Correction du partiel CMP

EPITA – Apprentis promotion 2012
Tous documents (notes de cours, polycopiés, livres) autorisés.
Calculatrices et ordinateurs interdits.

Juin 2010 (1h30)

Correction: Le sujet et sa correction ont été écrits par Roland Levillain. Le *best of* est tiré des copies des étudiants, fautes de français y compris, le cas échéant.

Bien lire les questions, chaque mot est important. Écrire court, juste, et bien. Une argumentation informelle mais convaincante est souvent suffisante.

Une lecture préalable du sujet est recommandée.

1 Programmation orientée objet

1. Définissez les termes suivants :
   (a) méthode

   Correction: Fonction attachée à un objet, et faisant partie de l'interface de celui-ci.

   (b) méthode polymorphe

   Correction: Méthode redéfinissable dans les sous-classes de son site de définition, et dont la sélection (liaison) s'effectue à l'exécution, selon le type dynamique de son objet-cible (argument « zéro », à gauche du point dans la syntaxe C++).
   À ne pas confondre avec la surcharge de méthodes (et de fonctions) ; cf. infra, question 4 de l'exercice 2.

   (c) multiméthode

   Correction: Méthode dont la sélection/liaison (dynamique) peut dépendre du type dynamique de plus d'un de ses arguments.

   Best-of:
   – Sincèrement, je ne pense pas avoir entendu parler de multiméthodes pendant le cours. Evidemment, je peux me tromper.

2. Donner un cas d'utilisation de multiméthode. Quelques exemples :

Correction:
- Le calcul de l'intersection de plusieurs formes géométriques, instances de classes `Rectangle`, `Circle`, etc., dérivant d'une classe abstraite `Shape`. Le point d'entrée de ce calcul prend deux `Shapes` en argument, la sélection (*dispatch*) de l'implémentation dépend de leurs types dynamiques.
- L'appariement (*matching*) d'arbres, comme les AST de `tc`, est un autre exemple où les multiméthodes seraient utiles : pour comparer les nœuds de deux arbres, il suffirait d'écrire les cas évidents de correspondances valides (`IntExp` correspond à `IntExp`, etc.) ainsi que les cas particuliers (`MetavarExp`, une métavariable de type « expression », peut-être appariée à n'importe quelle *Exp*).
- Enfin, les tâches effectuées par les visiteurs de `tc` sont un exemple que l'on pourrait aussi implémenter avec des multiméthodes (si le C++ les fournissait) : le polymorphisme aurait lieu à la fois sur le type de nœud à visiter/traiter (`IfExp`, `VarDec`, etc.), et sur le type de traitement à effectuer (pretty-printing, type-checking, etc.).

3. Citer un langage qui dispose de multiméthodes.

   Correction: Au choix : Common Lisp (via CLOS), Dylan, Nice, Slate, Cecil, R, Groovy, Perl 6, Clojure, C# 4.0, Fortress.

4. Comment peut-on simuler le mécanisme d'une multiméthode (à deux arguments) dans un langage qui en est dépourvu ?

   Correction: Avec le design pattern Visiteur.

5. Qu'appelle-t-on un objet-fonction en C++ ?

   Correction: Un objet qui peut se comporter comme une fonction, c'est-à-dire qui dispose d'un `operator()`.

6. Écrivez la définition d'un objet-fonction C++ représentant la fonction mathématique

$$f_a : \left\{ \begin{array}{ccc} \mathbb{R} & \to & \mathbb{R} \\ x & \mapsto & \sin(ax) \end{array} \right. \quad \text{avec } a \in \mathbb{R}.$$

Correction:

```
class f_t
{
public:
  f_t (float a)
    : a_(a)
  {
  }

  float operator() (float x)
  {
    return std::sin (a_ * x);
  }

private:
  float a_;
};
```

2 Langages

1. De quel type de passage par argument dispose-t-on en langage C ?

   Correction: Le passage par valeur (ou copie), éventuellement constante. Également accepté : le passage par pointeur (ou adresse), éventuellement constant(e).

2. Même question avec le langage Tiger.

   Correction: Le passage par valeur pour les types atomiques/primitifs (`int`, `string` et `nil`), par référence pour les autres types (tableaux, enregistrements, objets).

3. En C++, quels sont les bénéfices apportés par un passage par référence ?

   Correction:
   (a) On manipule l'original, pas une copie, ce qui permet de faire des effets de bord sur les arguments plus élégamment et plus sûrement qu'en C (i.e., sans pointeurs).
   (b) Plutôt que de copier une valeur entière (passage par valeur), le passage par référence permet de ne passer que son adresse (de façon transparente) : pour un objet plus lourd qu'une adresse, ce type de passage est plus efficace que le passage par valeur.

4. Qu'appelle-t-on polymorphisme de surcharge ? Quelle différence y a-t-il avec le polymorphisme d'inclusion ?

   Correction: Le polymorphisme de surcharge (ou *ad hoc*) permet de définir plusieurs fonctions ou méthodes homonymes, différant par leurs arguments (nombre et type) et dans le cas d'une méthode, par leur qualificatif `const`. Il s'agit d'un mécanisme *statique* : la sélection et la liaison s'effectue à la compilation, en utilisant les types statiques des arguments.
   Le polymorphisme d'inclusion s'appuie sur les méthodes polymorphes (appelées aussi fonctions membres virtuelles en C++) : il s'agit d'un mécanisme *dynamique* : la sélection (*dispatch*) s'effectue à l'exécution, en utilisant le type dynamique de la cible (argument 0 de la méthode).

5. Pour écrire (resp. lire) des données dans (resp. depuis) un flux, on utilise l'opérateur '<<' (resp. '>>') en C++. Ces opérateurs sont définis sous forme de fonctions surchargées (à deux arguments), et non de méthodes surchargées (à un argument). Pour quelle raison ?

Correction: Parce que :

(a) La syntaxe de ces opérateurs est telle que le flux de sortie (resp. d'entrée) est situé à leur gauche, tandis que la valeur à écrire (resp. lire) est à leur droite :

```
ostr << var;
istr >> var;
```

(b) Lorsque l'on définit un opérateur binaire (comme '<<' et '>>') dans une classe, le type du premier opérande (gauche) est fixé : c'est celui de la classe. Or d'après le point 5a, il s'agit forcément du flux. Un opérateur de flux défini sous forme de méthode devrait donc faire partie de la classe du flux.

(c) D'après 5b, pour ajouter de nouveaux opérateurs de flux définis sous forme de méthodes, il nous faudrait donc pouvoir modifier les classes de flux. Or ces flux sont fournis par la bibliothèque standard du C++ (instances de classes dérivant de `std::ios_base`), que l'on ne peut ni modifier (pour ne pas casser l'API (Application Program Interface) C++ standard), ni étendre de façon non intrusive (le langage ne le permet pas).

Par conséquent, on ne peut pas définir d'opérateurs de flux sous forme de méthodes : il faut utiliser des fonctions.

6. Pourquoi les opérateurs de flux '<<' et '>>' sont-ils plus sûrs que la fonction `printf` ?

Correction: Les opérateurs de flux '<<' et '>>' permettent un typage statique fort de leur arguments, qui permet d'éviter de nombreuses erreurs de programmation. À l'inverse, `printf` repose complètement sur la cohérence entre la chaîne de format qui lui est passée en premier argument, et les arguments suivants, que le compilateur ne peut pas compter, et dont il peut encore moins vérifier les types, puisque le second argument de `printf` est '`...`' (*ellipsis*). Il est donc possible de passer moins d'arguments ou plus d'arguments qu'attendus, et avec de mauvais types, sans réelle possibilité de vérification statique, et avec très peu de contrôle possible à l'exécution (le mécanisme de passage d'argument derrière '`...`' est bas niveau).

Best-of:
- Il sont plus sur que `printf` car il n'y a pas de bufferisation.
- Ces opérateurs sont plus sûrs car `printf` est une couche supplémentaire qui repose sur ces opérateurs.

3 Compilation

Voici un programme Tiger incorrect :

```
1 let
2   function fib (n : int) : int =
3     if n <= 1
4     then 1;
5     else fib (n − 1) + fib (n − 2);
6 in
7   fib (10);
8 end
```

1. Quelle est la nature du problème ici ? (Dit autrement : quel composant du compilateur va détecter qu'il y a une erreur ?)

Correction: Il y a des points-virgules en trop après les clause `then` et `else`, ainsi qu'après l'appel `fib (10)`, qui font de ce code un programme faux du point de vue syntaxique : c'est le parser qui détectera cette erreur. En effet, le point-virgule est un *séparateur* en Tiger (alors qu'il s'agit d'un *terminateur* en C++, par exemple).

2. Proposez une correction de ce programme.

   Correction: Retirer les points-virgules à la fin des lignes 4, 5 et 7.

3. Le comportement par défaut d'un couple scanner-parser lors d'une erreur lexicale ou grammaticale est d'arrêter l'analyse. Lors du projet Tiger, vous avez dû utiliser une technique disponible dans Bison permettant malgré tout d'obtenir (le plus souvent) un arbre de syntaxe abstraite (évidemment faux), représentant partiellement le programme en entrée.

   (a) Comment s'appelle cette technique ?

   Correction: Il s'agit de la reprise sur erreur par suppression de tokens, utilisable tant dans le scanner que dans le parser.

   (b) Comment la met-on en œuvre dans un parser ?

   Correction: En utilisant le token error dans le parser.
   (Quant aux erreurs lexicales, un token invalide est tout simplement ignoré.)

   (c) Comment fonctionne-t-elle ?

   Correction: Lorsque le parser LR arrive dans un état « erreur », il effectue les actions suivantes :

   i. dépiler les éléments de la pile de l'automate (si besoin est) jusqu'à ce que l'automate soit dans un état où l'action correspondant au token error soit un décalage (*shift*) ;
   ii. pousser (*shift*) le token error sur la pile ;
   iii. rejeter tous les symboles de l'entrée (si nécessaire) jusqu'à ce qu'un état ayant une action autre qu'une action d'erreur soit atteint ;
   iv. reprendre normalement l'analyse syntaxique depuis ce point.

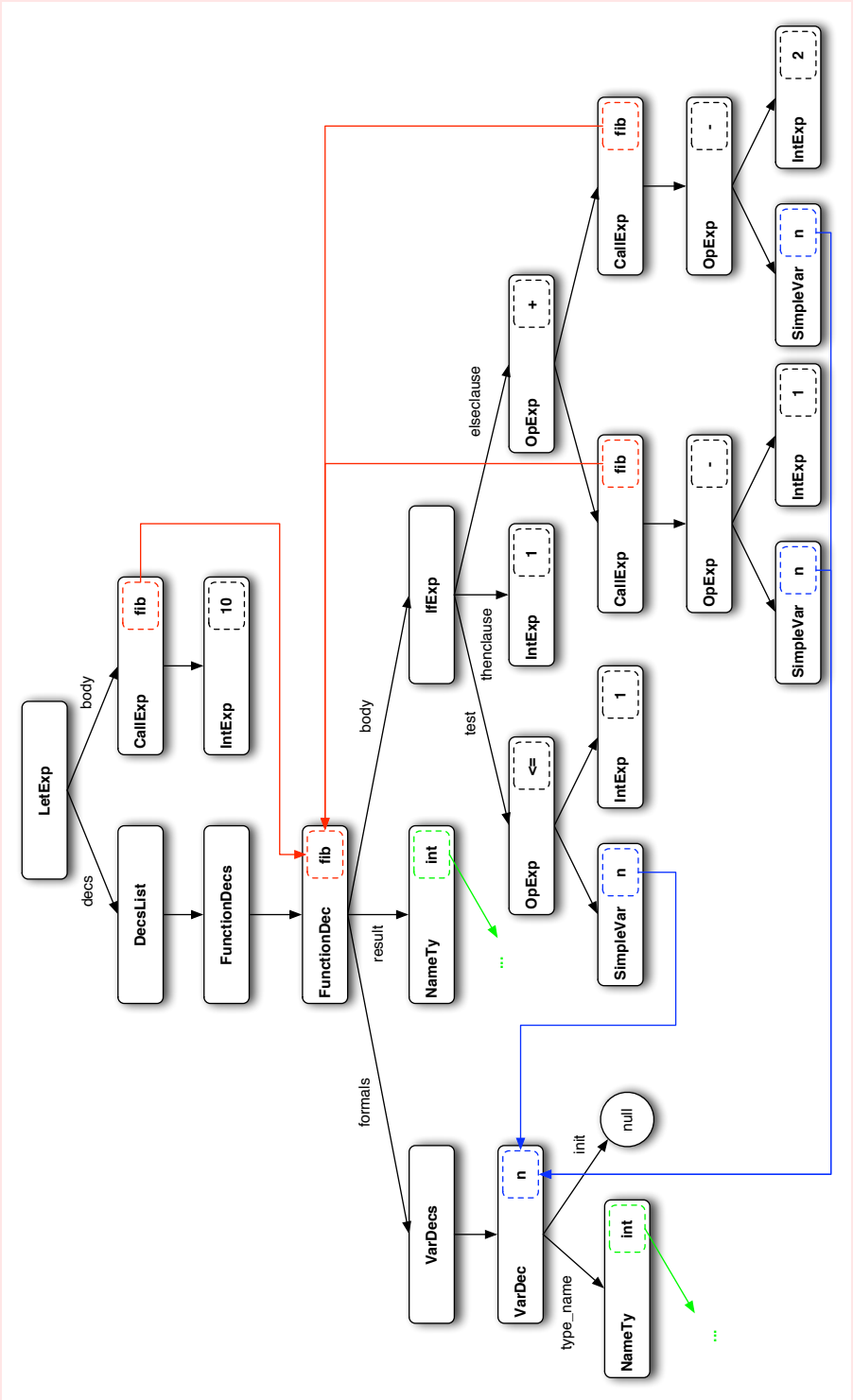
   Pour rendre cette approche effective, il faut faire en sorte que les tokens soient soit bien « encadrés » dans la partie droite des productions, afin d'éviter de jeter l'intégralité de l'entrée en tentant de retrouver un état de reprise, ou au contraire de reprendre trop tôt depuis une situation qui déclenchera une nouvelle erreur plus tard.

4. Quelle différence faites-vous entre un arbre de dérivation et un arbre de syntaxe abstraite (ou Abstract Syntax Tree (AST)) ?

Correction: L'arbre de dérivation ou *parse tree* est un produit direct du parser, et reproduit fidèlement la syntaxe concrète du programme en entrée. Un AST est une représentation synthétique du programme, débarrassée des artefacts de la syntaxe concrète.
On trouvera donc dans un arbre de dérivation des éléments qui sont absents de la syntaxe abstraite :
- tous les non terminaux intermédiaires faisant apparaître toutes les productions utilisées lors de l'analyse syntaxique (par ex. : `program` → `exp` → `INT`) ;
- la ponctuation : virgules, points-virgules (séparateurs ou terminateurs), deux-points, parenthèses, crochets, etc.
- le sucre syntaxique, éventuellement (partiellement) supprimé dans l'AST : opérateurs de logique booléens, `if-then` sans `else`, moins unaire, etc.
- le sel syntaxique, c'est-à-dire des éléments de syntaxe qui pourraient ne pas faire partie de l'expression d'un programme, et qui permettent (essentiellement) à l'utilisateur de mieux repérer ses erreurs et aux outils de l'aider dans cette tâche. Les mots-clefs terminateurs du Bourne Shell (`fi`, `done`, `esac`, etc.) remplissent cette fonction (en sus de simplifier l'analyse du langage).

5. Donnez une représentation de l'AST du programme Tiger ci-dessus (après correction). Un aide-mémoire comportant une liste (partielle) des nœuds de la syntaxe abstraite de Tiger est disponible en annexe A.

Correction:

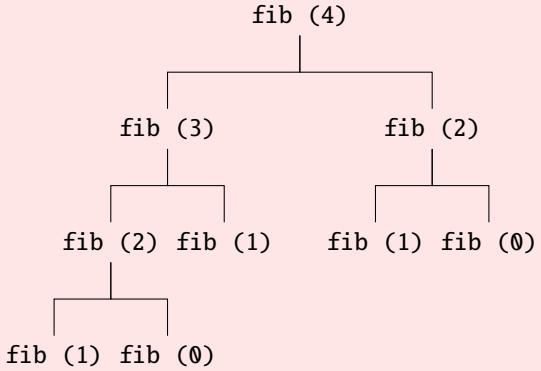


6. Sur le schéma de la question précédente, indiquer les liens entre sites d'utilisations et sites de définitions à l'aide de flèches partant des premiers et allant vers les seconds.

Correction: Voir la réponse précédente. On notera que les nœuds '`NameTy int`' n'ont pas de cible réellement matérialisée, puisque c'est la sémantique du langage qui fournit cette liaison « par défaut ».

7. L'implémentation de `fib` ci-dessus est foncièrement inefficace. Pourquoi ? Proposez une implémentation plus rapide.

Correction: Notons tout d'abord que cette fonction n'est pas lente à cause du simple fait qu'elle est récursive ! Une implémentation écrite pour tirer parti de l'optimisation des *tail calls* serait parfaitement comparable à une version itérative du point de vue des performances.
La vraie raison de l'inefficacité de la fonction `fib` ci-dessus est qu'elle ne factorise pas les appels récursifs identiques, ce qui rend sa complexité exponentielle. À titre d'exemple, le calcul de `fib (4)` déclenche les appels récursifs suivants :



On constate donc que `fib (2)` est appelée deux fois, `fib (1)` trois fois, et `fib (0)` deux fois, et pourtant le résultat de ces appels de ne varie pas (`fib` est une fonction pure, qui n'effectue pas d'effet de bord). On peut démontrer que cette implémentation de `fib` a une complexité exponentielle (laissé en exercice au lecteur).
Il est cependant possible d'écrire une fonction en temps linéaire de la suite de Fibonacci en accumulant successivement les termes de la suite, de sorte à ne les calculer qu'une seule fois. En voici une version récursive, facilement transposable en un équivalent itératif :

```
let
  function fib (n : int) : int =
    let
      function fib_ (i : int, a : int, b : int) : int =
        if i = n
        then b
        else fib_ (i + 1, b, a + b)
  in
    fib_ (0, 0, 1)
  end
in
  fib (10);
end
```

On notera que la fonction auxiliaire `fib_` est une fonction récursive terminale, qu'un compilateur pourrait optimiser pour rendre son exécution aussi rapide qu'une boucle (*tail call elimination*).

4 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 qui vous est remise. N'y passez pas plus de dix minutes.

Cette épreuve

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 (estimation).
   E Plus de 120 minutes (estimation).
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

3. Quelle a été votre implication dans les cours de Construction des compilateurs ?
   A Rien.
   B Bachotage récent.
   C Relu les notes entre chaque cours.
   D Fait les annales.
   E Lu d'autres sources.
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.
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.

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 par semaine).
   C Supportable (environ une heure de travail par semaine).
   D Légère (quelques minutes par semaine).

Les formateurs

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.
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

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.
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 de travail par semaine).
    D Légère (une ou deux heures par semaine).
    E J'ai été dispensé du projet.
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

12. Formation au C++.
13. Formation à la modélisation orientée objet et aux *design patterns*.
14. Formation à l'anglais technique.
15. Formation à la compréhension du fonctionnement des ordinateurs.
16. Formation à la compréhension du fonctionnement des langages de programmation.
17. Formation au travail collaboratif.
18. Formation aux outils de développement (contrôle de version, systèmes de construction, débogueurs, générateurs de code, etc.)

Questions 19-30 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.

19. Rush `.tig` : mini-projet en Tiger (Bistromatig).
20. `TC-0`, Scanner & Parser.
21. `TC-1`, Scanner & Parser, Tâches, Autotools.
22. `TC-2`, Construction de l'AST et pretty-printer.
23. `TC-3`, Liaison des noms et renommage.
24. `TC-4`, Typage et calcul des échappements.
25. `TC-5`, Traduction vers représentation intermédiaire.
26. Option `TC-D`, Suppression du sucre syntaxique (boucles `for`, comparaisons de chaînes de caractères).
27. Option `TC-I`, Mise en ligne du corps des fonctions.
28. Option `TC-B`, Vérification dynamique des bornes de tableaux.
29. Option `TC-A`, Surcharge des fonctions.
30. Option `TC-O`, Désucrage des constructions objets (transformation Tiger → Panther). .

A 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     ()
    RuleDec         (Exp match, Exp build)
    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)

/Metavariable/      (unsigned id)
  MetavarExp        ()

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

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