CHARLES CARNUS
LYCÉE TECHNIQUE
ENSEIGNEMENT SUPÉRIEUR

Langage C

Gestion des fichiers

Prof : **Kamal Boudjelaba**

22 septembre 2022

B T S S N

LTP Charles Carnus
Avenue de Saint Pierre - Bourran - 12000 RODEZ, France

IR–EC

Table des matières

1. Introduction

Le C offre la possibilité de lire et d'écrire des données dans un fichier.
Pour des raisons d'efficacité, les accès à un fichier se font par l'intermédiaire d'une mémoire-tampon (buffer), ce qui permet de réduire le nombre d'accès aux périphériques (disque, clé ...).
Pour pouvoir manipuler un fichier, un programme a besoin d'un certain nombre d'informations :

— l'adresse de l'endroit de la mémoire-tampon où se trouve le fichier

— la position de la tête de lecture

— le mode d'accès au fichier (lecture ou écriture)

— ...

Ces informations sont rassemblées dans une structure dont le type, `FILE *`, est défini dans `stdio.h`. Un objet de type `FILE *` est appelé flot (flux) de données (stream). Avant de lire ou d'écrire dans un fichier, on notifie son accès par la commande `fopen`. Cette fonction prend comme argument le nom du fichier et initialise un flot de données, qui sera ensuite utilisé lors de l'écriture ou de la lecture. Après les traitements, on annule la liaison entre le fichier et le flot de données grâce à la fonction `fclose`.

2. Ouverture et fermeture d'un fichier

2.1 La fonction `fopen`

Cette fonction, de type `FILE*` ouvre un fichier et lui associe un flot de données. Sa syntaxe est :

```
fopen("nom_de_fichier","mode")
```

La valeur retournée par `fopen` est un flot de données. Si l'exécution de cette fonction ne se déroule pas normalement, la valeur retournée est le pointeur NULL. Il est donc recommandé de toujours tester si la valeur renvoyée par la fonction `fopen` est égale à NULL afin de détecter les erreurs (lecture d'un fichier inexistant ...).
Le premier argument de `fopen` est le nom du fichier, fourni sous forme d'une chaîne de caractères. On peut aussi définir le nom du fichier par une constante symbolique au moyen de la directive `define` plutôt que d'expliciter le nom de fichier dans le corps du programme.
Le second argument, mode, est une chaîne de caractères qui spécifie le mode d'accès au fichier. Les spécificateurs de mode d'accès diffèrent suivant le type de fichier considéré :

— les fichiers textes, pour lesquels les caractères de contrôle (retour à la ligne ...) seront interprétés en tant que tels lors de la lecture et de l'écriture.

— les fichiers binaires, pour lesquels les caractères de contrôle se sont pas interprétés.

Les différents modes d'accès sont les suivants :

Table 1. Les entiers.

Mode	Signification
`"r"`	ouverture d'un fichier texte en lecture
`"w"`	ouverture d'un fichier texte en écriture
`"a"`	ouverture d'un fichier texte en écriture à la fin
`"rb"`	ouverture d'un fichier binaire en lecture
`"wb"`	ouverture d'un fichier binaire en écriture
`"ab"`	ouverture d'un fichier binaire en écriture à la fin (en mode append)
`"r+"`	ouverture d'un fichier texte en lecture/écriture
`"w+"`	ouverture d'un fichier texte en lecture/écriture
`"a+"`	ouverture d'un fichier texte en lecture/écriture à la fin (en mode append)
`"r+b"`	ouverture d'un fichier binaire en lecture/écriture
`"w+b"`	ouverture d'un fichier binaire en lecture/écriture
`"a+b"`	ouverture d'un fichier binaire en lecture/écriture à la fin (en mode append)

Remarque : Ces modes d'accès ont pour particularités

- Si le mode contient la lettre `r`, le fichier doit exister
- Si le mode contient la lettre `w`, le fichier peut ne pas exister. Dans ce cas, il sera créé. Si le fichier existe déjà, son ancien contenu sera perdu
- Si le mode contient la lettre `a`, le fichier peut ne pas exister. Dans ce cas, il sera créé. Si le fichier existe déjà, les nouvelles données seront ajoutées à la fin du fichier précédent.

Trois flots standard peuvent être utilisés en C sans qu'il soit nécessaire de les ouvrir ou de les fermer :

- `stdin` (standard input) : unité d'entrée (par défaut, le clavier)
- `stdout` (standard output) : unité de sortie (par défaut, l'écran)
- `stderr` (standard error) : unité d'affichage des messages d'erreur (par défaut, l'écran)

Il est conseillé d'afficher systématiquement les messages d'erreur sur `stderr` afin que ces messages apparaissent à l'écran même lorsque la sortie standard est redirigée.

2.2 La fonction `fclose`

Elle permet de fermer le flot qui a été associé à un fichier par la fonction `fopen`. Sa syntaxe est :

```
fclose(flot)
```

où : `flot` est le flot de type `FILE *` retourné par la fonction `fopen` correspondant.
La fonction `fclose` retourne un entier qui vaut zéro si l'opération s'est déroulée normalement (et une valeur non nulle en cas d'erreur).

3. Les entrées–sorties formatées

3.1 La fonction d'écriture `fprintf`

La fonction `fprintf`, analogue à `printf`, permet d'écrire des données dans un fichier. Sa syntaxe est :

```
fprintf(flot, "chaine de controle", expression_1, ..., expression_n)
```

où `flot` est le flot de données retourné par la fonction `fopen`. Les spécifications de format utilisées pour la fonction `fprintf` sont les mêmes que pour `printf`.

3.2 La fonction de saisie fscanf

La fonction fscanf, analogue à scanf, permet de lire des données dans un fichier. Sa syntaxe est :
`fscanf(flot, "chaine de controle", argument_1, ..., argument_n)`

où flot est le flot de données retourné par la fonction fopen. Les spécifications de format utilisées pour la fonction fscanf sont les mêmes que pour scanf.

4. Impression et lecture de caractères

Similaires aux fonctions getchar et putchar, les fonctions fgetc et fputc permettent respectivement de lire et d'écrire un caractère dans un fichier. La fonction fgetc, de type int, retourne le caractère lu dans le fichier. Elle retourne la constante EOF lorsqu'elle détecte la fin du fichier. Son prototype est :
`int fgetc(FILE*, flot)`
où flot est le flot de type FILE* retourné par la fonction fopen. Comme pour la fonction getchar, il est conseillé de déclarer de type int la variable destinée à recevoir la valeur de retour de fgetc pour pouvoir détecter correctement la fin de fichier.

La fonction fputc écrit "caractere" dans le flot de données :
`int fputc(int caractere, FILE *flot)`
Elle retourne l'entier correspondant au caractère lu (ou la constante EOF en cas d'erreur).
Il existe également deux versions optimisées des fonctions fgetc et fputc qui sont implémentées par des macros. Il s'agit respectivement de getc et putc. Leur syntaxe est similaire à celle de fgetc et fputc :

```
int getc(FILE* flot);
int putc(int caractere, FILE *flot)
```

Le programme suivant lit le contenu du fichier texte entree, et le recopie caractère par caractère dans le fichier sortie :

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

#define ENTREE "entree.txt"
#define SORTIE "sortie.txt"

int main(void)
{
  FILE *f_in, *f_out;
  int c;
  if ((f_in = fopen(ENTREE,"r")) == NULL)
    {
      fprintf(stderr, "\nErreur: Impossible de lire le fichier %s\n",ENTREE);
      return(EXIT_FAILURE);
    }
  if ((f_out = fopen(SORTIE,"w")) == NULL)
    {
      fprintf(stderr, "\nErreur: Impossible d'ecrire dans le fichier %s\n", SORTIE);
      return(EXIT_FAILURE);
    }
  while ((c = fgetc(f_in)) != EOF)
    fputc(c, f_out);
  fclose(f_in);
  fclose(f_out);
  return(EXIT_SUCCESS);
}
```

5. 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 caractere caractere (converti en unsigned char) dans le flot flot. En particulier, si caractere 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 EOF). Par exemple, l'exécution du programme suivant sur le fichier `entree.txt` dont le contenu est 097023 affiche à l'écran `0.970.23`

```
#include <stdio.h>
#include <stdlib.h>
#define ENTREE "entree.txt"
int main(void)
{
  FILE *f_in;
  int c;
  if ((f_in = fopen(ENTREE,"r")) == NULL)
    {
      fprintf(stderr, "\nErreur: Impossible de lire le fichier %s\n",ENTREE);
      return(EXIT_FAILURE);
    }
  while ((c = fgetc(f_in)) != EOF)
    {
      if (c == '0')
        ungetc('.',f_in);
      putchar(c);
    }
  fclose(f_in);
  return(EXIT_SUCCESS);
}
```

6. Les entrées–sorties binaires

Les fonctions d'entrées–sorties binaires permettent de transférer des données dans un fichier sans transcodage. Elles sont donc plus efficaces que les fonctions d'entrée–sortie standard, mais les fichiers produits ne sont pas portables puisque le codage des données dépend des machines.
Elles sont notamment utiles pour manipuler des données de grande taille ou ayant un type composé. Leurs prototypes sont :

```
size t fread(void *pointeur, size_t taille, size_t nombre, FILE *flot);
size t fwrite(void *pointeur, size_t taille, size_t nombre, FILE *flot);
```

où `pointeur` est l'adresse du début des données à transférer, `taille` la taille des objets à transférer, `nombre` leur nombre. Rappelons que le type `size_t`, défini dans `stddef.h`, correspond au type du résultat de l'évaluation de `sizeof`. Il s'agit du plus grand type entier non signé.
La fonction `fread` lit les données sur le flot `flot` et la fonction `fwrite` les écrit. Elles retournent toutes deux le nombre de données transférées.
Par exemple, le programme suivant écrit un tableau d'entiers (contenant les 50 premiers entiers) avec `fwrite` dans le fichier `sortie`, puis lit ce fichier avec `fread` et imprime les éléments du tableau.

```
#include <stdio.h>
#include <stdlib.h>
#define NB 50
#define F_SORTIE "sortie"
int main(void)
{
    FILE *f_in, *f_out;
    int *tab1, *tab2;
    int i;
    tab1 = (int*)malloc(NB * sizeof(int));
    tab2 = (int*)malloc(NB * sizeof(int));
    for (i = 0 ; i < NB; i++)
    tab1[i] = i;
    /* Ecriture du tableau dans F_SORTIE */
    if ((f_out = fopen(F_SORTIE, "w")) == NULL)
      {
        fprintf(stderr, "\nImpossible d'ecrire dans le fichier %s\n",F_SORTIE);
        return(EXIT_FAILURE);
      }
    fwrite(tab1, NB * sizeof(int), 1, f_out);
    fclose(f_out);
    /* Lecture dans F_SORTIE */
    if ((f_in = fopen(F_SORTIE, "r")) == NULL)
      {
```

```
        fprintf(stderr, "\nImpossible de lire dans le fichier %s\n",F_SORTIE);
        return(EXIT_FAILURE);
      }
    fread(tab2, NB * sizeof(int), 1, f_in);
    fclose(f_in);
    for (i = 0 ; i < NB; i++)
      printf("%d\t",tab2[i]);
    printf("\n");
    return(EXIT_SUCCESS);
}
```

Les éléments du tableau sont bien affichés à lécran. Par contre, on constate que le contenu du fichier `sortie` n'est pas encodé.

7. Positionnement dans un fichier

Les différentes fonctions d'entrées–sorties permettent d'accéder à un fichier en mode séquentiel : les données du fichier sont lues ou écrites les unes à la suite des autres. Il est également possible d'accéder à un fichier en mode direct, c'est-à-dire que l'on peut se positionner à n'importe quel endroit du fichier. La fonction `fseek` permet de se positionner à un endroit précis ; elle a pour prototype :
`int fseek(FILE *flot, long deplacement, int origine);`
La variable `deplacement` détermine la nouvelle position dans le fichier. Il s'agit d'un déplacement relatif par rapport à l'origine ; il est compté en nombre d'octets. La variable `origine` peut prendre trois valeurs :

- `SEEK_SET` (égale à 0) : début du fichier ;
- `SEEK_CUR` (égale à 1) : position courante ;
- `SEEK_END` (égale à 2) : fin du fichier.

La fonction `int rewind(FILE *flot)` permet de se positionner au début du fichier. Elle est équivalente à `fseek(flot, 0, SEEK_SET)`.
La fonction `long ftell(FILE *flot);` retourne la position courante dans le fichier (en nombre d'octets depuis l'origine).

```
#include <stdio.h>
#include <stdlib.h>
#define NB 50
#define F_SORTIE "sortie"
int main(void)
{
    FILE *f_in, *f_out;
    int *tab;
    int i;
    tab = (int*)malloc(NB * sizeof(int));
    for (i = 0 ; i < NB; i++)
    tab[i] = i;
    /* Ecriture du tableau dans F_SORTIE */
    if ((f_out = fopen(F_SORTIE, "w")) == NULL)
    {
      fprintf(stderr, "\nImpossible d'ecrire dans le fichier %s\n",F_SORTIE);
      return(EXIT_FAILURE);
    }
    fwrite(tab, NB * sizeof(int), 1, f_out);
    fclose(f_out);
    /* Lecture dans F_SORTIE */
    if ((f_in = fopen(F_SORTIE, "r")) == NULL)
    {
      fprintf(stderr, "\nImpossible de lire dans le fichier %s\n",F_SORTIE);
      return(EXIT_FAILURE);
    }
    /* On se positionne à la fin du fichier */
    fseek(f_in, 0, SEEK_END);
    printf("\n position %ld", ftell(f_in));
    /* Déplacement de 10 int en arrière */
    fseek(f_in, -10 * sizeof(int), SEEK_END);
    printf("\n position %ld", ftell(f_in));
    fread(&i, sizeof(i), 1, f_in);
```

```
    printf("\t i = %d", i);
    /* retour au debut du fichier */
    rewind(f_in);
    printf("\n position %ld", ftell(f_in));
    fread(&i, sizeof(i), 1, f_in);
    printf("\t i = %d", i);
    /* Déplacement de 5 int en avant */
    fseek(f_in, 5 * sizeof(int), SEEK_CUR);
    printf("\n position %ld", ftell(f_in));
    fread(&i, sizeof(i), 1, f_in);
    printf("\t i = %d\n", i);
    fclose(f_in);
    return(EXIT_SUCCESS);
}
```

L'exécution de ce programme affiche à l'écran :

```
position 200
position 160    i = 40
position 0      i = 0
position 24     i = 6
```

On constate en particulier que l'emploi de la fonction `fread` provoque un déplacement correspondant à la taille de l'objet lu à partir de la position courante.

8. Exemples

Exemple 1

Le code ci-dessous tente d'ouvrir un fichier nommé `texte.txt` en lecture seule dans le dossier courant. Notez que dans le cas où il n'existe pas, la fonction fopen() retournera un pointeur nul (seul le mode r permet de produire ce comportement).

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

int main(void)
{
    FILE *fp = fopen("Fichier_Exo1.txt", "r");

    if (fp == NULL)
    {
        printf("Le fichier texte.txt n'a pas pu être ouvert\n");
        return EXIT_FAILURE;
    }
    printf("Le fichier Fichier_Exo1.txt existe\n");
    if (fclose(fp) == EOF)
    {
        printf("Erreur lors de la fermeture du flux\n");
        return EXIT_FAILURE;
    }
    return 0;
}
```

Exemple 2

L'exemple ci-dessous écrit le caractère "C" dans le fichier `texte.txt`. Étant donné qu'on utilise le mode w, le fichier est soit créé s'il n'existe pas, soit vidé de son contenu s'il existe.

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

int main(void)
{
    FILE *fp = fopen("texte.txt", "w");

    if (fp == NULL)
    {
        printf("Le fichier texte.txt n'a pas pu être ouvert\n");
        return EXIT_FAILURE;
    }
    if (fputc('C', fp) == EOF)
    {
        printf("Erreur lors de l'écriture d'un caractère\n");
        return EXIT_FAILURE;
    }
    if (fclose(fp) == EOF)
    {
        printf("Erreur lors de la fermeture du flux\n");
        return EXIT_FAILURE;
    }

    return 0;
}
```

Exemple 3

L'exemple suivant écrit le mot "Charles" suivi d'un caractère de fin de ligne au sein du fichier `texte.txt`.

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

int main(void)
{
    FILE *fp = fopen("texte.txt", "w");
```

```
    if (fp == NULL)
    {
        fputs("Le fichier texte.txt n'a pas pu être ouvert\n", stderr);
        return EXIT_FAILURE;
    }
    if (fputs("Charles\n", fp) == EOF)
    {
        fputs("Erreur lors de l'écriture d'une ligne\n", stderr);
        return EXIT_FAILURE;
    }
    if (fclose(fp) == EOF)
    {
        fputs("Erreur lors de la fermeture du flux\n", stderr);
        return EXIT_FAILURE;
    }

    return 0;
}
```

Exemple 4

L'exemple ci-dessous lit un caractère provenant du fichier `texte.txt`.

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

int main(void)
{
    FILE *fp = fopen("texte.txt", "r");

    if (fp == NULL)
    {
        fprintf(stderr, "Le fichier texte.txt n'a pas pu être ouvert\n");
        return EXIT_FAILURE;
    }

    int ch = fgetc(fp);

    if (ch != EOF)
        printf("%c\n", ch);
    if (fclose(fp) == EOF)
    {
        fprintf(stderr, "Erreur lors de la fermeture du flux\n");
        return EXIT_FAILURE;
    }

    return 0;
}
```

Exemple 5

L'exemple ci-dessous réalise donc la même opération que le code précédent, mais en utilisant la fonction fgets() : une ligne peut contenir jusqu'à 254 caractères (caractère de fin de ligne inclus), on utilise un tableau de 255 caractères pour les contenir (puisqu'il est nécessaire de prévoir un espace pour le caractère nul).

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

int main(void)
{
    char buf[255];
    FILE *fp = fopen("texte.txt", "r");

    if (fp == NULL)
    {
        fprintf(stderr, "Le fichier texte.txt n'a pas pu être ouvert\n");
        return EXIT_FAILURE;
    }
    if (fgets(buf, sizeof buf, fp) != NULL)
```

```
        printf("%s\n", buf);
    if (fclose(fp) == EOF)
    {
        fprintf(stderr, "Erreur lors de la fermeture du flux\n");
        return EXIT_FAILURE;
    }

    return 0;
}
```

Exemple 6

La fonction fwrite() écrit le tableau référencé par ptr composé de nombre éléments de taille multiplets dans le flux flux. Elle retourne une valeur égale à nombre en cas de succès et une valeur inférieure en cas d'échec.
L'exemple suivant écrit le contenu du tableau tab dans le fichier binaire.bin. Dans le cas où un int fait 4 octets, 20 octets seront donc écrits.

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

int main(void)
{
    int tab[5] = { 1, 2, 3, 4, 5 };
    const size_t n = sizeof tab / sizeof tab[0];
    FILE *fp = fopen("binaire.bin", "wb");

    if (fp == NULL)
    {
        fprintf(stderr, "Le fichier binaire.bin n'a pas pu être ouvert\n");
        return EXIT_FAILURE;
    }
    if (fwrite(&tab, sizeof tab[0], n, fp) != n)
    {
            fprintf(stderr, "Erreur lors de l'écriture du tableau\n");
            return EXIT_FAILURE;
    }
    if (fclose(fp) == EOF)
    {
        fprintf(stderr, "Erreur lors de la fermeture du flux\n");
        return EXIT_FAILURE;
    }

    return 0;
}
```

Exemple 7

Lecture du fichier binaire.

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

int main(void)
{
    int tab[5] = { 0 };
    const size_t n = sizeof tab / sizeof tab[0];
    FILE *fp = fopen("binaire.bin", "rb");

    if (fp == NULL)
    {
        fprintf(stderr, "Le fichier binaire.bin n'a pas pu être ouvert\n");
        return EXIT_FAILURE;
    }
    if (fread(&tab, sizeof tab[0], n, fp) != n)
    {
            fprintf(stderr, "Erreur lors de la lecture du tableau\n");
            return EXIT_FAILURE;
```

```
    }
    if (fclose(fp) == EOF)
    {
        fprintf(stderr, "Erreur lors de la fermeture du flux\n");
        return EXIT_FAILURE;
    }

    for (unsigned i = 0; i < n; ++i)
        printf("tab[%u] = %d\n", i, tab[i]);

    return 0;
}
```

9. Exercices

Exercice 1

Créer puis afficher à l'écran le fichier `Fichier_Exo1.txt` dont les informations sont structurées de la manière suivante :

- Numéro de l'élève (entier)
- Nom (chaîne de caractères)
- Prénom (chaîne de caractères)
- Le nombre d'enregistrements à créer est à entrer au clavier par l'utilisateur.

Exercice 2

Ecrire un programme qui crée un fichier `Fichier_Exo2.txt` qui est la copie exacte (enregistrement par enregistrement) du fichier `Fichier_Exo1.txt`.

Exercice 3

Ajouter un nouvel enregistrement (entré au clavier) à la fin de `Fichier_Exo1.txt` et sauvegarder le nouveau fichier sous le nom `Fichier_Exo3.txt`.

Exercice 4

Créer une structure nom, prénom, âge. Ecrire un programme de gestion de fichier (texte) avec menu d'accueil : possibilité de créer le fichier, de le lire, d'y ajouter une fiche, d'en rechercher une.