Projet Langage de Scripts – SHELL : « Un tableur en Shell »

Licence3 Informatique

1 Objectifs

L'objectif de ce projet est de réaliser en *shell* (`/bin/sh`) un mini tableur. Ce tableur prendra en paramètre une feuille de calculs et devra fournir en retour le tableau complètement renseigné.

2 Description

2.1 Ligne de commandes

La syntaxe de votre commande sera :
`tableur [-in feuille] [-out résultat] [-scin sep] [-scout sep] [-slin sep] [-slout sep] [-inverse]`
Les différentes options sont les suivantes, leur ordre d'apparition n'est pas et ne doit pas être fixé :

- `-in feuille` : permet d'indiquer dans quel fichier (`feuille`) se trouve la feuille de calculs. Si cette option n'est pas spécifiée sur la ligne de commande, la feuille de calculs sera lue depuis l'entrée standard ;
- `-out résultat` : permet d'indiquer dans quel fichier (`résultat`) doit être écrite la feuille calculée. Si cette option n'est pas spécifiée, le résultat sera affiché sur la sortie standard ;
- `-scin sep` : permet de spécifier le séparateur (`sep`) de colonnes de la feuille de calculs initiale. Par défaut si cette option n'est pas spécifiée, ce séparateur est la tabulation ;
- `-slin sep` : permet de spécifier le séparateur (`sep`) de lignes de la feuille de calculs initiale. Par défaut si cette option n'est pas spécifiée, ce séparateur est le retour chariot ;
- `-scout sep` : permet de spécifier le séparateur (`sep`) de colonnes de la feuille calculée. Par défaut si cette option n'est pas spécifiée, ce séparateur est identique au séparateur de colonnes de la feuille de calculs initiale ;
- `-slout sep` : permet de spécifier le séparateur (`sep`) de lignes de la feuille calculée. Par défaut si cette option n'est pas spécifiée, ce séparateur est identique au séparateur de lignes de la feuille de calculs initiale ;
- `-inverse` : cette option provoque l'inversion lignes/colonnes de la feuille calculée.

Exemple :

```
tableur -in foo -scin : -scout |
```

permet de calculer la feuille stockée dans le fichier `foo` où le séparateur de colonnes est le symbole deux points. La feuille calculée sera affichée à l'écran avec comme séparateur de colonnes le symbole `|`.

2.2 Format des feuilles de calculs

Les feuilles de calculs seront de simples fichiers texte. Implicitement les lignes et les colonnes sont numérotées. Pour identifier la cellule de ligne numéro i et colonne j on utilisera la notation : `l`i`c`j. Toute cellule ne commençant pas par le symbole spécial `=` sera considérée comme une cellule contenant du texte. Une cellule commençant par `=` sera interprétée comme un calcul à effectuer. Les différents calculs possibles sont :

- `[`*cel*`]` : permet d'obtenir la valeur de la cellule *cel* ;
- `+(`val_1`,`val_2`)` : somme de val_1 et val_2 où val_i est soit un valeur numérique, soit une référence à une cellule, soit le résultat d'un autre calcul (ex : `=+(3,+(l2c4,7))` effectue la somme de 3 avec le résultat de la somme de 7 et du contenu de la cellule de ligne numéro 2 et de colonne 4) ;
- `-(`val_1`,`val_2`)` : différence entre val_1 et val_2 ;

- `*(`val_1`,`val_2`)` : produit de val_1 par val_2 ;
- `/(`val_1`,`val_2`)` : quotient de val_1 et val_2 ;
- `^(`val_1`,`val_2`)` : élève val_1 à la puissance val_2 ;
- `ln(`val_1`)` : calcule le logarithme népérien de val_1 ;
- `e(`val_1`)` : calcule l'exponentiation de e à la puissance val_1 ;
- `sqrt(`val_1`)` : calcule la racine carrée de val_1 ;
- `somme(`cel_1`,`cel_2`)` : calcule la somme des cellules appartenant à l'intervalle de cel_1 à cel_2 (ex : `=somme(l1c1,l2c4)` effectue la somme des cellules `l1c1, l1c2, l1c3, l1c4, l2c1, l2c2, l2c3, l2c4` ;
- `moyenne(`cel_1`,`cel_2`)` : calcule la moyenne des cellules appartenant à l'intervalle de cel_1 à cel_2 ;
- `variance(`cel_1`,`cel_2`)` : calcule la variance des cellules appartenant à l'intervalle de cel_1 à cel_2 ;
- `ecarttype(`cel_1`,`cel_2`)` : calcule l'écart type des cellules appartenant à l'intervalle de cel_1 à cel_2 ;
- `mediane(`cel_1`,`cel_2`)` : calcule la valeur médiane des cellules appartenant à l'intervalle de cel_1 à cel_2 ;
- `min(`cel_1`,`cel_2`)` : fournit la plus petite valeur des cellules de l'intervalle de cel_1 à cel_2 ;
- `min(`cel_1`,`cel_2`)` : fournit la plus grande valeur des cellules de l'intervalle de cel_1 à cel_2 ;
- `concat(`val_1`,`val_2`)` : effectue la concaténation des deux chaînes de caractères val_1 et val_2 ;
- `length(`val`)` : fournit la longueur de la chaîne de caractères val ;
- `subsitute(`val_1`,`val_2`,`val_3`)` : remplace dans la chaîne de caractères val_1, la chaîne val_2 par val_3 ;
- `size(`val`)` : récupère la taille du fichier identifié par val ;
- `lines(`val`)` : récupère le nombre de lignes du fichier identifié par val ;
- `shell(`val`)` : remplace le contenu de la cellule par le résultat de la commande val ;
- `display(`cel_1`,`cel_2`)` : permet de spécifier un ensemble de cellules à afficher. Si cette commande apparaît dans la feuille de calculs, seul l'ensemble de cellules défini par cel_1 et cel_2 sera fourni en sortie. Si cette commande apparaît plusieurs fois, tous les ensembles définis seront fournis en sortie. Si cette commande n'apparaît pas dans la feuille de calculs (comportement par défaut) toute la feuille de calculs est considérée en sortie.

Ces fonctions ne constituent qu'une liste minimale des fonctionnalités à intégrer dans votre tableur. Vous êtes libres d'ajouter tout autre fonctionnalité que vous jugerez nécessaire.

Exemple de feuille de calculs :
Le séparateur de colonnes est `:` et le séparateur de lignes le retour chariot.

```
1:2:3:=somme(l1c1,l1c3)
4:=[l1c4]:7:=moyenne(l2c1,l2c3)
=+(l1c1,size(l3c3)):=shell(expr 12 + 89):/bin/ls:=/(l3c1,l3c2)
```

Après passage au tableur, la feuille résultat doit être :

```
1:2:3:6
4:6:7:5.66666666666666666
67669:101:/bin/ls:669.99009900990099009900
```

Lorsqu'une erreur est identifiée (ex : `=+(2,foo)`) votre programme indiquera la cellule incriminée et la raison de l'erreur.

Remarques :
- Attention, il se peut que le calcul de certaines cellules nécessite d'autres calculs préalables. Dans ce cas, votre programme devra très certainement se rappeler récursivement avec une feuille partiellement calculée pour obtenir le résultat final.
- Attention aux définitions récursives de cellules (exemple : `=[l1c2]:=[l1c1]`). Cette erreur n'est pas à gérer (sauf si cela vous amuse).

2.3 Opérations sur des réels en shell

La commande `expr` permet uniquement d'effectuer des calculs sur des entiers. Pour ce projet, vous devez disposer d'un outil de calcul sur des réels. Cet outil existe, il s'appelle `bc`. Cette commande lit les opérations à effectuer depuis l'entrée standard. En conséquence, pour effectuer un calcul en ligne de commande, il faudra utiliser une redirection comme l'illustre l'exemple ci-dessous :

```
hibou:~>echo "1.2 / 15" | bc -l
.08000000000000000000
```

Par ailleurs, pour effectuer des calculs avec des réels, il faut utiliser l'option `-l`. Cette commande intègre également les fonctionnalités nécessaires pour obtenir les valeurs de différentes fonctions usuelles (logarithme népérien, exponentielle, ...). Pour plus d'informations sur cette commande, vous êtes conviés à lire sa page `man :-)`.

3 Recommandations

3.1 Comparaison de chaînes de caractères

Pour comparer deux chaînes de caractères en *shell*, il est possible d'utiliser la commande `test` avec les opérateurs < et > bien que ces derniers ne soient pas référencés dans la page *man*.

Exemple :

```
#!/bin/sh
chaine1="foo"
chaine2="bar"
if test $chaine1 \< $chaine2
then
   echo OK
else
   echo KO
fi
if test $chaine1 \> $chaine2
then
   echo OK
else
   echo KO
fi
```

Le premier test produira l'affichage de `KO` et le second celui de `OK`. Remarquez que l'utilisation du *backslash* (\) est obligatoire sinon le *shell* interprète les caractères < et > comme des caractères de redirections.

3.2 Fonctions en *shell*

Il est possible de programmer des fonctions en *shell*. Une fonction du *shell* mémorise une série de commandes pour permettre une exécution ulterieure. Les déclarations des fonctions se font au début du programme et respectent la syntaxe suivante :

```
function  nom()
{
   instructions composant la fonction
}
```

Les fonctions sont exécutées dans le contexte de l'interpréteur en cours. On ne crée pas de nouveau processus pour interpréter une fonction, contrairement à l'exécution d'un script.

Les arguments d'une fonction sont placés dans les paramètres positionnels (`$1, $2, $3, ...`) durant son exécution. Le paramètre spécial `$#` est mis a jour. Le paramètre positionnel `$0` n'est pas modifié. À la fin de l'exécution de la fonction, les paramètres positionnels sont rétablis automatiquement.

Il est possible de déclarer des variables locales à une fonction qui ne conserveront pas leur valeur et qui ne seront donc pas visibles à l'extérieur de la fonction. Les variables locales d'une fonction peuvent être déclarées en utilisant la commande interne `local`. Autrement, les variables et leurs valeurs sont partagées entre la fonction et son appelant. C'est-à-dire que toutes les variables du script sont visibles dans les fonctions et toutes les variables non locales déclarées dans la fonction sont visibles à l'extérieur de la fonction.

Si la commande interne `return` est exécutée dans une fonction, celle-ci se termine et l'exécution reprend avec la commande suivant l'appel de fonction. Quand une fonction se termine, les paramètres positionnels et le paramètre spécial `$#` reprennent les valeurs qu'ils avaient avant l'appel de fonction.

Les fonctions peuvent être récursives. Aucune limite n'est imposée quant au nombre d'appels récursifs.

La valeur de retour de la fonction est celle de la dernière instruction exécutée.

L'appel d'une fonction se fait en indiquant son nom suivi de ses paramètres séparés.

Exemple :

```
#!/bin/sh

function calc()
{
  local op

  op=$2
  if test "$op" = "x"
  then
     res=‘expr $1 \* $3‘
  else
     res=‘expr $1 $op $3‘
  fi
}

if test $# -ne 3
then
  echo "Erreur 3 paramètres"
else
  case $2 in
  /) if test $3 -eq 0
     then
        echo "Division par zéro"
        exit 1
 else
calc $1 $2 $3
echo $res
     fi
     ;;
  +|-|x)
     calc $1 $2 $3
     echo $res
     ;;
  *)  echo "Opérateur inconnu"
     ;;
esac
fi
```

4 Informations pratiques

Ce projet est à réaliser en binôme.

Votre programme qui s'appellera `tableur` devra être envoyé par mail à l'adresse `chevallier@cril.fr` avant le mardi 03 décembre 2019 minuit. Un mail d' accusé de réception vous sera adressé en retour. Le sujet de votre mail devra être : `[PROJET-SHELL] nom_binome1-nom_binome2` et vous attacherez une archive de votre projet contenant le programme, des fichiers d'exemple et un readme détaillant les fonctionalités implémentées et les difficultés rencontrées. Enfin vous indiquerez la part (en pourcentage) de travail de chacun des binômes sur le projet.