Réseaux 1

TP 8 - Programmation CGI

Objectif : écrire des scripts CGI pour créer des pages HTML de manière dynamique

Note : vous placerez vos scripts CGI sur *nyx*, dans un répertoire `~/web/rx1/TD8`

1. Premiers exercices

Date sur le serveur

1.1. En Bourne Shell, écrire un script `test_date.cgi` qui renvoie un document html avec la date et l'heure locale sur le serveur comme dans l'exemple ci-dessous.

1.2. Insérer un lien dans `index.html` permettant d'exécuter le script `test_date.cgi`.

Variables d'environnement

1.3. En Bourne Shell, écrire un script `test_env.cgi` qui renvoie la liste des variables d'environnement initialisées par le démon `httpd` (utilisez la commande Unix **`printenv`**).

1.4. Insérer un lien dans `index.html` permettant d'exécuter le script `test_env.cgi`, examinez les variables d'environnement et leurs valeurs.

1.5. Insérer un lien dans `index.html` permettant d'exécuter le script `test_env.cgi` avec le paramètre `une+query+string`, examinez la variable d'environnement **QUERY_STRING**.

1.6. Insérer un lien dans `index.html` permettant d'exécuter le script `test_env.cgi` avec le paramètre `une+query+string` et `un/path/info`, examinez les variables d'environnement **PATH_INFO** et **QUERY_STRING**.

Caractéristiques de votre arpenteur

1.7. En Bourne shell, écrire un script test_nav.cgi qui renvoie un document présentant les caractéristiques de votre arpenteur comme dans l'exemple ci-dessous.

1.8. Insérer un lien dans `Index.html` pour voir les caractéristiques de votre arpenteur.

Répertoire d'images

1.9. En Bourne Shell, écrire un script `test_img.cgi` permettant de visualiser toutes les images contenues dans un répertoire d'images à la racine de `www-iutinfo` le nom du répertoire est passé en paramètre.

1.10. Insérer un lien dans `index.html` permettant de visualiser les icônes dans le répertoire `images/icons` à la racine `web` sur *nyx*.

Affichage des programmes sources

1.11. Ecrivez un script `voir_source.cgi` permettant d'afficher en pre formaté le code source d'un programme, le nom du fichier contenant le code source étant passé en paramètre (utiliser la commande Unix **`cat`**)

1.12. Testez en insérant dans votre page l'url suivante :

1.13. Ecrivez une deuxième version permettant de ne pas interpréter les balises html (utilisez la commande **`sed`** pour remplacer les caractères < et > respectivement par les entités `<` et `>`

Programmes CGI en C

1.14. Ecrivez en C un programme qui affiche le contenu des variables d'environnement dans un tableau HTML (les variables d'environnement sont accessibles en C à l'aide des arguments en ligne).

1.15. Créez un fichier de texte `capitales.txt` contenant les noms des capitales de différents pays, chaque ligne comportera le nom d'un pays séparé du nom de sa capitale par un caractère :

1.16. En C, écrire un programme `test_capitales.c` qui lit le fichier `capitale.txt` lignes par lignes, et renvoie une page html présentant les noms des pays et des capitales dans un tableau.

1.17. Compiler, puis tester le programme `test_capitales.c`.

1.18. Insérer un lien permettant d'exécuter le script `test_capitales.cgi`.

Compteur d'accès

1.19. On désire visualiser sur la page `index.html` un compteur indiquant le nombre d'accès à cette page.

Vous utiliserez le programme `compteur.c` (voir le source en Annexe) qui crée une image bitmap du compteur. Vous trouvez ce programme dans $DOCUMENT_ROOT de Apache sur *nyx*).

1.20. Créer un fichier texte `count.text` dans lequel sera stocké la valeur du compteur, y insérer une première valeur `0000001` (sans retour chariot), et donner le droit d'écriture pour tout le monde.

```
nyx$ echo 0000001"\c" > count.txt
nyx$ chmod o+w count.txt
```

1.21. Dans le source `compteur.c`, affecter la constante LE_COMPTEUR, cette constante spécifie l'emplacement absolu du fichier `count.txt`, compile, puis tester le programme (nommer l'exécutable `compteur.cgi`).

```
nyx$ cc compteur.c -o compteur.cgi
```

1.22. Insérer dans `index.html` une balise **IMG** utilisant le compteur.

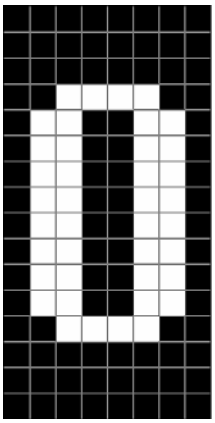
```
Compteur <img src="compteur.cgi"/></img>
```

2. Annexe - Programme compteur.c

Le programme `compteur.c` crée sur la sortie standard une image bitmap (voir en dessous) d'un compteur. La valeur du compteur est stockée dans un fichier texte. A chaque appel le programme lit la valeur du compteur, il incrémente sa valeur, et met à jour le fichier.

Format des fichiers bitmap

Un **bitmap** est une grille rectangulaire de points.
La figure ci-dessous illustre un bitmap baptisé `zero`, dont le contenu est défini en dessous :



```
#define zero_width 8
#define zero_height 16
static char zero_bits[] = {
   0xff, 0xff, 0xff, 0xc3, 0x99, 0x99, 0x99, 0x99,
   0x99, 0x99, 0x99, 0x99, 0xc3, 0xff, 0xff, 0xff};
```

où:

- *width*, *height*: dimensions en pixel du bitmap
- *bits* : tableau d'octets initialisé avec les valeurs des bits composant le bitmap.

Listing du programme compteur.c

```
#include <stdio.h>
#include <stdlib.h>

/* EMPLACEMENT ABSOLU DU COMPTEUR */
#define   LE_COMPTEUR   "/ . . . . . /web/rx1/TD5/count.txt"

/* Definition bitmap des 10 chiffres */
/* Chaque chiffre est correspond a 8*16 points */
char *digits[] =
      {"0xff","0xff","0xff","0xc3","0x99","0x99","0x99","0x99",
            "0x99","0x99","0x99","0x99","0xc3","0xff","0xff","0xff",
            "0xff","0xff","0xff","0xcf","0xc7","0xcf","0xcf","0xcf",
            "0xcf","0xcf","0xcf","0xcf","0xcf","0xff","0xff","0xff",
            "0xff","0xff","0xff","0xc3","0x99","0x9f","0x9f","0xcf",
            "0xe7","0xf3","0xf9","0xf9","0x81","0xff","0xff","0xff",
            "0xff","0xff","0xff","0xc3","0x99","0x9f","0x9f","0xc7",
            "0x9f","0x9f","0x9f","0x99","0xc3","0xff","0xff","0xff",
            "0xff","0xff","0xff","0xcf","0xcf","0xc7","0xc7","0xcb",
            "0xcb","0xcd","0x81","0xcf","0x87","0xff","0xff","0xff",
```

```
           "0xff","0xff","0xff","0x81","0xf9","0xf9","0xf9","0xc1",
           "0x9f","0x9f","0x9f","0x99","0xc3","0xff","0xff","0xff",
           "0xff","0xff","0xff","0xc7","0xf3","0xf9","0xf9","0xc1",
           "0x99","0x99","0x99","0x99","0xc3","0xff","0xff","0xff",
           "0xff","0xff","0xff","0x81","0x99","0x9f","0x9f","0xcf",
           "0xcf","0xe7","0xe7","0xf3","0xf3","0xff","0xff","0xff",
           "0xff","0xff","0xff","0xc3","0x99","0x99","0x99","0xc3",
           "0x99","0x99","0x99","0x99","0xc3","0xff","0xff","0xff",
           "0xff","0xff","0xff","0xc3","0x99","0x99","0x99","0x99",
           "0x83","0x9f","0x9f","0xcf","0xe3","0xff","0xff","0xff"
           };


main () {
       FILE *fp = NULL;
       FILE *out = NULL;
       char numb[7];
       char hold[8]= "00000000";
       char cc[]= "0";
       int holdlen;
       int num, len, x, y, c, i;

/* On recupere l'ancien nombre d'acces dans le fichier count.txt */
   fp = fopen(LE_COMPTEUR,"r");
   fgets(numb, 8, fp);
   fclose(fp);
   sscanf(numb,"%d",&num);
/* On incremente de 1 le nombre d'acces */
   num++;
/* On met a jour le fichier count.txt */
   out = fopen(LE_COMPTEUR,"w");
   fprintf(out,"%d",num);
   fclose(out);
/* On met Si le nombre d'acces est 1234 alors numb="1234" */
/* et hold="00001234" */
   len = strlen(numb);
   for (i=0; i<len; i++) {
        hold[8-len+i] = numb[i];
   }

/* Creation de l'entete de la sortie standard du script */
   printf ("Content-type: image/x-xbitmap%c%c",10,10);

/* Creation du corps */
   printf ("#define count_width 56\n");
   printf ("#define count_height 16\n");
   printf ("static char count_bits[] = {\n");
/* Le bitmap est ecrit sur la sortie standard */
/* ligne par ligne */
   for (x=0; x<16; x++) {
        for (y=1; y<8; y++) {
             cc[0]=hold[y];
             sscanf(cc,"%d",&c);
             printf(digits[((c*16)+x)]);
             if (y<7) { printf(", "); }
        }
        if (x==15) { printf("};");}{printf(",\n");}
   }
   printf("\n");
}
```