64**(expresión), el compilador debe efectuar una conversión del tipo del argumento al tipo indicado por la palabra reservada. (en este caso **f64**). Dado que el otro tipo asignado al grupo es entero (1-2-3-4-5-6), el argumento de una conversión, debe ser de dicho tipo. Entonces, sólo se podrán efectuar operaciones entre dos operandos de distinto tipo, si se convierte el operando de tipo entero (1-2-3-4-5-6) al tipo de punto flotante asignado, mediante la conversión explícita que corresponda. En otro caso, se debe informar error.
- En el caso de asignaciones, si el lado izquierdo es del tipo entero (1-2-3-4-5-6), y la expresión del lado derecho es de tipo diferente, se deberá informar error por incompatibilidad de tipos.

Tema 22: Conversiones Implícitas

- Todos los grupos que tienen asignado el tema 22, 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 entero (1-2-3-4-5-6) al otro tipo asignado al grupo. Por ejemplo, el compilador incorporará una conversión del tipo **i16** a **f32**, si se intenta efectuar una operación entre dos operandos de dichos tipos.
- En el caso de asignaciones, si el lado izquierdo es del tipo entero (1-2-3-4-5-6), 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 (1-2-3-4-5-6), el compilador deberá incorporar la conversión que corresponda al lado derecho de la asignación.
- ***Nota***: 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 23: Sin conversiones

Los grupos que tienen asignado el tema 23, 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.