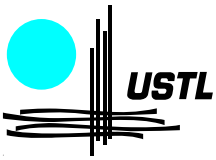

Licence d'informatique
Module de Pratique du C

Travaux pratiques

Mes commandes Unix

Philippe MARQUET

Octobre 2004

Le but de ce TP est de développer des versions basiques de quelques commandes Unix. Ces développements illustreront principalement les manipulations des tableaux et chaînes de caractères ainsi que l'organisation en multiples fichiers sources d'un développement en C.

1 Description des commandes

Les commandes qui seront développées sont des versions simplifiées des commandes Unix suivantes :

- `colrm` et `cut`
- `look` et `grep`

Vous pouvez vous référer au manuel en ligne pour une description détaillées de leur fonctionnement.

Nos versions de ces commandes seront nommées `mcolrm`, `mcut`, etc.

Spécifications communes

Chacune des commandes développées sera un *filtre*. Un filtre est une commande qui lit un texte à traiter sur son entre standard (`stdin`), produit son résultat sur la sortie standard (`stdout`), et produit éventuellement des messages d'erreur sur la sortie d'erreur (`stderr`).

Une commande se termine normalement sur un succès. Elle retourne une valeur non nulle à l'environnement si et seulement si une erreur est survenue.

Le comportement des commandes est exclusivement paramétré par des options qui sont passées sur la ligne de commande lors de leur invocation.

Les commandes traitent l'entrée standard ligne par ligne. On convient que nos versions basiques de ces commandes considèrent que les lignes font au plus 80 caractères. Si l'entrée standard fournie contient une ligne de plus de 80 caractères, la commande affiche un message d'erreur et se termine sur un échec.

Couper des colonnes

Les commandes `colrm` et `cut` coupent (ou conservent) des colonnes de chacune des lignes de l'entrée standard. Pour la commande `mcolrm`, une colonne est définie comme un caractère d'une ligne. Pour la commande `mcut`, une colonne est définie comme une suite de caractères (éventuellement vide), deux colonnes étant séparées par un caractère délimiteur. Les colonnes sont numérotées à partir de 1.

La syntaxe de ces commandes est la suivante :

```
mcolrm col [end]
mcut delim fieldno [fieldno]...
```

La commande `mcolrm` supprime la colonne `col` ou les colonnes `col` à `end`.

La commande `mcut` considère le délimiteur de champs `delim` et ne garde que les colonnes indiquées par les `fieldno`.

Recherche de lignes

Les commandes `mlook` et `mgrep` recherchent les lignes contenant un mot donné.
La syntaxe de ces commandes est la suivante :

```
mlook word
mgrep word
```

La commande `mlook` affiche les lignes commençant par le mot `word`. Les lignes fournies sur l'entrée de la commande doivent être triées ; une erreur est reportée sinon. La commande se termine sur un succès si et seulement si une ligne débutant par le mot a été trouvée.

La commande `mgrep` affiche les lignes contenant le mot `word`. Elle se termine sur un succès si et seulement si une telle ligne a été trouvée.

2 Organisation des développements

Dans vos développements, il vous faut favoriser la réutilisation de code. Définissez des fonctions communes à toutes les commandes et isolez les dans des fichiers sources distincts.

Vous pouvez par exemple isoler dans une bibliothèque `readl` la fonction dont l'interface est fournie par le fichier `readl.h` suivant :

```
#define MAXLINE  81

/* Lit une ligne sur l'entree standard.
   Cette ligne doit comporter moins de MAXLINE caracteres.

   Le resultat est retourne dans line.
   Un \0 est ecrit en fin de la chaine.

   Le tableau line doit etre de taille au moins MAXLINE+1.

   Retourne le nombre de caracteres lu, non compris le \0 final.
   Retourne EOF si la fin de fichier est atteinte.

   Termine le programme sur une erreur si rencontre une ligne de plus
   de MAXLINE caracteres.
*/
extern int readl(char line[]);
```

ou considérer la bibliothèque `tools` dont un extrait du fichier d'interface `tools.h` peut comporter

```
/* Termine l'execution du programme sur une erreur fatale.

   Si assert est faux, affiche le message sur la sortie d'erreur et
   termine en retournant la valeur status a l'environnement.
*/
extern void fatal(int assert, const char *message, int status);
```

La compilation et la mise à jour des exécutables en fonction des évolutions du code se fera en utilisant la commande `make`.

3 Quelques éléments d'aide

Les quelques indications données ici vous seront utiles. Ces indications ne sont pas complètes. Il vous faudra consulter le manuel en ligne pour une information plus complète.

Lecture de l'entrée standard La lecture d'une ligne sur l'entrée standard se fait par la fonction `fgets()` :

```
#include <stdio.h>
char * fgets(char * s, int size, FILE * stream);
```

qui lit au plus `size-1` caractères depuis le flux `stream` et les place dans le tableau `s`. Le flux associé à l'entrée standard est identifié par la valeur `stdin` définie dans `stdio.h`. La lecture s'arrête après `EOF` ou un retour-chariot.

Production sur la sortie standard et la sortie d'erreur De la même manière que `printf()` produit un message sur la sortie standard, la fonction

```
#include <stdio.h>
int fprintf(FILE *stream, const char *format, ...);
```

produit un message sur le flux désigné par le paramètre `stream`. Deux valeurs de type `FILE *` sont définies dans `stdio.h` pour désigner la sortie standard et la sortie d'erreur, `stdout` et `stderr`. Ainsi `printf(...)` est équivalent à `fprintf(stdout, ...)`.

Comparaison de chaînes de caractères Les chaînes de caractères ne peuvent être directement comparées avec un opérateur du langage C. On utilise la fonction

```
#include <string.h>
int strcmp(const char *s1, const char *s2);
```

qui retourne la différence entre les premiers caractères différents des chaîne et `s2`. En cas d'égalité, `strcmp()` retourne une valeur nulle (donc une valeur fausse, attention).

Recherche de sous-chaînes La fonction

```
#include <string.h>
char *strstr(const char *big, const char *little);
```

recherche la chaîne de caractères `little` dans la chaîne de caractères `big`. En particulier, elle retourne

- `NULL` ssi `little` n'apparaît pas dans `big`;
- `big` ssi `big` débute par `little`.

Terminaison du programme La fonction

```
#include <stdlib.h>
void exit(int status);
```

termine l'exécution du programme en cours. La valeur de `status` est retournée à l'environnement. La norme C définit les valeurs `EXIT_SUCCESS` (0) et `EXIT_FAILURE` comme des valeurs possible de `status`.

Lecture des paramètres de la ligne de commande Le prototype de la fonction `main()` peut être complété pour accéder aux paramètres de la ligne de commande :

```
int main(int argc, char *argv[]);
```

La valeur de `argc` donne le nombre de paramètres ayant été passés à la commande (y compris le nom du programme). Le tableau `argv` contient les valeurs de ces paramètres. Essayez le programme d'exemple suivant :

```
#include <stdio.h>
#include <stdlib.h>

int
main (int argc, char *argv[])
```

```
{
    int i;

    fprintf(stderr, "   argc = %d\n", argc);

    for (i=0; i<argc ; i++) {
        fprintf(stderr, "argv[%d] = %s\n", i, argv[i]);
    }

    exit(EXIT_SUCCESS);
}
```

Les arguments sont considérés comme des chaînes de caractères. On utilisera la fonction

```
#include <stdlib.h>
int atoi (const char *str);
```

pour convertir un argument de la commande (i.e. un élément du tableau `argv`) qui en une valeur entière.