Responsable pédagogique	**AF**	AM	PB	PM
Période	Sem1	Sem2	Sem3	Sem4
Volume horaire	Cours/TD		TP	
			6	

Types de données
Pourquoi et Comment ?

Indicateur temporel :

questions	1h	2h	3h	4h	5h	6h
Chargement des données						
Stockage des données						
Utilisation des données						
Suppression/tri e						

Documents à rendre : Sujet de TP annoté + mainTest.c et indiv.h imprimés

Introduction

Le but de cette série de TP est de comprendre pourquoi est-il utile voir nécessaire d'utiliser des types de données avancés !

Rappelons tout d'abord ce qu'est un type de données avancés, prenez votre RCI sur le chapitre type complexes si vous en ressentez le besoin.

Cahier des Charges

Principe de base

On se propose de réaliser une application logicielle (*contacts*) permettant à un utilisateur de gérer un agenda privé dont chaque entrée pourra contenir les informations suivantes :

- nom de famille,
- prénom usuel,
- adresse postale,
- numéro de téléphone,
- adresse électronique (*email*).

Les données du carnet seront stockée dans un fichier nommé par défaut *contacts.dta*. L'application doit permettre la consultation des données, l'édition pour modification d'une fiche contact existante, la création de nouvelles fiches, la suppression d'une fiche, ou encore la recherche de fiches à partir du nom de famille...

Le contenu de ces données, si elles existent, doit être automatiquement chargé dès le lancement de l'application. Les fiches devront être ordonnées suivant l'ordre lexicographique des informations nom et prénom ;

Lorsque l'utilisateur quitte l'application, celle-ci doit mettre automatiquement à jour la « base de données » (autrement dit le fichier chargé au départ) en fonction des changements effectués.

Afin de pouvoir réaliser les premiers essais, un embryon de fichier *contacts.dta* est gracieusement fourni. Celui-ci contient une vingtaine de fiches partiellement remplies, mais non triées suivant l'ordre alphabétique...

Format du fichier « contacts.dta » et type INDIV

Le fichier contacts.dta fourni contient une succession d'enregistrements de type `INDIV`.

- Ce type `INDIV` comme « individu » contient `nom, prénom, adresse, tel, mail`
- En C la création d'un type passe par `typedef`

Ce type `INDIV` ce décrit comme ceci dans un fichier d'entête « .h ».

```
#define TNOM 20
#define TPRENOM 20
#define TADR 40
#define TTEL 15
#define TMAIL 25

typedef struct {
            char nom[TNOM] ;
            char prenom[TPRENOM] ;
            char adresse[TADR] ;
            char tel[TTEL] ;
            char email[TMAIL] ;
} INDIV ;
```

Question ?

Quelle est la taille utile maximum de l'email en nombre de caractères ?

1ere étape, Chargement des données

Dans un premier temps nous allons nous intéresser à `indiv.h` et à `mainTest.c`.

`indiv.h` contient la déclaration du type `INDIV` vu précédemment.

`mainTest.c` contient une fonction de chargement et un `main` de test.

A FAIRE : Complétez indiv .h avec le type INDIV

A FAIRE : Compilez et exécutez le programme généré

Question ?

Ecrivez la ligne de compilation

Question ?

Ecrivez la ligne d'exécution du programme avec le fichier contact.dta

---

Question ?

Combien d'enregistrements sont présents dans contact.dta ?

Question ?

mainTest.c contient trois fonctions, donnez leur nom et expliquez en brièvement le rôle **et** le fonctionnement.

1)_____

---

2)_____

---

3)_____

Stockage des données

Chargement des données

Intéressons nous maintenant plus particulièrement à la fonction de chargement.

```
int    chargement( char* nf )
{
       int nbr = 0 ;
       INDIV indiv ;
       int hfile = open( nf, O_RDONLY ) ;

       while ( read( hfile, &indiv, sizeof(INDIV) ) ) {
              nbr++ ;
              afficher( &indiv ) ;
       }

       close( hfile ) ;
       return nbr ;
}
```

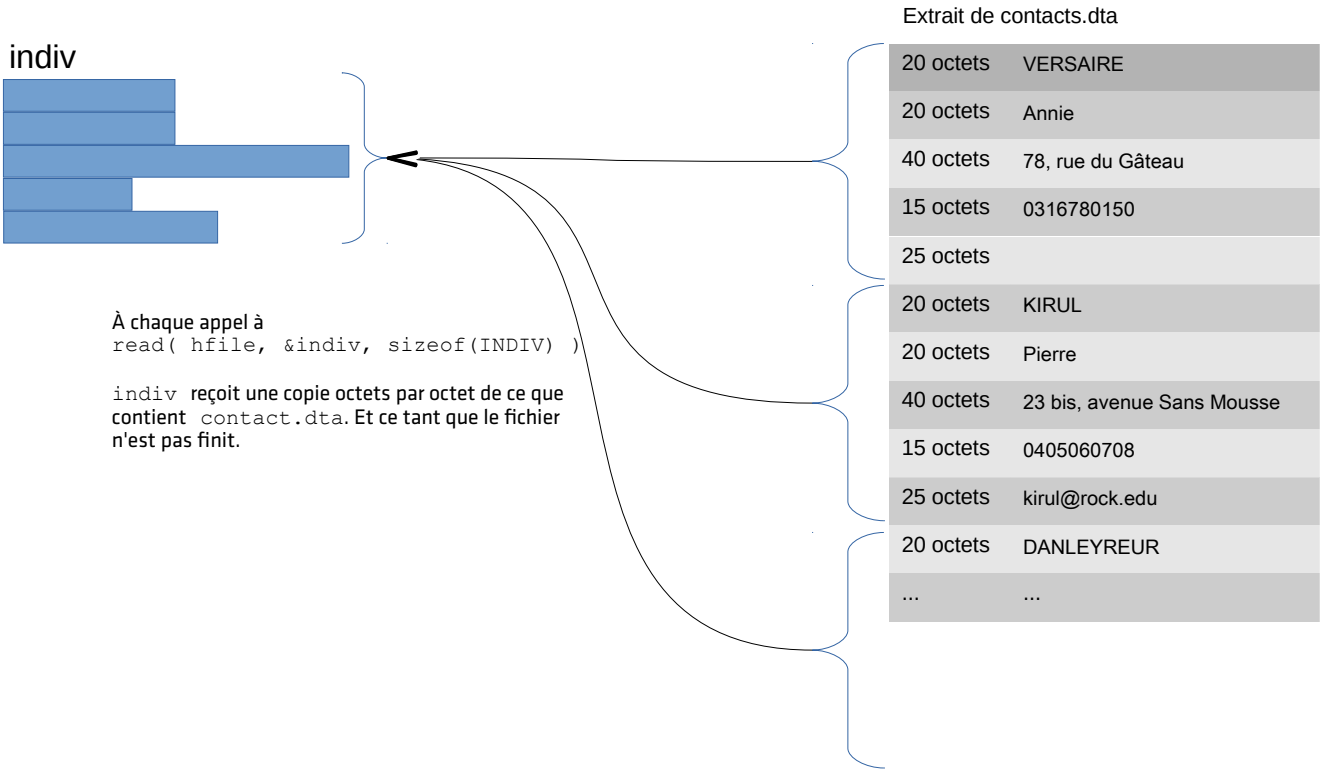
Cette fonction ouvre le fichier dont le nom est passé en argument par `nf`. Ce fichier est ouvert en mode lecture seule par `open`. La présence du fichier et l'autorisation d'y accéder ayant été vérifié dans le main, nous nous passons ici de vérifier la présence du fichier.

Intéressons nous maintenant à ceci : `read( hfile, &indiv, sizeof(INDIV) )`.

Pour mémoire `read` (`ssize_t read(int fd, void *buf, size_t count);`) essai de lire jusqu'à `count` octets depuis le descripteur de fichier `fd` dans le buffer pointé par `buf`. Si il n'y

arrive pas il renvoi des code d'erreur en retour 0 si c'est la fin du fichier ou -1 si par exemple le fichier n'est pas ou mal ouvert, sinon il renvoi la taille lu en octets.

Appliqué à notre cas on peux l'expliquer ainsi, `read` essai de lire autant d'octets que peux en contenir une variable de type `INDIV` depuis le descripteur `hfile` ouvert sur le fichier `contacts.dta`. Il stocke le résultat à l'adresse de `indiv` (dans le code `&indiv`). Puis il renvoi la taille lu, soit la taille d'un contact **ou** 0 **ou** -1.



Question ?

```
while ( read( hfile, &indiv, sizeof(INDIV) ) ) {
      nbr++ ;
      afficher( &indiv ) ;
}
```

Dans quel cas l'instruction `while` fait-elle une itération de plus (une boucle de plus) ?

Question réflexion ?

A la fin de la boucle la taille de `indiv` a-t-elle augmentée ?

Stockage des données

Nous en venons maintenant au coeur de sujet de notre TP de découverte de la séquence.

Pour travailler dans une application sur des quantités dynamique de données il faut les stocker en mémoire. On peut les stocker dans des tableaux, dans des arbres, dans des listes chainées, dans des vecteurs, ... Dans plein de choses qui permettent ensuite de travailler sur ces données comme ajouter un éléments, supprimer un éléments, les trier, les afficher.

Ces structures de données seront travaillées en cours. Dans un premier temps nous allons simplement les stocker dans un tableau. Cela semble le plus simple... détrompez vous, ça va être probablement assez difficile !

A chaque boucle vous devrez faire appel à la fonction `ajouter()`. Cette fonction prendra en argument un pointeur sur un `INDIV`, un peu comme la fonction `afficher()`.

Cette fonction devra stocker l'individu courant dans le tableau `tab`. La problématique est que la taille du tableau n'est pas connu à l'avance... vous devrez donc utiliser la fonction `realloc()`.

A FAIRE : créer un pointeur nommé `tab` permettant de stocker votre tableau dynamique d'`INDIV`. C'est une variable vous déclarerez global.

A FAIRE : créer un entier nommé `nbIndiv` qui vous permettra de connaître le nombre d'individu dans votre tableau. C'est une variable vous déclarerez global.

A FAIRE : créer la fonction `ajouter()` proposition de démarrage ci dessous.

```
void ajouter( INDIV* p ){

      // realloc() de tab avec un INDIV de plus
      // et vérification qu'il n'y ai pas d'erreur dans l'allocation

...

      // copie de p dans tab (au bon endroit)
...

}
```

A FAIRE : modifier la fonction `chargement()` pour qu'elle appel votre fonction `ajouter()` à chaque tour de boucle (vous pouvez remplacer `afficher()`).

A FAIRE : ajouter à la fin du main l'instruction permettant de libérer la mémoire de `tab`.

Normalement si tout vas bien vous n'avez pas de `Segmentation fault`. Mais vous ne pouvez pas vérifier votre travail ! C'est l'objet de la question suivante.

Affichage du tableau

Nous allons maintenant nous intéresser à l'affichage des données. La fonction `afficheTout()` permettra d'afficher les données. Elle ajoutera un id devant le contact, ceci permettra d'accéder plus rapidement à une fiche.

Tel que:

```
...
 2 : VERSAIRE     Annie    78, rue du Gâteau 0316780150
...
```

A FAIRE : créer la fonction `afficheTout()` proposition de démarrage ci dessous.

```
void afficheTout()
{
      // boucle de parcours de tab et appel à afficher() et affichage de l'id

}
```

A FAIRE : appeler la fonction `afficheTout()` à la fin du main (voir commentaire TODO).

Utilisation des données

Maintenant que nous sommes capable de lire les données en provenance du fichier et que nous pouvons les stocker, nous allons utiliser ces données.

Ajout de contact

Nous allons maintenant permettre d'ajouter un individu à nos données.

Pour cela nous allons utiliser une fonction nouveau(). La saisie doit se faire au clavier sans toutefois dépasser la taille maximum des champs de la structure du type INDIV. Ceci permettra de ne pas faire d'overflow.

La proposition est la suivante :

```
void  nouveau()
{
      INDIV indiv ;
      memset( &indiv, 0, sizeof( INDIV ) ) ;

      puts("Nouveau contact :" ) ;
      fflush( stdin ) ;

      printf("Nom     ? " ) ;
TODO ...
      printf("Prenom  ? " ) ;
TODO ...
      printf("Adresse ? " ) ;
TODO ...
      printf("Tel     ? " ) ;
TODO ...
      printf("Email   ? " ) ;
TODO ...

      if ( indiv.nom[0] ) {
            ajouter( &indiv ) ;
      }
}
```

A FAIRE : completer la fonction `nouveau()` et l'inclure dans le fichier `mainTest.c`.

A FAIRE : appeler la fonction `nouveau()` à la fin du main mais avant l'affichage.

Afficher un contact à partir de son id

Nous allons maintenant permettre d'afficher un individu particulier.

Pour cela nous allons créer une fonction `afficheId()`.

La proposition est la suivante :

```
int   afficheId()
{
```

```
...
      // saisie du numéro de fiche à afficher
      // et vérification de l'existence de cette fiche
...


      // affichage de la fiche
...

      // retourne le numéro de fiche
...
}
```

A FAIRE : compléter la fonction `afficheId()`.

A FAIRE : Inclure dans le fichier `mainTest.c` pour test.

Editer un contact à partir de son id

Nous allons maintenant permettre d'éditer un individu particulier.

Pour cela nous allons utiliser la fonction `afficheId()` que nous avons déjà écrite et nous allons créer la fonction edition(). Cette fonction va faire appel à affiche id puis vas permettre de modifer les champs.

La proposition est la suivante :

```
void edition()
{
      // appel à afficheId pour récupérer l'id Contact
...

      // affichage des champs et modification de ces derniers.
...



      // affichage de la nouvelle fiche
...
}
```

A FAIRE : completer la fonction `edition()`.

A FAIRE : remplacer `afficheId()` par `edition()` dans le fichier `mainTest.c`.

Supprimer un contact à partir de son id

Proposer une solution pour la suppression d'un contact en utilisant afficheId().

```
void supprimer()
{
	...
}
```

Explication de la solution

Trier le tableau de contact

Proposer une solution pour le trie les données par ordre alphabétique nom puis prénom.

```
void trier()
{
	// tri à bulle
	...
}
```

Explication de la solution