Universidad Nacional de Río Cuarto
Facultad de Ciencias Exactas Físico-Químicas y Naturales
Departamento de Computación
Taller de Diseño de Software
(Cod. 3306)

Compilador de codigo C-TDS

Bongiovanni Ignacio
Año 2016

1. Descripción del lenguaje

El proyecto de la materia consiste en implementar un compilador para un lenguaje orientado a objetos simple, similar a C, llamado C-TDS.

Para las consideraciones del Lexico, Gramática y Semántica se siguieron los parametros indicados en el documento de especificacion del lenguaje provisto por la materia.

2. Etapas del Proyecto

2.1. Análisis Lexico-Sintactico.

Decisiones de Diseño:

- Los numeros reales no son discriminados por el Scanner, el mismo solo reconoce enteros, por lo que los reales son pasados al Parser como un entero (INT_LIT), seguido de un punto (DOT), y otro entero. Luego el Parser compone estos tres elementos en una regla que genera un no-terminal 'float_literal'.
- Las expresiones binarias (aritmeticas, relacionales, comparacionales y condicionales) fueron agrupadas bajo la misma regla (expression) para simplificar las cuestiones de precedencia de operadores.

2.2. Análisis Semantico.

Estrucutra del Árbol Sintactico Abstracto:

La estructura del AST para representar el cuerpo de un programa se realizó a partir del codigo y esquema provisto por la materia, realizando, cuando se consideró conveniente, el rediseño de las clases y jerarquias del mismo.
Junto a este documento se puede encontrar el diagrama de clases en formato UML que corresponde al paquete *ir.ast* (archivo 'ast-class-diagram.svg'). La clase raíz (AST) es de la cual extienden todos los demas componentes, y el segundo nivel de la jerarquia se compone de las siguientes clases:
Program, ClassDecl, Body, Declaration, Expression, Statement. Las primeras tres son clases concretas, mientras que las ultimas son clases abstractas a partir de las cuales se definen el resto de los elementos.

Decisiones de Diseño:

- Las **declaraciones** se agruparon bajo una superclase llamada 'Declaration'. Las declaraciones que tienen un identificador se agruparon a su vez bajo una superclase denominada 'NamedDecl' que es descendiente directa de 'Declaration'.
- Los **tipos** se manejan mediante Strings, para permitir el uso de tipos definidos por el usuario. Se delega en la clase Type la capacidad de, mediante metodos estaticos (es decir que no se necesita instanciar un objeto de la clase Type), decidir cuando un tipo es basico, numerico, booleano, dos tipos son compatibles entre sí, etc.
- La **tabla de simbolos**, utilizada por el Builder Visitor, se comporta como una pila, donde cada elemento representa un nivel de ambiente de ejecución, donde se definen clases, atributos y metodos, parametros, o variables locales según sea el tipo del nivel corriente.
- Los parametros y las declaraciones internas a un metodo se encuentran a diferentes niveles, por lo que la declaracion de una variable local del mismo nombre que un parametro, hace inutilizable a este último.

En los visitantes:

Responsabiliades asignadas al Builder Visitor: Las cuestiones basicas de busqueda y asignacion de referencias entre locaciones y declaraciones de metodos, variables y arreglos. Además, por cuestiones de comodidad, se le asignaron las siguientes responsabilidades extras:

- Controlar que el numero de argumentos en la llamada a un metodo sea el mismo que la cantidad de parametros formales en la declaracion.
- Al finalizar el proceso de busqueda de referencias, se debe determinar si se definió una clase con nombre 'main'.
- Determinar que la clase main contenga un metodo 'main' y que el mismo

sea de tipo void.
- Controlar que los attributos de una clase sean de tipos basicos. Con el uso de la tabla de simbolos, se puede discriminar cuando el analisis de un elemento del tipo FieldDecl pertenece a la etapa de declaracion de atributos o es la declaracion de un field de variables locales. Por lo que de ser el primer caso, se debe restringir el tipo de las declaraciones. Es por esto que esta tarea que podría pensarse como mas apropiada para la etapa de chequeo de tipos se realiza en esta instancia.

Responsabiliades del TypeCheckVisitor:
Debe controlar el uso de tipos basicos para los retornos de los metodos y los parametros de los mismos.
Que las asignaciones se hagan sobre tipos basicos (y en caso de que sean incrementos o decremetos se realicen sobre tipos numericos).
Que el tipo de la expresion a asignar sea compatible con el tipo de la locacion.
En las expresiones de retorno, se controla que respeten el contexto del metodo al que pertenecen (e.g. un return void es invalido en el contexto de un metodo de un tipo concreto, una expresion de tipo bool no puede retornarse si el contexto no es de tipo bool).
Las condiciones de los if y los while deben ser de tipo bool.
Las sentencias break y continue no pueden estar fuera de un ciclo.
Las expresiones de un for, tanto inicializacion como limite deben ser de tipo integer.
En las expresiones binarias y unarias: si es una expresion aritmetica o relacional, sus operandos deben ser numericos; si es una expresion logica (AND, OR, NOT), sus operandos deben ser de tipo bool.
El tipo de cada argumento en la llamada a un metodo debe ser compatible con el del parametro definido en la posicion correspondiente.
Los metodos de tipo void no pueden ser utilizados como expresion.

Limitaciones encontradas:

No se logró determinar que un metodo de un tipo concreto termine siempre con una setencia de return, es decir, no llegue al final de la declaración.