Les Fichiers

Qu'est-ce qu'un fichier ?

Fichiers texte / fichier binaires

manipulation d'un fichier texte

ouverture , lecture-écriture, fermeture

manipulation d'un fichier binaire

structure du fichier

ouverture, lecture-écriture-déplacement, fermeture

Qu'est-ce qu'un fichier ?

Données organisées ou non stockées sur une mémoire de masse, support de sauvegarde (disquette, disque dur, CD/DVD, bande DAT)

peut contenir du texte, de la vidéo, des données pour des applications.

Mémoire de masse persistante (pas effacée quand hors-tension), mais très très lente d'accès.

Les programmes n'utilisent pas les mémoires de masse pour stocker les variables, mais la RAM, plus rapide.

Faire des transferts entre valeurs en RAM et fichiers pour lire/sauvegarder des données.

Manipulation de fichiers

Opérations standard à mener pour travailler avec un fichier :

ouvrir un fichier : lui associer une variable que l'on appelle descripteur de fichier : permettra l'accès au fichier.

Lire ou écrire des informations à partir de ce descripteur avec des fonctions spécialement prévues pour les fichiers.

Fermer le fichier : indiquer que l'on a terminé de travailler avec ce fichier : important dans un environnement multi-tâches et/ou multi-utilisateur.

Deux types de fichier : les fichiers 'texte', ou fichiers à accès séquentiel : les données n'ont aucune structure particulière, on doit lire caractère par caractère ou ligne par ligne.

Manipulation de fichiers

Caractère séquentiel : obligé de parcourir le fichier depuis le début pour accéder à une donnée particulière.

Attention, ne contient pas forcément que du texte ! Le terme 'texte' est employé parce que le fichier est vu comme une suite de caractères (pas forcément lisibles), les caractères sont des entiers.

Les fichiers binaires ou fichiers à accès direct (aléatoire) : fichiers organisés par une structure sous-jacente : il est constitué non pas de caractères mais d'enregistrements (au même titre que les struct).

Connaissant la taille d'un enregistrement, on peut se rendre directement à un endroit précis du fichier : d'où le caractère direct.

Manipulations différentes de celles du fichier texte.

Descripteurs de fichier

Canal de communication entre le programme et un fichier situé quelque part sur le disque.

Type de variable spécifique : le type FILE. On utilise toujours un pointeur sur descripteur de fichier et pas un descripteur :

si l'on veut déclarer une variable fic pour manipuler un fichier :

FILE *fic;

on peut ensuite procéder aux différentes opérations, toujours dans l'ordre :

ouvrir / lire ou écrire / fermer le fichier.

Ouverture de fichiers

Fichier texte : ouverture du fichier : faire le lien entre le fichier sur disque (qui porte un nom bien défini) et le descripteur.

Fonction *fopen* indiquant si l'ouverture s'est correctement effectuée :

prototype :

FILE **fopen*(char **nom_du_fichier*, char **mode_d_ouverture*);

le nom du fichier est le nom classique (relatif ou absolu), par exemple : **valeurs.txt** ou **C:\essais\test\values.dat** tel qu'on le connaît sous DOS par exemple.

Attention, en C, au caractère \ dans les chaînes de caractère ! Écrire \\ dans les chaînes de caractère !

"C:\\essais\\test\\values.dat"

Ouverture de fichiers

Les modes d'ouverture : important ! Détermine le type d'accès autorisés pour le fichier. Valeurs déjà définies :

"`r`" (Read) : accès en lecture seulement, le fichier doit déjà exister.

"`w`" (Write) : ouvre un fichier vide pour y écrire. Si le fichier existe déjà, son contenu est détruit.

"`a`" (Append) : ouvre un fichier en écriture, si le fichier existe déjà, écritures à la suite des données existantes.

"`r+`" : ouverture en lecture/écriture, le fichier doit déjà exister;

"`w+`" : ouverture d'un fichier vide en lecture/écriture, si le fichier existe déjà, son contenu est détruit.

"`a+`" : comme "`a`"

Ouverture de fichiers

Les modes d'ouverture : on peut préciser si on travaille avec des fichiers **t**exte ou **b**inaire, en ajoutant 't' ou 'b' au mode d'ouverture :

"rt" : (Read Text)

"wb" : (Write Binary)

"wt+" : (Write Text, lecture/écriture)

choix non obligatoire, dépend d'une variable d'environnement… risqué ! Toujours préciser **t** ou **b**.

résultats donnés par *fopen* : si l'ouverture n'a pas fonctionné correctement, *fopen* renvoie la valeur NULL (pointeur indéfini), sinon, le pointeur de fichier est initialisé (la valeur elle-même n'est pas intéressante).

Ouverture de fichiers

Gestion du résultat de fopen : si le fichier n'est pas ouvert correctement, inutile de continuer le programme : on ne pourra rien faire.

Erreurs dues : à un mauvais nom de fichier la plupart du temps !

Test systématique de la valeur du descripteur de fichier juste après *fopen*.

Ouverture par fopen

si l'ouverture s'est bien passée (et seulement à cette condition)

| suite du programme

Ouverture de fichiers

Traduction en C : exemple de programme où l'on cherche à lire le contenu d'un fichier de type texte.

```
#include <stdio.h> /* pour fopen également*/
void main()
{
	FILE *monFic;

	monFic=fopen("toto.txt","rt"); /* rt : Read Text */

	if (monfic==NULL) /* test de l'ouverture */
	{
		printf("fichier non ouvert !\n");
	}
	else
	{
	/* suite du programme */
	/* parmi lequel lecture des valeurs dans le fichier */
	}
}
```

Ouverture de fichiers

Autre forme 'classique' : test fait sur la même ligne que l'ouverture. Réalise exactement la même chose !

```
#include <stdio.h> /* pour fopen également*/
void main()
{
	FILE *monFic;

	if ( (monFic=fopen("toto.txt","rt")) ==NULL)
	{
		printf("fichier non ouvert !\n");
	}
	else
	{
	/* suite du programme */
	/* parmi lequel lecture des valeurs dans le fichier */
	}
}
```

Traitement des fichiers texte

Comment lire ou écrire des valeurs dans un fichier texte ?

Principe : ressemblance avec saisies (scanf) et affichage (printf) mais avec des sources (de saisie) et destinations (d'affichage) différentes.

En fait, une saisie est une lecture depuis le clavier. On peut faire ces lectures à partir d'autres dispositifs d'entrées ou **flots** d'entrée. Un fichier peut jouer ce rôle.

De même, un affichage est une écriture vers l'écran. Écritures possibles vers d'autres dispositifs de sortie ou **flots de sortie**. Un fichier peut aussi jouer ce rôle.

Position du descripteur

Le descripteur, indique, entre autres, l'endroit du fichier où se déroule la prochaine opération. Après une ouverture, le descripteur indique la première valeur (en fait le premier octet du fichier).

A chaque fois qu'un opération a lieu, le descripteur indiquera l'endroit où s'est terminée cette opération, et donc là où aura lieu la suivante.

Exemples : si on lit 4 octets, le descripteur indiquera 4 octets plus loin que là ou il était.

Traitement des fichiers texte

Lire des valeurs dans un fichier texte : comment est composé un fichier texte ?

Constitué de lignes. Comme pour le texte, les lignes sont terminées par un retour à la ligne '\n' : délimiteur, sert de point de repère.

Chaque ligne est constitué de caractères.

La fin du fichier est marquée par un caractère spécial (CTRL-Z) de fin de fichier. Cette fin de fichier est aussi nommée EOF (End Of File).

Accès séquentiel : accès au premier élément du fichier, on ne sait pas combien de lignes il y a, ni combien de caractère sur chaque ligne. On peut juste demander si on est à la fin du fichier ou non.

Traitement des fichiers texte

Lire des valeurs dans un fichier texte : comment est composé un fichier texte ?

Pour lire des valeurs, fonction *fscanf* (scanf, connu, le **f** précédent indique que l'on va lire un **f**ichier).

Lecture formatée comme pour scanf : on peut lire des entiers, des nombres à virgule, des caractères : ne pas tout mélanger pour repérer les fins de ligne !

En général, format du fichier (manière dont sont organisées les données) connu avant la lecture.

Arguments différents de scanf : il faut préciser le descripteur du fichier avec lequel on veut travailler !

Traitement des fichiers texte

Un exemple de fichier texte : une petite fiche de renseignements.

On a collecté quelques informations sur une personne : nom, prénom, âge, taille en m et poids en kg. Le fichier texte est le suivant :

```
Martin
Jean
28
1.83
94
```

en fait, avec les délimiteurs, il est le suivant:

`Martin\nJean\n28\n1.83\n94\nCTRL-Z`

rappel : constitué de caractères !

Traitement des fichiers texte

Un exemple de fichier texte : une petite fiche de renseignements.

Choix des formats de lecture :
1 chaîne de caractères : %s
1 chaîne de caractères : %s
1 entier : %d
1float : %f
1 entier : %d

donc 5 fscanf à la suite avec les formats appropriés :

prototype de fscanf :

*int fscanf(FILE *fichier, char *chaine_format,variables…);*

la valeur de retour est le nombre de lectures réussies avec le format spécifié.

Traitement des fichiers texte

```
#include <stdio.h>
void main()
{
	FILE *fildesc;
	char nom[50], prenom[50];
	int age, poids;
	float taille;

	fildesc=fopen("data.txt","rt");

	if (!fildesc)
	{
		printf("fichier non ouvert !\n");
	}
```

Traitement des fichiers texte

```
else
        {
                fscanf(fildesc,"%s",nom);
                fscanf(fildesc,"%s",prenom);
                fscanf(fildesc,"%d",&age);
                fscanf(fildesc,"%f",&taille);
                fscanf(fildesc,"%d",&poids);

                printf("donnees lues: %s %s \n %d %f %d\n",nom,
prenom, age, taille, poids);
        }

        fclose(fildesc); /* fermeture du fichier */
}
```

Formats choisis en fonction de ce que contient le fichier.

Traitement des fichiers texte

Formats non imposés : on peut lire le fichier caractère par caractère avec le format %c : approche différente.

Algorithme : lire un caractère dans le fichier, l'afficher à l'écran dans une boucle.

A faire tant que l'on n'est pas à la fin du fichier.

Fin du fichier repéré grâce à l'emploi de la fonction *feof* :

prototype :

int feof(FILE fic);*

indique si on est à la fin du fichier si la valeur de retour est 0, sinon la valeur de retour est non nulle.

Traitement des fichiers texte

```
#include <stdio.h>

void main()
{
	FILE *fildesc;

	char carac;

	fildesc=fopen("data.txt","rt");
	if (!fildesc)
	{
		printf("fichier non ouvert !\n");
	}
```

Traitement des fichiers texte

```
        else
        {
                printf("fichier ouvert !\n");

                while(feof(fildesc) != 0)
                {
                        fscanf(fildesc,"%c",&carac);
                        printf("%c ",carac);
                }
        }

        fclose(fildesc);
}
```

Toujours garder une trace du format d'un fichier texte pour pouvoir le relire par la suite !

Traitement des fichiers texte

Possibilité d'avoir plusieurs valeurs sur une ligne : attention à bien choisir le format en conséquence. Exemple : fichier texte nommé **vals.txt** contenant une suite de coordonnées de points en 2 dimensions.

1.57 2.105
2.1384 3.2546
1.714 1.002

chaque ligne comporte deux éléments à lire : plusieurs possibilités

lire individuellement chaque valeur float par un fscanf

lire les deux valeurs de chaque ligne par un seul fscanf :

solutions :

Traitement des fichiers texte

Reprise du programme précédent; à la place de char carac; on a la ligne : float val1, val2;
on fait 6 lectures d'un float dans la boucle, sans gérer la fin de fichier :

```
fscanf(fildesc,"%f",&val1);
printf("%f\n",val1);
```
} Répété 6 fois

ou

```
fscanf(fildesc,"%f %f",&val1,&val2);
printf("%f %f\n",val1,val2);
```
} Répété 3 fois

Même effet : lire les 6 valeurs au format %f.

toujours lisible caractère par caractère !

Traitement des fichiers texte

Écriture de caractères dans un fichier texte :
utilisation de la fonction *fprintf* dont le prototype est :

int fprintf(FILE *fic, char *format, variables…);

la valeur de retour indique le nombre de caractères écrits
fic : descripteur de fichier sur lequel on veut lire
autres paramètres : comme printf
un '\n' écrit dans un fichier provoquera un passage à la ligne
le caractère de fin de fichier est ajouté automatiquement, pas besoin de le mettre.

Exemple : on veut écrire un fichier texte (nouveau) pour stocker les renseignements vus tout à l'heure.

Traitement des fichiers texte

```
#include <stdio.h>

void main()
{
	char nom[50]="Martin";
	char prenom[50]="Jean";
	int age=28;
	float taille=1.84;
	int poids = 95;
	FILE *ficdata;

	ficdata=fopen("donnees.txt","wt");

	if (ficdata==NULL)
	{
		printf("fichier non ouvert\n");
	}
```

Traitement des fichiers texte

```
        else
        {
                                        fprintf(ficdata,"%s\n%s\n%d\n
%f\n%d\n",nom,prenom,age,taille,poids);

        fclose(ficdata);
        system("type donnees.txt");
        }
}
```

Ou pouvait aussi écrire données par données, ou même les valeurs directement avec des caractères avec la ligne suivante :

```
fprintf(ficdata,"Martin\nJean\n28\n1.84\n95\n");
```

Traitement des fichiers binaires

Données non formatées, pas de repère, pas de lecture/écriture formatée ni caractère par caractère : associé au stockage de données complexes comme les struct.

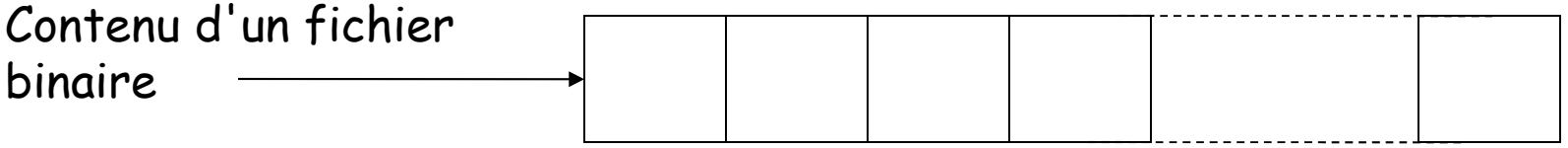
Gestion séquentielle comme pour les fichiers texte, mais possibilité d'accès direct en déplaçant le pointeur (descripteur) à un endroit précis du fichier sans avoir à tout lire.

On doit par contre lire ou écrire paquet de données par paquet de données.

On représentera un fichier binaire comme une suite de données complexes qui ont toutes le même format : comme un tableau qui serait situé sur le disque.

Traitement des fichiers binaires

Représentation comme un tableau sur disque, chaque case comprenant une donné complexe.



Transfert d'un élément minimum à chaque fois grâce aux fonctions de lecture / écriture suivantes :

fread et fwrite

nécessitent l'utilisation d'un tampon pour accueillir les données à transférer : de la mémoire libre (comme pour les variables en fait)

Traitement des fichiers binaires

Prototype de fread :

*size_t fread(void *tampon, size_t size, size_t count, FILE *fic);*

Rôle des différents paramètres :

tampon : zone de mémoire destinée à recevoir les données lues : en fait, lorsque l'on fait une lecture de valeur (scanf, fscanf), le résultat est rangé en mémoire (dans une variable); la donnée récupérée doit être rangée quelque part : ici, plus de liberté pour choisir (ce qui peut être troublant).

Type void* : indique un pointeur sur un type qui n'est pas encore défini, permet de la souplesse en utilisant le transtypage.

Traitement des fichiers binaires

Prototype de fread :

*size_t fread(void *tampon, size_t size, size_t count, FILE *fic);*

Rôle des différents paramètres :

size_t **size** : le type size_t indique un entier non signé qui indique une taille (normalement donné par l'emploi de sizeof).

Ici, la taille en octets d'un élément complexe à lire, cela permet de décaler correctement le descripteur

size_t **count** : un entier non signé indiquant le nombre d'éléments complexes à lire.

Le nombre total d'octets lus sera donc size * count.

Traitement des fichiers binaires

Prototype de fread :

*size_t fread(void *tampon, size_t size, size_t count, FILE *fic);*

Rôle des différents paramètres :

FILE* **fic** : le fichier qui est concerné par cette opération.

La valeur de retour est le nombre d'éléments complexes effectivement lus grâce à fread. Ce nombre peut être plus petit que le paramètre count si on rencontre la fin de fichier.

Traitement des fichiers binaires

Prototype de fwrite :

*size_t fwrite(void *tampon, size_t size, size_t count, FILE *fic);*

Même rôle pour les paramètres qu'avec fread :

tampon : zone de mémoire où se trouvent les éléments à écrire dans le fichier

size : taille en octets d'un élément

count : nombre d'éléments à écrire

fic : fichier où aura lieu l'écriture.

La valeur de retour indique le nombre d'éléments effectivements écrits.

Fichiers binaires et tableaux

Exemple de programmes utilisant des fichiers binaires : utiliser la similitude entre tableaux et fichiers binaires : seule application vue dans ce cours.

Premier programme : tableau contenant des struct, à écrire dans un fichier binaire.

La structure est définie ainsi :

```
typedef struct s_complex
{
float  re;
float  im;
}t_complex;
```

Fichiers binaires et tableaux

Que va faire le programme ?

Déclarer un tableau contenant des valeurs de type t_complex;

Initialiser le tableau;

ouvrir un fichier binaire en écriture (sans append)

si l'ouverture s'est bien déroulée

    écrire les éléments du tableau dans le fichier

    fermer le fichier

Fichiers binaires et tableaux

```
#include <stdio.h>
/* définition du type complex */
void main()
{
	t_complex tabc[3]={{1.0,1.0},{2.0,3.0},{3.0,-1.0}};

	FILE *ficbin;

	if ((ficbin=fopen("comp.bin","wb")) != NULL)
	{
		fwrite(tabc,sizeof(t_complex),1,ficbin);
		fwrite(tabc+1,sizeof(t_complex),2,ficbin);
		fclose(ficbin);
	}
}
```

où remplacer les 2 fwrite par un seul :

```
	fwrite(tabc,sizeof(t_complex),3,ficbin);
```

Fichiers binaires et tableaux

Lecture d'un fichier binaire pour placer les valeurs lues dans un tableau : il faut par contre connaître par avance le nombre d'éléments stockés dans le fichier ou utiliser une boucle pour lire le fichier.

Dans l'exemple qui suit, on va reprendre le fichier créé par l'écriture précédente et le relire.

On garde le type créé t_complex, défini de la même manière.

Programme très ressemblant, on utilisera fread à la place de fwrite !

Fichiers binaires et tableaux

```
#include <stdio.h>

/* définition du type complex */

void main()
{
	t_complex tabc[3]; /* non initialisé, on va lire */

	FILE *ficbin;

	if ((ficbin=fopen("comp.bin","rb")) != NULL)
	{
		fread(tabc,sizeof(t_complex),3,ficbin);
		fclose(ficbin);
	}
}
```

Accès direct

La fonction fseek permet de se déplacer dans le fichier d'un certain nombre d'octets : attention à convertir ce nombre !

Par exemple, on doit aller lire le $k^{ième}$ élément stocké dans un fichier binaire, la valeur des éléments précédents en nous intéresse pas.

Prototype de la fonction fseek :

int fseek(FILE **fic*, long *depl*, int *origin*);

fseek renvoie la valeur 0 si elle a bien fonctionné.

Rôle des paramètres :

fic : le fichier sur lequel on travaille

Accès direct

depl : valeur du déplacement en nombre d'octets : si l'on connaît la taille en octets d'un élément (on le peut grâce à sizeof), on multiplie par le nombre d'éléments pour obtenir cette valeur.

Origin : indique à partir de quel endroit du fichier est calculé le déplacement, il a 3 valeurs possibles seulement :

SEEK_CUR : par apport à la position actuelle

SEEK_END : par rapport à la fin du fichier

SEEK_SET : par rapport au début du fichier

Accès direct

Exemple : avec le fichier contenant les valeurs des nombres complexes, on veut seulement lire la troisième valeur sans se préoccuper des autres :

on va ouvrir le fichier, puis déplacer le descripteur pour aller directement là ou se trouve les valeurs du 3ème élément.

On doit se décaler de 2 éléments à partir du début du fichier : calcul du nombre d'octets dont il faut se déplacer :

2*sizeof(t_complex) !

fseek(ficbin, 2*sizeof(t_complex),SEEK_SET);

Accès direct

autre exemple : toujours avec l'exemple sur les nombres complexes : après une lecture par fread, on veut revenir un élément en arrière dans le fichier :

on se déplacera à partir de l'endroit où l'on se trouve, le nombre d'octets du déplacement sera donné par : 1*sizeof(t_complex);

de plus le déplacement se fait en arrière : valeur négative :

fseek(ficbin,-sizeof(t_complex),SEEK_CUR);

faire très attention à la valeur du déplacement, sinon risque d'être perdu dans le fichier.

Fermeture des fichiers

Quelque soit le type de fichier utilisé, texte ou binaire, il faut, après avoir fini de travailler avec, le fermer : ajoute la fin de fichier.

Utilisation de la fonction fclose :

int fclose(FILE* fic);

la valeur de retour vaut 0 si la fermeture s'est bien passée. Inutile de tester cette valeur !

On n'effectue le fclose que si le fichier était bien ouvert, c'est à dire si le fopen du fichier s'est bien déroulé.

À retenir quand on travaille avec un fichier : ouverture - si l'ouverture s'est bien déroulée alors opérations et traitements puis fermeture.

Gestion de la fin de fichier

Attention à l'emploi de la fonction feof : on a précisé qu'elle indique si oui ou non on est à la fin du fichier. En fait, elle indique si la dernière opération menée sur le fichier a rencontré la fin de fichier ou non. C'est légèrement différent !

Exemple : lecture d'un fichier texte contenant un nombre inconnu de caractères…(on pourrait le connaître, mais comment ?)

En fait,on risque de traiter 2 fois la dernière demande de lecture par fscanf !

Considérons la boucle typique (déjà présentée d'ailleurs) de lecture avec test de fin de fichier :

Gestion de la fin de fichier

```
while (feof(fic) != 0)
{
        lecture suivante;
        rangement de la valeur lue;
}
```

et appliquons là à un fichier de taille inconnue. On considère que l'on a lu tous les caractères sauf le dernier.

- feof(fic) indique que l'on n'est pas à la fin
- on lit le dernier caractère : la lecture n'a pas encore rencontré la fin de fichier, puisque c'est le dernier caractère !
- on range la valeur lue
- on reteste feof(fic) : on n'a pas encore rencontré la fin
- lecture suivante : on rencontre la fin de fichier !
- rangement de la valeur lue !

Gestion de la fin de fichier

```
while (feof(fic) != 0)
{
        lecture suivante;
        rangement de la valeur lue;
}
```

on est obligé de faire la lecture suivante pour rencontrer la fin de fichier, par contre, on doit tester *feof* avant de ranger la valeur lue.

Rajouter un *feof* supplémentaire, ou utiliser une variable booléenne pour marquer que l'on a rencontré la fin du fichier :

```
while (pasFini == 1)
{
        lecture suivante;
        if (feof(fic)!=0
        {rangement de la valeur lue;}
        else
        {pasFini = 0}
}
```