des arguments formels et le corps de la fonction, mais aussi l'environnement `e` *à étendre avec les arguments pour exécuter le corps de la fonction. Cet environnement est, comme en Caml, l'environnement de définition de la fonction. Par exemple, le programme*

```
int f () {return x;}

int x = 4;

int main () {
 printf("%d\n",f());
 return 0;}
```

est incorrect.

Enfin, les compilateurs de C évaluent aussi les arguments d'une fonction de la gauche vers la droite, comme en Java. Cependant, la définition de C, comme celle de Caml, ne spécifie pas l'ordre d'évaluation des arguments d'une fonction et c'est au programmeur d'écrire des programmes dont le résultat ne dépend pas de l'ordre d'évaluation.

Exercice 2.8

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

2.3 Les expressions comme instructions

Maintenant que nous avons défini le résultat de l'évaluation d'une expression comme un couple formé d'une valeur et d'un état, nous pouvons mieux comprendre le lien entre expressions et instructions.

En C, n'importe quelle expression suivie d'un point-virgule est une instruction. La valeur de l'expression est simplement ignorée quand on l'utilise comme une instruction. Ainsi, si Θ`(t,e,m,G)` est le couple `(v,m')` alors Σ`(t ;,e,m,G)` est le couple formé du booléen `normal` et de la mémoire `m'`. La situation est un peu similaire en Java, sauf que seules quelques expressions sont éligibles pour être utilisées comme instructions. Par exemple, si `f` est une fonction, alors `f(`t_1`,...,`t_n`) ;` est, comme nous l'avons vu, une instruction, mais ce n'est pas le cas de `1 ;`. En Caml, il n'y a pas de différence entre instructions et expressions, les instructions sont simplement des expressions de type `unit`.