TP 3

Manipulation de fichiers

Durée prévue : 1 séance (1h30)

Trucs et astuces pour accéder à l'historique du shell :

— La flèche du haut !

— La commande `history` ;

— (Ctrl + R) : commencez par taper le début de la commande recherchée, puis utilisez Ctrl + R et Ctrl + S pour parcourir les commandes correspondantes dans l'historique.

Archive pour le TP : récupérez l'archive contenant les squelettes de codes à remplir sur la page web du cours :

— copiez l'adresse (clic-droit « copier l'adresse du lien » ou un truc du genre) ;

— se placer dans un terminal à un bon endroit (`~/LIFASR5/TP3/` semble une bonne idée) ;

— utilisez la commande `wget` suivie de l'adresse de l'archive (qu'il vous suffit de coller) ;

— désarchivez avec `tar -xvzf lifasr5_tp3.tgz` sur la ligne de commandes.

EXERCICE 1 ▶ Une fonction readline

Le but est d'écrire avec la primitive `read()` une fonction pour lire une ligne de texte reçue sur l'entrée standard.

1) Consultez le site `https://www.cplusplus.com/reference/`, et répondez aux questions suivantes. Quelle méthode de la classe `std::string` permet :

— de vider de son contenu une chaîne de caractères de cette classe ?

— d'ajouter un caractère (de type `char`) à la fin d'une chaîne de caractères de cette classe ?

2) Dans un fichier source `myreadline.cpp`, définissez une fonction d'en-tête `void myreadline(std::string &s)` dont le rôle sera le suivant : elle doit permettre de lire une ligne de texte reçue sur l'entrée standard. Pour cela, la fonction doit vider la chaîne `s` qui lui est passée en paramètre, puis lire caractère par caractère sur l'entrée standard avec la primitive `read()`, en rangeant les caractères lus dans `s` ; la lecture s'arrête lorsque le caractère de retour à la ligne `'\n'` est rencontré. Ne vous occupez pas de la gestion des cas d'erreurs.

3) Toujours dans le fichier `myreadline.cpp`, définissez une fonction principale `int main(void)` afin de tester votre fonction `myreadline()` : votre programme doit simplement inviter l'utilisateur à entrer une ligne de texte, puis afficher la ligne entrée. Compilez votre programme (avec `g++ -Wall -o myreadline myreadline.cpp` par exemple), puis testez le : déboguez en cas de besoin !

4) Que se passe-t-il lorsque la fonction `myreadline()` reçoit le caractère de fin de fichier `EOF` alors qu'elle est bloquée en attente de lecture ? Vous pouvez testez cela en envoyant vous même le caractère de fin de fichier à votre programme avec Ctrl + D , mais on attend que vous répondiez à cette question en expliquant le comportement de la fonction, pas simplement en rapportant vos observations.

EXERCICE 2 ▶ Paramètres de la ligne de commande

En C/C++, on peut récupérer les paramètres passés de la ligne de commande au programme *via* les paramètres de la fonction `main`, dont le prototype peut être :

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

ou encore

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

Dans les deux cas, `argc` donne le nombre d'arguments entrés sur la ligne de commande **plus un** (le nom de l'exécutable compte pour un). `argv` est un tableau de chaînes de caractères :

— `argv[0]` contient le nom de l'exécutable,

— `argv[1]`, ..., `argv[argc]` contiennent chacun des arguments passés à la ligne de commande.

On vous fournit un squelette de code appelé `paraldc.cpp` que vous compilerez en un binaire `paraldc` :

1) Comment se comporte le programme fourni ? Testez avec différents arguments dans la ligne de commande :

```
$ ./paraldc
$ ./paraldc toto
$ ./paralcd titi toto 42
```

2) Modifiez le programme pour qu'il n'accepte que les lignes de commande avec un seul argument ; si cela n'est pas le cas, le programme affiche un message d'erreur sur la sortie d'erreur standard :

```
$ ./paraldc titi 65
Il faut exactement un argument après la commande.
```

EXERCICE 3 ▶ **Une commande** `wc`

Le but est d'écrire un petit programme qui compte le nombre d'octets que contient un fichier.

1) Complétez pour cela le programme `mywc.cpp` (fourni dans l'archive venant avec le TP).

```
// ...
int main(int argc, char *argv[]) {
  int fd; // descripteur de fichier
  char c; // servira ici de buffer
  int nbrd, nbbytes;

  if(argc < 2) {
    print_usage(argv[0]);
    return -1;
  }
  // TODO HERE
  // open file in read only mode + ...

  return 0;
}
```

Vous le compilerez avec `g++ -Wall mywc.cpp -o mywc`. Testez ensuite avec `./mywc onefilename` ; vous comparez votre résultat avec celui de `wc -c onefilename`

2) Que se passe-t-il si vous appelez votre programme avec comme paramètre le nom d'un fichier qui n'existe pas ? Mettez en place une gestion de l'erreur, avec un message adéquat (`man open`, `man errno`, `man strerror` ou `man perror`) :

```
if (fd < 0) {...
 if (errno == ...)
}
```