Cours de Programmation 2 (Programmation C - structures de données) cours n°5 (2e partie) Entrées-sorties (1re partie : fichiers de type texte) Entrées-sorties (2e partie : fichiers binaires)

Christophe Tollu
(Première version des diapos par Julien David)

A209 (poste 3691) - ct@lipn.univ-paris13.fr

23 avril 2024

Ouverture / fermeture de fichiers en C

Les fichiers en C

Résumé

Tout comme pour les interactions avec le terminal :

- le programme n'interagit pas directement avec le disque dur ;
- c'est le système d'exploitation qui se charge de la communication entre les différents composants ;
- quand un programme (processus) veut accéder à un fichier, il demande au système de lui donner un accès ;
- le système va alors créer un espace mémoire où le programme pourra lire et/ou écrire ;
- cet espace mémoire est appelé **tampon** (*buffer* en anglais) ;
- on va voir aujourd'hui comment manipuler cet espace mémoire.

Les fichiers : un nouveau type de variable

Le type `FILE`

- Le type `FILE` permet de manipuler des fichiers.
- Il s'agit d'une structure (définie dans `<libio.h>`, incluse dans `<stdio.h>`) qui contient
  - l'adresse du tampon,
  - la position de la **tête de lecture/écriture** dans le fichier,
  - les droits que le programme possède sur le fichier.

En pratique

- On utilise toujours une variable de type `FILE *` .
- Un objet de type `FILE *` est (aussi) appelé un *flux (de données)* (*stream* en anglais).

Les fonctions : ouverture d'un fichier

Ouvrir un fichier

- `FILE *fopen(const char *path, const char *mode);`

Arguments

- `const char *path` : nom du fichier à ouvrir (comprenant le chemin absolu, si `path` commence par "/", ou relatif vers ce fichier).
- `const char *mode` : mode d'ouverture du fichier.

Les fonctions : ouverture d'un fichier

Commmande	Mode	Position	Fichier inexistant
”r”	lecture	Début	renvoie NULL
”r+”	lecture+écriture	Début	renvoie NULL
”w”	écriture	Début	Crée le fichier.
”w+”	lecture+écriture	Début	Crée le fichier.
”a”	écriture	Fin	Crée le fichier.
”a+”	lecture+écriture	R : Début W :Fin	Crée le fichier.

Les fonctions : ouverture d'un fichier

Valeur de retour de `fopen`

- `fopen` renvoie le flux de données associé au fichier ouvert si l'appel a réussi à ouvrir le fichier.
- `fopen` renvoie `NULL` sinon.
- Il faut **toujours tester la valeur de retour de cette fonction**.

Ouverture d'un fichier

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

int main(){
  FILE * f;
  if((f=fopen("test.txt","r"))!=NULL){
    fprintf(stderr, "Le fichier existe\n");
    fclose(f);
    return EXIT_SUCCESS;
  }
  else {
    fprintf(stderr, "Le fichier n'existe pas\n");
    return EXIT_FAILURE;
  }
}
```

Les fonctions : fermeture d'un fichier

La fonction `fclose`

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

- La fonction `fopen` exécute un `malloc` pour créer le tampon/buffer qui va servir à interagir avec le disque dur.
- La fonction `fclose` permet de **libérer ce bloc mémoire** et si besoin, force le système à écrire sur le disque les données du tampon qui n'ont pas encore été transférées.
- Elle renvoie 0 si l'opération s'est déroulée normalement, EOF (souvent -1) dans le cas contraire.

La manipulation de texte en C

Les entrées-sorties formatées : écriture dans un fichier

La fonction `fprintf`

- Fonctionne (presque) comme la fonction `printf` : le premier argument de la fonction est le flux associé au fichier dans lequel on souhaite écrire.
- Son entête est
  ```
  int fprintf(FILE *flux, const char *format, ...)
  ```
- Elle **renvoie le nombre total de caractères écrits dans le tampon** en cas de succès, un nombre négatif en cas d'échec.

Écriture dans un fichier

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

int main(){
  FILE * fp;
  char c;

  fp = fopen ("nouveau.txt", "a");
  fprintf(fp, "%s %s %s %d", "Nous", "sommes", "en", 2021);
  fclose(fp);

  fp = fopen("nouveau.txt","r");
  fscanf(fp,"%c",&c);
  while (!feof(fp)){
    printf("%c", c);
    fscanf(fp,"%c",&c);
  }
  printf("\n");
  fclose(fp);
  return EXIT_SUCCESS;
}
```

Les entrées-sorties formatées : lecture à partir d'un fichier

La fonction `fscanf`

- Fonctionne (presque) comme la fonction `scanf` : le premier argument de la fonction est le fichier dans lequel on souhaite lire.
- Son entête est
  ```
  int fscanf(FILE *flux, const char *format, ...)
  ```
- Elle **renvoie le nombre de valeurs convenablement lues**.

Lecture et écriture dans un fichier

Les limites de l'écriture

- On peut écrire dans le fichier tant qu'il y a de la place sur le disque dur.
- On s'autorise à considérer qu'on ne sera jamais à court de place.

Les limites de la lecture

- On lit dans le fichier jusqu'à ce qu'il soit terminé.
- La fonction `int feof(FILE *f)` renvoie une valeur non nulle si la fin du fichier a été atteinte.

Lecture et écriture dans un fichier

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

int main(){
  FILE * f;
  char test[10];

  if((f=fopen("test.txt","w+"))!=NULL){
    printf("Le fichier existe\n");
    fprintf(f,"Je suis un fichier test\n");
    fclose(f);
  }
  else
    printf("Le fichier n'existe pas\n");

  if((f=fopen("test.txt","r"))!=NULL){
    while(!feof(f)){
      if(fscanf(f,"%s",test)==1)
        printf("%s\n",test);
    }
    fclose(f);
  }
  else
    printf("Le fichier n'existe pas\n");

  return EXIT_SUCCESS;
}
```

Fichiers texte
binaire

Les différents types de fichiers

Les fichiers texte

- Les caractères comme le retour à la ligne, la tabulation, etc. y sont interprétés comme tels lors de la lecture et de l'écriture.
- Ils peuvent être lus et modifiés directement par un éditeur.
- Les fonctions d'entrées-sorties (`fprintf` et `fscanf`) **lisent et écrivent des chaînes de caractères formatées**.

Les fichiers binaires

- Les caractères comme le retour à la ligne, la tabulation, etc. n'y sont pas interprétés.
- Il est très compliqué de les lire avec un éditeur sans avoir d'informations complémentaires.
- Les données peuvent y être mieux compressées.
- Des fonctions d'entrées-sorties binaires **lisent ou écrivent le contenu intégral d'un bloc mémoire sans se soucier de son interprétation**.

Fichiers textes

Fichiers texte

- Dans un fichier texte, **chaque octet est interprété comme un caractère ASCII**.

Fonctions spécifiques pour la manipulation des fichiers textes

- Les fonctions `fprintf` et `fscanf`.
- La fonction `int fgetc(FILE *flux)` lit un caractère (`unsigned char`) du flux et le renvoie (converti en `int`) ; elle renvoie EOF si la tête de lecture a atteint la fin du fichier ou en cas d'échec.
- La fonction `int fputc(int car, FILE *flux)` écrit le caractère passé en argument dans le flux et le renvoie ; elle renvoie EOF en cas d'échec de l'écriture.

Les fonctions d'entrée-sortie

`printf` et `fprintf`

- Ces fonctions **écrivent toujours les caractères ASCII des valeurs passées en arguments**..
- Pour les chaînes de caractères, c'est assez intuitif.
- Pour les nombres. . . voyons avec un exemple.

```
int x=5000000;
printf("%d",x);
```

  La fonction va afficher 7 caractères, correspondant aux 7 chiffres.
- Si on effectue la même opération avec `fprintf`
  - on écrira 7 caractères dans un fichier.
  - on écrira donc 7 octets de données.
- Or un `int` occupe 4 octets et peut stocker des valeurs bien plus grandes.

Fichiers texte / fichiers binaires

Format de stockage

Pour les données numériques en particulier :

- on stocke directement la représentation binaire des valeurs
- plutôt que le code ASCII de chaque chiffre.

On obtient un fichier qui occupe moins d'espace en mémoire.

La manipulation de binaire en C

Fonctions d'entrée-sortie sans formatage

Fonction de lecture `fread` (`<stdio.h>`)

```
size_t fread(void *ptr, size_t taille, size_t nmemb, FILE *flux);
```

- Lit dans le flux de données `flux` au maximum `nmemb` blocs de `taille` octets.
- Copie le contenu à l'adresse `ptr`.
- Renvoie le nombre de blocs réellement lus.

Fonction d'écriture `fwrite` (`<stdio.h>`)

```
size_t fwrite(const void *ptr, size_t taille, size_t nmemb, FILE *flux);
```

- Écrit dans le flux de données `flux` `nmemb` blocs de `taille` octets chacun.
- Le contenu écrit est celui qui se trouve à l'adresse `ptr`.
- Renvoie le nombre de blocs réellement écrits.

Fichiers binaires / fichiers texte

Fonctions `fread` et `fwrite`

- Les fonctions `fread` et `fwrite` permettent de copier des blocs de mémoire...
  - de la RAM vers un fichier (fwrite).
  - d'un fichier vers la RAM (fread).
- En particulier, la fonction `fread` permet de charger et de découper très rapidement le contenu d'un fichier.

Et les fichiers texte ?

- On peut utiliser `fread` pour lire dans un fichier texte.
- Si la zone de mémoire lue contient des caractères ASCII, ces fonctions les préservent.

Fichiers binaires

Formats de fichiers

Il est essentiel de convenir d'un format lorsqu'on définit un fichier de données binaires.

Exemple

On va voir un exemple avec un format de fichier pour stocker des images.

Exemple d'un format image

Exemple : le format `.ppm`

Les fichiers `.ppm` sont des fichiers de type image dont le format est le suivant :

- en-tête (en mode texte) :
  - les deux premiers caractères permettent d'annoncer le format de fichier : *P*6,
  - les deux valeurs suivantes correspondent à la largeur et à la hauteur de l'image,
  - la dernière valeur v fixe le nombre de nuances possibles pour une couleur (pour faire simple, on suppose que $v = 255$) ;
- données de l'image (en mode binaire) :
  - chaque pixel est décomposé en nuances Rouge-Vert-Bleu (3 valeurs comprises entre 1 et v),
  - le nombre de pixels est $\textit{largeur} \times \textit{hauteur}$,
  - la taille du bloc mémoire nécessaire pour stocker toute l'image est $3 \times \textit{largeur} \times \textit{hauteur}$ octets.

La structure image

```
1  #ifndef _IMAGE_H
2  #define _IMAGE_H
3
4  struct image_s{
5    int longueur;
6    int hauteur;
7    int max;
8    unsigned char * data;
9  };
0
1  typedef struct image_s image_t;
2
3
4  image_t * image_creer(int longueur,int hauteur,int max);
5  void image_detruire(image_t * im);
6  image_t * image_charger(char * chemin);
7  void image_ecrire(char * chemin,image_t * im);
8
9  #endif
```

Charger un fichier .ppm

```
1  image_t * image_charger(char * chemin){
2    FILE * f;
3    image_t *res;
4    int longueur,hauteur,max;
5    if((f=fopen(chemin,"r"))!=NULL){
6      fscanf(f,"P6 %d %d %d",&longueur,&hauteur,&max);
7      res=image_creer(longueur,hauteur,max);
8      fread(res->data,sizeof(unsigned char),3*longueur*hauteur,f);
9      fclose(f);
0    }
1    return res;
2  }
```

Sauvegarder un fichier .ppm

```
1 void image_ecrire(char * chemin,image_t * im){
2   FILE * f;
3   if((f=fopen(chemin,"w"))!=NULL){
4     fprintf(f,"P6 %d %d %d\n",im->longueur,im->hauteur,im->max);
5     fwrite(im->data,sizeof(unsigned char),3*im->longueur*im->hauteur,f);
6     fclose(f);
7   }
8 }
```

« Accè
fichiers

Fonctions manipulant le « pointeur de fichier »

Accès direct

```
int fseek(FILE * flux, long int deplacement, int position) ;
```

- Permet d'**accéder directement à un octet** dans un fichier.
- La position par rapport à laquelle on calcule le déplacement (en nombre d'octets) est donnée par l'une des **trois constantes** suivantes : `SEEK_SET` (début du fichier), `SEEK_CUR` (position courante), `SEEK_END` (fin du fichier).
- Renvoie 0 si l'accès a été un succès, une valeur non nulle dans le cas contraire.

Position courante

```
long int ftell(FILE * flux) ;
```

- Renvoie la **position courante** du pointeur de fichier.
- Attention, **dans le cas d'un fichier texte**, la valeur renvoyée n'est pas facile à interpréter car certains caractères peuvent être représentés par plusieurs octets (selon le codage adopté) !

Quizz

```c
#include <stdio.h>
#include <stdlib.h>
int main(void){
  FILE *f;
  int *tab;
  int i;

  tab = malloc(sizeof(int)*100);
  for (i = 0 ; i < 100; i++)
    tab[i] = i;
  /* Ecriture du tableau dans le fichier "fichier_test" */
  if ((f = fopen("fichier_test", "w")) == NULL) {
    printf("Impossible d'ecrire dans le fichier %s\n","fichier_test");
    return EXIT_FAILURE;
  }
  fwrite(tab, sizeof(int), 100, f);
  fclose(f);
  /* lecture dans le fichier "fichier_test" */
  if ((f = fopen("fichier_test", "r")) == NULL) {
    printf("Impossible de lire dans le fichier %s\n","fichier_test");
    return EXIT_FAILURE;
  }
  /* on se positionne a la fin du fichier */
  fseek(f, 0, SEEK_END);
  printf("position %ld", ftell(f));
  /* deplacement de 20 entiers (80 octets) en arriere */
  fseek(f, sizeof(int)*(-20), SEEK_END);
  printf("\n position %ld", ftell(f));
  fread(&i, sizeof(i), 1, f);
```

Les entêtes

- Un entête, ou `header` est un ensemble d'informations que l'on place au début du fichier et qui donne des informations sur :
  - le format dans lequel est écrit le fichier ;
  - les informations qu'il contient.

Par exemple

- on peut indiquer combien de lignes le fichier contient, ce qui va faciliter l'écriture du programme ;
- pour une image ou une vidéo, on peut préciser ses dimensions, le format (bmp,jpeg,gif,png,mpeg,avi).

Solution avec entête

4		
Programmation Impérative	20	Très bien
Logique	15	
Algèbre 2	08	étudiant hostile à Bourbaki
ISDL	18	

Découpage de l'information

- La première ligne constitue l'entête.
- On sait donc dès le début combien d'espace mémoire il va falloir allouer et comment utiliser l'appel à `fread`.

Découpage « propre » de l'information contenue dans un fichier

```
1  /* Memes inclusions et definition de type que pour le programme precedent */
2  int main(int argc,char ** argv){
3    FILE * f;
4    matiere_t * m;
5    int nb_ligne,i;
6    if((f=fopen(argv[1],"r"))!=NULL){
7      fscanf(f,"%d \n",&nb_ligne);
8      printf("Le nombre de lignes est %d\n",nb_ligne);
9     m=(matiere_t *)malloc(sizeof(matiere_t)*nb_ligne);
0      fread(m,sizeof(matiere_t),nb_ligne,f);
1      for(i=0;i<nb_ligne;i++){
2        m[i].nom[29]='\0';
3         printf("-----------------------------\n");
4         printf("Nom du cours : %s\n",m[i].nom);
5        m[i].note[2]='\0';
6         printf("Note : %s\n",m[i].note);
7        m[i].commentaire[29]='\0';
8         printf("Commentaire : %s\n",m[i].commentaire);
9      }
0          free(m);
1      fclose(f);
2    }
3    else
4      printf("Le fichier %s n'existe pas\n",argv[1]);
5      return EXIT_SUCCESS;
6  }
```

Conclusion

Pour que les interactions fichiers/programmes soient optimales

- Il faut que le format du fichier soit réfléchi, que la disposition de l'information soit organisée, normalisée.
- On a tout juste entrevu quelques possibilités de formats de fichiers.