Proyecto No. 1
Análisis Semántico

I. Descripción

Esta parte del proyecto se enfocará en la fase de análisis semántico. Esta fase se hará sobre un programa escrito en el lenguaje Decaf. El análisis semántico implementará principalmente un sistema de tipos y otras validaciones semánticas que tendrá que cumplir el programa para considerarse válido. Esta fase incluye las siguientes tareas:

- Agregar acciones semánticas al analizador sintáctico, de tal forma que se construya un árbol sintáctico. Durante la construcción de este árbol o en un recorrido posterior, evaluar las reglas semánticas que crea conveniente.
- Construir la tabla de símbolos que interactuará con cada una de las fases de compilación posterior. El diseño de dicha tabla debe de contemplar el manejo de ámbitos, además de almacenar la información que usted considere necesaria para esta fase. Nótese que dicha tabla tendrá que ser utilizada en las fases restantes de compilación.

II. ¿Qué entregar?

- Documentación
  - Descripción de las reglas semánticas y sistema de tipos implementado.
  - Diseño de la aplicación (diagrama de clases, flujo de datos, herramientas utilizadas, etc.)
- GUI para pruebas.
  - Aunque no es punto de evaluación para este curso, la implementación de una interfaz para el usuario es muy importante. Dicha interfaz permitirá escribir programas en lenguaje Decaf, compilación (análisis semántico) y reporte de errores.

III. Reglas semánticas (Resumen)

Este es un resumen de las reglas semánticas básicas que deben implementarse. Para una mayor comprensión deben de leer el documento de Especificación Decaf UVG.

- Ningún identificador es declarado dos veces en el mismo ámbito
- Ningún identificador es utilizado antes de ser declarado.
- El programa contiene una definición de un método **main** sin parámetros, en donde se empezará la ejecución del programa.
- **num** en la declaración de un arreglo debe de ser mayor a 0.
- El número y tipos de argumentos en la llamada a un método deben de ser los mismos que los argumentos formales, es decir las firmas deben de ser idénticas.
- Si un método es utilizado en una expresión, este debe de devolver un resultado.
- La instrucción **return** no debe de tener ningún valor de retorno, si el método se declara del tipo **void**.
- El valor de retorno de un método debe de ser del mismo tipo con que fue declarado el método.
- Si se tiene la expresión id[<expr>], id debe de ser un arreglo y el tipo de <expr> debe de ser **int**.
- El tipo de <expr> en la estructura **if y while**, debe de ser del tipo **boolean**.
- Los tipos de operandos para los operadores <arith_op> y <rel_op> deben de ser **int**.
- Los tipos de operandos para los operadores <eq_ops> deben de ser **int**, **char o boolean**, y además ambos operandos deben de ser del mismo tipo.
- Los tipos de operandos para los operadores <cond_ops> y el operador **!** deben de ser del tipo **boolean**
- El valor del lado derecho de una asignación, debe de ser del mismo tipo que la locación del lado izquierdo.

IV. Sugerencias

- Se recomienda poner esfuerzo en el diseño y documentación de la aplicación. Recuerden que su diseño les acompañará el resto del proyecto.
- Los errores que sean reportados deberán estar acompañados con la información más relevante. Por ejemplo, número de línea, descripción del lexema que se refiera al error, etc.
- El análisis semántico se realiza de una forma más intuitiva recorriendo el árbol desde la raíz a las hojas. Además una implementación del patrón de diseño "visitante" para las reglas semánticas podría serles de utilidad durante la implementación de su proyecto (además esta implementación es ¡elegante!:D)