Série d'exercices #8

IFT-2035

November 11, 2024

8.1 Définition par cas

Soit une macro iftcase en ELisp qui peut s'utiliser comme suit:

```
(iftcase exp
  ((1 3 5) exp1)
  ((4) exp2)
  (_ exp3))
```

qui signifierait: evalue d'abord *exp* puis évalue exp_1 si *exp* rend 1, 3, ou 5; ou évalue exp_2 si *exp* rend 4; ou évalue exp_3 sinon.

1. Montrer l'expansion désirée pour ce code
2. Identifier les risques possibles d'évaluation répétée excessive et de capture de nom, si la macro est définie trop naïvement.
3. Donner une définition de cette macro qui ne souffre pas de ces problèmes.

8.2 Déterminer le passage de paramètres

Écrire un programme (dans un langage hypothétique avec une syntaxe similaire à celle de C), qui donnerait un résultat différent dans chacune des 4 méthodes de passage de paramètres:

- passage de paramètres par valeur
- passage de paramètres par référence
- passage de paramètres par valeur-résultat
- passage de paramètres par nom

Bonus portée Modifier le programme de sorte qu'il produise des résultats différents selon que la portée des variables est lexicale ou dynamique (pour un total de 8 résultats différents pour les 8 combinaisons possibles).

8.3 Passage de paramètres

Soit le code suivant dans le langage hypothétique Zlika:

```
VAR a : INTEGER, b, c : ARRAY OF INTEGER;
...
PROCEDURE move (x, y : INTEGER, n : INTEGER)
LOCAL VAR a
BEGIN
   a := y;
   x := a + n;
   y := y - n;
END
...
move (a, b[a], c[a])
```

Réécrire 4 fois le code ci-dessus en C:

- Une fois, en supposant que Zlika passe les arguments par valeur.
- Une fois, en supposant que Zlika passe les arguments par référence.
- Une fois, en supposant que Zlika passe les arguments par valeur-résultat.
- Une fois, en supposant que Zlika passe les arguments par nom.

Questions complémentaires:

- Combien de styles de passage de paramètres le langage C offre-t-il?
- Que se passe-t-il si on essaie de faire de telles simulations dans un langage qui utilise un autre style de passage de paramètres?

8.4 Raisonner

Soit le morceau de code suivant, où `f` est une fonction quelconque que l'on ne connaît pas et où le langage utilise une syntaxe de style C:

```
{
   int table[2] = {0, 1};
   int size = 2;
   int tmp = 0;

   f (table, size);
   ...
}
```

On aimerait savoir si certaines conditions sont nécessairement toujours vraies aprés l'appel à `f`. On s'intéresse plus particulièrement aux conditions suivantes:

- `table[0] == 0`

- `size == 2`
- `tmp == 0`

Indiquer lesquelles de ces trois conditions sont nécessairement vraies dans chacun des cas suivants:

1. Le langage est exactement comme C: portée statique, passage d'arguments par valeur, affectation autorisée.
2. Le langage est comme C sauf que l'affectation (autre que l'initialization) est interdite.
3. Le langage est comme C sauf que les arguments sont passés par référence.
4. Le langage est comme C mais avec portée dynamique.