Support de cours n°2
Structures complexes

DESS TEXTE

Introduction

En perl, il est possible de travailler avec divers types de structures : des scalaires, des listes, et des tableaux associatifs. En composant ces objets entre eux, nous pouvons disposer de structures plus complexes telles que listes de listes (pour stocker les mots d'un fichier par exemple), ou encore hachages de listes (cf exercice sur le classement des villes par pays). Cela passe par l'utilisation de références, i.e. d'adresses de structures (voir chapitre 2).
L'intérêt des structures complexes réside dans le fait qu'elles permettent de représenter divers types d'objets tels que les arbres d'analyses, les structures de traits, etc.
Dans le présent document, nous allons, à partir d'exemples, aborder d'autres structures complexes telles que listes de hachages et hachages de hachages. Nous verrons notamment comment déclarer de telles structures, comment les générer à partir d'un fichier, et enfin comment y accéder (pour afficher leur contenu par exemple).

1 Les hachages de listes

Un exemple d'utilisation des hachages de liste est la représentation de l'ensemble des prononciations ayant plusieurs graphies.
On considère par exemple les différentes orthographes des sons "a", "alé", etc :

§a	=>	chat, chats, chas
alé	=>	aller,allée

1.1 Déclaration

Nous allons stocker dans un tableau associatif, les différentes prononciations, et la liste des orthographes possibles (en fait une référence à cette liste). La première façon de faire est d'écrire directement le contenu du hachage. La syntaxe perl est la suivante (on remarque l'emploi des [] pour créer une référence sur un tableau) :

```
%HoL = (
§a => [ "chat", "chats", "chas" ],
```

```
alé => [ "aller", "allée" ],
so => [ "sot", "sots", "seau", "sceau", "saut", "seaux", "sceaux", "sauts"]
);
```

1.2 Generation

Nous pouvons aussi récupérer ces informations dans un fichier texte. Supposons que nous disposons d'un fichier ayant les entrées suivantes :

§a : chat chats chas
alé : aller allée

Il faut alors parcourir le fichier et découper les prononciations et orthographes associées, au moyen d'une expression régulière :

```
while ($ligne= <IN> ) {
  my($phon,$graph);
  my(@lgraph);
  chop($ligne);
  ($phon, $graph) = split(/:/,$a);
  @lgraph = split(/ /, $graph);
  $HoL{$phon} = [ @lgraph ];
}
```

Ce qui s'écrit encore (sans variables temporaires) :

```
my($ligne);
while ($ligne= <IN> ) {
  chop($ligne);
  $a=~/(.*):(.*)/;
  $HoL{$1} = [ split(/ /,$2) ];
}
```

Si par la suite, nous souhaitons ajouter de nouvelles graphies à une liste existante, nous pouvons utiliser la fonction `push` pour placer des éléments en fin de la liste dont l'adresse est associée à la prononciation considérée :

```
push(@{ $HoL{"alé"} }, "allais", "allait", "allaient");
```

1.3 Accès et Affichage

Voici comment accéder à un élément directement :

```
${$HoL{alé}}[0] = "aller";
```

Si nous souhaitons afficher l'ensemble des graphies associées à chaque prononciation, il nous faut parcourir les clés de la table de hachage au moyen d'une boucle (foreach par exemple) :

```
foreach $phon ( keys %HoL ) {
print "$phon: @{ $HoL{$phon} }\n"
}
```

Ici, nous parcourons chaque prononciation, et nous affichons l'ensemble de la liste de graphies associées (après déréférencement). Si nous voulions afficher les graphies une à une, il nous faudrait une seconde boucle appelée pour chaque prononciation, et parcourant le tableau des graphies :

```
foreach $phon ( keys %HoL ) {
  print "$phon : ";
  for($i=0;$i<=$#{ $HoL{$phon} };$i++) {
    print " $i = ${$HoL{$phon}}[$i]";
  }
  print "\n";
}
```

2 Les listes de hachages

Les listes de hachages peuvent être employées, par exemple, pour représenter un arbre d'analyse d'une phrase à partir d'une grammaire hors-contexte. On imagine l'arbre d'analyse de profondeur 2 suivant :

s	->	np vp
np	->	det n
vp	->	v np

2.1 Déclaration

Pour déclarer une liste de tableaux associatifs, on utilise la syntaxe perl suivante :

```
@LoH = (
{  "pere" => "s",
   "filsgauche" => "np",
   "filsdroit" => "vp"  },
{  "pere" => "np",
   "filsgauche" => "det",
   "filsdroit" => "n"   },
{  "pere" => "vp",
   "filsgauche" => "v",
   "filsdroit" => "np"  });
```

2.2 Generation

Comme précédemment, il est possible de récupérer les données de la liste de hachages dans un fichier texte. Supposons que les lignes de notre fichier ont le format suivant :

pere=s filsgauche=np filsdroit=vp

L'idée est de parcourir le fichier ligne par ligne, et de découper celle-ci via split pour récupérer les 3 données. Dans le bloc d'instructions du `while`, on déclare une référence sur un tableau associatif anonyme au moyen de {} :

```
while ($ligne= <IN> ) {
  my($arbre,$noeud,$att,$val);
  my @Tmp;
  chop($ligne);
  $arbre = {};
  @Tmp = split(/ /,$ligne) ;
  for $noeud (@Tmp) {
    ($att, $val) = split (/=/, $noeud);
    $arbre->{$att} = $val;
  }
  push(@LoH, $arbre);
}
```

La même opération est réalisable sans variable temporaire :

```
while ($ligne= <IN> ) {
  chop($ligne);
  push (@LoH, { split(/[\s+=]/,$ligne) });
}
```

Le principe ici est de découper la ligne courante suivant les espaces et les signes "="[1], et de stocker le tout dans un hachage anonyme dont on place l'adresse en fin de la liste @LoH.
Si, par la suite, nous souhaitons ajouter des couples à un hachage de notre liste de hachages, nous procédons comme suit :

```
${$LoH[0]}{'pere'} = "np";
${$LoH[0]}{'filsgauche'} = "det";
${$LoH[0]}{'filsdroit'} = "n";
```

Rappel : $HoL[0] contient l'adresse d'un hachage.

2.3 Accès et Affichage

Pour afficher un couple d'un hachage de notre liste de hachages, nous pouvons le désigner directement :

```
print "${$LoH[0]}{'filsgauche'}";
```

Nous pouvons également, au moyen d'une boucle, accéder à l'ensemble des adresses des hachages de la liste, et pour chacun des hachages ainsi référencé, nous pouvons, via une autre boucle, afficher les éléments qui le composent :

```
foreach $href ( @LoH ) {
  print "{ Arbre : ";
  foreach $role ( keys %{$href} ) {
    print "$role=$href->{$role} ";
  }
  print "}\n";
}
```

[1] Les deux instructions `%hash = ("Yes"=>"Oui","No"=>"Non") ;` et `%hash = ("Yes","Oui","No","Non") ;` sont équivalentes.

Ce qui s'écrit également au moyen d'une boucle for au lieu de foreach :

```
for($i=0;$i<=$#LoH;$i++) {
  print "{ Arbre n° $i ";
  foreach $role ( keys %{ $LoH[$i] } ) {
    print "$role=${$LoH[$i]}{$role} ";
  }
print "}\n";
}
```

3 Les hachages de hachages

Dans cette section, nous allons considérer une représentation fonctionnelle (à la LFG) d'une phrase simple.
Exemple : "Marie mange une banane"

predicat	:	lex = manger, cat= verbe, temps = present, mode = indicatif, sa = <sujet, objet>, nb= sg, per = 3
sujet	:	lex = Marie, cat=nom, ntype= propre, nb = sg, ge = fem, per = 3, sem= hum
objet	:	lex = banane, cat = nom, ntype = commun, nb = sg, ge = fem, det = undef, sem = concret

3.1 Déclaration

Pour définir le hachage de hachages contenant ces informations, nous adoptons la syntaxe suivante :

```
%HoH = (
pred => {
lex => "manger",
cat => "verbe",
temps => "present",
mode => "indicatif",
sa => "<sujet,objet>",
nb => "sg",
pers => "3"
},
sujet => {
lex => "Marie",
cat => "nom",
ntype => "propre",
sem => "hum",
nb => "sg",
pers => "3",
ge => "fem"
},
objet => {
```

```
lex => "banane",
cat => "nom",
ntype => "commun",
sem => "concret",
nb => "sg",
def => "undef",
ge => "fem"
});
```

3.2 Génération

Là encore, nous pouvons remplir notre hachage de hachages à partir d'un fichier texte. Supposons que nous disposons du fichier contenant les entrées suivantes :

predicat : lex=manger cat=verbe temps=present mode=indicatif sa=<sujet, objet> nb=sg pers=3

L'acquisition se fait au moyen d'une boucle while (lecture ligne par ligne).

```
while ($ligne= <IN> ) {
  my($fg,$reste,$ref,$key,$value,$attval);
  my @Tmp;
  chop($ligne);
  $ligne =~ /(.*):(.*)/;
  $fg = $1;
  $reste=$2;
  $ref = {};
  $HoH{$fg} = $ref;
  @Tmp = split(/ /, $reste);
  for $attval ( @Tmp ) {
    ($key, $value) = split (/=/, $attval);
    $ref->{$key} = $value;
  }
}
```

Pour ensuite ajouter de nouveaux couples att,vals à un hachage de hachages existant[2], nous procédons par affectation directe :

```
%nouv_st = ("optionel" => "oui","role" => "theme");
for $att (keys %nouv_st) {
  ${$HoH{'objet'}}{$att} = $nouv_st{$att};
}
```

Ici, nous avons affecté au hachage dont l'adresse est associée à l'entrée *objet* les couples *optionel=oui* et *role=theme*.

[2]Notre hachage de hachages représente en fait une structure de traits.

3.3 Accès et Affichage

Pour afficher un élément, la syntaxe est la suivante :

```
${$HoH{sujet}}{lex} = "gertrude";
```

Pour afficher l'ensemble des couples des hachages contenus dans notre hachage de hachages, nous utilisons deux boucles. La première parcourt les clés du hash de hachages, et pour chaque clé, une seconde boucle passe en revue les couples du hachage dont l'adresse est la valeur associée à cette clé :

```
foreach $fg ( keys %HoH ) {
  print "$fg: { ";
  foreach $attval ( keys %{$HoH{$fg}} ) {
    print "$attval=${$HoH{$fg}}{$attval} ";
  }
print "}\n";
}
```

4 Les enregistrements plus complexes

Pour finir, intéressons nous à des structures légèrement plus complexes. Imaginons des structures dont les champs sont de type différent. Un exemple est donné dans ce qui suit.

Déclaration

La déclaration ci-dessous définit un hachage %TV associant un hachage à des noms de série télévisée. Ce hachage associe lui-même respectivement un scalaire, une liste et un hachage à ses clés (les noms de série).

```
%TV = (
flintstones => {
series => "flintstones",
nights => ["monday","thursday","friday"],
members => [
{ name => "fred", role => "lead", age => 36, },
{ name => "wilma", role => "wife", age => 31, },
{ name => "pebbles", role => "kid", age => 4, }]},
jetsons => {
series => "jetsons",
nights => ["wednesday","saturday"],
members => [
{ name => "george", role => "lead", age => 41, },
{ name => "jane", role => "wife", age => 39, },
{ name => "elroy", role => "kid", age => 9, }]},
simpsons => {
series => "simpsons",
nights => ["monday"],
```

```
members => [
{ name => "homer", role => "lead", age => 34, },
{ name => "marge", role => "wife", age => 37, },
{ name => "bart", role => "kid", age => 11, }]});
```

Exemples d'utilisation :

```
print $TV{simpsons}->{members}->[0]->{name},"\n";
$nb_nuits = $#{$TV{jetsons}->{nights}}+1;
push(@{$TV{flintstones}->{nights}}, "sunday");
```

5 conclusion

Nous avons vu que le langage Perl permet la représentation de diverses informations. Les notations utilisées deviennent vite très lourdes, notamment à cause de l'emploi de références. La seule façon de devenir plus ou moins familié avec les structures de données complexes est de pratiquer, c'est-à-dire d'écrire des petits programmes manipulant de telles structures.