Programmation C
TP n° 9 : Manipulation de fichiers

Exercice 1 : Manipulation de fichiers

1. Pour la manipulation de fichiers, on va se servir des fonctions `fopen`, `fclose`, `fputs` et `fgets` de la librairie `stdio.h`. Écrivez un programme `fill.c` qui attend en argument un nom de fichier et un chiffre entier `n`, crée le fichier correspondant et le remplit de `n` lignes avec un message choisi par vous même sur chaque ligne. (pensez à utiliser `atoi`). Que se passe-t-il si vous appelez deux fois votre programme avec le même nom de fichier ?
2. Comment modifier votre programme pour que si le fichier donné en argument existe déjà, les lignes soient ajoutées à la fin ?
3. Dans un ficher `fillib.c`, écrivez une fonction `int copie(FILE *fsrc, FILE *fdst)` qui prend deux flots et copie le contenu du premier dans le deuxième. Utilisez les fonctions `fputc` et `fgetc`. On suppose que le premier flot contient du texte lorsqu'il est repositionné au début du fichier. La fonction renverra 0 en cas de succès et -1 en cas d'échec.

   Les deux commandes suivantes utiliseront la fonction `copie`. Il faudra donc recopier le code de cette fonction dans les fichiers faisant appel à elle.
4. Créez un programme `mycp.c` qui prend en argument deux noms de fichiers et copie le premier dans le deuxième. Vérifiez que `./mycp fichierin fichierout` se comporte comme `cp`. Testez votre programme sur des fichiers textes et des fichiers binaires, e.g., un exécutable. Comparez la taille du fichier original avec la taille de la copie.
5. Créez un programme `mycat.c` qui prend en argument un nombre arbitraire de noms de fichier et affiche le contenu des fichiers correspondants sur la sortie standard (l'un après l'autre, sans séparateur). Si aucun argument n'est donné, le programme affiche le contenu de `stdin`. (Pensez à utiliser votre fonction `copie`.) Vérifiez que `./mycat` se comporte comme `cat` (sans options).
6. Ajoutez à votre fichier `fillib.c`, une fonction `int copie_par_lignes(FILE *fsrc, FILE *fds, int lig, int len)` qui prend deux flots, un nombre de lignes et une longueur et copie les `lig` premières lignes du premier flot dans le deuxième flot. La fonction retourne 0 si la fin du fichier est atteinte et 1 sinon. On supposera que le premier flot contient du texte et qu'une ligne ne contient pas plus de `len` caractères. Il faudra vérifier (avec `assert`) que les lignes ne dépassent pas cette longueur. Utilisez les fonctions `fputs` et `fgets`.
7. Créez un programme `myless.c` qui prend en argument un nombre arbitraire de noms de fichier et affiche le contenu des fichiers correspondants sur la sortie standard (l'un après l'autre, sans séparateur). Chaque fichier sera affiché `k` lignes par `k` lignes, l'utilisateur tapant sur la touche *Entrée* lorsqu'il veut afficher les `k` lignes suivantes. `k` a pour valeur le nombre de lignes du terminal dans lequel est exécuté la commande `./myless`. Le nombre de lignes du terminal est donné par la variable shell `LINES`. Afin de définir une variable d'environnement `LIG` du processus exécutant `./myless` qui prend la valeur de la variable `LINES`, tapez la commande suivante :

   ```
   > LIG=$LINES ./myless arg_1 ... arg_n
   ```

   De plus chaque ligne ne devra pas dépasser `LEN` caractères où `LEN` sera une variable également passée au processus exécutant `./myless`. Par exemple si les lignes ne doivent pas dépasser 80 caractères, on peut modifier la commande précédente de la façon suivante :

   ```
   > LIG=$LINES LEN=80 ./myless arg_1 ... arg_n
   ```

   Si `LEN` n'est pas définie, la valeur par défaut de la taille maximale d'une ligne sera 100 et si `LIG` n'est pas définie, on fera un affichage 10 lignes par 10 lignes. Et pensez à utiliser la fonction `getenv` ensuite.

Exercice 2 : Vers un éditeur de texte simplissime

Le but de cet exercice est de coder les fonctionalités les plus basiques d'un éditeur de texte : splitting de lignes, fusion de lignes et bufferisation.

1. Ecrivez la fonction `int longueur(char *name)` qui prend un nom de fichier en paramètre et retourne la longueur du fichier. Vous pourrez utiliser la fonction `fgetc`.
2. Implémentez la fonction `int longueur_max_lignes(char *name)` qui retourne la longueur maximale des lignes du fichier dont le nom est passé en paramètre. Vous pourrez à nouveau utiliser la fonction `fgetc`.
3. Ecrivez la fonction `document *decouper_en_lignes(char *name)` qui doit lire le fichier `name` et retourner un `document` tel que :
   - `len` est la longueur du fichier,
   - `txt` est un pointeur vers un tampon qui contient `len+1` caractères, les caractères du fichier suivis du caractère de fin de chaînes de caractères,
   - `nbl` indique le nombre d'éléments dans le vecteur pointé par `lignes`,
   - `lignes` est vecteur qui contient les pointeurs vers le début de chaque ligne de `txt`. Il faudra allouer la place pour `max_ligne(name)+1` caractères pour chaque buffer de ligne.

```
1            typedef struct{
2              unsigned len;
3              char *txt;
4              unsigned nbl;
5              char **lignes;
6            } document;
7
```

4. Implémentez une fonction `void afficher(document *doc)` permettant d'afficher les caractéristiques d'un document, c'est à dire sa longueur, son nombre de lignes et les lignes du document.
5. Implémentez une fonction `void sauvegarder(document *doc, char *name)` permettant de sauvegarder le document dans le fichier.
6. Implémentez la fonction `int couper(document *doc, unsigned i, size_t k)` qui sépare la ligne `i` en deux lignes à la position `k`. La fonction retourne `-1` si la ligne n'existe pas ou si la ligne contient moins de `k` caractères, sinon la fonction retourne l'indice de la nouvelle ligne créée.
7. Implémentez la fonction `int fusionner(document *doc, int i)`, qui supprime `\n` à la fin de la ligne `i`, ce qui fait que la ligne `i` colle avec la ligne suivante. Pensez à mettre à jour les pointeurs dans le vecteur `lignes`. La fonction retourne le nouveau nombre de lignes, `-1` si la ligne demandée n'existe pas.
8. Le site Projet Gutenberg [1] propose une vaste collection d'eBooks librement accessibles. Récupérez un fichier de taille env. 200kB au format Plain Text UTF-8 pour tester votre programme. Par exemple, les anciens terminaux étaient souvent limités à 80 caractères par ligne, vous pouvez proposer un programme qui convertit l'eBook de votre choix en un eBook où chaque ligne contient au plus 80 caractères. Fusionnez les lignes contigües tant que possible sans dépasser la limite de 80 caractères, ceci permettant un affichage plus compact. En vous aidant d'une fonction `afficher(document *doc, unsigned l, unsigned k)` qui affiche les lignes d'indice compris entre l et k, testez vos fonctions en affichant les lignes autour des lignes modifiées et comparer (visuellement) les résultas.

1. `https://www.gutenberg.org`