Série d'exercices #9

IFT-2035

November 20, 2023

9.1 Expression conditionnelle à multiple branches

Écrire une macro mif en ELisp qui peut s'utiliser comme suit:

```
(mif
  (cond1 exp1)
  (cond2 exp2)
  ...
  (_ expn))
```

et dont l'expansion devrait être

```
(if cond1 exp1
  (if cond2 exp2
    ...
      expn)..)
```

Ensuite étendre cette macro pour que si exp_n manque, la macro renvoie la valeur de $cond_n$ à la place.

9.2 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).

9.3 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?

9.4 Programmation fonctionelle en C

Soit la fonction C suivante:

```
void main (void)
{ /* Élimine les caractères répétés et
   * stoppe après la première ligne vide.  */
  int c;
  int last = EOF;
  while ((c = getchar ()) != EOF) {
    if (c == last) {
      if (c == '\n') {
        break;
      } else {
        continue;
      }
    }
    putchar (last = c);
  }
}
```

D'abord, réécrire le code dans un style de programmation structurée stricte, c'est à dire sans utiliser de continue, break, ou goto.

Ensuite, réécrire le code à nouveau mais cette fois dans un style fonctionnel, c'est à dire sans opération d'affectation (sauf bien sûr dans les initialisations, e.g. `int last = EOF`). Il faudra introduire une fonction récursive auxiliaire et éliminer while, break, et continue.

Essayer de faire ces réécritures en les subdivisant en plusieurs petits pas simples, où on peut facilement voir que chaque pas préserve la sémantique du code precedent, sans avoir besoin de comprendre le code dans son ensemble.