NF16 - TP 3 – Listes chaînées

Ce TP est basé sur le problème du médian de NF16 proposé à l'Automne 2017. Il a pour objectif de se familiariser avec les listes chaînées et les différentes opérations nécessaires pour les manipuler.

Enoncé:

L'UTC décide d'introduire sa propre monnaie, sur le modèle du Bitcoin, pour permettre aux étudiants d'acheter des repas à la cafétéria : le *EATCoin*.

Le prix de chaque repas proposé est exprimé en *EATCoin*. Chaque étudiant peut :

- Recharger son compte (créditer des *EATCoin*) ;
- Acheter un repas (débiter des *EATCoin*).

En s'inspirant toujours du modèle du Bitcoin, l'UTC décide de sauvegarder toutes les informations sur l'état des comptes des étudiants au sein d'une *BlockChain*. Celle-ci fonctionne de la manière suivante :

- Chaque fois qu'un étudiant recharge son compte ou achète un repas, l'opération est enregistrée sous la forme d'une *transaction* ;
- La liste de toutes les transactions qui se déroulent dans une même journée est enregistrée dans un *bloc* ; un nouveau bloc est créé chaque jour.
- La liste de tous les blocs forme la *chaîne de blocs* (*BlockChain* en anglais)

Un étudiant est identifié par un numéro unique.
Une transaction est caractérisée par :

- L'identifiant de l'étudiant - *de type int*
- Le montant en *EATCoin* payé ou reçu (par exemple 10 si l'étudiant vient de créditer 10 *EATCoin* sur son compte ou -3.5 si l'étudiant a acheté un sandwich à 3.5 *EATCoin*) - *de type float*
- Une description (par exemple « Achat Sandwich »)

Un bloc est caractérisé par :

- Un identifiant unique de bloc - *de type int*
- La date du bloc sous la forme *AAAAMMJJ* (ex : 20191014 pour le 14/10/2019)
- La liste des transactions de la journée correspondant au bloc

A. Structures

1. Définir la structure **Transaction** et le type correspondant **T_Transaction**.
2. Définir la structure **Block** et le type correspondant **T_Block**.
3. Définir le type **BlockChain** qui représente la liste de tous les blocs.

B. Fonctions requises

1. Ajout d'une transaction en tête d'une liste de transactions :

T_Transaction *ajouterTransaction(int idEtu, float montant, char *descr, T_Transaction *listeTransaction)

Cette fonction renvoie un pointeur vers le premier élément de la liste.

2. Ajout d'un bloc en tête de la *BlockChain* :

BlockChain ajouterBlock(BlockChain bc)

L'identifiant du nouveau bloc sera égal à l'identifiant du bloc suivant incrémenté de 1 (ou 0 s'il s'agit du premier bloc). Initialement, le bloc ajouté contiendra une liste vide de transactions.

3. Calcul de la somme des *EATCoin* crédités et dépensés par un étudiant sur une journée :

float totalTransactionEtudiantBlock(int idEtu, T_Block b)

4. Calcul du solde total d'un étudiant:

float soldeEtudiant(int idEtu, BlockChain bc)

5. Rechargement du compte d'un étudiant :

void crediter(int idEtu, float montant, char *descr, BlockChain bc)

Lorsqu'un étudiant recharge sa carte, une nouvelle transaction est ajoutée dans le bloc de tête de la B*lockChain*.

6. Paiement d'un repas :

int payer(int idEtu, float montant, char *descr, BlockChain bc)

Lorsqu'un étudiant souhaite payer un repas, une vérification est d'abord effectuée pour voir si le nombre de *EATCoin* qu'il possède est suffisant. Si c'est le cas, une nouvelle transaction est ajoutée à la *BlockChain*. Dans le cas où le solde de l'étudiant est insuffisant, la fonction renverra 0 (1 sinon).

7. Historique d'un étudiant :

void consulter(int idEtu, BlockChain bc)

On affichera le solde de l'étudiant ainsi que les informations sur les 5 dernières transactions réalisées : la date de la transaction, sa description et son montant.

8. Transfert de *EATCoins* entre deux étudiants :

int transfert(int idSource, int idDestination, float montant, char *descr, BlockChain bc)

La fonction retournera 1 si le transfert a été effectivement réalisé, 0 sinon.

C. Programme Principal :

Utiliser les fonctions précédentes pour proposer à l'utilisateur le menu interactif suivant :

1. Afficher la liste des blocs de la BlockChain
2. Afficher toutes les transactions d'un bloc
3. Afficher toutes les transactions du jour pour un étudiant
4. Afficher l'historique pour un étudiant
5. Créditer un compte
6. Payer un repas
7. Transférer des EATCoins entre deux étudiants
8. Quitter

N.B : votre programme devra également gérer une date courante, initialisée avec une date donnée; on pourra ensuite avancer cette date d'un ou plusieurs jours afin de pouvoir tester l'ajout de transactions sur plusieurs journées différentes.

Consignes générales :

- **Sources**

À la fin du programme, les blocs de mémoire dynamiquement alloués doivent être proprement libérés. Vous devrez également être attentifs à la complexité des algorithmes implémentés.

L'organisation MINIMALE du projet est la suivante :

- Fichier d'en-tête tp3.h, contenant la déclaration des structures/fonctions de base,
- Fichier source tp3.c, contenant la définition de chaque fonction,
- Fichier source main.c, contenant le programme principal.

- **Rapport**

Votre rapport de quatre pages maximum contiendra :

- La liste des structures et des fonctions supplémentaires que vous avez choisi d'implémenter et les raisons de ces choix.
- Un exposé succinct de la complexité de chacune des fonctions implémentées.

Votre rapport et vos fichiers source feront l'objet d'une remise de devoir sur Moodle dans l'espace qui sera ouvert à cet effet quelques jours suivant votre démonstration au chargé de TP (un seul rendu de devoir par binôme).