Taller de Diseño de Software - Compiladores

Integrantes del grupo: Ezequiel Depetris
Gaston Massimino

A continuación se presentan consideraciones tomadas en cuenta, cambios y decisiones de diseño en cada una de las etapas llevadas a cabo en el desarrollo del compilador:

- Analizador Sintáctico: Se respeto la gramática como fue presentada en un principio pero se decidió agregar un no terminal extra al cual llamamos lambda y es utilizado para representar el caso vacío. Otra decisión no de diseño sino que viene dada por leer el mail con las condiciones un poco mas tarde, fue usar la extensión *.plum* para nuestros test. No tienen nada que ver con el diseño sino que solo fue por no leer el mail a tiempo donde ya se presentaba un formato para los mail y no querer cambiar la extensión de 54 archivos.
- Analizador Semántico: Para esta etapa como primera medida se corrigieron detalles de la entrega pasada. Se le agregó comentarios a todos los test, donde se explica el propósito del mismo(que es lo que se quiere testear). Algunos test se modificaron en detalles mínimos, ademas de contar con otros nuevos. Se realizó una refactorización del código entregado en la etapa anterior. El método main ya no forma parte del archivo *Parser.cup*. Se cambió la implementación de la declaración de atributos, se dejo de lado el modelo *type* seguido de una lista de *id's* para pasar a tener una lista de elementos de la forma *type id*. Se usaron varias implementaciones del patrón Visitor, entre las cuales mencionamos las siguientes:
  - ASTVisitor: interfaz que implementan todos los visitor's
  - PrintAST: Realiza una pasada por el árbol imprimiendo su estructura
  - DeclarationChecker: Recorre el árbol decorando todas las expresiones con sus correspondientes tipos
  - TypeChecker: Realiza el checkeo de tipos de métodos, variables, parámetros y expresiones
  - MainChecker: Revisa que el programa cuente con una clase *main*, la cual contiene un método *main*, y checkea que no halla clases repetidas
  - CicleChecker: Recorre el árbol en búsqueda de expresiones *break* y *continue* para revisar que solo se encuentren dentro de un ciclo
  - ReturnChecker: Recorre el árbol buscando sentencias *return* para verificar que estén dentro del cuerpo de un método

Por ultimo, se decidió implementar una clase *Error* con todos los métodos utilizados para lanzar o identificar los errores del programa por linea de comandos. Hasta el momento solo se informa de manera muy sencilla si hay algún error de tipos o errores semánticos en las estructuras, dando una ubicación de linea y columna del error encontrado.