Programmation Internet (Info 214), L2–MI 8 Janvier 2014
Université Paris-Sud

Examen

L'examen est sur 20 points et comprend 8 pages (dont un aide-mémoire HTML et PHP à partir de la page 6). Les quatres exercices sont indépendants. Aucun document n'est autorisé. Les réponses doivent être soigneusement justifiées. L'utilisation de téléphones portables ou calculatrices est interdite. Le barème est donné uniquement à titre indicatif. Le barême réflète la longueur/difficulté des exercices (il vous est donc conseillé de passer environ 1h sur l'exercice 4, mais pas plus).

Les instructions suivantes doivent être respectées scrupuleusement, sous peine de non correction :

- Reporter sur **toutes** les copies le numéro d'anonymat imprimé sur la copie principale. Cette copie **doit** être rendue, même blanche.
- Chaque copie doit être numérotée et mentionner le nombre total de copies rendues (1/3, 2/3, ...). Si une seule copie est rendue, la numéroter 1/1. Les copies non numérotées **ne seront pas corrigées**.
- Toutes les réponses doivent être rédigées **au stylo**, sur des copies. Les parties rédigées au crayon de papier ou sur les feuilles de brouillons **ne seront pas corrigées**.
- Les copies doivent être anonymées, mais uniquement après vérification d'identité par un surveillant et signature de la liste d'émargement.

1 Questions de cours (2 points)

1. (1 point) Donner **un** inconvénient d'un système de cryptage symétrique (2 lignes max).
2. (1 point) Quelle fonction PHP permet d'initialiser la valeur et la durée de validité d'un *cookie* ?

2 Ligne de commande, permissions, chemins Unix (4 points)

Cet exercice sera corrigé de manière stricte, les points ne seront donnés **que** si la réponse, entrée à l'identique dans un terminal produit le résultat attendu ; 0 point dans le cas contraire.

Donner une *unique* commande (pas plusieurs commandes séparées par des retours à la ligne/espaces/points virgules) permettant d'effectuer les actions ci-dessous. On suppose que pour **chaque question** on se trouve dans le répertoire utilisateur de l'utilisateur `lambda`. On suppose que le répertoire utilisateur de `lambda` contient un répertoire `travail`. (0,5 point par réponse)

1. afficher la liste détaillée des fichiers et répertoires du répertoire courant
2. afficher la liste détaillée des fichiers et répertoires dont le nom contient un chiffre
3. faire du répertoire `travail` le répertoire courant
4. changer les persmissions du répertoire `travail` pour le rendre traversable, lisible et écrivable par `lambda` et son groupe, et sans permission pour les autres
5. déplacer tous les fichiers d'extension `.txt` dont le nom commence par une majuscule et fini par un chiffre (avant l'extension) du répertoire courant vers le répertoire `travail`
6. déplacer tous les fichiers dont le nom n'est ni `abc.txt` ni `def.txt` du répertoire courant vers le répertoire `travail`.
7. supprimer tous les fichiers dont le nom est une suite de `a` de longueur 1 ou plus du répertoire courant
8. changer les permissions du répertoire `travail` pour le rendre traversable et lisible par tout le monde (utilisateur, groupe, autres) et écrivable par personne (ni utilisateur, ni groupe, ni les autres)

TSVP ▶▶|

3 Réseaux (4 points)

Dans toutes les questions suivantes, on considère le mode d'adressage par *sous-réseau* (i.e. utilisant un masque pour déterminer la partie machine et la partie réseau d'une adresse IP).

1. (1 point) On souhaîte mettre sur un même sous-réseau 9000 machines. Donner la valeur minimale du masque de sous-réseau correspondant et dites combien de machines peuvent être stockées sur un sous-réseau ayant ce masque.
2. (3 points) On considère les adresses IP suivantes : `129.175.28.101`, `129.175.30.1`, `129.175.31.10`, `129.175.33.35`, `129.175.34.92`, `129.175.35.129`, `129.175.36.200`, avec le masque de sous-réseau `255.255.252.0`. On suppose que chaque adresse correspond à une machine différente (donc chaque machine a une seule carte réseau). On souhaite rajouter des routeurs ayant exactement deux cartes (et donc deux adresses) à ce réseau, de manière à ce que toute machine soit atteignable depuis n'importe quelle autre. Donner le nombre minimal de routeurs à ajouter (vous justifierez votre réponse).

4 Programmation HTML et PHP (10 points)

Toutes les fonctions et balises qui pourraient être nécessaire sont listées dans l'aide-mémoire de la page 6 (cela ne signifie **pas** que vous devez toutes les utiliser coûte que coûte !).

On souhaite réaliser un jeu de labyrinthe rudimentaire. Un fichier texte décrivant le labyrinthe est chargé (ce fichier indique les espaces libres, les murs, la position de départ et la position d'arrivée). La figure 1 illustre le format du fichier ainsi que l'interface HTML utilisée pour jouer. Comme indiqué sur les captures d'écran, l'affichage comprend trois zones :

— un formulaire permettant d'envoyer un fichier de labyrinthe sur le serveur
— une zone où le labyrinthe est affiché (en cas d'erreur, un message est affiché à la place du labyrinthe)
— un formulaire permettant de saisir une commande de déplacement

Une commande est une suite de lettre `H`, `B`, `G`, `D` (haut, bas, gauche, droite) de longueur 0 ou plus. Une commande est *correcte* si et seulement si elle est bien formée *et* qu'elle ne traverse pas un mur, et ne sort pas des limites du labyrinthe. Après l'envoi d'une commande correcte, la page est redessinée en faisant apparaître la trace parcourue entre la position courante et la position d'arrivée. De plus, si la position d'arrivé est la case « but », la page affiche « **succès** ». Une commande peut être envoyée en plusieur fois. Par exemple, on pourrait soumettre (dans l'exemple de la figure) une première commande `GGGG`, une deuxième commande `GBBDBBG` et une troisième commande `B` pour arriver au but.

Le code (incomplet) permettant d'obtenir un tel affichage est donné à la figure 2. Comme on le voit dans la figure, les deux formulaires sont en HTML statique (pas de PHP) alors que la zone centrale (qui affiche le labyrinthe) est calculée par un petit bout de code PHP. Ce dernier vérifie en premier lieu si un fichier a été correctement envoyé au serveur et si oui, appelle la fonction `charger_labyrinthe` en lui passant le nom du fichier temporaire contenant le labyrinthe. Ensuite, le code appelle la fonction `evaluer_commande` et en fonction du résultat de cette dernière, affiche un message d'erreur ou un message de succès ainsi que le labyrinthe mis à jour (ce qui est fait au moyen de la fonction `afficher_labyrinthe`). **Le but de l'exercice est de coder en PHP ces 3 fonctions**.

Questions

1. (1 point) Pour permettre d'envoyer plusieurs commandes successives, le labyrinthe ainsi que la position actuelle sont stockées dans des variables de session. Indiquer quel est la première instruction à executer dans le code à compléter (en début de fichier).
2. (3 points) Écrire la fonction `charger_labyrinthe($nomf)`. Cette fonction prend en argument un nom de fichier et charge son contenu comme un tableau de la manière suivante :
   (a) création d'un tableau vide `$tab`
   (b) initialisation une variable `$largeur` à `-1`
   (c) pour chaque ligne i du fichier passé en argument

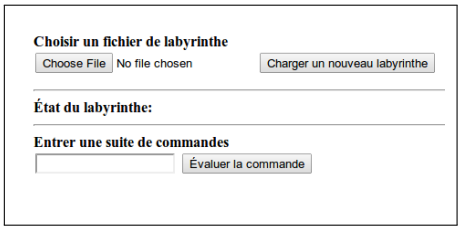
TSVP ▶▶I

Fichier `test.laby`

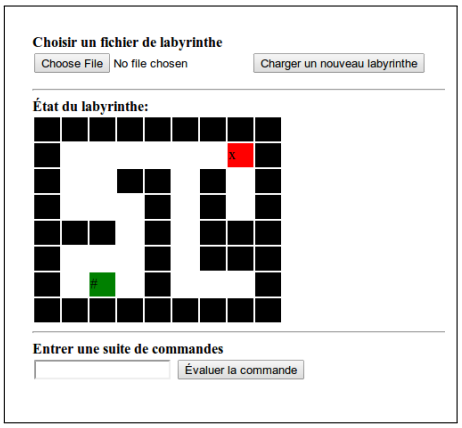
```
111111111
100000031
100110101
100010101
111010111
100010111
104010001
111111111
```

`0` : espace vide
`1` : mur
`3` : départ
`4` : arrivée

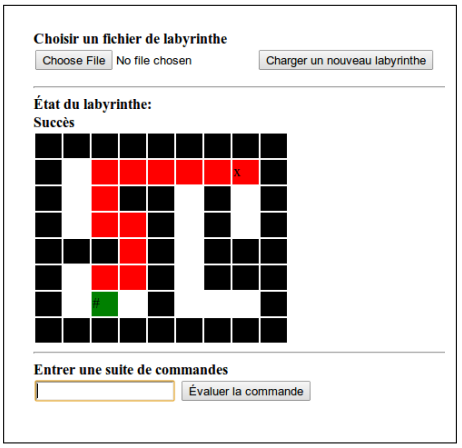
Affichage initial



Après le chargement de `test.laby`



Après l'envoie de la commande `GGGGGBBDBBGB`



Après une commande invalide

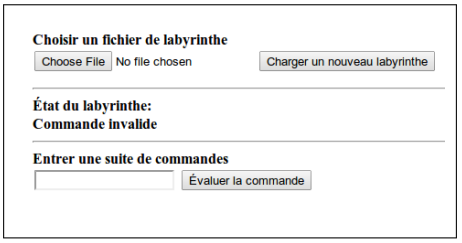


FIGURE 1 – Le contenu d'un fichier de labyrinthe et des captures d'écran de l'interface

```
<?php
//Cette partie est à compléter
?>

<html>
<head>
<title>Labyrinthe</title>
<style>
td { width: 20pt;
     height: 20pt;
 }
.mur { background: black; }
.vide {background: white; }
.trace {background : red; }
.pos {background : yellow; }
.but {background : green; }

</style>
</head>
<body>
<form action="laby.php" method="post"
enctype="multipart/form-data">
<b>Choisir un fichier de labyrinthe</b><br/>
<input type="file" name="fichier" size="20"> </input>
<button type="submit">Charger un nouveau labyrinthe</button>
</form>
<hr/>
<b>État du labyrinthe:</b><br/>

<?php
if (isset($_FILES["fichier"]) &&
    $_FILES["fichier"]["error"] === 0) {
charger_labyrinthe( $_FILES["fichier"]["tmp_name"]);
};
$res = evaluer_commande();
if ($res === 1) echo "<b>Commande invalide</b><br/>";
else {
if ($res === 2) {
echo "<b>Succès</b><br/>";
}
afficher_labyrinthe();
}
?>

<hr/>
<form action="laby.php" method="post"
enctype="multipart/form-data">
<b>Entrer une suite de commandes</b><br/>
<input type="text" name="commande" size="20"> </input>
<button type="submit">Évaluer la commande</button>
</form>
</body>
</html>
```

FIGURE 2 – Le code source du fichier PHP, à compléter

i. retirer le `\n` terminal

ii. si `$largeur` vaut `-1`, stocker la longueur de la ligne dans `$largeur`

iii. si la longueur de la ligne est différente de `$largeur`, la fonction retourne immédiatement sans rien faire d'autre

iv. si la ligne courante n'est pas composée exactement des caractères `'0'`, `'1'`, `'3'`, `'4'`, la fonction retourne immédiatement sans rien faire d'autre

v. par chaque caractère d'indice j de la ligne courante, si le caractère dans la case j vaut `'3'`, alors j est sauvegardé dans la variable de session `"x"` et i est sauvegardé dans la variable de session `"y"`

vi. stocker la ligne courante dans la case `$tab[`i`]`

(d) une fois toutes les lignes parcourues, le tableau `$tab` est stocké dans la variable de session `"laby"`, la largeur du tableau dans la variable de session `"l"` et la hauteur du tableau dans la variable de session `"h"`. Suite à quoi la fonction retourne.

On suppose que le fichier contient exactement un `3` mais on ne demande pas de le vérifier.

3. (3 points) Écrire la fonction `evaluer_commande()`. Cette fonction ne prend pas d'argument mais vérifie le contenu du paramètre `POST "commande"`. Si ce dernier n'est pas défini, la fonction renvoie `0`. Sinon, la fonction vérifie que ce paramètre contient une suite de lettres `H`, `G`, `D`, `B` uniquement. Elle initialise ensuite deux variables locales `$x` et `$y` à partir des variables de session `"x"` et `"y"`. Elle parcourt ensuite la liste des commandes et modifie `$x` et `$y` de la manière suivante :

   `H :` `$y` est décrémenté, `$x` est inchangé

   `B :` `$y` est incrémenté, `$x` est inchangé

   `G :` `$x` est décrémenté, `$y` est inchangé

   `D :` `$x` est incrémenté, `$y` est inchangé

   si a un moment quelconque la case (`$x`, `$y`) du tableau `"laby"` vaut `'1'`, ou que `$x` ou `$y` sont en dehors des bornes du tableau, la fonction renvoie 1 et termine. Si a un moment quelconque la case (`$x`, `$y`) du tableau `"laby"` vaut 4, la fonction renvoie 2 et termine. Si toutes les commandes ont été parcourues sans erreur, la fonction renvoie `0` (et termine).

4. (3 points) Écrire la fonction `afficher_labyrinthe()`. La fonction affiche sous la forme d'une table `HTML` le contenu du tableau `"laby"`. Si une case contient un `'0'`, `'1'`, `'2'`, `'3'` ou `'4'` la case de la `table` HTML correspondante aura l'attribut `class` valant `"vide"`, `"mur"`, `"trace"`, `"pos"` ou `"but"` respectivement. De plus, si la case vaut `3`, la case de la table contient un `x` et si la case contient `'4'`, la case de la table contient un `#`.

Aide-mémoire

HTML

Balise	Description
`<a href="`s`">...</a>`	Lien hypertexte ayant pour cible l'url s
`<b>...</b>`	Mise en gras du texte
`<body>...</body>`	Corps de la page Web
`<button type="submit">` l `</button>`	Bouton de soumission de formulaire, avec le texte l. Doit apparaître sous un `form`
`<div>...</div>`	Définit une section
`<form action="`a`" method="`m`">...</form>`	Formulaire ayant pour cible l'url a et utilisant la méthode m comme passage de paramètres
`<h1>...</h1>`	Titre de sections de niveau 1. Il existe `h2`, `h3`, ...pour des titre de sous-sections, sous-sous-sections *etc.*
`<head>...</head>`	En-tête de la page Web
`<html>...</html>`	Balise principale de la page Web
`<i>...</i>`	Mise en italique du texte
`<img src="`u`"></img>`	Affiche l'image se trouvant à l'url u
`<input type="`t`" name="`n`">...</input>`	Affiche un élément graphique de type t qui permet de saisir une valeur stockée dans le paramètre de nom n. Doit apparaître sous un `form`. Les différents types pour t sont `text` (champ de texte), `file` (sélecteur de fichier), `button` (bouton), ...
`<li>...</li>`	Élément d'une liste. Doit apparaître sous un `ul` ou `ol`
`<ol>...</ol>`	Liste énumérée
`<option value="`v`">`t`</option>`	Une des valeurs possibles pour une liste énumérée. v est la valeur stockée dans le paramètre, t est le texte correspondant, affiché par le navigateur. Doit apparaître sous un `select`
`<p>...</p>`	Paragraphe
`<select name="`n`">...</select>`	Affiche une liste permettant de choisir une valeur parmi plusieurs (les valeurs sont données par l'élément `option`). La valeur choisie est associée au paramètre de nom n. Doit apparaître sous un `form`.
`<table>...</table>`	Définit une table HTML
`<td>...</td>`	Définit une cellule d'une table HTML. Doit apparaître sous un `tr` ou `th`
`<th>...</th>`	Définit une ligne d'en-tête de colonnes d'une table HTML. Doit apparaître sous un `table`
`<th>...</th>`	Définit une ligne d'une table HTML. Doit apparaître sous un `table`
`<ul>...</ul>`	Liste non ordonnée

PHP

Variables globales

Nom	Description
`$_GET`	Tableau dont l'indice n contient la valeur du paramètre `get` de nom n
`$_POST`	Tableau dont l'indice n contient la valeur du paramètre `post` de nom n
`$_SESSION`	Tableau dont l'indice n contient la valeur de la variable de session de nom n
`$_COOKIE`	Tableau dont l'indice n contient la valeur du *cookie* de nom n

Expressions de bases

Nom	Description
`NULL`	Valeur nulle (renvoyée pour les variables non déclarées, les indices de tableaux inexistants, ...)
`TRUE`, `FALSE`	Booléens vrai et faux
`isset(`e`)`	e peut être une variable ou un accès à un tableau. Renvoie vrai si e est défini et faux sinon
`+`, `-`, `*`, `/`, `%`	Opérations arithmétiques addition, soustraction, multiplication, division, modulo
`.`	Concaténation de chaînes de caractères
`echo` e	Affiche le résultat de l'expression e sur la sortie renvoyée au client

Tableaux

Nom	Description
`array (`x_0 `=>` y_0`, ...,` x_n `=>` y_n`)`	Création d'un tableau et initialisation de l'indice x_i avec la valeur y_i
`count($t)`	Renvoie la taille du tableau `$t`
`$t[`v`]`	Accès à la case d'indice v dans le tableau `$t`. Renvoie `NULL` si l'indice n'est pas dans la tableau
`asort($t)`	Trie le tableau `$t` par ordre de valeurs
`ksort($t)`	Trie le tableau `$t` par ordre d'indices

Lecture de fichiers

Nom	Description
`file($f)`	Prend en argument une chaîne représentant un nom de fichier et renvoie un tableau indicé par des entiers et dont la case i contient la $i^{\text{ème}}$ ligne du fichier. Les retours à la ligne éventuels sont conservés en fin de chaîne
`file_get_contents($f)`	Prend en argument une chaîne représentant un nom de fichier et renvoie le contenu du fichier sous forme d'une chaîne de caractères

Manipulation de chaînes de caractères

Nom	Description
`trim($s)`	Supprime les caractères blancs (espaces, retour à la ligne, tabulation, ...) en début et fin de la chaîne passée en argument et renvoie la nouvelle chaîne
`preg_split($r, $s)`	Découpe la chaîne de caractères `$s` en utilisant comme séparateur les suites de caractères correspondant à l'expression régulière `$r` et renvoie le résultat sous forme d'un tableau indicé par des entiers. Les séparateurs sont supprimés. Exemple : `preg_split('/,+/', 'a,,b,cd')` renvoie un tableau `( 0 => 'a', 1 => 'b', 2 => 'cd' )`
`preg_match($r, $s)`	Renvoie 1 si une sous chaîne de la chaîne `$s` correspond à l'expression régulière `$r` et 0 sinon

Les expressions régulières PHP sont des chaînes de caractères commençant et finissant par « `/` » et utilisant les opérateurs :

TSVP ▶▶|

`.` n'importe quel car.	e`?` Rep. 0 ou 1 fois	`[`$c_1 \dots c_n$`]` 1 car. dans $c_1 \dots c_n$
e_1`|`e_2 e_1 ou e_2	e`*` Rep. ≥ 0	`[^`$c_1 \dots c_n$`]` 1 car. diff. de $c_1 \dots c_n$
`(`e`)` e	e`+` Rep. ≥ 1	`[`c_1`-`c_n`]` 1 car. dans $c_1 \dots c_n$
`^` Début de texte	`$` Fin de texte	

Gestion des cookies et des sessions

Nom	Description
`setcookie($n, $v, $t)`	Stocke dans le *cookie* de nom `$n`, la valeur `$v` avec la date d'expiration `$t` (qui est un nombre de secondes depuis 1/1/1970)
`session_start()`	Démarre une session

Gestion du temps

Nom	Description
`time()`	Renvoie l'heure courante, en nombre de secondes écoulées depuis la date de référence `EPOCH` (1/1/1970). On suppose que cette heure est celle au fuseau horraire UTC+0 (temps universel)
`date($s, $t)`	Prend en argument un chaîne de format `$s` et un nombre de seconde depuis `EPOCH $t` et renvoie une représentation de la date sous forme de chaîne de caractères. Les substitutions suivantes sont opérées dans la chaîne de format : `d` est remplacé par le jour du mois, sur 2 chiffres `m` est remplacé par le mois, sur 2 chiffres `Y` est remplacé par l'année, sur 4 chiffres `H` est remplacé par l'heure, sur 2 chiffres entre 00 et 24 `i` est remplacé par les minutes, sur 2 chiffres entre 00 et 59 `s` est remplacé par les secondes, sur 2 chiffres entre 00 et 59 Les autres caractères sont inchangés. Exemple : `date('H,i,s--Y-m', 45)` renvoie la chaîne de caractères : `"00,00,45--1970-01"` (car 45 correspond au 1/1/1970, 0h 00m 45s)