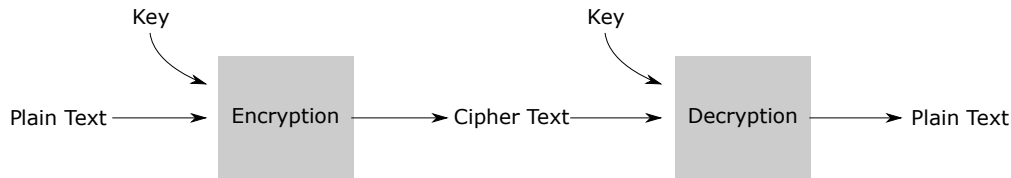


Exercice (ICC-C) (20 points) :

Le **chiffrement** est le processus d'encodage secret d'information : il consiste à convertir une représentation initiale, appelé **texte en clair**, ou **plaintext** en anglais, en une forme alternative chiffrée, appelée **texte chiffré**, ou **ciphertext** en anglais. Le **déchiffrement** est l'opposé du chiffrement ; c'est le processus par lequel on récupère l'information chiffrée. En d'autres termes, le déchiffrement reconvertit le **texte chiffré** en son **texte en clair** original. Les algorithmes de chiffrement utilisent généralement une clé de chiffrement, disponible uniquement aux parties autorisées.



Chiffrement et déchiffrement.

Dans ce devoir, nous vous demandons d'écrire un programme qui chiffrera une chaîne de caractères donnée en entrée. Par souci de simplicité, nous ne considérerons que deux algorithmes de chiffrement : l'algorithme par **décalage** (**shifting** en anglais) et l'algorithme par **permutation**, que nous présenterons en détail plus bas. Vous pourrez supposer que le texte en clair contient les 26 lettres de l'alphabet latin, en minuscule : a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y et z. Dans le tableau de code ASCII, ces lettres sont consécutives et leur code décimal correspondant va de 97 pour la lettre a à 122 pour la lettre z.

Votre programme sera appelé depuis le terminal de la façon suivante :

```
./program_name plaintext key
```

où **plaintext** est la chaîne de caractères à chiffrer, et **key** est un paramètre **optionnel**. Si **key** est fourni, le programme doit utiliser le chiffrement par décalage avec **key** comme clé de chiffrement. Si **key** n'est pas fourni, alors le programme doit utiliser le chiffrement par permutation.

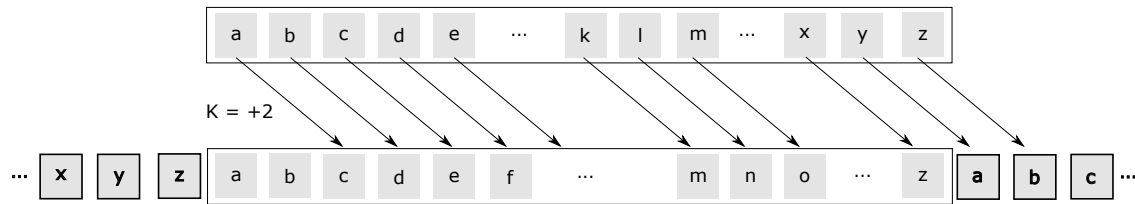
Votre programme doit afficher au terminal le **texte chiffré** résultant du chiffrement.

Algorithme de chiffrement par décalage

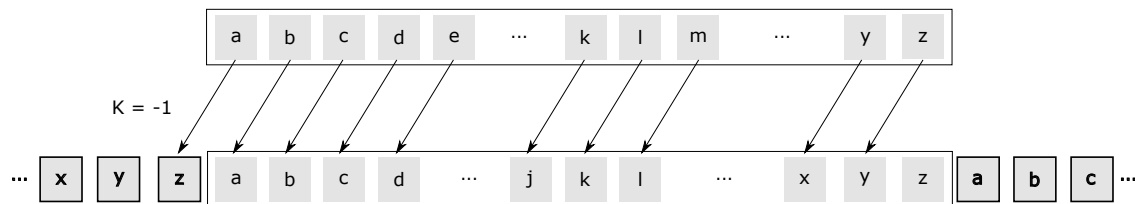
Le chiffrement par décalage nécessite en entrée une chaîne de caractères (le **texte en clair**) et une clé K , où K peut être **n'importe quel** entier positif ou négatif. Pour obtenir le texte chiffré, chaque lettre du texte en clair d'entrée est remplacé par la lettre se situant à une distance de K de celle-ci dans l'alphabet. Notez que cette distance peut être positive (pour K positif), ou négative (pour K négatif). Vous pouvez imaginer que l'alphabet latin est répété infiniment des deux cotés. Ainsi, un chiffrement avec K égal à 30 doit être exactement identique à un chiffrement avec K égal à 4. Pareillement, un chiffrement avec K égal à -30 doit être exactement identique à un chiffrement avec K égal à -4. Les Figures 2 et 3 donnent deux exemples de chiffrement, caractère par caractère, pour l'intégralité de l'alphabet latin. Dans ces exemples, les clés valent +2 et -1, respectivement.

Algorithme de chiffrement par permutation

Le chiffrement par permutation nécessite un tableau d'associations entre les caractères en clair et leur caractère chiffré correspondant. Chaque association est donc une paire de deux caractères : un caractère en clair et le caractère chiffré qui lui est associé. Le tableau d'associations doit satisfaire certaines propriétés : le nombre d'éléments qu'il contient doit être **exactement** le même que le

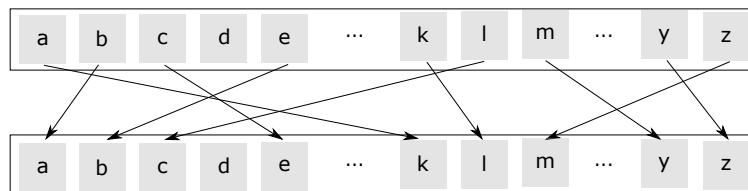


Chiffrement par décalage avec clé $K = +2$. La ligne du haut indique les lettres du texte en clair, et la ligne du bas indique les lettres du texte chiffré. Chiffrer la chaîne de caractères `iccexam` donnerait donc `keegzco`.



Chiffrement par décalage avec clé $K = -1$. La ligne du haut indique les lettres du texte en clair, et la ligne du bas indique les lettres du texte chiffré. Chiffrer la chaîne de caractères `iccexam` donnerait donc `hbdbwzl`.

nombre de lettres **distinctes** dans l'alphabet latin ; **aucun** caractère ne peut apparaître plus d'une fois comme caractère en clair (respectivement, chiffré) (autrement dit, il ne peut pas y avoir de doublon parmi les caractères en clair, ni parmi les caractères chiffrés) ; et les caractères en clair et leur caractère chiffré associé doivent être différents (donc une lettre ne peut pas être associée à elle-même). Le chiffrement par permutation est simple : chaque caractère de la chaîne d'entrée doit être remplacé par le caractère chiffré correspondant, spécifié dans le tableau d'associations.



Exemple de chiffrement par permutation.

Pour le reste de ce devoir, vous pouvez supposer les points suivants :

- Une structure `enc_map_char` contenant deux éléments de type `char` est définie pour vous. Le premier élément s'appelle `pt_char` (comme *plaintext character*, le caractère en clair), et le second s'appelle `ct_char` (comme *ciphertext character*, le caractère chiffré).
- Un tableau appelé `enc_map`, contenant 26 éléments de type `struct enc_map_char` est déclaré et initialisé au début de la fonction `main`. Ce tableau doit être utilisé par votre programme pour implémenter le chiffrement par permutation. Vous pouvez supposer que le contenu de ce tableau est valide, comme décrit ci-dessus.
- Le texte en clair, s'il est fourni, ne contient que des lettres minuscules du dictionnaire latin de 26 lettres.

Les tâches que vous devez effectuer pour ce devoir sont listées ci-dessous.

1. (4 points) Écrivez la fonction `enc_shifting_char` qui prend deux paramètres : un caractère `ch` et un entier `key`. La fonction effectue le chiffrement par décalage du caractère `ch` à l'aide de la clé `key`, et retourne le caractère chiffré résultant.

```
char enc_shifting_char(char ch, int key)
```

2. (4 points) Écrivez la fonction `enc_shifting_string`, qui prend trois paramètres : deux chaînes de caractères `pt` et `ct`, et un entier `key`. La fonction effectue le chiffrement par décalage de la chaîne `pt` à l'aide de la clé `key`. La chaîne de caractères chiffrée résultante doit être stockée dans l'argument `ct`, et retournée par la fonction. Dans le cas où une chaîne de caractères donnée en entrée est incorrecte (si elle est `NULL`), la fonction doit retourner le pointeur `NULL`.

```
char* enc_shifting_string(char *pt, char *ct, int key)
```

3. (5 points) Écrivez la fonction `sort_enc_map`, qui prend un tableau non-trié `enc_map` de `struct enc_map_chars`, et le trie dans l'ordre **croissant** par rapport aux caractères en clair `pt_char`. La fonction ne retourne rien. Votre solution n'a pas besoin d'être la plus efficace possible. Quelle est la complexité de votre solution ?

```
void sort_enc_map(struct enc_map_char * enc_map)
```

4. (2 points) Écrivez la fonction `enc_permutation_string`, qui prend trois paramètres : deux chaînes de caractères `pt` et `ct`, et le tableau d'associations **trié** `enc_map`. La fonction effectue le chiffrement par permutation de `pt`. Le texte chiffré résultant doit être stocké dans l'argument `ct`, et retourné par la fonction. Dans le cas où une chaîne de caractères donnée en entrée est incorrecte (si elle est `NULL`), la fonction doit retourner le pointeur `NULL`. Votre solution doit utiliser le fait que `enc_map` est trié pour être plus efficace.

```
char* enc_permutation_string(char *pt, char *ct, struct enc_map_char *enc_map)
```

5. (5 points) Complétez votre programme en écrivant la fonction `main`. Si le programme est appelé avec un nombre incorrect d'arguments, il doit afficher "Nombre d'arguments incorrect". Si la clé est zéro ou un multiple de 26, le programme doit afficher "Clé invalide". Si le chiffrement ou quoique ce soit d'autre échoue, le programme doit afficher "Chiffrement échoué". Pour le reste de la description des fonctionnalités du programme et des paramètres qu'il doit supporter, référez-vous à la section d'introduction du devoir.

Note : vous devriez utiliser l'allocation dynamique de mémoire pour créer la chaîne de caractères chiffrée. Faites attention aux fuites de mémoire !

```
int main(int argc, char *argv[])
```
