

Méthodes Empiriques et Langages de Script

TP 6

G. A. Musillo
musillo4@etu.unige.ch

October 24, 2006

1 Exercice 1: un concordancier

Un concordancier est un outil d'exploration de données textuelles, qui, étant donné un corpus de textes et un mot, extrait les occurrences de ce mot ainsi que les contextes gauche et droit dans lesquels ce mot apparaît.

On vous demande de programmer un tel concordancier. Si on fournissait, par exemple, à votre concordancier le paragraphe précédent et le mot `un`, il afficherait le résultat suivant:

```
un concordancier est un outil d'expl...
un concordancier est un outil d'exploration de donne...
..., qui, etant donne un corpus de textes et un mot,...
...un corpus de textes et un mot, extrait les occur...
```

2 Exercice 2: les structures complexes et les références anonymes

Au moyen de *références*, on peut créer en Perl des structures arbitrairement complexes. Une référence sur une variable scalaire, une variable de tableau ou une variable de tableau associatif est obtenue en préfixant le symbole `'\'` à cette variable. Voici quelques exemples:

```
$i = 3.14159;
@a = ('log', 'exp', 'DIV');
%h = ('Google' => 1, 'Microsoft' => 0);
$ri = \$i; # $ri pointe sur $i
$ra = \@a; # $ra pointe sur @a
$rh = \%h; # $rh pointe sur %h
```

On peut déréférencer un scalaire en mettant entre accolades la référence scalaire et en préfixant le type (`$`, `@` ou `%`) de la variable référencée. Par

exemple:

```
$dri = ${$ri}; # $dri prend pour valeur le scalaire $i
@dra = @{$ra}; # @dra prend pour valeur le tableau @a
%drh = %{$rh}; # %drh prend pour valeur le tableau %h
```

Il y a deux façons d'accéder aux objets enregistrés dans un tableau associatif ou non référencé par un scalaire:

```
1)
$k = ${$ri}[1]; # $k vaut le deuxieme scalaire du tableau @{$ri}
$l = ${$rh}{'Google'}; # $l vaut la valeur de la cle 'Google' du hash %{$rh}
2)
$k = $ri->[1]; # $k vaut le deuxieme scalaire du tableau @{$ri}
$l = $rh->{'Google'}; # $l vaut la valeur de la cle 'Google' du hash %{$rh}
```

Perl vous permet de créer des références sur des tableaux ou des hashes anonymes. Une référence sur un tableau anonyme est obtenue en plaçant les éléments de ce tableau entre '[' et ']'. Une référence sur un tableau associatif anonyme est obtenue en plaçant les paires *clé/valeur* de ce tableau associatif entre '{' et '}'. Voici deux exemples de références anonymes:

```
$ranoa = ['log', 'exp', 'DIV'];
# $ranoa pointe sur le tableau anonyme ('log', 'exp', 'DIV')
$ranoh = {'Google' => 1, 'Microsoft' => 0};
# $ranoh pointe sur le hash anonyme ('Google' => 1, 'Microsoft' => 0)
```

Pour réaliser cet exercice, considérez les déclarations ci-dessous:

```
$pi = 3.14159;
@a = ('log', 'exp', 'DIV');
%h = ('Google' => 1, 'Microsoft' => 0);
$rh = {
    'pi' => \$pi;
    'ra' => \@a;
    'rh' => \%h;
};
```

On vous demande d'écrire, de deux façons distinctes, le code qui affiche sur la sortie standard la valeur:

1. de la clé 'pi' de la référence \$rh
2. du second élément du tableau référencé associé a la cle 'ra'
3. de la clé 'Google' du hash référencé associé a la cle 'rh'

3 Exercice 3: Un module pour les anneaux

Un anneau ou une queue circulaire (en anglais *ring buffer*) est une structure de données de type *FIFO*: les données sont retirées dans l'ordre dans lequel elles ont été insérées. Ces structures circulaires sont typiquement utilisées pour transmettre des données entre deux processus asynchrones, un processus *producteur* et un processus *consommateur*. Le processus *producteur* produit des données insérées en queue d'anneau et le processus *consommateur* consomme les données en tête d'anneau.

Une queue circulaire est formée d'un tableau dont la capacité a été fixée. Les scalaires de ce tableau sont des références sur des hashes dont les clés sont `'_object'` et `'_next'`. La valeur d'une clé `'_object'` est l'objet que le client enregistre dans l'anneau. La valeur d'une clé `'_next'` est l'indice du prochain scalaire du tableau. Par exemple, si l'anneau est formé d'un tableau `@buffer` de 3 scalaires, alors `$buffer[0]->{'_next'}` vaut 1, `$buffer[1]->{'_next'}` vaut 2, et, puisque la queue est circulaire, `$buffer[2]->{'_next'}` vaut 0.

Pour insérer des éléments en queue ou retirer des éléments en tête, on utilise deux scalaires `$head` et `$tail`, qui sont, respectivement, les indices du premier et du dernier élément de la queue. Le scalaire `$head` est incrémenté lorsqu'un élément est retiré, et le scalaire `$tail` est incrémenté lorsqu'un élément est inséré. Bien entendu, aucun élément ne peut être retiré d'un anneau vide et aucun élément ne peut être inséré dans un anneau plein.

On vous demande de compléter le module ci-dessous et de coder un programme qui teste votre module complété:

```
#!/usr/bin/perl
use strict; use warnings;
package ModuleRingBuffer;
BEGIN {
    sub init($); # initialize ring buffer with given capacity
    sub store($); # store a scalar argument in ring buffer
    sub retrieve; # return element at front of buffer
    sub printringb; # print ring buffer
    sub full; # decide whether buffer is full or not
    sub empty; # decide whether buffer is empty or not
    use Exporter;
    our @ISA = qw/Exporter/;
    our @EXPORT = qw/init store retrieve printringb full empty/; # exported subs
}
# ...
END { }
1;
```

4 Exercice 4: l'évaluation de références anonymes, les expressions régulières et les modules

PERL permet de créer des références sur des objets anonymes. Par exemple, le code suivant

```
$r_ano_array = [ 'a', 'b', 'c' ];
```

crée un tableau ('a', 'b', 'c') et retourne une référence affectée à `$r_ano_array`. PERL interprète donc `['a', 'b', 'c']` comme une référence sur un tableau. On dit que ce tableau est anonyme, car on ne peut y accéder que *via* la référence `$r_ano_array`.

On peut former des structures anonymes plus complexes: il suffit qu'un élément du tableau soit une référence sur un tableau anonyme. Par exemple, le code suivant

```
$r_ano_array = ['*', 'a', ['- ', 'b', 'c', 'd']];
```

crée une structure enchassée qu'on peut interpréter comme un arbre et sur laquelle le scalaire `$r_ano_array` pointe.

La fonction `eval` de PERL est magique: elle interprète son argument, qu'elle considère comme du code PERL, et retourne sa valeur. Elle permet donc d'appeler l'interpréteur PERL dans le corps d'un programme. Il s'ensuit que si on lui passait une chaîne telle que

```
“['*', 'a', ['- ', 'b', 'c', 'd']]”
```

elle retournerait une référence sur la structure anonyme

```
['*', 'a', ['- ', 'b', 'c', 'd']].
```

On vous demande de coder un programme

- qui prend en entrée la représentation parenthésée d'un arbre syntaxique (vous trouverez dans le fichier `syms.dat` quelques exemples à traiter),
- qui la transforme au moyen d'expressions régulières de telle sorte qu'elle puisse être interprétée par la fonction `eval` comme une structure complexe
- et, enfin, qui affiche cette structure complexe en appelant la fonction `print_tree` que le module `PrettyPrinting.pm` exporte.