université PARIS DIDEROT PARIS 7

EXAMEN | Programmation C et Système | 3/1/2017 9^{30}–11^{30}

Les documents sont autorisés.
La copie ou les brouillons du voisin ne sont pas des supports autorisés, ni les dispositifs électroniques.
Éteignez impérativement vos mobiles.

1 Exercice

Question 1
Écrire une commande qui peut être invoquée *via* une commande comme `otp` *key input output* et qui permet de chiffrer le contenu du fichier de nom correspondant à *input* dans le fichier *output*.
Le chiffrement consiste simplement à extraire chaque octet du fichier d'origine et à lui appliquer un ou-exclusif (xor - opérateur `^`) avec le caractère correspondant de la clé *key*, si la clé est de longueur l, son premier caractère servira à chiffrer les octets 0, l, $2l$, etc, le second sert à chiffrer les octets 1, $l+1$, $2l+1$, etc, le troisième 2, $l+2$, $2l+2$, etc.
Ainsi si le fichier contient les octets 0x00, 0x02, 0x77, 0xFF, 0xAA et que la clé est "abc" (donc Ascii 0x61 0x62 0x63) le résultat sera le résultat des opérations 0x00^0x61, 0x02^0x62, 0x77^0x63, 0xFF^0x61, 0xAA^0x62.
On veillera à capter tous les cas d'erreurs raisonnables...

Question 2
Comment pourrait-on accélérer le chiffrement ? (Piste : s'affranchir de la lecture et du traitement caractère par caractère). Décrire le mécanisme envisagé sans chercher à l'implémenter, mais en prenant soin d'étudier tous les cas «problématiques».

2 Exercice

Il n'est pas toujours possible d'envoyer de trop gros fichier par mail, une technique consiste alors à découper le fichier original en plus petits morceaux. On cherche donc à écrire des commandes permettant de découper puis reconstituer un fichier.

Question 1
Écrire une commande `split` *t*`o|k|m` *fichier* permettant de découper le *fichier* en morceaux de taille t (les unités pour t pouvant être `o` pour octets, `k` pour kibi-octet 1024 octets, ou `m` pour mébi-octet 2^{20} octets). Le découpage permet alors d'obtenir des fichiers (autant que nécessaire) dont les noms seront de la forme *fichier*+*suffixe*, où le `suffixe` est nécessairement de longueur constante (il faut donc déterminer cette longueur) et dont les valeurs permettent d'ordonner les morceaux par ordre lexicographique (les suffixes seront des suites de lettres minuscules).
Ainsi le découpage d'un fichier de nom `toto` et de longueur 1000 octets en morceaux de longueur 100 octets devrait produire les fichiers `toto+a`, `toto+b`, ... `toto+j`. Pour un fichier de nom `titi` et de longueur 1000 octets, son découpage en morceaux de longueur 10 octets devrait produire les fichiers `titi+aa`, `titi+ab`, ..., `titi+az`, `titi+ba`, `titi+bb`,...
On notera que le dernier morceau peut être plus court que les autres...

Question 2 Piste : `opendir..., qsort`.
Écrire une commande `unsplit` *fichier* qui permet de reconstituer le *fichier* en recherchant dans le répertoire courant les morceaux s'ils existent.
Attention, il peut être nécessaire d'utiliser un tri.

3 Exercice

On cherche à écrire de quoi de lire un fichier de données textuelles selon un format spécifié (en ligne de commande). Chaque ligne du fichier d'entrée sera une suite de données textuelles chacune pouvant représenter soit une chaîne de caractères (sans espace) (type spécifié `s`), soit un nombre entier (type spécifié `i`), soit un nombre flottant de grande précision (type spécifié `d`). De sorte qu'une ligne de commande comme
monprogramme fichier `sids`
puisse permettre de lire un *fichier* qui pourrait contenir les données suivantes (les séparateurs sont de simples espaces) :

```
television 23 3.4 pois
armoire 999 3.2 pneu
```

Dans la suite, il n'est pas demandé d'écrire la fonction `main`, seulement les fonctions permettant de coder/décoder.

Question 1 (version simplifiée) (Piste : `*scanf`, `*open`, `malloc`...)
Écrire une fonction `int parse_sis(const char *nom, struct sis *tab, int max);` permettant de lire un fichier de `nom` spécifié et selon le format (fixé) `sis` (c'est-à-dire des lignes de la forme *chaine entier chaine*). Les données de chaque ligne lue devront être placées dans une structure C de la forme :

```
struct sis {
    char *s1;
    int v;
    char *s2;
};
```

Et l'ensemble des données devra constituer le tableau `tab` de ces structures et qui a la taille maximale `max` indiquée. Le retour de la fonction sera le nombre d'éléments lus (au plus `max`) et -1 en cas d'erreur.

Question 2 (dynamique) (Piste : `realloc`)
Écrire une fonction `int parse_sis_dynamic(const char *nom, struct sis **tab);` qui fonctionne comme la précédente mais produit un tableau de taille exactement égale au nombre d'éléments lus. En cas d'erreur, cette fonction ne doit pas provoquer de fuite mémoire.

Question 3 (générique) (Piste : les casts...)
Pour pouvoir lire un fichier de format quelconque il est nécessaire d'être entièrement dynamique, c'est-à-dire qu'on ne peut plus fixer la structure. Afin de contenir les données, on devra donc utiliser l'allocation dynamique pour tous les types. Les données d'une ligne du fichier seront donc stockées *via* un tableau de pointeurs de type `void *`. Chaque pointeur devant pointer vers une zone mémoire permettant de stocker un élément du type considéré.
Écrire une fonction `void parse_line_generic(const char *ligne, const char *format, void **data);` permettant de décoder une `ligne` du fichier lue selon le `format` spécifié en stockant les données appropriées dans un tableau de pointeurs adéquats (`data`).

Question 4 (générique suite...)
Écrire une fonction `int parse_generic(const char *nom, const char *format, void ***data);` permettant de lire le contenu du fichier `nom` selon le `format` spécifié et stockant ce qui est lu dans le tableau `data`. La valeur de retour sera le nombre d'éléments lus et -1 en cas d'erreur. On veillera à ce qu'en cas d'erreur l'appel ne provoque pas de fuite mémoire.

Question 5 (générique ter...)
Écrire une fonction `void print(void ***data, int ndata, const char *format, const char *ordre);` permettant d'afficher à l'écran les `ndata` données stockées dans `data` et qui sont au `format` indiqué, le tout dans l'`ordre` spécifié. L'`ordre` est une chaîne de caractères constituée uniquement de chiffres décimaux (ex. : `"15324"` ou `"321"`) et permet d'indiquer dans quel ordre les champs doivent être affichés. Par exemple, s'il y a deux champs `"si"` (donc d'abord une chaîne suivie d'un entier), l'ordre `"21"` permettra d'afficher d'abord l'entier et ensuite la chaîne (en les séparant par un simple espace).