IFIPS – Cycle Ingénieur 2^{e} année Informatique
Compilation 2006–2007

Travaux Dirigés n^{o}8
Génération de code

1 Compilation des instructions `break` et `continue`

Dans cet exercice, on s'intéresse à la compilation des instructions `break` et `continue` dans un programme Java. On ne considère ici que les instructions `break` et `continue` sans argument et utilisées à l'intérieur de boucles `while` et `for`.

On suppose que le compilateur manipule une syntaxe abstraite des programmes Java de la forme

```
abstract class Expr { ... }
abstract class Stmt { ... }
class Break extends Stmt { ... }
class Continue extends Stmt { ... }
class While extends Stmt { Expr e; Stmt s; ... }
class For extends Stmt { Stmt s1; Expr e; Stmt s2; Stmt s3; ... }
```

munie d'une interface `Visitor` pour les visiteurs.

Écrire les méthodes correspondant aux instructions `break`, `continue`, `for` et `while` pour un visiteur réalisant la production de code. On supposera que le code est produit par l'intermédiaire de la méthode statique

```
static void emit(String s);
```

et que des étiquettes fraîches peuvent être obtenues par l'intermédiaire d'une autre méthode statique

```
static String new_label();
```

2 Compilation de l'instruction `switch`

Dans cet exercice, on s'intéresse à la compilation de l'instruction `switch` dans un programme Java. Plus précisément, on ne considère que des instructions `switch` de la forme

`switch`(e) {
  `case` v_1: s_1;
  ⋮
  `case` v_n: s_n;
  `default` : s_{n+1};
}

où e est une expression de type `int`, chaque v_i une constante entière et chaque s_i une séquence d'instructions. On se donne la syntaxe abstraite suivante pour cette construction :

```
class Switch extends Stmt {
    Expr e;
    Vector cases; // vecteur de SwitchCase
    Stmt def; ...
}
class SwitchCase { int v; Stmt s; ... }
```

1. Dans un premier temps, on souhaite compiler l'instruction `switch` par une séquence de tests successifs. Écrire la méthode correspondant à l'instruction `switch` pour un visiteur réalisant la production de code. On réutilisera les méthodes statiques `emit` et `new_label` de l'exercice précédent, ainsi que le schéma de compilation de l'instruction `break`.
2. On souhaite maintenant compiler l'instruction `switch` plus efficacement, en effectuant moins de tests. On suppose que la machine virtuelle possède une nouvelle instruction `JUMPA` qui dépile une adresse de code et y effectue un saut inconditionnel. Donner un nouveau schéma de compilation de l'instruction `switch` basé sur une table d'adresses de code. Quels sont les avantages et les inconvénients d'une telle table ?