DEVOIR TM SCHEME : Cryptage

à rendre en séances de TM du **16 au 19 avril 2013**

Cryptage

On s'intéresse au cryptage (ou cryptage) et décryptage de textes. Nous nous limitons dans ces exercices à essentiellement deux techniques simples dans lesquelles tout caractère d'un texte est transformé de la même manière quelle que soit sa position. Il existe d'autres techniques de cryptage qui se proposent en effet de transformer les textes en jouant sur l'ordre des caractères, ou encore qui cryptent le texte par groupes de caractères, diminuant ainsi les chances de découvrir le système de cryptage appliqué.

De plus nous nous intéressons au cryptage paramétré par une " clé ". Une clé est un mot qui permet de générer un code à partir d'une technique de cryptage. La même technique de cryptage peut générer plusieurs codes différents à partir de clés différentes. Pour pouvoir ensuite décrypter un texte, il suffit de connaître la technique et la clé utilisés lors de son cryptage.

On peut utiliser pour les exercices ci-dessous les fonctions sur les chaînes de caractères définies précédemment (voir TM n°2) dans le fichier chaines.ss. On rappelle que l'instruction `load` de Scheme permet d'utiliser ces opérations sans en recopier la définition dans le fichier contenant le programme de cryptage. Alternativement, on peut directement utiliser les fonctions standard de Scheme pour la manipulation de chaînes.

1 Cryptage par décalage d'alphabet

La première technique de cryptage que nous souhaitons réaliser est basée sur un décalage de l'alphabet. La clé n'est déterminante à la génération du code que par sa longueur (son nombre de caractères). Cette longueur est utilisée pour introduire un décalage dans l'alphabet (si la longueur de la clé est supérieure à la longueur de l'alphabet, le décalage est effectué modulo la longueur de l'alphabet). Par exemple, si la clé est `"clef"`, l'alphabet `"abcdefghijklmnopqrstuvwxyz"` est transformé en `"wxyzabcdefghijklmnopqrstuv"`, la longueur de `"clef"` étant `4`. Avec ce code, le caractère `#\a` est remplacé par `#\w`, `#\b` par `#\x`, `#\c` par `#\y`, `#\d` par `#\z`, `#\e` par `#\a`, et ainsi de suite jusqu'à `#\z` qui est remplacé par `#\v`. En d'autres termes, un caractère est remplacé par le caractère de même rang de l'alphabet transformé. On montre ci-dessous comment cette technique est appliquée sur un autre exemple.

```
Clé : "bonjour" (longueur 7)
Alphabet :              "abcdefghijklmnopqrstuvwxyz"
                           ↓ ↓   ↓   ↓ ↓   ↓↓↓
Alphabet transformé :   "tuvwxyzabcdefghijklmnopqrs"

Texte non crypté :      "coucou, c'est moi !"
                         ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
Texte crypté :          "vhnvhn, v'xlm fhb !"
```

Les caractères du texte à crypter non présents dans l'alphabet sont laissés inchangés par le processus de cryptage. Tout autre clé de longueur `7` (`"oiseaux"`, par exemple) produit le même alphabet transformé et donc le même résultat sur le cryptage du texte. Le décryptage peut s'effectuer simplement en effectuant la transformation inverse.

Pour réaliser ce type de cryptage, on peut en premier lieu écrire les deux fonctions suivantes.

- Une fonction `decalDroite` admettant pour paramètres une chaîne de caractères et un entier positif ou nul, et renvoyant une chaîne de caractères décalée à droite de la valeur de l'entier. Par exemple, `(decalDroite "abcdef" 2)` renvoie `"efabcd"`, et `(decalDroite "abcdef" 9)` renvoie `"defabc"`.
- Une fonction `carDeMemeRang` admettant pour paramètres un caractère et deux chaînes de caractères de même longueur, et recherchant ce caractère dans la première chaîne. S'il est trouvé, le caractère de même rang de la deuxième chaîne est renvoyé en résultat ; si le caractère n'est pas trouvé, il est renvoyé en résultat de la fonction. Par exemple, `(carDeMemeRang #\r "bernard" "pauline")` renvoie `#\u` (`#\r` est trouvé en 3ème place dans `"bernard"` et `#\u` est le 3ème caractère de `"pauline"`), et `(carDeMemeRang #\i "bernard" "pauline")` renvoie `#\i` (`#\i` n'étant pas trouvé dans `"bernard"`).

2 Cryptage par permutation d'alphabet

La deuxième technique de cryptage que nous considérons est basée sur une permutation de l'alphabet. Un code est généré par cette technique en modifiant l'ordre des caractères dans l'alphabet de telle sorte que les caractères de la clé se retrouvent en tête de celui-ci. Par exemple, si la clé est `"clef"`, l'alphabet `"abcdefghijklmnopqrstuvwxyz"` est transformé en `"clefabdghijkmnopqrstuvwxyz"`. Le cryptage s'effectue ensuite de la même manière que pour le cryptage par décalage à partir de l'alphabet et de l'alphabet transformé, comme le montre l'autre exemple ci-dessous.

```
Clé : "bonjour" ("bonjur" est retenu)
Alphabet :              "abcdefghijklmnopqrstuvwxyz"
                           ↓ ↓   ↓   ↓ ↓   ↓↓↓
Alphabet transformé :   "bonjuracdefghiklmpqstvwxyz"

Texte non crypté :      "coucou, c'est moi !"
                         ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
Texte crypté :          "nktnkt, n'uqs hkd !"
```

On peut remarquer que les occurrences multiples de caractères dans la clé sont ignorées afin de conserver des alphabets de même longueur. Ainsi on retient `"bonjur"` pour la clé (`#\o` apparaissant 2 fois dans `"bonjour"`, seule sa première occurrence est conservée), et d'autres clés peuvent éventuellement générer le même code (`"bonjourno"` entre autres). Le décryptage peut s'effectuer simplement en effectuant la transformation inverse.

Pour réaliser ce type de cryptage, on peut commencer par écrire les fonctions suivantes.

- Une fonction `appartient?` admettant pour paramètres un caractère et une chaîne de caractère et renvoyant `#t` si et seulement si le caractère apparaît dans la chaîne, `#f` sinon. Par exemple, `(appartient? #\r "bernard")` renvoie `#t`, et `(appartient? #\r "pauline")` renvoie `#f`.
- Une fonction `supprDoublons` prenant en paramètre une chaîne de caractères et renvoyant une chaîne équivalente dans laquelle les occurrences multiples de caractères ont été éliminées en ne gardant que la première occurrence de chaque caractère. Par exemple `(supprDoublons "beurrier")` renvoie `"beuri"`.
- Une fonction `groupeEnTete` admettant pour paramètres deux chaînes de caractères et renvoyant une chaîne de caractères, sans occurrences multiples, correspondant à l'ensemble des caractères présents dans les deux chaînes donnés dans leur ordre d'apparition dans la première chaîne, puis la deuxième chaîne. Par exemple, `(groupeEnTete "bernard" "pauline")` renvoie `"bernadpuli"`.

3 Programme de cryptage

Ce qui diffère entre les deux techniques de cryptage présentées ci-dessus, c'est uniquement la manière de calculer l'alphabet transformé pour un code, d'après la clé. On peut même envisager une troisième technique de calcul de l'alphabet transformé combinant ces deux premières techniques : l'alphabet subit à la fois une permutation puis un décalage et, par exemple, avec la clé `"clef"`, l'alphabet `"abcdefghijklmnopqrstuvwxyz"` devient `"wxyzclefabdghijkmnopqrstuv"`. Une fois l'alphabet transformé obtenu, le même processus de traduction d'un alphabet vers un autre peut être appliqué pour crypter ou décrypter un texte. On se propose d'écrire un programme permettant de crypter ou de décrypter un texte saisi au clavier, d'après le type de cryptage et la clé utilisés. Le dialogue entre l'utilisateur et ce programme doit avoir la forme suivante.

```
Souhaitez vous [C]rypter ou [D]écrypter un texte ? C
Tapez le texte à crypter et terminez par un $ :
On m'appelle le chevalier blanc, par monts et par vaux,
je vais en chantant ...$
Quelle est la clé de cryptage ? gerardlanvin
Quel type de cryptage (décalage, permutation, combiné) ? permutation
Résultat du cryptage :
kj h'gmmdffd fd rvdugfidp efgjr, mgp hkjsq ds mgp ugtx,
bd ugiq dj rvgjsgjs ...
```

On peut décomposer ce programme en fonctions, en y intégrant les fonctions détaillées dans les sections 1 et 2 et en y ajoutant les fonctions suivantes.

- Une fonction `lireTexte` ne prenant aucun paramètre et renvoyant une chaîne de caractères. Cette fonction doit permettre de saisir un texte se terminant par un caractère `#\$` au clavier. Cette fonction est

similaire à `lireChaine` de chaines.ss, à la différence que `lireChaine` ne permet pas de saisir des chaînes contenant des espaces ou encore des virgules.

- Une fonction `alphaTrans` prenant en paramètre un symbole dénotant un type de cryptage et une chaîne de caractères dénotant la clé de cryptage. Cette fonction renvoie l'alphabet transformé à partir de `"abcdefghijklmnopqrstuvwxyz"` pour la technique de cryptage et la clé spécifiée. Les valeurs devant être acceptées comme type de cryptage sont `'décalage`, `'permutation` et `'combiné`. Par exemple, `(alphaTrans 'combiné "clef")` renvoie `"wxyzclefabdghijkmnopqrstuv"`.
- Une fonction `traduit` prenant en paramètre 3 chaînes de caractères (un texte à traduire, un alphabet source et un alphabet cible), et renvoyant le texte traduit d'un alphabet à l'autre. Par exemple, `(traduit "nktnkt, n'uqs hkd !" "bonjuracdefghiklmpqstvwxyz" "abcdefghijklmnopqrstuvwxyz")` renvoie `"coucou, c'est moi !"`.
- Deux fonctions `crypte` et `decrypte`, admettant chacune en paramètres 2 chaînes de caractères (une clé et un texte) et un symbole (type de cryptage), et permettant respectivement de crypter ou décrypter puis renvoyer un texte.

Il vous est également demandé d'écrire un programme principal permettant le dialogue avec l'utilisateur illustré ci-dessus.

4 Travail à rendre la semaine du 15 avril 2013

Le travail doit être rendu au plus tard lors dans les séances de TM du 16 au 19 avril 2013. Il peut être effectué en binôme ou seul. Chaque étudiant ou binôme doit envoyer par email à betul.aydin@imag.fr un fichier, ayant pour nom le nom des étudiants (dupont_durand.ss par exemple). Une impression du contenu de ce fichier doit également être rendue et doit porter le nom des étudiants.

La présentation des programmes (spécifications et commentaires) doit être soignée, une partie de la note porte sur cette présentation. Il est bien sûr possible de définir d'autres fonctions que celles qui sont explicitement demandées dans les sections 1 à 3.

Indications supplémentaires :

- la fonction prédéfinie `read-char` permet de lire un seul caractère au clavier ;
- la fonction prédéfinie `char-downcase` permet de transformer un caractère en un caractère correspondant minuscule.