Aide : Manipulation de Fichiers en Langage C et Python

1 Langage C :

1.1 Ouverture

Les flots sont désignés par des variables de type `FILE *`, pointeur sur une structure de type `FILE`.
Ce type est défini dans le fichier d'interface `stdio.h`.
La définition de ce type est dépendante du système d'exploitation, mais correspond toujours à une structure (ou descripteur) qui contient les informations suivantes :

- l'adresse de la mémoire tampon,
- la position de la tête de lecture/écriture
- le type d'accès au fichier (lecture, écriture),
- l'état d'erreur,
- ...

Pour utiliser un fichier dans un programme C, il faut donc inclure le fichier `stdio.h` et définir une variable de type `FILE *` :

```
#include<stdio.h>;
FILE *fic1;
```

Pour utiliser un fichier, il faut l'ouvrir avec la commande `fopen` dont voici le prototype :

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

L'ouverture d'un fichier initialise le descripteur décrit plus haut.
Nous fournissons à la fonction `fopen` le nom du fichier (premier paramètre, c'est une chaîne de caractères) et le mode d'accès sur ce fichier.

Les opérations les plus utilisées sont listées ci-dessous. La fonction retourne un pointeur vers une structure de type `FILE` qui sera utilisée par la suite pour les opérations d'entrées-sorties.

C'est ce pointeur qui identifie le fichier que nous utilisons.

"r"	**ouvrir le fichier en lecture seule** erreur si le fichier n'existe pas
"w"	**ouvrir le fichier vide en écriture seule** création du fichier s'il n'existe pas,sinon, son contenu est écrasé
"a"	**ouvrir le fichier en ajout**(écriture en fin de fichier) création du fichier s'il n'existe pas
"r+"	**ouvrir le fichier en lecture et en écriture** erreur si le fichier n'existe pas
"w+"	**ouvrir le fichier vide en lecture et en écriture** création du fichier s'il n'existe pas, sinon, son contenu est écrasé
"a+"	**ouvrir le fichier en lecture et en ajout** (écriture en fin de fichier) création du fichier s'il n'existe pas

Si l'ouverture s'est avérée impossible, `fopen` rend la valeur `NULL` comme dans l'exemple :

1.2 Fermeture

Pour fermer un fichier il est nécessaire d'utiliser la fonction `fclose`.
Même si le système se charge en principe de fermer tous les fichiers ouverts, à la fin de l'exécution d'un programme, il est vivement conseillé de fermer les fichiers explicitement à la fin de leur utilisation.

Cela permet de réduire le nombre de descripteurs ouverts et de se prémunir contre toute interruption brutale du programme.

Le prototype de la fonction `fclose` est :

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

Le paramètre est un pointeur vers une structure de type `FILE` (celui qui vous a été retourné par la fonction `fopen`).

La fonction `fclose` retourne la valeur EOF si la fermeture s'est mal passée, 0 sinon.

1.3 Quelques Fonctions

Vider le tampon : la fonction `fflush` permet de vider immédiatement le contenu du tampon, elle retourne EOF en cas d'erreur, 0 dans les autres cas :

```
int fflush(FILE *flot);
```

Tester la fin de fichier : la fonction `feof` teste l'indicateur de fin de fichier du flot spécifié et retourne une valeur non nulle si la fin de fichier est atteinte :

```
int feof(FILE *flot);
```

1.4 Lecture et Écriture en Mode Caractères

Écriture : la fonction `fputc` transfère le caractère spécifié dans le fichier représenté par le flot passé en paramètre.

La valeur de retour est le caractère ou la valeur (-1) qui correspond au retour de la macro EOF en cas d'erreur :

```
int fputc(int car, FILE *flot);
```

Lecture : la fonction `fgetc` retourne le caractère lu dans le fichier représenté par le paramètre flot. En cas d'erreur ou de fin de fichier, elle retourne la valeur EOF :

```
int fgetc(FILE *flot);
```

Relecture d'un caractère : Il est possible de replacer un caractère dans un flot au moyen de la fonction `ungetc`.

```
int ungetc(int caractere, FILE *flot);
```

Cette fonction place le caractère `caractere` (converti en unsigned char) dans le flot `flot`.

En particulier, si caractère est égal au dernier caractère lu dans le flot, elle annule le déplacement provoqué par la lecture précédente.

Toutefois, `ungetc` peut être utilisée avec n'importe quel caractère (sauf le retour de la macro EOF).

1.5 Lecture et Écriture en mode Chaînes

Écriture : la fonction (`fputs`) écrit une chaîne de caractères dans le flot spécifié (elle n'écrit pas le caractère '\ 0'), elle retourne une valeur non négative si aucune erreur ne s'est produite, EOF sinon :

```
int fputs(const char *s, FILE *flot);
```

Lecture : la fonction `fgets` lit une chaîne de caractères dans le flot spécifié et la range dans un tampon défini (et alloué) :

```
char* fgets(char *s, int taille, FILE *flot);
```

Plus précisément, la fonction lit taille-1 caractères dans le fichier, ou lit jusqu'au caractère '\ n' (une ligne tout au plus), et :

- `s` : est de type pointeur vers char et pointe vers un tableau de caractères (alloué).
- `taille` : est la taille en octets du tableau de caractères pointé par s.
- `flot` : est de type pointeur sur FILE et pointe vers le fichier à partir duquel se fait la lecture.
- la valeur rendue est le pointeur `s` en cas de lecture sans erreur, ou `NULL` dans le cas de fin de fichier ou d'erreur.

1.6 Lecture et Écriture en mode Bloc de Caractères

Écriture : la fonction `fwrite` permet d'écrire un ensemble de blocs de caractères dans le flot spécifié.
Elle retourne une valeur non nulle du nombre de blocs qui ont été écrits dans le flot.
Si cette valeur est inférieure aux nombre de blocs à écrire ou si elle est nulle, l'écriture ne s'est pas déroulée correctement :

```
size_t fwrite (const void *buffer, size_t size, size_t count, FILE *stream);
```

`*buffer` : est le pointeur du bloc de données qui va être copié dans le flot.
`size_t size` : est la taille du bloc.
`size_t count` : nombre de blocs à écrire.
`FILE *stream` : le flot vers lequel l'écriture doit se faire.

Lecture : la fonction `fread` permet de lire un ensemble de blocs de caractères depuis le flot spécifié, elle retourne une valeur non nulle du nombre de blocs qui ont été lus depuis le flot.

Si cette valeur est inférieure aux nombre de blocs à lire ou si elle est nulle, la lecture ne s'est pas déroulée correctement :

```
size_t fread (const void *buffer, size_t size, size_t count, FILE *stream);
```

`*buffer` : est le pointeur vers lequel le bloc de données va être écrit.
`size_t size` : est la taille de chaque bloc.
`size_t count` : nombre de blocs à lire.
`FILE *stream` : le flot à partir duquel la lecture doit se faire.

1.7 Fonctions de Déplacement dans un Flot

Se positionner en début de fichier :

La fonction `rewind` permet de positionner le pointeur de flot en début du fichier qui va être lu ou dans lequel vous souhaitez écrire.

Lorsque vous ouvrez un flot avec les directives "r", "r+", "w", "w+", le pointeur du flot se situe en début de fichier, par conséquent ce repositionnement interviendra suite à une ouverture avec les directives "a", "a+", ou si vous souhaitez revenir en début de fichier pour une opération de recherche par exemple.

```
void rewind( FILE *stream);
```

`FILE *stream` : le flot sur lequel le pointeur doit être positionné en début de fichier.

Position du pointeur depuis le début du fichier :

La fonction `ftell` renvoie la position du pointeur de fichier depuis son début (comme le décalage depuis le début du fichier).
Par conséquent la valeur renvoyée n'est pas une adresse mais un entier long correspondant à l'offset depuis le début du fichier.
Si la fonction a trouvé la position du pointeur celle-ci renvoie une valeur positive ou nulle (sinon elle renvoie -1).

```
long ftell(FILE *stream);
```

`FILE *stream` : le flot sur lequel le pointeur doit être positionné en début de fichier.

Se positionner sur une position du fichier :

La fonction `fseek` permet de positionner le pointeur de flot sur une position particulière du fichier. Si le positionnement s'est correctement effectué, la fonction renvoie la valeur 0.

```
int fseek(FILE *stream, long offset, int origin);
```

`FILE *stream` : le flot sur lequel le pointeur doit être positionné en début de fichier.
`long offset` : le nombre de bytes (ou octets) de décalage depuis l'origine.
`int origin` : Le point de départ à partir du quel va être réalisé le décalage. De manière standard, trois positions d'origine sont possibles : `SEEK_CUR` (position courante du pointeur sur le fichier), `SEEK_END` (origine en fin du fichier), et `SEEK_SET` (origine en début du fichier).

Repérer la position du pointeur sur le fichier

La fonction `fgetpos` permet de sauvegarder la position courante du pointeur sur le fichier. Cette position pourra être utilisée ultérieurement pour y revenir avec la fonction `fsetpos`. Si la capture de la position du pointeur dans le fichier s'est correctement déroulée, la fonction renvoie la valeur 0.

```
int fgetpos(FILE *stream, fpos_t *pos);
```

`FILE *stream` : le flot sur lequel l'opération va être effectuée.
`fpos_t *pos` : le pointeur dans lequel sera sauvegardé la position courante du pointeur sur le fichier.

Se déplacer vers une position particulière du fichier :

La fonction `fsetpos` permet de se déplacer directement vers une position du fichier.
Cette fonction doit être utilisée après une fonction `fgetpos`.
Si le déplacement vers la position dans le fichier s'est correctement déroulé, la fonction renvoie la valeur 0.

```
int fsetpos(FILE *stream, fpos_t *pos);
```

`FILE *stream` : le flot sur lequel l'opération va être effectuée.
`fpos_t *pos` : le pointeur de la position vers laquelle doit être positionné le pointeur sur le fichier.

1.8 Lecture et Écriture Formatées

Les fonctions `fprintf` et `fscanf` permettent de faire des lectures et écritures formatées.
Les fonctions `printf` et `scanf` sont des raccourcis pour les flots d'entrée/sortie standards.

1.8.1 Écriture Formatée avec `fprintf`

La fonction `fprintf` admet un nombre variable de paramètres :

```
int fprintf(flot,format,param1,param2, ...,paramN);
```

avec,

- `flot` : type pointeur sur `FILE`, le fichier sur lequel nous écrivons,
- `format` : chaîne de caractères qui spécifie ce qui doit être écrit,
- `param1` : expression dont nous voulons écrire la valeur,
- ...
- `paramN` : expression dont nous voulons écrire la valeur.

La valeur retournée est égale au nombre de caractères écrits, ou à une valeur négative si il y a eu une erreur d'entrée-sortie.

1.8.2 Lecture Formatée

La fonction `fscanf` admet un nombre variable de paramètres :

```
int fscanf (flot,format,param1,param2, ... ,paramN)
```

avec :

- `flot` : de type pointeur sur `FILE`, le fichier sur lequel nous lisons,
- `format` : chaîne de caractères qui spécifie la forme de ce qui doit être lu,
- `param1 ...paramN` : pointeurs sur des variables dans lesquelles `fscanf` range les valeurs lues dans le flot, après les avoir converties en binaire,

La valeur retournée est égale au nombre de paramètres affectés, si aucun paramètre n'a été affecté (cas de fin de fichier ou d'erreur avant toute affectation), la valeur retournée est EOF (-1).

2 Langage Python :

La fonction `open()` renvoie un objet fichier et est le plus souvent utilisée avec deux arguments : `open(nomfichier, mode)`.

Par exemple, `fic=open('fichier', 'w')`

Il est recommandé d'utiliser le mot-clé `with` avec les fichiers.

Par exemple, `with open('fichier') as fic:`

Si vous n'utilisez pas l'instruction `with`, il faut écrire `fic.close()` pour fermer le fichier.

Pour lire le contenu d'un fichier, nous pouvons utiliser `fic.read(taille)`.
Cette instruction lit une certaine quantité de données et les donne sous la forme d'une chaîne (en mode texte). `taille` est alors un argument optionnel.

L'instruction `f.readline()` lit une seule ligne du fichier ; un caractère de fin de ligne (\n) est laissé à la fin de la chaîne.

Il est possible de construire une liste avec toutes les lignes d'un fichier, en utilisant l'instruction `fic.readlines()`.

`fic.write(chaine)` permet d'écrire le contenu de `chaine` dans le fichier et renvoie le nombre de caractères écrits.

`fic.tell()` est une instruction renvoyant un entier indiquant la position actuelle dans le fichier, mesurée en octets à partir du début du fichier lorsque le fichier est ouvert en mode binaire, ou une valeur numérique en mode texte.

Pour se positionner dans un fichier, il faut écrire l'instruction `fic.seek(decalage,debut)`.
La nouvelle position est obtenue en ajoutant le contenu de `decalage` à un point de référence ; ce dernier est déterminé par l'argument `debut` : 0 pour le début du fichier, 1 pour la position actuelle et 2 pour la fin du fichier.