I2010 : langage C (TP7)

Fonctions (suite et fin)

1. Prototypes de fonctions

Cet exercice consiste à concevoir le prototype de fonctions. Vous ne devez pas les implémenter !

1. a. Ecrivez le prototype d'une fonction qui reçoit en argument le chemin complet d'un fichier. Elle renvoie vrai si un tableau passé en paramètre a été rempli avec les lignes du fichier converties en entiers et ses tailles physique et logique mises à jour ; faux sinon.
Nommez les paramètres de façon explicite.

1. b. Ecrivez le prototype d'une fonction qui reçoit en argument le chemin complet d'un fichier. Elle renvoie vrai si un tableau passé en paramètre a été rempli avec les mots du fichier et ses tailles physique et logique mises à jour ; faux sinon.
Nommez les paramètres de façon explicite.

1. c. Ecrivez le prototype d'une fonction qui sépare les mots d'une chaîne de caractères. Elle les sauve dans un tableau passé en paramètre, sauve sa taille physique fournie en paramètre et renvoie sa taille logique. Elle renvoie -1 si une erreur s'est produite.
Nommez les paramètres de façon explicite.

2. Implémentation de fonctions

2. a. Ecrivez une fonction qui permute la valeur de deux entiers.

2. b. Ecrivez une fonction qui permute le contenu de deux chaînes de caractères, sans permuter une à une leurs valeurs. On suppose que les tableaux ont été alloués dynamiquement.

2. c. Ecrivez une fonction qui permute le contenu de deux tableaux d'entiers, sans permuter une à une leurs valeurs. On suppose que les tableaux ont été alloués dynamiquement.

3. Débogage de fonction

Expliquez le(s) problème(s) que présente le code suivant et proposez une solution, sans modifer le prototype de la fonction :

```
/* Crée un tableau avec les puissances d'un nombre
 * PRE: n: nombre entier strictement positif
 * RES: renvoie un tableau de taille n+1 contenant les puissances de n
 */
int* creerTable (int n) {
   int t[n+1];
   t[0] = 1;
   for (int i=1; i<=n; i++)
      t[i] = t[i-1] * n;
   return (int*)t;
}
```

4. *Manipulation de tables dynamiques de chaînes par des fonctions*

Ecrire un programme qui reçoit des arguments sur la ligne de commande et les traite comme suit :

- il copie les chaînes passées en argument dans un nouveau tableau de chaînes ;
- un argument ne sera copié dans cette table que s'il ne s'y trouve pas déjà (pas de doublons possibles) ;
- le programme trie la table par ordre alphabétique grâce à un tri par sélection, dont voici le pseudo-code :

```
procédure tri_selection (tableau t)
    n ← longueur(t)
    pour i de 0 à n-2
        min ← i
        pour j de i+1 à n-1
            si t[j] < t[min], alors min ← j
        fin pour
        si min ≠ i, alors échanger t[i] et t[min]
    fin pour
```

Nous vous demandons de compléter le fichier ***ex7.4.c***, i.e. la fonction principale `main()` et les fonctions suivantes (les specs sont précisées dans le fichier source) :

```
char** copierArgs (char** tab, int n, int* ncp);
void trier (char** tab, int n);
void afficher (char** tab, int n);
```

Pour rappel, les principales fonctions traitant les chaînes sont : `strlen`, `strcpy`, `strcat` et `strcmp`. Référez-vous au `man` pour plus de détails sur leur utilisation.

Exemple d'exécution :

```
> ./a.out toton titi toto tata toto titi
Arguments non triés:
        1 - 'toton'
        2 - 'titi'
        3 - 'toto'
        4 - 'tata'
Arguments triés:
        1 - 'tata'
        2 - 'titi'
        3 - 'toto'
        4 - 'toton'
```

5. *Manipulation de tables dynamiques d'entiers par des fonctions*

Le but de cet exercice est de construire une application en C qui manipule une table de `int`. Le programme reçoit des nombres entiers en argument qu'il charge dans une table. Ensuite il complète la table avec des nombres entiers introduits sur `stdin`. A chaque étape, il affiche le contenu de la table précédé d'une légende.

Ce programme utilisera les 4 fonctions `imprimerTable`, `lireLigne`, `chargerTable` et `ajouterTable` décrites ci-dessous.

Dans **un premier temps**, nous vous demandons d'écrire les signatures de ces fonctions.

Dans **un second temps**, nous vous demandons d'implémenter ces fonctions, ainsi qu'un programme `main()` pour les tester.

Dans **un troisième temps**, nous vous demandons de spécifier ces fonctions (en termes de conditions PRE / POST / RES).

- `imprimerTable`, de type `void`, reçoit 3 paramètres :
  - la légende `legende`,
  - la table `tab` de `int` et sa taille logique `sz`.

  Cette fonction affiche la chaîne `legende` puis les `sz` entiers de `tab`.

- `ajouterTable`[1], de type `bool` reçoit en paramètre :
  - une table `tab` de `int` (`tab` est `NULL` si tableau non alloué),
  - sa taille logique `tailleL`,
  - sa taille physique `tailleP`,
  - et la valeur entière à ajouter .

  Cette fonction ajoute l'entier fourni dans le tableau `tab`.

  Elle retourne `false` en cas de souci, `true` sinon. Elle doit gérer l'allocation dynamique de la table d'entiers : au premier appel sur un tableau non encore alloué (i.e. `tab` est `NULL`), elle crée une table de trois `int` et quand la table est saturée, elle doit réallouer la table en doublant sa capacité et libérer la mémoire de l'ancienne table.

  Notez que cette fonction modifie certains paramètres. Ces paramètres seront donc des pointeurs ou des pointeurs de pointeur puisqu'en C, tous les paramètres sont passés par valeur.

- `chargerTable` de type `int` reçoit en paramètre,
  - une table `mots` de chaînes de caractères représentant des entiers,
  - et sa taille logique `nbrMots`,
  - la table d'entiers `tab` à remplir (`tab` est `NULL` si tableau non alloué),
  - et sa taille physique `tailleP`,

1. L'ajout dans une table nécessite deux tailles : la *taille physique* (nombre d'éléments de la table en mémoire) et la *taille logique* (nombre d'éléments réellement chargés dans la table). La taille physique est mise à jour par la fonction d'ajout. Par conséquent, elle sera définie comme un paramètre modifiable par cette fonction. La taille logique est quant à elle nécessaire pour exploiter le contenu de la table ; elle doit donc toujours accompagner la table.

Cette fonction traduit un tableau de chaînes de caractères représentant des entiers en un tableau de `int`. La conversion d'une chaîne en entier se fait via la fonction `atoi` (cf. `man atoi`). Vous n'avez pas à vérifier que les chaînes sont valides.

La fonction retourne la taille logique de la table ; -1 en cas de souci. Elle doit utiliser la fonction `ajouterTable`.

Notez que cette fonction modifie certains paramètres. Ces paramètres seront donc des pointeurs ou des pointeurs de pointeur puisqu'en C, tous les paramètres sont passés par valeur.

- `lireLigne`, de type `char*`, ne reçoit pas de paramètre

  Elle lit une ligne de caractères de taille quelconque sur `stdin` et renvoie celle-ci. Elle ne renvoie pas le '\n'. Par contre, elle renvoie NULL en cas d'erreur ou EOF.
  Idéalement, `strlen(s)+1` est égal à la taille physique de `s` (où s est la chaîne renvoyée).

Ce programme – et donc ces fonctions – ne peuvent utiliser aucune variable globale !

Voici un exemple d'exécution de ce programme :

```
anthony@IP-DEA044-2:solutions$ ./a.out 5 3 13 9 2

Avant traitement:
        [table vide]


Table des arguments:
        5
        3
        13
        9
        2

Entrez des entiers a ajouter a la table [un par ligne ; Ctrl-D pour terminer]:
21
1
50

Table completee:
        5
        3
        13
        9
        2
        21
        1
        50

anthony@IP-DEA044-2:solutions$
```

6. *Exercice récapitulatif : examen janvier 2023*

Les fichiers de l'examen se trouvent dans la section « Enoncé de l'examen de janvier 2023 » sur Moovin.