esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

TD n°6 INF1031

Une histoire de tableaux (et de lapins)

Arnaud Bannier
Nicolas Bodin
Aurélien Texier

Cette fois ci, pas de cours ! Mais cela ne veut pas dire pas de travail ... L'objectif de ce TD est de découvrir une nouvelle notion de programmation en C au travers de la résolution d'un problème. Vous allez devoir aider Douglas à se soigner en découvrant la notion de tableaux. Réalisez les exercices les uns après les autres, sans en négliger un seul. Au moindre souci ou incompréhension, sollicitez votre chargé de TD ou de TP.

1. Introduction : Douglas se sent mal

Douglas est malade ... Il se sent ballonné depuis plusieurs jours et décide, suite aux conseils de l'ibijau, de surveiller son transit de plus près afin de présenter les résultats à Linux, son vétérinaire. L'objectif est de vérifier si sa moyenne journalière de nombre de crottes est normale (entre 300 et 400) et si la répartition de ses passages à la litière dans la journée est normale (plus fréquents en soirée et la nuit). Douglas doit donc consigner, heure par heure, le nombre de crottes qu'il a émis et doit saisir ces données dans un programme qui analysera son transit. Sachant que ses soucis de santé sont récurrents, il veut également que ce programme puisse être réutilisé aussi souvent que nécessaire, sur des périodes de temps arbitraires.

En l'état actuel de ses connaissances informatiques, Douglas est incapable de réaliser un tel programme. Mais l'ibijau, qui a suivi les cours de l'ESIEA il y a quelques années, lui explique qu'un tel programme est réalisable à l'aide d'un *tableau*. Il lui explique alors la chose suivante.

« Il est possible de simuler l'utilisation d'une quantité arbitraire de variables au travers d'un *tableau*. Il s'agit d'un **ensemble de valeurs**, stockées de façon **contigües en mémoire**, et **toutes du même type**. Chacune de ces valeurs se trouve dans une *case* repérée à l'aide d'un *indice*. De façon bizarre, mais explicable, la première case est numérotée 0.

Par exemple, grâce à un tableau, tu pourras stocker 15 entiers, et chacun de ces entiers correspondra au nombre de crottes que tu as émises en une heure. Tu aura ainsi surveillé 15 heures de ton transit, qui sera très simple à analyser à l'aide d'un teste statistique. »

Douglas lui répond alors : « C'est parfait mon ibijau, mais je ne vois pas bien en quoi ce tableau change la donne par rapport à 15 variables que j'aurais déclarées et affectées à l'aide de 15 appels à la fonction `scanf()` ? »

« Et bien si tu veux réutiliser ton programme une autre fois, répondit l'ibijau, et que tu es malade plus longtemps, tu ne pourras toujours saisir que 15 valeurs, soit 15 heures, alors que tu aurais dû surveiller ton transit sur une autre période de temps. Et puis grâce

au tableau, tu auras le gros avantage de pouvoir condenser ton code en utilisant une simple instruction de boucle pour accéder à tes données à l'aide d'un simple nom de variable ! Aies confiance, ce n'est pas si compliqué. »

Encore plein de doutes, Douglas se résigna donc à écouter l'ibijau et à réaliser les exercices qu'il lui demandait de pratiquer afin d'oublier sa douleur lancinante.

« Comme je t'ai dit, l'ensemble des valeurs d'un tableau est accessible à l'aide du nom du tableau et de l'indice de la case à laquelle tu veux accéder. Mais pour cela, il te faut déclarer un tableau. Tu peux par exemple utiliser l'instruction suivante. »

```
1    int tab[10];  // Declaration d'un tableau de 10 entiers
```

« Cette instruction a donc réservé **10** emplacements dans la mémoire, indicés de 0 à 9, chaque emplacement permettant de stocker un entier à l'aide du mot clé **int**. Tu pourras par la suite accéder à tes éléments à l'aide du nom du tableau, qui est ici **tab**. Note bien que le nombre d'éléments de ton tableau (10 ici) est fixe. **Cette taille doit être une constante** ! Cela peut sembler bloquant pour la modularité du code, mais j'ai une idée. Tu peux surdimensionner ton tableau lors de la déclaration. Par exemple, tu sais que tu ne peux pas suivre ton transit sur plus d'une semaine (sinon tu meurs), donc la taille de ton tableau serait de 7 jours × 24 heures, ce qui donne 168 cases. »

« Il est alors possible de remplir les premières cases de ton tableau en les affectant lors de la déclaration. Écris les valeurs entre des accolades séparées par des virgules de la façon suivante. La première case (d'indice 0) aura la valeur 3, la seconde la valeur -2 etc. Tu peux attribuer une valeurs aux premières cases, et si il manque des éléments, les cases suivantes seront à 0. »

```
1    short T[7] = {3, -2, 1};  // Declaration d'un tableau de 7 short
```

« Pour bien comprendre, voici à quoi ressemble le tableau `T` que je viens de définir. »

indice	0	1	2	3	4	5	6
valeur	3	-2	1	0	0	0	0

2. Application des tableaux : Douglas combat sa maladie

Exercice 1. Douglas déclare des tableaux

Maintenant que vous savez déclarer un tableau, répondez aux questions suivantes.

1.1) Rédigez l'instruction permettant de déclarer un tableau nommé `tabf`, et contenant 5 valeurs de type `float`.

1.2) En supposant que la constante `N` vaille 5, rédigez la déclaration d'un tableau nommé `tab`, contenant `N` caractères (du type que vous préférez).

1.3) Rédigez la déclaration d'un tableau `tabEntiers` de 100 cases dont les 5 premières contiennent les 5 premiers entiers naturels.

1.4) Rédigez la déclaration d'un tableau contenant les 5 premières lettres de l'alphabet.

Considérons maintenant l'instruction suivante.

```
1    float tabFloat[5] = {3.14f};
```

1.5) Quel est le nom et le type de données du tableau déclaré précédemment ?
1.6) Combien ce tableau comporte-t-il de cases ?
1.7) Combien d'octets ont-été réservés lors de la déclaration de ce tableau ?
1.8) Quelle est la valeur de la case d'indice 0 ? D'indice 1 ? D'indice 5 ?

Ragaillardi par cette introduction, Douglas demande plus de précisions à l'ibijau sur le fonctionnement d'un tableau. Il lui demande alors comment il peut accéder à l'élément situé dans une case particulière du tableau. L'ibijau répond alors : « Comme je te l'ai dit tout à l'heure, tu as simplement besoin du nom de ton tableau et de l'indice de la case à laquelle tu veux accéder. Tu pourras alors lire la valeur présente dans cette case et même la modifier. Regarde l'exemple suivant pour tout comprendre, puis fais mon petit exercice pour bien comprendre le fonctionnement de l'accès au tableau. »

```
1   int i = 0;
2   short T[7] = {3, -2, 1};
3   T[0] = 14;    // Modification de la case 0 du tableau T
4   T[3] = T[i];  // Modification de la case 3 du tableau T
5   for (i=0; i<7; i++)
6      T[i] = i;  // Parcours et remplissage du tableau
```

Exercice 2. Douglas modifie des tableaux

2.1) Rédigez l'instruction permettant d'affecter la valeur -2 à la case d'indice 6 du tableau `tab`.
2.2) En supposant que la variable entière `i` soit définie et initialisée, rédigez l'instruction permettant d'afficher la valeur contenue dans la case d'indice i du tableau `tab`.
2.3) Rédigez 3 instructions permettant d'afficher les cases 0, 1 et 2 du tableau `T` précédemment défini.
2.4) L'exemple précédent montre comment lire le contenu de la case `i` du tableau `T`. Utilisez ce principe pour afficher l'ensemble des cases du tableau `T` à l'aide d'une instruction de boucle.

« Ca y est ! Je sais comment faire ! Linux, mon vétérinaire, va enfin pouvoir analyser mon transit ! » s'écria Douglas. lL'ibijau lui répondit alors que si il savait créer son programme, il n'allait certainement pas être très modulable, car il ne comporterait aucun appel de fonction manipulant un tableau. En effet, Douglas a omis ce précepte important, et demanda donc à l'ibijau plus de précisions à ce sujet.

« Supposons que tu veuilles coder la fonction `Func()`, qui prend en paramètres un tableau, et qui renvoie la moyenne des `n` premiers éléments présents dans ce tableau. Tu vas devoir dans un premier temps rédiger le prototype de la fonction de la façon suivante. »

```
1   float Func (int tab[], int n)
```

L'ibijau lui expliqua alors que pour passer un tableau à une fonction en tant que paramètres, il était obligatoire d'écrire le type de données stockées dans le tableau, suivi du nom du tableau et des crochets. Nul besoin de mettre la taille du tableau entre crochets, cette dernière est passée également en argument de la fonction. Il suffit alors de réaliser

une boucle sur les `n` premiers éléments du tableau afin de les sommer dans une variable `resultat`, puis de diviser cette variable par `n` avant de la renvoyer pour obtenir le résultat souhaité.

« Oh mon ibijau, tu es vraiment trop fort ! C'est un parfait exemple de modularité que tu viens de me montrer. Mais comment puis-je appeler cette fonction sur mon tableau `tab` qui contient mes `n` données ? »

L'ibijau lui présenta alors l'instruction suivante montrant qu'il suffisait de mettre le nom du tableau dans l'appel de fonction (pas besoin de crochets ou autres) et Douglas réalisa les exercices donnés par l'ibijau pour s'entrainer.

```
1   int tab[10];
2   float res;
3   res = Func (tab, 4);
```

Exercice 3. Douglas manipule des tableaux

3.1) Rédigez le prototype de la fonction `AfficheTab()` ne retournant rien et prenant en paramètres un tableau d'entiers et sa taille.

3.2) Complétez la fonction `AfficheTab()` afin d'afficher l'ensemble des valeurs stockées dans un tableau, puis appelez-la sur le tableau `tab`.

3.3) Rédigez une fonction permettant de calculer la moyenne des éléments d'un tableau.

Encore un peu timide, et affaibli par autant de réflexion, Douglas demanda encore quelques précisions à l'ibijau sur la modification d'un tableau. Il lui assura que lorsqu'un tableau est passé en argument d'une fonction, il est possible de modifier les valeurs de ce tableau dans la fonction, de sorte que ces modifications subsistent encore après la fin de la fonction.

« Lorsque tu réalises l'appel d'une fonction dont le prototype est `void f(int tab[])` sur le tableau `T`, tu écris `f(T)`. En réalité, `tab` est considéré comme un pointeur contenant l'adresse de la première case du tableau `T`. Ainsi, lorsque tu écris dans ta fonction l'instruction `tab[i] = x`, tu modifies ce qu'il y a `i` cases après l'adresse stockée dans `tab`, ce qui revient à modifier la case `i` du tableau `T`. Tout cela semble très compliqué, mais ne t'inquiètes pas, ton maître saura t'expliquer cela très prochainement. Retiens simplement qu'il est possible de modifier un tableau en le passant à une fonction. »

« Courage mon Doudou, conclut l'ibijau, faut pas se laisser aller ! »

Exercice 4. Douglas compte ses crottes

Douglas a encore besoin d'un coup de main, mais ne veut pas se rendre ridicule devant l'ibijau. Linux lui a demandé de vérifier si la répartition de ses crottes au cours d'une journée était normale. Dans un premier temps, il vous demande de l'aider à saisir la répartition de ses crottes dans son programme.

4.1) Rédigez la fonction `SaisirCrottes()` prenant en paramètres un tableau d'entiers et sa taille `n`. La fonction doit remplir le tableau passé en paramètre en demandant à l'utilisateur de saisir le nombre de crottes que Douglas a produit chaque heure. Chaque nouvelle valeur doit être enregistrée dans une nouvelle case. A la fin de la saisie, l'utilisateur doit renseigner un nombre négatif pour terminer la saisie. Attention, l'utilisateur ne peut pas saisir plus de `n` valeurs ! La valeur retournée par la fonction correspond au nombre de cases remplies.

3. Tableaux 2D : Linux est dubitatif

Douglas put enfin voir son vétérinaire Linux pour qu'il analyse 3 jours de transit. Il reçut donc un tableau de 72 valeurs et le regarda d'un œil dubitatif.

« J'ai un souci avec votre tableau M. Douglas. Nous avons l'habitude, en tant que vétérinaires spécialisés dans le transit, d'étudier la répartition des crottes non pas heure par heure, mais par période de 6 heures. Aussi, j'ai besoin que vous transformiez ce tableau de façon à faire apparaitre le nombre de crottes produites la nuit, entre minuit et 6h, le matin entre 6h et 12h, l'après-midi entre 12h et 18h et le soir entre 18h et minuit. Revenez me voir avec ces données. »

Abattu par ses informations, il retourna dépité dans son enclot et resta prostré dans un coin de sa cabane. L'ibijau lui demanda alors ce qu'il se passait et après les explications de Douglas, trouva une idée pour solutionner son problème.

« Il existe le concept des *tableaux à deux dimensions*. Il s'agit en quelque sorte de tableaux de tableaux, qui permettent de stocker un ensemble de séries de données. Tu pourras ainsi avoir un tableau des jours de la semaine, et pour chaque jour, 4 valeurs donnant le nombre de crottes que tu as produites la nuit, le matin, l'après-midi et le soir. Ce n'est pas beaucoup plus compliqué que les tableaux 1D et Linux pourra étudier plus facilement ces valeurs. »

L'ibijau expliqua alors qu'un tableau 2D se déclarait un peu de la même façon qu'un tableau 1D, en écrivant d'abord le type des données, le nom du tableau puis la taille de chaque dimension entre crochets successifs. Et parce qu'il sait qu'un exemple est ce qu'il y a de mieux pour Douglas, il lui montra la série d'exemples suivants.

```
1  int tab2D[3][2];   // Declaration d'un tableau 3 lignes, 2 colonnes
2  short matrice[2][3] = {{1,2,3},{4,5,6}};
3  matrice[0][1] = 2; // Acces et modification d'un element
```

« Comme tu peux le voir, expliqua l'ibijau, là encore, il est possible de remplir un tableau 2D lors de sa déclaration. Le tableau nommé `matrice` comporte par exemple deux lignes de trois éléments et peut être schématisé de la façon suivante. »

	col.0	col.1	col.2
ligne 0	1	2	3
ligne 1	4	5	6

« Pour accéder à l'élément de la ligne 0, colonne 1, tu écris donc `matrice[0][1]`, qui vaut 2. En effet, `matrice[0]` est considéré comme un tableau, non comme une valeur. Il est donc impossible d'écrire `matrice[0] = 10`. En revanche, puisque `matrice[0]` est un tableau à 3 éléments, on peut accéder à l'élément d'indice `1` en écrivant `matrice[0][1]`. »

« Il est ainsi possible, comme pour les tableaux 1D, de parcourir l'ensemble des valeurs de ton tableau 2D. Pour cela, il te faut parcourir l'ensemble des lignes, appelées ici `matrice[i]`, pour `i` allant de 0 à 1, puis pour chaque ligne, parcourir l'ensemble des valeurs `matrice[i][j]` pour `j` allant de 0 à 2. Note que l'on appelle aussi les tableaux 2D des matrices. Entraine-toi sur les quelques exemples suivants pour bien comprendre. »

Exercice 5. Douglas pratique les tableaux 2D

5.1) Déclarez une matrice nommée `tab`, comportant 3 lignes et 7 colonnes.

5.2) Déclarez et initialisez une matrice nommée `mat`, comportant 2 lignes et 4 colonnes, et contenant sur la première ligne les valeurs 10, 20, 30 et 40, et sur la seconde ligne les valeurs -10, -20, -30 et -40.

5.3) Modifiez l'élément de la ligne 0 de la colonne 5 du tableau `tab` pour qu'il vaille -1.

5.4) Combien la matrice `tab` comporte-t-il d'éléments ?

« Et comme tout à l'heure, demanda Douglas, est-il possible d'utiliser des fonctions avec des matrices ? » L'ibijau lui répondu par l'affirmative en lui présentant le prototype de la fonction suivante.

```
1   float Moyenne (int mat[][2], int nbLignes, int nbCol);
```

Il lui expliqua que pour passer une matrice en paramètre d'une fonction, il était obligatoire d'indiquer le nombre d'éléments présents sur chaque ligne en renseignant ce nombre dans les seconds crochets. En effet, si l'on n'indiquait pas cette valeur, comment savoir où se situent les données en mémoire de la ligne 1, si l'on ne sait pas de combien d'éléments on doit se décaler pour y accéder ? Il indiqua également qu'il était souvent intéressant de donner également le nombre de lignes et de colonnes afin de se déplacer sans soucis dans la matrice. Finalement, l'appel de la fonction était ce qu'il y avait de plus simple à réaliser car comme pour les tableaux 1D, il suffisait de donner le nom du tableau comme dans l'exemple ci-dessous.

```
1   int tab2D[3][2];
2   float res;
3   res = Moyenne (tab2D, 3, 2);
```

« Tu peux donc réaliser ton programme pour Linux, indiqua l'ibijau. Nous allons réaliser une fonction qui prendra en argument un tableau 1D correspondant aux crottes que tu as produites chaque heure, ainsi qu'une matrice 7 lignes, 4 colonnes, qui présentera pour chaque jour de la semaine, le nombre de crottes de la nuit, du matin, de l'après midi et du soir. Cette fonction lira le premier tableau, pour remplir correctement la matrice. »

Douglas se sentit bien découragé face à l'ampleur de cette tâche. Mais heureusement, l'ibijau était un bon professeur et lui proposa de bons exercices pour s'entrainer.

Exercice 6. Douglas pratique la sorcellerie

Réalisez tout d'abord ces exercices d'entrainement avant de pratiquer la magie noire !

6.1) Réalisez la fonction `void AfficheMatrice (int mat[][4], int nbLignes)` permettant d'afficher l'ensemble des valeurs de la matrice `mat` comportant `nbLignes` lignes et 4 colonnes. Un retour à la ligne est nécessaire après l'affichage de chaque ligne de la matrice.

6.2) Réalisez une fonction permettant de calculer la moyenne des valeurs présentes dans une matrice.

Vous êtes alors prêts à passer du côté obscur de la force... Réalisez la fonction demandée par Linux en vous servant des indications de l'ibijau présentées dans les questions suivantes.

6.3) La fonction doit prendre en arguments un tableau 1D et une matrice 7 lignes, 4 colonnes. Quelle(s) variable(s) doit-on ajouter pour parcourir le tableau et la matrice sans soucis ?

6.4) Vous devez maintenant lire le tableau 6 cases par 6 cases pour stocker leur somme dans une des cases de la matrice. Combien de variables sont nécessaires ? Déclarez-les puis réalisez le code nécessaire au parcours du tableau.

6.5) Pour chaque groupe de 6 valeurs lues dans le tableau, vous devez les stocker dans une nouvelle case de la matrice. Cela implique de changer de ligne sur la matrice toutes les 4 cases remplies, et donc toutes les 24 cases du tableau lues. De quelle opération arithmétique avez-vous besoin pour parcourir correctement cette matrice ?

6.6) Nous supposerons que si le tableau ne contient pas assez d'informations pour remplir complètement la matrice, nous laisserons les cases à 0. Cela suppose que la matrice possède toutes les valeurs à 0 avant de la remplir. Implémentez cette action au début de votre fonction.

6.7) Vous avez maintenant toutes les informations pour mener à bien votre tâche. Utilisez la déclaration de tableau ci-dessous présentant le transit intestinal de Douglas sur 72 heures et affichez la matrice associée pour vérifier que votre code est correct.

```
int transit[72] = {18, 16, 20, 26, 30, 15, 12, 4, 0, 3, 12, 18,
  13, 7, 0, 0, 1, 5, 17, 20, 21, 20, 17, 13,
  27, 34, 21, 29, 34, 16, 7, 5, 7, 13, 0, 16,
   7, 6, 0, 0, 0, 24, 26, 21, 34, 18, 13, 14,
   7, 15, 8, 6, 9, 11, 6, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 7, 9, 12, 8, 7, 16};
```

4. Un peu de statistiques

Enfin, après tant de labeur, Douglas put partir voir Linux pour se faire soigner. Ce dernier lui expliqua comment il allait réaliser son diagnostic.

« En théorie, un lapin produit 40% de ses crottes journalières la nuit, 20 % le matin, 10% l'après midi et 30% le soir. On veut vérifier si le tableau de crottes que vous avez réellement émis sur 24h suit cette loi de probabilité. Soit n le nombre total de crottes que vous avez émis sur la journée. Soit $t[i]$ le pourcentage de crottes théoriquement émises dans l'intervalle horaire $[6i, 6(i+1)[$ et $r[i]$ le nombre que vous avez réellement émis durant cette même période. Il est possible de vérifier si la répartition de vos crottes au cours d'une journée suit la répartition théorique en réalisant un *test du* χ^2. Il est nécessaire pour cela de calculer la valeur v de la façon suivante.

$$v = \sum_{i=0}^{3} \frac{(r[i] - nt[i])^2}{nt[i]}.$$

Plus cette valeur v est petite, plus la répartition observée est proche de la répartition théorique. On considère que vous avez très certainement un problème de transit quand $v > 7,81$. »

Exercice 7. Linux annonce son verdict

Aidez Linux à prendre sa décision en réalisant l'exercice suivant.

7.1) Créez une fonction prenant en entrée un tableau de quatre cases donnant le nombre de crottes émises par Douglas la nuit, le matin, l'après-midi et le soir et renvoyant la valeur v.

7.2) Utilisez vos codes précédents afin de lancer un test du χ^2 sur les trois premières journées du transit présentées précédemment.

Après que Linux ait expliqué à Douglas le fonctionnement d'un test du χ^2 il le lança sur le jeu de données de Douglas et put observer un fait inhabituel.

« M. Douglas, il y a un souci avec vos données. Sur les trois jours d'étude, vous indiquez une période sans aucune crotte pendant plus de 10h, ce qui est strictement impossible pour un lapin sans mourir. Êtes-vous mort ? Non. Je pense que vous êtes trop distrait, et qu'il vous faut un système de suivi plus simple et plus fiable à mettre en place. »

La mort dans l'âme, et la boule au ventre, il dût attendre le retour des étudiants de son maître car l'ibijau jouissait, lui, d'un bon transit et était indisponible.

5. Sauvons Douglas

5.1. Résumé des épisodes précédents

```
int transit[72] = {18, 16, 20, 26, 30, 15, 12, 4, 0, 3, 12, 18,
  13, 7, 0, 0, 1, 5, 17, 20, 21, 20, 17, 13,
  27, 34, 21, 29, 34, 16, 7, 5, 7, 13, 0, 16,
  7, 6, 0, 0, 0, 24, 26, 21, 34, 18, 13, 14,
  7, 15, 8, 6, 9, 11, 6, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 7, 9, 12, 8, 7, 16};
```

FIGURE 1 – Exemple de tableau horaire de transit

Récapitulons dans un premier temps ce que vous avez réalisé jusqu'à présent. Douglas devant suivre son transit quotidiennement, vous avez dans un premier temps codé un programme lui permettant de stocker dans un tableau le nombre de crottes qu'il a émis chaque heure. Nommons ce genre de tableau un *tableau horaire*. Un exemple de ce type de tableau est présenté dans la figure 1.

```
int mat_transit[7][4] = {{125, 49, 26, 108},
  {161, 48, 37, 126},
  { 56,  6,  0,  59},
  {  0,  0,  0,   0},
  {  0,  0,  0,   0},
  {  0,  0,  0,   0},
  {  0,  0,  0,   0}};
```

FIGURE 2 – Exemple de matrice hebdomadaire

Par la suite, les matrices vous ont permis de représenter le transit de Douglas par plage horaires de 6 heures, chaque ligne comportant le transit d'une journée. Vous avez ainsi obtenu une matrice 7 lignes, 4 colonnes, que nous appellerons par la suite *matrice hebdomadaire*. Un exemple de ce type de matrice est présenté dans la figure 2 qui est la matrice hebdomadaire associé au tableau horaire présenté dans la figure 1.

5.2. Suivi du transit simplifié

La pause a été bénéfique pour l'ibijau qui a enfin trouvé un moyen d'aider Douglas. Rappelons que ce dernier, trop distrait, n'arrivait pas à compter le nombre de crottes qu'il émettait chaque heure.

« Je peux te créer un programme qui permettrait de remplir une nouvelle case du tableau à chaque fois que tu vas sur ta litière. Imaginons que tu lances ton programme pour qu'il tourne pendant 2 jours. Chaque fois que tu vas sur ta litière et que tu émets une crotte, tu saisis une touche sur ton clavier. Cela va remplir une case de ton tableau avec l'heure actuelle. A la fin des 2 jours, chaque case de ton tableau contiendra l'instant à laquelle tu as produit une crotte. Mais j'ai peur que Linux ne sache pas lire ce tableau. Ce n'est qu'un chien après tout ... »

C'est alors que l'ibijau se mit au travail afin de résoudre le problème de Douglas et enfin le délivrer de cette douleur infernale.

Exercice 8. Suivi simplifié

Vous connaissez la fonction `time(NULL)` qui vous permet de récupérer le nombre de secondes écoulées depuis le 1er janvier 1970. Le principe du code que vous allez réaliser est le suivant. Le programme demande un nombre à l'utilisateur. Lorsque celui-ci le saisit (un nombre quelconque), le temps courant est enregistré dans un entier. Lors d'une deuxième saisie, la différence est faite entre le nouveau temps courant et le temps précédemment enregistré pour mesurer l'écart de temps en secondes entre les deux saisies.

8.1) Réalisez un programme calculant et affichant le temps qu'il s'est écoulé entre deux saisies utilisateur.

8.2) Complétez votre programme pour continuer le processus tant que l'utilisateur ne saisit pas l'entier 0.

Douglas peut maintenant utiliser votre code pour générer son tableau de surveillance. Afin que Linux puisse l'analyser, il faut transformer les données sous la forme d'une matrice dont chacune des sept lignes représente un jour de la semaine, et chacune des quatre colonnes représente une tranche de six heures sur une journée. La valeur entière dans la case `[i][j]` de la matrice indique donc le nombre de crottes émises dans l'intervalle horaire $[6j, 6(j+1)[$ du jour `i`.

Exercice 9. Transformation des données

9.1) Créez le programme permettant de transformer les données de temps en données d'émission de crottes par plage horaire. Votre programme doit prendre en entrée un tableau 1D donnant les intervalles de temps entre chaque crottes ainsi qu'une matrice hebdomadaire vide. Votre programme doit tout d'abord transformer les données de temps en tableau horaire de transit avant de remplir la matrice hebdomadaire à l'aide des codes déjà réalisés dans l'exercice 6.

9.2) Afin de vérifier le bon fonctionnement de votre code, ajouter à votre compilation le fichier .o fourni sur moodle. En appelant la fonction `Simulate(tab, nbDays, seed)`, vous complétez le tableau `tab` en simulant le suivi simplifié du transit de pantoufle sur `nbDays` jours. La valeur `seed` vous permet de générer des jeux de données différents. Utilisez donc ce programme pour simuler le tableau `tab` à convertir en matrice hebdomadaire.

Il est maintenant temps de vérifier si les jeux de données fournis par le .o correspondent à un transit normal de lapin. Pour cela, utilisez la matrice `mat_transit` définie ci-dessus qui présente le transit de Douglas sur trois jours consécutifs, sous la forme matricielle.

Exercice 10. Analyse des données

10.1) Vous devez tester dans un premier temps si le nombre de crottes journalières est normal. Rédigez la fonction `NbCrottesJour()` prenant en paramètres une matrice de transit sur 7 jours, et retournant l'indice du premier jour de transit anormal (nombre de crottes journalières non compris entre 300 et 400), -1 si tous les jours sont bons.

10.2) Vous devez maintenant analyser l'adéquation du transit journalier à la loi normale des lapins. Utilisez les fonctions que vous avez rédigées lors de l'exercice 7 de l'APP afin de créer la fonction `RegulariteTransit()` prenant en paramètres une matrice de transit sur 7 jours, et retournant l'indice du premier jour de transit anormal, -1 si tous les jours sont bons. On rappelle qu'un lapin expulse en moyenne 40% de ses crottes journalières la nuit, 20 % le matin, 10% l'après midi et 30% le soir.

6. Conclusion

Vous avez enfin sauvé Douglas ! Linux a pu lire correctement les données de son transit et a annoncé le verdict suivant.

« M. Douglas, vous souffrez visiblement d'un blocage de transit lié à une de vos boules poils que vous avez malencontreusement avalées avec votre nourriture. Rien de bien grave, prenez simplement 1 comprimé de MaxiPurge™ matin, midi et soir pendant 2 jours, et tout rentrera dans l'ordre ! »

Douglas pu donc passer à la pharmacie pour s'acheter ses comprimés et fût soigné plus rapidement que prévu. Deux semaines plus tard, il se retrouvera tout de même chez son vétérinaire Linux pour un limage de dents.

Suite à ce franc succès de la médecine moderne, Linux devint une personnalité influente et reconnue. C'est d'ailleurs en son honneur que Linus Torvalds décida de nommer son système d'exploitation ainsi.

L'ibijau quand à lui succomba à la facilité du Java et perdit ses amis qui ne le reconnaissaient plus.