Projet programmation système W02

Introduction

Cette semaine, nous souhaitons vous faire compléter un programme par vous-mêmes : une version simple de « wordcount » (compteur de mots).
Vous aurez à rendre le code correspondant (délais : dimanche 11 mars 23:59).

Pour avoir une idée de ce que fait « wordcount » (`wc`), tapez les commandes suivantes dans un terminal Unix :

```
printf 'un deux\n' > tmp1
printf 'trois\nquatre\n' > tmp2
wc tmp1 tmp2
```

Vous devriez obtenir :

```
  1    2    8 tmp1
  2    2   13 tmp2
  3    4   21 total
```

Explication : Les deux premières commandes créent chacune un fichier (`tmp1` et `tmp2`). Vous pouvez en voir le contenu comme suit :

```
cat tmp1
cat tmp2
```

puis la commande `wc` (« wordcount ») affiche des « statistiques » sur ces fichiers (et le total correspondant) :

- la première colonne compte le nombre de ligne ;
- la seconde le nombre de mots, au sens « séparés par des blancs » ;
- la troisième colonne est le nombres d'octets (= le nombre de caractères).

Comme d'habitude, pour plus de détails consultez la « manpage » de `wc` :

```
man wc
```

vous verrez qu'il existe d'autres options à `wc`.
Dans le projet de cette semaine, nous ne vous demandons de n'implémenter que la version de base décrite plus haut. Votre programme, `swc` (pour « sort of `wc` ») devra simplement fournir le même genre de résultat :

```
./swc tmp1 tmp2
  1    2    8 tmp1
  2    2   13 tmp2
  3    4   21 total
```

Matériel fourni

Récupérez les fichiers fournis pour cette semaine (`git pull`).
Vous devriez trouver dans le répertoire `provided/week02` :

`swc.c`

qui contient déjà une partie du programme, et que vous devez compléter.

Pour tout ce qui est des commandes git et de son fonctionnement, n'hésitez pas à revoir cette page complémentaire (semaine passée).

Travail à faire

En lisant le code fourni, vous verrez qu'il manque quatre choses (chercher le mot `TODO`) :

- un type nommé `counts` ;
- une fonction nommée `count` ;
- une partie de l'affichage dans la fonction `print_counts` ;
- des lignes dans le `main()` pour calculer et afficher les comptes totaux.

Vous devez écrire tout le code manquant de sorte à ce que le programme final compile et se comporte comme indiqué dans l'introduction.

Type counts

Le type `counts` à définir doit permettre de stocker les comptes du nombre de lignes, de mots et de « caractères » (= octets).

Comme ceci anticipe sur le cours à venir lundi, nous vous conseillons dans un premier temps vous concentrer sur le reste en le faisant avec des types de base que vous connaissez déjà, puis revenir terminer cette partie dès lundi (ce n'est pas très difficile).

Vous pouvez aussi, si vous préférez, consulter ici les transparents du cours de lundi prochain.

Fonction count()

La fonction `count` est celle qui fait l'essentiel du travail. Elle reçoit un « fichier » en paramètre (de type `FILE*`, fourni par le `main()` ; allez voir son appel dans le `main()`) et retourne un type `counts` défini ci-dessus comprenant les nombres de lignes, de mots et de caractères pour le fichier reçu en paramètre.

Pour calculer ces nombres, vous devez boucler sur le fichier tant qu'il y a quelque chose à lire. Utilisez pour cela la fonction `fgetc()`. Allez lire sa « manpage »

pour voir son prototype et le sens de sa valeur de retour. Notez donc en particulier qu'elle retourne la valeur `EOF` (qui est un `int` particulier, qui s'écrit tel quel en C : littéralement `EOF`) lorsque la lecture est terminée.

Le nombre de lignes du fichier est simplement le nombre de `'\n'` lus.

Le nombre de mots du fichier est le nombre de séquences de caractères non blancs séparés par des blancs. Par exemple, en écrivant « `<BLANC>` » pour un caractère blanc :

- « `Bonjour<BLANC>monsieur` » contient deux mots ;
- « `<BLANC><BLANC>Bonjour<BLANC><BLANC><BLANC>monsieur<BLANC>` » contient *aussi* deux mots.

Pour déterminer si un caractère lu est un blanc, utilisez la fonction `isspace()` (allez voir sa « manpage »).

Le nombre d'octets est simplement le nombre de caractères lus par `fgetc()`.

Nous insistons sur le fait que cela fait pleinement partie du travail de ce devoir que d'aller lire les « manpages » pour comprendre le fonctionnement des deux fonctions citées ici.
Et cela fait aussi partie du travail que de réfléchir et trouver l'algorithme pour compter les mots.

Fonction print_counts()

Complétez simplement la partie manquante en fonction de votre définition du type `counts`.

Fonction main()

Complétez la fonction `main()` partout où c'est nécessaire pour calculer, puis afficher les totaux des comptes calculez précédemment. Dans l'exemple de l'introduction, cela correspond à la dernière ligne affichée :

```
  3    4   21 total
```

A la différence du `wc` standard, votre programme devra afficher cette ligne de total même si le nombre de fichiers passés à `swc` est réduit à 1 :

```
./swc tmp1
  1    2    8 tmp1
  1    2    8 total
```

Testing

Pensez à tester votre programme (cela fait aussi partie des objectifs de ce devoir). Imaginez non seulement les cas standards, mais aussi tous les cas particuliers.

Rendu

Pour rendre le devoir, rajoutez le fichier `swc.c` complété à votre GitHub dans un répertoire `done/week02` (à la racine de votre GitHub, mais **PAS** dans `provided` ; vous ne devez **jamais** modifier le répertoire `provided`) puis « tagger » avec le tag `week02` et « pousser » le résultat vers GitHub (`git push --tags`).

Le délai pour faire le `push` de rendu est : dimanche 11 mars 23:59.

Et n'oubliez pas :

1. faire le rendu de la semaine passée avant dimanche soir ;
2. de vous inscrire par groupes de deux avant lundi 12 mars au soir (après il sera trop tard) au moyen du lien dans le menu à gauche ci-contre (**Note :** votre inscription est validée manuellement ; cela prend donc « *un peu* » de temps avant que vous ne receviez une confirmation et un dépôt GitHub pour votre groupe).