

Algorithmique et Structures de Données

TP 2 - Structure de données linéaire

Halim Djerroud (hdd@ai.univ-paris8.fr)

Partie I

Tous les programmes suivants sont à écrire en C :

Les listes chaînées simple et doubles :

1. On souhaite dans cet exercice implémenter les fonctions de base qui vont nous permettre de développer une application de gestion des contacts pour un téléphone portable. Les ressources son limitées donc il est important de bien gérer la mémoire et faire attention au gaspillage.

ATTENTION : Après avoir écrit les fonctions il faut systématiquement écrire un code pour les tester.

Vous devez écrire cet exercice en deux versions, la première sera avec une liste simplement chaînée et la seconde sera avec liste doublement chaînée. Les nom des fonctions est quasiment le même vous pouvez utiliser des préfixes de fonctions différents par exemple "lst_" pour la liste doublement chaînée et "lsts_" pour la liste simplement chaînée.

2. Déclarer une structure contact à l'aide de `typedef`, composée des éléments suivants :

```
id : int /* Remarque : l'id n'est pas indispensable */
phone_number : chaîne de caractère de taille 20
phone_name : chaîne de caractère de taille 50
```

Cette structure représente un enregistrement et constituera les nœuds d'une liste doublement chaînée. Donc il faut rajouter deux pointeur `next` et `prev` qui pointent respectivement sur l'élément suivant et précédent.

3. Déclarer une structure `list` à l'aide de `typedef`, cette structure représente une liste doublement chaînée elle contiendra juste un pointeur sur le début (nommé `HEAD`)et un autre sur la fin de liste (nommé `TAIL`).
4. Écrire une fonction `list* lst_create_liste()` qui permet créer la liste vide doublement chaînée. Cette fonction alloue dynamiquement la mémoire et retourne un pointeur sur la liste.

5. Écrire une fonction `contact* lst_create_node(int id, char* name, char* phone_number)` qui permet créer dynamiquement un contact et retourne un pointeur sur ce dernier.
6. Écrire une fonction `void lst_display_contact(contact* ptr)` qui permet d'afficher un élément contact pointé par `ptr`. Exemple l'affichage se présentera comme suit :

```
--
ID : 1
Nom : Jane Doe
Num : 01 02 03 04 05
```

7. Écrire deux fonctions `void lst_insert_head(list* lst, contact *pnew)` et `void lst_insert_tail(list* lst, contact *pnew)` qui insère respectivement l'élément pointé par `pnew` au début et à la fin de la liste `lst`.
8. Écrire une fonction `void lst_insert_after(list *lst, contact *pnew, contact *ptr)` qui permet d'insérer l'élément `pnew` juste après l'élément pointé par `ptr` dans la liste `lst`.
9. Écrire une fonction `void lst_display(list *lst)` qui affiche tout les éléments de la liste `lst`.
10. Écrire deux fonctions `void lst_delete_head(list *lst)` et `void lst_delete_tail (list *lst)` qui suppriment respectivement le premier et le dernier élément de la liste `lst`.
11. Écrire une fonction `void lst_delete_contact(list *lst, contact* ptr)` qui permet de supprimer l'élément pointé par `ptr` dans la liste `lst`
12. Écrire une fonction `void lst_erase(list *lst)` qui supprime tout les éléments de la liste `lst`.
13. Écrire une fonction `contact* lst_search_id(list *lst, int id_search)` qui permet de recherche dans la liste `lst` l'élément dont `id = id_search` et retourne un pointeur sur l'élément trouvé, sinon la fonction retourne `NULL` si l'élément n'existe pas.
14. Pour terminer créer un fichier `.h` qui contient les entêtes des fonctions afin de pouvoir réutiliser la liste chaînée dans les exercices suivants.

Tables de Hachages :

Dans cet exercice nous souhaitons augmenter encore les performance de la recherche des contacts, donc nous allons changer de structure de données pour le stockage des contacts, nous avons opté pour l'utilisation d'une table de hachage. Dans ce qui suit nous allons écrire les fonctions principales pour la gestion d'une table de hachage.

ATTENTION : Après avoir écrit les fonctions il faut systématiquement écrire un code pour les tester.

1. Créer une structure qui permet de déclarer une table de hachage comme suit :

```
typedef struct hashtab_{
    int size;
    list** tab;
}hashtab;
```

2. Créer une fonction `int hashtab_get_hash (hashtab* htab, int id)` qui prend en paramètre une table de hachage et un `id` et retourne un clé. Le calcul de la clé est basé sur un modulo.
3. Créer une fonction `list** hashtab_create_tab(int size)` qui permet de créer un tableau de liste chaînées (utiliser les fonction de l'exercice précédent).
4. Écrire une fonction `hashtab* hashtab_creat(int size)` qui permet de créer une table de hachage de taille `size`
5. Écrire une fonction `void hashtab_print(hashtab* htb)` qui permet d'afficher le contenu d'une table de hachage.
6. Écrire une fonction `void hashtab_insert_lem(hashtab * htab,intid,char * name,char * num)` qui permet d'insrer un lment dans la table de haschage.

Les piles et fils :

Toujours pour la problématique de la gestion des contacts du téléphone nous souhaitons gérer l'historique des appels entrant et sortant, pour cela nous allons utiliser une pile. Pour gérer les appels en absence nous allons utiliser une file. Dans cet exercice nous allons écrire les fonctions pour gérer une file et pile, vous pouvez vous inspirer des fonctions écrites dans l'exercice 1. Pour des raisons de performance nous allons juste utiliser un chaînage simple.

ATTENTION : Après avoir écrit les fonctions il faut systématiquement écrire un code pour les tester.

1. Proposer et implémenter un certain nombre de fonctions pour gérer une file, vous devez utiliser soit des listes simplement chaînées ou doublement chaînées, votre choix doit être justifié.
2. Proposer et implémenter un certain nombre de fonctions pour gérer une pile, vous devez utiliser soit des listes simplement chaînées ou doublement chaînées, votre choix doit être justifié.

Allez plus loin :

1. Écrire une librairie (.so ou .a) qui permet de proposer l'ensemble des fonctionnalités proposées dans ce TP.
2. Écrire une documentation pour votre librairie (man).
3. Diffuser votre bibliothèque et la documentation dans votre `public_html`.

Partie II

Tous les programmes suivants sont à écrire en C++.

Questions de cours :

Questions de cours :

Les deque (double-ended queue) :

1. Rappeler l'intérêt de l'utilisation des deque
2. Donner ses caractéristiques

Les listes :

1. Rappeler l'intérêt de l'utilisation des listes
2. Donner ses caractéristiques

Les queues (fifo) :

1. Comment elles sont définies ?

Les piles (stacks ou lifo) :

1. Comment elles sont définies ?

Exercice 1 : Liste de contacts

On souhaite écrire un module de gestion de contacts et appels téléphonique pour un téléphone portable. Les informations à stocker sont les suivantes :

--

Nom : Jane

Prénom : Jane

Num_1 : 01 02 03 04 05

Num_2 : 01 02 03 04 06

Num_n : 01 02 03 04 09

Chaque personne peut avoir plusieurs numéros de téléphones. On souhaite avoir les fonctionnalités suivantes :

Généralités

1. Créer un menu pour gérer toutes les fonctionnalités demandées.
2. Choisir un conteneur STL approprié.
3. Utiliser les outils fournis par la STL lorsque c'est possible.

Gestion des contacts

1. Ajouter simplement un contact avec saisie au clavier, il faut vérifier si le contact existe (même nom et prénom).
2. Afficher la totalité la liste de contacts par ordre alphabétique.
3. Afficher les contacts un par un, par ordre alphabétique.

4. Rechercher un contact avec une partie du nom ou du prénom, si y a plusieurs possibilités, alors afficher l'ensemble des possibilités.
5. Supprimer / Modifier un contact, ces options doivent être intégrées à la recherche.

Gestion des appels en absence

Un appel en absence est un numéro entrant sans réponse de l'utilisateur à une date et heure précise. Attention, on peut recevoir un appel qui n'existe pas dans la liste des contacts.

1. Ajouter un appel en absence (seulement un numéro de téléphone est fourni).
2. Afficher le nombre d'appels en absences.
3. Consultation d'un appel en absence, lors de la consultation, l'appel est automatiquement supprimé. Attention, l'affichage doit se faire du plus ancien au plus récent.
4. Supprimer l'ensemble des appels en absence.

Gestion des appels entrant/sortant

Un appel entrant/sortant, est un numéro entrant/sortant à une date et heure précise. Attention, on peut émettre/recevoir un appel qui n'existe pas dans la liste des contacts.

1. Ajouter un appel entrant/sortant (pour les appels sortants on peut fournir un contact ou un numéro).
2. Afficher le nombre d'appels entrants/sortants.
3. Consultation des appels entrants/sortants, lors de la consultation l'appel n'est pas supprimé. Attention, l'affichage doit se faire du plus récent au plus ancien.
4. Supprimer l'ensemble des appels entrant/sortant.