# Diseño de Compiladores I

Trabajo Práctico Especial Partes 3 y 4.

****

Integrantes:

* Marín, Cynthia Romina (cynthia.romina.marin@gmail.com)
* Sarti, Mauro (msarti@alumnos.exa.unicen.edu.ar)

Ayudante Asignado: Tommasel, Antonela

Número de Grupo: 10

Parte 3 – Generación de código Intermedio

**Introducción y Objetivos**

El objetivo de este trabajo práctico es la generación del código intermedio para las sentencias ejecutables. En nuestro caso nos tocó la representacion Terceto.

Se deberán generar tercetos para asignaciones, selecciones, sentencias de control asignadas al grupo, y sentencias OUT. En nuestro caso puntual, las sentecias de control asignadas fueron, IF y SWITCH.

En el caso de la sentencias SWITCH se incorporaron los caracteres “{” y “}” como caracteres válidos.

También se necesitó agregar información adicional a la tabla de símbolos como el tipo de dato que retornan las funciones, el significado de un identificador (variable o función), el tipo de cada variable. Al poder definir variables dentro de las funciones tambien nos vimos obligados a establecer ambitos para las variables, de esta forma podemos ver si una variable es alcanzable o no desde un punto específico del programa.

En cuanto a los chequeos semánticos, se verifica el uso de variables o funciones no declaradas, la redeclaracion de variables o funciones, los chequeos de compatibilidad de tipos, en nuestro caso no se permiten conversiones por lo tanto en caso de que los tipos no sean iguales no se permitirá ninguna operación.

Decisiones de Diseño e Implementación

Para este trabajo se decidió hacer una clase Sintáctico, la cual interactúa con las clases ya existentes en el proyecto y con la clase Parser, generada por el Yacc. La clase Parser es la que realiza los chequeos sintácticos recorriendo la gramática realizada y aplicando las distintas acciones semánticas que se asignaron a las reglas. También se realizó la implementación de una clase abstracta Terceto de la cual se extendió para realizar todos los tercetos necesarios, por ejemplo, TercetoAsignacion, TercetoSuma, TercetoResta, etc.

En la clase Sintáctico se encuentra, entre otras cosas, una lista de Tercetos, la cual contiene todos los tercetos en el orden en el que se crearon. Cada Terceto se crea en función de la gramática, por lo tanto, en función del código ingresado.

Junto con la clase Parser, Yacc genera otra clase, ParserVal, que es utilizada para acceder a los elementos reconocidos por la gramática.

La clase terceto tiene tres elementos principales, un operando de tipo String y dos elementos, primero y segundo de tipo String, los cuales representan los operadores del terceto o la posición del terceto al que referencian. También se agregaron dos atributos más, una posición, que representa el número de terceto y un tipo de dato, que representa el tipo de dato del terceto. Este último atributo se utiliza para los tercetos se operaciones como suma, resta, multiplicación y división, pero también para el terceto asignación y comparación ya que se debe revisar que sean del mismo tipo ambos lados de la operación.

Parte 2 – Generación de código Assembler

**Introducción y objetivos**

Esta cuarta parte del trabajo se trato de la generación del código Assembler para Pentium de 32 bits, a partir del código intermedio generado en el práctico anterior.

Como mecanismo de generación de código Assembler que se nos asigno fue Variables auxiliares.

Los controles en tiempo de ejecucion que nos tocaron fueron división por cero y overflow en sumas y multiplicaciones.

Decisiones de Diseño e Implementación

Para este trabajo se decidió hacer una clase GenCodigo que es la que arma el archivo .asm. La generación del archivo la separamos en dos partes, una primera parte de “encabezado” donde se realizan todos los imports necesarios y se vuelca el contenido de la tabla de símbolos, luego de haber generado todas las variables auxiliares.

En la segunda parte de la generación es donde se coloca todo el contenido de los tercetos, es decir, el código intermedio, los saltos para los chequeos en tiempo de ejecución y finalmente el código Assembler.

Anteriormente, en el práctico 3 se comentó que se decidió implementar los tercetos como clases separadas donde todas heredaban de la clase abstracta Terceto. Esto fue para que cada terceto sepa como “traducirse” a assembler. Todos los tercetos tienen un método getCodigo(), que devuelve un String donde se encuentra el assembler asociado.

Conclusiones