Le XOR Cipher

Table des matières

Après une présentation générale du projet et la description du travail à réaliser, vous trouverez la partie intitulée "Cahier des charges et spécification" qui précise exactement ce qu'il faut programmer. Prenez le temps de bien lire cette partie.

1 Le chiffrage XOR

Le chiffrage XOR est une méthode de **chiffrage symétrique**, également dite à clef secrète. Cette méthode repose sur l'utilisation d'un algorithme (cipher) et d'une clef. Alors que la clef doit rester secrète, l'algorithme peut être divulgué sans compromettre l'efficacité du chiffrage. L'émetteur et le destinataire du message sont seuls à connaître la clef qui, transmise à l'algorithme de chiffrage (XOR cipher), permet de chiffrer et de déchiffrer un message. La sécurité du chiffrage repose seulement sur le grand nombre de clefs possibles et sur la difficulté à mener une attaque exhaustive en un temps assez court. Cette sécurité calculatoire est directement liée à l'évolution de la puissance des calculateurs : L'algorithme DES (Data Encryption Standard) utilisé dans le système de mots de passe UNIX dans les années 1980 est basé sur une

clef de 56 bits ; aujourd'hui, on peut réaliser l'examen exhaustif de ces 2^{56} clefs en un temps raisonnable.

- **L'algorithme XOR cipher :** Chaque caractère du message et de la clef est considéré en binaire sur 8 bits, le codage s'effectue alors en appliquant la fonction XOR bit par bit.
  Comme la clef est en général plus courte que le message, il est nécessaire de la répéter autant de fois que nécessaire.
  Par exemple, si on veut coder le message `"Il va neiger avant la fin du mois de Décembre."` avec la clef `Azerty`, on réécrit autant que nécessaire la clef :

  ```
  Il va neiger avant la fin du mois de Décembre.
  AzertyAzertyAzertyAzertyAzertyAzertyAzertyAzer
  ```

  Le message codé sera composé des caractères : 'I' xor 'A', 'l' xor 'z', ' ' xor 'e', ...,'.' xor 'r'

  La fonction "OU exclusif" encore appelée xor est un opérateur logique de l'algèbre de Boole $\{0, 1\}$ muni des deux lois de compositions internes $+$ et $\times$. Cet opérateur binaire est souvent noté $\oplus$, il est défini par sa table de vérité :

a	b	$a \oplus b$	$a \oplus a$	$(a \oplus b) \oplus b$
0	0	0	0	0
0	1	1	0	0
1	0	1	0	1
1	1	0	0	1

  Cet opérateur est symétrique : $a \oplus b = b \oplus a$ et associatif $(a \oplus b) \oplus c = a \oplus (b \oplus c)$.
  La table ci-dessus prouve également la propriété :

  $$((a \oplus b) \oplus b) = a$$

  Cette propriété explique pourquoi, avec cette méthode de chiffrage, le codage et le décodage d'un message sont deux opérations totalement identiques : Un caractère du message a est codé en $a \oplus b$, en notant b le caractère de la clé associé. Si on fait agir à nouveau b par $\oplus$ on obtient $(a \oplus b) \oplus b$ qui est égal à a le caractère initial du message.

- **La clef :** une longueur de clef importante rend le cassage du code plus difficile si on utilise la méthode statistique détaillée section suivante. Dans le cas extrême, si la longueur de la clef est égale à la longueur du message, la méthode statistique est inopérante. Ainsi, avant l'avènement de machines de calcul rapides, un tel chiffrage était réputé incassable (Claude Shannon 1946). Ce système dit de "masque jetable" était utilisé entre la Maison Blanche et le Kremlin pendant la guerre froide.

2 Cassage du code

Le processus par lequel on tente de décoder un message est appelé une attaque. Le chiffrage et le déchiffrage d'un fichier peut se faire sans se préoccuper de la nature des octets. Par contre, pour casser un chiffre nous devons connaître la langue dans laquelle le message a été écrit, ainsi que le codage informatique de ses caractères.

Le codage informatique utilisé sur la plupart des ordinateurs récents est le UTF-8. C'est un codage qui utilise entre un et quatre octets pour coder un caractère et qui est prévu pour représenter tous les caractères des langues de la planète. Il reprend le codage ASCII sur sept bits mais pour les caractères européens (et donc les caractères accentués du français), il utilise deux octets dont le premier (C3 en hexadécimal) sert de balise. Pour simplifier nous considérons que le texte en français est codé selon la norme Latin-1 sur un octet et non en UTF-8. Il faudra donc convertir ou enregistrer les textes à cette norme, ce qui est prévu dans la plupart des éditeurs de texte.
Nous présentons dans cette sections trois types d'attaque. Ces attaques partent du principe que le message chiffré est un texte écrit en français (selon la norme Latin-1).

2.1 Validité des caractères du message à décrypter

L'ensemble des lettres du français, codées donc en Latin-1, est représenté par un sous ensemble des octets codés sur 8 bits. Une première attaque qui permet de trouver un ensemble de clés valides, consiste à vérifier que les caractères décodés par une clé candidate sont valides en français (toujours selon la norme Latin-1).

2.2 Recherche exhaustive et analyse fréquentielle

L'analyse fréquentielle examine les répétitions des lettres du message chiffré afin de trouver la clef. Elle est principalement utilisée contre les chiffrements mono-alphabétiques (substitution d'une lettre par une autre) qui présentent un biais statistique.
L'analyse fréquentielle est basée sur le fait que, dans chaque langue, certaines lettres ou combinaisons de lettres, apparaissent avec une certaine fréquence. Par exemple, en français, le e est la lettre la plus utilisée, suivie du a et du s. Inversement, le w est peu usité. Le tableau théorique des fréquences (exprimées en %) d'apparitions des lettres en français $Freq\_th$ est le suivant :

A	B	C	D	E	F	G	H	I	J	K	L	M
$9,42$	$1,02$	$2,64$	$3,39$	$15,87$	$0,95$	$1,04$	$0,77$	$8,41$	$0,89$	$0,00$	$5,34$	$3,24$
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
$7,15$	$5,14$	$2,86$	$1,06$	$6,46$	$7,90$	$7,26$	$6,24$	$2,15$	$0,00$	$0,30$	$0,24$	$0,32$

Dans ce tableau, les diverses formes des lettres sont prises en compte : par exemple, pour la lettre 'E' les formes 'e','E','é','è' et 'ê' sont prises en compte :.

Avec chaque clef candidate, on décode le cryptogramme et on calcule par comptage les diverses fréquences d'apparitions des lettres du texte décodé avec la clef. On obtient alors un tableau de fréquences noté $Freq$. La meilleure clef est celle pour laquelle les fréquences mesurées sont les plus proches des fréquences théoriques : $Freq\_th$. Cette notion de proximité et donc de distance entre deux tableaux sera mesurée par la fonction d définie par :

$$d(Freq\_th, Freq) = \sum_{i=1}^{26} (Freq\_th[i] - Freq[i])^2$$

Cette répartition des fréquences des lettres n'est qu'approximative, cela dépend de nombreux paramètres tels que le niveau de langue du texte, ainsi que du style d'écriture. Une deuxième

condition nécessaire pour appliquer cette technique est la longueur du cryptogramme. En effet, un texte trop court ne reflète pas obligatoirement la répartition générale des fréquences des lettres.
Quelquefois, on peut constater que la clef sélectionnée par la méthode précédente n'est pas la bonne. Cela provient du manque de représentativité du texte : trop petite taille, style trop particulier. On s'oriente alors vers une méthode utilisant un dictionnaire.

2.3 Validation par un dictionnaire

Cette méthode consiste à chercher la clef pour laquelle le texte décodé contient exclusivement des mots présents dans un dictionnaire. Cela suppose qu'on dispose d'un dictionnaire contenant toutes les formes nominales et verbales ; c'est le cas du dictionnaire du scrabble qui vous est fourni. Si on suppose le dictionnaire complet et le message écrit sans faute, la bonne clef est celle pour laquelle tous les mots sont dans le dictionnaire. Eu égard à la longueur de cette recherche on limitera cette vérification aux mots courts : 2,3 voire 4 lettres au maximum.

3 Travail à réaliser

Les différentes fonctionnalité décrites sont organisées sous formes de "modules" :

1. Module chiffrage/déchiffrage avec clef (CD)
2. Module cassage divisé lui-même en :
   (a) Test de validité des caractères (C1)
   (b) Test statistique (C2)
   (c) Validation par dictionnaire pour des clefs de longueur k (C3.1)
   (d) Validation par dictionnaire pour des clefs de longueur $k \in [3, 7]$ (C3.2)
3. Module *main* (Main)

Chaque module rapporte un certain nombre de points (s'il fonctionne bien sûr).

Vous devez choisir un ensemble de modules à programmer. Les modules ne vous seront crédités que s'ils fonctionnent et respectent le cahier des charges. Le plus simple est de commencer par le chiffrage/déchiffrage et d'ajouter graduellement les modules de cassage.

Critères de notation :
- le module fonctionne correctement en respectant le cahier des charges : 50% des points
- le code est clair et bien structuré 25% des points
- le code est commenté avec des commentaires utiles 25% des points

Le détail de chaque module est décrit dans la partie "Cahier des charges et spécifications".

Modules	Fonction	Complet	Minimum
CD	chiffrage déchiffrage xor d'un fichier quelconque avec clé décryptage sur texte français Latin-1 :	3	3
C1	validité	2	2
C2	fréquence	3	
C3.1	dico long=k	3	
C3.2	dico long≤k	4	
main	intégration des modules + balises+fichiers	4	4
Makefile	gestion de la compilation séparée	1	1
Total programmation (coeff. 1/2)		***20***	***10***
Bonus éventuel (ex : shell de test, github, rapidité, ...)		1	

Documents	Points (coeff. 1)
conception	3
Bilan	3
validation	3
fiche av.	1
Total doc.	***10***

Points pour le code	
ça fonctionne	50,00 %
bon code	25,00 %
commenté et ś	25,00 %

Note totale = note programmation + note documents

FIGURE 1 – Résumé de l'ensemble du projet

4 Cahier des charges et spécifications

Le programme sera écrit en **langage C**. Il comporte deux volets :

1. **Utilisation normale :** connaissant la clef, le programme permet de chiffrer et déchiffrer un texte. Dans cette phase la norme de codage et la nature des octets est sans intérêt. On doit pouvoir chiffrer et déchiffrer n'importe quel type de fichier : texte Latin-1, UTF-8, image JPG, fichier de son, etc. Le mieux est d'utiliser le type *unsigned char*, ou encore déclarer le type : **typedef unsigned char byte ;** si vous trouvez cela plus parlant.
2. **Cassage du code :** disposant d'un texte chiffré, le programme doit retrouver la clef à partir de laquelle le texte a été chiffré. Cette partie repose sur une recherche exhaustive ("brute force") en explorant complètement l'ensemble des clefs possibles : (suites finies de caractères imprimables issus du code Latin-1). On se placera successivement dans les deux situations suivantes :
   (a) Dans un premier temps, la longueur de la clef est supposée connue.
   (b) Ensuite on supposera seulement que la longueur de la clef est dans l'intervalle $[3, 7]$.

   Pour valider une clef, on utilisera les trois critères suivants :
   - **C1** : Validité des caractères : Sachant que le message initial est un texte en français, les données décryptées devront donc appartenir à l'ensemble des caractères valides pour un texte. Ce premier critère permet d'éliminer une bonne proportion des clefs candidates.

- **C2** : Analyse fréquentielle : Avec chaque clef satisfaisant à C1, on décode le cryptogramme et on détermine, pour le texte "décodé" la tableau des fréquences. Le calcul de la distance entre ce tableau de fréquences mesurées et le tableau de fréquences théoriques permet de classer les différentes clefs ; la meilleure clef est celle qui minimise cette distance.
- **C3** : Dictionnaire : Parmi les clefs retenues après C1, on cherche celle qui donne le maximum de mots du dictionnaire fourni.

4.1 Spécifications détaillée

- **Caractères utilisés** La clef et le message à coder sont composés seulement de certains caractères issus du code ISO/IEC 8859-1 ou Latin-1, qui rappelons-le, contient l'ASCII. La ligne de commande du terminal est en UTF-8 et donc pour que la clef reste compatible avec la norme Latin-1 elle ne devra utiliser que des caractères dont le code est le même dans toutes les normes. Le tableau ci-dessous précise quels sont les caractères autorisés dans la clef ou dans le message à décrypter ainsi que leurs codes ASCII/Latin-1 :

Caractères autorisés	**Code ASCII/Latin-1**	**où ?**
alphabet majuscule	65..90	clef et message
alphabet minuscule	97..122	clef et message
espace	32 ( )	seulement pour les messages
ponctuation	33(!),34("),39(’),40((),41()), 59(;)	seulement pour les messages
ponctuation	44(,),45(-),46(.),58(:), 63(?),95(_),123({),125(})	clef et message
nombres	48..57	seulement pour les clefs
lettres accentuées	A : 192, 194, 196, 224, 226, 228 C : 199, 131 E : 200, 201, 202, 203, 232, 233, 234 235 I : 206, 207, 238, 239 U : 217, 219, 249, 251 O : 212, 214, 244, 246	seulement pour les messages

Pour plus de détails, voir :
`https://en.wikipedia.org/wiki/ISO/IEC_8859-1`

Pour chiffrer et déchiffrer un message avec une clef, il n'y a pas de restriction sur les caractères ou norme du message, seulement pour la clef. Pour le décryptage par contre, on part du principe que le message est à la norme Latin-1. De plus, pour simplifier le traitement, de nombreux caractères ne sont autorisés ni dans le message ni dans la clef.

Précisions pour la clef : Elle est passée en argument de la ligne de commande qui est gérée par le Bash en UTF-8, en conséquence, on a pris soin d'interdire tout caractère interférant avec le bash :

· les caractères de séparation de commandes : espace (32), tabulation (9)
· les caractères de fin de commandes : point-virgule (59) ou retour chariot (13).
· les caractères accentués : une lettre accentuée aurait un code différent en UTF-8

de celui de la même lettre issue d'un fichier géré en Latin-1.
En clair, on n'utilisera dans la clé que les lettres de la langue anglaise, la ponctuation et les chiffres (Cf. le tableau précédent qui sera votre référence en cas de doute).

Précisions pour le message : Les chiffres sont interdits pour faciliter la méthode de validation par dictionnaire.

- **Critère C1 :** On ne validera pas d'une manière globale chaque clef candidate, ce serait trop long. On validera l'un après l'autre chaque caractère valide des clefs. Précisément si la clef a pour longueur 3, le caractère clef[0] sert à coder les caractères $1, 4, ..., 1+3k, ...$ du message. On détermine pour commencer les valeurs possibles de clef[0] puis, de manière tout à fait indépendante, on déterminera les valeurs possibles de clef[1] et de clef[2]. Par exemple, si on obtient :
  · $clef[0] \in ['e','C',':']$
  · $clef[1] \in ['A']$
  · $clef[2] \in ['v','d']$
  · alors on aura aisément la liste des $3 * 1 * 2 = 6$ clefs satisfaisant à C1 :

$$clef \in [\text{"}eAv\text{"},\text{"}eAd\text{"},\text{"}CAv\text{"},\text{"}CAd\text{"},\text{" : }Av\text{"},\text{" : }Ad\text{"}]$$

  Les caractères valides pour la clef peuvent être stockés dans un tableau statique ou dynamique à deux dimensions *clef[tailleClef][maxCaracteres]* ou *tailleClef* est la taille de la clef, et *maxCaracteres* le nombre de caractères possible pour un caractères de la clef.
- **Critère C2 :** On cherche la clef avec le meilleur score. Pour cela, à partir des clefs valides pour C1, on "décode" les textes et on calcule les fréquences . On retient la clef avec les caractères offrant le meilleur score en suivant l'analyse fréquentielle. Si plusieurs clés obtiennent le même score, on les classe avec un critère arbitraire (ordre lexicographique par exemple).
- **Critère C3 :** On cherche là aussi la clef avec le meilleur score. A partir des clefs valides pour C1 on décode les textes et on cherche chaque mot du texte "décodé" dans le dictionnaire fourni. Ce dictionnaire contient les accents à la norme Latin-1 sans aucune majuscule. On compte le nombre de mots valides suivant le dictionnaire et on choisit la clef donnant le score maximum.

  Nota bene : On suppose que seule la première lettre d'un mot du message peut éventuellement être une majuscule.

- **Compilation :** Votre programme doit compiler en tapant : `make` dans un terminal ouvert dans le répertoire des fichiers sources et doit produire un exécutable nommé : `xorcipher`

- **Exécution :** Le fichier exécutable sera appelé en ligne de commande avec les arguments nécessaires :
  — pour le chiffrage/déchiffrage : il faut fournir (dans un ordre quelconque)
    — après la balise `-i` : le nom du fichier texte contenant un message à coder
    — après la balise `-o` : le nom du fichier texte contenant un message codé
    — après la balise `-k` : la clef
    Par exemple :

`./xorcipher -i inTextFile -o outTextFile -k clef`

— pour le cassage : il faut fournir (dans un ordre quelconque)
  — après la balise `-i` : le nom du fichier texte contenant un message codé
  — après la balise `-m` : le mode (critère) utilisé pour trouver la clef, la sortie attendu dépends du mode, voir plus bas.
  — après la balise optionnelle `-l` : la longueur de la clef, si elle n'est pas fournie, on cherchera toutes les clefs de longueur entre 3 et 7.

  Par exemple :
  `./xorcipher -i inTextFile -m mode [-l longClef]`
  Les crochets signalant un paramètre optionnel.

Dans le cas du chiffrage/déchiffrage, le programme n'affichera rien : mais il traitera le message contenu dans le fichier *inTextFile* et écrira le résultat dans le fichier *outTextFile*. Ainsi si *inTextFile* contient un fichier en clair alors le résultat dans *outTextFile* contiendra le message codé (chiffrage) et inversement si *inTextFile* contient un fichier déjà codé, alors le résultat dans *outTextFile* contiendra le message en clair (déchiffrage).
En revanche pour le crack, on affichera directement la meilleure clef (sur une seule ligne).

Les "-i" pour input, "-o" pour output, "-k" pour clef, "-l" pour longueur de clef, "-m" pour le mode, sont là pour faciliter la tâche de celui qui lance l'exécution. Grâce à ces balises, il n'a pas à mémoriser l'ordre dans lequel il saisit les arguments : "-i" précède le nom du fichier d'entrée, "-k" précède la clef... Ainsi les deux appels :
· `./xorcipher -k KeyKey8 -i message1.txt -o sortie.txt`
· `./xorcipher -o sortie.txt -k KeyKey8 -i message1.txt`
sont des appels semblables.

Tout appel invalide affichera un message d'aide :
— Erreur Fichier : signifie que le fichier n'existe pas ou ne peut pas être ouvert
— Erreur Clef : signifie qu'il manque la clef ou elle contient des caractères invalides
— Erreur Commande : dans le cas d'erreur de syntaxe (balises inexistantes, mal placées...)

Après chiffrage d'un fichier, on obtient un fichier contenant des octets quelconques. Comme on a choisi d'utiliser la même ligne de commande pour crypter ou décrypter un fichier, on peut faire un contrôle du programme en chiffrant un fichier, puis en déchiffrant le fichier obtenu. On doit bien sûr obtenir un fichier déchiffré en tout point identique au fichier d'origine.

Dans le cas du cassage, on suppose que le texte d'origine était composés de caractères valides de la norme Latin-1.

Pour le critère C1, l'affichage sera l'affichage de la liste de chaque caractère valide pour la clef, entouré de '[' et ']'. En reprenant l'exemple donné précédemment l'affichage sera : `[eC:][A][vd]`.
S'il n'y a pas de clef valide, rien n'est affiché. Si la longueur de la clef recherchée n'est pas spécifiée, une liste est affichée par ligne et par longueur croissante des listes.

Pour le critère C2 et C3, uniquement la meilleure clef est affichée sur une seule ligne. Si la longueur de la clef recherchée n'est pas spécifiée, la meilleure clef valide pour chaque longueur

est affichée sur une seule ligne, par longueur croissante, et sans ligne vide.

Pour chaque action : chiffrage, déchiffrage, cassage avec les critères $C1$, $C2$, $C3.1$ ou $C3.2$, on affichera les temps d'exécution.

4.2 Exemples

- chiffrage d'un message :
  `./xorcipher -i petit.txt -o petit_crypt1.txt -k abc`
  `./xorcipher -i petit.txt -o petit_crypt2.txt -k exupery`
  `./xorcipher -i difficile.txt -o difficile_crypt.txt -k aab33c44`
- Cassage d'un cryptogramme :
  petit_crypt1.txt avec clef de longueur 3 :
  `./xorcipher -i petit_crypt1.txt -l 3` $--> $ abc

  petit_crypt1.txt avec clef de longueur 6 :
  `./xorcipher -i petit_crypt1.txt -l 6` $---> $ abcabc

  petit_crypt2.txt avec clef de longueur inconnue :
  `./xorcipher -i petit_crypt2.txt` $---> $ exupery

  difficile_crypt.txt avec clef de longueur inconnue : le critère C2 seul ne permet pas de décoder le message (meilleure clef aKb33c4.) , mais avec C3, on a bien :
  `./xorcipher -i difficile_crypt.txt` $--> $ aab33c44

- Messages d'erreur :
  On distinguera 3 types d'erreur :
  Syntaxe de la ligne de commande (nombre d'arguments, erreurs de balise,...) :
  `./xorcipher` $--> $ ERROR : command line
  `./xorcipher -i petit.txt -o crypt.txt -a exa` $--> $ ERROR : command line
  Fichiers inexistants : `./xorcipher -i cropt34.txt` $--> $ ERROR : file
  Clef interdite :
  `./xorcipher -i crypt.txt -o crypt4.txt -k La[clef` $--> $ ERROR : key

4.3 Codage en C

4.3.1 Typage des données, spécifications

Avant de vous lancer dans le codage, faites valider votre analyse par votre enseignant.

4.3.2 Opérateurs bit à bit

Le langage C propose des opérateurs bit à bit n'opérant que sur des valeurs de type entier ou caractère : ils ne peuvent pas être utilisés sur des flottants !
En particulier, il existe un opérateur XOR bit à bit noté `^` :

char a='A' ; -> $(65)_{10}$=$(41)_{16}$=$(01000001)_2$
char b='B' ; -> $(66)_{10}$=$(42)_{16}$=$(01000010)_2$

$(00000011)_2$ -> a^b=(char) 3 (caractère non imprimable)

Cet opérateur fonctionne aussi pour les types *unsigned char* ou *unsigned int*, donc pour le type *byte* défini plus haut.

4.3.3 La page de code Latin-1

Pour le cassage du code, l'utilisation d'un codage en UTF8 n'a pas été retenue car le codage muti-octets (1 à 4) ne facilite pas l'utilisation de XOR. Si l'affichage de caractères de la page Latin-1 est (en général...) bien acceptée par la plupart des éditeurs il n'en est pas de même de l'affichage dans une fenêtre de résultats via printf. (En particulier, problème de configuration des machines dans les salles de TP).
Certaines fonctions ne sont pas opérantes avec les codes ASCII étendus, il s'agit de *isalpha, isalphanum...* On testera donc la validité des caractères (pour les messages et pour la clef) directement en observant leur code ASCII étendu (codage Latin-1 sur 8 bits) en les considérant comme des entiers sur 8 bit.

Exemple :

```
 if ((c == 192) || (c == 194)) ...
// test des valeurs Latin-1 correspondant aux variantes de "A".
```

4.3.4 Arguments en ligne de commande

Lors de l'éxécution en ligne de commande, il est possible de passer des paramètres au programme, le nombre de paramètres sur la ligne (nom de la commande inclus) est stocké dans l'argument argc de la fonction main, et la liste des paramètres est stockée dans le tableau de chaînes de caractères argv[] de la fonction main dont l'entête sera :

$$\text{int main(int argc, char} * \text{argv[])}$$

Par exemple :

- Codage de message1.txt avec la clef "KeyKey8" :
  `./xorcipher -k KeyKey8 -o sortie.txt -i message1.txt`
  argc=7

i	$argv[i]$
0	$./xorcipher$
1	$-k$
2	$KeyKey8$
3	$-o$
4	$sortie.txt$
5	$-i$
6	$message1.txt$

- Cassage statistique avec une clef de longueur 5 :
  `./xorcipher -i essai.txt -l 5`

argc=5

i	$argv[i]$
0	$./xorcipher$
1	$-i$
2	$essai.txt$
3	$-l$
4	5

4.4 Utilitaires

4.4.1 La commande diff

Sous linux, elle permet d'afficher les éventuelles différences entre deux fichiers : diff fichier1 fichier2. Si les fichiers sont identiques, elle ne renvoie rien.

4.4.2 Détermination de la fin d'un fichier

Pour détecter la fin du fichier à traiter, vous pouvez utiliser la fonction feof() ou la ligne

while (fscanf(f_in,"%c",&wc) !=EOF){ ...}

Attention : c'est de *fscanf* qui renvoie le EOF. Il ne faut pas tester wc. Cette méthode ne fonctionne pas avec getc ou getchar.

4.4.3 Sauver des fichiers texte à la norme Latin-1

Avant de sauvegarder votre fichier à chiffrer dans votre éditeur de texte, choisissez l'encodage en Latin-1 dans le menu Préférences...