CMP1 – Construction des compilateurs
TYLA – Typologie des Langages

EPITA – Promo 2010
Tous documents (notes de cours, polycopiés, livres) autorisés

Janvier 2008 (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.

Cette épreuve est longue, le but est d'en faire un maximum, le mieux possible. Avant toute chose, une lecture complète du sujet est recommandée.

1 Incontournables

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

1. La surcharge de fonctions en C++ est un mécanisme qui dépend des types à l'exécution. Vrai ou faux ?
2. L'utilisation de fonctions virtuelles en C++ est incompatible avec la compilation séparée. Vrai ou faux ?
3. Parmi ces propositions, laquelle (lesquelles) est (sont) un pré-requis pour le typage statique fort dans un langage orienté objet ?
   (a) une liaison des noms résolue à la compilation ;
   (b) l'absence d'instructions de transtypage (*cast*) ;
   (c) l'absence de méthodes polymorphes abstraites (appelées fonctions membres virtuelles pures en C++) ;
   (d) la présence automatique d'une surclasse au sommet de toute hiérarchie de classes (`Object`, par exemple).
4. Quelle est le type (de Chomsky) du langage engendré par

$$S \to X|Y \qquad X \to Zp \qquad Y \to o \qquad Z \to pS$$

2 Let's Target C++ !

On se propose dans cette partie d'ajouter un *back-end* à notre compilateur Tiger ciblant le langage C++. Nous souhaitons donc pouvoir via cette nouvelle cible traduire un programme Tiger en un programme C++ équivalent (c'est-à-dire, dont les comportements sont similaires).

Notez que cette approche est devenue classique ; de nombreux compilateurs ciblent un autre langage, qui traduit ensuite le programme vers un langage d'assemblage (par exemple). On peut citer SmartEiffel (qui cible le C et le *bytecode* Java), GHC (compilateur Haskell, qui cible le C), le compilateur Stratego (qui cible le C), etc.

1. Pour quelles raisons peut-on souhaiter bénéficier de ce nouveau back-end ? Autrement dit, à quoi cela peut-il servir ? Citez des cas d'utilisation possibles.
2. Citer les 5 phases d'une partie frontale (*front-end*) d'un compilateur comme `tc`.
3. Où situer la phase de traduction vers le C++ dans le compilateur ? Justifiez votre réponse.
4. Y'a-t-il des modifications à apporter aux phases existantes du front-end ? Si oui, citez-les ; si non, expliquez pourquoi.
5. Le langage Tiger permet de définir des variables locales via la construction `let`. Comment peut-on gérer cela lors de la traduction en C++ ?
6. Les fonctions imbriquées n'existent pas en ISO C++ ; comment peut-on les traduire ?
7. Le mot clef `return` n'existe pas en Tiger, et il va pourtant falloir le faire apparaître dans le code C++ produit. Quelle règle (algorithme) faut-il suivre pour ce faire lors de la traduction du code Tiger vers C++ ?
8. Intéressons-nous aux types. Pour chaque morceau de code Tiger ci-dessous, donner une proposition de traduction C++ (répondre sur votre copie) :

```
type t = int
var t := 42
```

```
var string := "Je clair, Luc, ne pas ?"
```

```
type names_t = array of string
```

```
type point2d = { row : int, col : int }
var p := point2d { row = 3, col = 14 }
```

```
type int_list = { head : int, tail : int_list }
var l := int_list { head = 51, tail =
        int_list { head = 2501, tail =
          int_list { head = 2097, tail = nil }}}
```

```
type list1 = { head : int, tail : list2 }
type list2 = { head : int, tail : list1 }
```

9. Parmi les propositions que vous avez écrites en réponse à la question précédente, en est-il qui pose(nt) des problèmes du point de vue de la gestion de la mémoire (allocation et libération) ?
10. En Tiger, les opérateurs de comparaison (=, <>, <, >, <=, >=) sont surchargés pour tout ou partie des types du langage. Comment faut-il les traduire en C++ ?
11. Traduisez le code Tiger ci-dessous en C++. (*Hint* : il est probablement sage de faire un essai sur un brouillon au préalable.)

```
let
  type bool = int
  var true  : bool := 1
  var false : bool := 2

  /* Of course, these won't work on negative integers,
     but we don't care. */
  function is_even (x : int) : bool =
    if x = 0 then true else is_odd (x − 1)
  function is_odd (x : int) : bool =
    if x = 1 then true else is_even (x − 1)
in
```

```
if is_even (3) then
  is_odd (51) /* Yes, and no. */
else
  42
end
```

12. Même exercice.

```
let
  type Exp = class {
    /* abstract */ method eval () : int = 0 /* Dummy value. */
  }
  type Num = class extends Exp {
    var val := 0
    method eval () : int = self.val
  }
  type Bin = class extends Exp {
    var lhs : Exp := new Num
    var rhs : Exp := new Num
    method eval () : int = self.rhs.eval () + self.lhs.eval ()
}
  var b1 := new Num
  var b2 := new Num
  var b := new Bin
in
  b1.val := 42;
  b2.val := 51;
  b.lhs := b1;
  b.rhs := b2;
  print_int (b.eval ())
end
```

13. **Question Banco** Après avoir traduit un programme Tiger vers son équivalent C++, puis compilé ce dernier pour un microprocesseur de l'architecture Intel 64, on souhaite l'exécuter dans un debugger comme GDB. Cependant, les messages d'erreur font référence au code C++ généré, et pas au code Tiger originel. Comment peut-on améliorer cela ?
14. **Question Super Banco** On souhaite pouvoir utiliser une bibliothèque C++ existante depuis un programme Tiger traduit vers C++ grâce à notre nouveau back-end. Que faut-il ajouter à notre compilateur ?

3 The T-Files

En guise de dessert, nous vous proposons un petit exercice consistant à augmenter les fonctionnalités des entrées-sorties (I/O) en Tiger. Le langage actuel, tel que défini dans le Manuel de Référence du Compilateur Tiger, est assez pauvre en routines d'entrées-sorties, et ne comporte aucune abstraction (cf. annexe A, page 6).

On décide donc d'augmenter le prélude du langage avec les déclarations suivantes [1] :

```
/* The type of a stream. */
type stream

/* The standard streams. */
var stdin : stream
var stdout : stream
var stderr : stream

/* Open file with name PATH and obtain a stream. MODE can be "r", "w"
  or "a" for reading, writing or appending. The file will be created
  if it doesn't exist when opened for writing or appending; it will
  be truncated when opened for writing. Add a 'b' to the mode for
  binary files. Add a "+" to the mode to allow simultaneous reading
  and writing. If the file cannot be open or the mode is ill-formed,
  raise a runtime error: "fopen: invalid path." or "fopen: invalid
  mode." */
primitive fopen (path : string, mode : string) : stream
/* Close stream STREAM. If the file handle is invalid, raise a
  runtime error: "fopen: invalid stream." */
primitive fclose (stream : stream)

/* Print STRING on STREAM. */
primitive fprint (string: string, stream : stream)
/* Output INT in its decimal canonical form (equivalent to "%d" for
  fprintf()). */
primitive fprint_int (int: int, stream : stream)
/* Flush the buffer of STREAM. */
primitive fflush (stream : stream)
/* Read a character from STREAM. Return an empty string on an end of
  file. */
primitive fgetchar (stream : stream) : string
```

Étant donné que seul le front-end de `tc` a été vu en cours pour le moment, les réponses non formelles sont acceptées. L'objectif est de réfléchir et de comprendre quels sont les impacts de cette extension.

1. Que faut-il changer dans le front-end de `tc` (i.e., les étapes jusqu'à la représentation intermédiaire dans le langage Tree) ?
2. De même, faut-il enrichir la définition du langage Tree, utilisé pour la représentation intermédiaire par `tc` ? Si oui, quels sont les changements nécessaires ? Si non, pourquoi ?
3. À votre avis, faut-il apporter des changements dans le back-end de `tc` (la partie qui va de la représentation intermédiaire au langage d'assemblage, par exemple MIPS) ? En particulier, est-ce que ces parties du back-end seront affectées :
   – l'allocation de registres ;

1. Les plus attentifs auront remarqué que le code de cette extension du prélude n'est pas du Tiger valide : le langage ne permet pas de déclarer des types et des variables sans leur donner une valeur. Ici, c'est le runtime qui fournira les implémentations absentes. Nous considérerons pour cet exercice que cette syntaxe est valide.

– le runtime Tiger (support du langage à l'exécution) ;
– l'émission de code en langage d'assemblage.

4. **Question Mega Banco** Cette extension est plutôt simpliste. En particulier
   – elle est assez pauvre en routines ;
   – la gestion d'erreurs est brutale (sortie avec une erreur de runtime) ;
   – les flux d'entrées et de sorties ne sont pas distingués (il n'y a qu'un seul type `stream`).
   Proposez des idées pour améliorer la proposition initiale.

4 À propos de ce cours

Je copie ~~un peu~~ complètement sur Akim, et procède à un petit debriefing discrétisé sur quatre questions.

Bien entendu je m'engage à ne pas tenir compte des renseignements ci-dessous pour noter votre copie. Ils ne sont pas anonymes, car je suis curieux de confronter vos réponses à votre note. En échange, quelques points seront attribués pour avoir répondu. Merci d'avance.

Vous pouvez cocher plusieurs réponses par question. Répondez sur les feuilles de QCM qui vous sont remises.

1. Travail personnel
   a Rien
   b Bachotage récent
   c Relu les notes entre chaque cours
   d Fait les annales
   e Lu d'autres sources
2. 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
3. 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
4. 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

Bonne année et meilleurs vœux !

Annexes

A Bibliothèque Standard du langage Tiger

Le listing ci-dessous contient les déclarations (annotées) du prélude actuel du compilateur Tiger (`prelude.tih`).

```
/* Print STRING on the standard output. */
primitive print (string: string)
/* Note: this is an EPITA extension. Same as print(), but the output
   is written to the standard error. */
primitive print_err (string: string)
/* Note: this is an EPITA extension. Output INT in its decimal
   canonical form (equivalent to "%d" for printf()). */
primitive print_int (int: int)
/* Flush the output buffer. */
primitive flush ()
/* Read a character on input. Return an empty string on an end of
   file. */
primitive getchar () : string

/* Return the ascii code of the first character in STRING and -1 if
   the given string is empty. */
primitive ord (string: string) : int
/* Return the one character long string containing the character which
   code is CODE. If CODE does not belong to the range [0..255], raise
   a runtime error: "chr: character out of range." */
primitive chr (code: int) : string
/* Return the size in characters of the STRING. */
primitive size (string: string) : int

/* Note: this is an EPITA extension. Return 1 if the strings A and B
   are equal, 0 otherwise. Often faster than strcmp() to test string
   equality. */
primitive streq (s1: string, s2: string) : int
/* Note: this is an EPITA extension. Compare the strings A and B:
   return -1 if A < B, 0 if equal, and 1 otherwise. */
primitive strcmp (s1: string, s2: string) : int
/* Return a string composed of the characters of STRING starting at
   the FIRST character (0 being the origin), and composed of LENGTH
   characters (i.e., up to and including the character
   FIRST + LENGTH). */
primitive substring (string: string, start: int, length: int) : string
/* Concatenate FIRST and SECOND. */
primitive concat (first : string, second: string) : string

/* Return 1 if BOOLEAN = 1, else return 0. */
primitive not (boolean : int) : int

/* Exit the program with exit code STATUS. */
primitive exit (status: int)
```