Programmation II : SMI (S4)

TP N° 1 : Les Types composés

Partie I : Exercices préliminaires

Exercice 1 :

Soit **T** un tableau de **N** entiers non trié. Écrire un programme qui affiche l'élément qui apparaît le plus souvent dans le tableau **T**, ainsi que son nombre d'occurrences. Si plusieurs éléments différents répondent au problème, votre programme doit en fournir un, quel qu'il soit. Vous ne devez utiliser aucun autre tableau que celui sur lequel vous travaillez.

Exercice 2 : Polynôme

Ecrire un programme qui calcul et affiche la valeur d'un polynôme **P** de degré **n** en un point **x** donné. Les coefficients (a_n, a_{n-1}, …., a_0) de **P** sont contenus dans le tableau :

t (**t[0]**=a_0, **...**, **t[n]**=a_n) .

Schéma normal : $P(x) = a_n*x^n + a_{n-1}*x^{n-1} + ... + a_1*x^1 + a_0$

Schéma de Horner : $P(x) = (((a_n*x + a_{n-1})*x + ...)*x + a_1)*x + a_0$

Exercice 3 : Structures

1 On souhaite créer une structure **Point** pour représenter un point dans l'espace. Cette structure doit contenir les coordonnées réels **x**, **y** et **z** du point.

- ⇨ Donnez la définition de cette structure.
- ⇨ Ecrire un programme qui :
  - crée deux points et les initialise avec des réels.
  - affiche les deux points sous la forme : **(6.9, 7.3, 4.2)**
  - calcule la distance entre les deux points.

2 On souhaite aussi créer une structure **Segment** pour représenter un segment de droite défini par deux points (structure **Point**).

- ⇨ Donnez la définition de cette structure.

⇨ Ecrire un programme qui :

- crée un segment à partir de deux points.
- affiche le segment sur comme suit :

  [(6.9, 7.3, 4.2) --- (6.9, 7.3, 4.2)]

3. On souhaite par ailleurs créer une structure **Ligne** pour représenter une ligne brisée constituée de plusieurs points. Cette structure doit contenir le nombre de points manipulés **N** et un tableau automatique de points de taille constante **TMAX**.

⇨ Donnez la définition de cette structure.

⇨ Ecrire un programme qui :

- crée une ligne brisée et l'initialisé par les trois point **(3.2, 5.0, 1.0), (4.6, 3.0, 0.0), (2.0, 2.5, 3.0)**.
- affiche la ligne en affichant l'ensemble de points de la ligne.
- calcule et affiche la longueur de la ligne.

Partie II : Problème (Gestion d'une pharmacie)

On souhaite gérer une **pharmacie** ayant une liste de **clients** et une liste de **médicaments** en créant des structures **C** (**Client**, **Medicament** et **Pharmacie**) :

- le client est représenté par une structure **Client** ayant un nom (**chaine**) et un crédit (**double**). Le crédit représente la somme que **doit** le client à la pharmacie. Le crédit peut être négatif si le client a versé plus d'argent que le montant à payer.
- le médicament est représenté par une structure **Medicament** ayant un libelle (**chaine**), un prix (**double**) et une quantité en stock (**int**).
- la pharmacie est représenté par une structure **Pharmacie** ayant un tableau de client (**tabC**) de taille **MAXC** et un tableau de médicament (**tabM**) de taille **MAXM**. La pharmacie est définie aussi par le nombre de clients ajoutés à la pharmacie (**nbC**) et le nombre de médicaments ajoutés à la pharmacie (**nbM**).

1 Donner la définition des structures **Client**, **Medicament** et **Pharmacie** :

2 En utilisant les structures définies, écrire un programme **main** qui permet de :

a. Créer 2 clients et les initialisés avec les données suivantes :

	no	credit
Client c1 ;	Hamid	300
Client c2 ;	Aicha	-400

b. Créer 3 médicaments et les initialisés avec les données suivantes :

	libell	prix	qteEnStock
Medicament m1 ;	Cataflame	49	14
Medicament m2 ;	Doliprane	22	16
Medicament m3 ;	Baycutene	25	6

c. Créer 1 pharmacie et l'initialisé avec les données suivantes :

	Solde
Pharmacie ph ;	3000

d. Ajouter les clients à la pharmacie.

e. Réaliser un achat des médicaments par la pharmacie.

f. Afficher l'ensemble des clients à partir de la pharmacie crée.

g. Afficher l'ensemble des médicaments à partir de la pharmacie crée.

h. Réaliser un achat de la pharmacie du médicament m1.

i. Réaliser un vent de la pharmacie du médicament m2 au client c1.

j. Afficher l'ensemble des clients à partir de la pharmacie crée.

k. Afficher l'ensemble des médicaments à partir de la pharmacie crée.

TP N° 2 : Les Pointeurs

Partie I : Exercices préliminaires

Exercice 1 :

Compléter le tableau en indiquant les valeurs des différentes variables au terme de chaque instruction du programme suivant (on peut aussi indiquer sur quoi pointent les pointeurs):

programme	a	b	c	p1	*p1	p2	*p2
`int a, b, c, *p1, *p2 ;`	?	?	?	?	?	?	?
`a = 1, b = 2, c = 3 ;`							
`p1 = &a, p2 = &c ;`							
`*p1 = (*p2)++ ;`							
`p1 = p2 ;`							
`p2 = &b ;`							
`*p1 -= *p2 ;`							
`++*p2 ;`							
`*p1 *= *p2 ;`							
`a = ++*p2 * *p1 ;`							
`p1 = &a ;`							
`*p2 = *p1 /= *p2 ;`							

Exercice 2 :

Soit la structure **`Matrice`** représentant un type matrice d'entier:

```
typedef struct Matrice {
     int nbLignes ;
     int nbColonnes;
     int ** mat;
} Matrice;
```

Ecrire un programme qui :

1. Crée une matrice A et lui alloué l'espace de mémoire nécessaire.
2. Initialise la matrice A par des données saisies par l'utilisateur.
3. Affiche la matrice A.
4. Libère l'espace mémoire de la matrice A.

Partie II : Problème (Gestion de la pharmacie)

Reprendre la partie II du TP N° 1 en transformant le tableau automatique de client et le tableau de médicament de la structure **`Pharmacie`** sous forme d'un pointeur qui sera alloué dynamiquement au besoin avec une **`taille`** définie par l'utilisateur.

La structure **`Pharmacie`** aura donc la forme suivante :

```
typedef struct Pharmacie {
    double solde;
    Client * tabC;
    int MAXC;
    int nbC;
    Medicament * tabM;
    int MAXM;
    int nbM;
} Pharmacie;
```

TP N° 3 : Les Fonctions et la récursivité

Partie I : Exercices préliminaires

Exercice 1 :

On souhaite écrire un programme qui permet de modéliser des comptes bancaires. Définir la structure **`CompteBanque`**, qui permet de représenter des comptes bancaires. Cette structure se compose des champs suivants : numéro du compte (**`long`**), nom du client (**`chaine`**) et solde du compte (la somme d'argent en banque : **`double`**). Le numéro de compte est créé automatiquement, séquentiellement, à partir de 101 (101, 102, 103, …). Une variable global **`NextNumCpte`** initialisée à 100 sera incrémenté et affecté au numéro du compte nouvellement crée.

Définir les fonctions suivantes utilisant la structure **`CompteBanque`** :

- **`void initCB(CompteBanque * adrCB, char n[], double s)`** qui permet d'initialiser le nom et le solde d'un compte d'adresse **`adrCB`** à partir des paramètres : **`n`** et **`s`**.
- **`void crediter(CompteBanque * adrCB, double som`**) qui permet d'augmenter le solde d'un compte par une somme.
- **`void debiter(CompteBanque * adrCB, double som`**) qui permet de retirer une somme d'un compte dans la limite du solde.
- **`void virer(CompteBanque * adrCBSrc, CompteBanque * adrCBDest, double som)`** qui permet de transférer une somme depuis un compte source vers un compte destination.
- **`void afficher(CompteBanque cb`**) qui permet d'afficher les information d'un compte sous la forme : **`Alaoui, Compte numéro: 121, Solde : 15000 dh`**

Écrire le programme **`main`** qui teste la structure compte avec les différentes fonctions.

Partie II : Problème (Gestion de la pharmacie)

L'objectif de ce TP est d'alléger le programme **`main`** de la partie II du TP N° 2 en réécrivant les opérations effectué sur les structures sous forme d'appel de fonctions. Pour cela, on va définir les fonctions suivantes :

Pour la structure **`Client`** ajouter les fonctions suivantes :

- **`void initClient(Client * adrC, char n[], double s)`** qui permet d'initialiser le **`nom`** et le **`solde`** du client d'adresse **`adrC`** par les paramètre **`n`** et **`s`** respectivement.

- **`void afficherClient(Client c)`** qui permet d'afficher le client **`c`** passé en paramètre et son crédit respectif.

Pour la structure **`Medicament`** ajouter les fonctions suivantes :

- **`void initMedicament(Medicament * adrM, char lib[], double prix, int qteS)`** qui permet d'initialiser le **`libelle`**, le **`prix`** et la **`quantité en stock`** du médicament d'adresse **`adrM`** par les paramètre **`lib`**, **`pr`** et **`qteS`** respectivement.
- **`void afficherMedicament(Medicament m)`** qui permet d'afficher le médicament m passé en paramètre et sa quantité en stock respectif. .

Pour la structure **`Pharmacie`** ajouter les fonctions suivantes :

- **`enregistrerClient(…)`** permet d'ajouter un client à la liste des clients de la **`pharmacie`**.
- **`enregistrerMedicament(…)`** permet d'ajouter un médicament à la liste des médicaments de la pharmacie.
- **`approvisionner(…)`** permet d'approvisionner le stock d'un médicament :
    - Le nom du médicament à approvisionner ainsi que la quantité à ajouter au stock doivent être passés en paramètre. Lorsque le nom du médicament est passé, il faut vérifier qu'il s'agit bien d'un nom connu dans la liste des médicaments de la pharmacie.
        - ⇨ Vous pouvez prévoir une méthode de vérification **`verifMedicament(…)`**.
- **`achat(…)`** permet de traiter un achat fait par un client. L'achat porte sur un ***médicament donné*** dans une ***quantité donnée***. Pour cette transaction le client paie un certain prix. Une opération d'achat aura pour effet de déduire la quantité achetée du stock du médicament correspondant et débiter le crédit du client (d'un montant équivalent au montant de l'achat moins la somme payée).

  Les **noms du client et** du **médicament** doivent être passés en paramètres. Vous pouvez prévoir une méthode de vérification **`verifAchat(…)`**. La quantité achetée et le montant payé sont aussi passés en paramètres et seront supposés corrects.

Reprendre le programme **`main`** de TP N° 2 en se basant sur les appels des fonctions définis.

TP N° 4 : Les Fichiers

Partie I : Exercices préliminaires

Exercice 1 :

Soit un fichier de données texte "**etudiants.txt**" structuré en une suite de lignes contenant chacune le nom de l'étudiant, le prénom de l'étudiant, le semestre d'inscription de l'étudiant et la moyenne obtenue. La première ligne contenant le nombre d'étudiants enregistré dans le fichier.

Exemple :

```
4
Hamid       Tahiri          S1      12
Merieme     Karimi          S1      10
Aicha       El Gharbaoui    S3      7
Ali         Mhamdi          S2      11
```

1. Créer une structure **`Etudiant`** représentant les différentes informations figurant dans chaque ligne du fichier texte.
2. Ecrire une fonction qui récupère les données du fichier et les stockes dans un tableau de structure **`Etudiant`**.

Partie II : Problème (Gestion de la pharmacie)

L'objectif de ce TP est de modifier la partie II du TP N° 3 en remplaçant la création des objets clients en mémoire en les enregistrant dans des fichiers de données binaires.

Les fonctions envisagées sont :

- **`void enregisterClient(Client c, char fichClient[])`** : qui permet d'enregistrer le client passé en paramètre dans le fichier de données binaire **`fichClient`**.
- **`int nombreClients(char fichClient[])`** : qui permet de retourner le nombre de clients enregistrés dans le fichier **`fichClient`**.
- **`void listerClients(char fichClient[])`** : qui permet de récupérer l'ensemble des clients enregistrés dans le fichier de données binaire **`fichClient`** et les afficher.
- **`void lireClients(char fichClient[], Client * tabC)`** : qui permet de récupérer l'ensemble des clients enregistrés dans le fichier de données binaire **`fichClient`** et les stocker dans un tableau de structure **`tabC`**.

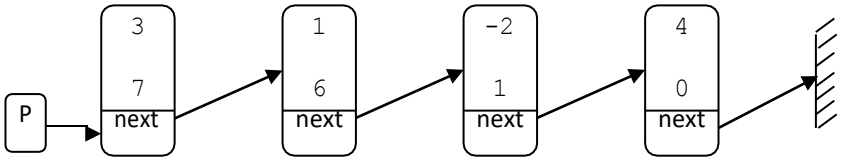
TP N° 5 : Les listes chainées

L'objectif de ce TP est de représenter un polynôme à coefficient réels par une liste chaînée de monômes (nœuds de la liste) définis par la structure suivante :

```
typedef struct Monome * Polynome;
typedef struct Monome{
      float  coeff;
      int   puiss;
      Polynome next;
}Monome;
```

Un polynôme **P** est une liste simplement chaînée de **Monome** dont l'adresse du premier élément est dans **P**.

Exemple : Le polynôme P(x) = $3x^7 + x^6 - 2x + 4$ sera représenté par la liste chainée suivante :



1 Écrire une fonction qui permet de créer un **Monome** et l'initialiser par un coefficient et une puissance passés en paramètre. La fonction retournera l'adresse du monôme créé et aura l'entête suivant :
`Monome * creerMonome(float coef, int exp)`

2 Écrire une fonction qui permet de créer et de saisir les monômes d'un polynôme, puis retourne l'adresse du premier élément. La saisie s'arrête lors de la saisie d'un exposant négatif. La fonction aura l'entête suivant : **`Polynome saisirPolynome()`**

3 Écrire une fonction qui affiche le polynôme **P** dont l'entête est :
`void afficherPolynome(Polynome P)`
L'affichage du polynôme P(x) de l'exemple sera comme suit :
`3x^7 + 1x^6 -2x^1 + 4x^0`
(Pensez à améliorer encore l'affichage).

4 Ecrire une fonction qui renvoie la valeur du polynôme **P** pour une valeur **x** dont l'entête est : **`float evaluerPolynome(Polynome P, float x)`**

5 Ecrire une fonction qui renvoie la dérivée du polynôme passé en paramètre dont l'entête est : **`Polynome deriverPolynome(Polynome P)`**
Exemple : la dérivée du polynôme P(x) est P'(x) = $21x^6 + 6x^5 - 2$

6 Écrire une fonction qui renvoie la somme de deux polynômes P et Q dont l'entête est : **`Polynome sommePolynome(Polynome P, Polynome Q)`**.

7. Écrire une fonction qui renvoie le produit d'un polynôme P par un monôme **m** dont l'entête est **`Polynome produitPolynomeMonome(Polynome P, Monome m)`**.

8 Écrire une fonction qui renvoie le polynôme résultat de la multiplication du polynôme P par le polynôme Q dont l'entête est **`Polynome produitPolynomes(Polynome P, Polynome Q)`** (penser à utiliser les deux dernières fonctions).

TP N° 6 : Les Piles

(Evaluation d'une expression arithmétique)

Une expression arithmétique est un ensemble d'opérandes reliés par des operateurs. L'écriture habituelle des expressions arithmétique est appelée ***notation infixée***, par la suite on s'intéresse à des expressions ***infixées*** complètement parenthèsées comme, par exemple,

`(((3+4)*8)-6)` ou **`(3+(4*(8-6)))`**

L'évaluation d'une expression en ***notation infixée*** peut se faire en deux étapes.

- Conversion en écriture ***postfixée***.
- Evaluation de l'expression ***postfixée***.

Conversion en notation Postfixe

Une expression peut être représentée par une chaine de caractères :

Exemple : l'expression **`(3+(4*(8-6)))`** est représentée comme suit :

`(`	`3`	`+`	`(`	`4`	`*`	`(`	`8`	`-`	`6`	`)`	`)`	`)`	`\0`

On considère une chaine de caractères **`expInf`** représentant une expression, la conversion de cette écriture en écriture ***postfixée*** se fait à l'aide d'une pile d'opérateurs selon les étapes suivantes:

- Création d'une chaine vide **`expPost`**
- Création d'une Pile d'éléments de type caractère
- Parcourir l'expression **`expInf`**
    - Si on rencontre une parenthèse ouvrante '**`(`**', on ne fait rien
    - Si c'est un opérande il est recopié directement dans la chaine **`expPost`**.
    - Si c'est un operateur (**`+`**, **`-`**, **`*`**, **`/`** ou **`%`**) il est empilé.
    - à chaque lecture d'une parenthèse fermante '**`)`**', on retire un opérateur de la pile et on le recopie dans **`expPost`**.

A la fin du parcours, la chaine **`expPost`** contient la ***notation postfixée*** de l'expression **`expInf`.**

Exemple :

Si **`expInf=(3+(4*(8-6)))`** alors **`expPost = 3 4 8 6 - * +`**

Evaluation de l'expression postfixée

Pour simplifier, on se limite aux opérandes entiers compris entre **`0`** et **`9`**, et pour limiter les problèmes d'arithmétique, on n'utilisera que les opérateurs **`+`**, **`-`**, **`*`** et **`/`**.

La ***notation postfixée*** s'évalue très simplement à l'aide d'une pile d'opérandes :

- les opérandes sont empilés au fur et à mesure de leur lecture

- les opérateurs s'appliquent immédiatement en retirant les deux opérandes situés au sommet de la pile et en y replaçant le résultat.
- le résultat final est la seule valeur qui reste dans la pile.

Exemple : l'évaluation de « **`3 4 8 6 - * +`** » donne **`11`**

L'expression	**`3`**	**`4`**	**`8`**	**`6`**	**`-`**	**`*`**	**`+`**
L'état de la pile				**`6`**			
			`8`	**`8`**	**`2`**		
		`4`	**`4`**	**`4`**	**`4`**	**`8`**	
	`3`	**`3`**	**`3`**	**`3`**	**`3`**	**`3`**	**`11`**

Et l'évaluation de **`3 4 + 8 * 6 -`** (en ***notation infixée*** **`(((3+4)*8)-6))`** donne **`50`**.

Travail à faire :

- Définir la structure d'une pile des opérandes **`PileOpd`**.
- Ecrire la fonction d'entête **`void initPileOpd(PileOpd & adrP)`** qui permet d'initialiser une pile vide d'opérandes d'adresse **`adrP`**.
- Ecrire la fonction d'entête **`void empilerOpd(PileOpd * adrP, int x)`** qui permet d'ajouter un élément **`x`** dans la pile d'adresse **`adrP`**.
- Ecrire la fonction d'entête **`int depilerOpd(PileOpd * adrP)`** qui permet de retirer et retourner le sommet de la pile.
- Refaire les questions précédentes avec une pile des operateurs **`PileOpr`** (*il faut changer les noms de fonctions*)
- Ecrire la fonction **`char * convertirInf2Post(char * expInf)`** qui prend en paramètre une expression en ***notation infixée*** et retourne sa conversion en ***notation postfixée***.
- Ecrire la fonction **`int calculer(int x, int y, char op)`** qui prend en paramètre deux opérandes **`x`** et **`y`** et un opérateur **`op`** sous forme de caractère pour calculer l'expression x op y :
  Exemple : l'appel **`calculer(3,7,'+')`** retournera la valeur **`10`**.
- Ecrire la fonction **`int evaluer(char * exp)`** qui prend en paramètre une expression ***postfixée*** et retourne un entier qui est le résultat de l'évaluation de **`exp`**, pour cela :
  - Convertir **`exp`** en une notation postfixée **`expPost`**
  - Créer une pile vide des entiers pour empiler/dépiler les opérandes.
  - Parcourir **`expPost`** en l'évaluant suivant la procédure décrite ci-dessus.
  - Retourner le résultat
- Ecrire la fonction principale **`main()`** qui permet de :
  - Déclarer et lire une expression infixée complètement parenthèsée **`expInf`**,
  - Evaluer **`expInf`**.

:

- **`int isdigit(int C)`** : retourne une valeur différente de zéro, si C est un chiffre décimal (**`ctype.h`**).
- **`int atoi(const char *CH)`** : retourne la valeur numérique représentée par CH comme int (**`stdlib.h`**).

TP N° 7 : Les Files

Exercice 1

Écrire une fonction **struct File * alloue_file(void)** qui crée une file vide.

Exercice 2

Écrire une fonction **int est_vide(struct File *f)** qui teste si une file est vide.

Exercice 3

Écrire une fonction **void empile(int x, struct File *f)** qui empile l'entier x à la fin de la file f.

Exercice 4

Écrire une fonction **int depile(struct File *f)** qui retire de la file f son premier élément, et le renvoie en résultat.

Exercice 5

Écrire une fonction **void detruit_file(struct File *f)** qui désalloue une file.