Instituto Tecnológico de Costa Rica ***Sede Regional San Carlos***	Segunda Tarea Programada Compiladores e Intérpretes
Prof. Oscar Mario Víquez Acuña	***Análisis Contextual***

Descripción:

La fase de análisis sintáctico ha sido concluida y ahora el problema a enfrentar es la solución del análisis contextual. Nuestro enemigo deja de ser la forma en que se escribe el programa y sus diferentes construcciones por parte del usuario (sintaxis) para convertirse en la validez con la que se usan los identificadores en el programa, tanto lo relacionado con la existencia de declaraciones previas, como a la igualdad de tipos en las expresiones.

Durante esta fase se debe construir una serie de procedimientos (usualmente conocidos como "visits" utilizando la estrategia de "Visitors") que se encarguen de verificar los tipos y alcances de los identificadores en el programa. Esta inspección debe ser auxiliada por una ***tabla de identificadores*** declarados, llevando para cada identificador mínimo su spelling (se pueden insertar en la tabla funciones y parámetros), su nivel de declaración y un apuntador a su declaración más reciente en el AST. Dicha tabla será, además del AST, una de las estructuras principales para el desarrollo de esta fase.

Es importante considerar, basado en la estructura del lenguaje que se implementa, la creación de otro tipo de tablas para el manejo de las clases básicas que permite el lenguaje. Esto por el motivo de que acceder a los campos de una clase tiene ligeras diferencias con respecto a acceder a una variable simple o a parámetros de un método.

Previamente a la implementación del analizador contextual se deben realizar los cambios necesarios a las estructuras creadas (AST) en la fase anterior. Básicamente agregar ciertos campos a nodos claves para el chequeo de tipos y alguna otra información más que sea necesaria según el criterio del programador (como fue visto en clase).

Tabla de símbolos o tabla de identificadores

Es necesario recordar que estamos ante la creación de un lenguaje con estructura de bloque planos, por lo que se debe realizar la implementación de esta tabla considerando los diferentes niveles de bloque que se pueden presentar. Para esto ver la siguiente tabla:

Nivel	***Tipo de identificador***

-1	Ambiente Estándar (en este caso los métodos predeclarados y cualquier otra variable de entorno que el programador considere necesaria).
0	Identificadores globales del "Goal" (clases). Variables y métodos.
1	Identificador local a un procedimiento/ función así como los parámetros.

La escogencia e implementación de las estructuras de datos es crucial en el desempeño de una herramienta de compilación. En este caso para la creación de la tabla es preciso escoger **la mejor** alternativa de estructura. Como recomendación es mejor utilizar herramientas implementadas en el lenguaje de programación con el objetivo de aprovechar la existencia y eficiencia de las mismas. Se debe documentar y justificar la escogencia de dicha estructura.

Volviendo a la creación de los "visits", estos tienen como objetivo seguir con el modelo de descenso recursivo visto en la primera fase, de manera que crearemos un "visitor" para visitar todos los nodos creados en el AST. Se realizará la primer llamada al "visitProgram()", dentro de su código se realizará eventualmente una llamada a los "visits" para cada uno de sus hijos instanciados según sea el caso, y esto se repetirá hasta que se hayan visitado todos las instancias del árbol creado. Antes de la llamada al primer "visit" se debe inicializar la tabla de símbolos.

Existen "visits" en el analizador contextual claves para el funcionamiento del mismo. Específicamente aquellos relacionados con las declaraciones de variables (en donde se realizarán las incorporaciones y borrados de identificadores en la tabla de símbolos) y aquellos que tengan involucradas el uso de expresiones (ya que son estas las que deben concordar con algún patrón de tipos o alguna declaración previa de valores). En este punto, en cada uno de dichos "visits" se realizará el adornado del árbol como se mencionó en clase agregando los punteros a las declaraciones y los tipos de las expresiones. VER Cap. 5 Watt (PLP in Java).

El otro punto clave en la implementación está relacionado entonces con el chequeo de tipos en donde básicamente se debe realizar la comprobación de que, para cada identificador utilizado, el valor asignado corresponda al tipo de su declaración (incluye el uso de funciones también). Además se deben chequear algunas otras expresiones condicionales utilizadas en estructuras de control condicional ya que en estas se utilizan expresiones que deben ser de tipo booleano para calcular el control de flujo.

Hay cinco tipos de operadores de aritmética binaria +, -, *, < y &&. Los operadores de suma y resta tienen precedencia más baja que los de multiplicación.

Hay 1 operador relacional binarios: <. Los tipos se manejan como se expresó anteriormente. Estos operadores deben evaluarse como tipos enteros (o sea que el tipo de adorno debe ser "int" siendo luego en la generación de código necesario definir el valor para el TRUE y para el FALSE – podrían ser 0 y 1 respectivamente). Se permite además en la gramática el uso de operadores and (&&) y puede aplicar a tipos enteros y boolean.

La salida de esta fase debe ser el árbol AST Decorado que hemos discutido en clase. Para esto y como se dijo anteriormente, se deben hacer las modificaciones pertinentes a la estructura del AST para agregar punteros a las declaraciones en los nodos donde sea necesario, además de campos de tipo en las expresiones, necesarios para realizar una identificación del tipo de las expresiones. **Se debe imprimir este árbol utilizando "visitors" al igual que en la primera etapa con la diferencia de que ahora se deben mostrar los adornos. Se recomienda para el tipo en las expresiones, cadenas String como por ejemplo "entero", "char", "arreglo de entero", "clase", etc. Para el caso de los punteros a las declaraciones se recomienda imprimir la dirección relativa de la clase (dirección de memoria) tanto en la declaración como en la utilización de todos los identificadores.**

Como punto final en esta definición veremos algunas reglas contextuales que nuestro compilador en esta fase DEBE cumplir:

1. Los arreglos y las clases son llamados tipos referencia, por lo que se deben manejar dentro de la tabla de símbolos con un tipo. Además los arreglos solamente pueden ser de enteros o caracteres; **no de clases**.
2. Se debe permitir la operación de suma (+) para valores String. El resultado es también de tipo String y es el resultado de realizar la concatenación de los operandos.
3. La sintaxis no permite "llamar identificadores de otras clases mediante instancias" pero si permite el uso de herencia mediante "extends". O sea la tabla de símbolos **debe** considerar identificadores "heredados".
4. Para efectos de chequeo de tipos, dos arreglo son iguales si sus tipos son los mismos.
5. Nombres predeclarados (para efectos del ambiente estándar en la tabla de símbolos): 1) chr(i); convierte el entero "i" a carácter. 2) ord(ch); convierte el carácter ch en entrero. 3) len(a); retorna el número de elementos de un arreglo o de una cadena String. Estos tres métodos predeclarados son funciones.
6. Ningún identificador debe ser declarados dos veces en el mismo ámbito.
7. Ningún identificador podrá ser utilizado sin haber sido declarado previamente.
8. Chequeo de tipos en las expresiones y algunos "statements" como se mencionó previamente en el documento.
9. El número y tipo de los parámetros en procedimientos/funciones deben ser

igual a los declarados por dicho procedimiento/función.

10. No debe permitir retornar un valor a menos que se encuentre dentro de una función. (el retorno es con una asignación al nombre de la función)
11. La expresión de retorno de una función debe tener el mismo tipo que fue declarado en la función.

Cualquier detalle de implementación o de reglas contextuales obviado en este anunciado puede ser asumido por el programador de la manera que éste considere conveniente previo supervisión y aprobación del profesor. Cualquier agregado o suposición hecha debe ser respectivamente documentada.

Documentación (no mayor a 5 hojas):

La documentación debe ser impresa y debe incluir las siguientes partes:

- Portada formal.
- Soluciones e implementación. Debe incluir cualquier agregado o cambio en las reglas contextuales definidas en este enunciado.
- Resultados obtenidos.
- Concusiones del trabajo.
- Bibliografía.

Aspectos Administrativos:

- La tarea se desarrollará en grupos de máximo dos personas. Los grupos deben ser los mismos de la tarea anterior y deben utilizar el código generado en dicha tarea
- La fecha de entrega será el Domingo **25** de Mayo de 2012 antes de las **10 PM** (**SIN EXCEPCIONES NI MODIFICACIONES A LA FECHA**).
- La entrega se realizará a través del portal tec-digital.
- Cualquier intento de plagio, copias totales o parciales de otras personas o de Internet, serán castigados con nota de 0.