CMP1 – Construction des compilateurs

EPITA – Promo 2014

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

Janvier 2012 (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 (entier) est recommandée.

1 Le retour du Logo

Considérez la grammaire EBNF suivante, qui s'inspire d'un sous-ensemble du langage Logo.

```
⟨instrs⟩ ::= ⟨instr⟩+

⟨instr⟩  ::= "forward" ⟨exp⟩
           | "left" ⟨exp⟩
           | "right" ⟨exp⟩

             # Loop.
           | "repeat" ⟨exp⟩ "[" ⟨instrs⟩ "]"

             # Subprogram definition.
           | "to" "id" ⟨args⟩ ⟨instrs⟩ "end"

             # Subprogram call.
           | "id" ⟨exps⟩

⟨exps⟩   ::= ⟨exp⟩*

⟨exp⟩    ::= "num"
           | ":" "id"

⟨args⟩   ::= ⟨arg⟩*

⟨arg⟩    ::= ":" "id"
```

Dans cette grammaire, le terminal "num" désigne un entier littéral et le terminal "id" un identifiant (qui suit les mêmes règles lexicales que les identifiants du langage Tiger).

Le langage Logo originel permet notamment de dessiner sur un écran à l'aide d'une "tortue" que l'on manipule avec des instructions comme `forward 10` (avance de 10 unité en traçant un trait) ou `right 30` (tourne à droite de 30 degrés). Il permet également la définition de sous-programmes (procédures) à l'aide du mot-clef `to`, qui sont exécutés lorsque leur nom est utilisé comme instruction.

1. On souhaite reconnaître ce langage à l'aide d'un scanner et d'un parser engendrés par Flex et Bison respectivement. Dressez une liste des tokens qui seront produits par le scanner et utilisés par le parser.

2. Comment faire comprendre à votre scanner Flex que l'entrée `'forward'` doit produire un token FORWARD et non un token ID (correspondant au symbole `'forward'`) ?

3. Pourquoi est-ce que votre scanner Flex générera à partir de l'entrée Tiger `'to'` un seul token TO, et non un token ID représentant l'identifiant `'t'` suivi d'un autre token ID représentant l'identifiant `'o'` ?

4. Cette grammaire présente-t-elle une (des) ambiguïté(s) ? Si oui, donner un exemple de mot du langage engendré par celle-ci, donnant lieu à plus d'une dérivation.

5. Que signifie EBNF ?

6. On décide d'implémenter un parser Bison pour ce langage. Quel(s) changement(s) éventuel(s) pensez-vous apporter à cette grammaire avant cette opération ?

7. On souhaiterait que le parser sache faire un peu de reprise sur erreur, afin que celui-ci produise des messages plus fournis (ne se limitant pas à la première erreur rencontrée). On utilise pour cela le token spécial `'error'` produit en cas d'erreur.

   Quelle(s) règle(s) ajouteriez-vous à un fichier d'entrée Bison issu de la grammaire modifiée de la question 6 afin d'ajouter cette reprise sur erreur ?

8. Les arbres de syntaxe abstraite ou Abstract Syntax Trees (ASTs) produits par Bison seront implémentés sous forme d'objets C++, à l'instar de l'implémentation des ASTs de `tc`. Donnez une description synthétique des classes représentant les différents nœuds de la syntaxe abstraite du langage étudié dans cet exercice, soit sous forme de diagramme de classes, soit sous forme de code C++. Vous pouvez indiquer les données (attributs) des dites classes, mais il est inutile de mentionner leurs méthodes.

9. Voici un programme Logo qui dessine une forme de rosace :

```
to square :length
repeat 4 [ forward :length right 90 ]
end
repeat 36 [ square 50 right 5 square 100 right 5 ]
```

   Donnez une représentation arborescente (graphique ou textuelle) de la syntaxe abstraite de ce programme.

10. Voici un extrait du code de la syntaxe abstraite de notre langage Logo.

```
class Ast
{
public:
  virtual ~Ast () {};
  virtual void accept (Visitor& v) const = 0;
};

class Instr : public Ast
{
};
```

```
class Subprogram : public Instr
{
public:
  Subprogram (const std::string& name, const ArgList* args,
              const InstrList* body)
    : name_ (name), args_ (args), body_ (body)
  {
  }
  virtual ~Subprogram () { delete args_; delete body_; }

  std::string      name_get() { return name_; } const
  const ArgList*   args_get() { return args_; } const
  const InstrList* body_get() { return body_; } const

  virtual void accept (Visitor& v) const { v (*this); }

private:
  std::string name_;
  const ArgList* args_;
  const InstrList* body_;
};

// ...
```

On souhaite implémenter un pretty-printer pour cette syntaxe abstraite grâce à un visiteur, affichant récursivement un AST. Celui-ci sera implémenté sous la forme d'une classe `PrettyPrinter`, dérivée de la classe abstraite `Visitor`, dont un extrait est donné ci-dessous :

```
struct Visitor
{
  virtual void operator() (const Right& e) = 0;
  virtual void operator() (const Left& e) = 0;
  virtual void operator() (const Subprogram& e) = 0;
  // ...
};
```

Écrivez la classe `PrettyPrinter`, en limitant les méthodes de visite ('`operator()`') à la seule classe `Subprogram` donnée ci-avant (on souhaite cependant voir le reste de la classe).

On rappelle qu'un visiteur permet l'encapsulation d'un traitement spécialisé pour toutes les classes concrètes d'une hiérarchie de classes (en l'occurrence, celle de notre syntaxe abstraite) *et* des données associées ; tenez-en compte dans votre réponse ! (Note : le code à écrire n'est pas très long ; environ une quinzaine de lignes.)

2 De la virtualité et de la destruction en C++

1. Qu'appelle-t-on une fonction membre virtuelle pure en C++ (également connue sous le nom de "méthode (polymorphe) abstraite") ?
2. Pourquoi ne doit-on pas appeler de méthodes virtuelles d'un objet lors de la construction de celui-ci ?

3. Lors de la situation de la question précédente, les compilateurs C++ déclenchent habituellement une erreur. Par exemple, avec le code ci-dessous :

```
struct A
{
  A () { m (); }
  virtual void m () = 0;
};

struct B : A
{
  virtual void m () {};
};

int main ()
{
  A* b = new B;
  b->m ();
  delete b;
}
```

on obtient l'erreur suivante à l'édition de liens avec g++ 4.4 :

```
a.cc: In constructor 'A::A()':
a.cc:3: warning: abstract virtual 'virtual void A::m()' called from
  constructor
/tmp/ccfvyPSG.o: In function 'A::A()':
a.cc:(.text._ZN1AC2Ev[A::A()]+0x1f): undefined reference to 'A::m()'
collect2: ld returned 1 exit status
```

Malgré cela, le code ci-dessous compile bel et bien avec le même compilateur :

```
struct A
{
  A () { f (); }
  void f () { m (); }
  virtual void m () = 0;
};

struct B : A
{
  virtual void m () {};
};

int main ()
{
  A* b = new B;
  b->m ();
  delete b;
}
```

Il affiche cependant l'erreur suivante à l'exécution :

```
pure virtual method called
terminate called without an active exception
```

(a) Expliquez ce qui s'est passé lors de l'exécution de ce programme.

(b) À votre avis, pourquoi le compilateur n'a pas été capable de prévenir que le programme était mal formé (*ill-formed*) ?

4. Considérons à présent le code C++ suivant :

```
struct C
{
  virtual ~C () = 0;
};

struct D : C
{
};

int
main ()
{
  D d;
}
```

Quelle peut être l'utilité de la ligne 3 ?

5. Le code de la question 4 compile bien, mais provoque une erreur à l'édition de liens. Le message produit par g++ 4.4 est le suivant :

```
/tmp/ccQdx4dr.o: In function 'D::~D()':
b.cc:(.text._ZN1DD1Ev[D::~D()]+0x1f): undefined reference to 'C::~C()'
/tmp/ccQdx4dr.o: In function 'D::~D()':
b.cc:(.text._ZN1DD0Ev[D::~D()]+0x1f): undefined reference to 'C::~C()'
collect2: ld returned 1 exit status
```

À quoi est due cette erreur ? ("Le destructeur de `C` est absent" n'est pas une réponse suffisante.)

6. Même si cela paraît contre nature, le langage C++ autorise la définition d'une implémentation pour une fonction membre virtuelle pure. Dans le cas d'un destructeur virtuel pur, le standard ISO C++ précise qu'une telle définition est même indispensable, si l'on veut être en mesure de pouvoir instancier la classe contenant ce destructeur ou l'une de ses classes dérivées.

Fort de cette information, qu'ajouteriez/modifieriez-vous dans le code de la question 4 pour le rendre propre à la compilation et à l'exécution ?