PRÁCTICA DE
PROCESADORES DEL LENGUAJE II
Curso 2016 – 2017
Entrega de Junio

Diego Diaz
Compilador Lenguaje C

Contenido

Introducción

El objetivo de esta práctica es poder realizar un procesador de lenguaje de tipo compilador. El desarrollo del mismo tiene un fin educativo y no comercial. El lenguaje que se ha de procesar se llama ***cUned***, el cual es una versión simplificada del lenguaje C. La parte de análisis léxico y sintáctico ya ha sido desarrollada por el equipo docente por lo que no se detalla su funcionamiento en este documento; la parte presentada en esta memoria corresponde con la última etapa del análisis (analizador semántico) y con la etapa de síntesis al completo (generación de código intermedio y código final).

1. El analizador semántico y la comprobación de tipos.

Para poder realizar el análisis semántico utilizando esquemas de traducción dirigido por la sintaxis es necesario definir en el fichero parser.cup los atributos que tiene cada símbolo no terminal de la gramática introduciendo código Java en las producciones de la gramática.

Los componentes de CUP son declaraciones que nos permiten incluir código (action code) como parte del "parser". Se pueden producir clases no públicas que contienen acciones específicas.

1.1. Descripción del manejo de la Tabla de Símbolos y de la Tabla de Tipos[1]

El ámbito (scope) puede ser global (se identifica como global), de la función principal (se

[1] Para más información ver el documento Comprobaciones-Finales.pdf

identifica como main) , dentro de un bloque de sentencias (se identifica como anonymus - numero) o de una función (tendrá el nombre de la función).

Se pueden repetir los identificadores de cualquier índole siempre y cuando **no** estén en el mismo ámbito, es por ello que los errores deben de aparecer en declaraciones hechas en un ámbito común.

La Tabla de símbolos y la Tabla de Tipos se utilizan en el análisis semántico para verificar que no se repiten identificadores en un mismo ámbito y para el control del valor de sus atributos.

Tablas de símbolos:

Se añaden los identificadores de las declaraciones de variables y constantes, junto con determinados atributos. Cuando se ve la información en el debugger se podrán ver clasificadas según su ámbito (por lo que aparecerán varias tablas).

Tablas de tipos:
Se añaden los identificadores de las declaraciones de tipos Registro junto con los atributos que se declaren para estas declaraciones.
Estos símbolos tiene asociado un ámbito de declaración (scope) por lo que el resultado final en el debugger se ven varias tablas separadas por ámbitos.

2. Generación de código intermedio

El código intermedio se crea para dar significado a las expresiones y a las sentencias. Dicho código es la primera parte de la fase de síntesis. En el parser.cup se comentan las líneas que muestran las cuádruplas en el debugger. Se pueden descomentar dichas líneas o ver el código intermedio junto con el código final en el archivo .ens generado tras la compilación completa por la tarea fiinalTest de Ant.

2.1. Descripción de la estructura utilizada

El código intermedio se genera en los dos paquetes que se muestran en la **imagen 1**. El paquete *compiler.intermediate.Factory* (1) contiene las clases que generan el código intermedio de expresiones y sentencias. Estas clases heredan de otras clases de tal manera que hace posible la organización de la traducción semántica en el parser.cup.
El paquete *compiler.intermediate* (2) contiene las clases que generan el código intermedio relacionado con los valores que toman las variables y parámetros.

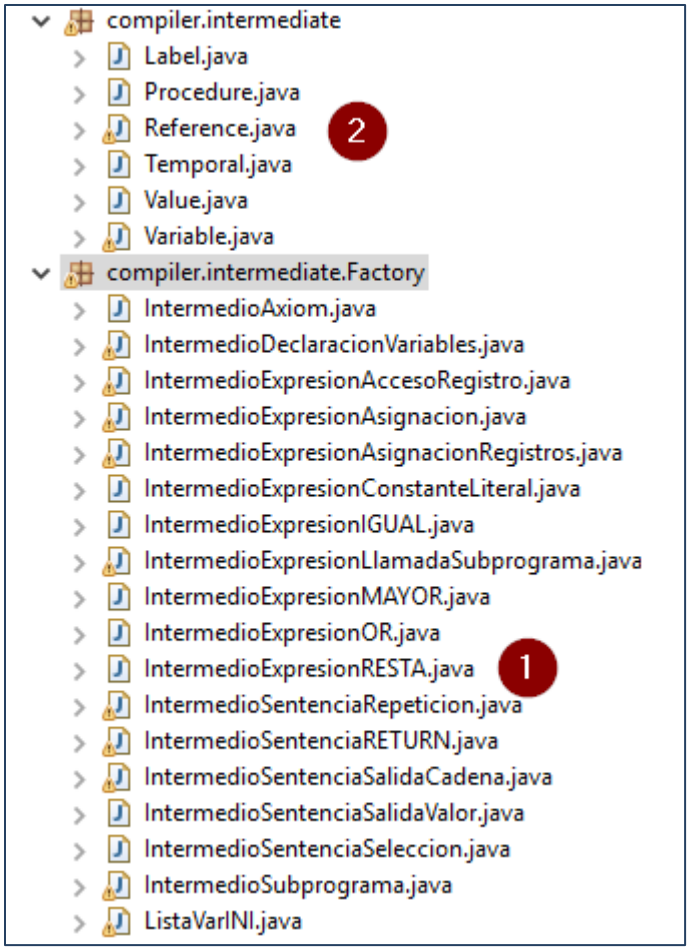

Imagen 1. Paquetes del proyecto donde se declara el código intermedio

Se detalla a continuación un ejemplo de cómo se puede hacer seguimiento de la creación del código intermedio. Tal como se puede ver en la **imagen 2**, se crea una instancia de la clase *IntermedioExpresionRESTA*, que se muestra en la **imagen 1**, con el constructor que está declarado en dicha clase (3). Se llama al método codigoIntermedio que recoge como parámetros los valores sintetizados exp1 (1) y exp2 (2) y crea la cuádrupla correspondiente (4). El atributo sintetizado que se sube en el árbol es dicho código intermedio (5).

```
1421  expLog ::= expLog:exp1 MINUS:operador expLog:exp2
1422          {:                    1                  2
1423
1424          // Codigo intermedio
1425
1426           IntermedioExpresionRESTA expResta = null;
1427
1428           expResta = new IntermedioExpresionRESTA(exp1.getTipoExpresion()); 3
1429           expResta.codigoIntermedio(exp1,exp2);  4
1430
1431            RESULT = expResta;  5
1432
1433          :}
```

Imagen 2. Action code con el código intermedio de la expresión RESTA

3. Generación de código final

El código final es la segunda parte de la etapa de síntesis y corresponde con la traducción del código intermedio a código máquina para el emulador ENS2001. En el paquete *compiler.code.translator* se encuentran las traducciones correspondientes.
Tras ejecutar la tarea finalTest de Ant se obtiene un fichero con extensión .ens en donde se puede ver el códio intermedio (como comentario en el lenguaje ENS2001) y su correspondiente traducción.

3.1. Descripción del registro de activación implementado

Para el diseño del registro de activación se tienen en cuenta las siguientes consideraciones en el lenguaje cUNED:

- Permite recursividad: para ello se necesita un enlace de control el cual permite que al finalizar una función se pueda continuar con la ejecución de la función llamante.

- Las variables en ámbitos internos ocultan a aquellas con las que comparten nombre en ámbitos más externos. Para ello se utiliza la técnica del display. La técnica del display consiste en mantener en todo momento en memoria un vector con el valor de los enlaces de acceso para cada nivel de anidamiento.

Se alacena en el RA: dirección de retorno, variables temporales, variables locales, puntero de marco, valor de retorno, parámetros.

4. Indicaciones especiales

- Se han modificado/eliminado algunas reglas sintácticas el fichero parser.cup.

- Se entrega una serie de programas complementarios que verifican otras especificaciones del lenguaje no contempladas en los test iniciales proporcionados.

- Se entrega junto a la memoria el documento anexo *Comprobaciones-Finales.pdf* donde se detalla en más profundidad el trabajo realizado con *cUNED*.

- Algunas comprobaciones se han comentado a medida que se ha avanzado en el proyecto debido a posibles problemas de compatibilidad con otras partes del desarrollo o porque son posibles ampliaciones no terminadas.