1 Explications

Il est possible en C de déclarer en une seule instruction plusieurs variables de même type. Ces variables seront rangées dans la mémoire les unes à la suite des autres : c'est ce qu'on appelle un tableau. Exemple avec un tableau d'entiers :

```
1  int tab[3]; /* reserve sur la pile trois entiers */
2  tab[0] = 1; /* le premier s'appelle tab[0] */
3  tab[1] = 2; /* le deuxieme s'appelle tab[1] */
4  tab[2] = 3; /* le troisieme s'appelle tab[2] */
5  tab[3] = 4; /* ATTENTION : pas de controle de debordement !*/
6
7  /* il est possible de declarer et definir en une seule instruction :*/
8  int tab[]={1,2,3};
9  /* il devient optionnel de mettre la taille dans les [] car le
10 compilateur peut compter les elements presents entre les accolades */
```

Il est possible de passer les tableaux en paramètre lors de l'appel de fonction. Dans ce cas attention, il faut deux variables : une variable qui identifie le début du tableau, et une variable qui donne la taille du tableau (puisque pas de controle de débordement). Attention, le type de la variable `tab` n'est pas un type "normal", c'est un *pointeur*. C'est en fait une valeur entière contenant l'adresse mémoire du tableau. Si la valeur pointée est de type `type`, le type du pointeur est `type*`. Exemple avec un tableau de caractères (les deux fonctions sont strictement équivalentes) :

```
1  void affiche_tableau(char tab[], int taille)
2  /* remarquez qu'il n'y a rien entre les [] */
3  {
4          int i;
5          for(i=0;i<taille;i++)
6                  printf("%d␣<->␣%c\n", tab[i], tab[i]);
7  }
8  void affiche_tableau_2(char* tab, int taille)
9  {
10         int i;
11         for(i=0;i<taille;i++)
12                 printf("%d␣<->␣%c\n", tab[i], tab[i]);
13 }
```

Le pointeur ne donnant comme information que le début du tableau, il est toujours nécessaire d'utiliser un deuxième paramètre pour la taille du tableau.

Exception a cette règle : si par convention le tableau contient un marqueur de fin. Par exemple, si on manipule des tableaux d'entiers positifs, il est possible d'utiliser une valeur négative pour marquer la fin du tableau :

```
1  /* Parametre : tableau contenant comme marqueur de fin la valeur -1 */
2  void affiche_tableau(int tab[])
3  {
4          int i;
5          for(i=0;tab[i]!=-1;i++)
6                  printf("%d␣", tab[i]);
7          printf("\n");
8  }
```

Cela ne marche évidemment que si la convention est respectée : si on écrit du code qui appelle la fonction avec un tableau ne contenant pas -1, la fonction risque de rentrer dans une boucle infinie et de créer un débordement mémoire.

Par convention, en C, une **chaine de caractères** est un tableau de caractères contenant comme marqueur de fin le caractère $'\backslash 0'$. C'est ce qui permet par exemple à printf de les afficher (format %s) sans que l'on passe la taille en paramètre.

2 Questions de compréhension

1. Code 1 (rappel)

```
1  #include <stdio.h>
2
3  void f(int a)
4  {
5          a=10;
6  }
7
8  int main()
9  {
10         int a=20;
11         f(a);
12         printf("%d\n",a);
13         return 0;
14 }
```

   Qu'affiche ce programme ? Que faut il modifier si l'on veut que l'appel de la fonction ait un effet sur l'affichage du main ligne 12 ? Dessinez la pile.

2. Code 2

```
1  #include <stdio.h>
2  void affiche_tableau(int tab[], int taille)
3  {
4          int i;
5          for(i=0;i<taille;i++)
6                  printf("%d␣",tab[i]);
7          printf("\n");
8  }
9  void f(int a[])
10 {
11         a[0]=11;
12         a[1]=12;
13         a[2]=13;
14 }
15 int main()
16 {
17         int a[]={1,2,3};
18         f(a);
19         affiche_tableau(a,3);
20         return 0;
21 }
```

   Qu'affiche ce programme ? Dessinez la pile.

3. Code 3

```c
#include <stdio.h>
void f(int a[])
{
        a[0]=10;
}
int main()
{
        int a=1;
        f(&a);
        printf("%d\n",a);
        return 0;
}
```

Qu'affiche ce programme ? Dessinez la pile.

4. Soit les déclarations :

```c
int a;
char tab[10];
```

Lorsqu'une expresssion avec pointeurs vous est donnée, réécrivez la sans, et vice versa :

(a) `*(&a)`
(b) `*tab`
(c) `*(tab + 0)`
(d) `(*tab) + 1`
(e) `&(tab[0])`
(f) `&(tab[i])`
(g) `++tab[i]`

3 Occupation mémoire

Il est demandé dans cet exercice de représenter en mémoire les données déclarées dans un programme, ainsi que leurs différentes valeurs, à un moment donné de l'exécution. Pour cela, vous représenterez l'occupation des données en mémoire dans un tableau à 3 colonnes comme montré ci-dessous :

identificateur	adresse	valeur
a	?	?

Pour déterminer les adresses, on fera les approximations suivantes :
— les données sont réservées en mémoire dans l'ordre de leur déclaration, par adresses croissantes
— la première adresse démarre à 10001
— l'architecture est 64 bits, les entiers sont codés sur 32 bits et les pointeurs sur 64.

Le programme est donné ci-dessous :

```c
#include <stdlib.h>
#include <stdio.h>
int main()
{
        int a = 10;
        int b = 5;
        int tab[3] = {1,2,3};
        int *p_int;
```

```
9           /* representer l'occupation memoire */
10
11          tab[0] = a;
12          *(tab + 1) = b;
13          p_int = tab + 2;
14          /* representer l'occupation memoire */
15
16          *p_int = *(p_int - 1);
17          --p_int;
18          *p_int = *(p_int - 1);
19          --p_int;
20          *p_int = *(p_int + 2);
21          /* representer l'occupation memoire */
22
23          printf("%d\t%d\t%d\t%d\t%d\n",a,b,tab[0],tab[1],tab[2]);
24          /* donner l'affichage */
25
26          return EXIT_SUCCESS;
27  }
```

4 Bonus

Attention : sizeof ne sert **pas** et ne doit **pas** servir à calculer la taille d'un tableau. Exemple à tester pour vous convaincre :

```
1   #include <stdio.h>
2   void f(int tab[])
3   {
4           printf("taille␣(dans␣la␣fonction)␣:␣%lu\n",sizeof(tab));
5   }
6   int main()
7   {
8           int tab[]={1,2,3};
9           printf("taille␣(dans␣le␣main)␣:␣%lu\n",sizeof(tab));
10          f(tab);
11          return 0;
12  }
```