CMP1 – Construction des compilateurs
TYLA – Typologie des Langages

EPITA – Apprentis promo 2011
Tous documents (notes de cours, polycopiés, livres) autorisés
Calculatrices et ordinateurs interdits.

Janvier 2009 (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. Une lecture préalable du sujet est recommandée.

Merci de répondre sur votre copie, même pour les questions à choix multiples.

1 Un peu de C++

Considérons le code C++ ci-dessous :

```
#include <cstddef>
struct Buffer
{
 typedef unsigned char value_type;
 Buffer (size_t n) : data_ (new value_type[n]) {}
 ~Buffer () { delete[] data_; }
 value_type& operator[] (size_t n) { return data_[n]; }
 value_type* data_;
};
```

Tel quel, ce code comporte des problèmes ou des limitations que nous allons examiner.

1. Le code précédent, utilisé dans l'exemple ci-après, provoque une erreur.

   ```
   int main ()
   {
    Buffer b (51);
    for (unsigned i = 0; i < 51; ++i)
     b[i] = 42;
    {
     Buffer b2 = b;
    }
    {
     Buffer b3 (1);
     b3[0] = 64;
    }
    unsigned char c = b[0];
   }
   ```

   Pourquoi ?

2. Comment corriger cela ?
3. Que peut-on changer pour simplifier le travail de debugging du développeur ?
4. Que peut-on changer pour étendre le champ d'application de cette classe (dit autrement : pour pouvoir vraiment la réutiliser) ?
5. Après les corrections de la question 2, on ajoute la ligne suivante à la fin de la définition de `main`.

```
std::cout << c << std::endl;
```

Qu'affiche-t-elle ?

2 Passage d'arguments

1. Quel(s) type(s) de passage(s) d'argument(s) est (sont) utilisé(s) en C ?
   (a) passage par valeur/copie (*call by value*)
   (b) passage par référence (*call by reference*)
   (c) passage par résultat (*call by result*)
   (d) passage par nom (*call by name*)
2. Même question avec C++ (attention, il y a un piège !).
3. Même question avec Tiger.
4. Il n'y a pas de type référence en Tiger ; comment peut-on les simuler avec des constructions du langage existantes ?

3 Const

À la différence de C++, Tiger est dépourvu de mot clef `const`. On se propose dans cet exercice de le rajouter dans le langage, en passant notamment en revue les différentes parties de notre compilateur afin de déterminer quels sont les aménagements nécessaires.

Rappel : le langage Tiger utilisé dans cet exercice est celui qui est décrit dans le Manuel de Référence du Compilateur Tiger, qui est utilisé comme référence tout au long du projet du même nom.

3.1 Préliminaires

1. À quoi sert `const` en C++ ? Il y a plusieurs réponses possibles, une seule est attendue).
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) void (C::*m)() const = &C::m1; (C().*m)();

(f) void (C::*m)() = &C::m3; (C().*m)();

3.2 Const en Tiger

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. **Spécification 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. **Spécification 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 ?