Reporte Compilador de Cool

Ricardo Ivan Valdés Rodriguez | C411 | [@RiiVa](https://github.com/RiiVa)
Gilbert Rafael Garcia Cabrear | C412 | [@GRGarcia066](https://github.com/GRGarcia066)

Arquitectura del compilador

Lexer y Parser

Para la implementación del Lexer y Parser se utilizó la herramienta ANTLR4 donde a partir de la gramática `Cool.g4` se generaron los tokens y donde después usaríamos el AST generado por ANTLR4 se encuentra en `CoolParser.cs`. Cabe destacar que ANTLR4 trata la recursión izquierda por nosotros eliminando la ambigüedad de esta y establece el orden de las operaciones acorde del orden en que definimos las producciones para cada no terminal.

ContextAST a AST

En esta parte definimos nuestro propio AST para más comodidad a la hora de recorrer el árbol para el chequeo semántico y la generación CIL, entonces en esta parte solo recorremos el AST creado por el ANTLR4 y lo convertimos en el nuestro.

Análisis semántico

Después de tener nuestro AST armado vendría el chequeo semántico donde recorremos nuestro AST 2 veces, antes ordenamos las clases por orden topológico y revisamos los casos de herencia cíclica o clases repetidas.

1- La primera pasada recorremos cada clase capturando la definición de los tipos, las propiedades y los métodos para cada clase, además de ir añadiendo al scope la herencia, para después saber los métodos del padre para cada clase.
2- Analizamos las expresiones definidas en el cuerpo de los métodos y las inicializaciones de las propiedades, analizando el tipo estático de cada método y atributo y analizando cada uno de los distintos expression definidos en la gramática.

Se implementó IScope para atrapar los `tipos`, los `métodos` y `propiedades` por cada uno, los `parámetros` para cada uno de los métodos además de tener funcionalidades que nos serán de gran ayuda como saber si un tipo o método está definido entre otras cosas.

Generación de código

Generando de código de Cool a Cil

En esta parte se hacen dos pasadas:

1- Para almacenar las clases básicas y las posibles nuevas definidas por el programador y definir los métodos que tiene cada clase en orden topológico de forma que cada vez que se define una clase nueva esta también almacena los métodos de la clase que hereda.
2- Para generar el código de las `direcciones` de las funciones.

Generando de código de Cil a Mips

El programa en mips se divide en 2 secciones fundamentales:

.data en la cual van las definiciones de tipos, excepciones, strings, y la herencia

.text que contiene todo el código del programa, empezando por los constructores de los tipos básicos y sus funciones, después un main que es donde comienza el código a generar.

La estructura de las clases en mips es:

Nombre de la clase,

Tamaño de la clase,

nombre de la clase de la que hereda,

puntero a la definición de la función por cada una de las funciones

Compilando el proyecto

```
$ cd src
$ make
```

Ejecutando el proyecto

Para lanzar el compilador, se ejecutará la siguiente instrucción:

Para linux

```bash
$ cd src
$ ./coolc.sh <input_file.cl>
```

Para Windows

```cmd
$ ./coolc.bat <input_file.cl>
```

El cual genera el un `<nombre>.asm` listo para correr en spim.