TRAVAUX DE RECHERCHES SUR LA DOMESTICATION DU TEMPS

Par Frank LOBA

F4HDB

www.f4hdb.fr

(Rédigé le mercredi 11 novembre 2020)

Propos

Ce document présente un premier programme informatique (loba1.c), dont l’objectif est d’essayer d’avoir accès à la connaissance de l’avenir. Il est écrit en langage C.

Le code source

```
// LOBA1.C
// Ecrit par Frank LOBA
// www.f4hdb.fr

/**** DEBUT DE FICHIER ****/
#ifndef LOBA1_C
#define LOBA1_C


/**** INCLUSION DES FICHIERS SUPPORTS ****/
#include <stdlib.h>
#include <stdio.h>


/**** DEFINITION DES CONSTANTES SYMBOLIQUES ****/
#define TIRAGES         "loto.txt"

#define ANCIENNETE      100


/**** DEFINITION DES TYPES DE DONNEES PERSONALISES ****/
typedef unsigned char   byte;


/**** DEFINITION DE LA FONCTION PRINCIPALE ****/
int main(void)
{
      // Déclaration des variables locales.
      unsigned long     Tirages[ANCIENNETE];

      unsigned long     Ref;
      byte              Bl1;
      byte              Bl2;
      byte              Bl3;
      byte              Bl4;
      byte              Bl5;
      FILE*             pFS;

      unsigned long     Cpt1, Cpt2, Cpt3, Cpt4, Cpt5, Cpt6;

      unsigned long     Tableau1[ANCIENNETE][ANCIENNETE];
      unsigned long     Tableau2[ANCIENNETE][ANCIENNETE];

      unsigned long     Mnt1, Mnt2;
```

```
// Consultation des tirages du LOTO.
pFS = NULL;
pFS = fopen(TIRAGES, "r");

if(pFS != NULL)
{
	Cpt1 = 0;
	while(fscanf(pFS, "%lu %hhu %hhu %hhu %hhu %hhu",
	&Ref, &Bl1, &Bl2, &Bl3, &Bl4, &Bl5) != EOF) Cpt1 ++;

	if(Cpt1 > ANCIENNETE - 1)
	{
		rewind(pFS);

		for(Cpt2 = 0; Cpt2 < (Cpt1 - ANCIENNETE); Cpt2 ++)
		fscanf(pFS, "%lu %hhu %hhu %hhu %hhu %hhu",
		&Ref, &Bl1, &Bl2, &Bl3, &Bl4, &Bl5);

		for(Cpt2 = 0; Cpt2 < ANCIENNETE; Cpt2 ++)
		{
			fscanf(pFS, "%lu %hhu %hhu %hhu %hhu %hhu",
			&Ref, &Bl1, &Bl2, &Bl3, &Bl4, &Bl5);

			Tirages[Cpt2] = Bl1 | (Bl2 << 6) | (Bl3 << 12)
			| (Bl4 << 18) | (Bl5 << 24);
		}

		fclose(pFS);
	}
	else
	{
		fclose(pFS);
		printf("Erreur\n");
		printf("Echec de consultation des tirages\n");
		putchar('\n');
		return 0;
	}
}
else
{
	printf("Erreur\n");
	printf("Echec de consultation des tirages\n");
	putchar('\n');
	return 0;
}
```

```
	// Construction des tableaux.
	for(Cpt1 = 0; Cpt1 < ANCIENNETE; Cpt1 ++)
	Tableau1[0][Cpt1] = Tirages[Cpt1];

	for(Cpt1 = 1; Cpt1 < ANCIENNETE; Cpt1 ++)
	for(Cpt2 = 0; Cpt2 < ANCIENNETE - Cpt1; Cpt2 ++)
	{
		Tableau1[Cpt1][Cpt2] = 0L;

		for(Cpt3 = 0; Cpt3 <= Cpt2 + 1; Cpt3 ++)
		Tableau1[Cpt1][Cpt2] ^= Tableau1[Cpt1 - 1][Cpt3];
	}

	for(Cpt1 = 0; Cpt1 < ANCIENNETE; Cpt1 ++)
	Tableau2[0][Cpt1] = Tableau1[Cpt1][ANCIENNETE - 1 - Cpt1];

	for(Cpt1 = 1; Cpt1 < ANCIENNETE; Cpt1 ++)
	for(Cpt2 = 0; Cpt2 < ANCIENNETE - Cpt1; Cpt2 ++)
	{
		Tableau2[Cpt1][Cpt2] = 0L;

		for(Cpt3 = 0; Cpt3 <= Cpt2 + 1; Cpt3 ++)
		Tableau2[Cpt1][Cpt2] ^= Tableau2[Cpt1 - 1][Cpt3];
	}


	// Réalisation des mesures.
	for(Cpt1 = 0; Cpt1 < ANCIENNETE - 1; Cpt1 ++)
	for(Cpt2 = Cpt1 + 1; Cpt2 < ANCIENNETE ; Cpt2 ++)
	for(Cpt3 = 0; Cpt3 < ANCIENNETE - Cpt2; Cpt3 ++)

	for(Cpt4 = 0; Cpt4 < ANCIENNETE - 1; Cpt4 ++)
	for(Cpt5 = Cpt4 + 1; Cpt5 < ANCIENNETE; Cpt5 ++)
	for(Cpt6 = 0; Cpt6 < ANCIENNETE - Cpt5; Cpt6 ++)
	{
		Mnt1 = 0L;
		Mnt2 = 0L;

		Mnt1 ^= Tableau2[Cpt1][Cpt3];
		Mnt1 ^= Tableau2[Cpt2][Cpt3];

		Mnt2 ^= Tableau1[Cpt4][Cpt6];
		Mnt2 ^= Tableau1[Cpt5][Cpt6];

		if(Mnt1 == Mnt2 && Cpt6 + Cpt5 > Cpt3 + Cpt2)
		printf("%lu\t%lu\t%lu\t%lu\t%lu\t%lu\t%lu\n",
		Cpt1, Cpt2, Cpt3, Cpt4, Cpt5, Cpt6, Mnt1);
	}

	// Fin de la fonction principale.
	return 0;
}


/**** FIN DE FICHIER ****/
#endif
```

Quelques explications

Les fichiers en-têtes

```
/**** INCLUSION DES FICHIERS SUPPORTS ****/
#include <stdlib.h>
#include <stdio.h>
```

Le programme nécessite l'inclusion de deux fichiers en-têtes (stdlib.h et stdio.h).

Les constantes symboliques

```
/**** DEFINITION DES CONSTANTES SYMBOLIQUES ****/
#define TIRAGES         "loto.txt"

#define ANCIENNETE      100
```

Pour fonctionner, le programme a besoin d'un historique des tirages du LOTO. Ce dernier contient suffisamment de désordre. Le nom du fichier (loto.txt) de l'historique des tirages du LOTO est mémorisé dans la constante TIRAGES. Il est possible de modifier le nom du fichier si vous l'appelez autrement.

La constante ANCIENNETE est le nombre de tirages dans l'historique que l'on considère. Il est possible de modifier cette valeur, qui dans cet exemple vaut 100. Mais cette valeur, ne doit pas dépasser le nombre total de tirages du fichier de l'historique.

Les types de données personnalisés

```
/**** DEFINITION DES TYPES DE DONNEES PERSONALISES ****/
typedef unsigned char   byte;
```

Il y en a un seul, le type byte, qui signifie octet. C'est un ensemble de 8 bits, équivalents à un caractère non signé.

Les variables locales

```
// Déclaration des variables locales.
unsigned long     Tirages[ANCIENNETE];

unsigned long     Ref;
byte              Bl1;
byte              Bl2;
byte              Bl3;
byte              Bl4;
byte              Bl5;
FILE*             pFS;

unsigned long     Cpt1, Cpt2, Cpt3, Cpt4, Cpt5, Cpt6;

unsigned long     Tableau1[ANCIENNETE][ANCIENNETE];
unsigned long     Tableau2[ANCIENNETE][ANCIENNETE];

unsigned long     Mnt1, Mnt2;
```

Le tableau de l’historique des tirages du LOTO

```
unsigned long     Tirages[ANCIENNETE];
```

Le tableau de l’historique des tirages du LOTO (Tirages) a une dimension. Chaque champ est codé sur un entier long de 32 bits non signé. Le nombre total de champs du tableau est égal à la constante symbolique ANCIENNETE. Ce permet de charger en mémoire vive, la portion du fichier de l’historique des tirages du LOTO que l’on considère.

Les variables tampons de consultation de l’historique des tirages du LOTO

```
unsigned long     Ref;
byte              Bl1;
byte              Bl2;
byte              Bl3;
byte              Bl4;
byte              Bl5;
FILE*             pFS;
```

Toutes ces variables sont nécessaires pour le chargement en mémoire vive, dans le tableau Tirages, de la portion du fichier de l’historique des tirages du LOTO que l’on considère.

pFS est un pointeur nécessaire aux fonctions de manipulations du fichier de l’historique des tirages du LOTO. Il permet d’identifier ce dernier.

Ref est un entier long non signé codé sur 32 bits. Il permettra de mémoriser la référence d’un tirage.

Bl1 à Bl5, sont cinq variables chacune de type octet codé sur 8 bits. Elles permettent de mémoriser les cinq boules d’un tirage.

Les compteurs

```
unsigned long     Cpt1, Cpt2, Cpt3, Cpt4, Cpt5, Cpt6;
```

Ces six compteurs sont chacun du type entier long non signé, codés sur 32 bits. Ils servent pour différentes manipulations.

Les tableaux

```
unsigned long     Tableau1[ANCIENNETE][ANCIENNETE];
unsigned long     Tableau2[ANCIENNETE][ANCIENNETE];
```

Il y en a deux. Chaque tableau a deux dimensions. On peut voir chaque tableau, comme un carré, de côté ayant la même longueur, que la portion de l'historique des tirages du LOTO considérée. C'est à dire ANCIENNETE. Chaque champ de ces tableaux est un entier long non signé, codé sur 32 bits.

Les moments

```
unsigned long     Mnt1, Mnt2;
```

Il y en a deux. Ce sont deux entiers longs non signés, codés sur 32 bits chacun. Ils seront utiles dans les mesures.

L'historique des tirages du LOTO (loto.txt)

```
2008081     19    33    41    24    27
2008082     24    48    32    41    22
2008083     46    22    10    39    20
2008084     11    10    41    37    48
2008085     49    28    18    20    40
2008086     44    35    4     32    39
2008087     43    7     48    31    15
2008088     11    28    12    48    14
2008089     13    11    22    46    19
2008090     46    37    44    4     6
```

Ceci est un extrait du fichier (loto.txt), de l'historique des tirages successifs du LOTO. Ils sont dans l'ordre de sortie des boules. Chaque ligne est un tirage. Il y a d'abord la référence, pour pouvoir identifier le tirage. Ensuite, il y a les cinq boules. Toutes ces valeurs sont séparées par des tabulations.

Consultations des tirages du LOTO

```
// Consultation des tirages du LOTO.
pFS = NULL;
pFS = fopen(TIRAGES, "r");

if(pFS != NULL)
{
      Cpt1 = 0;
      while(fscanf(pFS, "%lu %hhu %hhu %hhu %hhu %hhu",
      &Ref, &Bl1, &Bl2, &Bl3, &Bl4, &Bl5) != EOF) Cpt1 ++;

      if(Cpt1 > ANCIENNETE - 1)
      {
            rewind(pFS);

            for(Cpt2 = 0; Cpt2 < (Cpt1 - ANCIENNETE); Cpt2 ++)
                  fscanf(pFS, "%lu %hhu %hhu %hhu %hhu %hhu",
                  &Ref, &Bl1, &Bl2, &Bl3, &Bl4, &Bl5);

            for(Cpt2 = 0; Cpt2 < ANCIENNETE; Cpt2 ++)
            {
                  fscanf(pFS, "%lu %hhu %hhu %hhu %hhu %hhu",
                  &Ref, &Bl1, &Bl2, &Bl3, &Bl4, &Bl5);

                  Tirages[Cpt2] = Bl1 | (Bl2 << 6) | (Bl3 << 12)
                  | (Bl4 << 18) | (Bl5 << 24);
            }

            fclose(pFS);
      }
      else
      {
            fclose(pFS);
            printf("Erreur\n");
            printf("Echec de consultation des tirages\n");
            putchar('\n');
            return 0;
      }
}
else
{
      printf("Erreur\n");
      printf("Echec de consultation des tirages\n");
      putchar('\n');
      return 0;
}
```

Ouverture du fichier en lecture

```
pFS = NULL;
pFS = fopen(TIRAGES, "r");

if(pFS != NULL)
{
      ...
}
else
{
      printf("Erreur\n");
      printf("Echec de consultation des tirages\n");
      putchar('\n');
      return 0;
}
```

On ouvre le fichier en mode lecture seule, grâce à la fonction fopen. Si le pointeur pFS est différent de NULL, cela signifie que le fichier est ouvert avec succès. Alors on effectue le traitement. Sinon, on affiche un message d'erreur, et on met fin au programme.

Comptage du nombre de tirages dans le fichier

```
Cpt1 = 0;
while(fscanf(pFS, "%lu %hhu %hhu %hhu %hhu %hhu",
&Ref, &Bl1, &Bl2, &Bl3, &Bl4, &Bl5) != EOF) Cpt1 ++;

if(Cpt1 > ANCIENNETE - 1)
{
      …
}
else
{
      fclose(pFS);
      printf("Erreur\n");
      printf("Echec de consultation des tirages\n");
      putchar('\n');
      return 0;
}
```

Tant que l'on arrive à lire une ligne de tirage, on incrémente le compteur Cpt1. S'il y a assez de tirages par rapport à l'ancienneté que l'on considère, on effectue le traitement. Sinon, on affiche un message d'erreur, et on met fin au programme.

Chargement des tirages en mémoire vive

```
rewind(pFS);

for(Cpt2 = 0; Cpt2 < (Cpt1 - ANCIENNETE); Cpt2 ++)
      fscanf(pFS, "%lu %hhu %hhu %hhu %hhu %hhu",
      &Ref, &Bl1, &Bl2, &Bl3, &Bl4, &Bl5);

for(Cpt2 = 0; Cpt2 < ANCIENNETE; Cpt2 ++)
{
      fscanf(pFS, "%lu %hhu %hhu %hhu %hhu %hhu",
      &Ref, &Bl1, &Bl2, &Bl3, &Bl4, &Bl5);

      Tirages[Cpt2] = Bl1 | (Bl2 << 6) | (Bl3 << 12)
      | (Bl4 << 18) | (Bl5 << 24);
}

fclose(pFS);
```

On se place au début du fichier et on avance jusqu'au premier tirage qui nous intéresse. Ensuite, on stocke dans le tableau Tirages, les derniers tirages de l'historique, qui sont au nombre de la valeur d'ANCIENNETE. Dans cet exemple, on charge donc les 100 derniers tirages du fichier (loto.txt). Ensuite, on ferme le fichier.

Il faut 6 bits pour coder une boule d'un tirage. Il faut donc 30 bits pour mémoriser les cinq boules d'un tirage. Chaque champ du tableau Tirages comporte 32 bits, c'est donc suffisant. Il reste 2 bits inutilisés. Les six bits de poids le plus faible servent à stocker la première boule Bl1. Bl2 est décalé de 6 bits, bl3 de 12 bits, Bl4 de 18 bits et enfin Bl5 de 24 bits. Les bits de Bl5 auront donc le poids le plus fort.

Bits	**31**	**30**	**29..24**	**23..18**	**17..12**	**11..6**	**5..0**
Tirages[..]	0	0	Boule 5	Boule 4	Boule 3	Boule 2	Boule 1

Tirages[0] est le tirage le plus ancien, et Tirages[ANCIENNETE – 1] est le tirage le plus récent.

Cela permet d'avoir un tableau ayant des champs de 30 bits suffisamment aléatoires.

Construction des tableaux

Premier tableau

```
for(Cpt1 = 0; Cpt1 < ANCIENNETE; Cpt1 ++)
Tableau1[0][Cpt1] = Tirages[Cpt1];

for(Cpt1 = 1; Cpt1 < ANCIENNETE; Cpt1 ++)
for(Cpt2 = 0; Cpt2 < ANCIENNETE - Cpt1; Cpt2 ++)
{
      Tableau1[Cpt1][Cpt2] = 0L;

      for(Cpt3 = 0; Cpt3 <= Cpt2 + 1; Cpt3 ++)
      Tableau1[Cpt1][Cpt2] ^= Tableau1[Cpt1 - 1][Cpt3];
}
```

Le premier tableau a deux dimensions.

```
for(Cpt1 = 0; Cpt1 < ANCIENNETE; Cpt1 ++)
Tableau1[0][Cpt1] = Tirages[Cpt1];
```

On initialise la première ligne du premier tableau, avec l'historique des tirages du LOTO, que l'on considère.

```
for(Cpt1 = 1; Cpt1 < ANCIENNETE; Cpt1 ++)
for(Cpt2 = 0; Cpt2 < ANCIENNETE - Cpt1; Cpt2 ++)
{
      Tableau1[Cpt1][Cpt2] = 0L;

      for(Cpt3 = 0; Cpt3 <= Cpt2 + 1; Cpt3 ++)
      Tableau1[Cpt1][Cpt2] ^= Tableau1[Cpt1 - 1][Cpt3];
}
```

Cette portion de code, permet d'affecter des valeurs aux champs du premier tableau, de façon triangulaire.

Expliquons le procédé, par un exemple, avec un seul bit, et une ancienneté de 10 tirages, et imaginons que la première ligne du tableau soit initialisée avec les valeurs suivantes :

```
0 1 0 0 1 0 1 1 1 0

Tableau1[0][0] = 0
Tableau1[0][1] = 1
Tableau1[0][2] = 0
Tableau1[0][3] = 0
Tableau1[0][4] = 1
etc.
Tableau1[0][8] = 1
Tableau1[0][9] = 0
```

Comment se construisent les lignes suivante ?

```
Tableau1[1][0] =  Tableau1[0][0] XOR Tableau1[0][1]
Tableau1[1][1] =  Tableau1[0][0] XOR Tableau1[0][1] XOR Tableau1[0][2]
Tableau1[1][2] =  Tableau1[0][0] XOR Tableau1[0][1] XOR Tableau1[0][2] XOR
                  Tableau1[0][3]
etc.

Tableau1[2][0] =  Tableau1[1][0] XOR Tableau1[1][1]
Tableau1[2][1] =  Tableau1[1][0] XOR Tableau1[1][1] XOR Tableau1[1][2]
Tableau1[2][2] =  Tableau1[1][0] XOR Tableau1[1][1] XOR Tableau1[1][2] XOR
                  Tableau1[1][3]
etc.

Tableau1[3][0] =  Tableau1[2][0] XOR Tableau1[2][1]
Tableau1[3][1] =  Tableau1[2][0] XOR Tableau1[2][1] XOR Tableau1[2][2]
Tableau1[3][2] =  Tableau1[2][0] XOR Tableau1[2][1] XOR Tableau1[2][2] XOR
                  Tableau1[2][3]
etc.
```

Et ainsi de suite. XOR est l'opérateur OU EXCLUSIF.

Ce qui donne avec l'exemple le triangle suivant :

```
0
1 1
0 1 0
1 1 1 1
1 0 0 0 0
1 0 1 0 0 0
1 0 1 1 1 0 0
0 1 1 1 0 0 1 0
1 1 1 0 0 1 0 1 1
0 1 0 0 1 0 1 1 1 0
```

Dans notre programme, pour le premier tableau, on traite de cette façon, en parallèle, les 30 bits des différents tirages.

Second tableau

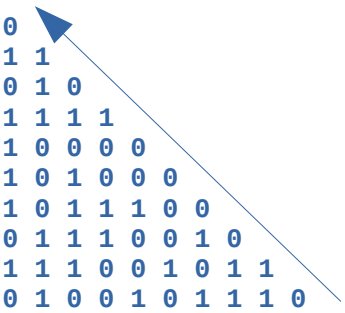
```
for(Cpt1 = 0; Cpt1 < ANCIENNETE; Cpt1 ++)
Tableau2[0][Cpt1] = Tableau1[Cpt1][ANCIENNETE - 1 - Cpt1];

for(Cpt1 = 1; Cpt1 < ANCIENNETE; Cpt1 ++)
for(Cpt2 = 0; Cpt2 < ANCIENNETE - Cpt1; Cpt2 ++)
{
      Tableau2[Cpt1][Cpt2] = 0L;

      for(Cpt3 = 0; Cpt3 <= Cpt2 + 1; Cpt3 ++)
      Tableau2[Cpt1][Cpt2] ^= Tableau2[Cpt1 - 1][Cpt3];
}
```

Le second tableau se construit de la même façon que le premier tableau. La seule différence réside dans l'initialisation de la première ligne. Pour le second tableau, on initialise la première ligne, avec la diagonale du premier tableau.

Reprenons l'exemple avec un seul bit de 10 tirages du premier tableau précédent :



La première ligne du second tableau a les valeurs de la diagonale lue du bas vers le haut :

```
Tableau2[0][0] = 0
Tableau2[0][1] = 1
Tableau2[0][2] = 0
Tableau2[0][3] = 0
Tableau2[0][4] = 0
etc.
Tableau2[0][8] = 1
Tableau2[0][9] = 0
```

On obtient le second triangle suivant dans le second tableau:

```
1
0 1
0 0 1
1 1 0 1
0 1 0 1 1
0 0 1 1 1 0
1 1 0 1 0 0 1
0 1 0 1 1 1 0 1
1 1 1 1 1 0 0 1 1
0 1 0 0 0 0 1 0 1 0
```

Réalisation des mesures

```
// Réalisation des mesures.
for(Cpt1 = 0; Cpt1 < ANCIENNETE - 1; Cpt1 ++)
for(Cpt2 = Cpt1 + 1; Cpt2 < ANCIENNETE ; Cpt2 ++)
for(Cpt3 = 0; Cpt3 < ANCIENNETE - Cpt2; Cpt3 ++)

for(Cpt4 = 0; Cpt4 < ANCIENNETE - 1; Cpt4 ++)
for(Cpt5 = Cpt4 + 1; Cpt5 < ANCIENNETE; Cpt5 ++)
for(Cpt6 = 0; Cpt6 < ANCIENNETE - Cpt5; Cpt6 ++)
{
      Mnt1 = 0L;
      Mnt2 = 0L;

      Mnt1 ^= Tableau2[Cpt1][Cpt3];
      Mnt1 ^= Tableau2[Cpt2][Cpt3];

      Mnt2 ^= Tableau1[Cpt4][Cpt6];
      Mnt2 ^= Tableau1[Cpt5][Cpt6];

      if(Mnt1 == Mnt2 && Cpt6 + Cpt5 > Cpt3 + Cpt2)
      printf("%lu\t%lu\t%lu\t%lu\t%lu\t%lu\t%lu\n",
      Cpt1, Cpt2, Cpt3, Cpt4, Cpt5, Cpt6, Mnt1);
}

// Fin de la fonction principale.
return 0;
```

Pour expliquer le principe des mesures, reprenons l'exemple avec un seul bit de 10 tirages des tableaux précédents :

Tableau 1 :

```
0
1 1
0 1 0
1 1 1 1
1 0 0 0 0
1 0 1 0 0 0
1 0 1 1 1 0 0
0 1 1 1 0 0 1 0
1 1 1 0 0 1 0 1 1
0 1 0 0 1 0 1 1 1 0
```

Tableau 2 :

```
1
0 1
0 0 1
1 1 0 1
0 1 0 1 1
0 0 1 1 1 0
1 1 0 1 0 0 1
0 1 0 1 1 1 0 1
1 1 1 1 1 0 0 1 1
0 1 0 0 0 0 1 0 1 0
```

On considère deux points verticaux de valeurs dans chaque triangle.

Cpt1 est l'ordonnée du point inférieur dans le triangle 2.
Cpt2 est l'ordonnée du point supérieur dans le triangle 2.
Cpt3 est l'abscisse commune des deux points dans le triangle 2.

Cpt4 est l'ordonnée du point inférieur dans le triangle 1.
Cpt5 est l'ordonnée du point supérieur dans le triangle 1.
Cpt6 est l'abscisse commune des deux points dans le triangle 1.

Mnt1 est le ou exclusif des deux valeurs verticales dans le triangle 2.
Mnt2 est le ou exclusif des deux valeurs verticales dans le triangle 1.

On balaye tous ces bipoints verticaux des deux triangles.

Si Cpt6 + Cpt5 est supérieur à Cpt3 + Cpt2, cela signifie que le point supérieur dans le triangle 1, à de l'avance par rapport au point supérieur dans le triangle 2. Si Mnt1 est égal à Mnt2, on affiche les coordonnées Cpt1 à Cpt6, et aussi la valeur de Mnt1, qui est la même que Mnt2.

Dans notre programme, nous effectuons ces mesures pour les 30 bits en parallèle.

Compilation et exécution du programme

Pour compiler le programme, avec GCC, dans un terminal, sous UBUNTU Linux, on utilise la ligne de commande suivante :

```
gcc -D_GNU_SOURCE -g -Wall -o loba1 loba1.c
```

Voici comment lancer le programme :

```
./loba1
```

Interprétation des résultats

Cpt1	Cpt2	Cpt3	Cpt4	Cpt5	Cpt6	Mnt1 (= Mnt2)
1	16	14	15	30	69	728609285
1	16	30	31	46	37	931560600
1	17	2	11	19	77	935083360
1	17	10	23	27	53	806689872
1	17	22	31	39	53	292104549
1	17	30	31	47	37	915273296
1	18	30	31	48	37	295708792
1	19	30	31	49	37	721967763
1	20	30	31	50	37	671210586
1	21	30	31	51	37	378291020
1	22	30	31	52	37	1006283909
1	23	30	31	53	37	844424276
1	24	30	31	54	37	317515687
1	25	2	19	27	61	1073337047
1	25	6	23	31	61	202313875
1	25	30	31	55	37	770334964
1	25	38	55	63	21	1002536611
1	26	30	31	56	37	814846185
1	27	30	31	57	37	442235975
1	28	30	31	58	37	581490249
1	29	30	31	59	37	211359991
1	30	30	31	60	37	814881582
1	31	30	31	61	37	759299498
1	32	30	31	62	37	345575542

Nous voyons ici, qu’il y a un certain ordre dans le désordre, et qu’il est possible d’avoir de l’avance.

En complétant ce programme pour réaliser des prévisions du futur, on constante que celles-ci sont aberrantes. Ce ne sont donc pas des conditions suffisantes pour connaître l’avenir. Mais c’est peut-être un petit progrès.

Merci pour votre lecture.

Table des matières

Propos....2
Le code source....3
Quelques explications....7
  Les fichiers en-têtes....7
  Les constantes symboliques....7
  Les types de données personnalisés....7
  Les variables locales....8
    Le tableau de l'historique des tirages du LOTO....8
    Les variables tampons de consultation de l'historique des tirages du LOTO....8
    Les compteurs....9
    Les tableaux....9
    Les moments....9
  L'historique des tirages du LOTO (loto.txt)....9
  Consultations des tirages du LOTO....10
    Ouverture du fichier en lecture....11
    Comptage du nombre de tirages dans le fichier....11
    Chargement des tirages en mémoire vive....12
  Construction des tableaux....13
    Premier tableau....13
    Second tableau....15
  Réalisation des mesures....16
Compilation et exécution du programme....18
Interprétation des résultats....19