Plan

Notion de fichier

Un **fichier** est un ensemble d'informations, codées par une suite d'octets, stockées sur un support de mémoire permanent (disques, clés, cédérom...), que l'on peut accéder en **lecture et/ou écriture** via un **nom** spécifiant son chemin d'accès.

Les systèmes de gestion de fichiers organisent l'accès aux fichiers via une **arborescence de répertoires**.

Les systèmes d'exploitation associent aux fichiers un certain nombre de **métadonnées** :

- taille (en nombre d'octets)
- droits d'accès `rwx` en lecture/écriture/exécution
- dates de création/modification
- ...

Des fichiers aux flux

Dans les langages de programmation, on traite de la même manière les opérations de lecture/écriture vers un fichier, ou un écran, un clavier. On abstrait les mécanismes d'accès physique à l'entité par la notion de **flux** qui permet de :

- garantir la portabilité d'un système à l'autre (Linux, Windows, Mac OS X...) en s'abstrayant des types de stockage et codage des différents systèmes ;
- optimiser les temps d'accès physique en gérant des **buffers d'entrée-sortie** évitant un accès octet par octet aux périphériques :
  - des buffers par **bloc de taille fixe** pour les périphériques de stockage
  - des buffers par **ligne** pour les autres périphériques (clavier, écran)

La bibliothèque `stdio.h` fournit un ensemble d'opérations de manipulation de ces flux.

Utilisation des flux

Les flux peuvent être de deux **types** :

- des **flux de textes** qui sont des suites de lignes ; une ligne est une suite de caractères terminée par un caractère fin de ligne `'\n'`
- des **flux binaires** qui sont des suites de séquences de bits de taille définie.

Pour utiliser un **fichier** il faut :

1. l'ouvrir
   - Un flux lui est alors associé et un **repère** mémorise en permanence l'emplacement de la prochaine lecture/écriture
2. faire des opérations de lecture/écriture
   - Le **repère** se décale automatiquement du nombre d'octets lus ou écrits (idem la tête de lecture/écriture a avancé).
3. le fermer

Ouverture de fichier

Définition

```
FILE *fopen(char *nom, char* mode);
```

- `nom` est est la référence (absolue ou relative) au fichier
- `mode` est une chaîne de caractères spécifiant le **mode d'ouverture** composée :
  - d'une spécification des opérations permises {`"r"`, `"w"`, `"a"`, `"r+"`, `"w+"`, `"a+"`}
  - qui peut être suivie d'un `"b"` spécifiant que l'on veut un flux binaire (non utile sur les systèmes classiques)
- `fopen` renvoie un pointeur de flux vers le fichier ou NULL si l'ouverture n'a pas été possible.
  ↪ Il faut donc toujours tester sa valeur de retour.

Les modes d'ouverture

"r" **lecture** : le fichier doit exister, le repère est positionné au début, on peut le lire. On ne peut pas l'écrire.

"w" **écriture** : le fichier est créé s'il n'existe pas, s'il existe son contenu est effacé, le repère est positionné au début, on peut écrire et écraser son contenu. On ne peut pas le lire.

"a" **ajout** : le fichier est créé s'il n'existe pas, le repère est positionné à la fin, toute opération d'écriture est toujours faite en fin. On ne peut ni le lire, ni écraser son contenu.

"r+" **lecture/écriture** : le fichier est créé s'il n'existe pas, le repère est positionné au début, on peut le lire, l'écrire et écraser son contenu.

"w+" **lecture/écriture** : le fichier est crée s'il n'existe pas, s'il existe son contenu est effacé, le repère est positionné au début, on peut le lire, l'écrire et écraser son contenu.

"a+" **lecture/ajout** : le fichier est créé s'il n'existe pas, le repère est positionné au début, on peut le lire et l'écrire à la fin. On ne peut pas écraser son contenu.

Les flux par défaut

3 flux sont automatiquement ouverts et fermés au début et à la fin d'exécution d'un programme :

- `stdin` est le flux d'entrée standard ouvert en lecture qui par défaut correspond au **clavier**. C'est sur ce flux que `scanf` lit ses données.
- `stdout` est le flux de sortie standard ouvert en ajout qui par défaut correspond à **l'écran**. C'est sur ce flux que `printf` écrit ses données.
- `stderr` est le flux d'erreur standard ouvert en ajout qui par défaut correspond à **l'écran**. Ce dernier est prévu pour l'affichage des messages d'erreur.

Fermeture de flux

Définition

```
int fclose(FILE *pf);
```

- `pf` est le pointeur de flux à fermer
- `fclose` renvoie `0` si la fermeture s'est bien passé. La constante `EOF` (end of file) est renvoyée sinon.

`EOF` est une constante de la librairie retournée lors d'une opération de lecture pour indiquer qu'il n'y a plus rien à lire.

- Le repère pointe sur la fin du flux : la fin du fichier est atteinte ou rien n'est dans le buffer.

Exemple : test d'existence d'un fichier

Le mode `"r"` est le seul mode qui ne crée pas un fichier s'il n'existe pas.

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

int main (int argc, char **argv) {
    if(argc != 2) {
        printf("Usage: %s nom_fichier\n",argv[0]);
        return EXIT_FAILURE;
    }
    FILE *pf = fopen(argv[1], "r");
    if(pf == NULL) {
        printf("Le fichier %s n'a pas pu etre ouvert\n",argv[1]);
        return EXIT_FAILURE;
    }
    printf("Le fichier %s existe\n",argv[1]);
    if(fclose(pf) == EOF) {
        printf("Erreur: fermeture du fichier impossible\n");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

Attention

Ne pas oublier de **fermer les fichiers** avant de quitter un programme.

Lecture/écriture de caractères dans un flux texte

Définition

```
int fgetc(FILE *pf);        int fputc(int c, FILE *pf);
```

- `pf` est le pointeur d'un flux.
- `c` contient le caractère à écrire (converti en `char` à l'écriture)
- `fgetc/fputc` retourne le caractère lu/écrit ou `EOF` en cas d'erreur

Exemple

```
...
int i = fgetc(stdin);
if(i==EOF || fputc(i,fp)==EOF) {
    fclose(fp);
    exit(EXIT_FAILURE);
}
...
```

Annulation de lecture

Définition

```
int ungetc(int c, FILE *pf);
```

- (re)place le caractère `c` dans le flux d'entrée `pf` qui devient donc le prochain caractère à lire (limité à la taille de son buffer associé).
- l'idée est de permettre de tester le prochain caractère à lire.

Exemple

```
void sauterEspace() {
    int c;
    do {
        c = fgetc(stdin);
    } while(c!=EOF && c==' ');
    if(c!=EOF) ungetc(c,stdin);
}
```

Lecture/écriture de `string` dans un flux texte

Définition

```
char *fgets(char *s, int long, FILE *pf);
```

- `s` est un tableau d'au moins `long+1` caractères.
- `fgets` tente de lire au max `long` caractères, les stocke dans `s`, ajoute `'\0'` et retourne `s` ou `NULL` en cas d'erreur

Définition

```
int fputs(char *ligne, FILE *pf);
```

- `ligne` contient la `string` à écrire (max 254 caractères)
- `fputs` écrit la string et retourne un entier positif ou `EOF` en cas d'erreur

Lecture/écriture formatées dans un flux texte

Définition

```
int fscanf(FILE *pf,"chaîne de contrôle",adr_1,...,adr_n);

int fprintf(FILE *pf,"chaîne de contrôle",exp_1,...,exp_n)
```

- Ces fonctions fonctionnent comme `scanf` et `printf` mais précisent les flux concernés.
- `fscanf` retourne le nombre de données saisies (entre `0` et `n`) ou `EOF` s'il n'y a plus rien à lire.
- `fprintf` retourne le nombre de caractères écrits ou une valeur négative en cas d'erreur.

Remarque

Pour le flux `stdin`, les lectures peuvent se faire directement via les fonctions `getc, gets, scanf`.
Pour le flux `stdout`, les écritures peuvent se faire directement via les fonctions `putc, puts, printf`.

Exemple : affichage d'un fichier d'entiers (au format texte)

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

int main (int argc, char **argv) {
    if(argc != 2) {fprintf(stderr,"Erreur parametres\n"); return EXIT_FAILURE;}
    FILE *pf = fopen(argv[1], "r");
    if(pf == NULL) {
        fprintf(stderr, "Le fichier %s n'a pas pu etre ouvert\n",argv[1]);
        return EXIT_FAILURE;
    }
    int i, res;
    do{ res = fscanf(pf,"%d",&i);
        if(res == 1) {
            if(printf("%d\n",i)<0) {
                fprintf(stderr, "Erreur ecriture\n");
                fclose(pf);
                return EXIT_FAILURE;
            }
        } else if(res==0) {
            fprintf(stderr, "Erreur de format\n");
            fclose(pf);
            return EXIT_FAILURE;
        }
    }while(res != EOF);
    if(fclose(pf)==EOF) {fprintf(stderr,"Pb fermeture\n"); return EXIT_FAILURE;}
    return EXIT_SUCCESS;
}
```

Lecture/écriture binaire

Lecture/écriture de blocs d'octets (l'unité n'est plus le caractère) :

Définition

```
size_t fread(void *buf, size_t size, size_t nb, FILE *pf)
size_t fwrite(void *buf, size_t size, size_t nb, FILE *pf)
```

- `buf` est un tableau d'au moins `nb*size` octets ;
- `fread/fwrite` essaie de lire/écrire `nb` blocs de `size` octets dans le flux `pf` et retourne le nombre de blocs réellement lus/écrits (entre 0 et `nb`). Quand le nombre retourné est différent de `nb` :
  - avec `fwrite` c'est qu'une erreur s'est produit
  - avec `fread` c'est une erreur ou une fin de fichier atteinte
    ↪ `feof` permet de savoir si c'est ce dernier cas.
  - Quand une erreur se produit, le repère n'a pas forcément été a déplacé d'un multiple de `size`.

Connaître la position du repère dans le fichier

2 fonctions permettent de connaître **l'emplacement du repère** de lecture/écriture :

Définition

```
int feof(FILE *pf)
```

- retourne vrai si le repère est en fin de fichier et faux (0) sinon.

Définition

```
long ftell(FILE *pf)
```

- retourne la position du repère en nombre d'octets depuis le début du fichier ou `-1` en cas d'erreur.

Modifier la position du repère

- Lors de l'ouverture du flux, le repère est **initialisé** en début ou fin de fichier en fonction du mode d'ouverture choisi
- Toutes les fonctions de lecture/écriture font **automatiquement avancer** le repère du nombre d'octets lus
- 1 fonction permet de **modifier sa position** sans lecture/écriture :

Définition

```
int fseek(FILE *pf, long offset, int whence)
```

- `offset` est un nombre positif ou négatif d'octets
- `whence` est une position du fichier :
  - SEEK_CUR désigne la position actuelle du repère : =`ftell`
  - SEEK_SET désigne le début du fichier : 0
  - SEEK_END désigne la fin du fichier : la taille du fichier
- `fseek` positionne le repère à `whence+offset` et retourne `0` si le positionnement s'est bien passé et `-1` en cas d'erreur.

Synchroniser/vider le buffer

2 fonctions permettent d'agir sur le buffer associé à un flux :

Définition

```
int fflush(FILE *pf)
```

- pour les flux de sortie force l'écriture des données du buffer. Utile pour les flux ouvert en lecture/écriture pour s'assurer qu'une opération a bien été prise en compte avant d'en faire une autre.

Définition

```
int fpurge(FILE *pf)
```

- vide le buffer associé au flux (sans affecter le flux). Utile pour la saisie de valeur par `scanf` !
- **Remarque** : cette dernière fonction proposée dans de nombreuses implémentations du `C` n'est pas dans la norme !

Exemple 1 : copie de fichier

Un fichier peut contenir n'importe quelle donnée
↪ utiliser les **flux binaires**.

Schéma de copie

1. Vérification des paramètres
2. Ouverture du fichier source en lecture
3. Test de non existence du fichier destination
4. Ouverture du fichier destination en écriture
5. Création d'un buffer en mémoire pour copie par bloc
6. Boucle
   - lecture d'un bloc
   - écriture du bloc
7. Fermeture des flux

Exemple 1 : copie de fichier

```
#include <stdio.h>
#include <stdlib.h>
#define BUFSIZE 256

int main(int argc, char* argv[]) {

    // Verification des parametres
    if(argc != 3) {
        fprintf(stderr,"Usage: %s source destination\n",argv[0]);
        return EXIT_FAILURE;
        }

    // Ouverture du fichier source
    FILE *pfs = fopen(argv[1], "r");
    if (pfs == NULL) {
        fprintf(stderr, "%s n'existe pas\n",argv[1]);
        return EXIT_FAILURE;
    }
...
```

Exemple 1 : copie de fichier

```
...
    // Test de non existence du fichier destination
    FILE *pfd = fopen(argv[2], "r");
    if (pfd != NULL) {
        fclose(pfd);
        printf("Voulez-vous ecraser %s ? (O/N) ",argv[2]);
        fpurge(stdin);
        char rep = 'N';
        scanf("%c",&rep);
        if(rep!='O') {
            fclose(pfs);
            return EXIT_FAILURE;
        }
    }

    // Ouverture du fichier destination
    pfd = fopen(argv[2], "w");
    if (pfd == NULL) {
        fprintf(stderr, "impossible de creer %s\n",argv[2]);
        fclose(pfs);
        return EXIT_FAILURE;
    }
...
```

Exemple 1 : copie de fichier

```
...
   // Creation d'un buffer en memoire pour copie par bloc
    void *buf = malloc(BUFSIZE);
    if(buf==NULL) {
        fprintf(stderr,"impossible de copier\n");
        fclose(pfs);
        fclose(pfd);
        return EXIT_FAILURE;
    }

    // Boucle de lecture/ecriture
    int nblus;
    while(!feof(pfs)) {
        nblus = fread(buf, 1, BUFSIZE, pfs);
        if(nblus==0 || fwrite(buf,1, nblus, pfd) != nblus) {
            fprintf(stderr,"copie incomplete\n");
            free(buf);
            fclose(pfs);
            fclose(pfd);
            return EXIT_FAILURE;
        }
    }
 ...
```

Exemple 1 : copie de fichier

```
...
    // Fermeture des flux
    free(buf);
    fclose(pfs);
    if(fclose(pfd)==EOF) {
        fprintf(stderr, "copie incomplete\n");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

Exemple 2 : lecture dans un fichier binaire structuré

Hypothèse :

- le fichier est ouvert en lecture
- le fichier est composé d'enregistrements de taille fixe (en binaire)
- un index donne le numéro de l'enregistrement à lire

```
struct enreg {
    ...
};

_Bool lecture(FILE *pf, int numEnreg, struct enreg *e) {
    // Positionnement du repere au debut de l'enregistrement
    if(fseek(pf, sizeof(struct enreg)*(numEnreg-1), SEEK_SET) == -1) {
        fprintf(stderr,"erreur de positionnement\n");
        return 0;
    }
    // Lecture de l'enregistrement
    if(fread(e, sizeof(struct enreg), 1, pf) != 1) {
        fprintf(stderr,"erreur de lecture\n");
        return 0;
    }
    return 1;
}
```

Exemple 2 : écriture dans un fichier structuré

le fichier est ouvert en écriture

```
_Bool ecriture(FILE *pf, int numEnreg, struct enreg e) {
    // Positionnement du repere au debut de l'enregistrement
    if(fseek(pf, sizeof(struct enreg)*(numEnreg-1), SEEK_SET) == -1) {
        fprintf(stderr,"erreur de positionnement\n");
        return 0;
    }
    // Ecriture de l'enregistrement
    if(fwrite(&e, sizeof(struct enreg), 1, pf) != 1) {
        fprintf(stderr,"erreur d'ecriture\n");
        return 0;
    }
    return 1;
}
```

Attention

Selon `num` et soit `nbEnreg` le nombre d'enregistrements du fichier on peut :

- $0 \leq$ `num` $\leq$ `nbEnreg` : modifier un enregistrement existant
- `num` $=$ `nbEnreg`$+1$: ajouter un nouvel enregistrement
- `num` $>$ `nbEnreg`$+1$: ajouter d'un coup `num` - `nbEnreg` dont tous sauf le dernier auront des valeurs indéterminées.

Plan