Proyecto: Compilador de Lenguaje C-TDS

Asignatura: Taller de Diseño de Software

2016

Integrantes: Leandro Buttignol, Martín Lopez, Mariano Coria

Docentes: Pancho y Ceci

**Introducción**

En el presente informe se han de detallar las decisiones de diseño tomadas en cada una de las etapas de contrucción del compilador para el lenguaje C-TDS.

Para conocer en profundidad las etapas y el lenguaje en sí, remitirse a los archivos complementarios a este informe, los mismos son citados a continuación.

**00-TDS-proyecto.pdf.** En este archivo se encuentra la descripción del proyecto.

**01-TDS-spec-lenguaje.pdf.** En este archivo se encuentra la especificación del lenguaje para el cual se quiere implementar un compilador.

***Primera Etapa: Análisis Léxico y Sintáctico***

**Herramientas de desarrollo utilizadas**

* + JFlex
  + Sublime text 3
  + Ant
  + CUP
  + Eclipse

**Objetivos de desarrollo en primer etapa**

Los objetivos parciales concernientes al desarrollo del compilador en esta etapa temprana son la implementación de scanner y parser, como así también la definición de la gramática.

**Grammar/Parser**

En esta primera etapa se busca corroborar que el código escrito en un archivo sea una cadena correspondiente al lenguaje C-TDS definido en la cátedra, es decir, discriminar las palabras reservadas del lenguaje.

En primer instancia, se realizó la implementación del scanner, para ello se utilizó la herramienta JFlex.

Una vez que dicha implementación del scanner se consideró terminada, se prosiguió con la implementación del parser utilizando la herramienta CUP. Es aquí donde se define la gramática de nuestro lenguaje C-TDS.

**Problemas**

Uno de los problemas enfrentados durante la ocurrencia de esta primera etapa fue el reconocimiento de multiples comentarios multilinea (el cual pudo ser solucionado). La inclusión de facilidades para el programador como el aumento y decremento automático, las mismas fueron causantes de complicaciones en la implementación por lo que se decidió no incluirlas en este entrega.

**Testing**

Se realizó una batería de numerosos tests con distintos niveles de complejidad y rigurosidad, en los mismos se realizaron casos de prueba para todas las sentencias que se pueden declarer en nuestro lenguaje, como así también para todos los tipos de variables y operaciones entre ellas.

Segunda etapa : Análisis Semantico

Intro/ objetivo

Al hablar de un compilador resulta inevitable la inherente conexión a los chequeos semánticos que determinen a priori la correción del código antes de proseguir a generar la traducción a assembler o el respectivo binario ejecutable. En esta etapa, nos dedicamos íntegramente a estos chequeos, los mismos pueden dividirse en 2 grandes ramas que son alcanzabilidad y coherencia de tipos.

Alcanzabilidad

Cada vez que se invoque a una variable o método, éste debe estar definido de tal manera que, respetando las convenciones del lenguaje, sea alcanzable desde el bloque de código en la cual se lo invoca. Como convención para ctds se adopta el sistema de alcance de java, que posee un alcance estático **-*static scope-*.** Por lo tanto, cada vez que se invoque a un identificador, éste ha de estar definido en el mismo nivel o uno superior y siempre respetando el encapsulamiento de información, es decir, en un nivel superior pero dentro de la misma clase.

Para llevar a cabo los chequeos de alcanzabilidad, se definió una clase visitor denominada BuilderVisitor, esta tiene entre sus atributos una instancia de la clase SymbolTable, que no es más que una pila, en la cual cada nivel es una lista de definiciones de símbolos, existiendo un nivel por cada bloque anidado. En concreto, cada vez que el builder visita a un bloque, genera un nuevo nivel en esa pila con una lista de todos los indentificadores que se encuentren definidos en el bloque en cuestión. Estos niveles se eliminan al terminar de recorrer el bloque. Entonces, cuando se invoca a un identificador se lo busca en la lista de símbolos del nivel actual, en caso de no encontrarlo se lo busca en el nivel inmediato inferior hasta encontrarlo o llegar al nivel cero (en este caso, el identificador no está definido y estamos frente a un error semántico). A su vez, cada vez que se define un identificador se chequea que no exista otro con el mismo nombre en el nivel actual, evitando así posibles ambigüedades.

Coherencia de tipos

Problemas/soluciones

testing

Tercera Etapa: Código intermedio

Intro/ objetivo

En esta etapa se busca obtener un programa equivalente al escrito por el usuario, pero en código de tres direccónes cuya semántica está dada por los integrantes del grupo. Básicamente, lo que se realizó fué deglozar las instrucciones de nuestro lenguaje ctds en código más cercano al assembler, es decir, una lista de instrucciones con tres parámetros cada una,sin tener en cuenta detalles como direcciones de memoria de las variables y registros del procesador, para abstraerse de algunos detalles de bajo nivel.

Problemas/soluciones

Él problema más grande de esta etapa fue construír el set de instrucciones que iba a soportar el lenguaje de código intermedio, y darle la semántica adecuada.

Algo que no se pudo resolver hasta la etapa siguiente fue el acceso a atributos de la clase y a arreglos.

Testing

Las pruebas en esta etapa requerían de mucho trabajo del programador debido a que era el encargado de ejecutar mentalmente las instrucciones del código de tres direcciones y determiner si el resultado era el adecuado o no de acuerdo a la semántica establecida para cada sentencia.

Cuarta Etapa: Generación de Assembler

Intro/ objetivo

Problemas/soluciones

testing

Quinta Etapa: Optimizaciones

Intro/ objetivo

Problemas/soluciones

testing

Detalles propios del lenguaje:

El lenguaje soporta todas las operaciones descriptas en la especificación (archive 01-TDS-spec-lenguaje.pdf).

Se pueden ejecutar programas con variables de tipo integer , bool , arreglos y objetos con atributos de estos dos tipos. El compilador está preparado para soportar instricciones con variables float solo que la traducción a assembler de las mismas no fue realizada por el sólo hecho de priorizar la etapa de optimización.