Programmer en langage C [1]

Adequin. R

16 mai 2013

Sommaire

I

Les tableaux, les pointeurs [2]

1 Les tableaux à un indice

1 Exemple d'utilisation d'un tableau en C

Supposons que nous souhaitons déterminer, à partir de vingt notes d'élèves (fournies en données), combien d'entre elles sont supérieures à la moyenne de la classe. S'il ne s'agissait que de calculer simplement la moyenne de ces notes, il nous suffirait d'en calculer la somme, en les cumulant dans une variable, au fur et à mesure de leurs lecture. mais ici, il nous faut à nouveau pouvoir consulter les notes pour déterminer combien d'entres elles sont superieur à la moyenne ainsi obttenue. il est donc nécessaire de pouvoir mémoriser ces vingt notes.

Pour ce faire, il paraît peu raisonnable de prévoir vingt variables scalaire differentes. Le **tableau** va nous offrir une solution convenable à ce problème, comme le montre le programme suivant.

```
#include <stdio.h>
main() {
   int i, som, nbm;
   float moyenne;
   int t[20];
   for (i = 0; i<20; i++){
      printf("donner la note numero %d : ", i+1);
      scanf("%d", &t[i]);
      printf("\n", i+1);
   }

   for(i = 0, som=0; i<20; i++){
      som += t[i];
   }

   moyenne = som/20;
   printf("\n n moyenne de la classe : %f \n ", moyenne);
   for (i = 0, nbm = 0; i<20; i++){
      if(t[i] > moyenne) nbm++;
   }
   printf("%d eleve ont plus de cette moyenne ", nbm);
}
```

La déclaration :
`int` t[20]
réserve l'emplacement de 20 elements de type `int`. Chaque élément est repéré par sa position dans le tableau, nommée indice. Conventionnelement, en , la premiere position porte le numéro `0`. Ici, donc, nos indicesvont de `0` à `19`. Le premier élément du tableau sera désigné par `t[0]` et le dernier par `t[19]`.

Plus généralement, une notation telle que `t[i]` désigne un élément dont la position dans le tableau est fournis par la valeur `i`. Elle joue le même rôle qu'ue variable scalaire de type `int`.
La notation `&t[i]` désigne l'adresse de cet élément `t[i]` de même que `&n` designe l'adresse de `n`.

2 Quelques règles

Les éléments de tableau

Un élément de tableau est une `lvalue`. Il peut donc apparaître à gauche d'un opérateur d'affectation comme dans :

```
t[2] = 5
```

Il peut aussi apparaître comme opérande d'un opérateur d'incrementation, comme dans

```
t[3]++        --t[i]
```

En revanche, il n'est pas possible, si `t1` et `t2` sont des tableaux d'entiers, d'écrire `t1 = t2 ;` en faite, le langage C n'offre aucune possibilité d'affectation globale de tableaux, comme c'était le cas, par exemple, en Pascal.

Les indices

Un indice peut prendre la forme de n'inporte quelle expression arithmétique de type entier (ou caractère, compte tenu des règles de conversion systématique). Par exemple, si `n, p, k` et `j` sont de type `int`, ces notation sont correctes :

```
t[n-3]
t[3*p-2*k+j%1]
```

il en va de même, si `c1` et `c2` sont de type `char`, de :

```
t[c1+3]
t[c1-c2]
```

La dimension d'un tableau

```
#define N 50
      ........
      int t[N];
      float h[2*N-1];
```

est correcte. en revanche, elle ne le srait pas (en C) si `N` était une constante symbolique définie par `const int N=50`, les expressions `N` et `2*N-1` n'étant alors plus calculables par le compilateur (elle sera cependant acceptée en `C++`)

Débordement d'indice

Aucun contrôle de débordement d'indice n'est mis en place par la plupart des compilateurs. De sorte qu'il est (si l'on peut dire !) de désigner et, donc de modifier, un emplacement situé avant ou après le tableau.

2 Les tableaux à plusieurs indices

1 Leur déclaration

Comme tous les langages, `C` autorise les tableaux à plusieurs indice (on dit aussi à plusieurs dimension). Par exemple, la déclaration :

```
int t[5][3];
```

réserve un tableau de `15 (5 x 3)` éléments. Un élément quelconque de ce tableau se trouve alors repéré par deux indices comme dans ces notation :

```
t[2][3]   t[i][j]    t[i-3][i*j]
```

Notez bien que, là encore, la notation désignant un element d'un tel tableau est une `lvalue`. il n'en ira pas toutefois pas de même notation telle que `t[3]` ou `t[j]` bien que, nous le verrons un peu plus tard, de telles notations aient un sense en `C++`

Aucune limitation ne pèse sur le nombre d'indice que peut comporter un tableau. Seules les limitation de taille mémoire liées à un environnement donné risque de se faire sentir.

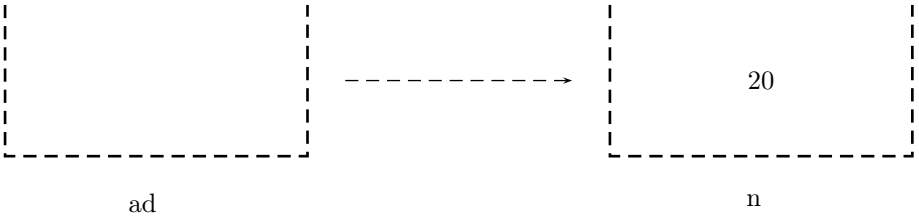
3 Notion de pointeur - les opérateurs * et &

1 Introduction

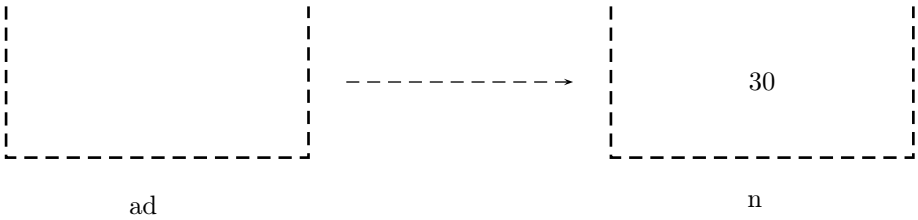
Nous avons déjà été amené à utiliser l'opérateur `&` pour désigner l'adresse d'une `lvalue`. D'une manière dénérale, le langage `C` permet de manipuler des adresses par l'intermédiaired de variable nommées pointeurs. En guise d'introduction à cette nouvelle notion, considerons les instructions :

```
   int * ad ;
int n ;
n = 20 ;
ad = &n ;
ad = 30 ;
```

La première réserve une variable nomée `ad` comme étant un pointeur sur des entiers. Nouverrons que `*` est un opérateur qui designe le contenu de l'adresse qui le suit. Ainsi, à titre mnémonique, on peut dire que cette déclaration signifie que `*ad`, c'est-à-dire l'objet d'adresse `ad`, est de type `int` ; ce qui signifie bien que `ad` est l'adresse d'un entier.
L'instruction :
`ad = &n ;`
Affecte à la variable `ad` la valeur de l'expression `&n`. L'opérateur `&` (que nous avons déjà utilisé avec `scanf`) est un opérateur unaire qui fournit comme résultat l'adresse de son opérande. Ainsi cette instruction place dans la variable `n`. Après son exécution, on peut schématiser ainsi la situation :



L'instruction suivante :
`*ad = 30 ;`
signifie : affecter à la `lvalue *ad` la valeur `30`. Or `*ad` représente l'entier ayant pour adresse `ad` (notez bien que nous disons l'entier et pas simplement la valeur car, ne l'oubliez pas, ad est un pointeur sur des entiers). Après exécution de cette instruction, la situation est la suivante :



Bien entendu, ici, nous aurions obtenu le même résultat avec :
`n = 30 ;`

4 Comment simuler une transition par adresse avec des pointeurs

Nous avons vu que le mode de transmission par valeur semblait interdire à une fonction de modifier la valeur de ses arguments effectifs et nous avions mentionné que les pointeurs fourniraient une solution à ce problème. Nous sommes maintenant en mesure d'écrire une fonction effectuant la ermutation des valeurs de deux variables. Voici un programme qui réalise cette opération avec des valeurs entières :

```
#include <sdtio.h>

void echange (int * ad1, int * ad2);

int main (){

   int a = 10; b = 20;
   printf("avant appel %d %d" , a,b);
   echange(&a,&b);
   printf("apres appel %d %d" , a,b);
}
void echange(int * ad1, int * ad2){
   int x;
   x = * ad1;
   * ad1 = * ad2;
   * ad2 = x;
}
```

```
avant appel 10 20
apres appel 20 10
```

Les arguments effectifs de l'appel de `echange` sont, cette fois, les adresses des variables `n` et `p` (et non plus leurs valeurs). Notez bien que la transmission se fait toujours par valeur, à savoir que l'on transmet à la fonction `echange` les valeurs des expressions `&n` et `&p`.

Voyez comme, dans `echange`, nous avons indiqué, comme arguments muets, deux variables pointeurs destinées à recevoir ces adresses. D'autre part, remarquez bien qu'il n'aurait pas fallu se contenter d'échanger simplement les valeurs de ces arguments en écrivant (par analogie avec la fonction `echange` du chapitre précédent) :

```
   int *x;
x = ad1;
ad1 = ad2;
ad2 = x;
```

Cela n'aurait conduit qu'à échanger (localement) les valeurs de ces deux adresse alors qu'il a fallu échanger les valeurs situées à ces adresse.

5 Un nom de tableau est un pointeur constant

En langage `C`, l'identificateur d'un tableau, lorsqu'il est employé seul (sans indices à sa suite), est considéré comme un pointeur (constant) sur le début du tableau. Nous allons en examiner les conséquences en commençant par le cas des tableaux à un indice ; nous verrons en effet que, pour les tableaux à plusieurs indices, il faudra tenir compte du type exact du pointeur en question.

1 Cas des tableaux à un indice

Supposons, par exemple, que l'on effectue la déclaration suivante :

```
int t[10]
```

La notation `t` est alors totalement équivalente à `&t[0]`. L'identificateur `t` est considéré comme étant de type pointeur sur le type correpondant aux éléments du tableau, c'est-à-dire, ici, `int *`. Ainsi, voici quelques exemples de notations équivalentes :

```
t+1          &t[1]
t+i          &t[i]
t[i]        * (t+i)
```

Pour illuster ces nouvelles possibilités de notation, voici plusieurs façon de placer la valeur `1` dans chacun des 10 éléments de notre tableau `t` :

```
int i ;
for (i=0 ; i<10 ; i++){
   * (t+i) = 1 ;
}
```

```
int i ;
int * p ;
for (p=t, i=0 ; i<10 ; i++, p++){
   * p = 1 ;
}
```

Dans la seconde façon, nous avons dû recopier la valeur représentée par `t` dans un pointeur nommé `p`. En effet, il ne faut pas perdre de vue que le symbole `t` représente une adresse constante (t est une constante de type pointeur sur des entiers). Autrement dit, une expression telle que `t++` aurait été invalide, au même titre que, par exemple, `3++`. **Un nom de tableau est un pointeur constant ; ce n'est pas une `lvalue`.**

2 Cas des tableaux à plusieurs indices

Comme pour les tableaux à un indice, l'identificateur d'un tableau, employé seul, représente toujours son adresse de début. Toutefois, si l'on s'intéresse à son type exact, il ne s'agit plus d'un pointeur sur des éléments du tableau. En pratique, ce point n'a d'importance que lorsque l'on effectue des calculs arithmétiques avec ce pointeur (ce qui est assez rare) ou lorsque l'on doit transmettre ce pointeur en argument d'une fonction ; dans ce dernier cas, cependant, nous verrons que le problème est automatiquement résolu par la mise en place de conversions, de sorte qu'on peut ne pas s'en préoccuper.

À simple titre indicatif, nous vous présentons ici les règles employées par `C`, en nous limitant au cas des tableaux à deux indices.
Lorsque le compilateur rencontre une déclaration telle que :
`int t[3][4] ;`

Index

Bibliographie

[1] Dennis Ritchie Brian W. Kernighan. *C Programming Language*. Prentice Hall, March 1988.

[2] Claudine Delannoy. *Programmer en langage C*. ÉDITIONS EYROLLES, September 5e édition 2009.