Séance 3 : Gestion de la mémoire et pointeurs

L2 Informatique – Université Côte d'Azur

Exercice 1 — Le B-A-BA des pointeurs

1. Écrivez une fonction `void incremente(int *a)` qui incrémente la valeur pointée par `a`.
2. Écrivez une fonction `void echange(int *a, int *b)` qui échange les valeurs pointées par les deux variables. Si avant la fonction `*a` vaut 4 et `*b` vaut 3, après appel, on aura `*a` qui vaudra 3 et `*b` qui vaudra 4.
3. Écrivez une fonction `int trier(int *a, int *b)` qui met la plus petite des valeurs dans `a` et la plus grande dans `b`. On utilisera la fonction précédente. On renverra `0` s'il n'y a pas eu d'échange et `1` s'il y a eu un échange.
4. Si ce n'est pas fait, testez vos fonctions et gardez les tests pour si le professeur vous les demande.

Dans la suite du TP, on aura besoin de réserver de l'espace mémoire sur le tas avec la fonction `malloc`. En C, il est indispensable de libérer la mémoire lorsque la variable n'est plus utilisée. Pour chaque `malloc` vous devez ajouter le `free` correspondant.

Exercice 2 — Manipulons les chaînes

1. Écrivez une fonction `longueur` calculant la longueur d'une chaîne. Testez-la dans une fonction `question1`. *Remarques : ne supprimer pas vos tests une fois qu'ils sont valides, mais au contraire, garder les dans une fonction.*
2. Écrivez une fonction `copie_chaine` qui renvoie une nouvelle chaîne allouée sur le tas mais identique à la précedente. Testez là-aussi votre fonction. Prenez l'habitude de libérer bien la mémoire lorsque vous n'utilisez plus la chaîne.
3. Écrivez une fonction `concatener` prenant deux chaînes en paramètre et renvoyant une nouvelle chaîne obtenue en concatenant les deux autres.

Exercice 3 — Des notes et des tableaux

On veut représenter dans une structure le relevé de notes des étudiants. Cette structure aura trois champs :

- `notes` : un tableau d'entier ;
- `taille` : un entier correspondant à la taille du tableau alloué en mémoire ;
- `n` : un entier correspondant au nombre de notes effectivement entrées dans le tableau.

1. Créez la structure `bulletin` avec ces trois champs.
2. Écrivez une fonction `bulletin nouveau(int taille)` renvoyant un nouveau `bulletin` avec un tableau initialisé à la bonne taille et ne contenant pour l'heure aucune note.
3. Écrivez une fonction `void libérer_mémoire(bulletin b)` qui libère la mémoire alloué sur le tas par la fonction précédente. Une fois cette fonction utilisée, la variable `b` passée en paramètre ne devra plus être utilisée (que se passe-t-il si vous le faîtes quand même ?).
4. Écrivez une fonction `int ajout_simple(bulletin *b, int note)` ajoutant une nouvelle note audit bulletin. Cette fonction renverra `0` si tout s'est bien passé, `-1` si la note n'est pas comprise entre `0` et `20` et `-2` si le tableau de notes est déjà plein. Dans ces deux derniers cas, le bulletin ne sera pas modifié.
5. Essayer d'écrire la fonction précédente en vous passant de pointeur. Est-ce possible ? Pourquoi ?
6. Afin de tester les deux questions précédentes, écrire une fonction `afficher_bulletin` qui affiche le nombre de note actuel sur la taille du tableau suivie des éventuelles notes. L'affichage pourra par exemple ressembler à cela :

stdin/stdout

```
0/3 []
1/3 [15]
2/3 [15, 9]
3/3 [15, 9, 10]
```

7. Écrivez une fonction `ajout` qui ajoute une nouvelle note au bulletin. Cependant, si le tableau est plein, on remplace la plus mauvaise note du tableau par celle donnée en paramètre (si cette dernière est meilleure, évidement). On utilisera la fonction `ajout_simple` et on renverra `0` si l'ajout s'est fait sans problème, `1` si le tableau étant plein, la pire note a été remplacée, `2` si le tableau étant plein, la nouvelle note a été ignorée car trop mauvaise et `-1` en cas de note non valide. *Remarque : vous avez le droit de créer des fonctions auxiliaires pour simplifier votre code.*

Exercice 4 — Tableaux dynamiques

Dans l'exercice précédant, le tableau était de taille fixe et ne permettait pas d'ajouter plus de note que prévu à l'origine. Nous allons créer un tableau à la Python dans lequel on peut ajouter et supprimer des éléments à notre guise.

1. Commencez par créer une structure `tableau`, contenant trois champs : un pointeur `memoire` vers un tableau, un entier `taille` représentant la taille du tableau `memoire` et un entier `n` représentant le nombre de valeurs effectivement stockées.
2. Écrivez une fonction `tableau * nouveau_tableau(int taille)` créant un nouveau tableau de taille correspondant mais ne contenant pour l'instant aucun élément.
3. Écrivez la fonction `void libérer_tableau(tableau *t)` libérant la mémoire allouée par le tableau donnée en argument.
4. Écrivez la fonction `void afficher_tableau(tableau *t)` sur le modèle de l'exercice précédent.
5. Écrivez une fonction `ajout_simple` prenant en argument un pointeur de `tableau` et un entier `valeur` et qui ajoute cet entier dans le tableau s'il reste de la place. La fonction renverra `0` en cas de succès et `-1` en cas d'échec.
6. Écrivez une fonction `void agrandir(tableau *t)` qui double la mémoire du tableau `t`. Pour cela on affectera au champs `memoire` une zone de taille double de la précédente, on copiera les données de l'ancienne vers la nouvelle et on libérera l'ancienne maintenant inutile. On s'assurera de garder cohérents les autres champs de notre structure.
7. En utilisant les deux fonctions précédentes, écrivez une procédure `void ajout(tableau *t, int valeur)` qui ajoute la valeur à la fin du tableau, quitte à augmenter la taille de ce dernier si nécessaire.
8. Écrivez une fonction `int inserer(tableau *t, int i, int valeur)` qui ajoute à l'indice `i` l'entier `valeur`, tous les élements d'indice supérieur au égale à `i` verront leur position être décalée dans le tableau. La fonction renverra `0` si tout s'est bien passé et `-1` si l'indice dépasse du tableau ; dans ce cas, ce dernier ne sera pas modifié.
9. Écrivez une fonction `int supprimer(tableau *t, int indice, int *valeur)` qui supprime du tableau l'élement situé à l'indice `i`, tous les élements d'indice supérieur au égale à `i` verront leur position être décalée dans le tableau. La valeur supprimée sera stockée dans le pointeur donnée en argument. La fonction renverra `0` si tout s'est bien passé et et `-1` si l'indice dépasse du tableau ; dans ce cas, ce dernier ne sera pas modifié.
10. Écrivez une fonction `void supprimer_doublon(tableau * t)` qui supprime tous les éléments présents en double dans le tableau. Par exemple à partir d'un tableau contenant : `[11, 31, 11, 12, 11, 31]`, le tableau sera modifié en `[11, 31, 12]`. On travaillera sur place (sans faire de `malloc`).

Exercice 5 — Défis : mise en forme des tableaux

1. Écrivez une fonction qui à partir d'un tableau d'entier renvoie une chaine correspondant à la représentation de ce tableau. Par exemple avec un tableau contenant les valeurs `11`, `-22` et `0`, la chaîne devra être de la forme : `"[11, -22, 0]"`. Vous n'aurez pas le droit d'utiliser des fonctions comme `sprintf`. Il faudra construire la chaîne à la main, de la conversion des entiers en chaîne à la concaténation de ces chaînes sous forme de tableau. Libre à vous de contruire toutes les fonctions auxillaires qui vous sembleront nécessaires. Attention, tous les `malloc` devront être libérés avec `free`.
2. Même question, mais en triant les nombres par ordre alphabétique (ceux qui commencent par 1 avant ceux qui commencent par 2, etc.)