Correction du partiel CMP1

EPITA – AppIng1 promotion 2014
Tous documents (notes de cours, polycopiés, livres) autorisés
Calculatrices, ordinateurs, tablettes et téléphones interdits.

Juin 2012 (1h30)

Correction: Le sujet et sa correction ont été écrits par Roland Levillain.

Best-of: Le *best of* est tiré des copies des étudiants ; les fautes de français aussi, le cas échéant.

Lisez bien les questions, chaque mot est important. Écrivez court, juste et bien ; servez vous d'un brouillon. Une argumentation informelle mais convaincante est souvent suffisante. Gérez votre temps, ne restez pas bloqué sur les questions les plus difficiles. Une lecture préalable du sujet est recommandée.

Cette correction contient 11 pages. Les pages 1–4 contiennent l'épreuve et son corrigé. Ce document comporte en pages 8–9 une enquête (facultative) sur le cours et le projet Tiger (y répondre sur la feuille de QCM qui vous est fournie). Enfin, les pages 10–11 sont dévolues aux annexes.

1 Échauffement

1. Combien de nombres entiers différents peut-on représenter avec 42 bits ?

   Correction: 2^{42}.

   Best-of:
   – $2^{41} - 1$
   – $2^{43} - 1$
   – 10^{42}
   – $\sum_{n=0}^{41} 2^n + 1$
   – Avec un octet on peut coder $2^8 = 256$ valeurs. 1 octet = 8 bits. Alors $\frac{42\text{ (bits)}}{8\text{ (bits)}} = 7$ (octets). Soit 7×256 valeurs d'entiers possibles sur 42 bits.

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

   Correction: Il s'agit de constructions syntaxiques supplémentaires proposées par un langage n'apportant rien de plus au niveau de sa syntaxe abstraite, mais permettant des variantes d'écriture. Le sucre syntaxique apporte souvent simplicité, lisibilité, raccourcis d'écriture, confort, etc. Il peut permettre l'expression de nouveaux concepts (la surcharge de fonctions peut être considérée comme telle, par exemple).

   Best-of:
   – (. . .) constructions synthaxiques (. . .)
   – Il [le sucre syntaxique] permet au compilateur de mieux comprendre le code.

3. À quoi sert le *design pattern* Visiteur (de façon générale, pas juste dans le projet Tiger) ?

   Correction: Un Visiteur est une technique qui permet de séparer un algorithme d'une structure de données sur laquelle il agit. Plus concrètement, un visiteur permet d'ajouter de nouvelles opérations à des structures de données de façon non intrusive (c'est-à-dire, sans modifier ces structures). Le design pattern Visiteur est implémenté à l'aide d'un « double aiguillage » (*double dispatch*).

4. Pourquoi l'expression C suivante, considérée en dehors de tout contexte, est ambiguë ?

```
foo * bar;
```

   Correction: Parce qu'elle peut représenter deux instructions différentes, correspondant à deux interprétations différentes. Il peut en effet s'agir
   – du produit des facteurs (variables) `foo` et `bar` ;
   – ou encore de la déclaration d'un pointeur nommé `bar` sur une variable de type `foo`.
   Sans *contexte* (ici, la nature de `foo`, qui peut être une variable ou un type), il est impossible de trancher. Cet exemple est une illustration du fait que la grammaire du C n'est *pas* hors-contexte.

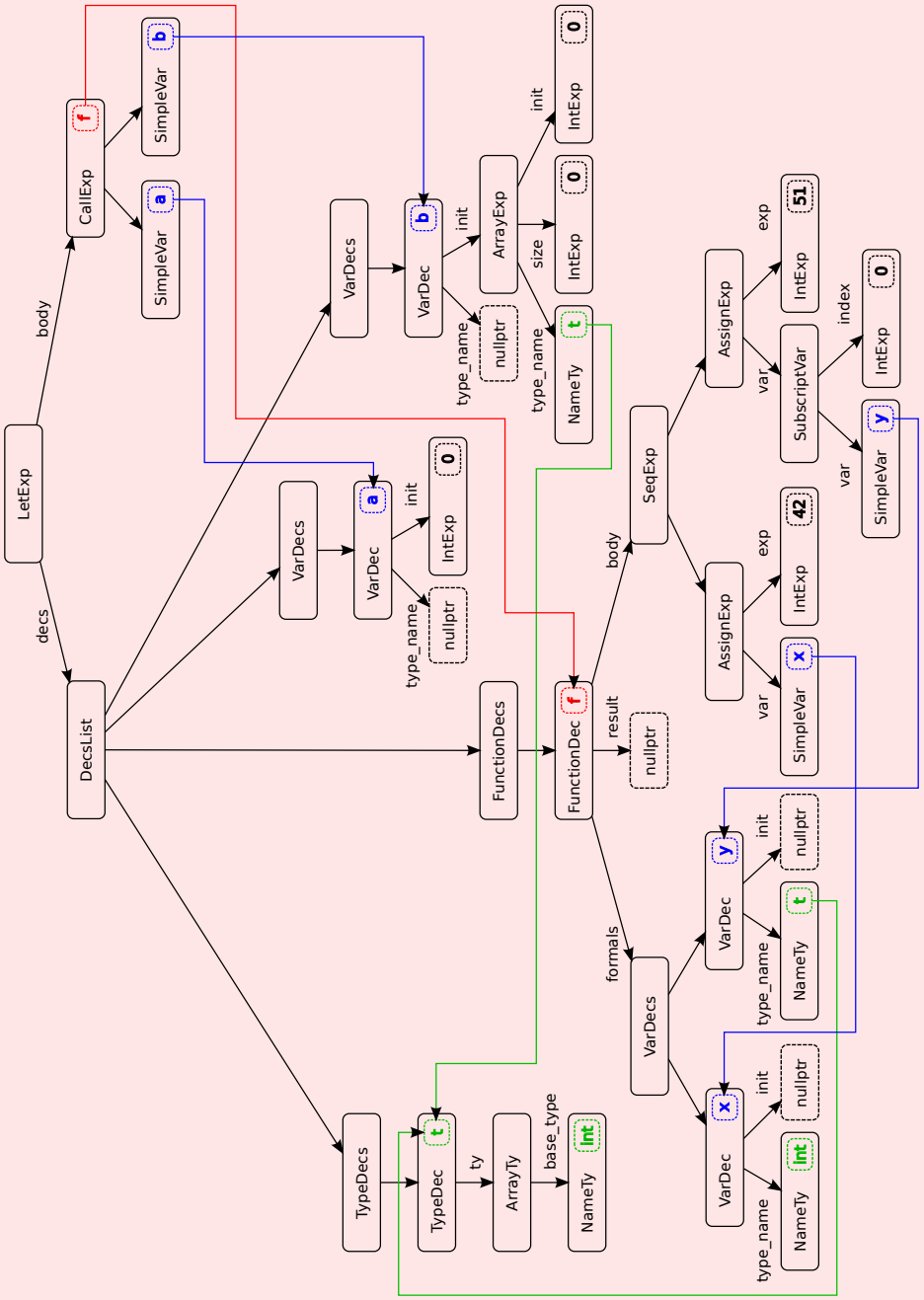
2 Fonction féline et passage d'arguments

Soit le programme Tiger suivant :

```
1  let
2    type t = array of int
3    function f (x : int,  y : t) =
4    (
5      x := 42;
6      y[0] := 51
7    )
8    var a := 0
9    var b := t[1] of 1
10 in
11   f (a, b)
12 end
```

1. Représentez l'arbre de syntaxe abstraite de ce programme sous forme d'un diagramme d'instances (objets) UML. Aidez-vous de l'annexe A à la fin de ce document.

Correction:



2. Sur le schéma de la question précédente, indiquez les liens entre sites d'utilisation et sites de définition à l'aide de flèches partant des premiers et allant vers les seconds (de préférence en utilisant une autre couleur de stylo).

   Correction: Voir le diagramme de la réponse précédente.

3. Quel(s) type(s) de passage(s) d'argument(s) est (sont) utilisé(s) en Tiger ?

   Correction: Le passage par valeur (copie) pour les valeurs de types atomiques/-primitifs (`int`, `string` et `nil`) et par référence pour les valeurs d'autres types (tableaux, enregistrements, objets).

Best-of:
[Passages d'arguments] implicite et explicite.

4. Quelles sont les valeurs de 'a' et 'b[0]' au retour de l'appel de la fonction f à la ligne 11 ?

   Correction: Après l'appel de f, 'a' et 'b[0]' valent respectivement 0 et 51.

5. Les valeur de 'a' et 'b[0]' peuvent avoir changé après cet appel à f.

   (a) Expliquez pourquoi 'a' *a* ou *n'a pas* changé de valeur après cet appel de fonction.

   Correction: Le premier argument de f lui est passé par valeur, car int est un type primitif. À l'intérieur de f, 'x' est donc une *copie* de l'argument originel ('a'). Ce dernier n'est donc pas modifié par f.

   (b) Même question pour 'b[0]'.

   Correction: Le second argument de f est quant à lui passé par référence (car c'est un tableau). À l'intérieur de f, 'y' est donc un *alias* de l'argument originel 'b'. Par suite, y[0] et b[0] désignent le même emplacement mémoire. En conséquence, l'instruction 'y[0] := 51' de f modifie bien b[0].

3 *Les liaisons sûres* (de Choderlos de la Closure)

1. Citez les 5 phases d'une partie frontale (*front-end*) d'un compilateur comme tc.

   Correction: (i) scanner, (ii) parser et construction de l'AST, (iii) liaison des noms, (iv) typage, (v) traduction vers une représentation intermédiaire.

   Best-of:
   – précompilation (.o)
   – gestion des liens (librairie (*arg !*), ...)

2. Expliquer la différence entre liaison statique et liaison dynamique de noms. Quels sont les avantages de l'une et de l'autre ?

   Correction: Liaison statique : effectuée à la compilation ; liaison dynamique : effectuée à l'exécution. La liaison statique est plus sûre, permet le typage statique ainsi que des optimisations à la compilation ; la liaison dynamique apporte une certaine souplesse nécessaire pour la réalisation de services tels que le chargement dynamique de modules, l'introspection et la réflexivité.

   Best-of:
   – Les liens statiques, à la différence des liens dynamiques, sont beaucoup plus coûteux.

3. Considérez le bout de code Tiger suivant :

```
1 let
2   type u = t
3   type t = int
4   function f () : u = 42
5 in
6   f ()
7 end
```

   (a) Les lignes de code 2 et 3 sont-elles valides ?

   Correction: Oui.

(b) Si oui, rappelez les règles du langage correspondantes ; si non, expliquer l'erreur et donnez une version corrigée de ces lignes.

Correction: Les déclarations des types `u` et `t` se suivent : elles forment un bloc (*chunk*) où les références vers l'avant sont permises. Ainsi, il est ici licite de définir `u` comme un alias de `t`, bien que celui-ci ne soit définit qu'après.

4. On rappelle qu'une variable qui s'échappe est une variable qui ne pourra être placée dans un registre *in fine*. Elle devra donc être stockée en mémoire, sur la pile. Donnez deux situations dans lesquelles une variable doit être placée sur la pile (que ce soit en Tiger ou en C++).

Correction: Au choix parmi les réponses suivantes :
- la variable est passée par référence ou encore son adresse est prise à un moment donné (avec l'opérateur '`&`' en C : il est donc nécessaire qu'elle ait une adresse mémoire ;
- une fonction imbriquée dans la fonction où est définie la variable accède à cette dernière (autrement dit : cette variable *s'échappe* ; elle est *non locale*) ;
- la variable est passé à travers une liste variable d'arguments à une fonction « variadique » (comme `printf` en C) : ce mécanisme repose sur des manipulations effectuées sur la pile ;
- la variable est trop grosse pour rentrer dans un registre (par exemple, un enregistrement) ;
- la variable est un tableau, pour lequel l'accès aux cellules nécessite de l'arithmétique sur les adresses mémoire ;
- le registre contenant la variable est réquisitionné pour une tâche spécifique (comme le passage d'arguments) : pour libérer ce registre, une solution peut être de placer la variable en mémoire (ou dans un autre registre) ;
- il y a tant de variables locales et de temporaires qu'elles ne peuvent toutes tenir dans les registres du processeur : il est donc nécessaire de « verser » (*spill*) certaines de ces variables dans le cadre de pile (en mémoire) pour résoudre ce problème d'allocation de registres.

Best-of:
- Lorsqu'on fait un `malloc` on place la variable directement sur la pile.

5. Tiger et le GNU C [1] permettent la déclaration de fonctions imbriquées ainsi que l'utilisation de variables non locales [2]. Rappelez ce qu'est une variable non locale.

Correction: Une variable est dite non locale si elle est définie dans une fonction `f` et utilisée (au moins une fois) dans une fonction `g` imbriquée (directement ou indirectement) dans `f`.
Attention à ne pas confondre fonctions et portées : une variable définie dans une portée s_1 d'une fonction et utilisée dans une portée s_2 (« incluse » dans s_1) de cette même fonction ne s'échappe *pas*, puisqu'il s'agit de la même fonction.

Best-of:
- C'est à dire ils ne peuvent pas avoir de variable global comme une classe.

6. Dans le programme GNU C ci-dessous la variable '`v`' est non locale.

```
int f (void)
{
  int v = 42;
```

1. On appelle GNU C la variante du langage C acceptée par le front end C de GCC pourvue d'extensions du langage.
2. Note : les fonctions imbriquées ne sont pas autorisées par la norme C ISO.

```
  int g (void)
  {
    int w = 51;
    return v + w;
  }

  return g ();
}

int main (void)
{
  return f();
}
```

Il est cependant possible de la rendre à nouveau locale (« non non locale ») en déplaçant la définition de `g` hors de `f` et en faisant les ajustements nécessaires. Réécrivez le programme précédent après application de cette transformation.

Correction:

```
int g (int* v)
{
  int w = 51;
  return *v + w;
}

int f (void)
{
  int v = 42;
  return g (&v);
}

int main (void)
{
  return f();
}
```

Cette solution est un exemple de *lambda lifting* : la variable non locale 'v' est passée par « référence » (en fait, par pointeur) à la fonction imbriquée.

7. La réponse à la question précédente utilise un trait du C absent de Tiger. De fait, il ne serait pas possible d'appliquer directement une transformation similaire en Tiger. Quelle est cette fonctionnalité du C manquant en Tiger ?

   Correction: L'utilisation d'un pointeur pour simuler un passage par référence (passage par pointeur/adresse). Ni les références, ni les pointeurs n'existent en Tiger.

8. **Bonus.** Comment simuler cette fonctionnalité en Tiger ?

Correction: Correction : en remplaçant la variable non locale par un tableau, un enregistrement ou un objet, qui sont tous manipulé par leur adresse : lors du passage d'un tel élément à une fonction, c'est son adresse qui est copiée, par son contenu.
Voici un exemple d'adaptation en Tiger du code C précédent :

```
let
  type int_ref = { val : int }

  function g (v : int_ref) : int =
    let var w := 51 in
      v.val + w
    end

  function f () : int =
    let
      var v := int_ref { val = 42 }
    in
      g (v)
    end
in
  f()
end
```

Le type `int_ref` est un enregistrement qui représente une référence vers un entier. Lorsqu'il est passé comme argument, la valeur référencée n'est pas copiée, ce qui permet de simuler un passage par référence. Pour information, cette technique est inspirée du langage Objective Caml, qui implémente les références sous la forme d'enregistrements à champ unique :

```
# let a = ref 42;;
val a : int ref = { contents = 42 }
```

4 À propos de ce cours

Pour terminer cette épreuve, nous vous invitons à répondre à un petit questionnaire. Les renseignements ci-dessous ne seront bien entendu pas utilisés pour noter votre copie. Ils ne sont pas anonymes, car nous souhaitons pouvoir confronter réponses et notes. En échange, quelques points seront attribués pour avoir répondu. Merci d'avance.

Sauf indication contraire, vous pouvez cocher plusieurs réponses par question. Répondez sur la feuille de QCM. N'y passez pas plus de dix minutes.

Cette épreuve

Q.1 Sans compter le temps mis pour remplir ce questionnaire, combien de temps ce partiel vous a-t-il demandé (si vous avez terminé dans les temps), ou combien vous aurait-il demandé (si vous aviez eu un peu plus de temps pour terminer) ?

a. Moins de 30 minutes.
b. Entre 30 et 60 minutes.
c. Entre 60 et 90 minutes.
d. Entre 90 et 120 minutes.
e. Plus de 120 minutes.

Q.2 Ce partiel vous a paru

a. Trop difficile.
b. Assez difficile.
c. D'une difficulté normale.
d. Assez facile.
e. Trop facile.

Le cours

Q.3 Quelle a été votre implication dans les cours CMP1 ?

a. Rien.
b. Bachotage récent.
c. Relu les notes entre chaque cours.
d. Fait les annales.
e. Lu d'autres sources.

Q.4 Ce cours

a. Est incompréhensible et j'ai rapidement abandonné.
b. Est difficile à suivre mais j'essaie.
c. Est facile à suivre une fois qu'on a compris le truc.
d. Est trop élémentaire.

Q.5 Ce cours

a. Ne m'a donné aucune satisfaction.
b. N'a aucun intérêt dans ma formation.
c. Est une agréable curiosité.
d. Est nécessaire mais pas intéressant.
e. Je le recommande.

Q.6 La charge générale du cours en sus de la présence en amphi (relecture de notes, compréhension, recherches supplémentaires, etc.) est

a. Telle que je n'ai pas pu suivre du tout.
b. Lourde (plusieurs heures de travail par semaine).
c. Supportable (environ une heure par semaine).
d. Légère (quelques minutes par semaine).

Les formateurs

Q.7 L'enseignant

a. N'est pas pédagogue.
b. Parle à des étudiants qui sont au dessus de mon niveau.
c. Me parle.
d. Se répète vraiment trop.
e. Se contente de trop simple et devrait pousser le niveau vers le haut.

Q.8 Les assistants

a. Ne sont pas pédagogues.

b. Parlent à des étudiants qui sont au dessus de mon niveau.
c. M'ont aidé à avancer dans le projet.
d. Ont résolu certains de mes gros problèmes, mais ne m'ont pas expliqué comment ils avaient fait.
e. Pourraient viser plus haut et enseigner des notions supplémentaires.

Le projet Tiger

Q.9 Vous avez contribué au développement du compilateur de votre groupe (une seule réponse attendue) :

a. Presque jamais.
b. Moins que les autres.
c. Équitablement avec vos pairs.
d. Plus que les autres.
e. Pratiquement seul.

Q.10 La charge générale du projet Tiger est

a. Telle que je n'ai pas pu suivre du tout.
b. Lourde (plusieurs jours de travail par semaine).
c. Supportable (plusieurs heures par semaine).
d. Légère (une ou deux heures par semaine).
e. J'ai été dispensé du projet.

Q.11 Y a-t-il de la triche dans le projet Tiger ? (Une seule réponse attendue.)

a. Pas à votre connaissance.
b. Vous connaissez un ou deux groupes concernés.
c. Quelques groupes.
d. Dans la plupart des groupes.
e. Dans tous les groupes.

Questions 12-18 Le projet Tiger vous a-t-il bien formé aux sujets suivants ? Répondre selon la grille qui suit. (Une seule réponse attendue par question.)

a	**b**	**c**	**d**	**e**
Pas du tout	Trop peu	Correctement	Bien	Très bien

Q.12 C++.
Q.13 modélisation orientée objet et *design patterns*.
Q.14 anglais technique.
Q.15 compréhension du fonctionnement des ordinateurs.
Q.16 compréhension du fonctionnement des langages de programmation.
Q.17 travail collaboratif.
Q.18 outils de développement (contrôle de version, systèmes de construction, débogueurs, générateurs de code, etc.)

Questions 19-23 Comment furent les étapes du projet ? Répondre selon la grille suivante. (Une seule réponse attendue par question ; ne pas répondre pour les étapes que vous n'avez pas faites.)

a	**b**	**c**	**d**	**e**
Trop facile.	Facile.	Nickel.	Difficile.	Trop difficile.

Q.19 Rush `.tig` : mini-projet en Tiger (Logomatig).
Q.20 `TC-0`, Scanner & Parser.
Q.21 `TC-1`, Scanner & Parser, Tâches, Autotools.
Q.22 `TC-2`, Construction de l'AST et pretty-printer.
Q.23 `TC-3`, Liaison des noms et renommage.

A Classes de la syntaxe abstraite du langage Tiger

Voici une copie du fichier 'src/ast/README' fourni avec le code de tc.

```
Tiger Abstract Syntax Tree nodes with their principal members.
Incomplete classes are tagged with a '*'.

/Ast/               (Location location)
  /Dec/             (symbol name)
    FunctionDec     (VarDecs formals, NameTy result, Exp body)
      MethodDec     ()
    TypeDec         (Ty ty)
    VarDec          (NameTy type_name, Exp init)

  /Exp/             ()
*   /Var/
      CastVar       (Var var, Ty ty)
*     FieldVar
      SimpleVar     (symbol name)
      SubscriptVar  (Var var, Exp index)

*   ArrayExp
*   AssignExp
*   BreakExp
*   CallExp
*     MethodCallExp
    CastExp         (Exp exp, Ty ty)
    ForExp          (VarDec vardec, Exp hi, Exp body)
*   IfExp
    IntExp          (int value)
*   LetExp
    MetavarExp      ()
    NilExp          ()
*   ObjectExp
    OpExp           (Exp left, Oper oper, Exp right)
*   RecordExp
*   SeqExp
*   StringExp
    WhileExp        (Exp test, Exp body)

  /Ty/              ()
    ArrayTy         (NameTy base_type)
    ClassTy         (NameTy super, DecsList decs)
    NameTy          (symbol name)
*   RecordTy

  /Decs/
    FunctionDecs    (std::list<FunctionDec*>)
    MethodDecs      (std::list<MethodDec*>)
    TypeDecs        (std::list<TypeDec*>)
    VarDecs         (std::list<VarDec*>)
```

```
/Ast/
  DecsList          (std::list<Decs*> decs)

  Field             (symbol name, NameTy type_name)

  FieldInit         (symbol name, Exp init)



Some of these classes also inherit from other classes.

/Escapable/
  VarDec            (NameTy type_name, Exp init)

/Metavariable/      (unsigned id)
  MetavarExp        ()

/Typable/
  /Dec/             (symbol name)
  /Exp/             ()
  /Ty/              ()

/TypeConstructor/
  /Ty/              ()
  FunctionDec       (VarDecs formals, NameTy result, Exp body)
  TypeDec           (Ty ty)
```