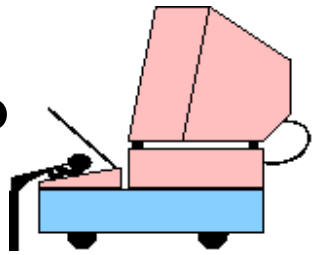
# Práctica 2: Análisis semántico



## Especificación del analizador semántico

Tal y como se realiza en un compilador habitualmente, en la fase de análisis semántico se debe comprobar que el programa fuente se ajusta a todas las especificaciones del lenguaje de programación fuente que no hayan podido ser comprobadas en la fase de análisis sintáctico. A estos efectos, se debe entender que el enunciado de la práctica 1 forma parte de dichas especificaciones, y por lo tanto, el analizador semántico deberá comprobar que el programa fuente se ajusta a todo lo indicado en dicho enunciado y que no se haya verificado en la fase de análisis sintáctico.

Esto aplica particularmente a las comprobaciones de tipos de las expresiones. Las expresiones que combinan una o 2 subexpresiones se deberán ajustar a lo indicado en la práctica 1. Por ejemplo, de acuerdo con el enunciado de la práctica 1, una expresión que combine una subexpresión de tipo `int` con otra subexpresión de tipo `string` por medio del operador '+' debería producir un error de tipo, detectado en la fase de análisis semántico.

Además el analizador semántico deberá comprobar lo siguiente:

- No se deben declarar en el programa fuente 2 variables con el mismo nombre (incluida la o las variables que se declaran en la cabecera del programa y que se utilizan para que el programa pueda recibir argumentos al ser ejecutado).
- Todas las variables utilizadas en un programa deberán haber sido declaradas.
- En una sentencia condicional (`if`), la expresión debe ser de tipo `bool`, mientras que las sentencias que forman parte de la sección `then`, de la sección `thenx` y de la sección `else` no deben tener errores de tipo.
- En una sentencia bucle (`while`), la expresión debe ser de tipo `bool`, mientras que las sentencias que forman el cuerpo del bucle no deben tener error de tipo.
- En una sentencia de asignación, el tipo de la variable en la parte izquierda de la asignación y el tipo de la expresión en la parte derecha debe ser el mismo.
- La expresión que se le pasa como argumento a una sentencia `print` debe ser de tipo `string`.

El analizador semántico debe lanzar una excepción, que no debe ser capturada, cuando se detecte un error semántico. La clase o clases utilizadas para las excepciones correspondientes a errores semánticos se pueden definir libremente, pero deben extender la clase `CompilerExc`, que se proporcionó en la práctica 1. Se valorará que los mensajes que se generen identiifiquen claramente el error producido.

## Requisitos que deben cumplir los ficheros JLex y CUP

Para esta práctica se deberá desarrollar una clase `Main` (puede utilizar como punto de partida la proporcionada en la práctica 1). La ejecución de dicha clase se realizará de acuerdo con lo siguiente:

```
java Main <nombre_fichero>
```

Donde `<nombre_fichero>` es el nombre del fichero (con el path si es necesario) del programa fuente a analizar.

El programa deberá de levantar una excepción que no deberá ser capturada en el caso de que el programa fuente tenga algún error léxico, sintáctico o semántico.

Obligatoriamente las clases generadas por CUP deberán pertenecer a un paquete llamado Parser y la clase generada por JLex deberá pertenecer a un paquete llamado Lexer.

Las clases Java que desarrolle en esta práctica **obligatoriamente** deberán estar organizadas en los siguientes paquetes:

- Paquete `Errors`, que debe contener todas las excepciones utilizadas (incluidas las clases proporcionadas en la práctica 1). Si las excepciones utilizasen alguna clase auxiliar, dicha clase deberá pertenecer asimismo a ese paquete.
- Paquete `AST`, contendrá todas las clases necesarias para representar árboles de sintaxis abstracta correspondientes a programas del lenguaje de programación fuente.
- Paquete `Compiler`, contendrá cualquier otra clase que necesite, incluyendo las de la tabla de símbolos.

La clase `Main` no pertenecerá a ningún paquete.

**Orden de compilación**

Antes de entregar la práctica deberá cerciorarse de que su práctica puede compilarse correctamente con `javac` siguiendo el siguiente orden:

1. Clases del paquete `Errors`.
2. Clases del paquete `Compiler`.
3. Clases del paquete `AST`.
4. Clases `parser` y `sym` generadas por CUP.
5. Clase `Yylex` generada por JLex.
6. Clase `Main`.

Aquellas prácticas que no se puedan compilar en este orden serán calificadas con 0.

## Ayudas y sugerencias

Se proporciona un juego de tests con el que probar la práctica. De entre ellos solamente deberían de producir un error los tests en carpetas cuyo nombre comienza por `ErrSem`, `ErrSint` y `ErrLex`. Puede ocurrir que alguno de los ejemplos `ErrLex` de error de sintaxis (y no léxico). Los tests de las carpetas cuyo nombre comienza por `ErrSem` deberían producir un error semántico.

Se advierte que se realizarán tests adicionales a los proporcionados a las prácticas recibidas, por lo que se recomienda a los alumnos que planifiquen tests complementarios a su práctica.

## Ficheros a entregar

Se deberán entregar exclusivamente los siguientes ficheros:

- fichero `Yylex` en formato JLex para el analizador léxico
- fichero `parser` en formato CUP para el analizador sintáctico
- fichero `java.zip`, que al descomprimirlo genere una carpeta de nombre `java` cuyo contenido debe ser **exactamente** el siguiente:
  - fichero `Main.java` que contenga la clase `Main`.
  - Carpeta `Errors` que contenga las clases Java del paquete `Errors` (proporcionar ficheros Java, no

ficheros `.class`).

- Carpeta `Compiler` que contenga las clases Java del paquete `Compiler` (proporcionar ficheros Java, no ficheros `.class`).
- Carpeta `AST` que contenga las clases Java del paquete `AST` (proporcionar ficheros Java, no ficheros `.class`).

Localización | Personal | Docencia | Investigación | Novedades | Intranet
inicio | mapa del web | contacta

---

*Last Revision: 02/24/2016 20:47:51*