TP3 : Formats de fichiers, interprétation, codage des entiers

CSH : Initiation au C et au shell
Première année

[CSH] **Préambule** : utilisez ***installeTP.sh*** pour créer un répertoire ***TP3***. Vous ferez ceci systématiquement au début de chacune des prochaines séances, ce ne sera plus indiqué dans les énoncés.

1 Compilation et exécution d'un programme C

[TP3] Écrivez un programme C qui calcule le maximum de 3 entiers saisis au clavier par l'utilisateur, et qui affiche le résultat. Compilez-le sous le nom ***max3***, et vérifiez que votre programme est correct. Quels tests effectuez-vous pour cela ?

2 Formats de fichiers

1. Tapez la commande `man ascii` pour connaître l'ensemble des codes Ascii. Quel est (en décimal) le code du caractère 'A' ? Et celui du caractère '5' ? Et celui du caractère *espace* (noté usuellement `SP`) ? À quel caractère correspond le code décimal 109 ? et 77 ? Notez l'absence des caractères 'è', 'ç', etc. Savez-vous comment sont codés ces caractères ?
2. La commande `file` prend un nom de fichier en argument et détermine le format du fichier, quand c'est possible. S'il s'agit d'un fichier de caractères Ascii, elle indique «ascii text», par exemple. Lorsque le format du fichier n'est pas identifié, elle indique «data» ou même parfois ne donne aucune réponse.
   En utilisant cette commande, déterminez les formats des fichiers de votre répertoire ***TP3***.Pourquoi le format du fichier ***annonce*** est-il «text» et non «ascii text» ? Ouvrez le fichier ***max3*** avec `emacs`. Commentez ce que vous voyez apparaître.
3. Le fichier ***exemple.html*** est un fichier de format *HTML*. Il s'agit d'un format de fichier permettant de représenter des documents structurés pouvant contenir des portions de texte avec des attributs particuliers d'affichage (gras, italique, souligné, etc). Ces portions de texte sont délimitées dans le fichier par des séquences de caractères particulières appelées *balises* (comme `<b>` et `</b>`). Visualisez ce document en utilisant la commande `firefox exemple.html &`
   Sans quitter `firefox`, examinez maintenant le contenu du fichier ***exemple.html*** avec la commande `less` dans la fenêtre Unix. En comparant les contenus des deux fenêtres, déterminez quelles sont les balises délimitant respectivement des portions de texte écrites en **gras**, souligné, et *italique*. Éditez le fichier ***exemple.html*** avec `emacs`. Modifiez-le de manière à inverser les attributs d'affichage gras et italique (ce qui est gras doit devenir italique et réciproquement). Vérifiez (avec `firefox`) que vos modifications sont correctes. En procédant de la même manière, modifiez le document pour que les mots soulignés apparaissent désormais en gras et en italique. Après avoir vérifié que le document obtenu correspond à ce qui est attendu, quittez `firefox`.
4. Le fichier ***TP-03.ps*** est un fichier de type *PostScript*. Ce format de fichier permet de représenter des documents comportant des directives d'affichages évoluées (tracer des lignes, des courbes, utiliser différentes polices de caractères, etc). Visualisez ce document avec la commande `evince TP-03.ps`. Quittez **evince** (choix *Quit* dans le menu *File*) et éditez le fichier ***tp3.ps***. Dans les premières lignes du fichier, remplacez la ligne `%%PageOrder : Ascend` par la ligne `%%PageOrder : Descend`. Exécutez la commande `evince TP-03.ps` pour observer les modifications d'affichage. Quittez `evince`.

3 Taille des entiers, temps d'exécution, débordement

[TP3] Lisez le contenu du fichier ***debordechar.c***, et vérifiez que vous comprenez ce qui va se passer lors de l'exécution. Compilez-le sous le nom ***debordechar***, et exécutez-le. Quelle est la taille en octets et en bits d'une variable de type `char` ? Quel est le plus grand entier représentable avec le type `char` ?

Créez une copie de ***debordechar.c*** sous le nom ***debordeshort.c***. Modifiez ce fichier afin qu'il serve à tester des variables de type `short int` (au lieu de `char`), et répondez aux mêmes questions que ci-dessus, pour le type `short`.

Effectuez la même expérience pour le type `int` (donnez à l'exécutable le nom ***debordeint***).

Par combien (environ) est multiplié le temps d'exécution entre `debordechar` et `debordeint` ?

Si l'on définit un type d'entiers représentés sur 8 octets, et qu'on effectue la même expérience, combien de temps (environ) faudra-t-il attendre pour voir s'afficher le résultat ? (extrapolez les résultats précédents)

4 Shell

[TP3] Lisez le texte du programme ***max2.sh***, qui affiche le maximum de 2 entiers donnés en arguments. Vérifiez qu'il s'exécute bien pour toutes les configurations possibles.

Dans le programme ***max2err.sh***, une erreur a été commise : laquelle ? *(indication : utilisez la commande* ***`diff`****)*. Cependant, ce programme fonctionne bien pour certaines configurations des données, lesquelles ? En quoi ceci a-t-il un rapport avec la terminologie que nous utilisons : «interpréteur de commandes» ?

Faites une erreur du même genre dans votre programme ***max3.c***. Que se passe-t-il ? Le programme peut-il s'exécuter dans certains cas seulement ?

Il existe plusieurs langages *shell* ayant chacun leur interpréteur de commandes. Celui que vous avez utilisé jusqu'à présent est *csh*. Un autre langage shell répandu est *bash*. Sa syntaxe diffère légèrement de celle de *sh*, notamment pour ce qui concerne les instructions composées (if, while ...). On précise habituellement le langage utilisé dans la première ligne du programme : `#!/bin/sh`, comme vous l'avez sans doute remarqué dans tous les programmes shell que vous avez utilisés.

Faites les expériences suivantes :
- Supprimez la première ligne de ***max2.sh*** et exécutez-le ;
- Mettez en première ligne `#!/bin/tcsh` et exécutez-le ;

5 Exercices pour la prochaine fois

1. En vous inspirant de ***max2.sh***, écrivez un programme ***max3.sh*** permettant de calculer le maximum de 3 entiers donnés en arguments.
2. Dans votre répertoire ***TP2***, quel est à votre avis l'effet des commandes suivantes ?

```
1) date  >! une_date  >! instant_suivant
2) une_date  < date  >! instant_suivant
3) instant_suivant.c  < une_date
4) instant_suivant  | date
5) date >! instant_suivant < une_date
6) date  | instant_suivant >! result
```