Programmation C - Pointeurs, tableaux

Laurent Signac – CC-BY-SA – 04-11-20 1522 a71ea296c878b6ea7f8b

1 Pointeurs

Les pointeurs sont très largement utilisés en C, mais sont parfois délicats à manipuler.

Nous avons vu la notion de variable qui peut contenir une valeur d'un certain type. En mémoire, cette valeur est inscrite à une certaine adresse. On peut donc parler d'adresse d'une variable. Cette adresse est constante, pendant toute la durée de vie de la variable.

Il est possible de connaître l'adresse d'une variable avec l'opérateur `&` :

```
int a=3;
printf("a est stockée à l'adresse %p\n",&a)
```

Un pointeur est simplement une variable qui ne contient une telle adresse. Si le pointeur est prévu pour contenir l'adresse d'une variable de type `int`, son type sera `int *` :

```
int a=3;
int *t;
t=&a;
printf("a est stockée à l'adresse %p\n",t);
```

Le mécanisme de déréférencement permet de manipuler une valeur au travers du pointeur qui la désigne. C'est l'intérêt des pointeurs :

```
int a=3;
int *t;
t=&a;
printf("a contient %d\n",*t);
*t=*t+1;
printf("a contient %d\n",*t);
printf("a contient vraiment %d\n",a);
```

La ligne `*t=*t+1` ajoute 1, non pas dans `t` (pour modifier la valeur de `t`, il faudrait écrire quelquechose comme `t = t + ...`), mais dans la variable pointée par `t`, c'est à dire dans `a`.

2 Passage des paramètres

L'intérêt des pointeurs apparaît particulièrement lors du passage de paramètres.

2.1 Passage par valeur

Le comportement « normal » est un passage par valeurs.

```
#include <stdio.h>
void fonc(int a) {
  printf("a(fonc) reçu : %d\n",a);
  a=a+1;
  printf("a(fonc) modifié : %d\n",a);
}

int main(void) {
  int a=4;
  printf("a(main) avant l'appel : %d\n",a);
  fonc(a);
  printf("a(main) après l'appel : %d\n",a);
```

```
  return 0;
}
```

2.2 Passage par adresse

```
#include <stdio.h>
void fonc(int *pa, int a) {

  printf("*pa(fonc), a(fonc) : %d %d\n",*pa,a);
  a=a*2;
  *pa=*pa*3;
  printf("*pa(fonc), a(fonc) modifiés : %d %d\n",*pa,a);
}

int main(void) {
  int a=5;
  printf("a(main) avant l'appel : %d\n",a);
  fonc(&a,a);
  printf("a(main) après l'appel : %d\n",a);
  return 0;
}
```

Utiliser les pointeurs donne donc un moyen de modifier un paramètre d'une fonction, depuis l'intérieur de la fonction.

3 Tableaux

Un tableau est une collection ordonnée d'éléments homogènes, stockés de manière contiguë en mémoire. Un tableau a une taille fixe. Sa taille doit être connue à la compilation (avant C99).

```
int mtab[20];
int i;
mtab[0]=0;
mtab[1]=1;
for (i=2;i<20;i++) {
    mtab[i]=3*mtab[i-1]-5*mtab[i-2];
    }
for (i=0;i<20;i++) printf("%d\n",mtab[i]);
```

3.1 Dimensions, taille, initialisation

Un tableau a autant de dimensions que de paires de crochets :

```
int tab[4][5][2];
```

En mémoire, les éléments sont rangés de manière séquentielle. Pour connaître l'ordre de rangement des éléments des tableaux multidimensionnels, on fait varier le dernier indice en premier : `t[0][0]`, `t[0][1]`, `t[0][2]`, `t[1][0]`,...

Attention à ne pas confondre la *dimension* d'un tableau, qui correspond au nombre de paires de crochet, et la *taille*, qui correspond au nombre d'éléments (en tout ou le long d'une dimension).

On peut initialiser un tableau à sa déclaration :

```
int t1[]={4,5,6,7,8};
int t2[3]={10,11,10};
int t2[2][3]={{2,3,4},{4,5,6}};
```

3.2 Passage des tableaux en paramètre d'une fonction

Une variable de type tableau est presque équivalente à un pointeur constant vers la première case du tableau.

```c
#include <stdio.h>
void affiche(int t[])  {
  int i;
  for(i=0;i<5;i++) printf("%d ",t[i]);
  printf("\n");
}
void fonc(int t[]) {
  int i;
  printf("t(fonc) reçu : ");
  affiche(t);
  for(i=0;i<5;i++) t[i]=t[i]*t[i];
  printf("t(fonc) modifié : ");
  affiche(t);
}

int main(void) {
  int t[]={1,2,3,4,5};
  printf("t(main) avant l'appel : ");
  affiche(t);
  fonc(t);
  printf("t(main) après l'appel : ");
  affiche(t);
  return 0;
}
```

De fait, si on passe un tableau en paramètre d'une fonction, on peut modifier les valeurs stockées dans le tableau depuis l'intérieur de la fonction.

3.3 Connaître la taille d'un tableau

Il est «habituel» que la taille des tableaux soit stockée dans des variables annexes. Il est toutefois possible de la connaître en utilisant l'opérateur `sizeof`.

`sizeof(t)` renvoie la taille en octets du tableau et `sizeof(t[0])` la taille en octets du premier élément. Le nombre d'éléments est donc `sizeof(t)/sizeof(t[0])`

```c
#define LEN(x) (sizeof(x)/sizeof(x[0]))

int main(void) {
  int t[] = {1,2,3,4};
  printf("Taille : %ld\n",LEN(t));
  return 0;
}
```

4 Pointeurs de fonctions

Utiliser des pointeurs de fonction permet d'écrire du code générique, de faire de la programmation événementielle (enregistrement des *callbacks*)…

On déclare un pointeur de fonction ainsi :

```
type_retour (* nom_var) (signature);
```

Le pointeur est ensuis utilisé ainsi :

```
...=(* nom_var)(params...)
```

Exemple de déclaration de la variable `pf`, pointeur vers une fonction prenant en paramètre un `int` et un `float` et renvoyant un `int`, puis utilisation de `pf` :

```
int a;
int (* pf) (int, float);
a=(*pf)(3,4.5);
```