INF123 - Partiel

Année 2015-2016- Partiel

8 mars 2016

Durée : 1h30.
Tout document interdit à l'exception du mémo bash non annoté.
Calculatrices, téléphones interdits.

Le barème est indicatif.
Pour chaque question, une partie des points peut tenir compte de la présentation.
Toute réponse même incomplète sera valorisée à partir du moment où elle réalise au moins une partie de ce qui est demandé.
Les questions sont relativement indépendantes, et dans tous les cas vous pouvez utiliser les fonctions demandées dans les questions précédentes même si vous n'avez pas réussi à les écrire.
Dans la première partie, vous pouvez utiliser les fonctions de la bibliothèque string.h vues en TD ou en TP.

1 Programmer une commande cp simplifiée ($\sim$ 14 points)

Préambule. Voici une partie de l'arborescence d'un étudiant de INF123 (tous les fichiers et tous les répertoires ne sont pas indiqués) :

```
INF123
+-- TP1
|   + ...
|
|-- TP42
|   +-- PROGS
|   +   +-- aa.c
|   +   + ...
|   +   +-- Makefile
|   +   +-- machin.sh
|   +   +-- truc.sh
|   +   + ...
|-- TP50
```

Question 1 :

Le répertoire courant étant **TP50** :

1. donnez ***une*** commande permettant de copier tous les fichiers suffixés par **.sh** du répertoire **PROGS** de **TP42** dans le répertoire **TP50**.
2. donnez une commande permettant de copier le **Makefile** du répertoire **PROGS** de **TP42** dans le répertoire **TP50**, sous le nom **Makefile42**.

Le but de cette section est de programmer **en C** un exécutable `mon_cp` semblable à la commande `cp`.
Cet exécutable s'utilisera de 2 façons :

1. copie d'un fichier vers un autre :

   `./mon_cp fichier_source fichier_destination`

   copie `fichier_source` dans `fichier_destination`

2. copie de plusieurs fichiers dans un répertoire :

   `./mon_cp fichier_source_1 fichier_source_2 ... fichier_source_n repertoire_destination`

   copie `fichier_source_1`, ... `fichier_source_n` ($n \geq 2$) dans `repertoire_destination`.

Outre les options (comme `-r` par exemple) qui ne sont pas prises en compte, la principale différence entre `cp` et `mon_cp` est que `mon_cp` ne permet pas de copier un fichier source unique dans un répertoire.

1.1 Copie d'un fichier vers un autre

Pour commencer, nous avons besoin d'une fonction `copie` de prototype :

```
int copie(char nom_src[], char nom_dest[]) ;
```

Cette fonction copie le contenu du fichier de nom `nom_src` dans le fichier de nom `nom_dest`. Elle retourne :
- 1 si l'ouverture du fichier `nom_src` échoue,
- 2 si l'ouverture du fichier `nom_dest` échoue,
- 0 sinon.

Cette fonction n'affiche aucun message.

Question 2 :

Donnez le code de la fonction `copie`.

Question 3 :

Écrivez la fonction `main` d'un programme qui prend 2 noms de fichiers en arguments de la ligne de commande, et copie le contenu du premier dans le second. Si le nombre d'arguments est incorrect, un message rappelant la syntaxe attendue est affiché, et le programme se termine. La copie est effectuée par la fonction `copie` écrite au paragraphe précédent et un message d'erreur explicite est affiché pour chaque code de retour non nul de cette fonction.

1.2 Copie de plusieurs fichiers dans un répertoire

Construction du nom de la cible. Lors de la copie du fichier de nom `fich` ou <`chemin`>`/fich` dans le répertoire de nom `rep`, le nom du fichier créé (que nous appellerons *cible*) est en réalité `rep/fich`. De plus, sur la ligne de commande, il est possible que le nom du répertoire soit suivi d'un slash (`'/'`).

Exemple
Si l'utilisateur saisit
`./mon_cp exemple.txt PROGS/mon_prog.c ../TP25/CR/compte_rendu.txt A_RENDRE`
ou
`./mon_cp exemple.txt PROGS/mon_prog.c ../TP25/CR/compte_rendu.txt A_RENDRE/`
(la seule différence entre les 2 lignes est le slash final)
les fichiers suivants seront crées :
- **A_RENDRE/exemple.txt** (copie de **exemple.txt**)
- **A_RENDRE/mon_prog.c** (copie de **PROGS/mon_prog.c**)
- **A_RENDRE/compte_rendu.txt** (copie de **../TP25/CR/compte_rendu.txt**)

Question 4 :

Écrivez une fonction de prototype :

```
int slash_fin(char d[], char s[]) ;
```

qui copie la chaîne `s` dans la chaîne `d`, et ajoute un slash à la fin de la chaîne `d` si `s` ne se termine pas par `'/'`. La valeur de retour est la longueur de `d`.
Par exemple, si `s` est la chaîne `"A_RENDRE"` ou `"A_RENDRE/"`, alors `d` sera la chaîne `"A_RENDRE/"` dans les 2 cas, et la valeur de retour sera 8.

Question 5 :

Écrivez une fonction de prototype :

```
int indice_apres_dernier_slash(char f[]) ;
```

qui retourne
- l'indice dans la chaîne `f` du caractère qui suit le dernier slash s'il existe.
- 0 si la chaîne `f` ne comporte pas de slash.

Par exemple :
- `indice_apres_dernier_slash("exemple.txt")` renvoie 0.
- `indice_apres_dernier_slash("../TP25/CR/compte_rendu.txt")` renvoie 11.

Question 6 :

En utilisant les fonctions précédentes, écrivez une fonction de prototype

```
void chemin_cible(char c[], char r[], char f[]) ;
```

qui remplit la chaîne `c` (nom de la cible) à partir des chaînes `r` (nom du répertoire) et `f` (nom du fichier).
Par exemple, si `r` est la chaîne `"A_RENDRE"` ou `"A_RENDRE/"`, et `f` la chaîne `"../TP25/CR/compte_rendu.txt"`, alors `c` sera la chaîne `"A_RENDRE/compte_rendu.txt"`.

Programme principal de `mon_cp`. À l'aide des fonctions écrites précédemment :

Question 7 :

Modifiez la fonction `main` de la question 3 pour que le programme ait le comportement attendu, comme décrit page 1 : copie d'un fichier vers un autre, ou copie d'au moins 2 fichiers dans un répertoire.
Si le nombre d'arguments est incorrect ou si une des copies échoue (parce que l'un des fichiers n'a pas pu être ouvert – cf le code de retour de la fonction `copie`), un message d'erreur est affiché et le programme se termine.
Indication : Déclarez un tableau de caractères `CH` de taille suffisante (par exemple `char CH[100] ;`) pour manipuler les noms des cibles à construire, et vous supposerez qu'il n'y aura pas de problème de débordement.

1.3 Fichiers d'entête et Makefile

Le programme est réparti dans des fichiers sources comme suit :
- **copie.c** contient la fonction `copie`
- **noms.c** contient les fonctions `slash_fin`, `indice_apres_dernier_slash` et `chemin_cible`
- **mon_cp.c** contient le programme principal `main`

Question 8 :

1. Quels sont les fichiers **.h** nécessaires pour compléter les fichiers sources, et quel en est le contenu ?
2. Dans chaque fichier source, quelles sont les directives `include` nécessaires (pour ces **.h**) ?

Question 9 :

1. Dessinez le graphe de dépendance de ce programme.
2. Écrivez le fichier `Makefile` permettant de construire l'exécutable `mon_cp` à partir de ces fichiers sources.

2 Gérer une corbeille (∼ 6 points)

Le but de cette section est de gérer une corbeille permettant la suppression de fichiers avec restauration possible de la dernière version.
La corbeille sera un sous-répertoire **Corbeille** du répertoire personnel de l'utilisateur.
Elle est gérée à l'aide de 2 scripts shell appelés **supprime.sh** et **restaure.sh**.
- **supprime.sh** prend un certain nombre de noms de fichiers en arguments, et les déplace dans la corbeille.
- **restaure.sh** prend un nom de fichier (sans chemin) en argument et récupère la version la plus récente de ce fichier présente dans la corbeille.

Pour que cette restauration soit possible, les noms des fichiers de la corbeille sont suffixés, lors de la suppression, par un numéro représentant le nombre de fichiers du même nom présents dans la corbeille. Le plus récent est donc celui de numéro le plus élevé. On utilisera également un script intermédiaire **nombre_dans_corbeille.sh** qui prend un nom de fichier **fich** (sans chemin) et qui affiche le nombre de fichiers de la forme **fich.**<numero> présents dans la corbeille.

IMPORTANT. On supposera que le nom d'un fichier dans la corbeille (sans son numéro) n'est jamais le préfixe du nom d'un autre fichier dans la corbeille. Par exemple, il n'y aura jamais dans la corbeille à la fois **ploum** et **ploum.v2**. Vous n'avez pas à le vérifier, et on ne demande pas que vos scripts respectent cette spécification.

Exemple.

commandes successives	contenu corbeille après exécution	commentaires
	vide	
`./supprime.sh ploum`	**ploum.1**	ploum est supprimé du répertoire courant
`./supprime.sh ploum`	**ploum.1 ploum.2**	la version la plus récente de ploum est ploum.2
`./nombre_dans_corbeille.sh ploum`	**ploum.1 ploum.2**	affiche 2
`./restaure.sh ploum`	**ploum.1**	restauration de ploum dans le répertoire courant
`./supprime.sh ../TP25/essai.c`	**ploum.1 essai.c.1**	nom de fichier sans chemin dans la corbeille
`./supprime.sh trucs.sh aaa.txt`	**ploum.1 essai.c.1 trucs.sh.1 aaa.txt.1**	suppression de plusieurs fichiers à la fois

2.1 Ecriture de nombre_dans_corbeille.sh

Rappel : On rappelle que la commande `wc -w` sans argument affiche à l'écran le nombre de mots de l'entrée standard.

Question 10 :

En "pipelinant" les commandes `ls` et `wc` (ou en redirigeant leurs sorties ou leurs entrées), écrivez le script **nombre_dans_corbeille.sh**. On ne fera pas de vérification sur les arguments, c'est-à-dire que vous supposerez que ce script est appelé avec un argument qui est un nom de fichier sans chemin.

2.2 Ecriture de supprime.sh

Question 11 :

En faisant appel à **nombre_dans_corbeille.sh**, écrivez le script **supprime.sh**, afin qu'il ait le comportement suivant :

- sans argument, il affiche un message d'erreur et se termine.
- s'il n'existe pas de répertoire **Corbeille** dans le répertoire personnel, ce répertoire est créé.
- pour chacun des arguments :
  - si l'argument n'est pas un fichier, un message est affiché.
  - sinon le fichier est déplacé dans la corbeille sous le nom décrit ci-dessus (c'est-à-dire sans le chemin éventuel, et suffixé avec le numéro approprié).

Indication : Utilisez les commandes `basename` et `expr` (cf memo bash).

2.3 Ecriture de restaure.sh

Question 12 :

Question bonus

Écrivez le script **restaure.sh** qui prend un nom de fichier sans chemin en argument, et déplace la version la plus récente de ce fichier (si elle existe) de la corbeille dans le répertoire courant.