CMP1 – Construction des compilateurs – S2

EPITA – Promo 2016

Notes de cours personnelles autorisées.
Livres, polycopiés et planches de cours imprimées interdits.
Machines (calculatrices, ordinateurs, tablettes, téléphones) interdites.

Avril 2014 (1h30)

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 3 pages.

1 Starters

1. Combien de caractères (au sens « glyphe ») différents peut-on théoriquement représenter en UTF-32, un encodage à taille fixe qui utilise exactement 32 bits pour coder chaque élément ?
2. Quel est l'intérêt d'un parser réentrant ?
3. Citez un bénéfice apporté par la liaison statique de noms dans un langage compilé.
4. Comment s'appelle la construction employée dans la définition de `incr` dans le code Objective Caml ci-dessous ?

```
let start x =
  let incr y = x + y
  in incr
in
  let x = -42 in
  let y = start x in
  y 51
;;
```

2 Un peu d'imagination

On considère la grammaire suivante au format BNF, engendrant un langage d'expressions complexes (au sens de $\mathbb{C}$, l'ensemble des nombres complexes).

```
r0  ⟨exp⟩ ::= ⟨num⟩                # A literal number.
r1          | "i"                  # The imaginary unit number.
r2          | ⟨exp⟩ "+" ⟨exp⟩      # Complex addition.
r3          | ⟨exp⟩ "-" ⟨exp⟩      # Complex subtraction.
r4          | ⟨exp⟩ "*" ⟨exp⟩      # Complex multiplication.
r5          | ⟨exp⟩ "/" ⟨exp⟩      # Complex division.
r6          | "re" ⟨exp⟩           # Real part of a complex number.
r7          | "im" ⟨exp⟩           # Imaginary part of a complex number.
r8          | "(" ⟨exp⟩ ")"        # Explicit grouping.
```

On souhaitera dans l'ensemble de cet exercice suivre les règles usuelles de priorité et d'associativité classiquement utilisées en mathématiques. On considérera aussi que les opérateurs unaires (`re`, `im`) sont plus prioritaires que les opérateurs binaires.

1. Que signifie BNF ?
2. On s'intéresse à l'implémentation de $\langle num \rangle$, qui désigne un nombre littéral (entier ou réel, non signé) du point de vue du scanner. Écrivez une expression rationnelle décrivant de tels nombres.

   Par la suite, et afin de ne pas compliquer l'exercice outre mesure, on considérera que les nombres littéraux traités par le scanner seront restreints aux entiers – c'est-à-dire que l'on posera $\langle num \rangle$ = `INT`, où `INT` est un token représentant un entier littéral non signé.
3. Quelles sont les symboles terminaux de cette grammaire ?
4. Quelles sont les symboles non-terminaux de cette grammaire ?
5. Cette grammaire est-elle ambiguë ? Justifiez votre réponse.
6. Qu'est-ce qu'une valeur sémantique ?
7. Proposez une syntaxe abstraite permettant de représenter les entrées du langage engendré par la grammaire ci-dessus, soit sous la forme d'une grammaire (de syntaxe abstraite), soit sous la forme d'un diagramme de classes.
8. Continuez (sur votre copie) le schéma du parsing de l'entrée `2 + 3 * i`, commencé ci-dessous, en affichant les états successifs de la pile (symbolique et sémantique) de l'automate LR engendré par la grammaire ci-dessus. Le symbole '`$`' désigne le marqueur de fin d'entrée. Cette analyse doit *in fine* produire un arbre de syntaxe abstraite (AST).

```
action          stack                                   input

                ⊢                                   2 + 3 * i $ ⊣

shift  (2)
                ⊢   INT                               + 3 * i $ ⊣
                     2
reduce (r0)
                ⊢   exp                               + 3 * i $ ⊣
                   Num(2)
...
```

   Écrivez ou dessinez l'AST issu de ce parsing.
9. Même question pour l'entrée `im (51 * i + 43)`.
10. On considère l'implémentation d'un parser pour le langage utilisé dans cet exercice avec Bison, et la question de la reprise sur erreur face à une entrée syntaxiquement incorrecte. Qu'écririez-vous dans le fichier passé à Bison pour afin d'effectuer une telle reprise sur erreur ?
11. On souhaite écrire en C++ un évaluateur des AST produits par notre parser. Comment s'appelle la technique objet usuelle pour mettre en œuvre un tel traitement ?

12. Considérez l'implémentation partielle de l'évaluateur ci-dessous, où les nombres complexes sont implémentés par la classe standard C++ `std::complex<float>`, qui fonctionne comme vous l'imaginez. Complétez (sur votre copie) le code manquant identifié par le commentaire bien connu.

```
class Evaluator : public Base
{
public:
  void operator()(const Num& e)
  {
    value_ = e.val_get();
  }

  void operator()(const I&)
  {
    value_ = std::complex<float>(0.f, 1.f);
  }

  void operator()(const Unary& e)
  {
    e.exp_get().accept(*this);
    switch (e.oper_get())
    {
    case Unary::re: value_ = std::real(value_); break;
    case Unary::im: value_ = std::imag(value_); break;
    }
  }

  void operator()(const Binary& e)
  {
    /* FIXME: Some code was deleted here.  */
  }

  std::complex<float> value_get() const
  {
    return value_;
  }

private:
  std::complex<float> value_;
};
```

13. Écrivez le code de la classe `Base`, dont dérive `Evaluator`.
14. Quelle nom meilleur pourrait-on donner à la classe `Base` du listing précédent ?