TD n° 4

I3 : Pointeurs, tableaux, chaînes de caractères et allocation dynamique de mémoire

Exercice 1 [Pointeurs et valeurs]

Considérons le début de programme C suivant :

```c
#include <stdio.h>
#include <stdlib.h>

#define TAILLE 5

int main(){
  int tab[TAILLE];
  int* tab2;
  int* i;
  int j;

  return EXIT_SUCCESS;
}
```

1. Quelle sera la taille du tableau `tab` ?
2. Comment initialise-t-on chaque case du tableau `tab` avec la valeur 0 ?
3. Demandez à l'utilisateur de saisir les valeurs contenues dans le tableau `tab`
4. Initialisez les valeurs contenues dans le tableau avec la valeur de l'indice des élements
5. Quelle est la valeur de `tab` ?
6. Comment l'affiche-t-on ?
7. Quelle est la valeur de `*tab` ? Comment l'affiche-t-on ?
8. Peut-on effectuer l'assignation suivante : `i = tab[0]`
9. Quelle est la différence entre `i = &(tab[1])` et `*i = tab[1]` ?
10. Considérons que la suite du programme contient le code suivant. Qu'affiche le programme ?

```c
int x;
i = &x;
tab[1] = 1;
*i = tab[1];
tab[1] = 10;
printf( "*i = %d\n", *i );
i = &(tab[1]);
tab[1] = 100;
printf( "*i = %d\n", *i );
```

11. Peut-on effectuer l'assignation suivante : `tab2 = tab` ?
12. À quoi cette instruction correspond-elle en mémoire ?
13. Comment peut-on *dupliquer* le tableau `tab` à un endroit de la mémoire pointé par `tab2` ?

Exercice 2 [Permutation de valeurs]

1. Ecrire une procédure C `permuterGauche()` réalisant la permutation circulaire vers la gauche des valeurs de trois variables entières, à la manière de l'algorithme d'échange des valeurs de deux variables vu en TD d'algorithmique.
2. Ecrire un programme C qui initialise les trois variables en demandant à l'utilisateur de saisir des valeurs une par une, affiche leurs valeurs avant permutation, appelle la fonction `permuterGauche()` et enfin affiche les valeurs de ces variables après permutation.

Exercice 3 [Transposée d'une matrice carrée]

On appelle *matrice* un tableau à deux dimensions. La *transposée* d'une matrice est son symétrique par rapport à la diagonale. Par exemple, $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ a pour transposée $\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$.

Écrivez une procédure en C qui réalise la transposition *en place* (en utilisant une seule matrice, sans tableau supplémentaire) d'une matrice d'entiers $N \times N$. La procédure prend en arguments un pointeur vers la matrice et la taille N.

Exercice 4 [Tableaux en C]

Considérons le morceau de code C suivant :

```c
int* tab;
tab[0] = 1;
tab[1] = 2;
```

1. Que va-t-il se passer ?
2. On modifie le programme comme suit. Que va-t-il se passer ?

```c
int* tab;
tab = (int*) malloc( 2 * sizeof( int ) );
tab[0] = 1;
tab[1] = 2;
tab[2] = 3;
```

3. Comment faut-il modifier le dernier morceau de code pour qu'il s'exécute correctement ?

Exercice 5 [Saisie d'une phrase]

Lorsque l'utilisateur saisit une chaîne de caractères, celle-ci est stockée dans un tableau de caractères de taille au moins égale à la longueur de la chaîne de caractères (en la terminant par le caractère spécial `\0`). Cependant, il arrive que l'on ne sache pas à l'avance quelle sera la longueur de la chaîne de caractères que l'utilisateur va saisir.

Écrivez une fonction qui permet à un utilisateur de saisir une phrase d'une longueur quelconque et retourne cette phrase. Rappel : une phrase se termine par un point.

Exercice 6 [Supprimer les espaces]

1. Écrivez une fonction qui prend en entrée une chaîne de caractères, construit une chaîne de caractères à partir de la chaîne d'entrée en supprimant les espaces et retourne la chaîne ainsi construite. La chaîne construite utilisera le même espace mémoire que la chaîne d'entrée. On marque la fin d'une chaîne de caractères par le caractère spécial `\0`.

2. Écrivez une fonction qui réalise la même chose que la question précédente, mais en utilisant exactement l'espace mémoire nécessaire pour la deuxième chaîne de caractères.