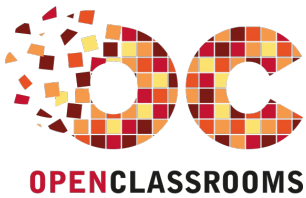
Parcourir les dossiers avec dirent.h

Par Monsieur_JaKy



www.openclassrooms.com

Dernière mise à jour le 20/10/2009

Sommaire

Parcourir les dossiers avec dirent.h

Par Monsieur_JaKy
Mise à jour : 20/10/2009
Difficulté : Intermédiaire

Bonjour à tous !

Sur la plupart des forums, on peut voir des Zér0s se poser ce genre de question : « Comment parcourir les dossiers en C ? », « Comment lister les fichiers qui se trouvent dedans ? », etc.

Ce mini-tuto est pour eux ! Grâce au fichier d'en-tête **dirent.h**, nous allons pouvoir faire ce genre de chose, et plus encore !

Les bases du C sont nécessaires pour suivre et comprendre ce tutoriel. Si vous pensez ne pas les posséder, lisez le tutoriel officiel du site.

N.B. : Si vous êtes à l'aise concernant la manipulation des fichiers, alors ce tutoriel ne devrait *a priori* vous poser aucun problème.

Sommaire du tutoriel :

Ouverture et fermeture de dossiers

Le point sur dirent.h

Pour commencer, qu'est-ce que dirent.h ? Comme le .h pouvait le laisser supposer, c'est un header, et il va falloir l'inclure dans nos programmes.

Ce header est *portable*, il est donc disponible sur Windows, Mac et Linux. D'ailleurs, vous pouvez aller voir dans le dossier include de votre compilateur, il s'y trouve bel et bien.

Mais si j'ai mis le mot « portable » en italique, c'est parce qu'il ne l'est pas totalement. dirent.h est un header sous la norme POSIX... Ce qui signifie que c'est un header **standard** pour les systèmes UNIX. Les codes et les possibilités seront exactement les mêmes pour Linux et Mac par exemple.

Mais... et pour Windows alors ?

dirent.h est bien disponible sous Windows, puisqu'il a été adapté sur cet OS. Néanmoins, certaines petites choses sont différentes, mais dans les grandes lignes, rien ne change énormément, ne vous inquiétez pas.

Les directives de préprocesseur

Je vois que vous en avez déjà marre du blabla. Bien. Passons donc au code.

Nous allons avoir besoin de quelques headers à inclure :

Code : C

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

#include <dirent.h>

#ifndef WIN32
#include <sys/types.h>
#endif
```

J'ai décidé d'inclure stdio et stdlib pour pouvoir utiliser quelques fonctions standards. Vient ensuite notre fameux dirent.

Qu'est-ce que c'est sys/types ?

Cela fait partie des changements dont j'ai parlé précédemment. Ces lignes de codes signifient : « Si on n'utilise **pas** un système d'exploitation Windows, alors on inclut sys/types.h ».
Ce header contient des types « inventés » qui ne seront pas forcément indispensables au long de ce chapitre, mais qui se révèleront utiles. Sous Windows (ne me demandez pas pourquoi), on n'a pas besoin d'utiliser ces types.

Ouvrir un répertoire

Vous allez voir, les prototypes des fonctions d'ouverture et de fermeture de dossier ne vont pas vous dépayser des fichiers, la manière de procéder est grosso modo la même.

Code : C

```
DIR* opendir(const char* chemin) ;
```

opendir() attend une chaîne de caractères et renvoie un pointeur sur ~~FILE~~ DIR.

Cékoiça DIR ? Et quel chemin faut-il mettre ?

Je vais y aller pas à pas.
DIR est une structure qui *simule* le dossier que l'on a ouvert. D'ailleurs, petite question... Que signifie DIR ? Directory ! Et que signifie directory ? ... C'est la traduction anglaise de **répertoire** !
Ensuite, le chemin à mettre, ça, vous avez appris à le faire dans le tutoriel officiel sur la manipulation des fichiers ! Si vous avez un petit trou de mémoire, c'est par ici. N'oubliez pas d'adapter le chemin pour chaque OS ("C:\" pour Windows, "/" pour UNIX).

Si le dossier n'a pas pu être ouvert, opendir() renverra NULL.

Fermer un répertoire

Code : C

```
int closedir(DIR* repertoire);
```

On dirait fclose pas vrai ?
Cette fonction attend un pointeur sur DIR et renvoie un int qui correspond à un code d'erreur, 0 si la fermeture s'est bien exécutée, -1 s'il y a eu un souci.

Le code complet commenté

Code : C

```c
#include <stdlib.h>
#include <stdio.h>
/* stdlib pour exit(), et stdio pour puts() */

#include <dirent.h>
/* Pour l'utilisation des dossiers */

#ifndef WIN32
#include <sys/types.h>
#endif

int main(void)
{
    DIR* rep = NULL;
    rep = opendir("D:\\Mes Documents\\JaKy"); /* Ouverture d'un
dossier */

    if (rep == NULL) /* Si le dossier n'a pas pu être ouvert */
        exit(1); /* (mauvais chemin par exemple) */


    puts("Le dossier a ete ouvert avec succes");

    if (closedir(rep) == -1) /* S'il y a eu un souci avec la
fermeture */
        exit(-1);

    puts("Le dossier a ete ferme avec succes");

    return 0;
}
```

Je pense que ce code est suffisamment explicite pour ne pas être expliqué.

Utiliser errno

errno ? Drôle de nom... Cela a-t-il un rapport direct avec la manipulation des dossiers ?

Non en effet, les deux ne sont pas directement liés. Néanmoins, j'ai décidé d'en toucher quelques mots pour deux raisons :

- si j'en parle, c'est qu'il y a quand même un rapport avec dirent ;
- j'ai l'impression que l'utilisation d'errno est très méconnue sur le SdZ... Elle est pourtant très utile, alors pourquoi ne pas vous expliquer son fonctionnement ?

La variable errno

Entrons dans le vif du sujet. Je pose le problème : vous avez essayé d'ouvrir un dossier mais pour une raison obscure, l'ouverture a échoué. Comment faire pour savoir d'où vient le problème ? Est-ce tout simplement parce que le dossier n'existe pas, ou peut-être est-il protégé ?
errno est là pour tout nous éclaircir.
Cette variable (globale) de type **int** peut prendre plusieurs valeurs de defines prédéfinies. Si vous regardez ce lien, vous pouvez voir une longue liste de defines ayant chacun une valeur (valeur que nous ne connaissons pas, et d'ailleurs on s'en moque bien).
Quand une erreur est déclarée, errno prend la valeur d'une de ces valeurs pour permettre au programmeur de connaître la source

du problème.

Comment utiliser errno ?

C'est une simple variable, vous n'avez qu'à faire un test d'égalité pour connaître sa valeur :

Code : C

```
if (errno == EACCES) /* Accès interdit */
    puts("Acces interdit");
```

N'oubliez pas d'inclure errno.h dans votre code, sinon c'est l'erreur de compilation assurée !

Une autre manière d'utiliser errno

Utiliser errno de cette façon, c'est bien, mais c'est assez barbant de faire des tests d'égalité, surtout quand on voit qu'il y a environ 40 valeurs d'erreurs possibles à tester... C'est pour cette raison qu'il existe une autre fonction, **perror()** permettant d'obtenir les erreurs dans un format lisible.

Code : C

```
void perror(const char* str) ;
```

La fonction attend une chaîne de caractères qui sera écrite à l'écran (à la manière d'un printf()). Ensuite, perror() se contentera de décrire l'erreur (*in English of course*).

Code : C

```
/* Imaginons que errno ait pris la valeur ENOENT. */
perror("Erreur : ");
```

Code : Console

```
Erreur : : No such file or directory
```

Ce qui signifie : Pas de fichier ou de répertoire.

C'est pratique. Mais pourquoi y a-t-il deux points en plus ?

Si la chaîne envoyée à perror est nulle, alors les deux points ne seront pas imprimés. Mais si vous envoyez une chaîne de caractères non nulle, ils le seront.

Coupler perror et opendir

Maintenant que vous connaissez perror(), tout devient plus simple pour connaître la source d'une erreur !

Code : C

```
DIR* rep = NULL;
rep = opendir("D:\\Mes Documents\\JaKy"); /* Ouverture d'un dossier
*/

if (rep == NULL) /* Si le dossier n'a pas pu être ouvert, */
```

```
        perror(""); /* perror() va nous écrire l'erreur. */
```

Sachez que errno, c'est encore mieux que ça ! Si vous regardez la documentation de la fonction opendir (man pages), vous pourrez voir une liste de valeurs que errno peut prendre.
D'ailleurs, vous pouvez faire de même pour la fonction closedir(). S'il y a échec de cette fonction, errno prendra la valeur EBADF.

Rappelez-vous : errno ne se limite pas à la manipulation des dossiers. D'autres fonctions standards l'utilisent, comme fopen() par exemple.

Manipuler des fichiers à l'intérieur d'un dossier

Après s'être intéressés à errno, nous allons pouvoir coder des choses intéressantes. Maintenant que l'on sait ouvrir et fermer un répertoire, pourquoi n'irait-on pas regarder ce qui se trouve dedans ?

La fonction readdir

C'est sûrement la fonction la plus importante du tutoriel. Voyons son prototype :

Code : C

```
struct dirent* readdir(DIR* repertoire) ;
```

readdir() prend en paramètre un pointeur sur DIR ; ça, vous connaissez.
Par contre elle renvoie... un pointeur sur une structure dirent.

Eh bien cette structure, elle *simule* le fichier du dossier qui sera lu ! C'est une structure qui est également définie dans dirent.h. D'ailleurs, encore un peu d'*english* : dirent signifie **DIRectory ENTity**, qui signifie entité de répertoire, qui signifie : c'est l'objet que contient un répertoire (dossier, fichiers...).

Pour ne pas vous embrouiller avec le mot entité, j'emploierai le terme de fichier, mais ne faites pas de confusions avec les FILE*.

readdir() renvoie NULL s'il n'y a plus de fichier à lire.

Vous voulez du code ? Parfait. Ce code permet de lire le premier fichier d'un répertoire (je vous passe les directives de préprocesseur) :

Code : C

```
int main()
{
    DIR* rep = NULL;
    struct dirent* fichierLu = NULL; /* Déclaration d'un pointeur
vers la structure dirent. */
    rep = opendir("D:\\Mes Documents\\JaKy");
    if (rep == NULL)
        exit(1);

    fichierLu = readdir(rep); /* On lit le premier répertoire du
dossier. */

    if (closedir(rep) == -1)
        exit(-1);
```

```
        return 0;
    }
```

Après l'appel de readdir, le pointeur de structure *fichierLu* pointe vers le premier répertoire du dossier JaKy !

Le mot-clé struct me gêne ! J'ai essayé de l'enlever et j'ai une erreur de compilation : dirent undeclared...

En effet, les développeurs n'ont pas fait de typedef à cette structure. Néanmoins, je suis persuadé que vous savez le faire. 

Lister le contenu d'un répertoire

C'est bien joli d'utiliser readdir mais... comment savoir quel est le nom du fichier qui est lu ?
C'est bien simple : la structure dirent contient une variable membre, une chaîne de caractères précisément, appelée **d_name**, qui contient le nom de notre fichier !

Code : C

```
    fichierLu = readdir(rep);
    printf("Le fichier lu s'appelle '%s'", fichierLu->d_name); /* On
    utilise la flèche
    car fichierLu est un POINTEUR de structure. */
```

Résultat :

Code : Console

```
Le fichier lu s'appelle '.'
```

Gné ? Pourquoi on a droit à un point ? 

Le point représente le dossier actuel, celui dans lequel on se trouve ! Si vous connaissez la commande cd, alors :

Code : Bash

```
    cd ./Machin/Truc
```

Ceci devrait vous éclairer l'esprit. 

Je vous propose maintenant de continuer à lire notre répertoire. On va donc rajouter une 2e fois la fonction readdir().

Code : C

```
    fichierLu = readdir(rep);
    printf("Le fichier lu s'appelle '%s'\n", fichierLu->d_name);
    fichierLu = readdir(rep);
    printf("Le fichier lu s'appelle '%s'", fichierLu->d_name);
```

Code : Console

```
Le fichier lu s'appelle '.'
```

```
Le fichier lu s'appelle '..'
```

Les deux points représentent le dossier précédent, le dossier qui *contient* celui dans lequel on se trouve.

Continuons encore un peu. Comme nous voulons lister TOUT le répertoire, il va donc falloir utiliser une boucle qui lira le fichier tant que l'on n'est pas à la fin du dossier... autrement dit, tant que notre fichier est différent de NULL, je l'avais précisé juste avant.

Code : C

```c
while ((fichierLu = readdir(rep)) != NULL)
    printf("Le fichier lu s'appelle '%s'\n", fichierLu->d_name);
```

Normalement vous devriez reconnaître la syntaxe de la boucle while() dans le TP du Pendu et dans l'annexe sur la saisie sécurisée. On lit le répertoire tant qu'il y a quelque chose à lire, et on affiche le nom.

Pour savoir quel fichier readdir() doit lire, un curseur virtuel est créé : à chaque appel de cette fonction(), le curseur avance d'un pas.

Ouvrir un fichier

O.K., maintenant, on sait lire et afficher le contenu de tout le répertoire. Afficher, c'est bien beau, mais pas très utile, alors pourquoi ne pas charger quelque chose, comme un fichier .txt ou un .jpg par exemple ?
La plupart des fonctions permettant de charger une ressource (comme SDL_LoadBMP (de SDL) pour les .bmp, FSOUND_Sample_Load (de FMOD) pour un .mp3, ou même opendir) attendent, comme argument, une chaîne de caractères qui elle-même contient le nom de la ressource à charger.
Quelle coïncidence ! La variable membre de la structure dirent est aussi une chaîne de caractères !
Conclusion : pour charger quelque chose, il suffit juste de passer en paramètre **d_name**.

Code : C

```c
/* Imaginons que d_name = "image.bmp" */
img = SDL_LoadBMP(fichierLu->d_name);
```

Si vous utilisez des chemins absolus, vous devez créer une nouvelle chaîne assez grande contenant C:\Program Files\ + leNomDuFichier par exemple, car d_name contient seulement le nom du fichier lu et **PAS** son arborescence.

Déplacement dans un répertoire

Ça y est ! On a fini le "gros" du tutoriel. Néanmoins, dirent.h propose quelques fonctions utiles à connaître dont il serait impensable que je ne parle pas.

Théorie

Cette partie est uniquement théorique, la pratique arrive juste après, je vous expliquerai pourquoi.

Juste avant, je vous ai parlé d'un curseur virtuel, qui s'incrémente à chaque fois que l'on lit un fichier. Je vais vous montrer les quelques fonctions qui permettent de l'utiliser.

Où sommes-nous dans le répertoire ?

Il existe une fonction qui nous permet de savoir où se trouve ce curseur. Si vous avez de la mémoire (concernant les fichiers) et

un peu de jugeote, vous ne devriez pas peiner à trouver le nom de la fonction... c'est **telldir()**.

Code : C

```
long telldir(DIR* rep) ;
```

Je ne vais même pas détailler cette fonction tellement elle coule de source.

Si on retournait au début ?

La fonction **rewinddir()** permet de faire cela :

Code : C

```
void rewinddir(DIR* rep) ;
```

Je vous l'avais dit : ça ressemble énormément aux fichiers !

Je veux aller ici !

La fonction **seekdir()** permet de se positionner dans un dossier.

Code : C

```
void seekdir(DIR* rep, long pos) ;
```

Elle attend un répertoire, et un long correspondant à la position à laquelle le curseur doit se placer.

Pratique

La raison pour laquelle j'ai fait deux parties est la suivante : il y a des différences (celles dont je vous ai parlé au début du tutoriel) entre Unix et Windows. L'utilisation des fonctions énoncées juste avant est plus *simple* sous Windows par rapport à un système UNIX (ce n'est pas un troll je vous rassure) : par exemple, la fonction telldir() renverra des positions lisibles pour nous alors que ce n'est pas le cas pour les systèmes UNIX. Je vais donc découper la suite en 2 partie : une pour Unix et une autre pour Windows.

Je vous parlerai uniquement de telldir() et seekdir(), puisque la fonction rewinddir() ne change pas.

Windows

Pour prouver ce que j'ai dit précédemment, un petit exemple s'impose :

Code : C

```
while ((fichierLu = readdir(rep)) != NULL)
    printf("%ld -> %s\n", telldir(rep), fichierLu->d_name);

seekdir(rep, 2); /* On se place au 2e fichier. */
printf("\n%s\n", readdir(rep)->d_name);

seekdir(rep, 4); /* On se place au 4e fichier. */
printf("\n%s\n", readdir(rep)->d_name);
```

Code : Console

```
1 -> .
2 -> ..
```

```
3 -> Boulot
4 -> Lettre.doc
5 -> guitare.mp3
6 -> text.txt

Boulot

guitare.mp3
```

Comme readdir() avance d'un pas, nos informations sont décalées. 

Voilà, vous pouvez vous amuser à vous placer, déplacer, un p'tit coup de rewinddir, etc.  Il n'y a rien de déroutant ici.

Unix

Vous allez peut-être trouver ça étrange, mais la valeur renvoyée par la fonction telldir() n'est pas faite pour être comprise par les humains.  D'ailleurs, vous pouvez essayer par vous-même :

Code : C

```c
for (i = 0; i < 5; i++)
{
    ent = readdir(rep);
    printf("%s a la position %ld\n", ent->d_name, telldir(rep));
}
```

Vous pouvez constater que pour la position numéro 1, on a bien droit à 1.  Mais par la suite, vous allez avoir des chiffres faramineux comme 17548554 ou 754104119...

Mais alors, cette fonction ne nous sert à rien ! Et c'est pareil pour seekdir, non ?

En effet, pour la première question, utiliser telldir() tel quel ne nous sert à rien... Mais couplée à seekdir(), elle peut se révéler utile !

La fonction seekdir(), elle aussi, attend une position incompréhensible pour nous... mais elle comprend la position renvoyée par telldir().
Vous ne voyez pas où je veux en venir ? Eh bien, imaginez que vous avez besoin de lire un certain nombre précis de fichiers, et que vous devez retourner au début du répertoire, puis retourner au dernier fichier qu'on a lu tout à l'heure... Vous n'allez pas refaire une boucle de readdir(). 

Code : C

```c
for (i = 0; i < 5; i++)
{
    ent = readdir(rep); /* On lit jusqu'au 5e fichier du
répertoire. */
}

pos = telldir(rep); /* On récupère la position actuelle dans une
variable de type long. */
printf("%s : %ld\n", ent->d_name, pos);

rewinddir(rep); /* Pour une quelconque raison, on doit revenir au
début. */

seekdir(rep, pos); /* On se place à la 5e position en envoyant la
variable pos. */

ent = readdir(rep); /* On lit le répertoire suivant. */
```

Voilà la seule utilisation possible de seekdir() et de telldir(). C'est certes compliqué, mais de toute façon, il y a peu de chances que vous utilisiez le curseur virtuel vous-même.

Exercices

Quoi de mieux pour s'entraîner, me direz-vous ?
Dans cette partie, je vais vous proposer 3 exercices, à difficulté croissante.

Compter le nombre de fichiers

Créez une fonction qui attend un pointeur sur DIR* et qui renvoie un int contenant le nombre de fichiers et/ou de dossiers (vous pouvez faire les 2) dans ce répertoire. Attention : les dossiers parents et actuels (. et ..) ne seront pas comptés !

Corrigé

Secret (cliquez pour afficher)

Code : C

```c
int compterFichier(DIR* dir)
{
    int nbr = 0;
    struct dirent* ent = NULL;

    while ((ent = readdir(dir)) != NULL)
    {
        if (strcmp(ent->d_name, ".") != 0 && /* Si le fichier lu n'est pas . */
            strcmp(ent->d_name, "..") != 0) /* Et n'est pas .. non plus */
        nbr++; /* Alors on incrémente le compteur */
    }

    return nbr;
}
```

Pas très très difficile n'est-ce pas ?

Savoir si le fichier lu est un répertoire ou non

Votre mission, si vous l'acceptez, est la suivante : vous devez créer une fonction qui prend en paramètre un pointeur sur une structure dirent (ou une chaîne de caractères qui contiendrait un chemin) et qui renvoie 1 si ce pointeur de structure est un répertoire, ou 0 s'il n'en est pas un.

Code : C

```c
int isDir(struct dirent* ent) ;
```

Voici un très très gros indice si vous ne voyez pas comment procéder.

Secret (cliquez pour afficher)

Vérifier si le nom du fichier a une extension (.jpg, .mp3, etc.).

Corrigé

Secret (cliquez pour afficher)

Code : C

```
int isDir(struct dirent* ent)
{
    if ((strchr(ent->d_name, '.')) == NULL) /* Si le nom du
fichier n'a pas de point (une extension). */
        return 1;
    else
        return 0;
}
```

Se déplacer librement dans un répertoire

Là, on passe à un cran au-dessus ! Le but est de se déplacer librement dans une arborescence de répertoire. Voici comment le programme devra se dérouler.

- L'utilisateur ouvre un dossier de son choix par le biais d'une chaîne de caractères sous la forme "nomDuDossier/". Si l'entrée n'existe pas, on quitte tout.
- On affiche tous les fichiers du répertoire ouvert.
- L'utilisateur entre une nouvelle chaîne de caractères. Si c'est un fichier, on le précise et on ferme le programme, si c'est un répertoire, on efface l'écran, on entre dans ce répertoire, et on affiche les fichiers qui sont dedans.
- Etc.

Voilà qui devrait vous occuper plus de temps. Mais je ne vais tout de même pas vous abandonner là, je vais vous donner quelques conseils pour procéder :

Secret (cliquez pour afficher)

Créer un main() contenant 2 variables : le chemin du fichier et un DIR*. Envoyez ces variables à une fonction parcourirDossier. La fonction parcourirDossier appellera d'autres fonctions : une fonction lire() permettant de lire le chemin, une fonction lireDossier permettant de faire une boucle de readdir() et faire en sorte qu'on rappelle la fonction parcourirDossier pour continuer... Voici quelques pistes pour démarrer.

Corrigé

Secret (cliquez pour afficher)

Code : C

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

#ifndef WIN32
#include <sys/types.h>
#define CLEAR "clear"
    /* system("clear") pour UNIX */
#else
#define CLEAR "cls"
    /* system("cls") pour Windows */
#endif

int isDir(char* s)
{
    if ((strchr(s, '.')) == NULL) /* Si le nom du chemin n'a pas
de point (une extension). */
        return 1;
    else
```

```
        return 0;
}

void lire(char* s)
{
    char *enter = NULL;
    char temp[100] = ""; /* Chaîne de caractères temporaire
contenant la saisie de l'utilisateur. */

    fgets(temp, 99, stdin);

    enter = strchr(temp, '\n'); /* Voir le chapitre des saisies
sécurisées. */
    if (enter != NULL)
        *enter = '\0';

    strcat(s, temp); /* On ajoute à la suite le nom du dossier
pour obtenir quelque chose comme C:/nom/nom/ pour Win
ou /nom/nom/ pour UNIX. */
}


void lireDossier(char* s, DIR* rep)
{
    struct dirent* ent = NULL;

    printf(" -- Lecture du dossier '%s' -- \n", s);

    while ((ent = readdir(rep)) != NULL) /* Lecture du dossier.
*/
        printf("%s\n", ent->d_name);

    printf("\n\n -- Que voulez-vous ouvrir -- ?\n");
}

void parcourirDossier(DIR* rep, char* chemin)
{
    lire(chemin); /* Lecture du nouveau chemin; */

    system(CLEAR); /* On efface l'écran. */

    if (!isDir(chemin))
    {
        printf("%s n'est pas un dossier", chemin);
        exit(-1);
    }

    rep = opendir(chemin);

    if (rep == NULL)
    {
        printf("Le dossier '%s' n'a pas pu etre ouvert", chemin);
        exit(-1);
    }

    lireDossier(chemin, rep); /* Lecture... */

    closedir(rep); /* Fermeture du répertoire. */

    parcourirDossier(rep, chemin); /* On rappelle la fonction
parcourirDossier (récursivité). */
}

int main()
{
    DIR* rep = NULL;
    char chemin[500] = "";

    parcourirDossier(rep, chemin);
```

```
        closedir(rep);

        return 0;
    }
```

Si vous avez réussi ce programme sans regarder la solution, je vous félicite. Pour les autres, l'exercice étant difficile, ce n'est pas très grave. Néanmoins, jetez un coup d'œil à mon code source, essayez de le comprendre et de faire... quelques améliorations !

Améliorations

Pour vous exercer (et aussi parce que le code ci-dessus a encore quelques défauts), voici quelques améliorations que vous pouvez implémenter.

- C'est assez barbant pour l'utilisateur de devoir retaper "/" à chaque fois que l'on entre dans un nouveau dossier... Pourquoi ne pas faire en sorte qu'il n'ait pas besoin de le taper ?
- Le point (.) ne marche pas... Ré-affichez le dossier actuel.
- Plus difficile : les deux points (..) qui permettent de retourner au dossier précédent.
- Faire un petit menu qui contiendrait plusieurs actions : rename, remove...
- Si le fichier lu n'est pas un dossier, utilisez un FILE* et affichez le contenu du fichier (attention à certains formats comme les images, c'est assez... , préférez les .txt !).

Et maintenant, histoire de fatiguer vos méninges encore plus, voici un QCM.

Q.C.M.

Le premier QCM de ce cours vous est offert en libre accès.
Pour accéder aux suivants

Pourquoi on ne considère pas dirent.h comme *portable* ?

- ◯ À la base, c'est un header pour Windows, mais a été porté pour les systèmes UNIX.
- ◯ dirent.h est portable.
- ◯ Il n'existe pas sous Windows.
- ◯ À la base, c'est un header pour UNIX, mais il a été porté pour Windows.

Quel est le problème dans ce code :

Code : C

```
int main()
{
    DIR* rep;

    rep = opendir(".");

    if (rep != NULL)
        return 1;

    closedir(rep);

    return 0;
}
```

- ◯ Tu n'as pas initialisé le pointeur sur DIR.
- ◯ La condition d'ouverture de répertoire est mauvaise.
- ◯ Tu as oublié le mot struct avant DIR.
- ◯ "." n'est pas un répertoire.

- ◯ DIR doit être en minuscules.

Sous Windows, quel est le problème avec les fonctions de manipulation de curseur (telldir() et seekdir()) ?

- ◯ Les valeurs peuvent être parfois erronnées.
- ◯ Il n'y a pas de problème.
- ◯ Les valeurs renvoyées ne sont pas compréhensibles par les humains.

Nous avons un pointeur de structure sur dirent du nom de **ent** et voulons savoir son nom. Comment y accéder ?

- ◯ ent.d_name.
- ◯ ent->d_name.
- ◯ ent->name.
- ◯ ent->_name.

Correction !

Statistiques de réponses au QCM

Vous êtes arrivés à la fin de ce tutoriel ! Félicitations !

Avant de nous quitter, j'ai plusieurs choses à vous dire : sous Windows, pour manipuler les répertoires de manière standard, vous ~~devez~~ pouvez utiliser l'API Windows. Un début de piste ici. Car comme je l'ai dit au début, dirent est un standard pour UNIX qui a été porté pour Windows par la suite.

Je tiens à énormément remercier **Pierre89**, qui m'a donné accès au tutoriel qu'il avait fait sur le même sujet et qui n'avait malheureusement pas été validé, qui m'a donné une base pour écrire le mien, et qui m'a donné quelques renseignements ; un merci à **Lunaro** aussi pour la relecture du tutoriel, et à **Nanoc** pour ses conseils.

Sur ce, j'espère vous avoir aidé.
Monsieur_JaKy.

Ce tutoriel a été corrigé par les zCorrecteurs.