POLYTECH PARIS-SUD

Et3 - Tronc commun

Programmation C

TP5 - Mémoire, partie 2

Objectifs :

— Consolider sa maîtrise des tableaux dynamiques.
— Gérer et travailler avec des tableaux multidimensionnels.
— Maîtriser les interactions entre tableaux et fonctions.

On portera une attention particulière à la gestion sécurisée de la mémoire (libérer la mémoire lorsque c'est possible et l'absence de mémoire disponible).

Dans la suite "rendre" ou "renvoyer" se rapporte à la valeur de retour de la fonction considérée (et non à un affichage dans le terminal).

Rappels

Pour la déclaration d'une chaîne de caractères, un moyen pratique d'initialiser toute une chaîne d'un seul coup est d'utiliser la notation avec les guillemets autour de la chaîne de caractères, par exemple "bonjour". Une telle chaîne est appelée une chaîne de caractères littérale ("string literal" en anglais). Ces chaînes sont stockées dans une zone de la mémoire qui est en lecture seule.

Quand on déclare une chaîne de caractères de la façon suivante :

```
char mot[] = "bonjour";
```

la mémoire est allouée à l'adresse pointée par mot (cette fois-ci à un emplacement mémoire où l'on peut écrire) et la chaîne "bonjour" (terminée par le caractère de fin '\0') est copiée vers cet emplacement mémoire. Il est alors possible d'afficher et de modifier les caractères stockés dans ce tableau. Par exemple :

```
mot2[0] = 'd';
```

est parfaitement valide.

Si vous voulez simplement accéder aux valeurs à chaque indice d'une chaîne de caractères (sans les modifier), vous pouvez aussi utiliser la notation suivante pour déclarer une chaîne de caractères :

```
char *mot2 = "ciao";
```

Dans ce cas, la chaîne "ciao" est d'abord stockée dans un emplacement mémoire en lecture seule allouées à cet effet. Ensuite, mot2 est affecté avec l'adresse de cet emplacement. La chaîne n'est pas copiée.

On peut alors afficher les caractères de cette chaîne :

```
printf("%c", mot2[0]);
```

Mais on ne peut plus les modifier, puisque la chaîne est dans un emplacement mémoire en lecture seule :

```
mot2[0] = 'd';
```

En revanche, on peut changer le pointeur mot2 et le faire pointer vers une autre chaîne "hello", en faisant :

```
*mot2 = "hello";
```

En résumé, si vous voulez pouvoir modifier une chaîne de caractères, il faut utiliser cette déclaration :

```
char mot[] = "bonjour";
```

Pour faire encore plus simple : dans tous les cas, et sauf si vous êtes sûrs de vous, utilisez pour toutes les chaînes que vous déclarez et affectez en même temps une déclaration de type :

```
char mot[] = "bonjour";
```

Lorsque vous ne pouvez pas affecter votre chaîne au moment de sa déclaration, vous devez naturellement utiliser

```
char *mot;
```

puis l'allouer par un malloc.

Si vous voulez voir par vous-mêmes la différence entre les façons de déclarer une chaîne, essayez le code suivant :

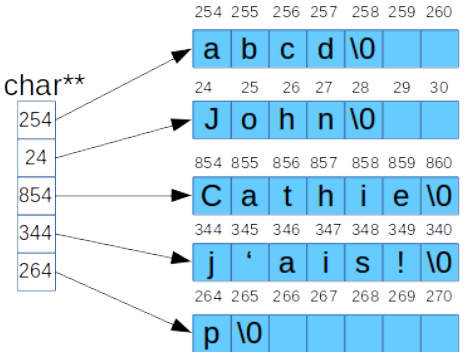
```
char *str1 = "Hello";                                   // Pointeur vers un "string
    literal"
char str2[] = "goodbye";                               // Chaîne de caractères
printf("l'adresse de str1 est %p\n", str1);
printf("l'adresse de str2 est %p\n", str2);
str1 = "Bonjour";                                       // On peut faire pointer str1
     vers un autre "string literal"
str2[0] = 'G';                                          // On peut modifier
    individuellement les caractères de str2
printf("l'adresse de str1 est %p\n", str1);   // L'adresse aura changée
printf("l'adresse de str2 est %p\n", str2);   // L'adresse n'aura pas changée
```

Pourquoi est-ce que l'adresse de `str1` change lorsqu'on le modifie ?

Exercice : 1 Retour sur les chaînes de caractères

Commençons par quelques fonctions simples pour s'échauffer. N'hésitez pas à découper en plusieurs fonctions si nécessaire.

1. Écrivez une fonction `char * read_name()` qui demande son nom à l'utilisateur sur le terminal, le récupère, et renvoie la chaîne de caractères contenant son nom. Faites attention à bien gérer la possibilité d'absence de mémoire disponible.
2. Écrivez une fonction `char * lower_case(char* str)` qui prend une chaîne de caractères et en renvoie une nouvelle où toutes les lettres en majuscule ont été remplacées par des lettres en minuscule.
3. Écrivez une fonction `char * upper_case(char* str)` qui prend une chaîne de caractères et en renvoie une nouvelle où toutes les lettres en minuscule ont été remplacées par des lettres en majuscule.
4. Écrivez une fonction `int letter_to_num(char c)` qui prend un caractère et renvoie son rang dans l'alphabet (1-26) si c'est une lettre et -1 sinon.
5. Écrivez une fonction `append` qui ajoute une chaîne de caractères `char*` à un tableau de chaîne de caractères `char**`. Faites attention à bien gérer la possibilité d'absence de mémoire disponible.



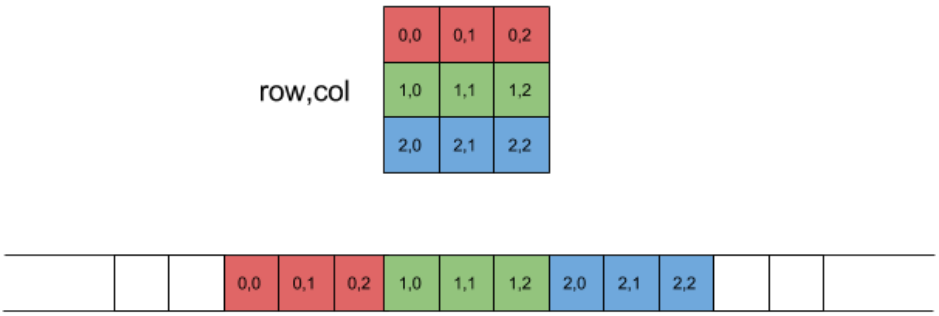
Représentation d'un tableau de chaînes de caractères (`char**`) dans la mémoire.

Exercice : 2 Liste de noms triée

1. Écrivez une fonction `int bigger(char* left, char* right)` qui prend 2 chaînes de caractères `left, right` et rend 1 si `left > right` ou 0 sinon.
2. Écrivez une fonction `int smaller(char* left, char* right)` qui prend 2 chaînes de caractères `left, right` et rend 1 si `left < right` ou 0 sinon.
3. Écrivez une fonction `int compare(char* left, char* right)` qui prend 2 chaînes de caractères `left, right` et rend 1 si `left > right` ou -1 si `left < right` ou 0 si `left == right`.
4. Écrivez une fonction `swap_str(left, right)` qui prend en entrée deux chaînes de caractères `left` et `right` et qui échange les chaînes de caractères tel que la chaîne de caractères initialement pointée par `right` sera pointée par `left` et vice versa. (Astuce : on peut se contenter d'échanger les adresses).
5. En utilisant `bigger, smaller` et/ou `compare`, écrivez une fonction qui prend en entrée un tableau de chaînes de caractères et le trie dans l'ordre alphabétique.

Exercice : 3 Tableaux 2D, les bases du calcul numérique

On suppose que l'on dispose de 2 tableaux 2D A, B de nombres flottants de même taille $n \times m$.



Représentation d'un tableau 2D dans la mémoire.

source : http://eli.thegreenplace.net/2015/memory-layout-of-multi-dimensional-arrays

Implémentez :

1. Une fonction qui affiche un tableau 2D dans le terminal (attention à l'alignement).
2. Une fonction qui remplit un tableau 2D avec une valeur donnée.
3. L'addition terme à terme : $c_{ij} = a_{ij} + b_{ij}$. Cette fonction doit renvoyer un nouveau tableau 2D, C de la même taille $n \times m$.
4. La soustraction terme à terme : $c_{ij} = a_{ij} - b_{ij}$. Cette fonction doit renvoyer un nouveau tableau 2D, C de la même taille $n \times m$.
5. La multiplication terme à terme : $c_{ij} = a_{ij} \times b_{ij}$. Cette fonction doit renvoyer un nouveau tableau 2D, C de la même taille $n \times m$.
6. La transposée : $c_{ij} = a_{ji}$. Cette fonction doit renvoyer un nouveau tableau 2D, C de taille $m \times n$.
7. La somme des lignes : $c_i = \sum_{j=0}^{m} a_{ij}$. Cette fonction doit renvoyer un nouveau tableau 1D de taille n.
8. La somme des colonnes : $c_j = \sum_{i=0}^{n} a_{ij}$. Cette fonction doit renvoyer un nouveau tableau 1D de taille m.

Exercice : 4 Multiplication matricielle

Écrivez un programme qui effectue la multiplication de deux matrices A et B. Le résultat de la multiplication sera mémorisé dans une troisième matrice C qui sera ensuite affichée. Pour mémoire, en multipliant

une matrice A de dimensions n et m avec une matrice B de dimensions m et p on obtient une matrice C de dimensions n et p :

$$A(n, m) \times B(m, p) = C(n, p)$$

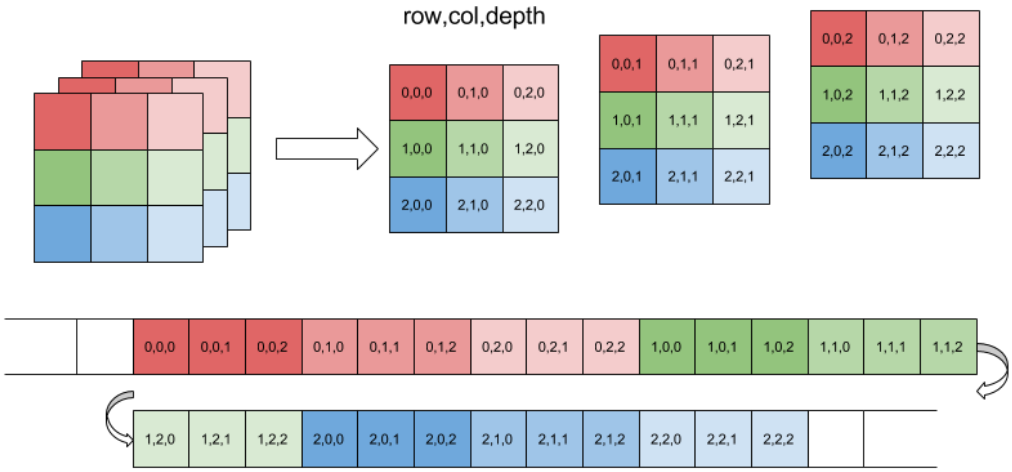
La multiplication de deux matrices se fait en multipliant les composantes des deux matrices lignes par colonnes :

$$c_{i,j} = \sum_{k=1}^{k=m} (a_{i,k} \times b_{k,j})$$

N'oubliez pas de décomposer en fonctions spécialisées !

Exercice : 5 Tableaux n-D, les bases du calcul numérique

On suppose que l'on dispose de 2 tableaux n-D A, B de nombres flottants de même taille $(n \times m \times \dots)$ Dans chacune des questions ci-dessous, vous devrez construire un nouveau tableau n-D, C, de même taille $(n \times m \times \dots)$ et le renvoyer.



Représentation d'un tableau 3D dans la mémoire.

source : http://eli.thegreenplace.net/2015/memory-layout-of-multi-dimensional-arrays

Implémentez :

1. Une fonction qui remplit un tableau n-D avec une valeur donnée.
2. L'addition terme à terme : $c_{ij\dots} = a_{ij\dots} + b_{ij\dots}$
3. La soustraction terme à terme : $c_{ij\dots} = a_{ij\dots} - b_{ij\dots}$
4. La multiplication terme à terme : $c_{ij\dots} = a_{ij\dots} \times b_{ij\dots}$

Astuce : ces opérations sont indépendantes de l'arrangement dimensionnel du tableau.

Exercice : 6 Bonus : Démineur

Vous avez déjà fini ?

Implémentez le jeu démineur.