Guía Práctica 4 - MII 779
Optimización y Generación de Código
Desarrollo de Lenguajes Orientados a Objeto y Compiladores Avanzados

Profesor: Ricardo Soto

Ejercicio 1: Instale el generador de código para el lenguaje Mile++

- Descargue el archivo `milePP-gen.zip` desde `http://www.inf.ucv.cl/~rsoto/cursos/MII779/milePP-gen.zip`
- Descomprima y copie en su workspace.
- Cree un proyecto seleccionando como fuente del proyecto la carpeta recientemente descomprimida.

Ejercicio 2: Comprenda la nueva clase `CodeGenerator.java`

- Diríjase a `src->cl.ucv.inf.mileppcompiler->compilers->CodeGenerator.java`. Esta clase permitirá generar código Java desde nuestro archivo fuente mile. Los métodos definidos en esta clase serán invocados por el iterador de AST `MilePPCodeGen.g`
- La clase `CodeGenerator.java` contiene variados métodos para generar código, entre otros, para generar las cabeceras de las clases, de los métodos, asignaciones, etc. Además se han incluído algunos métodos para controlar la indentación (`ind, addInd,subInd` y `resetInd`) del archivo a generar.

En particular, el método `addIdent` participa en uno de los procesos de optimización de código llamado "propagación de constantes", el cual se encarga de reemplazar variables por sus correspondientes valores en el caso de su existencia. Considere las siguientes dos líneas de código,

```
a=2;
b=a+5;
```

al aplicar la propagación de constantes, el resultado es el siguiente:

```
a=2;
b=2+5;
```

La propagación de constantes se ha implementado de la siguiente manera. El método `addIdent`, es el encargado de generar el código correspondiente a una variable, este método llama a `getValue`, el cual recupera el valor de la variable si existe. Este valor se ha almacenado en el correspondiente objeto `MileAttribute` en la segunda pasada del analisis semántico, por medio de `addConstantIfExists` invocado por la regla `assign`.

Ejercicio 3: Implemente la propagación de constantes para los accesos.

Considere el siguiente ejemplo:

```
class foo(){
  numeric a;
}

class foo2(){
  foo b;
  numeric c;
  method numeric calc(){
    b.a = 2;
    c = b.a + 5;
    return c;
  }
}
```

La propagación de constantes debería entregar el siguiente resultado:

```
class foo2(){
  foo b;
  numeric c;
  method numeric calc(){
    b.a = 2;
    c = 2 + 5;
    return c;
  }
}
```

Ejercicio 4: Implemente el ensamblamiento (folding)

El ensamblamiento es otra fase de optimización de código, el cual consiste en reemplazar las expresiones por su correspondiente resultado, considere el siguiente ejemplo:

```
b.a = 2;
c = b.a + 5;
```

Luego de la propagación de constantes:

```
b.a = 2;
c = 2 + 5;
```

Luego del ensamblamiento:

```
b.a = 2;
c = 7;
```

Ejercicio 5: Complete y pruebe el generador de código de Mile++.

Ejercicio 6: Implemente el optimizador y generador de código de su lenguaje.
