Universidad del Valle de Guatemala. Semestre 1, 2018. Departamento en Ingeniería en Ciencias de la Computación y Tecnologías de la Información.

Fase No. 1, Proyecto – Construcción de Compiladores

BROLO, G.
Carnet 15105

Documentación presentada para el curso de CONSTRUCCIÓN DE COMPILADORES, S. 10, Universidad del Valle de Guatemala.

Abstract

Este proyecto abarca la implementación de un sistema de tipos para el lenguaje decaf, así como la implementación de una tabla de símbolos y la implementación de las reglas semánticas para el mismo lenguaje[1], en pro de la construcción de un compilador para el lenguaje Decaf. Asimismo, se integran estas funcionalidades a través de un IDE. Se utilizó el lenguaje Java como lenguaje principal para la construcción del compilador, el framework Vaadin para la interfaz gráfica del IDE y ANTLR 4 para la construcción del árbol sintáctico y el recorrido de este para la implementación del sistema de tipos y las reglas semánticas.

Objetivos

Implementar el conjunto de reglas semánticas descritas en el proyecto, así como implementar el sistema de tipos propuesto. Agregar las acciones semánticas al analizador sintáctico y construir la tabla de símbolos del compilador. Realizar un IDE gráfico para la implementación de pruebas.

[1] Obtenidas de la guía de la fase 1 del proyecto.

Metodología

El conjunto de reglas semánticas aplicadas es: ningún identificador se puede declarar dos o más veces en un ámbito, ni puede ser utilizado sin ser antes declarado. Existe un método *main* en cada programa, de lo contrario el programa no puede compilarse. Al declarar arreglos, *num* (el valor del tamaño del arreglo) es un entero mayor a cero. El número y tipo de parámetros en la llamada a un método debe coincidir con la definición del método, i.e. cada parámetro corresponde al tipo y se debe proporcionar la cantidad de parámetros que es.

Si el método es de tipo *void*, no existe *return*, de lo contrario, debe existir un *return* con el mismo tipo que debería devolver el método. Dentro de un *if* y *while*, la condición es de tipo *boolean*. Los operadores aritméticos y relacionales solo pueden operarse con operandos de tipo *int*, mientras que los operandos para los operadores de igualdad deben ser *int*, *char* o *boolean* y los operandos deben de ser del mismo tipo. Los operandos para los operadores condicionales son de tipo *boolean*. El valor del lado derecho en una asignación debe corresponder con el tipo de la *location* del lado izquierdo.

Dentro del sistema de tipos se encuentran las siguientes operaciones: existe un tipo de dato llamado *int*, existe otro tipo de dato llamado *char*, existe otro tipo de dato llamado *struct*, existe un tipo de dato llamado *boolean* y finalmente, existe un tipo de dato llamado *void*. Dados A y B, con tipo *int*, A y B pueden ser operados con los operadores +, -, /, * y %, devolviendo un dato C de tipo *int*.

Dados A y B, con tipo *int*, A y B pueden operarse con los operadores >, <, ==, !, >= y <=, devolviendo un dato C de tipo *boolean*. Dados M y N con cualquiera de los tipos existentes, M = N es una operación válida sí y solo si M y N son del mismo tipo; la operación devuelve un tipo de dato *void*. Existe un dato *true*, y es de tipo *boolean*. Existe un dato *false*, y es de tipo *boolean*. Para la estructura del *if*, el tipo del parámetro A es de tipo *boolean*. Para la estructura while, el tipo de parámetro es boolean. A continuación, se muestra la definición del sistema de tipos empleado:

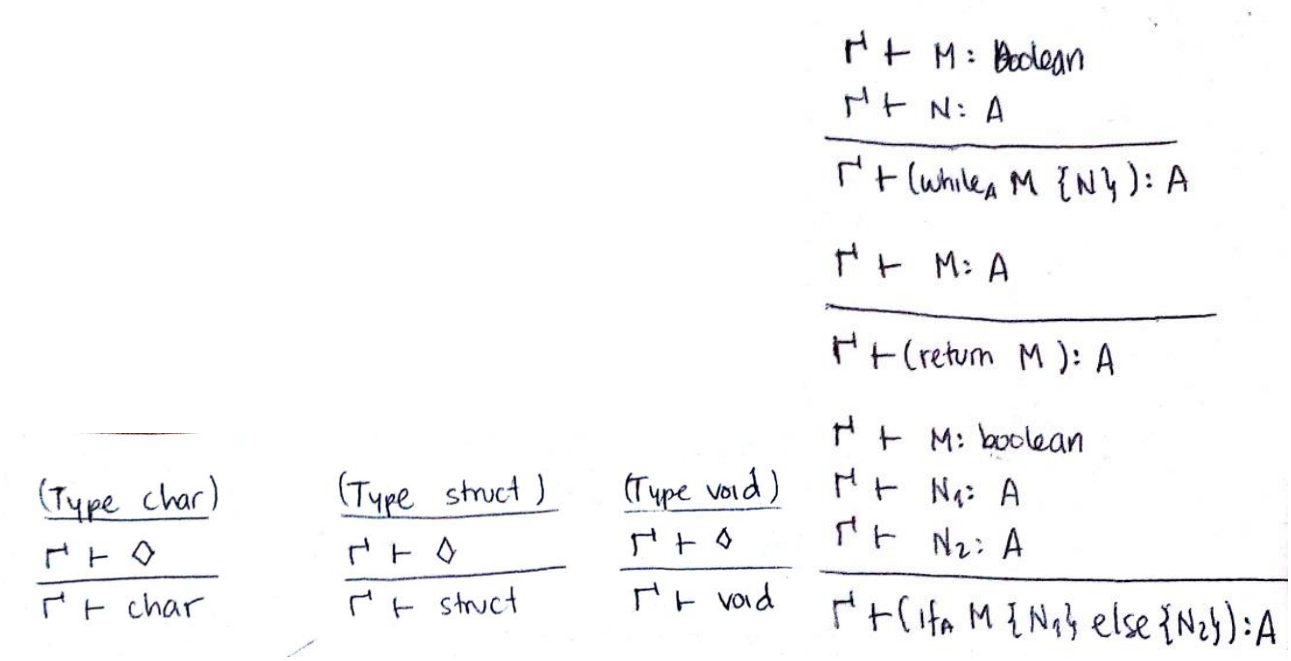

Figura 1. Parte del sistema de tipos empleado.

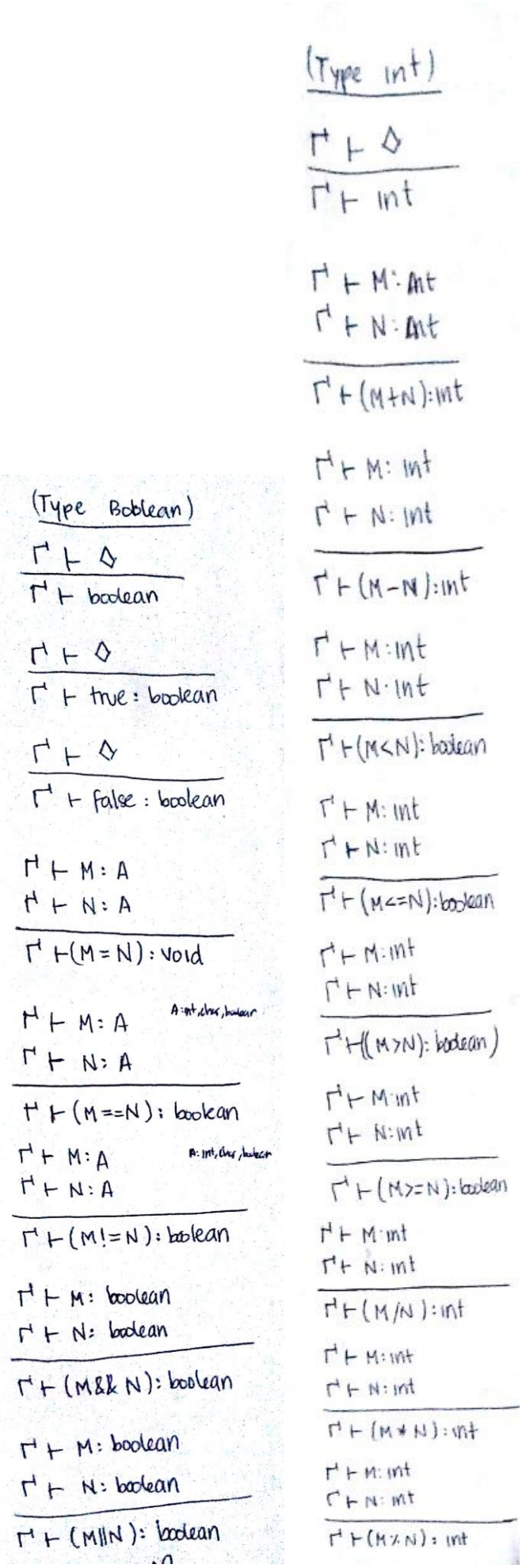

Figura 2. Otra parte del sistema de tipos empleado.

Se utilizó el lenguaje Java 8 para la construcción del proyecto. El *framework* web *Vaadin* para la interfaz gráfica y la presentación de resultados de compilación. Se utilizó ANTLR 4 para la generación del árbol sintáctico. Se utilizó un *listener* de ANTLR para recorrer el árbol e ir verificando las acciones semánticas de la gramática y para implementar el sistema de tipos. La tabla de símbolos se implementó como una lista que guarda objetos de diversos tipos; existen objetos de tipo Variable y tipo Método, así la tabla sabrá que tipo esta guardando y luego se podrá verificar que las implementaciones del sistema de tipos y las reglas se den con los tipos correctos. Para el tipo Struct se hizo una extensión del tipo de objeto de Método, verificando además cada objeto de tipo variable dentro de su declaración.

El flujo común del programa se encuentra plasmado en el siguiente diagrama:

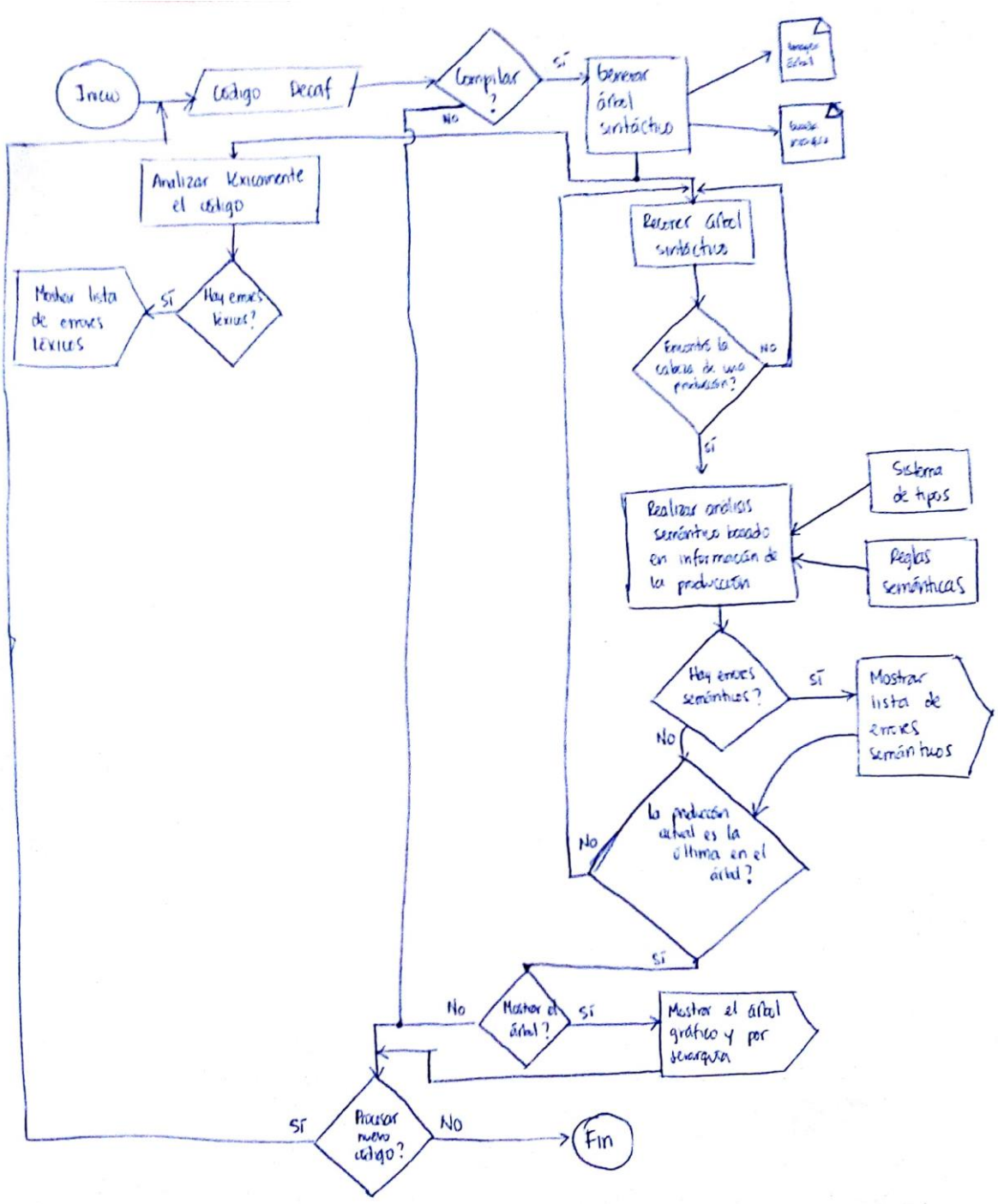

Figura 3. Flujo del programa..

Pruebas

Se realizaron diversas pruebas para verificar la propia implementación de las reglas semánticas y el sistema de tipos. Los archivos de prueba se encuentran en el directorio del proyecto bajo la ruta: *examples/*.

Resultados y comentarios

Se implementaron las reglas semánticas de la guía del proyecto y el sistema de tipos propuesto. Se construyó además un IDE web para poder compilar código en Decaf. Por esto mismo, se cumplió con las funcionalidades propuestas en la guía del proyecto. Se podría mejorar en la presentación de algunos errores, ya que hay errores de css en dónde algunos errores no se ven por completo en el área de la consola.

Durante la presentación se encontró que varias reglas no se habían terminado de implementar. Esto se debió en su mayor parte a que no se habían entendido por completo las reglas y se pensó que se habían terminado de implementar, en algunos casos y en otros casos no se consideraron casos pequeños en donde estas reglas podían fallar. Sin embargo, los errores ya fueron arreglados. En un futuro sería mejor estar totalmente seguros de haber entendido qué funcionalidad es la que se implementará.

Con fines de mostrar brevemente el IDE y algunas ejecuciones, se muestra a continuación, un ejemplo de una ejecución exitosa:

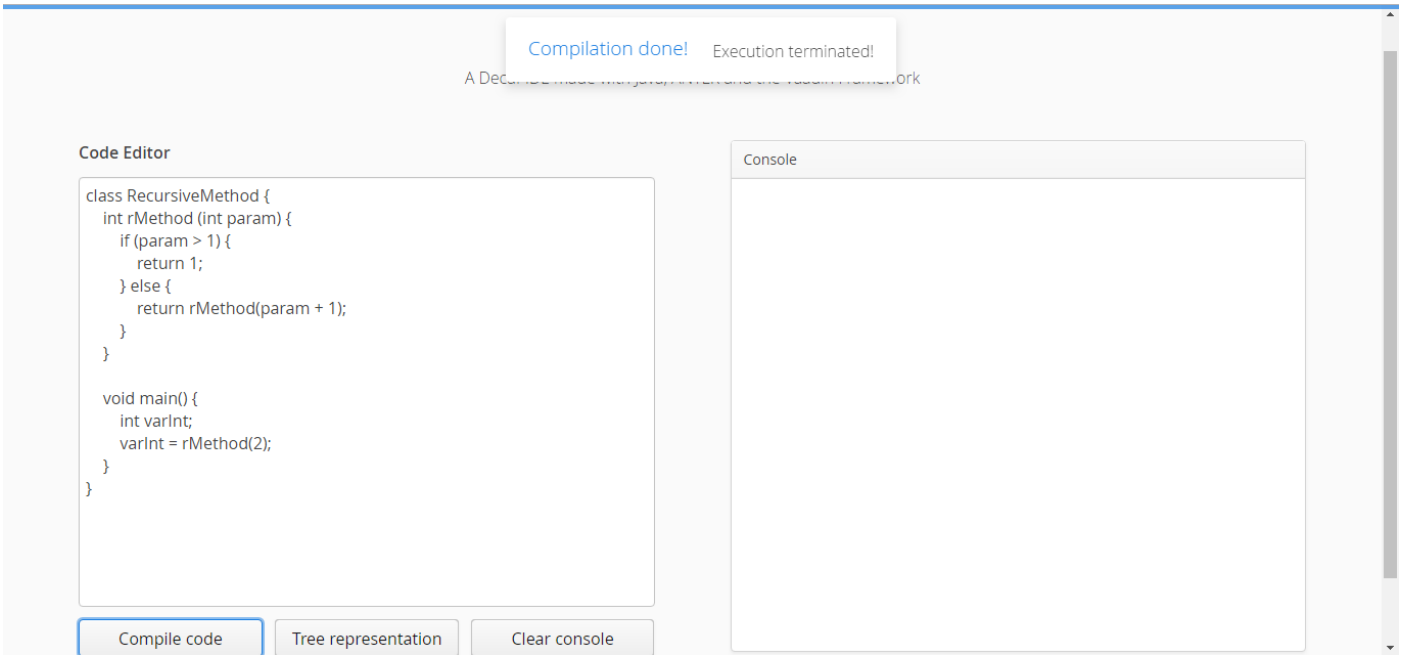

Figura 4. Ejecución correcta.

Ahora, una ejecución con errores:

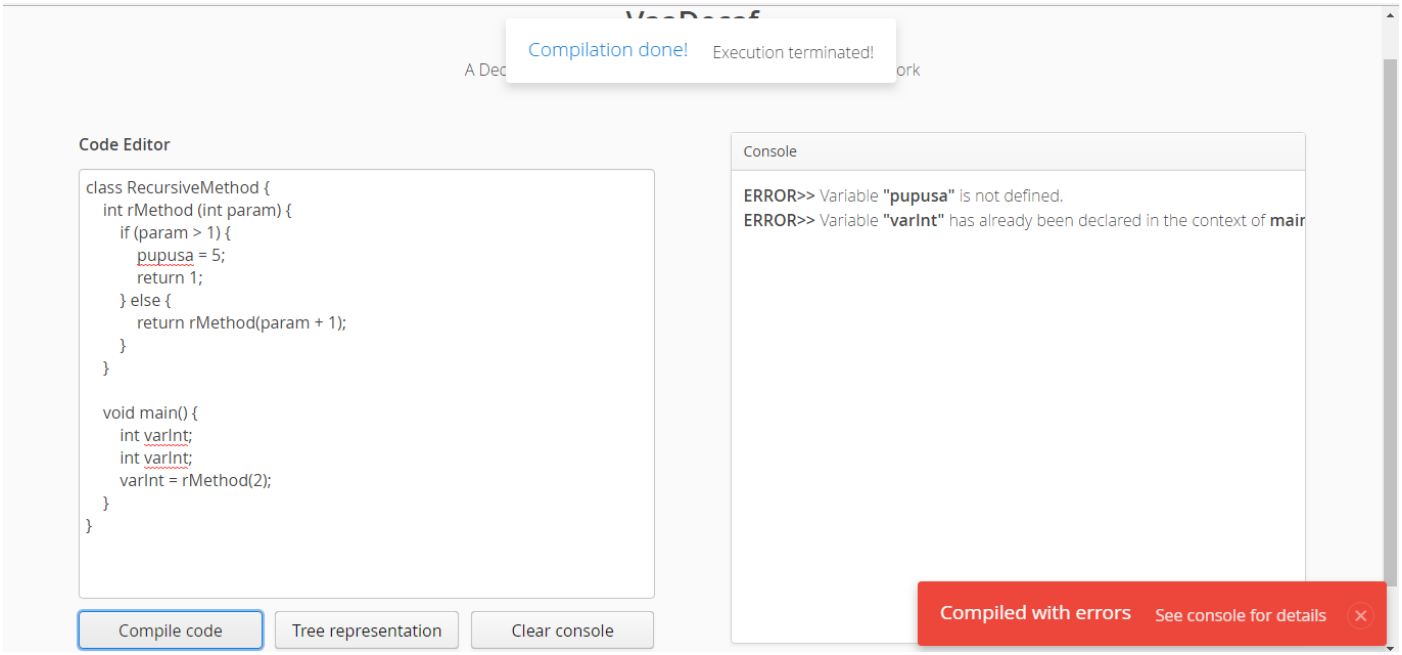

Figura 5. Ejecución con errores.

Repositorio

El repositorio **privado** del proyecto se encuentra en: https://github.com/gbrolo/decaf-ide

Diagrama de Clases UML

Se adjunta un diagrama de clases en el directorio del proyecto, en la ruta: *src/main/java/com/brolius/UML/class-diagram.png* En dicho archivo se puede visualizar de mejor forma el diagrama, el cual es el mismo presentado a continuación: