TD 03 Programmer en C (3)

A faire TD 3.

Table des matières

1 Tableau

Du point de vue algorithmique, un tableau est une **collection** de données du même type, qui sont accessibles à travers un **indice entier**. Les indices possibles forment un intervalle.

En C, un tableau est une zone en mémoire contenant des données de même type, consécutives. Notation `tableau[expression]`. L'indice de la première donnée est 0.

1.1 Déclaration d'un tableau, avec réservation

Dans le cas d'une **déclaration de variable automatique ou statique**, la déclaration d'un tableau mentionne sa taille (en nombre d'éléments), parce qu'elle sert à réserver de l'espace pour "loger" le tableau.

```
int    nb_jours[12];
struct Personne equipe[5];
```

La taille est en général une **constante connue à la compilation**.

La taille peut aussi être déduite des valeurs initiales

```
int t[] = { 10, 3, 200};
```

L'espace est réservé, soit dans la pile d'exécution (pour les variables automatiques), soit dans la zone statique (avec les variables globales).

Observation

- définir un type structure quelconque (nom, prenom, age ?)
- faire afficher sa taille (operateur `sizeof`)
- déclarer un tableau de 5 structures
- faire afficher l'adresse des 5 structures

Pour afficher une adresse, il faut

- utiliser le format "%p" (pointer)
- "caster" l'adresse en la précédant par `(void *)`

exemple

```
int a;
printf("adresse de a = %p", (void *) &a);
```

Le type `(void *)` est un type "générique" de pointeur, dont la représentation est assez grande pour contenir tous types de pointeurs (ils ne sont pas forcément de la même taille, mais c'est une autre histoire).

C autorise aussi la déclaration de tableaux dont la taille est fixée par une expression (VLA = **variable length array**). **Important** : vérifier la taille, avant de réserver le tableau.

1.2 Déclaration d'un paramètre de type tableau

Pour la déclaration d'un **paramètre** (dans le prototype d'une fonction), la taille est facultative

```
int somme_tableau(int tableau[], int taille);
```

et cette déclaration, qui ne réserve pas d'espace pour le contenu du tableau, est en réalité celle d'un pointeur.

```
int somme_tableau(int *tableau, int taille);  // équivalent
```

1.3 Compatibilité tableau/pointeur

1. Le nom d'un tableau équivaut à l'adresse de son premier élément (d'indice 0). Elle peut donc être affectée dans un pointeur

```
int t[100];
int *ptr;
ptr = t;    // équivaut à   & (t[0])
```

2. La notation "crochets" est utilisable avec les pointeurs

```
ptr[10] = 3;  // met 3 dans le 10ieme entier après celui pointé par ptr
```

1.4 Arithmétique des pointeurs

1. La somme `p+n` d'un pointeur `p` et d'un entier `n` représente l'adresse du `n`-ième objet qui suit celui pointé par `p`, c'est-à-dire `& (p[n])`. Autrement dit, on `p[n]` équivaut à `*(p+n)`.
2. De façon cohérente, l'instruction `p++` fera en sorte que le pointeur `p` "avance" vers l'élément qui suit celui qu'il pointait précédemment.
3. Inversement, la différence entre deux pointeurs (vers deux éléments d'un même tableau) indique la "distance" entre ces éléments. Elle est exprimée en nombre d'éléments", pas d'octets, ce qui permet une portabilité des applications (sur des machines d'architectures différentes, les données peuvent ne pas avoir la même taille).

Cette "arithmétique des pointeurs" (expressions combinant pointeurs et entiers) est abondamment utilisée en C.

2 Chaines de caractères

2.1 Représentation des chaines

Les chaines de caractères C sont représentées par des suites d'octets, terminées par un octet nul ('\0').

(Attention, `char` signifie en réalité `octet`, en UTF8 un caractère peut être codé par plusieurs octets)

Exemple : la déclaration

```
char *chaine = "abc";
```

déclare un pointeur vers une zone réservée de 4 (quatre) octets, à cause du caractère nul final. Ces caractères sont (quand l'architecture matérielle le permet) dans une zone qui est protégée en écriture. Faire `chaine[0] = 'x';` va causer une violation mémoire.

Par contre

```
char chaine[] = "abc";
```

déclare un tableau de 4 octets qui sont initialisés. On peut modifier le contenu du tableau.

2.2 Fonctions sur les chaines

On ne pas affecter, par un simple “=”, une chaine dans une autre. On appelle pour cela une fonction `strcpy` (string copy) de la bibliothèque `string.h`

```
char *strcpy(char *dest, const char *src);
```

Attention : la destination doit être assez grande pour contenir la chaine transférée.

```
char *strcpy(char *dest, const char *src)
{
   int i = 0;
   while( src[i] != '\0') {
       dest[i] = src[i];
       i++;
   }
   *dest[i] = '\O';
   return dest;
}
```

Une autre, plus compacte,

```
char *strcpy(char *dest, const char *src)
{
   char *d = dest;
   while ((*d++ = *src++) != '\0') {
   }
   return dest;
}
```

Exercices : écrire des fonctions équivalentes à celles de la bibliothèque standard :

```
size_t strlen(const char *s);
char *strcat(char *dest, const char *src);
int strcmp(const char *s1, const char *s2);
```

- `size_t` est un type entier non signé assez grand pour y loger une taille
- `strcat` ajoute contenu de `src` à la fin de `dest`
- `strcmp` retourne 0 si les deux chaines ont le même contenu, un nombre négatif si la première vient avant la seconde dans l’ordre lexicographique, et positif sinon.