Semaine 6 - Utilisation de la mémoire en C

Remarque : Cette séance est une séance de TP sur machine.

Exercice 1 – Gestion de la mémoire en C

L'allocation dynamique de la mémoire consiste à étendre, pendant l'exécution d'un programme, la mémoire qui lui est attribuée.
Les principales fonctions d'allocation dynamiques sont :

- **malloc** pour allouer un bloc de mémoire,
- **calloc** pour allouer un bloc de mémoire et l'initialiser à zéro,
- **realloc** pour agrandir la taille d'un bloc de mémoire,
- **free** pour libérer un bloc de mémoire.

Ces fonctions se trouvent dans la bibliothèque standard `<stdlib.h>` Les prototypes de ces quatres fonctions sont décrits dans leurs pages `man`. Le type `size_t` qui est utilisé est equivalent au type `unsigned long int` sous Linux.

1. On souhaite créer un tableau de `n` entiers dans une fonction. Sachant que les tableaux sont gérés comme des pointeurs en C, expliquez pourquoi le code suivant n'est pas correct :

```
int *creer_tableau(int n) {
    int tab[n];
    return tab;
}
int main() {
    int *t = creer_tableau(100);
    ...
    return 0;
}
```

2. Écrire la fonction `int *allouer_tableau(int dimension, int val)` qui alloue la mémoire d'un tableau de taille `dimension`, puis qui l'initialise en mettant chacune de ses cases à la valeur `val`.

   Testez vos fonctions au fur et à mesure dans le programme (fonction `main`).

3. Écrire une fonction `int *lire_n_entiers(int n)` qui lit `n` entiers entrés au clavier et les place dans un tableau de taille `n`. Vous utiliserez la fonction `allouer_tableau` définie précédemment.

4. Écrire une fonction `void liberer_tableau(int *tab)`, qui libère la mémoire utilisée par le tableau `tab` et une fonction `void afficher_tableau(int *tab, int dimension)` qui affiche ce tableau d'entiers (de taille `dimension`).

5. Écrire une fonction `int *lire_entiers(void)` qui lit une séquence d'entiers terminées par la valeur 0, et place la séquence complète dans un tableau suffisament grand. Commencez par créer un tableau de taille 4 et à chaque fois que le tableau est rempli et qu'il reste des valeurs à lire doublez sa taille.

6. Écrire une fonction `int **allouer_matrice(int lignes, int colonnes, int val)` qui alloue la mémoire d'une matrice de taille `lignes` × `colonnes` puis qui initialise tous ses éléments à la valeur `val`.

7. écrire une fonction `void liberer_matrice(int **mat, int lignes)`, qui libère la mémoire utilisée par la matrice `mat` ayant `lignes` lignes, ainsi qu'une fonction `void afficher_matrice **mat, int lignes, int colonnes)` permettant d'afficher une matrice. (Pensez à réutiliser les fonction déjà écrites pour les tableaux)

Exercice 2 – Exercice 2 – Gestion de la mémoire en C

Bibliothèque `<string.h>`

— La fonction `void *memcpy(void *dest, const void *src, size_t n)` de la bibliothèque `<string.h>` permet de copier `n` octets depuis la zone mémoire allouée `src` vers la zone mémoire `dest`. Attention : Les deux zones ne doivent pas se chevaucher.
— La fonction `void *memmove(void *dest, const void *src, size_t n)` copie `n` octets depuis la zone mémoire `src` vers la zone mémoire `dest`. Les deux zones peuvent se chevaucher.
— La fonction `void *memset(void *dst, int c, size_t n)` remplit les `n` premiers octets de la zone mémoire allouée vers laquelle pointe `dst` avec la valeur `c`.

Questions

1. On souhaite utiliser la fonction **memcpy** pour remplir un tableau de taille n avec n valeurs identiques (la valeur v sera passée en paramètre). Pour être plus efficace en mémoire, on souhaite utiliser une méthode de remplissage appelée "recopie de mémoire".

   Le principe est le suivant : au départ, seule la première case du tableau est initialisée (avec la valeur v). On recopie son contenu dans la case suivante puis on double la taille de l'espace considéré et on recommence (on recopie les deux premières cases dans les 2 suivantes, puis les 4 premières cases dans les 4 suivantes, etc).

   Complétez le code suivant :

```
int *initialiser_tableau_v1(int n, int valeur) {
    int len = 1;
    int *tab = malloc(...);
    ...
    while (2 * len <= n) {
        ...
        ...
    }
    ...
    return tab;
}
```

2. En utilisant la fonction **memset**, écrivez la fonction `char *initialiser_tableau_char(int dimension, char c)` qui initialise les cases d'un tableau de caractère de taille `dimension`, en mettant dans chacune de ses cases le caractère `c`.
3. En utilisation la fonction **memmove**, écrivez la fonction `void copier_chaine(char *tab, int s1, int taille, int s2)` qui copie la sous-chaine de taille `taille` commençant à la position `s1` à la position `s2`.

   Expliquez pourquoi il est nécessaire d'utiliser **memmove** plutôt que **memcpy**.

1 À préparer chez soi – Questions de cours

1. Donnez les deux grandes classes d'algorithmes de remplacement de pages.
2. En pratique, quel est le meilleur algorithme ? Pourquoi ?
3. Quel est l'inconvénient de l'algorithme FIFO avec seconde chance (FIFO-2) ?
4. Quel est la particularité de l'algorithme NRU ?
5. Quel est l'inconvénient de l'algorithme NRU ?
6. Quel est l'inconvénient de l'algorithme LRU ?