Table des matières

[1. Introduction 2](#_Toc514189474)

[2. Mode d’emploi 3](#_Toc514189475)

[2.1 Navigation 3](#_Toc514189476)

[2.2 Écran de chargement 3](#_Toc514189477)

[2.3 L’interface générale 3](#_Toc514189478)

[2.4 Menu Fichier 4](#_Toc514189479)

[2.5 Menu Historique 4](#_Toc514189480)

[2.6 Menu Transaction 6](#_Toc514189481)

[2.7 Menu Utilisateur 6](#_Toc514189482)

[2.8 Menu Aide 7](#_Toc514189483)

[3. Présentation du projet 8](#_Toc514189484)

[3.1 Structure du projet 8](#_Toc514189485)

[3.2 Sources utilisées 8](#_Toc514189486)

[3.3 Librairies de fonctions 9](#_Toc514189487)

[3.4 Tableau des fonctions 10](#_Toc514189488)

[4. Algorithmes des fonctions 12](#_Toc514189489)

[4.1 Fonction affichPeriode (historique sur période) 12](#_Toc514189490)

[4.2 Fonction hacher (hachage cryptographique) 12](#_Toc514189491)

[5. Répartition des tâches 14](#_Toc514189492)

[5.1 Création de la librairie de chiffrement 14](#_Toc514189493)

[5.2 Création de la libraire de blockchain. 14](#_Toc514189494)

[5.3 Création du programme principal. 14](#_Toc514189495)

[5.4 Rapport 15](#_Toc514189496)

[5 Tests fonctionnels 16](#_Toc514189497)

[6 Améliorations à apporter 18](#_Toc514189498)

[7 Matières utilisées 19](#_Toc514189499)

[8 Conclusion 20](#_Toc514189500)

# Introduction

L’objectif principal du projet de cette année était la conception d’un programme de blockchain en langage C en utilisant les connaissances/compétences acquises dans 4 cours différents (MPP, LPP, Fichiers et Projet).

Les autres objectifs du programme étaient de :

* Chiffrer/Déchiffrer des chaînes de caractères
* Hacher des chaînes de caractères
* Créer un système de blockchain
* Parcourir une blockchain pour récupérer des informations
* Manipuler le format json (librairie fournie par le professeur)
* Enregistrer des transactions
* Afficher les différents historiques des transactions
* Manipuler une interface graphique et un menu
* Utiliser un système de rupture
* Découper un projet

Ce programme va donc enregistrer des transactions dans un système de blocs chainés nommé blockchain. Une blockchain peut être vue comme un système de stockage (ici en RAM) de données (ici de transactions), cette technique est notamment utilisée dans la crypto-monnaie (Ex : Bitcoin). L’historique d’une blockchain peut être visionné à n’importe quel moment par le public.

Un bloc est composé de plusieurs informations, dans notre cas il est divisé en deux parties :

* Le hashcode du bloc (empreinte unique permettant d’identifier un bloc)
* Le contenu du bloc
  + 3 transactions
  + Le hashcode du bloc précédent
  + Le numéro du bloc

# Mode d’emploi

Nous avons travaillé un maximum pour rendre le programme « user-friendly » (intuitif) au niveau de son interface comme de son utilisation.

Pour permettre à « n’importe qui » d’utiliser notre programme, nous y avons ajouté des messages de retour pour chaque cas que l’utilisateur pourrait rencontrer. Une section aide, dans laquelle on peut retrouver un « mode d’emploi minimal » du programme est disponible.

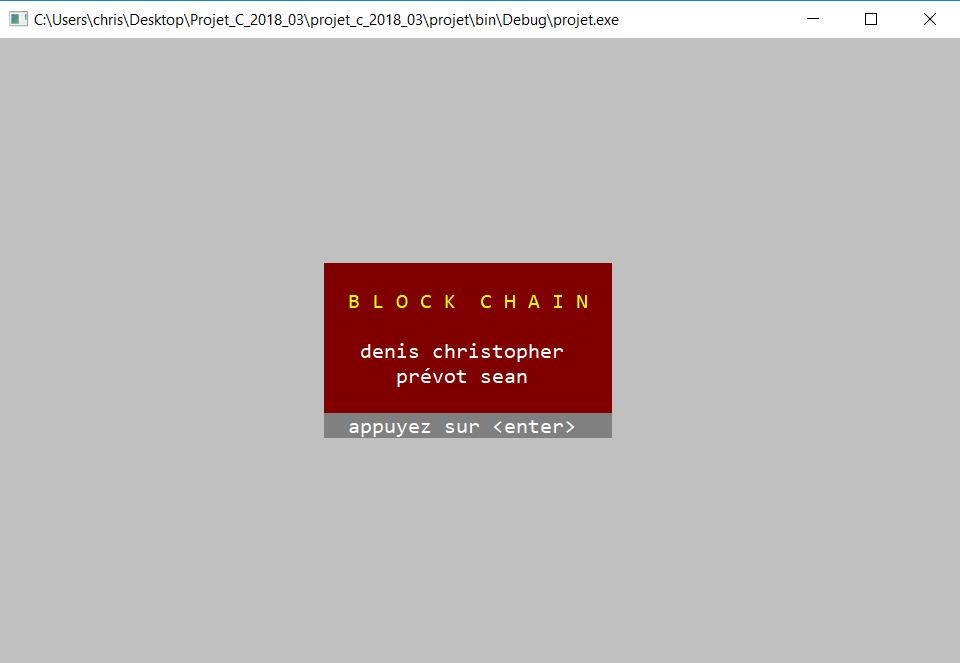
## Navigation

Pour naviguer dans les menus du programme, il faut utiliser uniquement le clavier. Voici l’ensemble des touches :

* Enter : Permet de valider.
* Escape : Permet de quitter un sous-menu ou le programme.
* Flèches directionnelles : Permet de naviguer dans le menu.
* F1 à F5 : Permet d’accéder directement à un menu.

## Écran de chargement

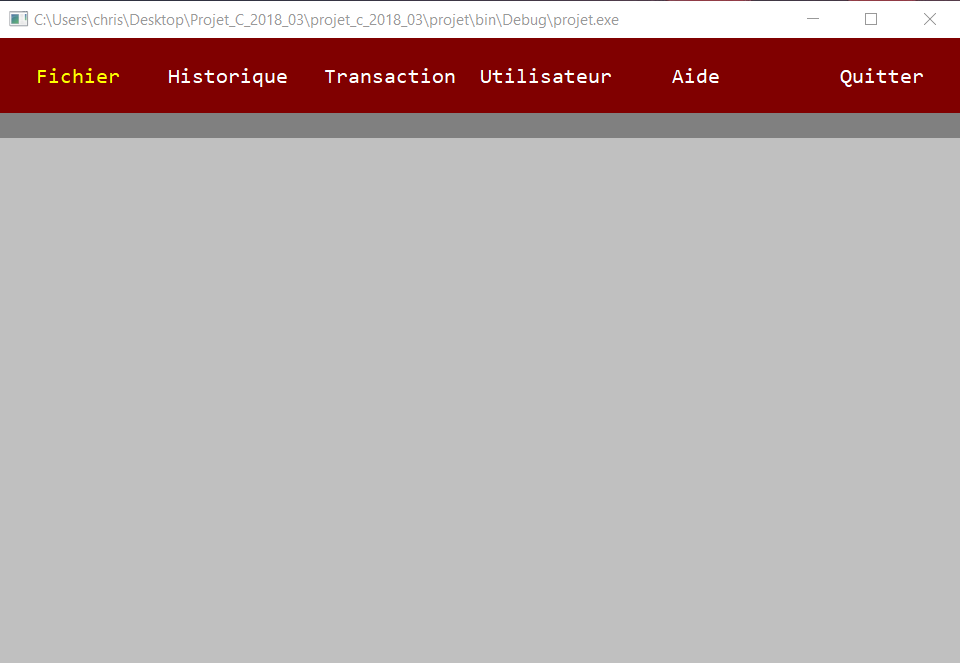
Nous y retrouvons le nom du programme ainsi que celui des développeurs.



## L’interface générale

Nous y retrouvons différents menus tels que :

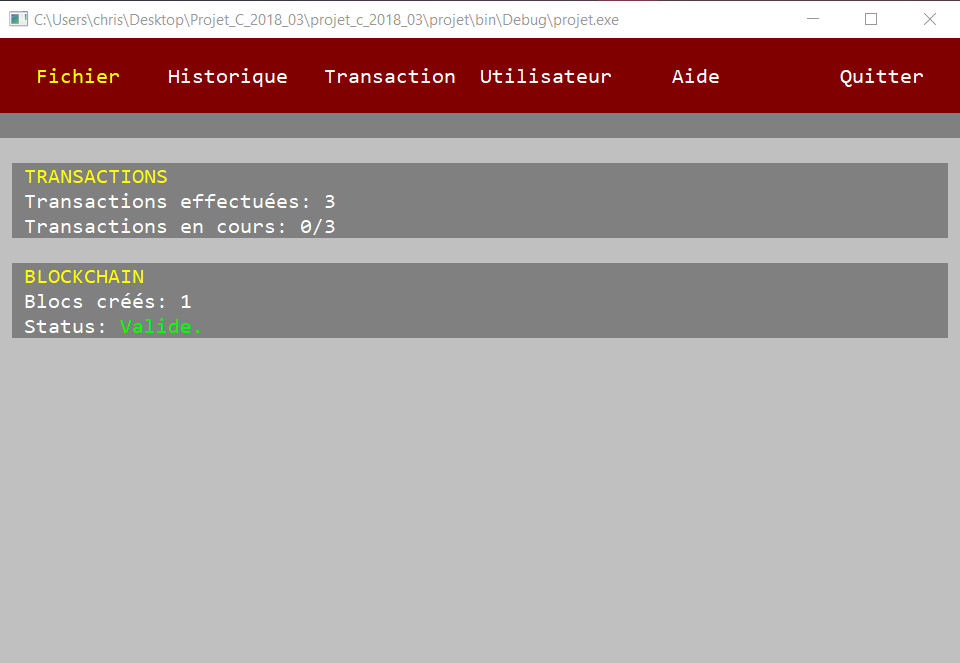
* Fichier
* Historique
* Transaction
* Utilisateur
* Aide



## Menu Fichier

Il n’est composé que d’un seul sous-menu, permettant de :

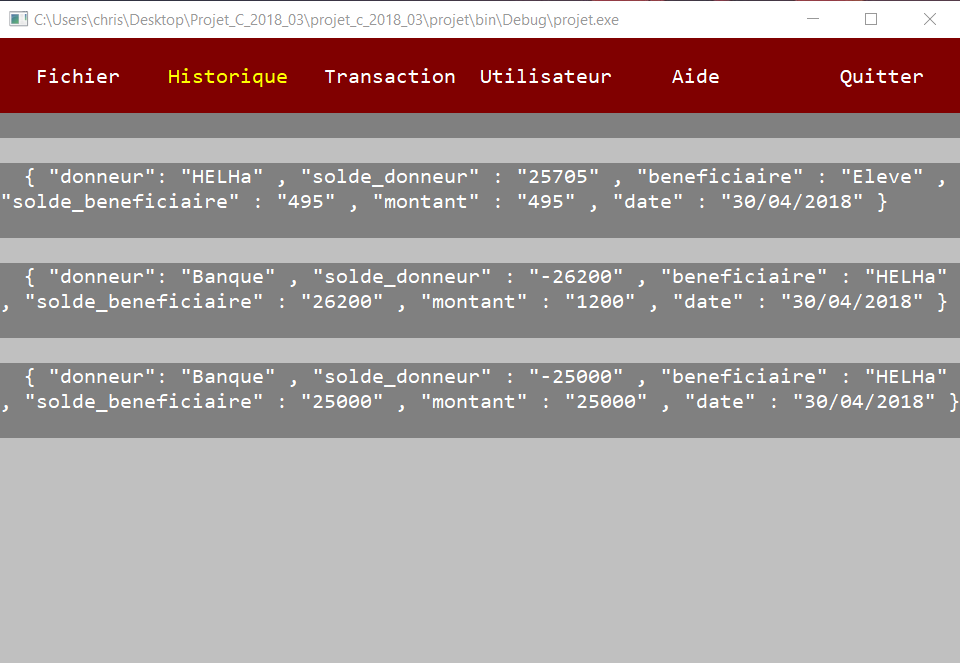
* Vérifier
  + Afficher le nombre total de transactions effectuées.
  + Afficher le nombre de transactions en cours (hors blockchain).
  + Vérifier la validité de la blockchain.



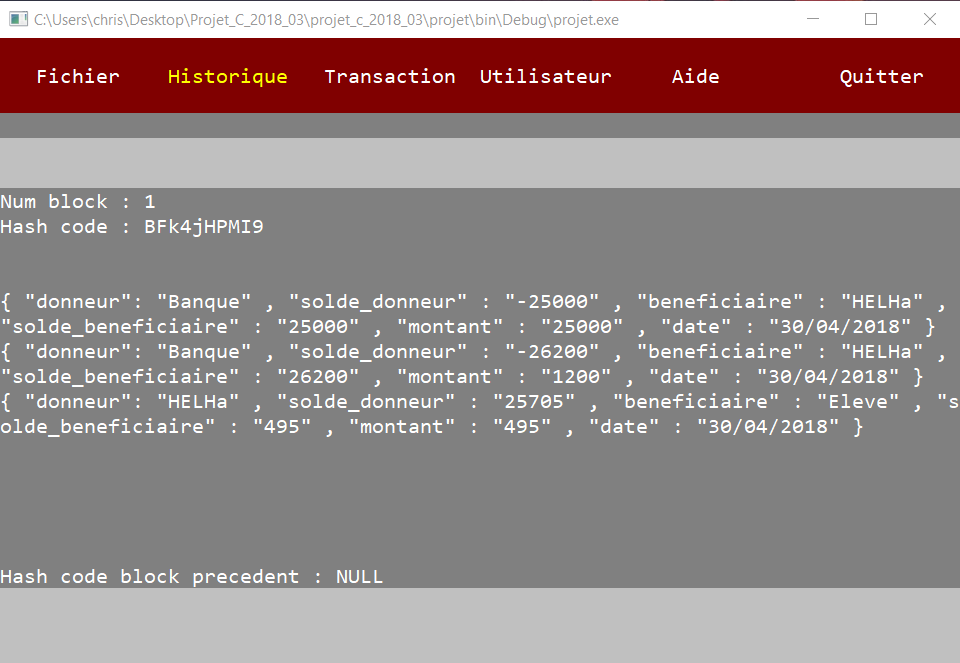
## Menu Historique

Il est composé de 3 sous-menus, permettant de :

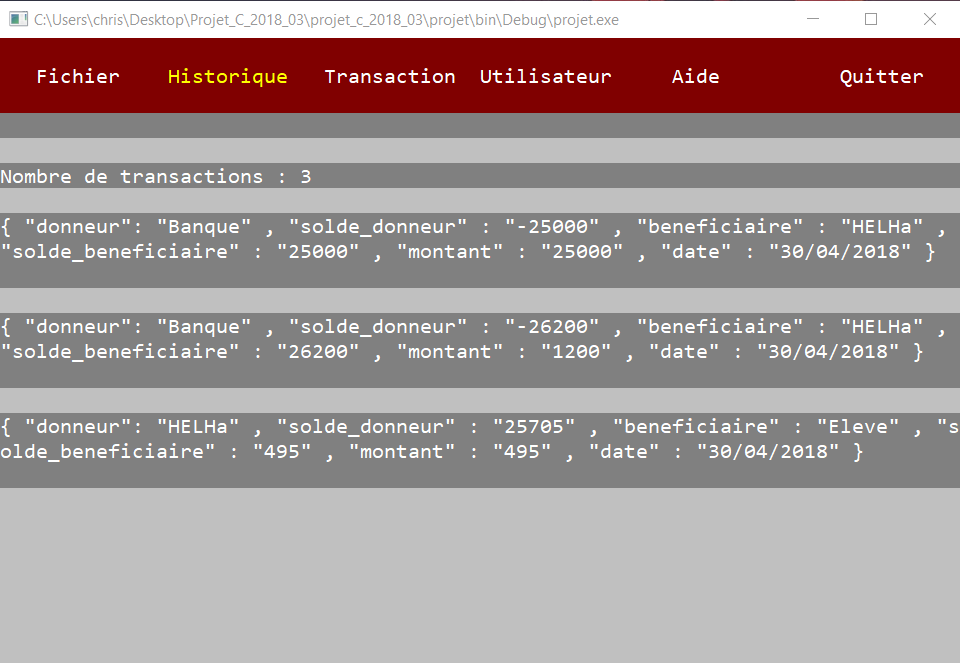
* Récent
  + Afficher les 5 dernières transactions effectuées.



* Complet
  + Afficher toutes les transactions enregistrées dans la blockchain. Les transactions qui sont en « attentes » (hors blockchain) ne sont pas affichées dans cet historique.

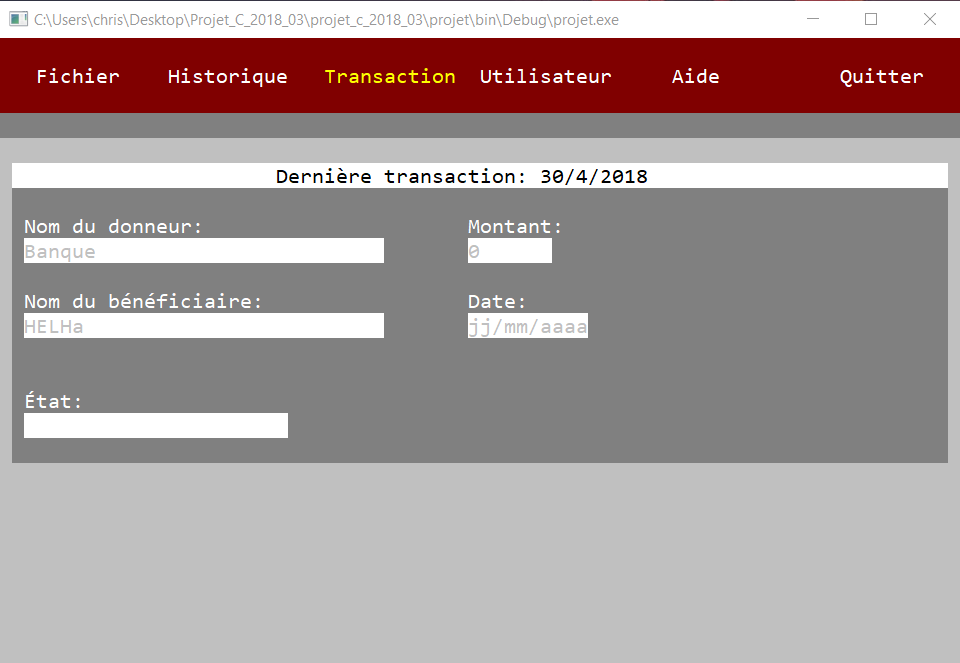


* Période
  + Afficher toutes les transactions effectuées sur une période choisie, ainsi que pour chaque date de cette période. Pour cela, il faut saisir deux dates valides contenant des transactions (entre ces deux dates).



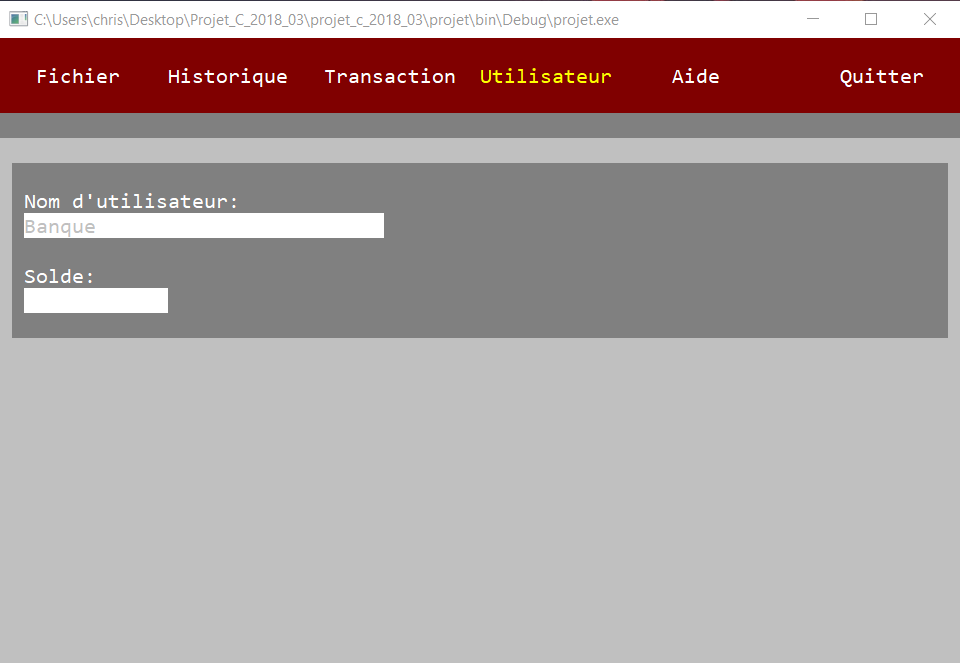
## Menu Transaction

* Ajouter
  + Ajouter une transaction dans le tableau de transactions. Ce menu gère aussi la création et l’ajout de nouveaux blocs à la blockchain. Pour enregistrer une transaction, il faut réunir plusieurs paramètres : un donneur dont le solde est supérieur à zéro (sauf Banque), un bénéficiaire, un montant supérieur à zéro ou inférieur/égal au solde du donneur, une date valide et chronologique.



## Menu Utilisateur

* Solde
  + Afficher le solde d’un utilisateur. Pour cela, il faut saisir le nom d’un utilisateur existant.



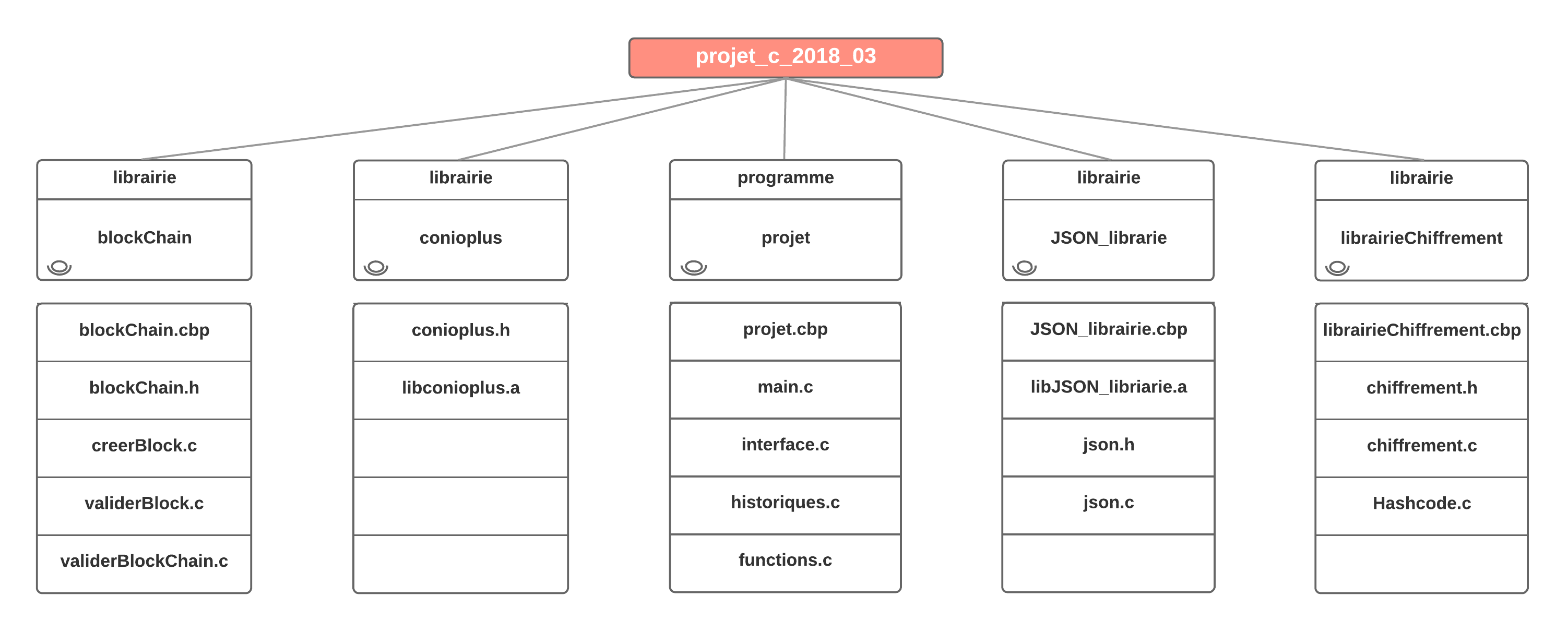
## Menu Aide

* À propos
  + Afficher des informations d’aide ou des formats à respecter.



# Présentation du projet

## Structure du projet



## Sources utilisées

Nous avons découpé notre programme en 5 fichiers :

* Programme principal
  + Projet.h

Dans ce fichier, se retrouve tous les « include » des librairies utilisées, les « define » des constantes et les prototypes des fonctions du programmes.

* + Main.c

Ce fichier est le « cœur » du programme, c’est lui qui appelle à la plupart des fonctions. C’est dans ce fichier que l’affichage du contenu des menus se fait, que la création de la blockchain se fait, ...

* + Interface.c

Ce fichier sert à gérer la partie essentielle de l’interface, c’est-à-dire : la navigation dans les menus et l’interface en général.

* + Functions.c

Dans ce fichier, nous avons regroupé toutes nos fonctions, cela permettant d’alléger le code du fichier main.c (création de transactions, vérification de dates, saisie de chaînes de caractères, …).

* + Historiques.c

Ce fichier contient toutes les fonctions « d’historique » (récent, complet et période).

* Blockchain
  + blockChain.h

Dans ce fichier, se retrouve tous les « include » des librairies utilisées, les « define » des constantes, les structures (BLOCK et CONTENU\_BLOC) et les prototypes des fonctions de la librairie.

* + creerBlock.c

Ce fichier contient la fonction nécessaire à la création de blocs pour la blockchain.

* + validerBlock.c

Ce fichier contient la fonction permet de signaler si un bloc est valide ou non.

* + validerBlockChain.c

Ce fichier contient la fonction permettant de signaler si la blockchain est valide ou non.

* librairieChiffrement
  + chiffrement.h

Dans ce fichier, se retrouve tous les « include » des librairies utilisées et les prototypes des fonctions de la libraire.

* + chiffrement.c

Ce fichier contient les fonctions permettant de chiffrer et de déchiffrer une chaîne de caractères.

* + Hashcode.c

Ce fichier contient la fonction permettant de créer un hashcode à partir d’une chaîne de caractères.

* Conioplus

C’est une librairie qui nous a été fournie, elle nous a permis la création de l’interface, le placement des textes, ...

* JSON\_librairie

C’est une librairie qui nous a été fournie, elle nous a permis la récupération d’informations dans une chaîne de caractères au format json.

## Librairies de fonctions

Dans notre projet, nous avons été amener à utiliser des librairies ayant déjà été programmées par un développeur (conioplus, JSON\_librairie, …) ainsi que nos propres librairies.

Nous avons programmé :

* Blockchain

Cette librairie n’intervient que dans le programme principal. Dans le sous-menu « vérifier » une des fonctions de cette librairie intervient pour indiquer le statut de la blockchain (valide ou non). Dans le sous-menu « ajouter », si 3 transactions ont bien été enregistrées, une autre fonction intervient pour la création de blocs.

* librairieChiffrement

Cette librairie est utilisée dans l’ensemble du projet. Dans la librairie « blockchain » lors de la création d’un bloc, de la validation d’un bloc ou de la validation de la blockchain, les fonctions de chiffrement et de déchiffrement interviennent. Dans le programme principal lors d’un appel au contenu de la blockchain (ex : récupération du solde d’un utilisateur), la fonction de déchiffrement permet de déchiffrer le contenu du bloc pour accéder aux transactions contenues dans celui-ci.

## Tableau des fonctions

|  |  |  |  |
| --- | --- | --- | --- |
| FONCTION | PARAMÈTRES | UTILITÉ | FICHIER SOURCE |
| char \*chiffrer | char \*pChaine, int taille | Chiffre des chaines des caractères. | chiffrement.c |
| char \*dechiffrer | char \*pChaineChiffree, int taille | Déchiffre des chaînes de caractères chiffrées. | chiffrement.c |
| char \*hacher | char pBytes, int taille | Crée un hashcode à partir d’une chaîne de caractères. | Hashcode.c |
| BLOCK \*creerBlock | char \*transactions[], int nbTransactions, char \*pHashCodePrec, int numBloc | Crée un bloc pouvant être ensuite ajouter à la blockchain. | creerBlock.c |
| int validerBlock | BLOCK \*pBlock | Vérifie la validité d’un bloc. | validerBlock.c |
| int validerBlockChain | BLOCK blocks[], int nbBlocks | Vérifie la validité de la blockchain. | validerBlockChain.c |
| int interface | Int emplacement | Permet l’affichage du menu et de l’interface en général. | Interface.c |
| Int affichmenu | Struct item \*menu, struct item\*pcour, int direction, int bg | Permet l’affichage des menus et sous-menus, ainsi que la navigation. | Interface.c |
| Unsigned int getMyKeys |  | Permet de renvoyer un code de retour en fonction de la touche pressée. | Interface.c |
| Char \*creerTransaction | Char \*donneur, char \*beneficiaire, int montant, char \*date | Crée une transaction en vérifiant si toutes les informations entrées sont valides. | Functions.c |
| Int checkDate | Int jj, int mm, int aa | Vérifie si une date existe. | Functions.c |
| Void saisirChaine | Char \*chaine, int taille\_chaine, int x, int y, int whichChar, int font\_color, int text\_color | Permet de saisir des chaînes de caractères en évitant le buffer overflow. | Functions.c |
| Char \*formatjson | Char \*donneur, char \*beneficiaire, int montant, int solde\_donneur, int solde\_beneficiaire, char \*date | Crée une chaîne de caractères au format json, à partir des données d’une transaction. | Functions.c |
| Void rechercheSoldeUtilisateur | Int \*found\_utilisateur, int \*solde\_utilisateur, char \*utilisateur | Permet de rechercher dans le tableau de transactions et dans la blockchain, le solde (le plus récent) d’un utilisateur. | Functions.c |
| Void affichRecent |  | Permet d’afficher les 5 dernières transactions effectuées. | Historiques.c |
| Void affichComplet |  | Permet d’afficher de manières chronologique toutes les informations des blocs dans la blockchain, bloc par bloc. | Historiques.c |
| Void affichPeriode |  | Permet d’afficher toutes les transactions effectuées sur une période déterminée, date par date. | Historiques.c |

# Algorithmes des fonctions

## 4.1 Fonction affichPeriode (historique sur période)

Présentation

Cette fonction permet d’afficher toutes les transactions effectuées sur une période déterminée. Elle affiche le nombre de transactions pour chaque date ainsi que les transactions enregistrées à ces dates. À la fin, celle-ci affiche le nombre total de transactions effectuées sur cette période.

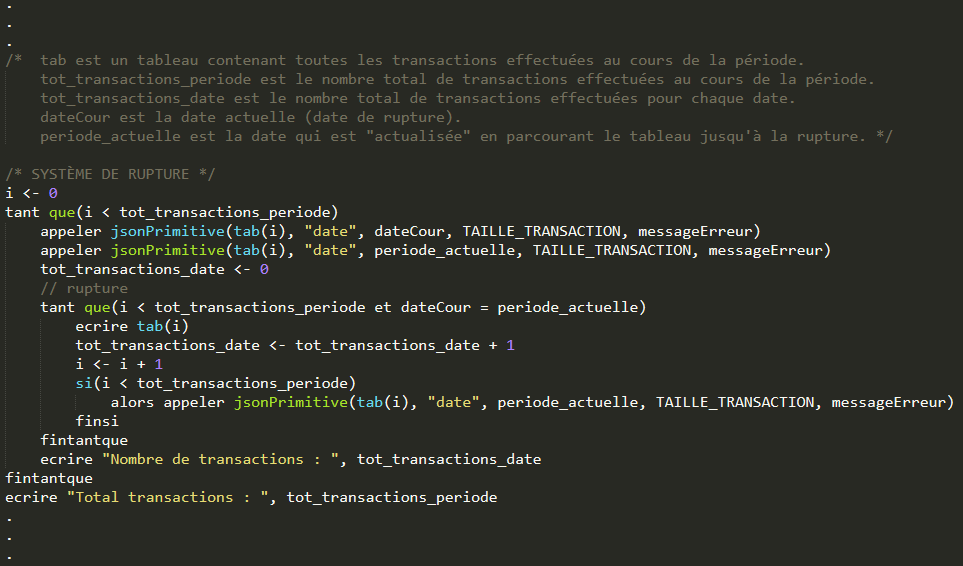
Explications

Cette fonction est basée sur un système à une rupture, c’est-à-dire (dans notre cas) qu’à chaque changement de date, nous afficherons le nombre de transactions effectuées à la date courante.

Tout d’abord, nous parcourons la blockchain (du début vers la fin) et le tableau de transactions en cours pour récupérer les transactions comprises entre les deux dates dans un autre tableau.

Ensuite, nous appliquons à ce nouveau tableau une rupture par date. Tant qu’il y a des transactions dans le tableau, nous récupérons la date de rupture et la date courante, nous initialisation le nombre de transaction de la date de rupture à zéro. Tant qu’il y a des transactions et que la date de rupture et la date courante sont les mêmes, nous affichons chaque transaction effectuée lors de la date de rupture. Dès que la date courante est différente de la date de rupture, nous effectuons à nouveau ce processus s’il reste des transactions.

Finalement, nous affichons le nombre total de transactions effectuées sur cette période.



## 4.2 Fonction hacher (hachage cryptographique)

Présentation

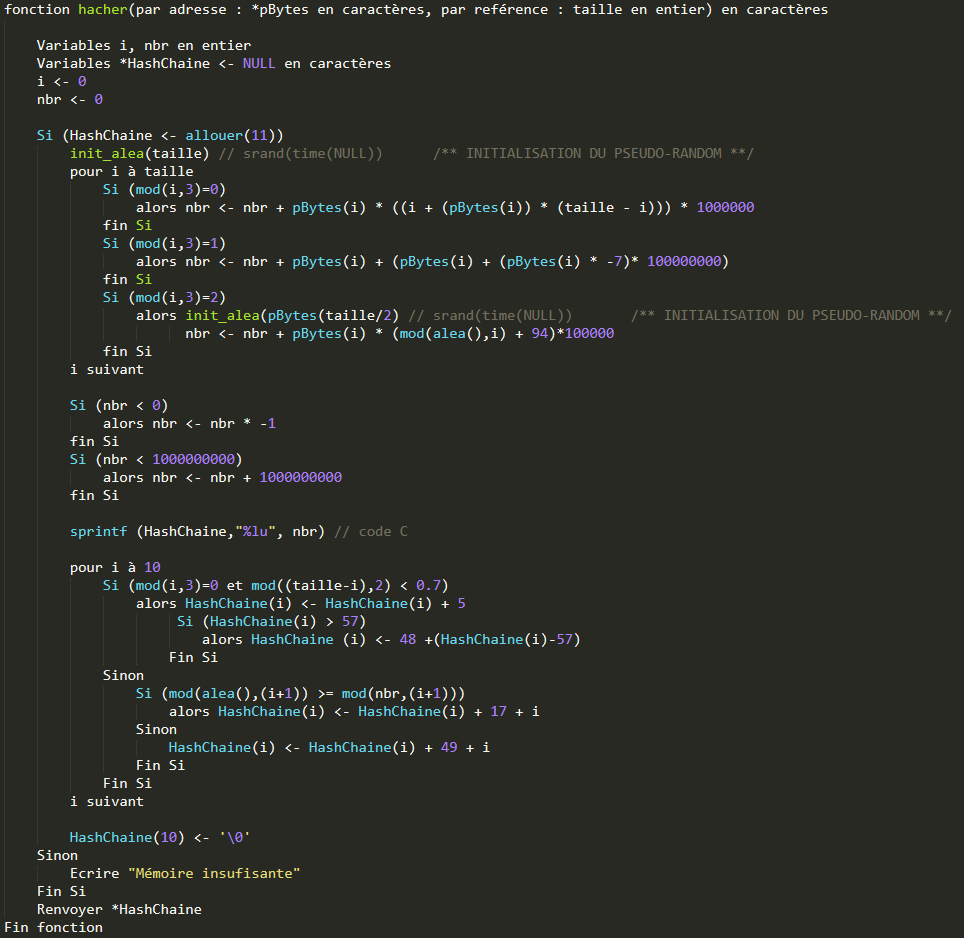
Cette fonction permet de créer un hashcode de 10 caractères (lettres et chiffres) à partir de n’importe quelle chaîne de caractères et de n’importe quelle taille.

Explications

La fonction de hachage se base sur le fait de ne pas avoir de vrai aléatoire dans l'informatique, tout cet aléatoire est alors initialisée grâce à la longueur de la chaine passée en argument, ensuite, chacun des caractères va suivre 3 cas possibles qui sont, pour 2 des cas, juste une équation arithmétique, mais, pour la 3éme, nous avons une réinitialisation de l'aléatoire à la position de la lettre divisée par 2.

La suite est assez simple, on regarde si le nombre est positif ou négatif (on le multiplie par -1 s'il est négatif) et on transforme le nombre en chaine de caractères pour pouvoir "jouer" avec. On vérifie si le nombre n’est pas inférieur à 1 000 000 000, s’il est inférieur on augmente ce nombre d’1 000 000 000.

Chaque caractère de la chaine sera alors modifié selon 4 possibilités chacune résultant en une lettre ou un chiffre.



# Répartition des tâches

Le projet a été scindé en trois itérations/étapes :

## 5.1 Création de la librairie de chiffrement

|  |  |  |
| --- | --- | --- |
| Tâches | Christopher | Sean |
| Fonction chiffrer | X |  |
| Fonction déchiffrer | X |  |
| Fonction hacher |  | X |
| Tests de fonctionnement |  | X |

## 5.2 Création de la libraire de blockchain.

|  |  |  |
| --- | --- | --- |
| Tâches | Christopher | Sean |
| Fonction creerBlock | X |  |
| Fonction validerBlock | X | X |
| Fonction validerBlockChain |  | X |
| Tests de fonctionnement | X |  |

## 5.3 Création du programme principal.

|  |  |  |
| --- | --- | --- |
| Tâches | Christopher | Sean |
| Fonction interface (adaptée à partir du code Mr Gouwy) | X |  |
| Fonction creerTransaction | X | X |
| Fonction checkDate (reprise de MPP Q1) | X |  |
| Fonction saisirChaine (provient d’un code personnel) | X |  |
| Fonction rechercheSoldeUtilisateur | X | X |
| Système de création de la blockchain | X | X |
| Fonction formatJson |  | X |
| Fonction affichRecent |  | X |
| Fonction affichComplet |  | X |
| Fonction affichPeriode | X | X |
| Tests de fonctionnement | X | X |

## 5.4 Rapport

|  |  |  |
| --- | --- | --- |
| Tâches | Christopher | Sean |
| Rédaction | X | X |

# Tests fonctionnels

Qu’est-ce qu’une transaction valide ?

* Donneur : existant (avec un solde suffisant) ou « Banque ».
* Bénéficiaire : existant ou non.
* Montant : supérieur à zéro (inférieur ou égal au solde du donneur).
* Date : au format jj/mm/aaaa (existante).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | DESCRIPTION TEST | PROCÉDURE | RÉSULTAT ATTENDU | RÉSULTAT OBTENU |
| 1 | Obtenir Solde  Client inexistant | Allez dans utilisateur et introduire un nom d’utilisateur inexistant. | "introuvable" |  |
| 2 | Obtenir Solde  Client existant  Après transaction | Créez une transaction valide, introduire un nom d’utilisateur existant dans utilisateur | "valeur du solde" |  |
| 3 | Obtenir Solde  Client inexistant  Après transaction | Créez une transaction valide, introduire un nom d’utilisateur inexistant dans utilisateur | "introuvable" |  |
| 4 | Vérification | Allez dans vérifier après avoir créé une transaction valide | "transaction effectuée : 1  Transaction en cours : 1/3  Blocs créés : 1  Statut : Veuillez créer au moins un bloc (3 transactions)" |  |
| 5 | Vérification après avoir créé un bloc | Allez dans vérifier après avoir créé un bloc de 3 transactions valides | "transaction effectuée : 3  Transaction en cours : 0/3  Blocs créés : 1  Statut : valide" |  |
| 6 | Test historique complet et récent quand aucune transaction existe | Allez dans historique et choisissez soit complet soit récent (même résultat attendu), sans créer de transaction | "Aucune transaction trouvée" |  |
| 7 | Test historique période quand aucune transaction n’existe dans la période | Allez dans historique et choisissez période et entrez une période où il n'y a aucune transaction (même résultat attendu) | "Aucune transaction trouvée" |  |
| 8 | Test historique période quand il y a des transactions dans la période | Créez au moins une transaction. Ensuite, allez dans historique période et sélectionnez une période contenant une ou plusieurs transactions. | « Nombre de transactions : …  { donneur : … }  ….  Total transactions : … » |  |
| 9 | Test transaction donneur inexistant | Allez dans transaction et entrez un donneur inexistant (sauf Banque) | "donneur introuvable" |  |
| 10 | Test transaction date invalide | Allez dans transaction et entrez une date inexistante, mais le reste des informations de la transaction doivent être valide | "ERREUR : Date invalide" |  |
| 11 | Test transaction même donneur et bénéficiaire | Allez dans transaction et entrez le même nom pour donneur et bénéficiaire | "ERREUR : Utilisateurs identiques" |  |
| 12 | Test transaction Montant négatif | Allez dans transaction et entrez un montant négatif | "ERREUR : Montant invalide" |  |
| 13 | Test transaction donneur inexistant & date invalide  & donneur = bénéficiaire  & Montant négatif | Allez dans les transactions et entrez un donneur inexistant, donnant à lui-même, à une date invalide un montant négatif | "ERREUR : Montant invalide" |  |

# Améliorations à apporter

Nous avons déjà apporté des améliorations à notre projet, telles que :

* Enregistrement des dates de manière chronologique.
* L’affichage de la date de la dernière transaction, lors de la création d’une transaction.
* L’affichage du nombre de transactions en cours et total dans le sous-menu « vérifier ».

Ce que nous aurions pu améliorer :

* L’intégration d’un fichier chiffré contenant le nom des utilisateurs ainsi que leur solde.
* L’intégration d’un fichier contenant la blockchain et ne pas la « stockée » en ram car celle-ci étant volatile, à l’extinction du programme ou de l’ordinateur, les données sont perdues (vidées).
* La possibilité de pouvoir ajouter une communication libre ou structurée lors de la création d’une transaction.
* L’ajout d’un sous-menu « ajouter » au menu « utilisateur », pour créer un utilisateur en lui attribuant un solde de départ.
* La recherche et l’affichage de la date de la dernière transaction effectuée par un utilisateur.

# Matières utilisées

* MPP

Nous avons appliqué le système de ruptures dans notre fonction permettant d’afficher l’historique sur une période déterminée, date par date. Nous avons aussi vu les allocations dynamiques, les fonctions et les pointeurs.

* LPP

Nous avons utilisé toutes nos connaissances pour saisir des chaînes sécurisées contre le « buffer overflow » (getche en vidant le codeur de touches), pour créer le menu et l’interface graphique. Nous avons aussi utilisé des fonctions, des pointeurs et les allocations dynamiques.

* Fichiers

Nous avons manipulé les structures pour la création et gestion de blocs.

* Projet

Nous avons découpé notre projet en plusieurs fichiers (.h et .c) et nous avons inclus/créé des librairies.

# Conclusion

Le projet a été un excellent moyen de juger notre niveau dans de plus belles proportions, nous avons pu faire face à nos faiblesses en C et en algorithmique pour nous dépasser et les surmonter. Travailler en équipe était aussi une belle expérience et l'utilisation de GIT a été compliquée aux premiers abords, mais au final GIT était assez simple et très pratique d’utilisation.

Ce projet, bien que compliqué, était intéressant et parfait pour prendre l'habitude de découper nous-même de gros projets ainsi que de se mettre à choisir une logique pour arriver à nos fins. Nous avons aussi « appris » à rendre nos projets dans les délais imposés.

Un grand merci à nos professeurs qui nous ont aidés et accompagnés durant ce projet ainsi qu'à Mr Gouwy qui, grâce à sa sévérité, nous a forcé à travailler de façon journalière.