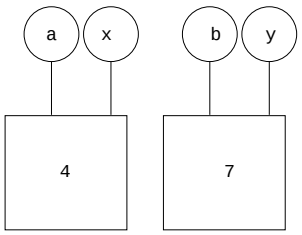
Quand on exécute l'instruction `swap(&a,&b)` ; *dans l'environnement* `e = [a = r1, b = r2]` *et la mémoire* `m = [r1 = 4, r2 = 7]`, *on fabrique l'environnement* `e = [a = r1, b = r2, x = r1, y = r2]` *et la mémoire* `m = [r1 = 4, r2 = 7]`.



Puis, la fonction `swap` *intervertit le contenu des références* r_1 *et* r_2 *et après l'exécution de la fonction, les contenus des références associées aux variables* `a` *et* `b` *ont bien été intervertis.*

Dans cet exemple, on remarquera la syntaxe de la déclaration de l'argument formel final `x`, `int* const x`, *qui interdit l'affectation* `x = t ;` *mais autorise l'affectation* `*x = t ;`. *La déclaration* `const int* x`, *à l'inverse, autoriserait l'affectation* `x = t ;` *mais interdirait l'affectation* `*x = t ;`. *La déclaration* `const int* const x` *enfin interdirait les deux formes d'affectation.*

Exercice 2.14

Qu'affiche le programme suivant ?

```
void swap (int* x, int* y) {
 int z;
 z = *x;
 *x = *y;
 *y = z;}

int main () {
 a = 4;
 b = 7;
 swap(&a,&b);
 printf("%d\n",a);
 printf("%d\n",b);
 return 0;}
```

Dessiner l'état dans lequel s'exécute le corps de la fonction.