TP3 – Les langages de programmation

Perl et les expressions régulières

1 Introduction

Perl est un langage de script interprété. Il est particulièrement adapté à la création de programmes manipulant des fichiers, des chaînes de caractères et des commandes systèmes. Un de ces domaines de prédilection est donc l'administration système.

Sa syntaxe est souvent relativement proche de celle du langage C avec des emprunts aux langages des *shells*.

2 Bonjour le monde

2.1 Exécution

Un programme `helloWorld.pl` écrit en perl est appelé un script. Il peut être exécuté par l'interpreteur perl de deux manières :
- lancement explicite de l'interpréteur `perl helloWorld.pl`,
- lancement implicite `helloWorld.pl` à deux conditions : 1) le fichier est exécutable 2) sa première ligne commence par `#!` et indique le chemin d'accès à l'interpréteur.

Par exemple :

```
#!/usr/bin/perl
# Les commentaires commencent par un #.
print("Hello");
print " World\n";
```

affiche la fameuse phrase. On voit alors deux appels de fonction, dont un semblable à un appel C ou Java. En Perl, les paranthèses autour des arguments ne sont pas toujours indispensables.

On peut rendre le programme plus sopistiqué en utilisant les arguments de la ligne de commande.

```
#!/usr/bin/perl -w
```

```
# @ARGV est le tableau des arguments de la ligne de commande
if (scalar(@ARGV) == 0) {
  ; # On ne fait rien
} else {
  $name = $ARGV[0];
  print "Hello $name\n";
}
```

Dans cet exemple, on voit apparaître deux des types de variables de Perl : les variables scalaires qui commence par un `$` et les variables de type tableau (ou liste) qui commence par un `@`. Perl n'étant pas un langage typé statiquement, il n'est pas necéssaire de déclarer des variables.

Les principales opérations et structures de contrôles sont (quasi) identiques à C ou Java. La principale différence résidant dans le caractère obligatoire des accolades.

2.2 Tableaux

Les tableaux sont des listes de données scalaires. Les tableaux constants sont déclarés à l'aide de parenthèses (1,2,4, "e").

On peut obtenir la taille d'un tableau d'au moins trois façons :
– $#ARGV qui est l'indice du dernier élément du tableau,
– scalar(@ARGV)
– @ARGV + 0
Il est possible d'affecter plusieurs variables en même temps par la syntaxe des tableaux :

```
($first,$second,$third) = @liste;
($first,$second,$third,@rest) = @liste;
```

Si le dernier élément est une liste, elle va contenir tous les autres éléments.

Exercice 1

Les tableaux peuvent être parcourus comme en C / Java mais aussi par des opérateurs. Tester puis expliquer les fonctions suivantes :

1. `push` :
2. `pop` :
3. `shift` :
4. `unshift` :
5. `sort` :
6. `reverse` :

2.3 Les chaînes de caractères

Les chaînes de caractères contenues entre " et " de Perl sont extrapolées, c'est-à-dire que toute variable scalaire est remplacée par sa valeur. Les opérateurs suivants sont disponibles sur les chaînes de caractères :

1. . : concaténation de deux chaînes
2. `eq` : égalité de deux chaînes
3. `ne` : non égalité de deux chaînes
4. `lt` : la première chaîne est plus petite strictement que la deuxième
5. `le` : la première chaîne est plus petite ou égale à la deuxième
6. `gt` : la première chaîne est plus grande strictement que la deuxième
7. `ge` : la première chaîne est plus grande ou égale à la deuxième
8. `chop`() : retire le dernier caractère de la chaîne passée en argument et le retourne
9. `chomp`() : retire le dernier caractère de la chaîne passée en argument si c'est une fin de ligne (*i.e.* le contenu de la variable `$/`) ; elle retourne le nombre de caractères supprimés
10. `length`() : retourne la taille en nombre de caractères de son argument
11. ...(voir la doc ...)

Si vous voulez des chaînes de caractères plus sophistiquées, allez voir dans la documentation.

Remarque :

Nous avons ajouté l'option `-w` *à la ligne de commande. Pour permettre de mieux surveiller l'exécution de programme (surtout valable pour les phases de mise au point).*

Exercice 2

Modifier le script précédent pour qu'il salut toutes les personnes dont le nom est fourni par la ligne de commande.

Il est également possible de parcourir un tableau par l'instruction `foreach` dont la syntaxe est la suivante :

```
#!/usr/bin/perl -w
@list = (1,2,4,7);
print "La liste contient :\n";
foreach $elem (@list) {
  print "\t$elem\n";
}
```

Exercice 3

Modifier le script précédent pour qu'il utilise une boucle `foreach`.

Enfin, il est également possible de manipuler un tableau par l'opérateur `shift` qui retire le premier élément du tableau et le retourne en résultat.

Exercice 4

Modifier le script précédent pour qu'il utilise l'opérateur `shift`.

2.4 Variable par défaut

Si on ne fournit pas d'argument à une fonction elle prend en paramètre la variable `$_`. De même, si le résultat d'une fonction n'est pas réutilisé, il est placé dans cette variable.

Exercice 5

Pour vous en convaincre modifier la version de l'exercice 3 pour qu'il utilise `$_`.

2.5 Entrée sortie

La principale fonction d'affichage a déjà été vu, il s'agit de `print`. Si son argument est une liste, elle affiche la concaténation des éléments de la liste.

Pour récupere du contenu sur l'entrée standard, on utilise la syntaxe <STDIN>. Si le résulta est placé dans une variable scalaire (qui commence par un $, une ligne est lue et placée dans la variable. Si le résultat est placé dans une liste toutes les lignes sont lues (jusqu'au carctère `control-D`).

Exercice 6

Modifier le script `helloWorld` pour que les noms ne viennent pas de la ligne de commande mais soient récupérés par l'entrée standard.

3 Gérer un stock

3.1 Les dictionnaires

Les dictionnaires sont aussi appelés hachages ou tableaux associatifs. Il s'agit d'une liste de couple (clé,contenu). Le nom d'une variable de type dictionnaire commence par un %. Par exemple, on peut définir le dictionnaire suivant :

```
%traduction = (
  'monday' => 'lundi',
  'tuesday' => 'mardi',
  'wednesday' => 'mercredi',
  'thursday' => 'jeudi',
  'friday' => 'vendredi',
  'saturday' => 'samedi',
  'sunday' => 'dimanche'
);
```

On accède alors à un contenu par l'intermédiaire de sa clé :

```
$traduction{'friday'}
```

On peut ajouter un couple (ou mettre à jour un couple existant par :

```
$traduction{'holiday'}='vacances';
$traduction{'friday'}='poisson';
```

Exercice 7

Tester puis expliquer le code suivant :

```
%hash = ('un',1,'deux',2,' trois ' ,3, 'quatre' ,4);
@liste_cle=keys(%hash);
@liste_val=values(%hash);
($cle,$valeur)=each(%hash);
delete($hash{'deux'});
```

3.2 Les entrées sorties dans des fichiers

3.2.1 Ouverture

L'ouverture permet de créer un descripteur de fichier et de l'associer à un fichier, il faut fournir le nom du fichier ainsi que le mode d'écriture : `open (FIC,'MODEfichier');` où la signification de MODE est décrit dans le tableau ci-dessous.

MODE	lecture	écriture	ajout	créer	vider
	OK	Nok	Nok	Nok	Nok
<	OK	Nok	Nok	Nok	Nok
>	Nok	OK	Nok	OK	OK
>>	Nok	OK	OK	OK	Nok
+<	OK	OK	Nok	Nok	Nok
+>	OK	OK	Nok	OK	OK
+>>	OK	OK	OK	OK	Nok

L'ouverture d'un fichier est souvent combiné par un ou (||) à la fonction `die` qui permet d'afficher un message (son argument) et quitter le programme :

```
$nomfichier = 'MonFichier';
open(FIC,$nomfichier) || die("Problème d'ouverture de [$nomfichier]\n");
```

3.2.2 Lecture et écriture

Les accès en lecture et écriture sont similaires aux entrées sorties clavier. L'opérateur *angle*, <FIC> permet de lire une ligne dans le fichier lié au descripteur FIC. La fonction `print FIC ()` permet d'écrire dans le fichier.

3.2.3 Fermeture

La fermeture d'un fichier se fait par la fonction close : `close(FIC);`.

Au final, une utilisation de fichier sera de la forme :

```
$nomfichier = 'MonFichier';
open(FIC,$nomfichier) || die("Problème d'ouverture de [$nomfichier]\n");
while (<FIC>) {
   ...
```

```
  print FIC "...";
}
close(FIC);
```

3.3 Les fonctions

La syntaxe qui permet de déclarer un sous-programme en Perl est la suivante :

```
sub NomDeLaFonction {
  instruction1;
  instuction2;
}
```

Le retour d'un résultat se fait par l'instruction `return` comme en C / Java.

Les arguments d'une fonction sont passés par valeur et sont récupérés dans le tableau `@_`. Ces paramètres peuvent alors être utilisés dans le corps de la fonction de diverses manières qui correspondent aux différentes manières de parcourir une liste.

1. par les parenthèses :

   ```
   my ($var1,$var2) = @_;
   ```

2. par les indices :

   ```
   my $var1=$_[0]; my $var2=$_[1];
   ```

3. par l'instruction `foreach` :

   ```
   foreach (@_) { ... }
   ```

4. par l'opérateur `shift` :

   ```
   my $var1=shift; my $var2=shift;
   ```

5. ...

L'utilisation du mot-clé `my` à la première utilisation d'une variable rend cette variable locale à la fonction ou au bloc dans lequel elle a été « déclarée ». L'appel d'une fonction se fait comme en C.

Il est également possible de passer des arguments par références. Pour obtenir une référence sur une variable, il suffit de précéder son nom par un *backslash* :

```
$s=4;
@tab=(1,2,3,4,5,6);
%h=("a" => 1,"b" => 2,"c" => 3);
$refs = \$s;
$reftab = \@tab;
$refh = \%h;
```

Pour manipuler la variable référencée, il faut alors faire précéder son identificateur de `$`,`@` ou % selon le type de variable qu'elle référence.

L'opérateur −> permet de manipuler les références vers des tableaux ou des dictionnaires :

```
%traduction = (
  'monday' => 'lundi',
  'tuesday' => 'mardi',
  'wednesday' => 'mercredi',
  'thursday' => 'jeudi',
  'friday' => 'vendredi',
  'saturday' => 'samedi',
  'sunday' => 'dimanche'
);

$reftrad = \%traduction;
print($reftrad,"\n");
print("$$reftrad{'friday'}\n");
$traduction{'friday'}='poisson';
print("$$reftrad{'friday'}\n");
print("$reftrad->{'friday'}\n");
```

3.4 Les boissons

Exercice 8

Il s'agit de construire un script permettant de gérer un stock de boisson. Chaque boisson aura un nom et une quantité en stock associée.

▷ **Question 8.1 :**
Écrire une fonction qui permet de remplir le stock de manière interactive.

▷ **Question 8.2 :**
Écrire deux fonctions d'entrée sortie, une qui permet de sauvegarder l'état du stock et une autre qui permet de le relire.

▷ **Question 8.3 :**
Réaliser un script qui :

1. **commence par récupèrer le stock dans un fichier (le nom est fourni en paramètre de la ligne de commande) ;**
2. **permet, au choix, de quitter, d'afficher l'état du stock ou d'ajouter des boissons dans le stock ;**
3. **lors de la terminaison, l'état du stock est sauvegardé dans le fichier.**

4 Les expressions régulières

4.1 Syntaxe

Les principaux éléments d'une expression régulière en Perl sont les suivants :
– . indique un caractère quelconque sauf la fin de ligne.
– [abc] indique l'un des cractères entre crochets.
– [^abc] indique tout caractère sauf ceux entre crochets.

– [a−z] indique tout carctère de l'intervalle.
– \d indique un chiffre (=[0−9]).
– \w indique une lettre, chiffre, ou underscore (=[a−zA−Z_0−9]).
– \s indique un blanc (=[\t\n\r\f]).
– \D, \W et \S sont les versions négatives de ces trois dernières expressions.
– ∗ signifie entre 0 et n fois le caractère précédent.
– + signifie entre 1 et n fois le caractère précédent.
– ? signifie 0 ou une fois le caractère précédent.
– ^ indique le début de ligne.
– $ marque la fin de ligne.
– | indique la diusjonction (ou) de deux expressions logiques.

Pour reconnaitre un caractère spécial, il faut le protéger par un \.

4.2 La recherche

En plaçant une expression régulière entre slashes, on recherche son existence dans la variable $_.L'opérateur retourne vrai ou faux. S'il retourne vrai, la variable $& contiendra l'occurence trouvée, la variable $‘ contient ce qui se trouve avant l'occurence et la variable $’ ce qui se trouve après l'occurence. Par exemple :

```
$_ = "Il fait beau";
if(/fait/){
  print('$& = ',$&,' $'."' = ",$','    $‘ = ',$‘,"\n");
}
```

Provoque l'afichage suivant :

```
$& = fait  $' =  beau   $‘ = Il
```

L'opérateur {} permet de sélectionner un nombre exact d'occurences. Par exemple, \d{1,5} recherche un nombre composé d'un à cinq chiffres.

Il est également possible de mémoriser des occurrences, pour cela, on parenthèse une partie de l'expression régulière. La valeur que cette portion de l'expression régulière reconnaitra sera accessible dans la suite de l'expression régulière par la syntaxe *num*. La valeur *num* étant le numéro d'ordre du parenthésage concerné. Par Exemple, pour reconnaître des balises de HTML, on peut utiliser l'expression régulière suivante :

```
if (/<(.*)>.*<\/\1>/) printf ("balise : $& \n");
```

Les résultats de la mise en correspondance des portions entre parenthèse sont également disponible en dehors de l'expression régulière dans les variables $1, ...

La recherche d'occurences peut également se faire sur une variable quelconque par l'opérateur =~.

Exercice 9

Écrire un script qui vérifie qu'une chaîne de caractère saisie peut correspondre à une plaque d'immatriculation d'un véhicule français.

4.3 L'opérateur de substitution

L'opérateur `s/regexp/chaine/param` appliquer à une variable (comme une recherche d'occurrence) permet de substituer les occurrence reconnue par la chaîne fournie. La partie paramètre permet de spécifier des options sur l'opération de substitution comme le fait de substituer toute les occurrences (g) ou le de ne pas tenir compte de la casse (i).

4.4 Les opérateurs `split` et `join`

La fonction `split` permet de découper une chaîne de caractères en une liste de chaînes. Elle est utilise comme délimiteur une expression régulière. Par défaut, le séparateur est l'espace.

La fonction `join` permet d'effectuer l'opération inverse, et donc de recoller des chaînes.

Exercice 10

Écrire un script qui ouvre les fichiers (dont le nom est fourni en ligne de commande) et calcul leurs nombre de lignes, de mots et de caracrtères.

5 Un peu de système

5.1 Exécution d'une commande système

L'exécution d'une commande système se fait comme en Shell : ‘`ls`‘

Exercice 11

Construire un script qui affiche le *login* de tous les utilisateurs du système ainsi que leur *shell* par défaut.

5.2 Manipulation de fichiers

Les fonctions `opendir` et `readdir` permettent de parcourir un répertoire. Si elle est utilisée dans un contexte de liste, `readdir` renvoie la liste des fichiers du répertoire. Dans un contexte scalaire, elle renvoie le nom du prochain fichier (ou `undef` à la fin).

Exercice 12

Construire un script qui affiche la liste des fichiers de tous les repertoire passés en ligne de commande.

Les fonctions `mkdir`, `rmdir` et `chdir` permettent respectivement de créer un repertoire, de détruire un repertoire ou de changer de repertoire courant.

Les fonctions `unlink`, `rename` et `chmod` permettent respectivement de détruire un fichier, de renommer un fichier ou de changer les permissions d'un fichier.

Les opérateurs de test suivants sont définis :

- `-r` fichier ou répertoire accessible en lecture
- `-w` fichier ou répertoire accessible en écriture
- `-e` fichier ou répertoire existant
- `-x` fichier exécutable
- `-z` fichier existant mais vide
- `-s` fichier ou répertoire non vide, la valeur retournée est la taille en octet
- `-f` fichier normal (répertoire et spéciaux exclus)
- `-d` répertoire
- `-T` fichier texte

Exercice 13

Écrire un script pour détruire récursivement un ensemble de repertoires.

Exercice 14

Écrire un script pour copier récursivement le contenu d'un repertoire dans un autre.