Tableaux, la théorie

Fini de rigoler maintenant ! Vous connaissez les tableaux, vous connaissez les pointeurs, il est temps de révéler une vérité qui dérange. On vous a menti, on vous a trompé, et tout cela volontairement ! Découvrez la vérité vraie au travers de ce chapitre.

1 Les tableaux en mémoire

Prenons tout d'abord le temps de préciser ce qu'il se passe en mémoire lorsque vous manipulez les tableaux. Vous l'avez appris dans le cours précédent un tableau permet de regrouper une **série de valeurs** de façon **contigüe en mémoire**. On repère un tableau (notamment lors de son passage dans une fonction) par l'**adresse de son premier élément**. Vous avez donc normalement toutes les bases pour comprendre ce qu'il se passe en mémoire lorsqu'on manipule un tableau. Il est donc temps de faire un peu de théorie en réalisant les exercices suivants.

TD **Exercice 1.**

TD **Exercice 2.**

TD **Exercice 3.**

Les matrices (ou tableaux 2D) ne sont pas si différentes des tableaux 1D. Un deuxième indice est rajouté pour parcourir les lignes et le prototype d'une fonction manipulant un tableau 2D est légèrement différent. Mais pas de craintes au niveau de la mémoire, la gestion de ces objets reste très similaire.

TD **Exercice 4.**

TD **Exercice 5.**

2 Pointeurs et décalage d'adresse

Nous voilà enfin arrivés sur un point critique. Vous avez appris lors du dernier chapitre que l'on repère un tableau grâce à l'adresse de son premier élément. Lors du passage d'un tableau à une fonction, on envoie donc l'adresse du tableau et non le tableau entier. Accéder à la case `i` d'un tableau `tab` s'écrit `tab[i]`. Mais finalement, une adresse n'est-elle pas un entier déguisé ? Auquel cas il serait possible da manipuler des adresses en réalisant des opérations arithmétiques... Appétissant tout ça !

TD **Exercice 6.**

L'exercice précédent vous a montré qu'ajouter un entier `i` à un pointeur `p` avait le même comportement qu'écrire `p[i]`. On appelle cette propriété l'*arithmétique des pointeurs*. Ajouter `i` à un pointeur le fait pointer `i` **cases** plus loin en mémoire, et pas forcément `i` **octets** plus loin. Un pointeur sur entier que l'on incrémente de 1 pointera 4 octets plus loin. Un pointeur sur caractère que l'on incrémente de 1 pointera 1 octet plus loin. Et un pointeur sur `void` ?

TD **Exercice 7.**

3 Chaines de caractères

Définition 1 (chaine de caractères). Une *chaine de caractères* est un tableau de caractères se terminant par le caractère `'\0'`.

Remarque 1. Une chaine de caractères se terminant par un caractère spécial, son passage à une fonction est plus simple. En effet, il est inutile de passer sa taille. Il suffit de détecter le caractère de fin de chaine pour savoir qu'on est à la fin du tableau. Une boucle `while` est alors préférable pour la manipulation de chaines de caractères.

Il s'agit d'une des principales utilisation des tableaux en C. Le mode de fonctionnement est identique à celui des autres tableaux 1D mais des raccourcis ont été ajoutés pour les chaines de caractères. Il existe plusieurs façons de déclarer une chaine de caractères.

- Comme un tableau classique, `char str1[7] = {'i','b','i','j','a','u'}` permet de créer la chaine `"ibijau"`. Cette méthode est peu utilisée car fastidieuse.
- On préfère définir la chaine de façon plus concise : `char str2[7] = "ibijau"`. La chaine est inscrite entre guillemets, la taille du tableau doit être strictement supérieure au nombre de caractères de la chaine pour laisser la place à `'\0'`.
- Les plus malins se passent de définir la taille du tableau. L'instruction `char str3[] = "ibijau"` réserve automatiquement 7 octets en mémoire pour y stocker la chaine `"ibijau"`.

Remarque 2. Notez bien la différence de notation entre un caractère et une chaine :

- l'expression `'c'` désigne la lettre `c`, donc le code ASCII 99 ;
- l'expression `"c"` désigne la chaine `"c"`, donc un tableau de 2 caractères.

Les trois méthodes précédentes permettent de définir un tableau tel que vous le connaissez jusqu'à présent. Ces tableaux sont donc modifiables à volonté. On rappelle en revanche que tous comportent un caractère de fin de chaine, sans quoi ils ne seraient que de pauvres tableaux de caractères. Puisqu'un tableau est un pointeur, il est également possible de déclarer des chaines dites *constantes* à l'aide de l'instruction suivante : `char *str4 = "ibijau"`. Ces chaines sont alors *non modifiables* : `"ibijau"` est enregistré en dur et les pointeur `str4` récupère simplement l'adresse de cette chaine. Il est alors possible de lire à cette adresse, mais pas d'y écrire.

TD **Exercice 8.**

TD **Exercice 9.**

Ce dernier exercice était un grand classique de la manipulation de chaine. Jetez maintenant un œil du côté de la librairie `<string.h>`. Vous y trouverez un ensemble de fonction bien utiles, et vous pouvez déjà les utiliser afin de vérifier que vos codes de l'exercice précédent sont corrects.