TP3 : Arguments en ligne de commande et E/S

★ **Exercice 1. Échauffement.**

Écrivez un programme C acceptant un nombre variable d'arguments sur la ligne de commande et affiche pour chacun d'eux le nombre de caractères correspondant. Ainsi, si ce programme est compilé sous le nom ***longueur_arg***, l'exécution de la commande
`longueur_arg 0 bonjour 2.56 adieu` produira la sortie :

```
l'argument no 0 contient 12 caractere(s)
l'argument no 1 contient 1 caractere(s)
l'argument no 2 contient 7 caractere(s)
l'argument no 3 contient 4 caractere(s)
l'argument no 4 contient 5 caractere(s)
```

Indication : On peut retrouver la longueur d'une chaîne de caractères avec la fonction `strlen()`, utilisable après avoir inclu le fichier d'entête `<string.h>`.

★ **Exercice 2. Les arguments de la ligne de commande sont des chaînes de caractères.**

▷ **Question 1.** Écrivez un programme duppliquant un fichier *source* sous le nom *destination* (que vous demanderez à l'utilisateur). Pour la copie, vous utiliserez les fonctions `fprintf` et `fscanf` avec la chaine de formatage `"%c"`.

▷ **Question 2.** Modifiez votre programme pour qu'il prenne ses arguments (`source` et `destination`) depuis la ligne de commande grâce à `argv`. Si aucun argument n'est fourni en ligne de commande, il faut encore demander interactivement à l'utilisateur le nom des fichiers concernés.

★ **Exercice 3. Les arguments de la ligne de commande sont des entiers.**

Lisez le code source ***max2.c***, disponible dans le dépot. Il demande interactivement deux entiers à l'utilisateur et renvoie la valeur maximum lue.

On souhaite modifier ***max2*** pour lui fournir les données en arguments de la ligne de commande.

▷ **Question 1.** Pour réaliser cela, un programmeur pressé a modifié ***max2.c***, sous le nom ***max2_v2.c***. Examinez ce programme et compilez-le. Vous obtenez un message d'erreur qui signifie que les parties gauches et droites d'affectations ne sont pas du même type : en effet, `a` et `b` sont des entiers, `argv[1]` et `argv[2]` sont des adresses (pointeurs). Exécutez le code exécutable (qui est tout de même généré, car ces erreurs sont des *warnings*).

▷ **Question 2.** Notre programmeur pressé a fait une première tentative de correction en forçant une conversion de type, dans le fichier ***max2_v3.c***. Lisez-le, compilez-le et vérifiez que l'erreur de compilation ne se produit plus. Exécutez le programme obtenu ... et concluez.

▷ **Question 3.** Corrigez le programme en utilisant la fonction `atoi` (cf. `man atoi`).

★ **Exercice 4. Arguments de la ligne de commande : dessin de motifs.**

On souhaite écrire un programme qui affiche des motifs répétitifs sur la sortie standard.
Les arguments de la lignes de commande sont les nombres de lignes et de colonnes qui mémorisent le nombre en lignes et en colonnes de motifs à tracer. Le motif dessiné est composé d'une alternance de deux motifs initialisés avec les valeurs reçues en arguments de la commande.

La commande comprend donc 4 arguments. Voici quelques exemples d'exécution de ce programme :

```
$ ./motif 3 15 ___/ \\___
___/\______/\______/\______/\______/\______/\______/\______/
\______/\______/\______/\______/\______/\______/\______/\___
___/\______/\______/\______/\______/\______/\______/\______/
$

$ ./motif 4 10 __] [__
__][____][____][____][____][__
[____][____][____][____][____]
__][____][____][____][____][__
[____][____][____][____][____]
$

$ ./motif 4 20 '\__' '/ '
\__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/
/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__
\__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/
/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__
$
```

Indication : Vous pouvez récupérer les motifs de la ligne de commande à l'aide de l'instruction `char *motif[] = {argv[3] , argv[4] };`

★ **Exercice 5. Mesurer la complexité d'un fichier C.**

On souhaite avoir une mesure de la complexité de divers programmes C. Pour cela, plusieurs métriques sont utilisables, que nous allons explorer au fil des questions.

La première idée est de mesurer la longueur du texte. Il est assez courant d'annoncer le nombre de lignes composant un programme donné (Noyau linux 2.6 : 4 millions ; GCC : 2,5 millions ; Windows XP : 40 millions, ...).

▷ **Question 1.** Faites un programme C `complexite` prenant le nom d'un fichier en argument et comptant le nombre de lignes le composant (il faut compter les occurences du caractère '\n').

▷ **Question 2.** Modifiez votre programme pour ne pas tenir compte des lignes blanches (il faut utiliser un booléen réinitialisé à chaque ligne indiquant si on a vu un caractère autre que '\t' et ' ').

▷ **Question 3.** (facultative) Modifiez votre programme pour ne pas compter ce qui se trouve à l'intérieur de commentaires. On souhaite traiter à la fois le style C (`/* ... */`) et le style C++ (`// ... fin de ligne`). On utilisera un booléen indiquant si on se trouve actuellement dans un commentaire ou non.

Cette métrique de complexité est trompeuse car un long programme ne comportant qu'un enchaînement de commandes sans `if`, `for` ou autres est sans doute moins complexe qu'un programme plus court présentant un schéma d'exécution plus complexe.

▷ **Question 4.** Modifiez votre programme pour qu'il compte les occurences du caractère '{', qui est présent dans la plupart des structures syntaxiques du C.

▷ **Question 5.** Voici deux extensions possibles pour votre programme, à réaliser si vous avez le temps :
- Comptez séparément les occurences des différentes constructions syntaxiques du C.
- Faites en sorte que votre programme compte les lignes lorsqu'on lui passe l'argument supplémentaire `--long` tandis qu'il compte la complexité si on lui passe l'option `--complexe`.