TP04 - GÉNÉRATION DE CODE TROIS ADRESSES

1. Objectif

L'objectif de ce TP est de générer du code trois adresses à partir d'un arbre abstrait. Pour chaque nœud visité une ou plusieurs instructions trois adresses sont produites.
L'ensemble des instructions ainsi que la nature de leurs opérandes est représenté dans la figure 1.

Type	Instruction	a1	a2	r	Syntaxe
arithmétique	`+ - * /`	ctv	ctv	tv	`r = a1 op a2`
affectation	`=`	ctv		tv	`r = a1`
saut test	`== < <= > >=`	ctv	ctv	e	`if a1 op a2 goto r`
saut direct	`goto`	e			`goto a1`
appel fonction	`call`	e		tv	`r = call a1` ou `call a1` [1]
lecture	`read`			tv	`r = read`
écriture	`write`	ctv			`write a1`
e/s fonction	`param ret`	ctv			`op a1`
début/fin fonc.	`fbegin fend`				`fbegin`

Figure 1. Jeu d'instructions trois adresses, avec, pour chaque instruction la nature de ses opérandes (`c` pour constante, `v` pour variable, `t` pour temporaire et `e` pour étiquette.)

Le parcours de l'arbre abstrait peut être effectué à l'aide du visiteur `SaDepthFirstVisitor` du package `sa` qui définit une méthode `visit` pour tout type de nœud de l'arbre abstrait. Il s'agit du visiteur utilisé pour la production de la table des symboles dans le TP précédent.

2. Le package `c3a`

Le package `c3a` définit une classe pour chaque instruction trois adresses. Ces classes définissent chacune un constructeur qui permet de créer une instance de l'instruction. Il définit aussi une classe pour chaque type d'opérande. Ces dernière sont au nombre de cinq :

- `C3aConstant`
- `C3aFunction`
- `C3aLabel`
- `C3aTemp`
- `C3aVar`

Il y a un type d'opérande de plus que dans la figure 1 car on distingue le cas où une étiquette correspond à une fonction du cas où elle correspond à une simple ligne de code, ne correspondant pas à une fonction. Les classes `C3aFunction` et `C3aVar` comportent chacune un lien vers une entrée d'une table de symboles (la table globale ou une table locale).
Le package `c3a` définit aussi la classe générale `C3a` qui comporte en particulier la liste des instructions trois adresses qui vont être produite lors du parcours de l'arbre abstrait.
La classe `C3a` définit quatre méthodes importantes ;

- `public void ajouteInst(C3aInst inst)` ajoute une instruction à la séquence d'instructions créées.
- `public C3aLabel newAutoLabel()` crée une nouvelle étiquette.
- `public C3aTemp newTemp()` crée un nouveau temporaire.
- `public void addLabelToNextInst(C3aLabel label)` ajoute une étiquette à la prochaine instruction qui sera ajoutée.

La classe `c3a` définit aussi deux constantes, `True` et `False`, utiles pour les sauts conditionnels.

3. La classe `Sa2C3a`

C'est la classe que vous devez implémenter (elle peut avoir le nom que vous voulez). Elle étend la classe `SaDepthFirstVisitor` et traduit chaque nœud de l'arbre en une séquence d'instructions. Le principe de la traduction est disponible dans les transparents du cours. Voici à titre d'exemple le traitement réalisé au niveau du nœud de type `SaExpAdd`.

```
public C3aOperand visit(SaExpAdd node)
{
    defaultIn(node);
    C3aOperand op1 = node.getOp1().accept(this);
    C3aOperand op2 = node.getOp2().accept(this);
    C3aOperand result = c3a.newTemp();

    c3a.ajouteInst(new C3aInstAdd(op1, op2, result, ""));
    defaultOut(node);
    return result;
}
```