DLST

UNIVERSITÉ Grenoble Alpes

INF203 - Partiel
Année 2023-2024

Durée : 2h.
Une feuille A4 manuscrite recto/verso autorisée, dictionnaire papier (non annoté) autorisé pour les étudiants étrangers uniquement.
Tout autre document, calculatrices et appareils électroniques interdits.
Pour chaque question, une partie des points peut tenir compte de la présentation.
Le barème est indicatif.

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 scripts et les fonctions demandées dans les questions précédentes même si vous n'avez pas réussi à les écrire.

Par ailleurs, tout code ou algorithme *élégant*, en plus d'être correct, sera bonifié.

1 Programmation en langage C (8 pts)

Le but de cet exercice est réaliser un exécutable `somme`, compilé à partir d'un code en langage C `somme.c`, qui effectue une addition rédigée sous la forme d'une *expression littérale* (d'une chaîne de caractères). Par exemple, à partir de l'expression littérale `"3216+128+72"` stockée dans le tableau de caractères `addition[]`, `somme` imprimera à l'écran :

```
Le résultat vaut 3344.
```

Questions :

1. (1 point) Donnez le code qui permet d'affecter à la variable `n` de type `int` la valeur du chiffre contenu dans le caractère `c` de type `char` (pris parmi `{'0',...,'9'}` qui se suivent dans la table ASCII).

   Solution:

   ```
   int n = c-'0';
   ```

2. (2 points) Supposons pour l'instant que l'expression littérale `addition[]` qui nous intéresse ne contienne que des chiffres, et donc aucun symbole `+`. Par exemple `addition[20]="2458"`.

   Écrivez une fonction `sum()` qui prend en argument un tableau de caractères (qui seront des chiffres) et qui renvoie un entier égal à la valeur entière du nombre décrit par ces chiffres. Dans notre exemple, `sum(addition)` renvoie l'entier `2458`.

   Aide : Voici le pseudo-code d'un algorithme permettant de transformer un tableau de chiffres en un nombre correspondant.

   ```
   nombre = 0
   Pour chaque chiffre x lu dans le tableau :
     nombre = nombre*10 + valeur du chiffre x
   ```

   Attention, il vous reste à traduire ce procédé en langage C en tenant compte des types de données utilisés.

   Solution:

   ```
   int sum(char chaine[]) {
     int i=0;

     int sum=0;
     while (chaine[i] != '\0') {
       sum=sum*10+(chaine[i]-'0');
   ```

```
    i++;
  }
  return sum;
}
```

3. (2 points) La chaîne de caractère `addition[]` peut désormais contenir des symboles `+`.
   Réécrivez la fonction `sum()` de sorte qu'elle effectue désormais la somme des entiers décrits dans `addition[]`. Par exemple, si désormais la chaîne de caractères `addition[]` contient l'expression littérale `"3216+128+72"`, `sum(addition)` renvoie l'entier `3344`.

Solution:

```
int sum(char chaine[]) {
  int i=0;

  int sum=0;
  int partial_sum=0;
  while (chaine[i] != '\0') {
    if (chaine[i] != '+')
            partial_sum=partial_sum*10+(chaine[i]-'0');
    else
    {
            sum+=partial_sum;
            partial_sum=0;
    }
    i++;
  }
  sum+=partial_sum;

  return sum;
}
```

4. (2 points) Complétons le code en incluant une fonction `main()`. Cette fonction contiendra la définition de la chaîne de caractères `addition[]`, appellera la fonction `sum()` sur cette chaîne et imprimera à l'écran :

   `Le résultat vaut XXXX.`

   où `XXXX` contient le résultat de l'addition.

Solution:

```
int main(){
  char addition[80] = "3216+128+72";

  printf("Le résultat vaut %d\n.",sum(addition));

  return 0;
}
```

5. (1 point) À l'aide de quelle commande compilez-vous alors le code `somme.c` afin de produire l'exécutable `somme` ? Comment exécutez-vous ensuite `somme` dans un terminal bash ?

Solution:

```
clang somme.c -o somme
./somme
```

2 Conversion de podcasts (15pts)

Vous avez récupéré des fichiers audio sur divers sites de podcasts, malheureusement ils sont tous encodés dans des formats différents, et votre vieux lecteur ne supporte que les fichiers mp3.

L'objectif de ce problème est d'automatiser la conversion de tous ces fichiers au bon format. Nous écrirons un script bash que nous complèterons au fur et à mesure.

Pour cela, on utilisera la commande `ffmpeg`, qui s'utilise (dans la version la plus simple) :

```
ffmpeg -i input_file output_file
```

pour transformer le fichier `input_file` en `output_file` (le programme se charge tout seul de choisir le bon format en fonction de l'extension donnée aux fichiers).

Questions :

1. (1 point) Quelle commande bash vous permet de retrouver le synopsis de la commande donné ci-dessus ?

   Solution:

   ```
   man ffmpeg
   ```

   Barème On peut accepter `ffmpeg -h` ou `ffmpeg -help`.

2. (2 points) Votre script devra prendre comme unique argument un nom de répertoire dans lequel sont présents les fichiers à convertir. Pour l'instant, on cherche uniquement à se déplacer dans le bon répertoire.
   Écrivez une suite de commandes qui vérifie si un argument a été donné au script, et si cet argument est bien un répertoire existant. Si c'est bien le cas, on se déplace dans ce répertoire.

   Solution:

   ```
   if [ $# -eq 1 -a -d $1 ]
   then
       cd $1
   fi
   ```

   Barème
   - ½ : Structure du `if`
   - ½ : Utilisation correcte de l'argument `$1`
   - ½ : Utilisation correcte de `$#`
   - ½ : Test d'existence du répertoire

3. (3 points) Dans un premier temps, on ne s'intéresse qu'aux fichiers de format `.mpga`.
   Écrire un script bash qui parcourt tous les fichiers du répertoire courant de la forme `nom_fichier.mpga`, et qui pour chacun de ces fichiers, les convertit via `ffmpeg` en un autre qui portera le même nom avec l'extension `.mp3`

   Solution:

   ```
   for INPUT in *.mpga
   do
     OUTPUT=$(basename $INPUT .mpga).mp3
     ffmpeg -i $INPUT $OUTPUT
   done
   ```

   On accepte aussi l'utilisation de `cut` comme dans la question suivante

Barème
- — ½ : Structure du `for`
- — ½ : Utilisation du métacaractère dans `*.mpga`
- — ½ : Utilisation de variables
- — ½ : Utilisation correcte de `basename`
- — ½ : Substitution de commande `$( ... )`
- — ½ : Reconstruction correcte du nom du fichier de sortie

4. (2 points) On veut maintenant être capable de gérer tous les formats audio possibles.
   On considère que tous les fichiers présents dans le répertoire courant portent un nom de la forme `nom.extension`.
   Donnez des commandes bash qui, étant donné un nom de fichier `$FICH`, permettent de récupérer séparement le nom et l'extension de ce fichier dans deux variables.

Solution:

```
NOM=$(echo $FICH | cut -d'.' -f1)
EXT=$(echo $FICH | cut -d'.' -f2)
```

Barème
- — ½ : Utilisation de `cut`
- — ½ : Choix d'arguments pertinents pour `cut`
- — ½ : Utilisation d'une redirection
- — ½ : Stockage des résultats dans des variables

5. (2 points) À l'aide des commandes écrites à la question précédente (que vous pouvez remplacer par un commentaire si vous n'avez pas réussi à les écrire), écrire un script qui convertit tous les fichiers du répertoire courant au format `.mp3`.
   On prévoira un test pour ne pas effectuer la conversion dans le cas où le fichier d'origine est déjà au format `.mp3`.

Solution:

```
for FICH in *
do
    NOM=$(echo $FICH | cut -d'.' -f1)
    EXT=$(echo $FICH | cut -d'.' -f2)
    if [ $EXT != mp3 -a ! -d $FICH -a $FICH != erreurs.log ]
    then
    ffmpeg -i $FICH $NOM.mp3
    fi
done
```

Barème
- — ½ : Boucle sur `*` utilisant la réponse de la question précédente (admise si besoin)
- — ½ : Imbrication correcte du `for` et du `if`
- — ½ : Test `$EXT != .mp3`
- — ½ : Reconstruction du nom du fichier de sortie avec `$NOM`

Les deux autres conditions du test (tests sur `$FICH`) ne sont pas demandées (elles ne seront utiles que dans la question suivante).

6. (2 points) On souhaite compléter le script précédent pour gérer les erreurs de conversion, ou les formats non reconnus par `ffmpeg`. À la fin de l'exécution de votre script, le répertoire courant devra contenir :
   - — les fichiers `.mp3` générés,
   - — un répertoire `Erreurs` qui contient les fichiers pour lesquels la conversion a échoué (il sera vide si aucune erreur ne se produit),

— et un fichier `erreurs.log` qui contient les noms des fichiers pour lesquels la conversion a échoué.
Écrivez un test qui réalise les opérations suivantes :
— en cas de réussite de la conversion, on efface le fichier d'origine ;
— en cas d'échec, on déplace le fichier dans le répertoire `Erreurs` et on ajoute le nom de ce fichier à la fin d'un fichier nommé `erreurs.log` (qui n'existe pas forcément initialement)
et précisez à quel endroit de votre script ce test doit être positionné.

Solution:

```
if [ $? -eq 0 ]
then
    rm $FICH
else
    mv $FICH Erreurs
    echo $FICH >> erreur.log
fi
```

à insérer immédiatement après l'appel à `ffmpeg`

Barème
- ½ : Utilisation de `$?`
- ½ : Utilisation correcte de`mv`
- ½ : Utilisation de la redirection `>>`
- ½ : Test positionné après `ffmpeg`

Il est aussi possible d'écrire `if ffmpeg -i $FICH $NOM.mp3`, dans ce cas pas besoin d'utiliser explicitement `$?`.

7. (1 point) Quelle commande permet, au début de votre script, de créer le répertoire `Erreurs` uniquement dans le cas où ce répertoire n'existe pas encore ?

Solution:

```
mkdir -p Erreurs
```

On accepte également un test `if [ -d Erreurs ]` avant l'appel à `mkdir`.

Barème
- ½ : Utilisation de `mkdir`
- ½ : Option ou test pertinent

8. (2 points) On veut estimer la taille totale des fichiers produits pour savoir si on peut les enregistrer sur le lecteur MP3.
La commande `stat -c%s nom_fichier` permet d'afficher la taille d'un fichier en octets.
Écrivez une suite de commandes bash qui permet de calculer et d'afficher la taille totale des fichiers `.mp3` présents dans le répertoire courant.

Solution:

```
TOTAL=0
for FICH in *.mp3
do
    TAILLE=$(stat -c%s $FICH)
    TOTAL=$(expr $TOTAL + $TAILLE)
done
echo $TOTAL octets au total
```

Beaucoup d'autres solutions possibles, toutes aussi valables :
- en imbriquant les appels à `stat` et à `expr`
- à base `ls -l` et de `cut`
- `cat *.mp3 | wc -c`
- en insérant les appels à `stat` dans la boucle de la question précédente
- etc.

Barème

- — ½ : Récupération de la taille d'un fichier
- — ½ : Utilisation d'une variable pour accumuler la taille totale
- — ½ : Initialisation et affichage final de la variable `TOTAL`
- — ½ : Utilisation de `expr`

A Annexe : rappels

On rappelle ici différentes commandes et options qui peuvent être utiles.

cut affiche une partie seulement de chaque ligne de ses entrées.

`-c debut-fin` permet de choisir les caractères affichés ;

`-f debut-fin` permet de choisir des colonnes ;

`-d` permet de choisir le délimiteur.

diff détermine si les contenus de 2 fichiers sont identiques :

— aucune réponse : les 2 fichiers sont identiques.

— la réponse est seulement que les 2 fichiers sont différents : l'un au moins n'est pas un fichier texte.

— la liste des différences entre les 2 fichiers : les 2 fichiers sont des fichiers texte.

expr calcule l'expression décrite par ses arguments.

grep n'affiche que les lignes d'un fichier contenant une chaîne de caractères donnée.

`-c` n'affiche que le nombre total de lignes contenant la chaîne recherchée.

`-v` n'affiche que les lignes ne contenant pas la chaîne donnée.

head et tail

`head` affiche le début de ses entrées, selon l'option choisie :

`-n N :` les `N` premières lignes

`-n -N :` toutes les lignes sauf les `N` dernières

De même la commande `tail` affiche la fin de ses entrées :

`-n N :` les `N` dernières lignes

`-n +N :` les dernières lignes à partir de la N^e

sed affiche les lignes de ses entrées en les filtrant et/ou modifiant selon les instructions données :

`sed /Candide/d` affiche celles ne contenant pas `Candide`

`sed s/Candide/Toto/` remplace le premier `Candide` par `Toto` sur chaque ligne

`sed s/Candide/Toto/g` remplace tout les `Candide` par `Toto`

sort affiche les lignes de ses entrées triées par ordre croissant

tr copie l'entrée standard sur la sortie standard en substituant

`tr chaîne1 chaîne2 :` chaque caractère présent dans `chaîne1` par le caractère de position correspondante dans `chaîne2`

`tr -s chaîne :` les répétitions des caractères présents dans `chaîne` par une occurrence unique

wc compte les caractères, les mots ou les lignes lus sur son entrée standard :

`-c` compte les caractères

`-w` compte les mots

`-l` compte les lignes