Plan

Les tableaux en C

En mémoire, un tableau est un bloc d'objets consécutifs de même type.

Sa déclaration est :

- similaire à une déclaration de variable ;
- il faut indiquer le nombre d'éléments entre `[]`.

Quelques exemples :

```
char s[22]; /* s tableau de 22 caract\'eres */
/* t1 tableau de 10 entiers longs et
   t2 tableau de 20 entiers longs */
long int t1[10], t2[20];
#define N 100
int tab[N/2];
```

Points importants :

- la taille d'un tableau est une constante **qui doit être calculable à la compilation :**

```
char tab[] = "123" ;

int main(void)
{
        return 0 ;
}
```

- les indices dans un tableau commencent en 0 ;

  Les indices d'un tableau de taille `N` vont de `0` à `N-1`.

Définition d'un tableau lors de sa déclaration

L'initialisation d'un tableau se fait :

- par des valeurs constantes placées entre {} séparées par des virgules (,) ;
- si il n'y a pas assez de valeurs : l'espace mémoire restant est soit indéterminé soit mis à 0 ;
- Par exemple : `int t[4] = { 1, 2, 3, 4 };`
- il n'y a pas de facteur de répétition.

Manipulations élémentaires sur les tableaux

Accès à un élément de tableau par opérateur d'indexation ;

- Syntaxe :
  expression $\Leftarrow$ *identificateur-de-tableau* `[` *expression*$_1$ `]`
- Sémantique :
  - *expression*$_1$ délivre une valeur entière ;
  - *expression* délivre l'élément d'indice *expression*$_1$;
  - *expression* peut être une valeur de gauche comme dans l'exemple `x = t[k]; t[i+j] = x;`.

L'identificateur `t` n'est pas une variable. Il est associé à une adresse constante correspondant au début de la mémoire allouée au tableau. En mémoire, on a les octets :

	t				
...	1	2	3	4	...

Comparer 2 identificateurs de tableau revient à comparer 2 adresses et non pas les objets stockés à ces adresses. De même, affecter quelque chose à cet identificateur `t = ...` n'a pas de sens.

Passage d'un tableau en paramètre d'une fonction

Puisque l'identificateur d'un tableau n'est pas une variable, quelle copie est faite lors du passage d'argument suivant :

```
void fct(int tib[]){
   tib[0] = 1 ;
   return ;
}
```

```
int main(void){
    int tab[2] = { 0, 1} ;
    fct(tab) ;
    return tab[0] ;
}
```

C'est l'adresse qui est copiée. Ceci implique que la fonction principale retourne 1 dans notre exemple.

Dans `fct`, `tib[0]` fait référence à la première *cellule mémoire* définie dans le tableau local à la fonction principale.

Nous étendrons ce principe (passage de paramètre par adresse) aux autres types en utilisant la notion de pointeur.

Tableau bidimensionel

Bien que stockés linéairement, les tableaux peuvent être définis comme multidimensionel :

```
char tab[3][4]={"123","456","789"} ;


int
main
(void)
{
   return 0 ;
}
```

La sémantique est la même que pour le cas monodimensionnel :

`tab[3][0] = tab[3][0]++` 1

Les structures

Une *structure* est le regroupement de plusieurs variables de types différents dans une même entité.

- cet objet est composé d'une séquence de membres de types divers ;
- chaque membre porte un nom interne à la structure ;
- le type des membres peut être quelconque (imbrication) ;
- les membres sont stockés de manière contiguë en mémoire ;
- déclaration :

```
struct identificateur_du_modèle
{
   type liste_identificateur_de_membre ;
   type liste_identificateur_de_membre ;
   ...
};
```

- déclaration de variable d'un type structure :
  `struct` *identificateur_de_modèle liste_identif_variable* ;
- définition et déclaration simultanées de variables :

```
struct  identificateur_de_modèle  {
    type  liste_identificateur_de_membre    ;
    type  liste_identificateur_de_membre    ;
  ...
} liste_identificateur_de_variable    ;
```

- le nommage de la structure est alors facultatif ;
- accès à un membre : opérateur `.` de sélection de champs
  identificateur_de_variable `.` *identificateur_de_membre* ;

```
struct mastructure {
      char o ;
      int  six ;
} ;
struct mastructure mavariable ;
mavariable.o='o' ; mavariable.six = 6 ;
```

Structure en memoire

Ma structure

```
struct personne {
        int numero;
        char nom[8];
        char age;
};
struct personne maPersonne;
maPersonne.numero = 123456;
strcpy(maPersonne.nom, "DUPONT");
maPersonne.age = 43;
```

Champ	numero				nom								age
Adresses	0	1	2	3	4	5	6	7	8	9	10	11	12
Valeurs	123456				D	U	P	O	N	T	\0	...	43

Ne pas confondre C et ses héritiers

Attention : C n'est pas un langage orienté objet et donc, il n'y a pas de constructeur en C.

Il n'y a pas d'initialisation "générique" associée à un type. Le code suivant n'est pas du C valide :

```
struct adresse
{
  int num = 36 ;
  char rue[40] = "Quai des Orf\'evres";
  long int code = 75001;
  char ville[20] = "Paris" ;
};
```

Copie et affectation d'une structure comme un tout

Contrairement aux tableaux, l'affectation

```
struct personne bobo;
int main(void){
  bobo = bibi ;
  return 0 ;
}
```

est possible et provoque une copie physique des données de l'espace mémoire associé à `bibi` dans celui associé à `bobo`.

En conséquence, on peut :

- passer des structures en argument de fonction (copie) ;
- utiliser une structure comme valeur de retour de fonction ;
- mais C étant un langage de bas niveau, les structures ne se comparent pas.