

[LC060]

POINTEURS

* élément pointé par
& adresse de

Exemple 1 :

```
{
int ivar, *iptr;
iptr = &ivar;
ivar = 421;
printf("adresse de ivar : %04X\n", &ivar);
printf("valeur de ivar : %d\n", ivar);
printf("valeur de iptr : %04X\n", iptr);
printf("valeur pointée : %d\n", *iptr);
}
```

complément des 0.
4 chiffres
X = hexadecimal

ivar

adresse de ivar : 3B5C
valeur de ivar : 421
valeur de iptr : 3B5C
valeur pointée : 421

Remarque :

L'affectation : **ivar = 421;** peut être remplacée par : ***iptr = 421;**

Exemple 2 :

Question : Quel est le résultat du programme suivant ?

```
{int ivar, *a, *b;
a = &ivar;
ivar = 421;
b = a;
*a = 517;
printf("%d %d\n", *a, *b);
}
```

a ptr vers ivar
b ptr vers ivar.

ivar ~~421~~ 517.

printf ("%d %d\n", *a, *b); 517 517.

Exemple 3 :

Question : Que pensez-vous du programme suivant ?

```
{int *iptr;
iptr = (int *) malloc( sizeof(int) ); if (iptr == 0) { erreur(1); }

*iptr = 421;
printf ("il s'agit du jeu de %d!\n", *iptr);
}
```

pb!!! on a réservé un ptr, pas un int

RÈGLES D'UTILISATION DES POINTEURS [TCU p 198]

Règle 1 :

Il faut toujours affecter une adresse à un pointeur avant de l'utiliser.

Règle 2 :

Il faut de plus que cette adresse corresponde à une zone mémoire prévue pour cet usage :

- Soit l'adresse d'une variable (ou d'un élément de tableau) de même type. (voir exemples 1, 2, 4)
- Soit une adresse obtenue par une fonction d'allocation. (voir plus loin : allocation dynamique)

ARITHMÉTIQUE DES POINTEURS (TCU p 200)

Règle 3 :

Toute modification d'un pointeur par addition ou soustraction d'un entier tient compte implicitement de la taille (sizeof) de l'élément pointé.

Exemple-exercice 4 :

```
#define NB 7
```

```
-----
```

```
-----
```

```
{
```

```
int list[NB] = {421, 53, 1806, -37, 216, -21, 0} ; /* list est un tableau d'entiers */
```

```
int *ptlist, i ; /* ptlist est un pointeur vers un entier ; i est un entier */
```

```
printf ("%d %d\n", sizeof (int), sizeof (int *)) ; 2 4 /* dépendent compil. & machine */
```

```
ptlist = list ; /* le nom d'un tableau (seul) : C'est l'adresse de base du tableau */
```

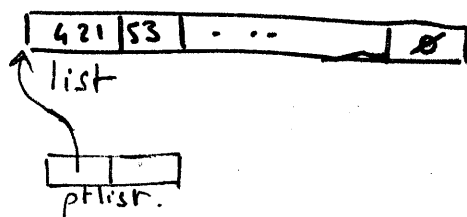
```
printf ("tailles : %d %d %d\n", sizeof (list), sizeof ptlist, sizeof *ptlist) ;
```

```
for (i = 0 ; i < NB ; i++) /* la boucle d'affichage */
```

```
printf ("adr:%04X val:%d \n", ptlist + i, *(ptlist + i)) ;
```

```
}
```

1°) Qu'affiche donc en fait ce programme ?



tailles :	14	4	2
adr : 06A8		val :	421
adr : 06AA		val :	53
adr : 06AC		val :	1806
adr : 06AE		val :	-37
adr : 06B0		val :	216
adr : 06B2		val :	-21
adr : 06B4		val :	0

2°) Réécrire la boucle d'affichage en incrémentant le pointeur à chaque tour.

```
for (i = 0 ; i < NB ; i++, ptlist++)
    printf (...);
```

3°) Idem en supprimant l'indice i, la valeur 0 finale servant pour le test d'arrêt.

```
while (!(*ptlist))
{
    printf (...);
    ptlist++;
}
```

POINTEURS ET TABLEAUX

Règle 4 : Le nom d'un tableau joue en fait le rôle d'un pointeur **CONSTANT**

Exemple 5 : On peut remplacer `ptlist` par `list` dans la boucle n° 1 de l'ex. précédent, mais pas dans les deux autres boucles.

~~`list ++`
`list =`~~

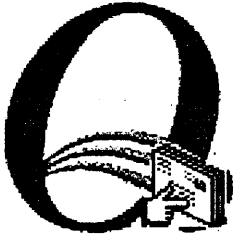
Règle 5 : Un pointeur peut être utilisé comme un tableau.
C'est à dire avec une notation indicielle.

<code>ptlist[i]</code>	\Leftrightarrow	<code>*(ptlist + i)</code>
<code>&ptlist[i]</code>	\Leftrightarrow	<code>ptlist + i</code>
<code>ptlist[0]</code>	\Leftrightarrow	<code>*ptlist</code>
<code>&ptlist[0]</code>	\Leftrightarrow	<code>ptlist</code>

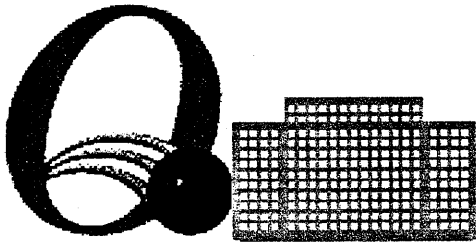
Exemple - exercice 7 : Réécrire la boucle d'affichage de l'exemple 4 avec la notation indicielle pour `ptlist`.

```
for (i = 0 ; i < NB ; i++)  
    printf("adr : %04x    val : %0d\n", &ptlist[i], ptlist[i])
```

UTILITES DES POINTEURS




Passage d'arguments par adresse à une f^h afin que celle-ci puisse les modifier.

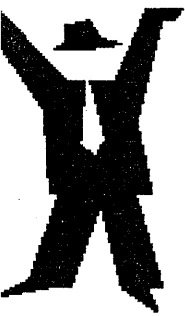


Passage d'arguments tableaux ou variables structurées à une f^h .



 ptr courant. sur un tableau.

(exemple 4 : boucles du 2° et du 3°)



allocat° dynamique de tableaux
(ou de variables).

(voir plus loin)

CHAÎNES

(cf fichier xmpchain.c et/ou xmpchntc.c)

NB : Dans ces exemples on suppose (pour simplifier) les pointeurs stockés sur 2 octets (soit 4 chiffres en hexa) d'où le format %04X utilisé pour les afficher. Suivant la machine effectivement utilisée, il faudrait en fait utiliser un format long %08X ou un format spécifique %p. De même l'affichage des tailles (sizeof) nécessitera parfois un format long %ld. On suppose en outre les déclarations des variables toutes globales.

- On peut déclarer et/ou définir une chaîne de caractères

→ comme un tableau :

*presque
pareil
sauf
sizeof affectat*

*possible de ne faire un
tab à l'adresse d'octet 4b*

```
char tab[28] = "ceci est une chaîne"; char a[2] = "lu", b[7] = "nettes"
```

```
char tac[] = "en voici une autre"; ← (initialiseur indispensable ici)
```

```
printf("%d %d %d\n", sizeof(tab), sizeof(a), sizeof(b));
```

```
printf("%d %s %s\n", sizeof(tac), b, a);
```

```
printf("%04X %04X\n", tab, tac);
```

```
printf("%s, %s\n", tab, tac);
```

→ ou comme un ptr. :

```
char *tad = "et encore une", *taf;
```

```
printf("%d %04X %04X\n", sizeof tad, tad, taf);
```

- On peut modifier la valeur d'un pointeur en lui affectant une chaîne : le pointeur reçoit l'adresse de la chaîne :

```
taf = tad; printf("%04X\n", taf);
```

```
tad = "chaîne toujours"; printf("%04X\n", tad);
```

```
printf("%s %s\n", taf, tad);
```

- Mais pour un tableau c'est IMPOSSIBLE... utiliser strcpy

- Les fonctions str... et mem... fonctionnent avec pointeur ou tableau

```
strcpy(tab, tac); printf("%04X\n", tab);
```

/* Attention! La copie précédente suppose que tab pointe vers une zone suffisamment grande pour recevoir la chaîne tac.

Est ce bien vrai ici ?

```
printf("%s\n", tab);
```

```
printf("%s\n", strncpy(taf, tad, 8));
```

/* Attention! strncpy ne copie l'octet nul que s'il est atteint */

```
printf("%d\n", memcmp(tad, taf, 11));
```

memcmp comparent deux chaînes de caractères suivant l'ordre lexicographique. Dans cet exemple on ne compare que les 11 premiers caractères des deux chaînes. Seul le signe du résultat est imposé : (<0) signifie que la première chaîne est avant la deuxième. (>0) signifie qu'elle est après. (=0) signifie que les 11 caractères sont identiques dans les deux chaînes */

TABLEAUX DYNAMIQUES

Les pointeurs permettent de créer des tableaux dont la *taille* n'est pas *prédéfinie*...

... grâce aux fonctions d'*allocation mémoire* : (déclarées dans `alloc.h` et dans `stdlib.h`)

```
void * malloc(size_t taille) ;          /* allocation mémoire taille = nbre octets */
void * calloc(size_t nbelt, size_t taille) ; /* allocation mémoire : nbelt × taille */
void * realloc(void * bloc, size_t taille) ; /* ajustt taille mém. allouée pour bloc */
void free(void * bloc) ;                /* libération mémoire allouée pour bloc */
```

Les 3 premières fonctions retournent l'adresse (void *) de la zone allouée sauf si l'allocation est impossible. Auquel cas elles retournent NULL.

Exemple 8 :

```
ptr = (int *) malloc(100 * sizeof(int));
ptr = (int *) calloc(100, sizeof(int));
ptr = (int *) realloc(700 * sizeof(int)); // si l'allocation n'a pu se faire, le ptr
// passe à NULL et ici, on panique tout.
float * ptr2 = NULL;
size_t dim = 215;
int i;

ptr = (float *) calloc(dim, sizeof(float));
if (ptr != NULL)
{
    for (i = 0; i < dim; i++)
    {
        .....; ptr[i] = .....;
    }
    .....
    .....
    dim = 314; ...
    ptr2 = (float *) realloc(ptr, dim * sizeof(float));
    if (ptr2 != NULL)
    {
        .....
    }
    .....
    .....
    if (ptr) { free(ptr); }
    .....
}
```

*Ami ! Toujours vérifieras
Que ton texte ne déborde pas.
Fins de chaînes en particulier
Tâche de ne pas oublier :
Cet octet nul qui termine la chaîne
Doit pouvoir s'y loger sans peine.*

*Un pointeur que tu alloueras
Qu'il n'est pas NULL testeras.
Quant à la mémoire allouée
N'oublie pas de la libérer !...*

(à suivre...)

©Jean-Paul BLANC

L'allocation dynamique n'est à utiliser que si l'on ne peut pas réserver à l'avance (à la compilation) la place nécessaire pour un tableau. Il est en général plus simple de déclarer un tableau sous la forme :

```
int tab[72] ;
```

que de déclarer un pointeur :

```
int * ptab ;
```

et de lui allouer ensuite de la mémoire :

```
ptab = (int *) calloc(72, sizeof(int));
```