Compilation Examen 4 février 2005

Prénom :
Nom :
Section :
Signature :

Exercice 1	Exercice 2	Exercice 3	Exercice 4	Total des points

Exercice 1 : Programmation en Eins *15 points*

On définit en EINS les arbres binaires au moyen de la classe suivante.

```
class Node {
  val contents: Int;
  val left: Node;
  val right: Node;
}
```

Chaque nœud de l'arbre contient une valeur entière supérieure ou égale à zéro. Les feuilles de l'arbre sont représentées par la valeur `null`.

Écrivez en EINS une fonction qui prend en argument un arbre et retourne la plus grande valeur contenue dans cette arbre. Si l'arbre est vide, cette fonction doit retourner la valeur `-1`.

Exercice 2 : Analyse syntaxique *15 points*

Écrivez en JAVA des fonctions d'analyse syntaxique pour le langage décrit par la grammaire EBNF ci-dessous. Dans celle-ci, le symbole `ident` est un symbole terminal.

```
expression ::= ident
             | '{' [ fields ] '}'
             | expression '.' ident
fields     ::= field
             | fields ',' field
field      ::= ident '=' expression
```

Notez que vous ne devez pas construire d'arbre de syntaxe abstraite, mais seulement décider si une phrase appartient au langage. Si ce n'est pas le cas, il faut lever une exception.

Précisons qu'on ne vous demande pas d'écrire une classe `Parser` complète, mais seulement des méthodes d'analyse syntaxique. Celles-ci pourront utiliser les éléments suivants mis à votre disposition :

- les tokens `IDENT`, `LBRACE`, `RBRACE`, `DOT`, `COMMA` et `EQUAL`,
- un champ `token` pour accéder au token courant, et
- une méthode `nextToken()` pour passer au token suivant.

Exercice 3 : Analyse de noms et de types *20 points (10+10)*

On veut rajouter au langage EINS un type d'énoncé (*statement*) qui permette de modifier la valeur d'un champ d'un objet.

On utilise la syntaxe

```
e1.f := e2;
```

pour dire que dans l'objet résultant de l'évaluation de l'expression `e1`, le champ `f` prend la valeur correspondant à l'évaluation de l'expression `e2`.

Voici un exemple d'utilisation de cette nouvelle construction :

```
{
  var lst: List = new List(2, null);
  lst.head := lst.head + 1;
  do printInt(lst.head); // affiche 3
}
```

1. Expliquez en quelques phrases quelles vérifications devraient, selon vous, être effectuées par l'analyseur au moment d'analyser un énoncé de ce type.

2. Formalisez votre idée sous la forme d'une règle de typage.

Exercice 4 : Production de code *20 points (10+10)*

On décide de rajouter à l'arbre de syntaxe abstraite de EINS un nœud `Imply` correspondant à l'implication logique. On rappelle la table de vérité de l'implication $a \Rightarrow b$:

$a \backslash b$	false	true
false	true	true
true	false	true

On ajoute dans le fichier `Tree.java` une classe `Imply` qui est une sous-classe de `Tree` et qui contient deux champs de type `Tree` : `left` pour la partie gauche de l'implication et `right` pour la partie droite.

1. Donnez une implémentation de la méthode `caseImply` dans la classe `GenCond`. Comme dans votre compilateur, vous pouvez, au besoin, utiliser les méthodes `gen`, `genLoad` et `genCond`, ainsi que toutes les méthodes de la classe `Code`.

```
private class GenCond extends DefaultVisitor implements Visitor, RISC {
    private Code.Label targetLabel;
    private boolean when;

    public void caseImply(Imply tree) {









    }

    // reste de la classe GenCond
}
```

2. Donnez la suite des instructions DLX générées par votre compilateur pour l'expression suivante :

```
if (x => y) 1 else 2
```

On suppose que les valeurs des variables `x` et `y` sont respectivement aux positions 4 et 8 par rapport au pointeur de pile (SP). Le résultat doit être stocké dans le registre R1. Pour désigner les destinations de sauts, vous pouvez utiliser des noms. Par exemple :

```
    BEQ R0, else
else: ...
```