Partiel de Compilation

<brains on>

1 Bison

Une `exp` est de type `exp_t`.

1. Qu'exprime la grammaire suivante ?
   ```
   exps: exp | exp ';' exps
   ```
2. Que pourrait-on lui reprocher et comment la réécriez-vous ?
3. Complétez votre proposition en incluant les actions et la reprise sur erreur.
4. Ecrivez une grammaire pour "une liste d'`exp` (0 ou plusieurs) séparés par des `;`", avec les actions et la reprise sur erreur.

2 Typage

Soient les fonctions suivantes :

```
function int_id (a: int) : int = a
function string_id (a: string) : string = a
```

1. Qu'est-ce qui est gênant dans le pouvoir d'expression de `Tiger`.
2. Il y a deux possibilités évidentes pour pallier ce défaut, l'une que l'on appellera *implicite* et l'autre *explicite*. Chacune de ces possibilités donnent deux extensions de `Tiger`. Réécrivez ce code pour chacune de ces deux saveurs de `Tiger`. Il ne vous est *pas* demandé du `Tiger` traditionnel, mais bien d'étendre `Tiger` de deux façons différentes.

3 Un gros vide

Pour passer à la page (0 point).

4 Mise en ligne (Inlining)

La *mise en ligne* consiste à replacer un appel de fonction par le corps de la fonction en elle-même. Par exemple la mise en ligne de

```
function add (a: int, b: int) : int = a + b
```

dans `add (20, 22)` donne bien entendu (Et encore...) `20 + 22`.

Sans chercher à aller jusqu'au bout, étudions les difficultés pour l'implémentation de la mise en ligne dans *notre* implémentation de `tc`.

1. En considérant e.g.

   ```
   function is_smaller (a: string, b: string) : int = a < b,
   ```

   à quelle étape de la compilation suggérez-vous d'effectuer la mise en ligne ?

2. La mise ne ligne naïve de

   ```
   function double (a: int) : int = a + a
   ```

   peut conduire à des résultats désagréable. Comme quoi ?

3. Comment faut-il faire alors ?

4. En vous basant sur les question précédentes, quelle différence faites-vous entre fonctions mises en ligne et macros en `C` ?

5. Comment faire dans le cas de

   ```
   let fact (n : int) : int =
     if n = 1 then
       0
     else
       n * fact (n - 1)
   in
     fact (42)
   end
   ```

   Qu'en déduisez-vous ?

6. Considérons

   ```
   1. let var delta := 1
   2.     function advance (x: int) : int = x + delta
   3.     function start_from (delta: int) : int = advance (0) + delta
   4. in
   5.     start_from (delta)
   6. end
   ```

   (a) A quel problème s'expose-t-on ?

   (b) Sauriez-vous réécrire cet exemple pour blinder contre ça ?

   (c) Quelle solution proposez-vous pour que l'implémentation de la mise en ligne ne soit pas galère à cause de ça ?

   (d) Comment implémenteriez-vous ceci dans `tc` ? Evidemment je ne veux pas de code ici, mais décrivez correctement votre suggestion.

`<brains off>`