est correct et affiche `4`*, le programme Caml*

```
let f () = x in let x = 4 in print_int(f())
```

est incorrect, car la variable `x` *dans le corps de* `f` *ne fait pas partie de l'environnement de définition de* `f`*. Il est nécessaire de déclarer cette variable avant* `f`

```
let x = 4 in let f () = x in print_int(f())
```

Autre différence, les compilateurs de Caml évaluent les arguments de la droite vers la gauche. Par exemple, le programme

```
let n = ref 0
in let f x y = x
in let g z = (n := !n + z; !n)
in print_int (f (g 2) (g 7))
```

donne le résultat `9` *et non* `2`*.*

Cependant, la définition de Caml ne spécifie pas l'ordre d'évaluation des arguments d'une fonction. Chaque concepteur d'un compilateur pour ce langage est libre de choisir d'évaluer les arguments dans l'ordre qu'il veut. C'est au programmeur d'écrire des programmes dont le résultat ne dépende pas de l'ordre d'évaluation, par exemple en distinguant les fonctions des procédures.

Enfin, il n'y a pas de `return` *en Caml et le résultat de l'exécution d'une instruction, comme de l'évaluation d'une expression, est un couple formé d'une valeur, éventuellement* `()`*, et d'une mémoire.*

Exercice 2.7

Donner la définition de la fonction Σ de Caml, en supposant que l'évaluation des arguments se fait toujours de la droite vers la gauche.

2.2.5 C

La définition de la fonction Σ de C présente également quelques différences avec la définition de la fonction Σ de Java.

Tout d'abord, en C, les références créées au moment de l'appel d'une fonction sont supprimées de la mémoire à la fin de l'exécution du corps de la fonction.

Ensuite, comme en Caml, il y a un seul espace de noms pour les fonctions et les variables, et les fonctions sont déclarées dans l'environnement comme les variables. Dans l'environnement, on associe au nom `f` *non seulement la liste*