CMP1 – Construction des compilateurs
TYLA – Typologie des Langages

EPITA – Promo 2009 – **Tout document autorisé**

Avril 2007 (1h30)

Une copie synthétique, bien orthographiée, avec un affichage clair des résultats, sera toujours mieux notée qu'une autre demandant une quelconque forme d'effort de la part du correcteur. Une argumentation informelle mais convaincante, sera souvent suffisante.

1 Incontournables

Il n'est pas admissible d'échouer sur une des questions suivantes : **chacune induit une pénalité sur la note finale**.

1. Quelle est le type (de Chomsky) du langage engendré par $S \to aX \quad S \to b \quad X \to Sc$
2. Combien existe-t-il de mots de m lettres prises dans un alphabet de taille n ?
3. La récurrence à droite est-elle possible en LALR(1) ?
4. Qu'est-ce que Yacc ?

2 Typologie des Langages

1. Parmi les polymorphismes ci-dessous, lesquels sont disponibles en C ISO 1999 ?
   - polymorphisme paramétrique
   - polymorphisme de surcharge (également appelé polymorphisme *ad hoc*)
   - polymorphisme de coercition (*coercion* en anglais)
   - polymorphisme d'inclusion
2. Même question, mais en C++ ISO 2003.
3. Classer ces quatre types de polymorphismes évoqués dans les deux questions précédentes dans les cases du tableau ci-dessous (à recopier sur votre copie).

	résolution statique	résolution dynamique
concerne les types/classes		
concerne les fonctions/méthodes		

4. On rappelle que la *mise en ligne* du corps d'une fonction (*inlining*) est une optimisation consistant à remplacer un appel de fonction ou de méthode par le corps de celle-ci au site d'appel (en remplaçant les arguments formels par les arguments effectifs).
   En C++, il existe plusieurs situations dans lesquelles une fonction (ou méthode) ne peut être mise en ligne ; citez-en deux. (On ne s'attache pas ici à la qualité de l'optimisation, donc une réponse telle que « le compilateur refuse l'inlining car la fonction à mettre en ligne est trop grosse » n'est pas pertinente).
5. *(Culture générale)* Quel informaticien célèbre est décédé le 17 mars 2007 ? Citer une de ses réalisations. (Bien entendu, vous n'avez pas le droit au coup de fil à un ami pour cette question.)

3 Construction des Compilateurs

3.1 Mise en bouche

1. Qu'est-ce que le sucre syntaxique ?
2. Qu'est-ce que le désucrage ?

3.2 Cassons du sucre sur le dos du félin

Dans les questions suivantes, on se propose de traiter quelques-uns des aspects du désucrage des constructions ayant trait à l'objet dans le langage Leopard. On rappelle que le traitement des dites structures objets dans `1c` passe par une transformation (désucrage) du langage autorisant ces éléments, à un sous-ensemble de Leopard n'en disposant pas, qu'on appellera *Tiger*.

1. Citer les 5 phases d'une partie frontale d'un compilateur comme `1c`.
2. Où situer la phase de désucrage de « Leopard avec objets » vers « Leopard sans objets » (i.e. Tiger) ? Justifiez votre réponse.
3. Considérons le programme suivant :

```
let
 type C = class
 {
  var a := 42
  method a_get () : int = self.a
 }
 var c := new C
 var a := 51
in
 c.a := a;
 print_int (c.a_get ());
 print ("\n")
end
```

   Proposer une version de ce programme réécrite en Tiger (dépourvue de constructions syntaxiques spécifique à l'objet, donc). À ce stade de l'exercice, on ne demande pas de prendre en compte l'héritage de classe, le polymorphisme d'inclusion, et les méthodes polymorphes.

4. On souhaite maintenant traiter le mot-clef `extends` pour ajouter l'héritage de classe. On ne considère ici que l'héritage de *données*, et pas la relation de typage dite de *généralisation* (« est un »), ni l'héritage de comportements (méthodes). Comment procédez-vous ? Appliquez votre proposition au programme ci-dessous.

```
let
 type C = class
 {
  var a := 0
 }
 class D extends C
 {
  var b := 4 + 8 + 15 + 16 + 23 + 42
 }
 var c : C := new C
 var d : D := new D
in
end
```

5. On s'intéresse ensuite au polymorphisme d'inclusion (en laissant de côté la résolution dynamique de méthodes pour le moment). Dans le programme ci-dessous, quelle(s) opération(s) nécessite(nt) le polymorphisme d'inclusion, et pourquoi ? (Vous êtes invité à mentionner dans votre réponse la ou les lignes pertinente(s) du listing).

```
1  let
2   type X = class
3   {
4    var t := 42
5   }
6   type Y = class extends X
7   {
8    var u := 51
9   }
10  var x : X := new X
11  var y : X := new Y
12 in
13  x := y;
14  x := new Y
15 end
```

   Proposer une solution de désucrage. On ne demande pas de traduire l'intégralité de l'exemple précédent en Tiger ; une argumentation informelle (mais certainement pas vague) sera suffisante.

6. Les appels de méthodes polymorphes en Leopard n'utilisent pas des tables de méthodes virtuelles comme en C++. Au lieu de cela, chaque objet comporte une étiquette (un numéro) indiquant quel est son *type dynamique*. Dans ces conditions, quelle approche peut-on employer pour résoudre un appel de méthode polymorphe ?

7. Proposer une solution de désucrage d'un appel de méthode polymorphe du langage Leopard. C'est l'idée générale de la solution qui est attendue ici, et non la traduction d'un exemple Leopard vers Tiger.

3.3 Vers le Leopard et au delà

1. On souhaite étendre la syntaxe de Leopard pour lui ajouter des constructeurs. Proposer des extensions du langage permettant de définir un constructeur dans une classe, ainsi que d'utiliser celui-ci (les réponses informelles seront tolérées, tant qu'elle sont intelligibles et non ambigües). Donner un exemple illustrant ces ajouts syntaxiques.

2. On souhaite désormais pouvoir ajouter plusieurs constructeurs dans une même classe. Que faut-il ajouter au langage pour que ça fonctionne ? Donner des exemples d'utilisations. Quels sont les changements à appliquer au compilateur ? Soyez précis, et indiquez quelles parties sont affectées par cette fonctionnalité.