TP3 : Interprétation, tableaux, structures

CSH : Initiation au C et au shell

Première année

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.

★ Exercice 1: Shell

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 ;

▷ **Question 1:** En vous inspirant de **max2.sh**, écrivez un programme ***max3.sh*** permettant de calculer le maximum de 3 entiers donnés en arguments.

▷ **Question 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
```

★ Exercice 2: Tableaux en C : le triangle de Pascal

On souhaite écrire un programme qui construit le triangle de Pascal de degré N et l'affiche jusqu'à la diagonale incluse.

Les éléments du triangle de Pascal se calculent comme suit (si P est la matrice mémorisant le triangle) : $P_{i,j} = P_{i-1,j} + P_{i-1,j-1}$

Voici quelques exemples d'affichage pour différentes valeurs de N.

Affichage quand N vaut 6

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10 10 5  1
1  6  15 20 15 6  1
```

Affichage quand N vaut 7

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10 10 5  1
1  6  15 20 15 6  1
1  7  21 35 35 21 7  1
```

▷ **Question 1:** Écrivez ce programme.

★ Exercice 3: Arguments de la ligne de commande : dessin de motifs.

On souhaite écrire un programme qui affiche des motifs répétitifs sur la sortie standard.

Les arguments de la lignes de commande sont les nombres de lignes et de colonnes qui mémorisent le nombre en lignes et en colonnes de motifs à tracer. Le motif dessiné est composé d'une alternance de deux motifs initialisés avec les valeurs reçues en arguments de la commande.

La commande comprend donc 4 arguments. Voici quelques exemples d'exécution de ce programme :

Premier exemple

```
$ ./motif 3 15 ___/ \\___
___/\______/\______/\______/\______/\______/\______/\______/
\______/\______/\______/\______/\______/\______/\______/\___
___/\______/\______/\______/\______/\______/\______/\______/
$
```

Deuxième exemple

```
$ ./motif 4 10 __] [__
__][____][____][____][____][__
 [____][____][____][____][____]
__][____][____][____][____][__
 [____][____][____][____][____]
$
```

Troisième exemple

```
$ ./motif 4 20 '\__' '/ '
\__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/
/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__
\__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/
/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__/ \__
$
```

Indication : Vous pouvez récupérer les motifs de la ligne de commande à l'aide de l'instruction `char *motif[] = {argv[3] , argv[4] };`

★ **Exercice 4: Structures en C : le programme annuaire**

On souhaite créer un programme en C gérant un annuaire très simplifié qui associe à un nom de personne un numéro de téléphone.

▷ **Question 1:** Créer une structure Personne pouvant contenir ces informations (nom et téléphone). Le nom peut contenir 32 caractères et le numéro 16 caractères.

▷ **Question 2:** Créer une nouvelle structure qui va représenter le carnet d'adresses. Cette structure Carnet contiendra un tableau de 20 Personne et un compteur indiquant le nombre de personnes dans le tableau.

▷ **Question 3:** Créer ensuite une fonction qui renvoie une structure Personne en prenant en argument un nom et un téléphone.

▷ **Question 4:** Rajouter une fonction qui affiche les informations contenues dans la structure Personne passée en argument.

▷ **Question 5:** Créer une fonction qui ajoute une personne dans un carnet.

▷ **Question 6:** Créer une fonction qui affiche un carnet.

▷ **Question 7:** À partir des étapes précédentes, faire programme gérant un carnet d'adresse. Créer un menu qui propose d'ajouter une nouvelle personne, d'afficher le carnet ou de quitter.

Une extension possible serait d'ajouter une fonction de sauvegarde de l'annuaire dans un fichier, et de faire en sorte que les données sauvegardées puissent être lues automatiquement au démarrage.