TP9 : Fichiers

Programmation en C (LC4)

Semaine du 26 mars 2007

1 Fichiers : les bases

Afin de pouvoir stocker des données, ou d'exploiter des données déjà existantes, il est indispensable de pouvoir manipuler des fichiers. C'est ce que nous allons voir dans ce TP.

1.1 La syntaxe : ouverture et fermeture

Pour toutes les opérations sur les fichiers, il faut inclure le header `stdio.h`.

Un fichier s'identifie par son nom. À partir du nom d'un fichier, on peut ouvrir un « canal » permettant de lire ou écrire dans ce fichier. Le type de ces canaux est nommé `FILE`. Il s'agit d'un type « abstrait », dont la définition dépend fortement du système. On ne doit donc pas chercher à regarder à l'intérieur, mais simplement appeler dessus les fonctions standards décrites ci-dessous. Ces fonctions travaillent en fait toutes avec des `FILE *`.

Pour ouvrir un canal vers un fichier, on utilise la fonction `FILE * fopen(char * nom_de_fichier, char * mode)`. Le `mode` sert à spécifier quelle sorte de canal l'on veut. On se contentera ici d'"utiliser le mode « écriture », dénommé par la chaîne `"w"` (write) et le mode « lecture », dénommé par la chaîne `"r"` (read). Ainsi les lignes :

```
FILE * f1;
f1 = fopen("blop","r");
```

ouvrent le fichier `blop` en lecture. Le fichier doit exister si on cherche à le lire, par contre, si on l'ouvre en mode écriture et qu'il n'existe pas, il sera créé automatiquement. S'il existe déjà, il sera tronqué. Si l'ouverture échoue, `fopen` renvoie `NULL`. Si elle réussie, `fopen` renvoie un pointeur vers un canal, et on peut par la suite passer ce pointeur en argument aux diverses fonctions travaillant sur des `FILE *`.

Pour fermer un canal après l'avoir utilisé, on utilise `int fclose(FILE * fichier)`. Il ne faut pas faire de `free`, car `fclose` fait déjà ce qu'il faut. Il est indispensable de fermer un canal après utilisation, pour au moins deux raisons :
- Les écritures dans un canal ne sont pas immédiates, elles sont regroupées en gros blocs, pour des raisons d'efficacité, donc il reste souvent des données en attente d'écriture, et la fermeture du canal déclenche l'écriture de ces données
- Il y a souvent des limites au nombre de canaux qu'un programme peut avoir ouvert simultanément.

Si la fermeture réussit, `fclose` renvoie 0, autrement c'est la valeur `EOF` qui est renvoyée (il s'agit d'une macro définie par `stdio.h`, qui a de multiples usages).

On se retrouve avec le code suivant :

```
// début de la manipulation du fichier
FILE * f1;
if ((f1 = fopen("blop","r"))==NULL){
  printf("erreur lors de l'ouverture de blop \n");
  exit(1);
}

// utilisation du fichier...

// fin de la manipulation du fichier
if (fclose(f1)==EOF){
  printf("erreur lors de la fermeture de blop \n");
  exit(1);
}
```

1.2 Lecture par caractère

La fonction pour lire un caractère depuis un canal `fich` est `int fgetc(FILE * fich)`. Elle renvoie un entier, qui est soit un caractère, soit la valeur `EOF` si on se trouve à la fin du fichier. Lorsqu'on ouvre un fichier, le programme a un «curseur» situé au début du fichier. Les premiers caractères lus seront les premiers caractères du fichiers. À chaque caractère lu, le curseur se décale d'un caractère vers la fin du fichier.

Exercice 1 Créez avec votre éditeur (`emacs` bien entendu ...) un fichier `exo1.txt` où figurera uniquement votre prénom puis votre nom séparés d'un espace.

Écrivez une fonction `void prenom(FILE * f)` qui lit le prenom sur le canal `f` censé pointer vers `exo1.txt`, et l'affiche à l'écran. Lancez-la deux fois de suite dans le `main`. Que se passe-t'il ?

Exercice 2 Écrivez une fonction `compte_c(FILE * f)` qui renvoie le nombre de caractères d'un fichier.

Écrivez une fonction `compte_m(FILE * F)` qui renvoie le nombre de mots d'un fichier. Deux mots sont séparés par un espace ou un retour à la ligne.

Écrivez une fonction `compte_l` qui renvoie le nombre de lignes.

Exercice 3 Faites un programme qui prend en argument sur la ligne de commande le nom d'un fichier, et une lettre (`c`,`m` ou `l`) et qui renvoie selon cette lettre soit le nombre de caractères, de mots ou de lignes du fichier.

1.3 Écriture par caractère

La fonction `int fputc(char m, FILE * f)` écrit dans le canal `f` le caractère `m` (et renvoit le caractère `m` ou `EOF` si l'écriture a échoué).

Exercice 4 Écrivez une fonction qui recopie le contenu d'un fichier dans un autre. Appelez cette fonction dans une fonction `main()` placé dans un fichier `exo4.c` afin de recopier le code de `exo4.c` dans `copie_exo4.c`.

1.4 Lecture et écriture

Plusieurs fonctions permettent de lire et écrire dans des fichiers. Il existe entre autres `fread` et `fwrite` qui s'emploient comme suit :

fread : `size_t fread(void * tableau, size_t taille, size_t n, FILE *f)`. Cette fonction tente de lire, dans le fichier `f`, `n` blocs de `taille` octets, et les écrit à l'adresse `tableau`, qui doit donc pointer vers un tableau suffisamment grand que l'on aura créé au préalable. Elle renvoie le nombre de blocs lus, qui peut être plus petit que `n` (par exemple si l'on a atteint la fin du fichier).

fwrite : `size_t fwrite(void *tableau, int taille, int n, FILE *f)` agit de manière similaire : elle écrit, dans le fichier `f` un nombre `n` de blocs de taille `taille` octets qu'elle lit dans le tableau pointé par `tableau`. Elle renvoie le nombre de blocs écrits en résultat. S'il est inférieur à `n`, c'est qu'il y a eu une erreur.

Exercice 5 À l'aide de ces deux fonction écrire les fonctions `mon_getc` et `mon_putc` qui agissent comme les fonctions vues précédemment.

Exercice 6 Écrivez un programme qui crée un formulaire : il prend en argument sur la ligne de commande un nom de fichier et des mots et crée un fichier avec un mot par ligne.

2 Entrée standard, sortie standard

Quand un programme est lancé, il dispose de trois canaux d'entrée sortie nommés `stdin`, `stdout`, et `stderr`. Ils peuvent être branchés sur des fichiers, mais c'est rarement le cas. En temps normal, le canal `stdin` permet au programme de lire ce que l'utilisateur tape sur le clavier, tandis que `stdout` et `stderr` sont affichés à l'écran (la différence entre ces deux canaux est par convention la suivante : on est censé utiliser `stdout` pour des messages normaux, et `stderr` pour des messages d'erreur, pour permettre à l'utilisateur de trier entre les deux).

Exercice 7 À l'aide des fonctions que vous avez vues, implémenter une fonction `affiche_ecran(char *s)` qui affiche le contenu de s à l'écran.

Exercice 8 Implémenter un programme qui demande à l'utilisateur de remplir un formulaire (par exemple nom, prénom, age, filière, adresse) et qui crée un fichier approprié avec toutes les données nécessaires.