A23

Rapport TP3 : Listes chaînées

## 1/ La liste des structures et des fonctions supplémentaires que vous avez choisi d’implémenter et les raisons de ces choix :

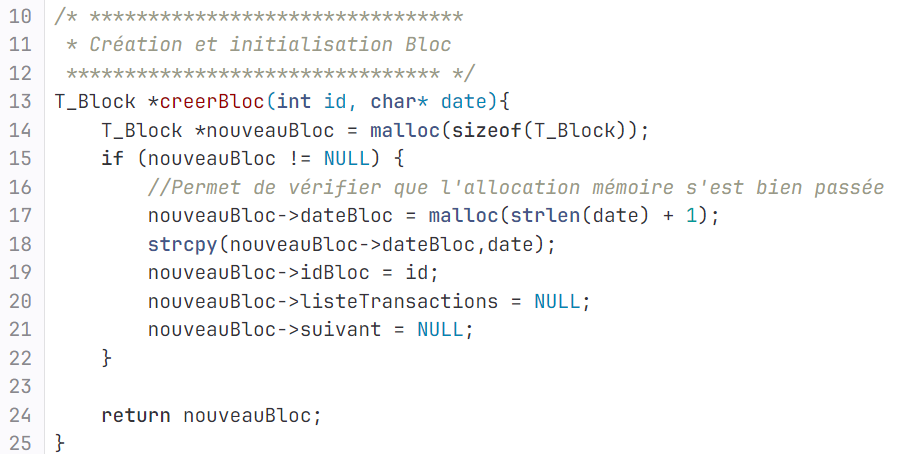
Nous avons une seule fonction supplémentaire qui s’appelle « libererEspace ». Nous aurions pu en ajouter une autre mais nous avons préféré développer le code de certains choix dans le main.c, notamment pour le choix 8, où nous avons développé un code permettant changer la date du jour et ainsi créer des blocs pour chaque jour.

Nous avons défini dateJ comme un char\* commun à tous les fichiers et initialisé avec la date du jour. Cette date s’affiche au lancement du programme et elle est modifiable grâce au choix 8.

Dans **libererEspace**, nous avons écrit des printf permettant de suivre le bon déroulement de l’opération de suppression.

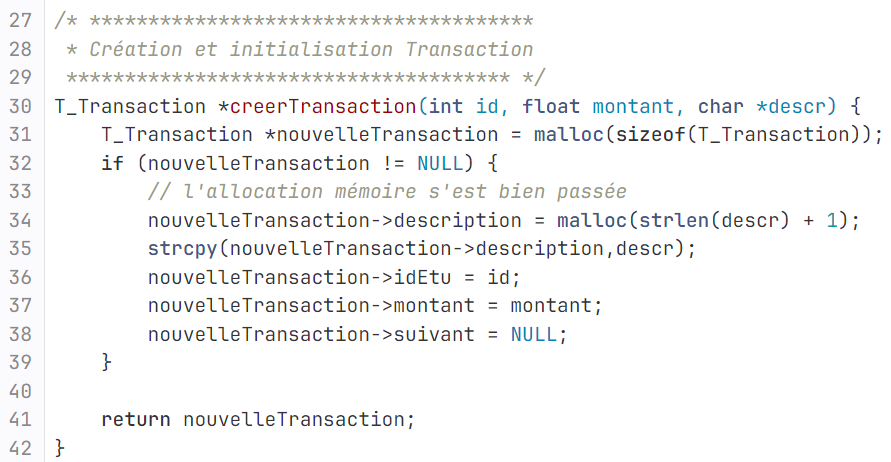
Nous observons que nous parcourons chaque liste de chaque blocs. Définissons la taille d’un bloc comme M et la taille d’une liste comme N. Ainsi le while (listeT1 != NULL) {…} est d’une complexité de O(n). Or pour parcourir toute la liste des blocs, il faut parcourir M blocs. Donc la complexité totale de libererEspace est de **O(m\*n).**

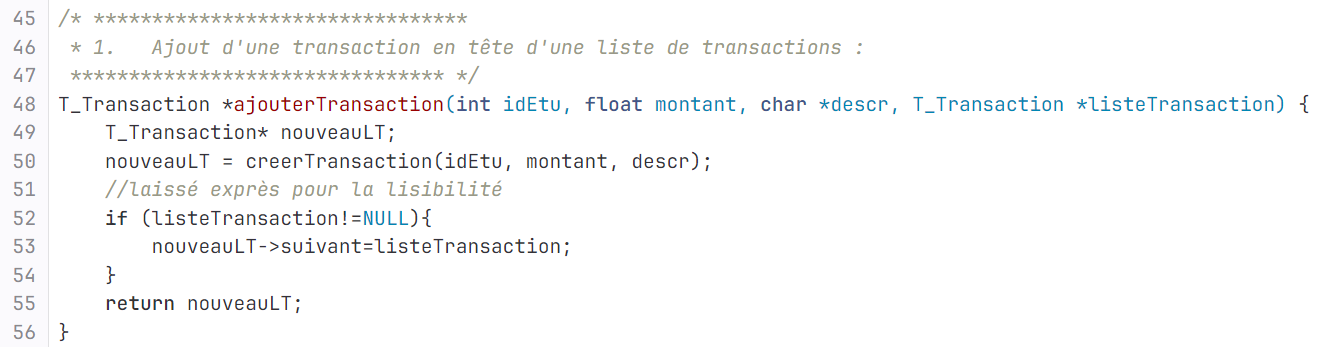
## 2/ Un exposé succinct de la complexité de chacune des fonctions implémentées :

Dans la fonction **creerBloc**:

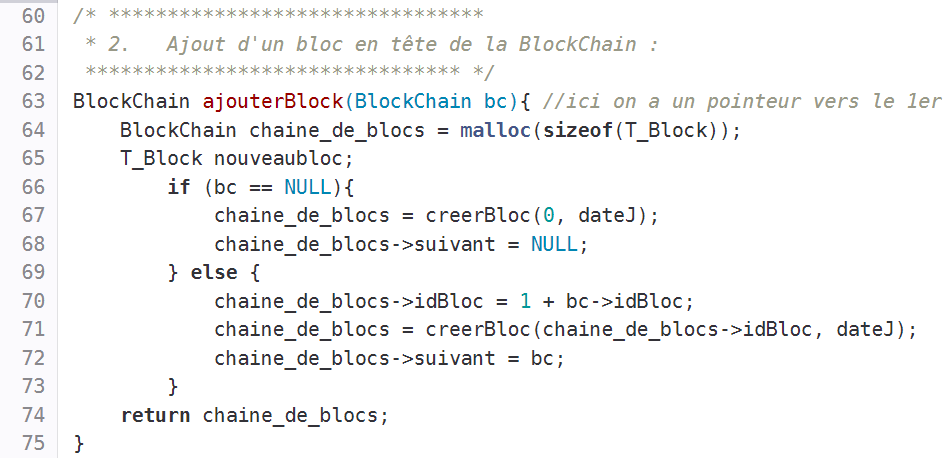
* la ligne 14 à une complexité O(1).
* les lignes 17 à 21 ont aussi une complexité d’O(1).
* la ligne 15 est aussi d’une complexité d’O(1).
* il en va de même pour la ligne 24.

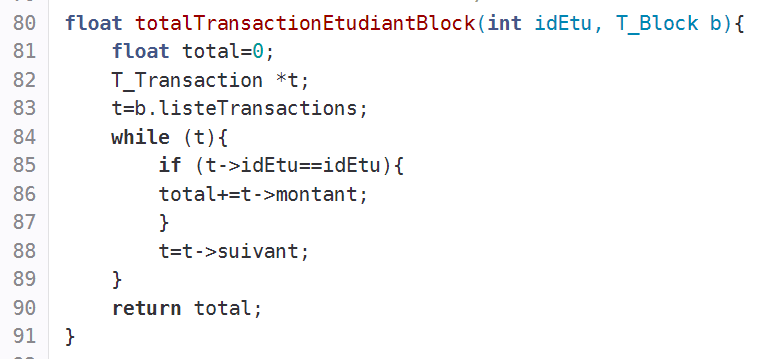
En conclusion, creerBloc est d’une complexité **d’O(1).** Il n’y a que de « simple » itération et pas de boucle ici.

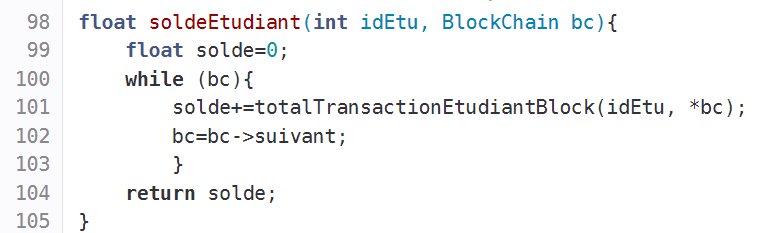
Dans la fonction **creerTransaction**, la première itération ligne 31 est d’une complexité d’O(1). Il en va de même pour les autres lignes de code. En effet, il n’y a pas de boucle ni de récursivité. La complexité de la fonction creerTransaction est de **O(1).**

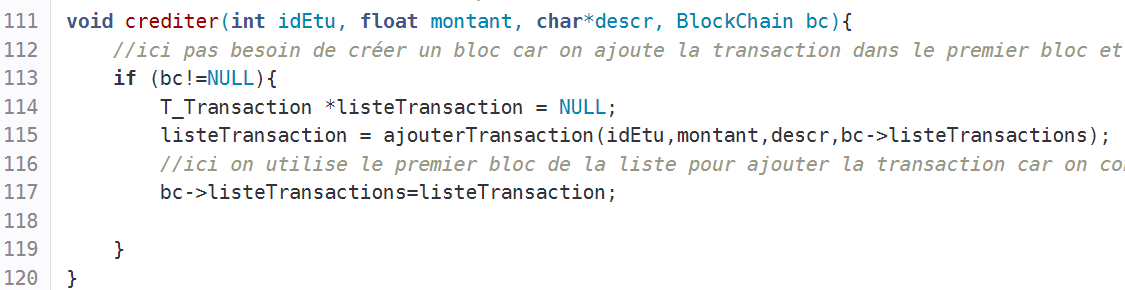


Dans la fonction **ajouterTransaction**, les lignes 51 à 57 ont une complexité d’O(1). En effet, il n’y a pas de boucle ni de récursivité. Et l’appel à la fonction creerTransaction est de O(1) car cette fonction est de O(1). Donc ajouterTransaction a une complexité **d’O(1).**

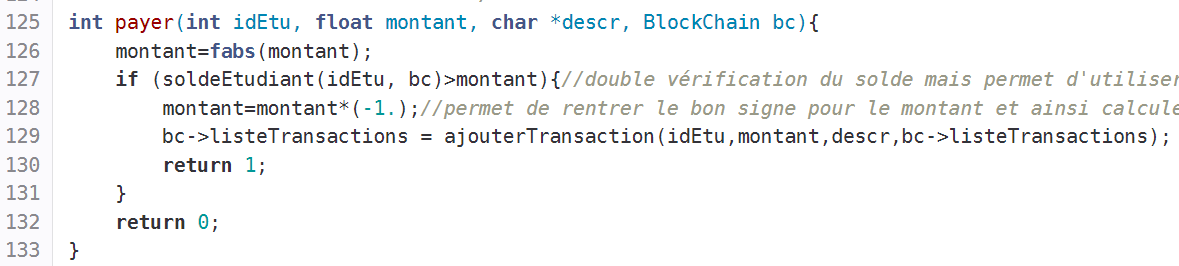
Dans la fonction **ajouterBlock**, il y a des if et des else et il est fait appel à la fonction creerBloc d’une complexité d’O(1). La fonction ajouterBlock est donc d’une complexité **d’O(1).**

Dans la fonction **totalTransactionEtudiantBlock**, il y a un parcours de la liste de transactions d’un block (de la journée en cours) à l’aide de la fonction while. Ainsi, si la taille de la liste est N, alors la complexité de la fonction totalTransactionEtudiantBlock est de **O(n).**

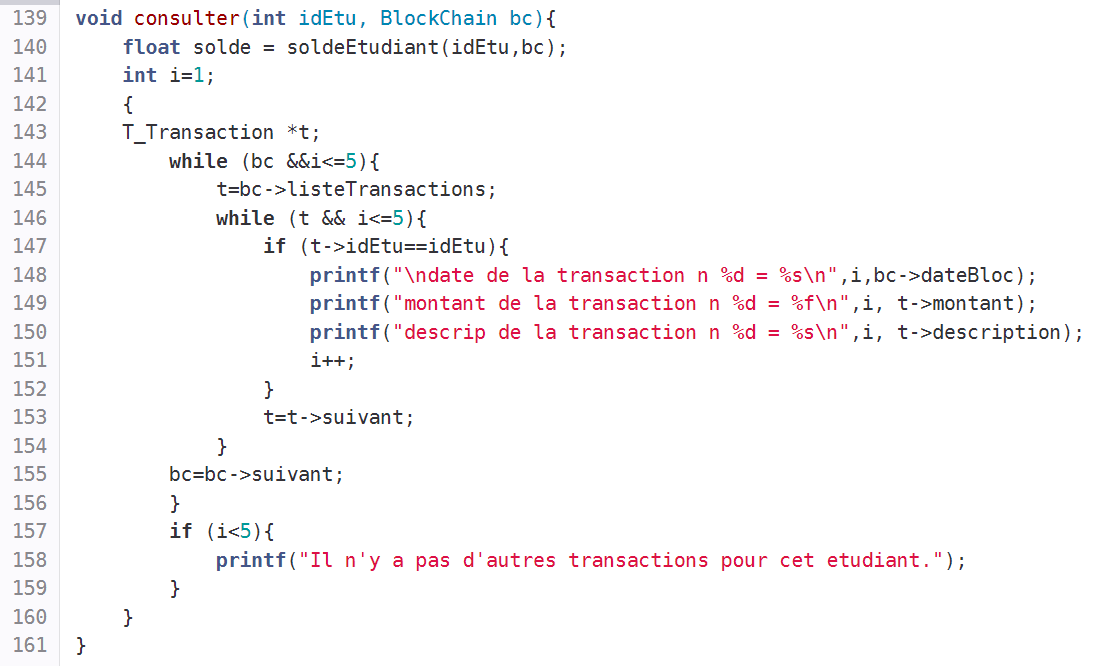
Dans la fonction **soldeEtudiant**, grâce au while, on parcourt la liste des blocs d’une taille M. De plus, à chaque bloc, on fait appel à la fonction totalTransactionEtudiantBlock d’une complexité d’O(n). Donc la complexité de soldeEtudiant est de **O(m\*n)** avec m le nombre de blocs et n le nombre de transactions maximal d’un bloc.

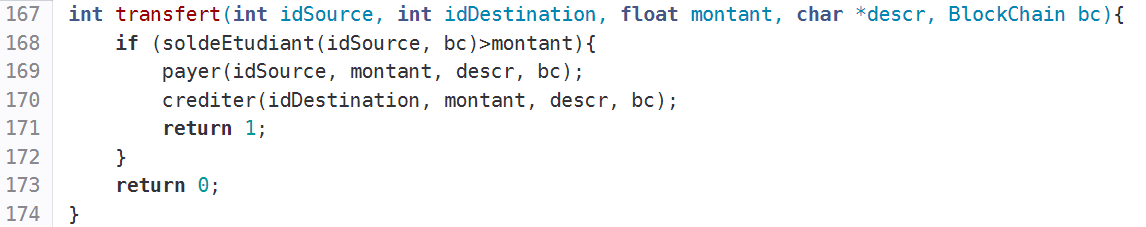


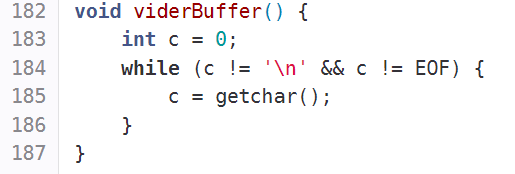
La fonction **crediter** possède un appel à la fonction ajouterTransaction d’une complexité d’O(1). La fonction crediter est **d’O(1)** car toutes ses itérations sont en O(1).



Dans la fonction **payer**, on fait appel à la fonction soldeEtudiant et à la fonction ajouterTransaction respectivement d’O(m\*n) et d’O(1). La fonction payer est donc d’une complexité **d’O(m\*n)** avec m le nombre de blocs et n le nombre max de transaction d’un bloc.

Dans la fonction **consulter**, on recherche 5 transactions maximums. Dans le pire des cas, on parcourt tous les blocs et toutes les transactions de chaque blocs pour en trouver 5 transactions ou moins. Donc la complexité est de **O(m\*n)** avec m le nombre de bloc et n le nombre de transactions maximal d’un bloc. C’est grâce aux boucles while que l’on parcourt les listes de blocs et de transactions.

Dans la fonction **transfert**, on fait appel aux fonctions payer (O(m\*n)) et crediter(O(1)). Donc la fonction transfert est d’une complexité **d’O(m\*n)** avec m le nombre de blocs et n le nombre de transaction max dans un bloc.

La fonction **viderBuffer** est d’une complexité **O(n)** avec n le nombre de caractères. En effet, il est fait appel à une boucle while pour parcourir une chaine de caractères stockée.

## Pour le programme principal :

Dans le programme principal, le switch (les choix) peut tourner pour autant de choix que l’utilisateur le demande. Donc le switch a une complexité **d’O(n)** avec n le nombre de choix.

Lorsque l’on rentre dans le détail des choix on a :

* Choix 1 : un parcours de la chaine de bloc donc une complexité **d’O(m)** avec m le nombre de blocs.
* Choix 2 : un parcours de la chaine de bloc et des transactions du bloc donc une complexité **d’O(m+n)** avec m le nombre de bloc et n le nombre de transaction.
* Choix 3 : un parcours des transactions d’un bloc, d’une complexité **d’O(n)** avec n transactions.
* Choix 4 : un appel au soldeEtudiant (O(m\*n)) et à consulter(m\*n) donc la complexité ici est de **O(m\*n)** avec m le nombre de blocs et n le nombre de transaction d’un bloc.
* Choix 5 : plusieurs appels à viderBuffer (O(c) avec c nombre de caractères), un appel à crediter (O(m\*n)). La complexité la plus importante(forte) étant O(m\*n) en comparaison de O(c), la complexité de la fonction est donc **O(m\*n)** avec m le nombre de blocs et n le nombre de transaction d‘un bloc.
* Choix 6 : il y a des appels à :
  + AjouterBlock (O(1))
  + soldeEtudiant (O(m\*n))
  + viderBuffer(O(c))
  + payer(O(m\*n))  
    Donc la complexité ici est de **O(m\*n)** car c’est le pire cas : avec m nombre de bloc et n nombre de transaction d’un bloc.
* Choix 7 : il y a des appels aux fonctions : viderBuffer, soldeEtudiant, transfert. Donc le pire cas étant pour la dernière transaction du dernier bloc (pour trouver le solde), on a une complexité de **O(m\*n)** avec m nombre de bloc et n nombre de transaction d’un bloc.
* Choix 8 : Ici, il va falloir ajouter un bloc pour chaque jour ajouter. Or, on peut changer la date comme on le souhaite. Il peut donc y avoir un nombre m de nouveaux blocs créés. La complexité est donc de **O(m)** avec m le nombre de blocs à ajouter, c’est-à-dire le nombre de jours entre la date précédente et la nouvelle date.
* Choix 9 : il y a un appel à la fonction libererEspace. La complexité est de **O(m\*n)** avec m le nombre de blocs et n le nombre de transaction d’un bloc.

Nos changements suite aux retours en TP : Le code étant fonctionnel sans bugs, nous n’avons pas eu de modifications techniques à faire. Mais nous avons tout de même modifié l’implémentation de la blockchain pour quelle commence sans être initialisée, modifiant par la même occasion les fonctions d’affichage. Cela permet de différencier les cas où il n’y a pas de liste de bloc de ceux où la blockchain n’a pas encore enregistrée de transaction.