CMP1 – Construction des compilateurs – S2

EPITA – Promo 2015

Tous documents (notes de cours, polycopiés, livres) autorisés
Calculatrices, ordinateurs, tablettes et téléphones interdits.

Avril 2013 (1h00)

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.

Ce sujet contient 9 pages. Les pages 1–2 contiennent l'épreuve elle-même. Ce document comporte en pages 3–5 une enquête (facultative) sur le cours et le projet Tiger (y répondre sur la feuille de QCM qui vous est fournie). Enfin, les pages 6–9 sont dévolues aux annexes.

1 Const

À la différence du C++, Tiger est dépourvu de mot-clef `const`. On se propose dans cet exercice de l'ajouter au langage et de passer en revue les différentes parties de notre compilateur afin de déterminer quels sont les aménagements nécessaires.

1.1 Préliminaires

1. À quoi sert `const` en C++ ? (Il y a plusieurs réponses possibles, une seule vous est demandée.)
2. Parmi les lignes suivantes, quelles sont celles qui sont valides en C++ ? Justifiez votre réponse.
   (a) `int i = 42; const int* const p = &i;`
   (b) `int i = 42; const int const * p = &i;`
   (c) `int i = 42; int const * p = &i;`
   (d) `int i = 42; int const * const p = &i;`
3. Même question.
   (a) `int i = 42; const int& const p = i;`
   (b) `int i = 42; const int const & p = i;`
   (c) `int i = 42; int const & p = i;`
   (d) `int i = 42; int const & const p = i;`
4. Soit la classe suivante :

```
struct C
{
  void m1 () {}        // (0).
  void m1 () const {}  // (1).
  void m2 () {}        // (2).
  void m3 () const {}  // (3).
};
```

Pour chacun des cas suivants, indiquer quelle méthode est appelée (0, 1, 2 ou 3) ou s'il y a une erreur de compilation, le cas échéant.

(a) `C o; o.m1();`

(b) `const C o = C(); o.m1();`

(c) `const C o = C(); o.m2();`

(d) `C o; o.m3();`

(e) `const void* o = new C(); o->m1();`

(f) `const C o; C(o).m1();`

1.2 Partie frontale (*front end*)

Cette partie s'intéresse aux modifications à apporter au front end Tiger pour prendre en compte `const`. Nous ne considérerons ce nouveau mot-clef que comme qualificatif de types (le cas des méthodes « constantes » n'est pas à traiter).

1. **Spécifications lexicales.** Que faut-il ajouter aux spécifications lexicales du langage pour supporter `const` ?
2. **Scanner.** Comment étendre le scanner, c'est-à-dire, que faut-il ajouter/modifier dans le fichier '`scantiger.ll`' pour supporter ces nouvelles spécifications lexicales ?
3. **Spécifications syntaxiques.** Que faut-il ajouter aux spécifications syntaxiques du langage pour supporter `const` ?
4. **Syntaxe abstraite.** Faut-il effectuer des modifications ou des ajouts dans la syntaxe abstraite du langage pour supporter `const` ? Si oui, lesquelles ?
5. **Parser.** Comment étendre le parser, c'est-à-dire, que faut-il ajouter/modifier dans le fichier '`parsetiger.yy`' pour supporter ces nouvelles spécifications syntaxiques ?
6. **Liaison des noms.** Que faut-il changer dans le `Binder` vis-à-vis de `const` ?
7. **Vérification des types.** Selon vous, le support de `const` requiert-il des changements au niveau de la vérification des types du compilateur ? Justifiez votre réponse.
8. **Représentation intermédiaire et traduction de Tiger vers Tree.** Y a-t-il des changements à apporter au langage intermédiaire Tree ou au visiteur chargé de la traduction vers la représentation intermédiaire ? Justifiez votre réponse.

2 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 Pas du tout	**b** Trop peu	**c** Correctement	**d** Bien	**e** 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 19-29 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 Trop facile. **b** Facile. **c** Nickel. **d** Difficile. **e** Trop difficile.

Q.19 Rush `.tig` : mini-projet à écrire en Tiger.

Q.20 `PTHL/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.

Q.24 `TC-4`, Typage et calcul des échappements.

Q.25 `TC-D (option)`, Désucrage (boucles `for`, comparaisons de chaînes de caractères).

Q.26 `TC-I (option)`, Mise en ligne du corps des fonctions.

Q.27 `TC-B (option)`, Vérification dynamique des bornes de tableaux.

Q.28 `TC-A (option)`, Surcharge des fonctions.

Q.29 `TC-O (option)`, Désucrage des constructions objets.

A Grammaire du langage Tiger

Cette grammaire utilise le formalisme Extended Backus Naur Form (EBNF), où '[' et ']' sont utilisés pour représenter zéro (0) or une (1) occurrence du terme encadré, tandis que '{' et '}' désignent un nombre arbitraire de répétitions (y compris zéro).

```
⟨program⟩ ::=
    ⟨exp⟩
  | ⟨decs⟩

⟨exp⟩ ::=
  # Literals.
    "nil"
  | integer
  | string

  # Array and record creations.
  | ⟨type-id⟩ "[" ⟨exp⟩ "]" "of" ⟨exp⟩
  | ⟨type-id⟩ "{" [ id "=" ⟨exp⟩ { "," id "=" ⟨exp⟩ } ] "}"

  # Object creation.
  | "new" ⟨type-id⟩

  # Variables, field, elements of an array.
  | ⟨lvalue⟩

  # Function call.
  | id "(" [ ⟨exp⟩ { "," ⟨exp⟩ } ] ")"

  # Method call.
  | ⟨lvalue⟩ "." id "(" [ ⟨exp⟩ { "," ⟨exp⟩ } ] ")"

  # Operations.
  | "-" ⟨exp⟩
  | ⟨exp⟩ ⟨op⟩ ⟨exp⟩
  | "(" ⟨exps⟩ ")"

  # Assignment.
  | ⟨lvalue⟩ ":=" ⟨exp⟩

  # Control structures.
  | "if" ⟨exp⟩ "then" ⟨exp⟩ [ "else" ⟨exp⟩ ]
  | "while" ⟨exp⟩ "do" ⟨exp⟩
  | "for" id ":=" ⟨exp⟩ "to" ⟨exp⟩ "do" ⟨exp⟩
  | "break"
  | "let" ⟨decs⟩ "in" ⟨exps⟩ "end"

⟨lvalue⟩ ::= id
  | ⟨lvalue⟩ "." id
  | ⟨lvalue⟩ "[" ⟨exp⟩ "]"

⟨exps⟩ ::= [ ⟨exp⟩ { ";" ⟨exp⟩ } ]
```

```
⟨decs⟩ ::= { ⟨dec⟩ }
⟨dec⟩ ::=
  # Type declaration.
    "type" id "=" ⟨ty⟩
  # Class definition (alternative form).
  | "class" id [ "extends" ⟨type-id⟩ ] "{" ⟨classfields⟩ "}"
  # Variable declaration.
  | ⟨vardec⟩
  # Function declaration.
  | "function" id "(" ⟨tyfields⟩ ")" [ ":" ⟨type-id⟩ ] "=" ⟨exp⟩
  # Primitive declaration.
  | "primitive" id "(" ⟨tyfields⟩ ")" [ ":" ⟨type-id⟩ ]
  # Importing a set of declarations.
  | "import" string

⟨vardec⟩ ::= "var" id [ ":" ⟨type-id⟩ ] ":=" ⟨exp⟩

⟨classfields⟩ ::= { ⟨classfield⟩ }
# Class fields.
⟨classfield⟩ ::=
  # Attribute declaration.
    ⟨vardec⟩
  # Method declaration.
  | "method" id "(" ⟨tyfields⟩ ")" [ ":" ⟨type-id⟩ ] "=" ⟨exp⟩

# Types.
⟨ty⟩ ::=
   # Type alias.
     ⟨type-id⟩
   # Record type definition.
   | "{" ⟨tyfields⟩ "}"
   # Array type definition.
   | "array" "of" ⟨type-id⟩
   # Class definition (canonical form).
   | "class" [ "extends" ⟨type-id⟩ ] "{" ⟨classfields⟩ "}"
⟨tyfields⟩ ::= [ id ":" ⟨type-id⟩ { "," id ":" ⟨type-id⟩ } ]
⟨type-id⟩ ::= id

⟨op⟩ ::= "+" | "−" | "*" | "/" | "=" | "<>" | ">" | "<" | ">=" | "<=" | "&" | "|"
```

Les priorités des opérateurs binaires (op), de la plus haute à la plus basse, sont comme suit :

```
* /
+ -
>= <= = <> < >
&
|
```

Les opérateurs de comparaison (<, <=, =, <>, >, >=) ne sont pas associatifs. Tous les autres opérateurs listés dans op sont associatifs à gauche.

B 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

  DecsList          (decs_type 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)

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

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