```
I) Réservation de place

  En C, il n'y a pas de type tableau proprement dit, juste une définition de
nom (étiquette), une arithmétique sur les pointeurs et pas de contrôle
systématique de indice/taille. Si l'indice dépasse la dimension du tableau, on
va taper dans les variables suivantes !!

(cf programme C demo_debordement_tableau).

  Pour stocker les tableaux (déclarés statiquement), on réserve une zone
mémoire de taille

  nb_elements * sizeof(element)

et on met en étiquette le nom du tableau (= l'adresse du premier element).

  En indicant à partir de 0, le premier élément t[0] est stocké
à l'adresse de la zone mémoire réservée pour le tableau. L'élément de rang i
est stocké à cette même adresse + i* sizeof(type_du_tableau).

Déclaration avec ou sans initialisation :

int puissance2[5]; /* elements puissance[0] a puissance[4] */
int puissance2[5] = {1,2,4,8,16}; /* puissance2 [0] : 1 puissance2 [1] = 2 */
int puissance2[5]={1,2,4};  /* initialisation partielle */

                .bss
puissance2:     .skip 20

                .data
puissance2:     .word 1
                .word 2
                .word 4
                .word 8
                .word 16

                .data
puissance2:     .word 1
                .word 2
                .word 4
                .skip 8

II) Operateur d'indexation

  Indicage à partir de 0.

  L'opérateur d'addition d'un entier à un pointeur n'a de sens que dans la
mesure où le pointeur contient l'adresse d'un élément de tableau.

  L'addition ou la soustraction d'un entier e à un pointeur consiste à
se déplacer de e éléments dans le tableau.

register long *pt;

  pt +=  2;             /* decrit registre += 8, soit 2*sizeof(long) */

Hors déclarations, l'opérateur d'indexation t[i] n'est qu'un raccourci
d'écriture pour * (t+i). Par definition t est égal à &t[0].

III) Exemple
```



```
long tab [5];
long *pt;
register int i;
        i = 3;
        tab [2] = 5;
        pt = &(tab[i]);         /* ou pt = tab+i */
        *pt = pt[1];            /* tab [3] = tab [4] */

Expansion :     sizeof (long) = 4

        i = 3;
        *(tab + 2) = 5;         /* adresse = tab + 2 * 4 */
        *&pt = tab + i          /* adresse = tab + i * 4 */
        **&pt = * (*&pt + 1)    /* adresse gauche = reg_pt,
                                   droite = reg_pt + 1 * 4 */

Remarque : fois 2puissanceX = décalage gauche X bits

        .bss
tab:    .skip   20              @ 20 = 5 elements de 4 octets

        .text
        .global main
main:
@ i = 3
        mov     r1,#3           @ i : stockage dans r1
@ * (tab + 2) = 5
        mov     r0, #5
        ldr     r4, =tab        @ tab est une etiquette
        str     r0,[r4,#8]      @ Mem [r4 + 2 * 4] = r0
@ *&pt = tab + i
        ldr     r4, =tab
        add     r4, r4, r1, LSL #2      @ tab + i (* sizeof)
        ldr     r5, =pt         @ r5 = &pt
        str     r4, [r5]        @ *&pt = r4
@ **&pt = **&pt + 1
        ldr     r5, =pt
        ldr     r4, [r5]        @ contenu de pt : adresse de l'entier pt[1]
        ldr     r0, [r4 , #4]   @ r0 = *(pt+1)  :  +1 *sizeof donne +4
        str     r0, [r4]        @ r4 contient deja *&pt --> **&pt = r0
        .ltorg

IV) Parcours de tableau

  Deux méthodes :

  Par indice (calcul à chaque tour de boucle)
  for (i=0; i<TAILLE;i++)
        t[i] = ...;

  Par pointeur :
  for (pt = t; pt < t+TAILLE; pt++)
        *pt  = ...;

V) Elements de taille autre qu'une puissance de 2

  Note : sizeof (element) intègre l'espace perdu dû aux contraintes
d'alignement internes à l'objet et entre éléments juxtaposés dans un tableau.

  Exemple : une structure
```

```
struct ascii_nombre {
   unsigned char ascii;         /* Un caractere 0 a 9, A a F ou a a f */
   unsigned long int valeur;    /* Valeur entre 0 et 15 inclus */
   }

  sizeof (unsigned long) : 4
  sizeof (unsigned char) : 1
  sizeof (ascii_nombre)  : 8    /* on perd 3 octets d'alignement entre 2 */

VI) Les chaînes de caractères en C

Considérées comme des tableaux de char.

Taille de la chaîne non stockée : caractère null '\0' (code ASCII 0) marque
la fin de chaîne.

char ch[] = "abc";    /* signifie  char ch[4] = {'a','b','c',0} */

char nom[10] = "moi"; /* char nom[10] = {'m','o','i','\0',0,0,0,0,0,0} */

strlen [moi] retourne 3 (le nombre de caractères sans le 0 de fin de chaîne)

Réservation de place :

ch:     .byte   'a'             /* ou  ch:      .asciz  "abc"  */
        .byte   'b'
        .byte   'c'
        .byte   0

nom:    .byte   'm'             /* ou nom:      .ascii  "nom"  */
        .byte   'o'                              .byte   0
        .byte   'i'                              .skip   6
        .byte   0
        .skip   6
```