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

Types de données
Version « Liste Chainée »
Pourquoi et Comment ?

Indicateur temporel :

questions	1h	2h	3h	4h	5h	6h
Ajouter						
Maillon						
Insérer						
Extraire / détruire						

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

Idem TP1

Cahier des charges et données.

Pour mémoire, ce type `INDIV` se décrit comme ceci dans « indiv.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 ;
```

Objectif général

Vous allez créer votre librairie de liste chainée.

Son fichier d'entête est le suivant :

```
#ifndef     LSDYN_H
#define     LSDYN_H

#include    "indiv.h"

void  ajouter( TOBJ* obj ) ;          /* ajout d'un maillon en fin de liste */
int   taille( void ) ;                /* retourne le nombre courant de maillons */
TOBJ* maillon( int pos ) ;            /* pointeur sur données utiles d'un maillon */
void  detruire( void ) ;              /* suppression de tous les maillons */
void  inserer(TOBJ* obj, int pos ) ;
TOBJ* extraire( int pos ) ;

#endif
```

Pour que cette librairie fonctionne nous ajoutons `typedef INDIV TOBJ ;` dans `indiv.h`. Ceci permettra de rendre le plus générique possible notre librairie.

Vous fonctions seront implémentée dans lsdyn.c.

les cellules de la liste chainées sont ainsi :

```
typedef struct elem {

      TOBJ obj ;                      /* objet utile défini par le client */
      struct elem* suiv ;             /* lien sur maillon suivant */
} ELEM ;                              /* type maillon */

ELEM* deb = NULL ;                    /* pointeur sur le début de liste */
ELEM* fin = NULL ;                    /* pointeur sur le dernier maillon */
int nbelem = 0 ;                      /* nombre courant de maillons */
```

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 ?

---

void ajouter(TOBJ* obj)

Implémenter la fonction ajouter().

```
void  ajouter( TOBJ* obj )
{
      ELEM* p;
      // Allocation de l'ELEM

      // INIT de "suiv"


      // copy du TOBJ avec memcpy

      // placement de l'ELEM à la fin de la liste


      // incrémentation du nombre d'élément
}
```

TOBJ*maillon(int pos)

Implémenter la fonction maillon().

```
TOBJ* maillon( int pos )
{
      ELEM* p ;

      // parcours avec recherche du maillon



}
```

void inserer(TOBJ* obj, int pos)

Implémenter la fonction inserer().

```
void  inserer(TOBJ* obj, int pos )
{
      ELEM* p ;

      // Allocation de l'ELEM

      // INIT de "suiv"


      // copy du TOBJ avec memcpy

      // placement de l'ELEM au bon endroit


      // incrémentation du nombre d'élément
}
```

TOBJ*extraire(int pos)

Implémenter la fonction extraire().

```
TOBJ* extraire( int pos )
{
	// extraction de l'élément à la position pos

	// nbelem --

	// retour de l'élément

}
```

void detruire(void)

Implémenter la fonction detruire().

```
void  detruire( void )
{
	// détruire tout
}
```