puis la fonction `swap` *intervertit le contenu des références* $\texttt{r}_1$ *et* $\texttt{r}_2$ *et après l'exécution de la fonction, les contenus des références associées aux variables* `a` *et* `b` *ont bien été intervertis.*

Exercice 2.11

Que fait le programme suivant ?

```
let swap x y = let z = !x in (x := !y; y := z)
```

2.4.3 C

En C, non plus, le passage par référence n'est pas une construction primitive, mais il peut se simuler en utilisant un mécanisme similaire à celui de Caml. Le type des références susceptibles d'être associées à une valeur de type `T` *dans la mémoire, écrit* `T ref` *en Caml, s'écrit* `T*` *en C. La construction de déréférencement, écrite* `!` *en Caml, s'écrit* `*` *en C. Par exemple, dans l'environnement* $[\texttt{u} = \texttt{r}_1]$ *et dans la mémoire* $[\texttt{r}_1 = \texttt{r}_2, \texttt{r}_2 = 4]$, *la valeur de l'expression* `u` *est la référence* $\texttt{r}_2$ *et la valeur de l'expression* `*u` *est l'entier* `4`.

Si `x` *est une variable, la référence associée à* `x` *dans l'environnement, écrite simplement* `x` *en Caml, s'écrit* `&x` *en C. Par exemple, dans l'environnement* `[x = r]` *et la mémoire* `[r = 4]` *la valeur de l'expression* `x` *est l'entier* `4`, *celle de l'expression* `&x` *est la référence* `r` *et celle de l'expression* `*&x` *est l'entier* `4`. *La construction* `&` *s'applique à une variable et non à une expression quelconque.*

Exercice 2.12

Qu'affiche le programme suivant ?

```
int main () {
 int x;
 int* u;

 x = 4;
 u = &x;
 printf("%d\n",*u);
 return 0;}
```

Il devient possible, avec ces constructions, de fabriquer des états dans lesquels des références sont associées à d'autres références. Il devient alors nécessaire d'enrichir la représentation graphique des états. Quand la mémoire