G. Hunault
Licence Professionnelle

Angers, février 2005

Développement *Web*

Vous zipperez tous vos fichiers-solution en une archive .zip avec vos initiales, comme indiqué sur la feuille d'émargement. Vous indiquerez sur votre copie le nom de cette archive (la liste des fichiers à fournir est indiquée en fin de sujet). Pour la correction, toutes vos pages *Web* seront testées avec le navigateur *mozilla* de **Sirius**.

1. Html et Xhtml

Reproduire en une page que vous nommerez lp1a.htm le tableau Html disponible à l'adresse :

`http://www.info.univ-angers.fr/pub/gh/internet/botablo.htm`

Il ne devra pas y avoir de balise globale autre que `<table>` dans la partie `<body>` de la page. On ne demande pas de reproduire le titre. Les nombres seront cadrés à droite et les points d'interrogation seront centrés. Le nom des colonnes sera géré avec la balise `<th>`.

Mettez ensuite dans lp1b.htm une copie de lp1a.htm qui est correcte au sens de HTML 4.01 (strict) ; vous le prouverez à l'aide du lien fourni par le W3C que vous mettrez en fin de page comme pour le document

`http://www.info.univ-angers.fr/pub/gh/internet/xmp_xhtml.htm`

2. Formulaires

Un utilisateur de bioInformatique qui utilise beaucoup le Web réalise régulièrement les manipulations suivantes :

- ouverture de la page de la "Protein data bank" : `http://www.rcsb.org/pdb/`
- clic sur le bouton radio (rond) `PDB ID`,
- saisie du code `1AIY` dans la zone d'entrée à gauche de `Search`,
- clic sur le bouton `Search`,
- clic sur le menu `Download/Display File` dans le panneau de gauche sur la page envoyée en retour,
- clic sur l'adresse marquée `HTML` dans la zone haute de l'écran (rubrique "Display the Structure File :", deuxième ligne du tableau, juste après "header only" (no coordinates).

Il voit dans la partie centrale de l'écran le panneau d'affichage de la protéine, qui commence par :

```
Save entry to disk

HEADER    HORMONE                         30-APR-97 1AIY
...
```

Pour aider cet utilisateur, vous devez écrire une page nommée lp2a.htm qui affiche un bouton nommé `1AIY`; si on clique sur ce bouton, la requête précédente pour la PDB est exécutée et on arrive directement au panneau d'affichage de la protéine `1AIY`.

Recopiez votre fichier en lp2b.htm et complétez-le afin d'avoir deux boutons et une zone de texte. Le premier bouton sera `1AIY` et le second bouton nommé `PROT` utilisera la zone de texte pour demander à la PDB de réaliser la requête complète (qui va jusqu'au panneau d'affichage de la protéine).

On pourra vérifier que si on écrit `1JXG` dans la zone de texte avant d'appuyer sur le bouton `PROT` on obtient comme début d'affichage

```
Save entry to disk

HEADER    PHOTOSYNTHESIS                  07-SEP-01   1JXG
...
```

3. Script pour SSI

On voudrait connaitre le nombre de fichiers perl présents dans le répertoire des scripts et dans ses sous-répertoires. Donner la suite de commandes *Unix* enchainées par | qui donne le nombre de ces fichiers sachant que leur suffixe est .pl ou .PL ou .prl mais pas .Pl ou PRL ; on admettra que le répertoire de stockage des scripts est le répertoire par défaut de l'exécution des scripts.

Ecrire lisiblement sur votre copie la suite de commandes Unix à utiliser. On ne demande aucune explication.

4. Javascript

Dans le cadre de la mise en place d'un site marchand, on désire afficher des prix en dollars et euros (on admettra qu'un euro vaut actuellement 1,284 dollar). Sachant que les clients fidèles ont droit à 5 % de remise, écrire une page Web constituée de 4 cadres (ou "frames") telle que : le cadre du haut est un formulaire pour saisir le prix normal TTC en euros dont le bouton "Tarifs" lance l'affichage dans les 3 autres cadres de même largeur ; le cadre bas-gauche donne les prix en dollars ; le cadre bas-milieu donne les prix normaux en euros et le cadre bas-droit donne les prix remisés en euros. Tous les prix seront affichés avec deux chiffres après la virgule.

On respectera l'affichage de la page

`http://www.info.univ-angers.fr/pub/gh/internet/prix.htm`

à savoir : on donne les prix pour 1,2..10 articles puis pour 15, 20, 25 et 50 articles.

On nommera `lp4.htm` la page principale, 7 `lp4h.htm`, `lp4bg.htm`, `lp4bm.htm` et `lp4bd.htm` les pages haut, bas-gauche, bas-milieu et bas-droit.

Chaque contenu de cadre du bas utilisera la même fonction javascript *tabPrix* pour afficher les prix ; le texte de la fonction sera défini dans `lp4.js` ; on pourra utiliser deux paramètres pour *tabPrix* : un pour le prix de base normal en euro pour un seul article et un autre pour repérer de quel cadre du bas il s'agit.

Si vous n'arrivez pas à programmer les 3 cadres du bas en même temps, vous pouvez n'utiliser qu'un seul cadre en bas, celui du milieu, avec les prix normaux.

5. Perl

Lorsqu'une page Web référence un site, elle utilise le marqueur `<A HREF`. On veut ici automatiser l'affichage d'une liste de sites à l'aide d'une liste de définition (marqueur `<DL>`), chaque site étant défini par son adresse et un commentaire. Par exemple pour le site `http://www.google.fr` avec le commentaire `Le meilleur moteur de recherches`, on voudrait produire la partie de page Web suivante :

```
<dt><a href="http://www.google.fr">http://www.google.fr</a></dt>
<dd>Le meilleur moteur de recherches</dd>
```

Ecrire une fonction Perl nommée `ecritSite` qui met en forme ses deux paramètres à l'aide des balises indiquées ; exemple d'utilisation :

```
&ecritSite($cmt,$adr);
```

Vous insérerez ensuite cette fonction dans un programme Perl nommé `lp5.pl` qui lit un fichier texte dont le nom est passé en paramètre. Ce fichier contient sur une ligne l'URL à utiliser et sur la ligne suivante le commentaire à ajouter. Il peut y avoir des lignes vides avant l'URL et/ou après le commentaire mais jamais entre l'URL et le commentaire.

Vous trouverez un exemple de tel fichier (qui pourra donc vous aidez à tester votre programme) à l'adresse

`http://www.info.univ-angers.fr/pub/gh/internet/lp5a.txt`

Pour les plus fort(e)s, indiquez comment on peut insérer des commentaires avec les symboles `"` et `'` ou même `<` comme pour `lp5b.txt` qui contient

- Département Informatique de l'Université d'Angers
- sans doute "le meilleur" `<<moteur de recherches>>`

6. La multiplicité des langages pour le Web est-elle un bien ?

Il existe dans le monde plusieurs centaines de langages de programmation dont la plupart peuvent prétendre à être utilisés dans des scripts ou directement dans des pages Web.

Vous discuterez les deux points suivants :

est-ce un bien pour le programmeur débutant de disposer d'autant de langages ?

est-ce un bien pour le programmeur professionnel de disposer d'autant de langages ?

Vous rédigerez votre argumentation sur une demi-page au minimum, sur deux pages complètes au maximum, exemples compris. Vous pourrez prendre des exemples concrets en utilisant les langages vus en cours (sans oublier le langage algorithmique). Vous pourrez utiliser des langages non encores vus en cours à condition de justifier leur importance.

Liste des fichiers à fournir

Votre archive .zip devra contenir au minimum les fichiers suivants :

lp1a.htm, lp1b.htm (exercice 1),

lp2a.htm, lp2b.htm (exercice 2),

lp4.js , lp4.htm, lp4h.htm, lp4bg.htm, lp4bm.htm, lp4bd.htm, (exercice 4),

lp5.pl (exercice 5).

Compte tenu de la durée de l'épreuve et du travail demandé, il n'est peut être pas judicieux de vouloir tout tester sur ordinateur.

Esquisse de Solution

Tous les fichiers demandés sont disponibles à l'adresse

`http://www.info.univ-angers.fr/pub/gh/internet/dw2005/`

1. Html et Xhtml

Pour réaliser le tableau demandé, il suffit de savoir utiliser les balises `rowspan` et `colspan`. Si on veut qu'une case occupe la largeur de deux colonnes, on écrit `<td colspan="2">`. Les balises `<th>` s'utilisent à la place de `<td>` à l'intérieur des balises `<tr>` et `</tr>` pour afficher les noms de colonnes.

Pour valider en `XHTML` la page, il d'abord faut respecter les règles indiquées sur la page

`ahttp://www.info.univ-angers.fr/pub/gh/internet/xhtml_rules.htm`

et en particulier il faut mettre un `DOCTYPE` en première ligne du fichier, comme :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
                      "http://www.w3.org/TR/html4/strict.dtd">
```

En fin de document, il faut mettre un lien pour que le `W3C` valide la page, à savoir :

```
<p>     Le code de cette page est valide en HTML 4.01 (strict).
<br />
        La preuve :
<a href="http://validator.w3.org/check/referer">W3C HTML 4.01
</a>
</p>
```

2. Formulaires

Lorsqu'on effectue les manipulations demandées, on voit que les URL respectives sont

```
http://www.rcsb.org/pdb/

http://www.rcsb.org/pdb/cgi/explore.cgi?
       pid=60311109584482&pdbId=1AIY

http://www.rcsb.org/pdb/cgi/explore.cgi?
       job=download&pdbId=1AIY&page=&pid=234221109588821

http://www.rcsb.org/pdb/cgi/explore.cgi?
       job=download;pdbId=1AIY;page=;pid=234221109588821
      &opt=show&format=PDB&header=1
```

Le premier formulaire demandé consiste simplement à recopier cette dernière URL et l'adresser via un bouton. Malheureusement mettre juste l'URL dans le champ `action` du formulaire ne suffit pas pour tous les navigateurs. Plutôt que d'écrire

```
<form action="http://www.rcsb.org/...
      <input type="submit"value="1AIY">
</form>
```

il est donc plus prudent de forcer l'ouverture via *javascript* soit le texte

```
<form>
  <input
     type ="submit" value="1JXG"
     onClick="javascript: NewWin=window.open('http://www.rcsb.org/...
</form>
```

Pour le deuxième formulaire, le plus simple serait d'utiliser PHP mais le cours de PHP ayant lieu après l'examen, on écrit une fonction *javascript* nommé `pdb` à qui on transmet le contenu du champ texte.

Voici le texte de cette fonction

```
<script>
function pdb( numpdb ) {
  // pour test : alert(numpdb)
  hr = "http://www.rcsb.org/pdb/cgi/explore.cgi?"
  hr = hr + "job=download;pdbId="+numpdb
  hr = hr + ";page=&opt=show&format=PDB&header=1"
  // window.location.href = hr ;
  NewWin=window.open(hr) ;
} ; // fin de fonction pdb
</script>

...
<form>
  <input type ="text" name="numpr">
  <input type ="submit" value="PROT"
         onClick="javascript: pdb(this.form.numpr.value)" >
</form>
```

3. Script pour SSI

Puisqu'on veut chercher des fichiers dans les répertoires et les sous-répertoires, la commande `find` est certainement plus adaptée que la commande `ls`. Au niveau du motif à détecter ensuite par `grep`, il faut la lettre `p` en majuscule ou en minuscule juste après le point qui indique l'extension du fichier, soit l'expression régulière

```
\.[pP]
```

Il faut ensuite éventuellement la lettre `r` puis la lettre `l` en majuscule ou en minuscule en fin de fichier, soit l'expression régulière

```
r\?[lL]$
```

Pour compter le nombre de fichiers trouvés par l'enchainement des commandes `find ... grep` on utilise comme d'habitude `wc -l`.

Une première version du script est

```
find . | grep "\.[pP]r\?[lL]$" | wc -l
```

Après avoir créé rapidement le fichier `A.PL` avec la commande

```
echo "# test" > A.PL
```

on se rend compte que notre script renvoie 1, ce qui fait un de trop. On refuse donc les fichiers `.PL` à l'aide de l'option `-v` de `grep`.

Enfin, pour que le script renvoie un message plus explicite au lieu du simple nombre de fichiers qui correspondent, on utilise la commande `echo` avant l'exécution de l'enchainement des commandes précédentes via l'anti-apostrophe (située en général sur le clavier au-dessu du `Y` et du `U`, accessible via la touche `Alt Gr`). Voici donc le script complet :

```
echo    `find . | grep "\.[pP]r\?[lL]$"
                | grep -v "\.PL"
                | wc -l `
        fichier\(s\) corresponde\(nt\).
```

4. Javascript

Le formulaire doit transmettre une valeur aux trois cadres. Cela se réalise via une fonction javascript nommée `affTar`. Voici un contenu possible pour le formulaire

```
<form>
   <font size=6 color="#000088">prix normal unitaire</font>
   <input type="text" name="pnu" value=" 10 " size="4">
   <font size="+2">&euro;</font>
   <input type="button" value="Tarifs" onClick="affTar(this.form)">
</form>
```

La fonction doit simplement modifier le contenu des cadres. Pour cela, il suffit de recharger chacun des cadres via l'atribut `location.href` pour chaque cadre.

La fonction `affTar` se réduit donc à remettre dans `location.href` le nom du fichier qui correspond au cadre :

```
function affTar(pds) {
  parent.dolla.location.href="tarfdgb.htm"
  parent.euros.location.href="tarfemb.htm"
  parent.remis.location.href="tarfrdb.htm"
} ; // fin de fonction affTar
```

Puisque chaque cadre doit utiliser une même fonction, le chargement de cette fonction se fait donc pour chacun des cadres par

```
<script src="tabprix.js"></script>
```

Pour que chaque cadre ait un affichage différent, on utilise la fonction `tabprix` avec un premier paramètre dont la valeur est 1, 2 ou 3 pour désigner le cadre gauche, milieu ou droit. Ensuite, il reste à lui transmettre comme deuxième paramètre la valeur du champ-texte du formulaire. Ainsi pour le cadre du milieu, le texte du cadre se réduit au texte

```
<script>
tabprix( 2 ,
         window.parent.frames[0].document.forms[0].pnu.value
       )
</script>
```

La fonction `tabprix` quant à elle est une boucle pour avec une variable supplémentaire pour gérer l'incrémentation afin d'obtenir les valeurs de fin 5, 10, 15, 20, 25 et 50 :

```
function tabprix(mode,pdb) {

if (pdb.length>0) {
// on n'affiche rien si ce n'est pas un nombre

var exprg = new RegExp("^[ 0-9]+.?[ 0-9]*$");
if (exprg.test(pdb)) {
   document.write("<table>")
```

```
   var p = 1.0 ;
   var j = 1 ;

   for (i = 1 ; i <= 10 ; i++) {

       window.document.writeln('<tr>')
       window.document.writeln('
          <td align=right><font size="5" color="#008000"><b>',j,'</td>')
       window.document.writeln('
          <td align=right><font size="5" color="#008000"><b>
           art.    </td>')

       p = pdb*j ;
       if (mode==1) { p = p * 1.284 }
       if (mode==3) { p = p * 0.95  }

       window.document.writeln('
          <td align=right><font size="5" color="#800000"><b> ',
          p.toFixed(2))

       if (mode==1) { document.write(" $ </td>") ;}
       if (mode==2) { document.write(" &euro;</td>") }
       if (mode==3) { document.write(" &euro;</td>") }

       window.document.writeln('</tr>')

       j = j + 1 ;
       if (i>=5) { j = j + 4 } ;
       if (i>=9) { j = j + 20 } ;

    } // fin de pour

  document.write("</table>")
} // fin de si sur expression reguliere
} // fin de si sur pdb non vide

} ; // fin de fonction tabprix
```

5. Perl

Pour ne traiter que les lignes non vides avec perl, si `$lig` est la ligne courante, un test comme `if (length($lig)>0) {...` suffit. Pour traiter soit `<dd>` soit `<dt>` on peut utiliser une variable `$dt` qu'on incrémente pour chaque ligne lue non vide et qu'on remet à zéro après avoir traiter le cas `$dt=2` qui correspond à la balise `<dt>`. Une première version du programme *perl* qui fonctionne avec `lp5a.txt` peut donc ressembler à :

```
#  test des paramètres

   if ($ARGV[0] eq "") {
        print " syntaxe : perl amenjs.pl nom_de_fichier\n" ;
        exit(-1) ;
   }  ; # fin de test sur les arguments

$fichier = $ARGV[0] ; # récupération du nom du fichier

# ouverture du fichier

   open( FIC ,"<$fichier")
     || die "\n impossible d'ouvrir le fichier nommé $fichier \n\n" ;

# réécriture pour javascript

$dt = 1 ;
while ($lig=<FIC>) {
  chomp($lig) ;
  if (length($lig)>0) {
     $dt++ ;
     if ($dt==2) {
        print "window.document.write(
           \'<dt><a href=\"$lig\">$lig</a></dt>\') ;\n" ;
        $dt = 0 ;
     } ; # fin de si on gère dt
     if ($dt==1) {
        print "window.document.write(\'<dd>$lig</dd>\') ;\n\n" ;
     } ; # fin de si on gère dd
  } ; # fin de si ligne non vide
} ; # fin de tant que
```

Pour mieux gérer les lignes dans la partie pour la balise `<dd>`, il faut gérer les caractères spéciaux que sont ' " < > ; on peut par exemple écrire avant d'afficher `$lig` :

```
$lig =~ s/'/\\'/g ;
$lig =~ s/"/\\"/g ;
$lig =~ s/\</&lt;/g ;
$lig =~ s/\>/&gt;/g ;
```

6. La multiplicité des langages pour le *Web* est-elle un bien ?

S'il est difficile de trouver "une" solution aux questions posées, car il y a du bon et du mauvais dans la multiplicité il est possible par contre de nuancer le terme multiplicité. En effet, du coté client il n'y a qu'un seul langage possible : *javascript*. Du coté serveur, on peut certainement aussi réduire la multiplicité : dans les faits, on n'utilise principalement que *php*, *java*, *perl*.

La multiplicité des langages correspond d'une part à la multiplicité des objectifs de développement et d'autre part à une conséquence des économies de type libéral : de nombreuses entreprises mettent sur le marché des produits, des langages différents mais équivalents dans le but de gagner de l'argent.

Cette "prolifération" de langages, produits, logiciels et systèmes de développement *Web* permet aux professionnels de disposer de langages spécialisés intégrés à des environnements complets de développement avec des boites à outils pour le retouche d'images, la conception graphique, la réalisation d'animations, l'intégration automatique d'interfaces à des bases de données.

Si le programmeur débutant qui débute seul peut être dérouté par ces langages et les "IDE" associés, le débutant encadré peut y trouver son compte : par exemple en se focalisant sur les actions à exécuter, la complétion automatique des commandes et des balises par l'environnement pour les formulaires évite les erreurs de syntaxe grossières.

Pour le programmeur chevronné, c'est au prix d'un apprentissage long et parfois laborieux que la diversité devient richesse, la spécialisation des logiciels et langages correspondant alors à la spécialisation du programmeur.