Programmation C
TP n° 10 : Fichiers

Exercice 1 : Lecture et écriture

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.
2. Modifier le programme pour que si le fichier donné en argument existe déjà, les lignes soient ajoutées à la fin.
3. Écrire un programme `fillb.c` qui attend un nom de fichier `fic` en argument et qui copie le flux standard du clavier dans le fichier. On utilisera la combinaison de touches CTRL+D (CTRL+Z sous Windows) pour terminer la saisie.
4. Écrire un programme `get_line.c` qui attend un entier n et un nom de fichier en argument et affiche la ligne numéro n du fichier sur la sortie standard. Il faut : trouver la position du début de la ligne ; mesurer sa taille ; allouer un tampon suffisant ; re-positionner le curseur et utiliser `fgets` pour lire la ligne en une fois. Si la ligne n'existe pas, le programme affiche un message d'erreur. Utiliser les fonctions `fseek` et `ftell` de la librairie `stdio.h`.
5. Écrire un programme `somme.c` qui attend un nom de fichier en argument. On supposera que le fichier ne contient que des entiers séparés par des caractères d'espacement (retour à la ligne, espace, etc...). Le programme doit calculer la somme de ces entiers et écrire le résultat dans un fichier `resultat`. Voici un exemple d'exécution :

```
$> cat data
1␣2
3␣
␣4
5
$> ./somme data ; cat resultat
15
```

Exercice 2 : Manipulation de fichiers

1. Écrire une fonction `int nblines(FILE *fic)` qui renvoie le nombre de lignes dans le fichier donné en argument (sachant qu'une ligne finit par le caractère '\n').
2. Écrire un programme `mycp.c` qui attend deux noms de fichier en argument, `fic1` et `fic2`, et recopie le contenu de `fic1` dans `fic2` en écrasant le contenu original de `fic2`, s'il y en avait un. Si `fic2` n'existe pas il doit être créé. Vérifiez que `./mycp` se comporte comme `cp` (sans options, avec deux noms de fichiers en argument). Tester votre programme sur des fichiers texte et des fichiers binaires, e.g., un executable. Comparer la taille du fichier original avec la taille de la copie.
3. Écrire un programme `mycp2.c` qui se comporte comme `mycp` et qui calcule la taille du fichier grâce à `fseek`, alloue un tampon de cette taille et réalise une seule lecture et une seul écriture pour faire la copie.

4. Écrire un programme `mycat.c` qui attend un nom de fichier et affiche le contenu en numérotant les lignes. Vérifier que le résultat est le même qu'en utilisant la commande `cat -n`. Les lignes sont comptées à partir de 1.
5. Modifiez votre programme `mycat.c` pour que si on lui donne l'option `-b`, il se comporte comme la commande `cat -b` (il n'affiche pas les numéros des lignes ne contenant aucun caractère).
6. Pour que l'affichage soit un peu plus joli, on voudrait aligner les numéros de ligne ; c'est-à-dire :

```
1 Debut du fichier
...
1374 Fin du fichier
```
⟶
```
1    Debut du fichier
...
1374 Fin du fichier
```

   On pourra avoir recours aux modificateur de format `%*d` de la fonction `printf` : `printf("%*d ",longueur,entier)`, où l'entier sera complété avec des espaces jusqu'à avoir longueur `*` spécifiée en argument.
7. Écrire un programme `mygrep.c` qui attend une chaîne `c` de caractères et nom de fichier et affiche sur la sortie standard toutes les lignes qui contiennent au moins une fois la chaîne `c`.

Exercice 3 : Manipulation de fichiers bonus

1. On suppose qu'un fichier manipulé a été créé par un musicien : il contient des barres de reprise. Quand une ligne se termine par `:||`, vous devez revenir en arrière pour reprendre la lecture au dernier `||:` rencontré (attention, chaque barre n'agit qu'une fois !). On suppose que le fichier contient une occurrence du `||:` avant `:||`. Ecrire un programme `mycat3.c` qui lit un fichier de musique. Par exemple :

```
Une ligne
||: Une autre ligne
et une troisieme:||
```
⟶
```
1 Une ligne
2 ||: Une autre ligne
3 et une troisieme :||
4 ||:Une autre ligne
5 et une troisieme :||
```