Fichier indexé

PRINCIPE

On nous demande d'écrire un programme pour gérer le volumineux fichier d'adresses de l'*Association des Survivants du DESS*. Les informations retenues pour chaque individu sont :

nom	chaîne limitée à 31 caractères utiles
prénom	chaîne limitée à 19 caractères utiles
adresse	structure à trois champs :
rue	chaîne limitée à 79 caractères utiles
ville	chaîne limitée à 19 caractères utiles
code	structure à deux champs :
département	entier
commune	entier

Le fichier en question, formé d'articles ayant la structure ci-dessus, sera appelé *fichier principal*.

On supposera (ce n'est pas possible en général) qu'il n'y a pas deux membres possédant le même nom. On peut ainsi mettre en bijection les articles du fichier principal et les noms des membres de l'association en question. Le but de cet exercice est d'utiliser cette bijection comme moyen d'accès aux articles du fichier, c'est ce qu'on appelle l'*accès indexé*. Le champ de chaque article retenu à cet effet, ici le champ *nom*, s'appelle la clé d'accès.

Pour pratiquer l'accès indexé, notre programme maintiendra une *table d'index*, c'est-à-dire une table de doublets (*clé* , *position*) par laquelle chaque clé *C* se trouve associée à la position dans le fichier principal de l'article ayant la clé *C*.

Ainsi, « commander l'accès à l'article ayant la clé *C* » se traduira par

- chercher dans la table d'index la position *P* correspondant à la clé *C* ;
- dans le fichier principal, provoquer l'accès direct à l'article ayant la position *P*.

La table d'index, moins volumineuse que le fichier qu'elle indexe, réside en mémoire centrale aussi longtemps que le programme est actif. Au début de la session elle est lue intégralement depuis un fichier séquentiel, dit *fichier d'index*. A la fin de la session, elle est sauvegardée dans un nouveau fichier d'index. Le fichier d'index n'a de sens qu'entre deux sessions : durant une session, la « bonne » table d'index se trouve en mémoire et le fichier d'index est obsolète.

Notre programme doit assurer les fonctions suivantes sur demande de l'utilisateur :

- affichage de l'article correspondant à une clé lue à la console ;
- saisie et insertion dans le fichier d'un nouvel article ;
- suppression d'un article existant.

INDICATIONS

1. La fonction d'insertion vérifiera qu'on ne crée pas de nouvel article associé à une clé déjà connue dans le fichier.

2. La place occupée par les articles supprimés doit pouvoir être récupérée pour les articles ultérieurement créés. A cet effet, les articles du fichier principal comporteront un champ supplémentaire, de type entier, réalisant un chaînage des articles libérés. La tête de cette liste chaînée (c'en est une, bien qu'elle ne soit pas construite en mémoire, ni à l'aide de pointeurs) sera conservée dans une variable du programme et sauvegardée avec la table d'index.

3. En définitive, le fichier d'index sera donc constitué par :
   - un entier, sauvegarde de la variable exprimant le nombre *N* d'articles utiles du fichier principal (i.e. le nombre de membres), qui est aussi le nombre d'éléments de la table d'index ;
   - un entier, sauvegarde de la variable exprimant le nombre total d'articles du fichier principal (articles utiles + articles supprimés et non encore réutilisés) ;

- un entier, sauvegarde de la variable qui exprime la position du premier article de la liste des articles réutilisables ;
- *N* paires (*clé* , *position*) qui sont la sauvegarde de la table d'index.

4. En temps normal, lors de l'activation de notre programme un fichier principal (créé progressivement lors des sessions précédentes) et un fichier d'index (produit à la fin de la dernière des sessions précédentes) doivent exister déjà. Cependant, pour permettre l'initialisation du système, il faudra prévoir aussi la possibilité de démarrer à partir de rien, c'est-à-dire sans fichiers préalables. Par exemple, on peut convenir que si l'ouverture du fichier d'index échoue, alors les tables et variables globales sont initialisées sans lecture de fichier, avec les valeurs suivantes :

- le nombre d'articles utiles du fichier principal est 0 (conséquence : la table d'index est vide) ;
- le nombre total d'articles du fichier principal est 0 ;
- la position du premier article de la liste des articles supprimés et non encore réutilises est -1.

DECOMPOSITION DU PROGRAMME

On pourra (mais ce n'est pas une obligation) décomposer le programme de la manière suivante :

```
void consultation(void);
void creation(void);
void suppression(void);
```

chacune de ces fonctions assure l'un des trois services que l'utilisateur peut demander. A leur tour, ces fonctions appellent

```
void saisir(ARTICLE *adrArticle);
void afficher(ARTICLE *adrArticle);
```

qui s'occupent respectivement de l'acquisition au clavier et de l'affichage à l'écran de l'ensemble des renseignements qui constituent un article. Le prédicat

```
int existe(char cle[], int *adrRang);
```

effectue la recherche de la clé indiquée dans la table d'index. Si elle s'y trouve, la fonction rend « vrai » et la position de la clé dans la table est copiée dans la variable pointée par *adrRang*. Si la clé ne se trouve pas dans la table, la fonction est « fausse » et la variable pointée par *adrRang* reçoit pour valeur la position dans la table où la clé doit, le cas échéant, être insérée. Les fonctions

```
void ajouterCle(char *cle, int position, int rang);
void supprimerCle(int rang);
```

s'occupent de l'ajout et de la suppression d'un doublet (*clé* , *position*), connaissant son rang dans la table d'index (ce rang a été trouvé avec la fonction *existe*). Les fonctions

```
int obtenirEmplacement(void);
void libererEmplacement(int position);
```

gèrent l'allocation des articles sur le fichier principal. L'appel de *obtenirEmplacement()* rend la position d'un article du fichier qu'on peut utiliser pour y enregistrer une fiche nouvellement saisie. L'appel *libererEmplacement(position)* notifie que l'information enregistrée dans l'article du fichier ayant la position indiquée n'a plus d'utilité. Les fonctions élémentaires

```
void chargerArticle(int position, ARTICLE *a);
void rangerArticle(int position, ARTICLE *a);
```

prennent en charge la lecture ou l'écriture de l'article du fichier principal à la position indiquée. Les fonctions

```
void sauverFichierIndex(void);
void chargerFichierIndex(void);
```

s'occupent de la sauvegarde et de la restauration de la table d'index (et quelques variables associées, voir plus haut). Enfin, pour la mise au point du programme, on pourra prévoir aussi une fonction pour afficher le contenu de la table d'index.

☺☺☺