REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
ECOLE NATIONALE SUPERIEURE D'INFORMATIQUE

CPI 1 **Année 2017 / 2018**

TP
EN ALGORITHMIQUE ET STRUCTURES DE DONNEES STATIQUES (ALSDS)

REALISE PAR :
BELATECHE Rahim
AZIZ Rezak

Section : B

Groupe N° : 06

Semestre 1

Table des matières

Introduction

De nos jours, les informations deviennent de plus en plus grandes et consistantes, ce qui engendre la difficulté particulière de les sauvegarder et les classer d'une manière à accéder à n'importe quelle donnée aisément.

Une application concrète du problème énoncé ci-dessus, est bel et bien la sauvegarde des informations concernant les étudiants au sein d'un établissement d'enseignement et le fait d'accéder à ces données très rapidement. Pour cela, les développeurs ont pensé à une solution pratique qui est effectuée d'une manière automatique à l'aide d'algorithmes. Cette solution est basée sur l'adressage direct qui consiste à attribuer à chaque étudiant une adresse sur le disque dur.

Le problème de la consistance des informations engendre le souci de sécuriser les données, ceci est un souci majeur pour toutes les entreprises et les administrations et notamment les particuliers

Pour régler ce problème, il faut trouver une technique pour faire de sorte qu'une personne ne puisse lire les envois même si elle réussit à en prendre connaissance. Cela n'est possible que si nous chiffrons nos messages envoyés et déchiffrons les messages reçus. Lorsqu'on code le message en clair, on obtient un cryptogramme.

La technique utilisée pour obtenir un cryptogramme est appelée « crypto système ».

On trouve plusieurs algorithmes de chiffrement : le chiffre de César, le chiffre de Vigenère, le chiffre de Vernam … etc

On sera amenés, dans le travail qui suit, à résoudre les deux problèmes énoncés ci-dessus, et ce, en utilisant un concept bien connu en algorithmiques, qui est la modularité.

Le Sujet

TP1 : Choix d'une formule de transformation

Une université accueille chaque année 15000 étudiants. L'équipe de développement de cette dernière a pour objectif de classer les données concernant les étudiants dans un fichier et y accéder très rapidement. Pour cela, elle fait appel à l'adressage direct qui permet d'affecter à chaque étudiant une adresse sur le disque dur. Ces adresses sont générées à partir des indicatifs qui caractérisent chaque étudiant selon les formules de transformations.

Il existe deux types de formules de transformations.

On trouve la première formule qui utilise la fonction Random qui génère des nombres aléatoires, l'extraction de l'adresse se fait en enlevant les 5 positions centrales de N dans le cas où N est composé d'un nombre impair de postions ; si N est composé d'un nombre pair de positions, on commence l'extraction à partir de la 2ème position de gauche incluse.

La deuxième formule est la transformation par pliage qui consiste à découper l'indicatif en 2 parties égales de 5 positions chacune, mais pour généraliser, on concevra notre module en découpant l'indicatif en 2 parties égales de x postions chacune, puis on additionne les deux nombres trouvés, le résultat sera ensuite divisé par le nombre premier le plus proche inférieurement de la taille du fichier. L'adresse représente le reste de cette division.

Après la mise en place des deux formules, l'équipe de développement doit choisir entre ces deux formules, et ce, en se basant sur les éléments suivants

1. La plage des adresses : l'écart entre la plus petite et la plus grande adresse générées par chaque formule.
2. Le nombre de collisions : des enregistrements différents mais avec une même adresse.

TP 2 : Initiation à la cryptographie

La sécurité des données est devenue un problème majeur pour tous, notamment pour les entreprises.

En effet, une entreprise veut sécuriser tous les messages envoyés à ses différentes agences, et cela, en utilisant la méthode appelée le « Chiffre de Vernam » en rajoutant une petite modification.

Il s'agit de chiffrer les messages envoyés, et de déchiffrer les messages reçus.

Le principe de chiffrement consiste à construire une clé dont les caractéristiques sont :

- Le nombre de caractères de la clé doit être supérieur ou égal à celui du message.
- Les caractères de la clé sont choisis aléatoirement.
- Une clé n'est utilisée qu'une seule fois.

Après l'obtention de la clé, on doit construire le cryptogramme qui est le message en clair codé.

Le principe de déchiffrement consiste à soustraire la valeur de la lettre de la clé de la lettre du cryptogramme correspondante, et si ce résultat est négatif on ajoute 26.

De petites modifications ont été rajoutées au Chiffre de Vernam, celles-ci sont :

- Les messages ne contiennent que des caractères alphabétiques.
- Un décalage est appliqué aux lettres de l'alphabet.
- Les messages ne dépassent jamais 250 caractères.

Le message en clair, le décalage et la clé sont mis dans un enregistrement.

Le message chiffré, le décalage et la clé sont aussi mis dans un enregistrement.

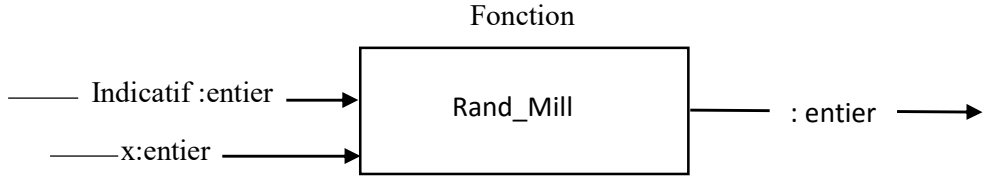
TP N°1

Choix d'une formule de transformation

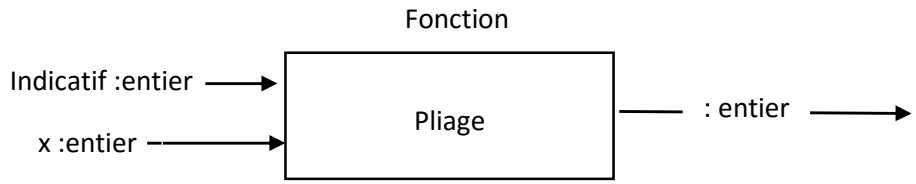
I. Découpage Modulaire

On aura besoin des modules suivants :

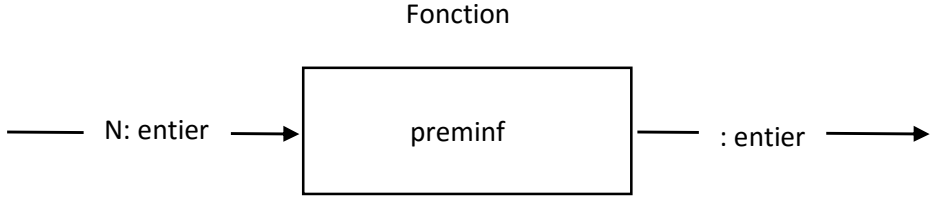
- Rand_Mill : nous génère une adresse à partir d'un indicatif selon la formule 1
- Pliage : nous donne adresse à partir d'un indicatif selon la formule 2 qui est la transformation par pliage.
- Preminf : donne le nombre premier le plus proche inférieurement de n.
- E_ident : donne le nombre d'éléments du tableau T qui sont identiques.
- Tritranspo : Tritranspo (Var A : Tab ; t : entier) // sert à trier un tableau à une dimension par transposition.
- Extr_nb : Extr_nb (n, x, p : entier) :entier // extraire de n x positions à partir de la p(ème) position de gauche incluse.
- Nbpos : nbpos(n : entier) : entier // donne le nombre de positions (chiffres) de n.
- Extrg : extrg(n,x : entier) : entier // extraire les x postions gauches de n.
- Extrd : extrd(n,x : entier) : entier // extraire les x positions droites de n.
- Premier : premier(n : entier) : booléen // donné vrai si n est un nombre premier.



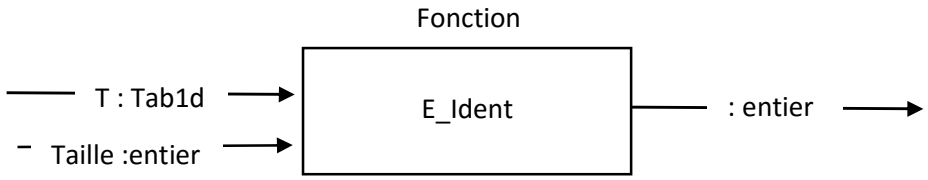
Rôle : calcule N=Random(Indicatif), puis extrait l'adresse à partir de N en enlevant les x positions centrales de N dans le cas où N est composé d'un nombre impair de positions. Dans le cas contraire, on extrait à partir de la 2ème position de gauche incluse.



Rôle : génère une adresse à partir de l'indicatif par pliage où l'indicatif est découpé en deux parties égales de x positions chacune, puis elles sont additionnées, l'adresse est le reste de la division du résultat de l'addition par le nombre premier le plus proche inférieurement de la taille du fichier.



Rôle : donne le nombre premier le plus proche inférieurement de n



Rôle : donne le nombre d'éléments du tableau T qui sont identiques.

II. Conception et réalisation

1. Fonction Rand_Mill :

i. Analyse :

Pour pouvoir fournir une adresse à partir de l'indicatif par la formule de Rand_Mill, on suit ceci :

On aura d'abord besoin des fonctions 'extr_nb' et 'nb_pos'

1. On génère un nombre aléatoire à partir de l'indicatif grâce à la fonction random et l'affecter à 'n'
2. Calculer le nombre de positions de n et l'affecter à 'np'
3. Si le nombre de positions de n est impair alors

   . calculer la position à partir de laquelle on extrait les x positions centrales de N. (p=((np-x) div2) +1)

   . affecter le résultat de l'extraction à Rand_Mill (rand_mill= extr_nb (n,x,p))
4. Sinon (nombre de position de n impair) : on extrait à partir de la 2ème position de gauche incluse et on affecte le résultat à Rand_mill.

ii. Algorithme

```
Fonction Rand_Mill (indicatif :entier ; x :entier) : entier
Variables n, np, p : entier
Fonction Nb_pos, extr_nb
DEBUT
| n ← random (indicatif)
| Np ← nb_pos(n)
| Si (np mod 2 = 1) alors
|     DSi
|     | p← ((np-x) div 2)+1
|     | Rand_mill ← extr_nb (n,x,p)
|     Fsi
|  Sinon rand_mill ← extr_nb (n,x,2)
FIN
```

iii. Jeu d'essai

indicatif	x	Rand_mill
201708745	5	95819
201714587	5	01714

iv. Programme

```
Function Rand_mil(indicatif:longint; x:longint) : longint;
Var n, np, p: longint;
{$i C:\ALSDS_TP\Module\extrnb.fon}
{$i C:\ALSDS_TP\Module\nb_pos.fon}
BEGIN
N : = random(indicatif); // on obtient un nombre pseudo aleatoire a partir de  l'indicatif
Np := nb_pos(N);
if (np mod 2 = 1) then // nombre de positions impair
        Begin
 p:=((np - x) div 2) + 1; // on commence a extraire a partir de p
 rand_Mil := extrnb(n,x,p);
        end
          else rand_Mil := extrnb(n,x,2); // nombre de positions pair
END;
```

v. Résultats du jeu d'essai

```
veuillez entrer l'indicatif:201708745

veuillez entrer le nombre de position a generer:5

l'adresse est: 95819
```

2. Fonction Preminf

i. Analyse :

Pour pouvoir trouver le nombre premier le plus proche inférieurement de n, on suit ceci

On aura besoin de la fonction 'premier'

1. On décrémente n tant qu'il n'est pas premier
2. On s'arrête au premier nombre premier trouvé
3. On affecte la valeur de 'n' à Preminf

ii. Algorithme :

```
Fonction preminf (n :entier) : entier
Fonction premier
DEBUT
  Tant que premier(n) <> vrai faire n← n-1
  Preminf ← n
FIN
```

iii. Jeu d'essai

n	premier	preminf
10	faux	
9	faux	
8	faux	
7	vrai	7
15000		14983

iv. Programme

```
FUNCTION PREMINF(n:longint):longint;
{$i C:\ALSDS_TP\Module\premier.fon}
BEGIN
while premier(n) <> true do n := n - 1; // on décrémente n et on s'arrete au premier Nb premier
preminf := n;
END;
```

v. Résultat du jeu d'essai

```
veuillez entrer votre nombre15000
le nombre premier inferieure a 15000 est: 14983
```

3. Fonction Pliage

i. Analyse :

Pour pouvoir générer une adresse à partir d'un indicatif par pliage, on suit ceci :
On aura besoin des modules 'extrdroite' , 'extrgauche' et 'Preminf'

1. On commence par extraire la partie A en faisant ceci : A=extrgauche(indicatif,x)
2. Puis on extrait la partie B en faisant ceci : B=extrdroite(indicatif, x)
3. On effectue le pliage en calculant ceci : Pliage = (A+B) mod Preminf (taiFic)

ii. Algorithme :

```
Fonction Pliage (indicatif, x, taific : entier) :entier
Variables A, B : entier
Fonctions extr_G, extr_D, Preminf
DEBUT
| A← extrgauche (indicatif, x)
| B← extrdroite (indicatif, x)
| Pliage ← (A+B) mod Preminf (TaifFic)
FIN
```

iii. Jeu d'essai

Indicatif	x	A	B	TaiFic	Pliage
201725320	5	20172	25320	15000	543
201700020	5	20170	20	15000	5207

iv. Le programme

```
function Pliage(indicatif,x,taific:longint):longint;
Var A,B:longint;
{$i C:\ALSDS_TP\Module\extdroite.fon}
{$i C:\ALSDS_TP\Module\extgauche.fon}
{$i C:\ALSDS_TP\Module\preminf.fon}
BEGIN
A:=Extgauche(indicatif,x); // on extrait la partie A
B:=Extdroite(indicatif,x); // on extrait la partie B
Pliage:=(A + B) mod Preminf(taiFic); // on effectue le pliage
END;
```

v. Résultat du jeu d'essai

```
Veuillez entrer l'indicatif: 201725320
Veuillez entrer la taille de l'adresse5
Veuillez entrer le taille de fichier15000
le code est: 11325
```

4. Fonction E_Ident

i. Analyse :

Pour pouvoir donner le nombre d'éléments du tableau T qui sont identiques, on suit ceci :
On aura besoin d'un module qui trie un tableau, on utilisera le tri par transposition dans ce cas.

1. On commence par trier le tableau
2. On initialise tot qui est le nombre total d'éléments identiques à 0.
3. On initialise nb qui est le nombre d'éléments qui se suivent à 1.
4. Pour i allant de 1 à taille-1 faire
   Si une case du tableau est identique à la case suivante on incrémente le compteur (nb=nb+1)
   Sinon
   Si un élément figure plus d'une fois alors
   On cumule les différents nb en faisant ceci (Tot=tot+nb)
   On réinitialise nb à 1
5. Si nb>1 alors tot=tot+nb
6. Affecter tot à E_ident

ii. Algorithme :

```
Fonction E_Ident (T : Tab1d ; taille : entier ) : entier
Variables tot, nb, i : entier
Fonction Transpotri
DEBUT
 Transpotri (T, taille)
 Tot ← 0
 Nb ← 1
 Pour i allant de 1 à taille-1 faire
  DP
   Si t[i] = t[i+1] alors nb ← nb+1
   Sinon
     Si nb>1 alors
                DSi
                  Tot ← tot+nb
                  Nb ← 1
                FSi
  FP
  Si nb>1 alors tot ← tot+nb
  E_ident ← tot
FIN
```

iii. Jeu d'essai :

T	I	Tot	nb	E_Ident
1 | 1 | 2 | 3 | 2 | 1	1	0	1	
1 | 1 | 1 | 2 | 2 | 3	2		2	
	3	3	3	
	4		1	
	5	5	1	5

iv. Programme :

```
Function E_Ident(T:tab;taille:longint) :longint;
var tot,nb,i:longint;
{$i C:\ALSDS_TP\Module\tritrans.pro}
BEGIN
Tri_trans(T,taille); // on trie le tableau
Tot:=0; // le nombre total d'elements identiques
nb:=1; // le nombre d'elements identiques qui se suivent
For i := 1 to taille - 1 do
Begin
if t[i] = t[i+1] then nb :=nb +1 // on compte les elements identiques qui se suivent
else
if nb>1 then // si un element figute plus de une fois
Begin
Tot:=tot+nb; // on cumule les differents Nb
nb:=1 // on réinitialise Nb
End;
End;
if nb >1 then tot := tot + nb; // attention : on sort du tableau
e_ident:=tot
END;
```

v. **Résultats du jeu d'essai** :

```
veuillez entrez la taille de tableau:6
veuillez introduire les element de tableau:
T[1]= 1
T[2]= 1
T[3]= 3
T[4]= 2
T[5]= 2
T[6]= 1
Voici votre tableau: 1 | 1 | 3 | 2 | 2 | 1 |

le nombre d'element identique est: 5
```

5. Algorithme principal

a) Analyse :

- On aura besoin des modules suivants : Rand_mill, pliage, e_ident, transpotri
- On définit le type suivant : Tab1d qui est un tableau de taille maximale 20000 d'entiers
- Soient « an » l'année et « nbet » le nombre d'etudiant
- Initialiser 'i' à 0
- Faire varier ind de (an*100000+1) à (an*100000+nbet) et faire

  .Incrémenter i

  .On génère le tableau des adresse par la formule 1 (TR[i] = rand_mill (ind, 5))

  .On génère le tableau des adresses par la formule 2 (TP[i] = pliage (ind, 5, nbet))

- On trie les deux tableaux
- On affiche selon la méthode demandée tel que la petite adresse est la première case de chaque tableau et la grande adresse est la derniere case de chaque tableau.
- L'écart est : écart = TR[nbet] – TR[1]
- Le nombre de collision est généré par la fonction e_ident

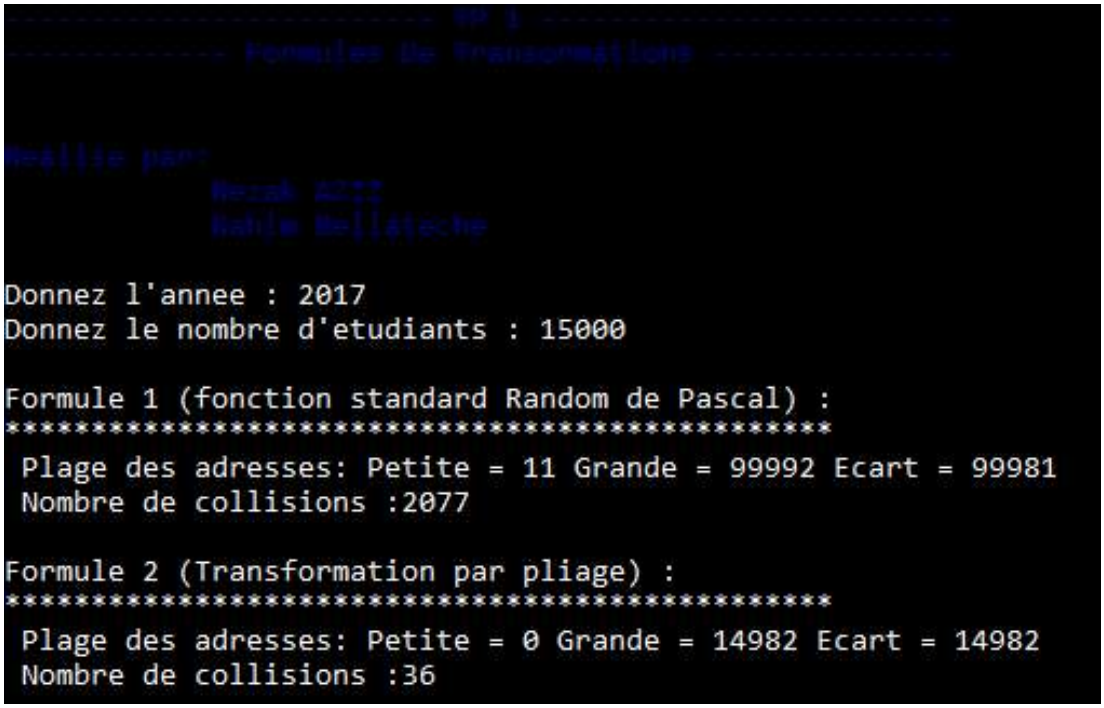
b) **Algorithme** :

```
Algorithme tp1_1718
Type Tab1d = tableau [1..20000] d'entier
Variables i, nbet, an, ind : entier ; TR, TP : tab1d
Fonctions Rand_mill, pliage, e_ident, transpotri
DEBUT
  Ecrire (' Donnez l'année : ')
  Lire (an)
  Ecrire ('Donnez le nombre d'étudiants : ')
  Lire (nbet)
  I ← 0
  Pour ind allant de an*100000+1 à an*100000+nbet faire
     DP
       I ← i+1
       TR[i] ← rand_mill(ind, 5)
       TP[i] ← pliage (ind, 5, nbet)
     FP
  Transpotri(TR, nbet)
  Transpotri (TP, nbet )
  Ecrire ('Formule 1 (fonction standard random de pascal) : ')
  Ecrire ('**************************************')
  Ecrire (' Plage des adresses : petite = ', TR[1], ' Grande =' , TR [nbet,), 'Ecart =', TR[nbet]-TR[1])
  Ecrire (' Nombre de collisions : ' e_ident(TR, nbet) )
  Ecrire ('Formule 2 (Transformation par pliage) : ')
  Ecrire ('**************************************')
  Ecrire (' Plage des adresses : petite = ', TP[1], ' Grande =' , TP [nbet,), 'Ecart =', TP[nbet]-TP[1])
  Ecrire (' Nombre de collisions :' , e_ident (TP, nbet) )
FIN
```

c) Programme

```
program tp1_1718;
uses crt;
type Tab =array[1..20000] of longint;
Var i,Nbet,an,ind :longint;
        Tr,TP:tab;
{$i C:\ALSDS_TP\module\randMil.fon}
{$i C:\ALSDS_TP\module\pliage.fon}
{$i C:\ALSDS_TP\module\E_Ident.fon}
{$i C:\ALSDS_TP\module\Tribulle.pro}
BEGIN
textcolor(1);
writeln('------------------------- TP 1 ------------------------');
writeln('------------- Formules De Transormations --------------');
Write (' Donnez l''annee : ');
Readln(an);
Write ('donnez le nombre d''etudiants : ');
readln(nbet);
writeln;
i:=0;
for ind := an*100000+1 to an*100000 + nbet do
Begin
i:=i+1;
TR[i]:=rand_Mil(ind,5); // on genere le tableau des adresses formule 1
TP[i]:=pliage(ind,5,nbet); // on genere le tableau des adresses formule 2
End;
Tribulle(TR,nbet); // on trie les 2 tableaux et on affiche les resultats
tribulle(TP,nbet);
Writeln('Formule 1 (fonction standard Random de Pascal) :');
Writeln('************************************************');
Writeln(' Plage des adresses: Petite = ',TR[1],' Grande = ',TR[nbet],' Ecart = ',Tr[nbet]-Tr[1]);
Writeln(' Nombre de collisions :',e_ident(TR,Nbet));
writeln;
Writeln('Formule 2 (Transformation par pliage) :');
Writeln('************************************************');
Writeln(' Plage des adresses: Petite = ',TP[1],' Grande = ',TP[nbet],' Ecart = ',TP[nbet]-TP[1]);
Writeln(' Nombre de collisions :',e_ident(TP,Nbet));
readln;
END.
```

d) Résultat du jeu d'essai :



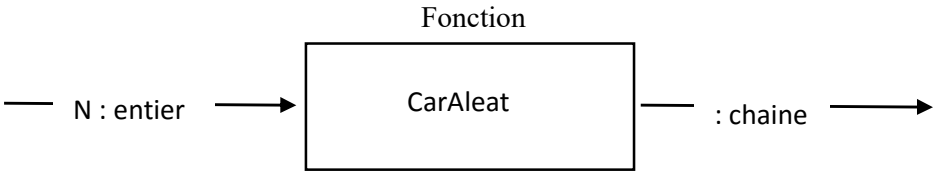
TP N° 2

INITIATION A LA CRYPTOGRAPHIE.

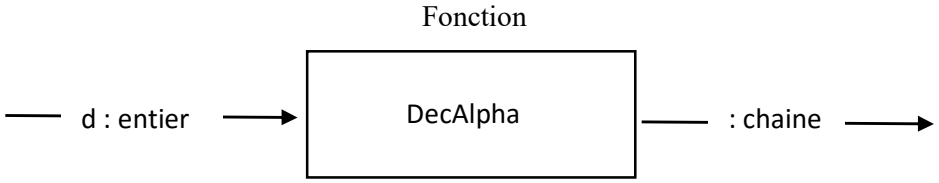
I. Découpage modulaire :

On aura besoin des modules suivants :

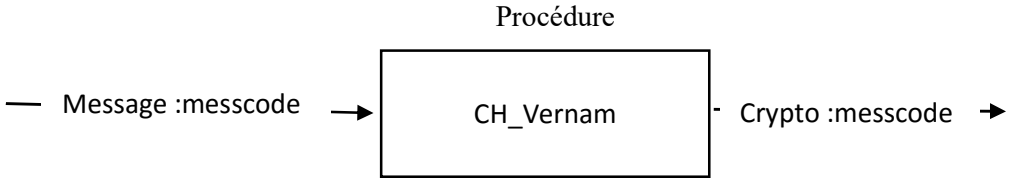
- CarAleat : donne une chaine de caractères contenant n caractères alphabétiques choisis aléatoirement, c'est-à-dire la clé.
- DecAlpha : effectue un décalage circulaire de d positions sur les lettres de l'alphabet
- CH_Vernam : chiffre le message selon le chiffre de Vernam.
- DECH_Vernam : déchiffre le message selon le chiffre de Vernam.
- Permutchar : permuter entre deux chaines de caractères.



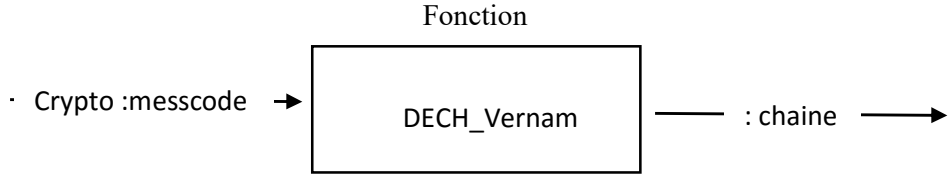
Rôle : donne une chaine de caractères contenant n caractères alphabétiques choisis aléatoirement.



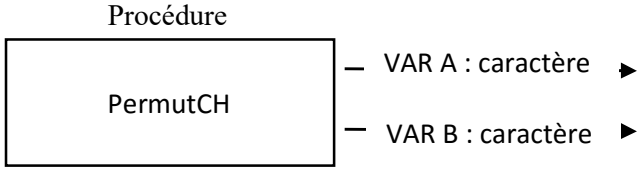
Rôle : effectue un décalage circulaire de d positions sur les lettres de l'alphabet.



Rôle : chiffre le message selon le chiffre de Vernam. En entrée message est accompagné du nombre de décalages circulaires et de la clé. En sortie crypto est accompagné du décalage et de la clé.



Rôle : déchiffre le message selon le chiffre de Vernam. Message est accompagné de la clé et du décalage.



Rôle : permuter entre A et B qui sont des caractères.

Type messcode = enregistrement

```
Ch1 : chaine
Decal : entier
Ch2 : entier
Fin
```

II. Conception et réalisation

1. Fonction CarAleat

i. Analyse :

Pour pouvoir construire une chaîne de caractères contenant n caractères alphabétiques choisis aléatoirement, à on suit ceci :

1. On aura besoin des fonctions standards de pascal, à savoir « random » et « CHR »
2. On fait varier i de 1 à n et à chaque fois on génère aléatoirement un nombre entre 65 et 91 et on associe ce dernier qui représente le code ASCII des lettres à la fonction CHR, et ce pour générer un caractère.
3. On concatène à chaque fois le caractère obtenu dans la chaîne S.
4. Affecter S à CarAleat.

ii. Algorithme :

```
Fonction CarAleat (n :entier) : chaîne
Variables s : chaîne ; i :entier
DEBUT
 S← ``
 Pour i allant de 1 à n faire
   S← S + chr (random(26)+65)
 CarAleat← S
FIN
```

iii. Jeu d'essai :

n	i	S	CarAleat
3	1	O	
	2	OP	
	3	OPS	OPS

iv. Programme

```
function caraleat(n:integer):string;
var s:string; i:integer;
begin
        s:= '';
        for i:=1 to n do
         s:= s + chr (random(26)+65);
         caraleat:= s;
end;
```

v. Résultats du jeu d'essai :

```
n=3
OPS
```

2. Fonction DecAlpha

i. Analyse

- On considère Alpha une chaîne de caractères qui contient la liste de l'alphabet.
- Pour effectuer un décalage de d positions sur les lettres, on doit déplacer les d positions finales et les déplacer au début de la chaîne, pour cela, on utilisera la fonction COPY.

ii. Algorithme

```
Fonction decalpha (decalage : entier ) : chaîne
 Variables alpha : chaîne[26]
DEBUT
 | Alpha ← 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
 | DecAlpha ← copy(alpha,26-decalage,decalage)+alpha
FIN
```

iii. Jeu d'essai

Alpha	d	DecAlpha
ABCDEFGHIJKLMNOPQRSTUVWXYZ	3	XYZABCDEFGHIJKLMNOPQRSTUVW
ABCDEFGHIJKLMNOPQRSTUVWXYZ	8	STUVWXYZABCDEFGHIJKLMNOPQR

iv. Programme

```
function DecAlpha (decalage:integer):string;
Var alpha,alphabet:string;
Begin
        alpha:= 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
        decalpha:= copy(alpha,26-decalage+1,decalage)+copy(alpha,1,26-decalage);
end;
```

v. Résultat du jeu d'essai

```
decalage=3
XYZABCDEFGHIJKLMNOPQRSTUVW
```

3. Procédure CH_Vernam

i. Analyse

Pour pouvoir chiffrer le message selon le chiffre de Vernam, on suit ceci :

- On effectue message.decal décalages à l'alphabet
- On met c='' , celui-ci va contenir le cryptogramme
- On prend un à un les caractères de notre message en clair, et on varie i de 1 à longueur(message.ch1)
  - Si la lettre message.ch1[i] est différente du blanc
    - On calcule V1
      V1=position de la lettre dans l'alphabet – 1
    - On calcule V2 de la même manière mais dans message.ch2
    - On calcule p=(A+B) modulo 26
    - On construit le cryptogramme. C=C+alphabet[p+1]
  - Si la lettre message.ch1[i] est égale à un blanc, on rajoute un blanc à c
- On met le cryptogramme dans crypto.ch1
- On met le décalage dans crypto.decal
- On met la clé dans crypto.ch2

ii. Algorithme

```
Procédure CH_Vernam (message :messcod ; var crypto :messcod)
Variables alphabet : chaîne [26]
          C :chaîne
          i, A, B, p : entier
fonction DecAlpha
DEBUT
  Alphabet ← DecAlpha (message.decal)
  C← ''
  Pour i allant de 1 à lenght(message.ch1) faire
     DPour
        Si message.ch1[i]<>' ' alors
             Dsi
               A← pos(message.ch1[i],alphabet)-1
               B← pos(message.ch2[i],alphabet)-1
               P← (A+B) mod 26
               C← c+alphabet [p+1]
             Fsi
        Sinon c← c + ' '
      FPour
 Avec crypto faire
  begin
  | Crypto.ch1 ← c
  | Crypto.decal ← message.decal
  | Crypto.ch2 ← message.ch2
  fin
FIN
```

iii. Jeu d'essai

Message.decal	Message.ch1	Message.ch2	i	A	B	p	c
4	TRI BULLE	AST XRBEX	1	23	4	1	X
			2	21	22	17	XN
			3	12	23	9	XNF
			4				XNF
			5	5	1	6	XNF C
			6	24	21	19	XNF CP
			7	15	5	21	XNF CPQ
			8	15	8	23	XNF CPQT
			9	8	1	9	XNF CPQTF

iv. Programme

```
procedure CH_Vernam (message:messcod;var crypto:messcod);
var alphabet : string[26]; c:string; i,A,B,p : integer;
{$i C:\ALSDS_TP\module\decalpha.fon}
begin
        alphabet:= decalpha(message.decal);
        c:= '' ;
        for i:= 1 to length(message.ch1) do
         begin
                if message.ch1[i]<>' ' then
                 begin
                           A:= pos(message.ch1[i],alphabet)-1;
                           B:= pos(message.ch2[i],alphabet)-1;
                           p:= (v1+v2) mod 26;
                           c:= c+ alphabet[p+1];
                 end
                else c:= c +  ' ' ;
          end;
       with crypto do
       begin
           crypto.ch1:= c;
           crypto.decal:= crypto.decal
           cypto.ch2:= message.ch2
       end;
end;
```

v. Résultat du jeu d'essai

```
le message en clair est :  TRI BULLE
la cle est : AST XRBEX
le cryptogramme est :  XNF CPQTF
```

4. DECH_Vernam

i. Analyse

- On effectue message.decal décalages à l'alphabet
- On met c= ' ' , c va contenir notre message déchiffré
- On prend un à un les caractères du cryptogramme, on varie i de 1 à length(message.ch1)
  - Si la lettre message.ch1[i] est différente de blanc
    - On calcule v ; v=position de la lettre dans l'alphabet – 1
    - On calcule v2 de la même manière mais dans message.ch2
    - On compare v1 et v2
    - Si v1<v2 , on ajoute 26 à p, puis on rajoute la (p+1)ème lettre de l'alphabet à c
    - Sinon , on rajoute à la (p+1)ème lettre de l'aplhabet à c
  - Si la lettre message.ch1[i] est égale à un blanc , on rajoute un blanc à c
- On affecte c à DECH_Vernam

ii. Algorithme

```
Fonction DECH_Vernam (crypto :messcod) :chaîne
Variables alphabet : chaine [26]
          C :chaîne
          i, A, B, p :entier
fonction DecAlpha
DEBUT
 Alphabet := DecAlpha (crypto.decal)
 C← ' '
 Pour i allant de 1 à length (crypto.ch1) faire
 DPour
  Si message.ch1[i]<>' ' alors
  Dsi
    A← pos(crypto.ch1[i], alphabet) – 1
    B ← pos(crypto.ch2[i], alphabet) – 1
    Si A<B alors c← c + alphabet [A-B+26+1]
    Sinon c ← c+ alphabet [A-B+1]
  Fsi
  Sinon c ← c + ' '
 FPour
 DECH_Vernam ← c
FIN
```

iii. Jeu d'essai

Crypto.decal	Crypto.ch1	Crypto.ch2	i	A	B	C	DECH_Vernam
4	XNF CPQTF	AST XRBEX	1	1	4	T	
			2	17	22	TR	
			3	9	23	TRI	
			4			TRI	
			5	6	1	TRI B	
			6	19	21	TRI BU	
			7	20	5	TRI BUL	
			8	23	8	TRI BULL	
			9	9	1	TRI BULLE	**TRI BULLE**

iv. Programme

```
function DECH_Vernam (crypto:messcod):string;
var alphabet : string[26]; c:string; i,B,A,p : integer;
{$i C:\Users\LENOVO\Desktop\decalpha.fon}
begin
        alphabet:= decalpha(crypto.decal);
        c:= '' ;
        for i:= 1 to length(crypto.ch1) do
         begin
                if crypto.ch1[i]<>' ' then
                 begin
                          A:= pos(crypto.ch1[i],alphabet)-1;
                          B:= pos(crypto.ch2[i],alphabet)-1;
                          if A<B then c:= c + alphabet [A-B+26+1]
                          else c:= c + alphabet [A-B+1]
           end
           else c:= c +  ' ' ;
       end;
      DECH_Vernam:= c;
   end;
```

v. Résultat du jeu d'essai

```
Le cryptogramme est :  XNF CPQTF
La clé est :  AST XRBEX
Le message déchiffré est :  TRI BULLE
```

5. Algorithme principal :

On définit la structure de données suivante :

Type Messcod = ENREGISTREMENT

Ch1 : chaîne
Decal : entier
Ch2 : chaîne
Fin

a) Analyse :

❖ Partie 1

- On lit le message en clair
- On lit le décalage
- On calcule la clé en utilisant la fonction CarAleat
- On chiffre le message pour obtenir le cryptogramme en utilisant la procédure CH_Vernam

❖ Partie 2

- On lit le cryptogramme
- On lit le décalage
- On lit la clé
- On génère le message en clair en utilisant la fonction DECH_Vernam

b) Algorithme

```
Algorithme tp2_1718
Type messcod =  Enregistrement
                    | Ch1 : chaîne
                    | Decal : entier
                    | Ch2 : chaîne
                  FIN
Variables message, crypto : messcod
Fonctions CarAleat, DECH_Vernam
Procédure CH_Vernam
DEBUT
 (********** Partie 1 *********)
 Ecrire (‘Donnez le message en clair : ‘)
 Lire (message.ch1)
 Ecrire (‘Donnez le décalage : ‘)
 Lire (message.decal)
 Message.ch2← CarAleat ( length(message.ch1) )
 Ecrire (‘ La clé est : ‘ , message.ch2)
 CH_Vernam (message, crypto)
 Ecrire (‘ Le cryptogramme est : ‘ , crypto.ch1)
 (********** Partie 2 *********)
 Ecrire (‘Donnez le message à décoder : ‘)
 Lire (crypto.ch1)
 Ecrire (‘Donnez le décalage :’)
 Lire (crypto.decal)
 Ecrire (‘Donnez la clé : ‘)
 Lire (crypto.ch2)
 Ecrire (‘ Le message déchiffré est :’ , DECH_Vernam (crypto) )
FIN
```

c) Programme

```
program tp2;
type messcod = record
                 ch1 : string ;
                 decal : integer;
                 ch2 : string;
               end;
var message, crypto : messcod;
{$i H:\TP2_1718\CarAleat.fon}
{$i H:\TP2_1718\DECH_Vernam.fon}
{$i H:\TP2_1718\DecAlpha.fon}
{$i H:\TP2_1718\CH_Vernam.pro}
BEGIN
        (******** Partie 1 ********)
        write('Donnez le message en clair : ' );
        readln(message.ch1);
        write('Donnez le décalage : ') ;
        readln(message.decal);
        message.ch2 := CarAleat ( length(message.ch1));
        writeln('La clé est : ' , message.ch2  );
        CH_Vernam(message, crypto);
        writeln('Le cryptogramme est : ' , crypto.ch1);
        (********** Partie 2 ********)
        writeln;
        write('Donnez le message à décoder : ');
        readln(crypto.ch1);
        write('Donnez le décalage : ');
        readln(crypto.decal);
        write('Donnez la clé : ');
        readln(crypto.ch2);
        writeln('Le message déchiffré est : ' , DECH_Vernam (crypto)) ;
        readln;
end.
```

d) Résultats du jeu d’essai

```
Donnez le message en clair : TRI BULLE
Donnez le décalage : 4
La clé est : OPSVPWOWLQ
Le cryptogramme est : LKE UUDLT

Donnez le message à décoder : LKE UUDLT
Donnez le décalage : 4
Donnez la clé : OPSVPWOWLQ
Le message déchiffré est : TRI BULLE
```

Conclusion

Au final, après le traitement des deux problèmes qui sont le choix d'une formule de transformation et le chiffrage et le déchiffrage des données, on déduit que le fait d'utiliser la modularité était primordiale, voire indispensable dans la résolution de ces problèmes.

En effet, nous avons réussi à transformer les problèmes qui étaient particulièrement longs et complexes en sous-problèmes courts et moins complexes.

Ces deux TPs ont englobé la plupart des notions vues en ALSDS durant le premier semestre, à savoir la manipulation des entiers, les tableaux, les chaînes de caractères et les enregistrements.

En conclusion, le travail en équipe était une expérience très enrichissante car il nous a permis de travailler d'une manière plus intelligente et efficace, il nous a permis également de partager nos connaissances, de confronter nos idées et de les corriger, cela nous a poussé à résoudre ces deux problèmes et par la suite aboutir aux résultats attendus.

« Se réunir est un début, rester ensemble est un progrès, travailler est la réussite »

Henry Ford, industriel et fondateur de FORD