Adresse et tableaux en C/C++

– Dans le monde merveilleux de l'algorithmique :
tab : **tableau[1..10] de entier**
Création de 10 variables contiguës de type entier : tab[1], tab[2], .., tab[10]
Et tab désigne la variable composée des 10 variables précédentes

– **Attention :**
Les tableaux du C et du C++ ne correspondent pas à une mise en œuvre fidèle de ce concept
…

int tab[10];
Création de 10 variables contiguës de type entier :
tab[0], tab[1], .., tab[9]

Mais l'identificateur tab utilisé seul désigne l'adresse de tab[0]

tab correspond à &tab[0]
Plus généralement :
&tab[i] est équivalent à tab+i
et tab[i] est équivalent à *(tab+i)

Attention : Pas de contrôle des débordements possibles

ex : tab[10]=5; //Qu'est-ce que tab[10]???

Pensez-vous que le fragment de programme suivant soit correct ?

```
char t1[3];
char t2[3];
…
t1=t2;
```


Remarque :

La taille d'un tableau doit correspondre à une constante entière
(ie) valeur connue à la compilation
(C89et90,C++98)

```
int tab[4];
int tabat[]={2,4,6}; // taille 3 déduite de l'initialisation

const int n=5;   //n initialisé (définitivement)
                 //avec une constante entière
double dab[n];   //Oui en C++98,
                 //mais pas en C89 et 90
```


Passage d'un tableau en argument d'une fonction C ou C++

```
void proc(int tab[2])
{
  …
}
```
On aurait pu écrire :
```
void proc(int tab[])
{
  …
}
```
ou alors
```
void proc(int * tab)
{
  …
}
```

tab paramètre formel
de type adresse du premier élément d'un tableau de **int**
(ie) adresse d'une variable de type **int**

Au moment de l'appel :
Une adresse (de premier élément d'un tableau) est fournie et copiée sur la pile
ex:
```
int tabat[2];
proc(tabat);
```


- Mise en oeuvre de procédures en C ou C++

– Mise en œuvre du passage en paramètre **donnée** d'un tableau, précisée par le qualificatif **const**
void proc(**const** int *tab) {…}
ou
void proc(**const** int tab[]) {…}

– Mise en œuvre du passage en paramètre **donnée-résultat** ou **résultat** d'un tableau

void proc(int *tab) {…}
ou void proc(int tab[]) {…}

Retour d'un tableau dans une fonction C ou C++

- Que pensez-vous de la fonction suivante ?

```
int * doubler(const int tab[3])
{
  int res[3];
  for (int i=0;i<3;i++)
    res[i]=2*tab[i];
  return res;
}
```


Arrgghhh! Retour de l'adresse d'un tableau local à la fonction (détruit après l'appel à la fonction)

Seule possibilité :
Retourner un tableau alloué dans le tas !

```
int * doubler(const int tab[3])
// Précondition : tab tableau d'(au moins) 3 int initialisés
// Résultat : adresse d'un tableau de 3 int
//            Charge à l'utilisateur de stocker cette adresse
//            dans une variable et d'invoquer l'opérateur
//            delete []sur cette variable quand il n'aura plus
//            besoin du tableau…
 {
   int *res=new int[3];
   for (int i=0;i<3;i++)
     res[i]=2*tab[i];
   return res;
 }
```


Exemple d'utilisation :

```
int montab[]={1,2,3};
int *ptr;
ptr=doubler(montab);
ptr[0]=4;
 ....
delete [] (ptr);
```


Chaînes de caractères

Les tableaux de char C/C++ servent également à manipuler des chaînes de caractères
(de taille inférieure à celle du tableau)

Possibilité d'utiliser des fonctions de la bibliothèque standard si '\0' en fin de la chaîne (convention)

Création d'un tableau de 6 char pour y stocker des chaînes d'au plus 5 char :

```
char ch[6];
ch[0]='i'; ch[1]='f'; ch[2]='5'; ch[3]='\0';
std::cout << ch << std::endl;
```


Initialisation à la définition des chaînes de caractères

1. Comme les autres tableaux
   char ch1[]={'b','i','c','h','e','\n','\0'};
   ch1 de taille 7

'b'	'i'	'c'	'h'	'e'	'\n'	0

   char ch2[8]={'o','u','i',0};

'o'	'u'	'i'	'\0'				

2. Avec une chaîne littérale
   char ch3[]="loup";
   ch3 de taille 5

'l'	'o'	'u'	'p'	'\0'

Attention :
char a;
Bien faire la différence entre a, 'a' et "a"

Manipulation des chaînes de caractères :

- Bibliotheque string
  (#include <cstring> (C++) ou (#include <string.h> (C))
- Par les fonctions de lecture/écriture
  sur entrée/sortie standard (iostream, stdio)

```
char ch[10];
std::cin >> ch ;
std::cout << ch << std::endl;

std::scanf("%s",ch);
std::printf("%s",ch);
```

Attention aux dépassements mémoire!
std::scanf("%9s",ch); // Un moyen de s'en prémunir
std::fgets(ch,10,stdin)