LES FICHIERS - LES REPERTOIRES

Utilisation des fonctions dites de "haut niveau"

1. OUVERTURE D'UN FICHIER : LA FONCTION fopen()

Il faut d'abord déclarer une variable de type **flux** appelée habituellement **pointeur de fichier**.

La déclaration **FILE *fp ;**

déclare une variable **fp** de type **flux** ou **pointeur de fichier**.

Le type **FILE** est défini dans **<stdio.h>** .

L'ouverture d'un flux associé à un fichier est effectuée par la fonction

FILE *fopen (const char *filename , const char *mode)

La fonction **fopen()** ouvre le fichier dont le nom est donné par **filename** et retourne un **flux** ou **pointeur de fichier** associé, elle retourne **NULL** si la tentative échoue.
Les valeurs possibles de l'argument **mode** sont :

"r"	ouvre un fichier en mode de lecture.
"w"	crée un fichier en écriture; écrase le contenu précédent si le fichier existait déjà.
"a"	ajout : ouvre ou crée un fichier et se positionne en écriture à la fin du fichier.
"r+"	ouvre un fichier en mode mise à jour, c.a.d. en lecture/écriture.
"w+"	crée un fichier en mode mise à jour, c.a.d. en lecture/écriture, mais écrase le contenu précédent si le fichier existait déjà.
"a+"	ajout : ouvre ou crée un fichier en mode mise à jour et se positionne en écriture à la fin du fichier.

Un fichier peut être qualifié de " binaire " car, par définition, il ne contient que des octets. Mais, malgré cette réalité, on distingue habituellement :

- Le fichier en mode " binaire " qui est constitué d'un bloc d'octets qui, par exemple, peuvent représenter le code machine d'un programme exécutable, des tableaux de données, des tableaux de structures, ...
- Le fichier en mode " texte " utilisé pour stocker du texte constitué d'une suite de lignes, chaque ligne étant constituée de caractères (éventuellement aucun pour une ligne vide) se terminant par le caractère d'échappement fin de ligne '\n'. Cette distinction vient du fait que certains caractères peuvent être interprétés et convertis par le système lorsque le fichier est en mode texte. Par exemple, le caractère '\n' dans un fichier en mode texte sera converti immédiatement en une séquence 0xD 0xA.

On peut aussi préciser dans l'argument **mode** si le fichier est binaire ou texte; c'est fait en suffixant **mode** par respectivement **t** pour texte ou **b** pour binaire, par exemple : "rt" pour une

ouverture d'un fichier texte en lecture, "wb" pour une création d'un fichier binaire, "r+t" pour une ouverture d'un fichier texte en mode mise à jour.
Si l'argument **mode** ne précise pas binaire ou texte, le mode d'ouverture est alors imposé par la valeur de la variable globale **_fmode** qui indique un mode texte par défaut.
Attention : L'ajout de b n'a aucune importance sous Linux.

Fragment de programme exemple

```
FILE  *fp ;

if  ( (fp = fopen( "doc1.txt" , "r+" ) ) ==  NULL )
        {
        puts("Erreur ouverture fichier !") ;
        exit (1) ;
        }
```

On tente ici d'associer le flux fp au fichier doc1.txt situé dans le répertoire courant.

Attention Sous Dos / Windows il est nécessaire de doubler les antislash (\ ) dans les chaînes de caractères contenant les noms de fichiers (par exemple c:\\mesdoc\\doc1.txt).

Les fonctions fdopen() et freopen()

FILE *fdopen (int desc , const char *mode)
FILE *freopen (const char *filename , const char *mode , FILE *flux)

La fonction **fdopen()** associe un flux avec un descripteur **desc** de fichier existant. Le mode donné doit être compatible avec celui du descripteur de fichier.

La fonction **freopen()** ouvre le ficher dont le nom se trouve dans le tampon pointé par **filename** et lui associe le flux pointé par **flux**.. Le flux original, s'il existe, est fermé. L'argument **mode** s'utilise comme avec **fopen()**. La principale utilisation de **freopen()** est de modifier le fichier associé aux flux standards **stdin**, **stdout** et **stderr**.

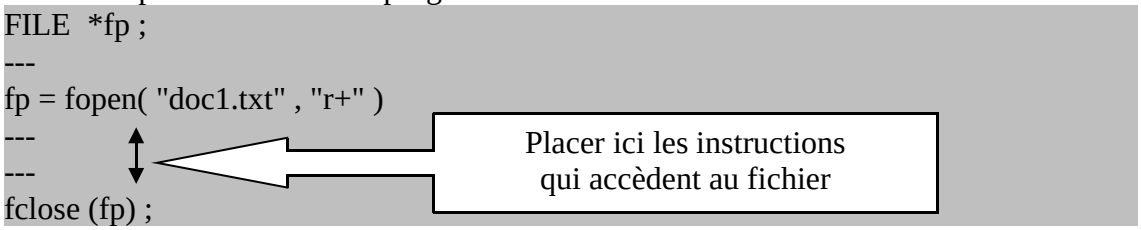
2. FERMETURE D'UN FICHIER : LA FONCTION fclose()

La fermeture du flux associé au fichier est assuré par la fonction **fclose()**.

int fclose(FILE *stream) ;

fclose force l'écriture des données non écrites dans le flux, libère tous les tampons alloués et ferme le flux **stream**. Elle retourne EOF en cas d'erreur, 0 sinon.

Le squelette simplifié de base d'un programme est le suivant :



3. LECTURE ET ECRITURE

Le langage C propose de nombreuses fonctions. On doit choisir les fonctions les plus appropriées au problème posé.

3.1 Entrées/sorties de caractères ou chaînes de caractères

3.1.1 Lecture

int fgetc (FILE *stream) ;

fgetc retourne le caractère suivant depuis le flux **stream** ou bien EOF si la fin de fichier est atteinte; le caractère est lu comme un unsigned char puis est converti en un int dans la valeur retournée.

int getc (FILE *stream) ;

getc est équivalente à **fgetc**.

char *fgets (char * s , int n , FILE *stream) ;

fgets lit au plus les **n-1** caractères suivants du flux **stream** ou bien s'arrête au caractère **'\n'** s'il est rencontré; elle place les caractères lus dans le tampon pointé par **s** et ajoute le caractère fin de chaîne **'\0'** à la fin de la ligne dans le tampon. Elle retourne **s** ou bien NULL si la fin de fichier est atteinte ou si une erreur survient.

3.1.2 Ecriture

int fputc (int car , FILE *stream) ;

fputc écrit le caractère **car** (car est converti en unsigned char) dans le flux **stream**; elle retourne le caractère écrit ou EOF en cas d'erreur.

int putc (int car , FILE *stream) ;

putc est équivalente à **fputc**.

int fputs (const char *s , FILE *stream) ;

fputs écrit la chaîne **s** sur le flux **stream sans écrire le '\0' final**, elle retourne une valeur positive ou nulle ou bien EOF en cas d'erreur.

1er exemple : création d'un fichier texte

On crée un fichier contenant 4 chaînes de caractères tapées par l'utilisateur, on insère dans le fichier le caractère " fin de ligne " à la fin de chaque chaîne.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int i  ;
char  t[200] ;
FILE  *fp ;
if  ( (fp = fopen("texte2","w+")) == NULL)
```

```
        { puts("Erreur sur le nom du fichier");
         exit(1) ; }
for ( i=0 ; i < 4 ; i++ )
        { printf ("\nLigne N° %d :", i+1) ;
        scanf("%s",t) ;
        fputs (t , fp ) ;          /* écriture de la chaîne */
        fputc('\n' ,fp) ;          /* écriture du retour à la ligne */
        }
fclose(fp) ;
}
```

2ème exemple : lecture du fichier texte du 1er exemple
On effectue la lecture chaîne par chaîne.

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

int main()
{
char t[200] ;
FILE *fp ;
if ( (fp = fopen("texte2","r")) == NULL)
        { puts("Erreur sur le nom du fichier");
         exit(1) ; }
while (  fgets ( t , 200 , fp )  != NULL )  printf ("%s",  t) ;
fclose(fp) ;
return 0 ;
}
```

3ème exemple : lecture du fichier texte du 1er exemple
On effectue la lecture caractère par caractère.

```
int main()
{
int i , nb;
char t[200] ;
FILE *fp ;
if ( (fp = fopen("texte2","r")) == NULL)
        { puts("Erreur sur le nom du fichier");
         exit(1) ; }
while (  ( i = fgetc ( fp ) )  != EOF ) printf ("%c", i) ;
fclose(fp) ;
return 0 ;
}
```

3.2 Entrées/sorties de plusieurs éléments

Une structure est ici considérée comme un élément, un tableau est un élément, les types de base sont aussi des éléments : un entier (int) est un élément, un caractère (char) également …
Chaque élément a une taille en octets bien définie.

Les fonctions suivantes permettent de transférer plusieurs éléments identiques.
On rappelle que le type **size_t** est une redéfinition du type **unsigned int** .

size_t fread (void *ptr , size_t size , size_t n , FILE *stream) ;

fread lit sur le flux **stream** au plus **n** éléments de taille **size** et les place à partir de l'adresse **ptr**. Elle retourne le nombre d'éléments lu qui peut être inférieur à **n** si la fin de fichier est atteinte. Elle retourne 0 quand il n'y a plus d'éléments à lire.

size_t fwrite (void *ptr , size_t size , size_t n , FILE *stream) ;

fwrite écrit sur le flux **stream n** éléments de taille **size**, ces **n** éléments résidant à partir de l'adresse **ptr**. Elle retourne le nombre d'éléments écrit qui peut être inférieur à **n** en cas d'erreur.

La fonction fflush()

int fflush(FILE *flux) ;

Force l'écriture dans le flux spécifié de toutes les données en attente.
Retourne 0 si elle réussit, sinon EOF.

1er exemple ce programme déclare une structure et 2 variables de ce nouveau type, puis ouvre un fichier et stocke les 2 variables dans ce fichier.

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

typedef  struct  {
         char nom[20] ;
         char  prenom[20] ;
         int  age ;
         }  personne ;

int  main()
{
FILE *fp ;
personne p[2] ;
strcpy(p[0].nom,"Aimar") ;
strcpy(p[0].prenom ,"Jean");
p[0].age = 30 ;
strcpy(p[1].nom,"Suffit") ;
strcpy(p[1].prenom ,"Sam");
p[1].age = 35 ;
fp = fopen("texte","w+") ;
if ( fwrite( p , sizeof (personne) , 2 , fp ) != 2 )
         printf("Erreur d'écriture\n");
fclose(fp) ;
return 0 ;
}
```

2ème exemple ce programme ouvre le fichier précédent et affiche le résultat de sa lecture à l'écran.

```
typedef struct  {
          char nom[20] ;
        char  prenom[20] ;
        int  age ;
        }  personne ;

int main()
{
FILE *fp ;

personne t[2] ;

fp = fopen("texte","r") ;
if ( fread( t , sizeof (personne) , 2 , fp ) != 2 ) printf("Erreur d'ouverture\n");
fclose(fp) ;

printf("P1 : nom = %s, prénom = %s, âge = %d\n",t[0].nom,t[0].prenom,t[0].age) ;
printf("P2 : nom = %s, prénom = %s, âge = %d\n",t[1].nom,t[1].prenom,t[1].age) ;
return 0 ;
}
```

3.3 Entrées/sorties avec mise en forme

fprintf (FILE * stream , constant char *format [, arguments, ...]) ;

fprintf écrit dans le flux **stream** chaque **argument** avec la spécification correspondante rencontrée dans la chaîne **format**. Sa syntaxe est la même que la fonction **printf** mis à part qu'on peut choisir le flux destination.
fprintf retourne le nombre d'octets écrits ou **EOF** en cas d'erreur.

fscanf (FILE * stream , constant char *format [, arguments, ...]) ;

fscanf lit des données dans le flux **stream**, convertit chaque donnée avec la spécification rencontrée dans la chaîne **format** et stocke le résultat à l'adresse correspondante fournie par **argument**. Sa syntaxe est la même que la fonction **scanf** mis à part le choix du flux source.
fscanf retourne le nombre de champs d'entrée qui ont été successivement lus, convertis et mémorisés avec succès. Elle retourne **EOF** si elle tente de lire à la fin du fichier.

1er exemple création d'un fichier qui contient des données de type différent

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
char *t1 = "Bonjour ", *t2 = "Monsieur du " ;
int t3 = 10 ;
```

```
FILE *fp ;
if ( (fp = fopen("texte3","w+")) == NULL)
          { puts("Erreur sur le nom du fichier");
           exit(1) ;
           }
fprintf (fp , "%s%s%d", t1 , t2 ,t3 ) ;
fclose(fp) ;
return 0 ;
}
```

2ème exemple lecture du fichier précédent
Le chaîne "Bonjour Monsieur du" présente des “ espaces ”, l’espace est un délimiteur de chaîne, il faut donc lire la chaîne mot par mot.

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

int main()
{
char t1[20], t2[20], t3[20]  ;
int t4 ;
FILE *fp ;
if ( (fp = fopen("texte3","r")) == NULL)
          { puts("Erreur sur le nom du fichier");
           exit(1) ;
           }
fscanf (fp , "%s%s%s%d", t1 , t2 ,t3 ,&t4 ) ;
fclose(fp) ;
printf("%s %s %s %d\n", t1, t2, t3, t4) ;
return 0 ;
}
```

4. GESTION DU DEPLACEMENT DANS UN FICHIER

Les données écrites dans un fichier sont des octets rangés les uns derrière les autres, chaque octet a donc son propre emplacement qui détermine sa distance relative “ en nombre d’octets ” par rapport au début du fichier.

On peut associer un “ curseur ” à un fichier pour indiquer l’emplacement où se fera les écritures/lectures dans ce fichier.
Ce curseur est toujours mis au début du fichier dans tous les modes mis à part dans le mode “ ajout ”. En effet, il est normal d’écrire de nouvelles données en fin de fichier sauf si on veut modifier partiellement ou complètement le contenu du fichier.

On peut déplacer volontairement le “ curseur ” afin de le lire ou d’écrire à partir d’une position quelconque dans le fichier.

int fseek (FILE *stream , long déplacement , int origine) ;

fseek déplace le curseur associé au flux **stream** d'une valeur **déplacement** par rapport à une position indiquée par **origine**. **fseek** retourne une valeur non nulle en cas d'erreur.

Pour les fichiers ouverts en mode texte, le **déplacement** doit être 0 ou une valeur retournée par **ftell**.

L'**origine** peut prendre une valeur parmi les nombres 0, 1 ou 2, valeurs identifiées par des constantes symboliques définies dans stdio.h :

Origine Constantes symboliques	Origine Valeurs numériques	Position de départ correspondante
SEEK_SET	0	Début du fichier
SEEK_CUR	1	Position courante dans le fichier
SEEK_END	2	Fin du fichier

int ftell (FILE *stream) ;

ftell retourne la position courante du curseur dans le fichier pour le flux **stream**, ou bien **-1L** en cas d'erreur.

void rewind (FILE *stream) ;

rewind repositionne le " curseur " au début du fichier associé au flux **stream**. Cette fonction est équivalente à **fseek (fp , 0L , SEEK_SET)** .

int fgetpos (FILE *stream , fpos_t *ptr) ;

fgetpos mémorise dans **ptr** la position courante du curseur dans le fichier associé au flux **stream**. Le type **fpos_t** convient pour cs type de valeur. **fgetpos** retourne une valeur non nulle en cas d'erreur.

int fsetpos (FILE *stream , fpos_t *ptr) ;

fsetpos positionne à **ptr** la position courante du curseur dans le fichier associé au flux **stream**. **fsetpos** retourne une valeur non nulle en cas d'erreur.

Exemple le programme suivant écrit 3 variables de type structuré dans un fichier, repositionne le curseur sur la 2ème variable puis effectue sa lecture.

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

typedef struct {
        char  nom[20] ;
        char  prenom[20] ;
        int   age ;
        } personne ;

int main()
{
```

```
personne jean = {"Clerc","Jean",25} ;
personne pierre = {"Tombe", "Pierre", 41} ;
personne paul , px ;
FILE *fp ;
strcpy ( paul.nom , "Anski") ;
strcpy ( paul.prenom, "Paul") ;
paul.age = 27 ;
if ( (fp = fopen ("texte4","w+") ) == NULL )
        {printf("Création fichier impossible");
        exit(1) ; }

fwrite ( &jean , sizeof (personne) , 1 , fp ) ;
fwrite ( &pierre , sizeof (personne) , 1 , fp ) ;
fwrite ( &paul , sizeof (personne) , 1 , fp ) ;

fseek ( fp , -(long)2*sizeof(personne) , SEEK_END ) ;

fread ( &px , sizeof (personne) , 1 , fp ) ;
fclose(fp) ;
printf("%s  %s  %d\n" ,px.nom, px.prenom, px.age);
return 0 ;
}
```

L'affichage donne :

```
Tombe  Pierre  41
```

Détection de la fin d'un fichier

int feof (FILE *stream)

feof teste la position du curseur associé au flux **stream**, **feof** retourne la valeur 0 si le curseur ne se trouve pas en fin de fichier, une valeur non nulle si la fin de fichier est détectée.

5. EXERCICES

5.1 Proposer un programme qui compte le nombre de mots et le nombre de caractères d'un fichier puis qui affiche les résultats.

5.2 Proposer un programme qui effectue une copie de sauvegarde d'un fichier.

5.3 Proposer un programme qui effectue une sauvegarde d'une liste de nom tapée au clavier.

Faire l'exercice 6 à la place du 5.4 si le concept de liste chaînée est connu.

5.4 On veut stocker les informations suivantes pour une personne : nom, prénom, âge et son état marital, célibataire ou marié. Puis par la suite pouvoir gérer ces informations pour plusieurs individus.

Proposer une structure pour mémoriser ces informations.

Ecrire un programme qui demande ces informations pour un individu puis qui relit les informations stockées et affiche le résultat à l'écran.
Ecrire un programme qui stocke les informations précédentes dans un fichier. Faire un test pour 3 individus.
Ecrire un programme qui lit le fichier précédent.
Modifier le programme précédent afin de retrouver les informations d'une personne dont le rang sera demandé. On choisit le rang car la recherche portant sur le rang est ici plus facile que si elle portée sur le nom
Modifier le programme précédent afin de supprimer les informations d'une personne dont le rang est demandé puis qui réarrange le fichier.

Modifier le programme précédent afin de proposer un menu qui offre les choix suivants :

a) Saisie des informations pour une personne.
b) Lecture des informations relatives à toutes les personnes.
c) Lecture des informations pour une personne dont le rang sera demandé.
d) Saisie et ajout des informations relatives à une nouvelle personne.
e) Sortie du programme.

L'option c) offrira la possibilité de supprimer l'enregistrement relatif à la personne sélectionnée et, dans ce cas, entraînera le réarrangement du fichier.

6. Mini projet : liste chaînée et fichier

On veut mémoriser les éléments suivants concernant une " personne " : nom, prénom, âge et date de naissance. L'ensemble de ces éléments constitue une " **fiche** " individuelle.

On veut saisir ces informations pour 5 personnes.

La structure **fiche** est constituée de la manière suivante :

Nom : tableau de 20 caractères
Prénom : tableau de 20 caractères.
Age : entier.
Date de naissance : tableau de 20 caractères (par exemple : "20/10/1990").
Suivant : pointeur sur une structure **fiche**.

6.1 Ecrire un programme qui effectue la saisie des informations précédentes pour 5 personnes puis qui affiche les informations saisies. Le pointeur Suivant sera initialisé à NULL pour chaque **fiche**.

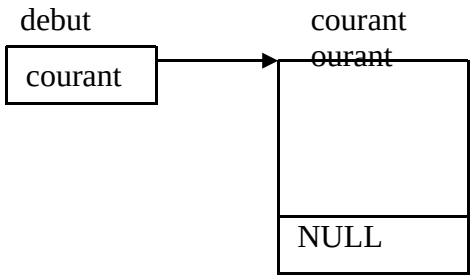
6.2 Modifier le programme précédent afin de stocker les fiches dans une **liste chaînée**. Le programme devra proposer 3 choix : la création d'une nouvelle fiche, la visualisation de toutes les fiches entrées et la fin. On demande d'écrire des fonctions. Tester le fonctionnement.

Rappel sur les listes chaînées

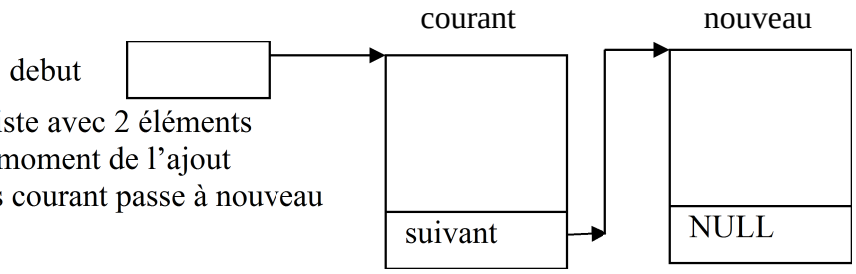
On doit utiliser 3 pointeurs sur une structure de type **fiche** : **debut**, **courant**, **nouveau**.
debut est initialisé à NULL.

1)La liste est vide, le pointeur **début** vaut NULL

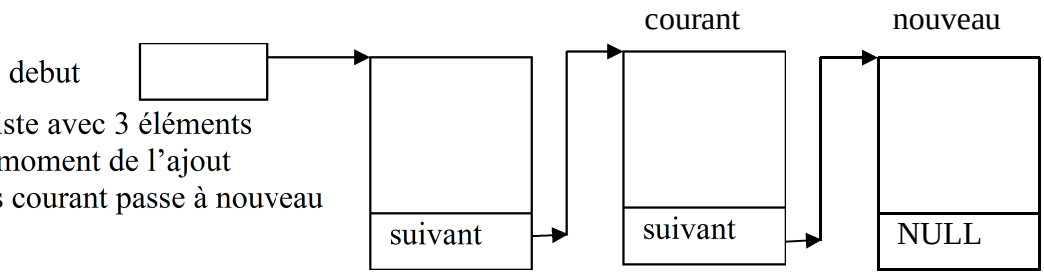
2) Liste contenant un élément



3)Liste avec 2 éléments
Au moment de l'ajout
Puis courant passe à nouveau



4)Liste avec 3 éléments
Au moment de l'ajout
Puis courant passe à nouveau



Proposition d'algorithme pour **ajouter** un nouvel élément

nouveau ← allocation mémoire pour mémoriser une fiche
Remplir les champs de nouveau

Si debut = NULL
Alors courant← nouveau **(dessin 2)**
debut ← courant
courant→suivant ← NULL
Sinon courant→suivant ← nouveau **(dessin 3, 4)**
courant = nouveau
courant→suivant ← NULL
Finsi

6.3 On veut maintenant les possibilités suivantes :

- Création d'une fiche.
- Affichage de la liste.
- Sauvegarde des fiches dans un fichier.
- Sortie du programme.

Il faut tester si un fichier existe lors du lancement du programme. Si c'est le cas, il faut construire la liste chaînée. Attention : les pointeurs sauvegardés dans le fichier ne sont plus valables.

6.4 On veut maintenant les possibilités suivantes :

- Création d'une fiche.
- Affichage de la liste.
- Recherche d'une fiche particulière à partir du nom et affichage.
- Sauvegarde des fiches dans un fichier.
- Sortie du programme.

6.5 Ajouter au menu les 2 choix supplémentaires suivants :

- Insertion d'une fiche en donnant le nom de la fiche après laquelle on veut l'insérer.
- Suppression d'un élément en donnant le nom de la fiche à supprimer.

6.6 On veut maintenant pouvoir afficher la liste dans l'ordre inverse de la saisie. Par exemple, on affiche l'élément précédent en appuyant sur « p » ou l'élément suivant en appuyant sur « s ». Proposer une solution qui utilise 2 pointeurs dans chaque structure : liste doublement chaînée.

7. EXPLORER LES REPERTOIRES

Un nouveau type « flux répertoire » de nom **DIR** a été créé afin de manipuler les répertoires, de manière analogue avec les fichiers qui ont le type **FILE**.

Les fichiers entête suivants sont à inclure :

```
#include <sys/types/h>
#include <dirent.h>
```

Ouvrir un répertoire

DIR *opendir (constant char * name) ;

La fonction **opendir()** ouvre un flux répertoire correspondant au nom **name** et renvoie un pointeur sur ce flux. Le flux est positionné sur la 1ère entrée du répertoire. Retourne NULL si une erreur se produit.

Lire un répertoire

struct dirent *readdir(DIR *dir)

La fonction **readdir()** renvoie un pointeur sur une structure **dirent.**

- Lors de sa 1ère utilisation, la fonction **readdir()** remplit la structure **dirent** avec des informations sur le 1er élément contenu dans le répertoire.
- Lors de sa 2ème utilisation, elle remplit la structure **dirent** avec des informations sur le 2ème élément contenu dans le répertoire.
- Et ainsi de suite, lors de sa nème utilisation, readdir() remplit la structure **dirent** avec des informations sur le nème élément du répertoire.
- La fonction **readdir()** retourne NULL lorsque la fin du répertoire est atteinte.

La structure **dirent** contient 4 champs, les champs **d_ino** pour le N° d'inode du fichier et **d_name** pour le nom de ce fichier sont les plus importants.

```
struct  dirent
{
long   d_ino ;                    /* N° de l'inode (linux)*/
off_t  d_off ;                    /* Offset par-rapport à cette structure */
unsigned  short  d_reclen ;       /* longueur de ce d_name  */
char  d_name[NAME_MAX+1] ;        /* Nom du fichier avec 0 en fin de nom */
} ;
```

Fermer un répertoire

int closedir (DIR *dir) ;

Cette fonction ferme le répertoire associé au flux **dir**.
Renvoie 0 si elle réussit sinon –1.

Autres fonctions

void rewinddir(DIR *dir)
Initialise la position du curseur associé au flux dir au début du repertoire.

off_t telldir(DIR *dir, off_t offset) ;
Retourne la position courante du curseur associé au flux donné **dir**. Le type off_t est une redéfinition du type long.

void seekdir(DIR *dir) ;
Fixe la position du curseur associé au flux **dir** d'où le prochain appel à **readdir** démarrera. **seekdir** doit être employée avec une position **offset** retournée auparavant par **telldir**.

int chdir (const char *dir) ;
int fchdir(int desc) ;
Ces 2 fonctions permettent de changer de répertoire courant, le **nom** du nouveau répertoire est donné directement pour **chdir** alors qu'on donne son **descripteur** pour **fchdir**.

```
int  scandir ( constant char *dir, struct dirent ***liste ,
               int (*select) (const struct dirent * ) ,
               int (*compar)(const struct dirent ** , const struct dirent **)) ;
```
Utiliser man scandir pour obtenir de l'aide sur cette fonction .

8. CREATION ET SUPPRESSION D'UN REPERTOIRE

Les fichiers entête suivants sont à inclure :

```
#include <sys/types/h>
#include <fcntl.h>
#include <unistd.h>
```

Créer un répertoire

int mkdir (const char * nom , mode_t mode) ;

mkdir crée un nouveau répertoire nommé **nom**. Les droits sont calculés à partir de **mode**, ils sont donnés par **mode & ~umask** .
mkdir renvoie 0 si elle réussit ou –1 si elle échoue.
Utiliser la fonction perror pour connaître la nature de l'erreur.

```
#include <sys/types/h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>

int main()
{
if  ( mkdir ("dir" , 0777) == -1 )
        { perror ("erreur");
        exit(1);}
printf("Création effectuée\n");
return 0 ;
}
```

- Une 1ère exécution de ce programme crée le répertoire dir.
- Une 2ème exécution de ce programme échoue, le message **erreur : le fichier existe** est affiché à l'écran.
- Il faut supprimer ce répertoire pour effectuer d'autres essais avec de nouveaux droits.

Supprimer un répertoire

int rmdir (const char * nom) ;

Supprime le répertoire dont le nom est donné ;
La fonction **rmdir** retourne 0 si elle réussit, -1 dans le cas d'un échec.

9. EXERCICES

9.1 Tester l'exemple du cours.

9.2 Ecrire en programme qui analyse un répertoire et affiche les noms des fichiers contenus ainsi que les N° d'inode correspondants (Linux). Le nom du répertoire sera donné comme argument du main, le programme traitera le répertoire courant si aucun argument n'est donné.

9.3 Améliorer le programme précédent afin d'afficher le type de l'élément contenu : est-ce un fichier ? est-ce un répertoire ? L'affichage permet de vérifier le fonctionnement du programme, par exemple :

```
Texte      : fichier
ex.c       : fichier
rep1       : Répertoire
---
```

9.4 Améliorer le programme précédent afin d'analyser toute l'arborescence contenue dans le répertoire choisi. Ce programme fait appel au concept de récursivité.