POLYTECH PARIS-SUD

Et3 - Tronc commun

Programmation C

TP4 - Mémoire, partie 1

Objectifs :

— Comprendre la notion de mémoire et d'adresse
— Savoir définir « pointeur », « tableau » et « allocation dynamique »
— Savoir gérer la durée de vie des données allouées dynamiquement

Avant tout

Avant de commencer, lisez bien la feuille « Rappel sur les pointeurs ». N'hésitez pas à tester vos propres exemples pour bien comprendre la différence entre « valeur » et « adresse » et savoir les manipuler.

Les paramètres du `main`

On a toujours utilisé la déclaration `int main()`. Cette fonction renvoie un entier et ne prend aucun argument. Il existe cependant une version de la fonction `main` prenant deux arguments. La déclaration correspondante est la suivante : `int main(int argc, char *argv[])` ou `int main(int argc, char **argv)`.

Le paramètre `argv` est un pointeur sur un tableau dont chaque élément est une chaîne de caractères. La première chaîne de caractères est toujours le nom du programme. Le paramètre `argc` est le nombre d'éléments de `argv` (nom du programme compris).

On peut ainsi démarrer un programme depuis la ligne de commande en lui passant des paramètres :

```c
int main(int argc, char **argv) {

    printf("Nombre d arguments : %d\n", argc);

    int i;
    for (i = 0; i < argc; i++) {
        printf("Argument %d: %s\n", i+1, argv[i]);
    }

    return 0;
}
```

Si on compile ce code et qu'on l'exécute avec `./nomprogramme salut hello`, qu'est-ce qu'il va afficher ?

Application : Ecrivez un programme qui accepte un nombre variable d'arguments. Lire le premier paramètre. Vérifiez que c'est un entier égal à 2 ou 3. (Remarque : Faire attention à la conversion de type !)

Exercice : 1 Manipulation de tableaux

Pour comprendre comment manipuler des tableaux dans les fonctions, vous allez écrire plusieurs fonctions utiles de manipulation de tableaux.

Vous écrirez vos fonctions dans un fichier que vous appellerez *tableau.c*, distinct du fichier principal (celui qui contient la fonction `main`). N'oubliez pas de mettre les prototypes de vos fonctions dans un fichier *tableau.h* dans le même dossier.

Vous appellerez chaque fonction depuis le fichier principal, afin de la tester. Pensez à tester des exemples différents et tous ceux qui pourrait potentiellement poser problème pour votre fonction, par exemple le cas d'un tableau vide.

Écrivez des fonctions qui effectuent les tâches suivantes :

1. afficher un tableau i) d'`int`s, ii) de `float`s. Tester en utlisant le programme précédent : si le paramètre est un 2, afficher un tableau d'`int`s, si c'est 3, un tableau de `float`s.
2. (ré-)initialiser toutes les cases d'un tableau à une certaine valeur commune donnée en paramètre. Tester en passant cette valeur en paramètre du main.
3. chercher une valeur donnée dans un tableau d'`int`s. Renvoyer 1 si la valeur est trouvée et 0 sinon. Tester en passant cette valeur en paramètre du main.
4. compter le nombre d'occurrences d'une valeur donnée dans un tableau d'`int`s. Renvoyer le nombre d'occurences. Tester en passant cette valeur en paramètre du main.
5. créer un tableau d'`int`s aléatoires tous compris entre `min` et `max` ; la longueur du tableau ainsi que les valeurs `min` et `max` sont données en paramètres. Tester en passant ces valeurs en paramètre du main.
6. créer un tableau d'`int`s aléatoire, dont les valeurs `min` et `max` sont données en paramètres, mais où, cette fois-ci, la longueur du tableau est également choisie aléatoirement entre `longueurMin` et `longueurMax`. Vous pouvez appeler les fonctions définies à la question précédente pour éviter de dupliquer du code. Tester en passant ces valeurs en paramètre du main.

N.B. N'oubliez pas de libérer la mémoire allouée dynamiquement quand vous n'en avez plus besoin !

Exercice : 2 Permutation de valeurs

Écrivez une fonction `swap`, qui permette d'échanger les valeurs de deux entiers `a` et `b`.

Exercice : 3 Tri à bulles

Vous allez écrire une fonction qui implémente un algorithme classique de tri, appelé « tri à bulles ». Cette fonction devra prendre un tableau en entrée (ainsi que sa longueur) et faire en sorte qu'il soit trié par ordre croissant. L'algorithme consiste à balayer tout le tableau, en comparant les éléments adjacents et en les échangeant s'ils ne sont pas dans le bon ordre.

Nous utiliserons une variante de l'algorithme habituel. Dans cette variante, dès que l'on effectue un échange le balayage est arrêté et on revient au début du tableau. Naturellement, il faut alors recommencer un nouveau balayage, et ainsi de suite jusqu'à ce qu'aucun échange ne soit plus possible : le tableau est alors trié.

Vous pouvez tester votre programme sur le tableau suivant :

5	3	1	2	2

Pensez aussi à tester des cas spéciaux, tels que le tableau vide ou un tableau de longueur 1. Vous pouvez aussi introduire un compteur pour compter le nombre de comparaisons faites lors du tri d'un tableau donné.

Exercice : 4 Fusion de tableaux triés

Écrivez une fonction permettant de fusionner deux tableaux d'entiers triés, de sorte que le tableau résultant soit directement trié (sans qu'un tri ne soit nécessaire pour le trier).

Testez votre programme en générant aléatoirement deux tableaux (longueur et contenu), en les triant chacun grâce au tri à bulles. Ils peuvent alors vous permettre de tester votre fonction de fusion de tableaux triés.

Exercice : 5 Tirage au loto

Cet exercice est similaire au tirage du TP2.

Écrivez un programme permettant d'effectuer le tirage du loto de *nbTirages* valeurs tirées aléatoirement entre 1 et 49, sans remise des valeurs (i.e. le même nombre ne peut être tiré qu'une seule fois). Le résultat sera stocké dans un tableau.

Décomposez votre programme en plusieurs fonctions différentes :

- fonction de génération d'un nombre aléatoire entre `min` et `max`,
- fonction de tirage sans remise de *nbTirages* valeurs comprises entre `min` et `max`,
- fonction d'affichage du résultat,
- fonction `main`, qui appelle la fonction de tirage entre une valeur minimale de 1 et maximale de 49, puis qui appelle la fonction d'affichage du résultat.