



Langage C – Cours 7

Lélia Blin

lelia.blin@irif.fr

2023 - 2024

Manipulation de fichiers texte

Modularités du codes

Les fichiers vues comme des *streams*

- On va voir comment manipuler des fichiers pour lire et écrire dedans
- Les accès aux fichiers présentés ici se font via des buffers (mémoire tampon) pour limiter l'accès au disque
- Pour manipuler un fichier on a besoin des informations suivantes :
 - l'adresse du buffer où se trouve le fichier
 - la position de la tête de lecture (à quel endroit on est en train de lire/écrire)
 - le mode d'accès au fichier (lecture/écriture)
- Ces informations sont stockées dans un objet de type **FILE *** défini dans **<stdio.h>**
- Un tel objet est appelé flux de donnée, *stream* en anglais

Ouvrir un fichier

- Pour ouvrir un fichier on utilise la fonction (**dans <stdio.h>**) :
 - **FILE *fopen(const char * restrict path, const char * restrict mode);**
- où path est le chemin du fichier (il peut être relatif ou absolu)
- et mode est le mode d'ouverture du fichier
- Si l'exécution ne se déroule pas correctement la fonction renvoie **NULL**
- Le mode permet de préciser si l'on veut :
 - écrire ou lire (ou les deux dans le fichier)
 - de dire si l'on veut créer ou non le fichier
 - si l'on veut garder ou effacer son contenu
 - ou se fait l'écriture, à la position courante ou à la fin du fichier
- On va également distinguer si le fichier est un **fichier texte** ou un **fichier binaire** (pour lequel ce qu'on lit/écrit n'est pas interprété comme du texte)

Modes pour les fichiers texte

- **"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
- **"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
- Spécificités :
 - Si le mode contient **r**, **le fichier doit exister**
 - Si le mode contient **w**, le fichier peut ne pas exister. Dans ce cas, il sera créé, sinon **son contenu sera effacé**
 - Si le mode contient **a**, le fichier peut ne pas exister. Dans ce cas, il sera créé, sinon **les données seront ajoutées à la fin**

Fermeture du flux

- Pour fermer un flux on utilise la fonction (**dans <stdio.h>**) :
 - **int fclose(FILE *stream);**
- Si la fermeture se déroule correctement la fonction renvoie 0 sinon elle renvoie une valeur différente de 0 en mettant à jour errno
- Que l'appel à cette fonction se passe bien pas, le flux ne peut plus être utilisé après appel à cette fonction
- **Il ne faut pas oublier de fermer les flux ouverts**

Exemple

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

int main(){
    FILE *flux=fopen("fic1.txt","r");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
    }else{
        /*Lecture du fichier*/
        int r=fclose(flux);
        if(r!=0){
            perror("Probleme fermeture de fichier");
        }
    }
}
```

fichier1.c

Si le fichier fic1.txt n'existe pas, on aura un message d'erreur

Exemple

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

int main(){
    FILE *flux=fopen("fic2.txt","w+");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
    }else{
        /*Lecture du fichier*/
        int r=fclose(flux);
        if(r!=0){
            perror("Probleme fermeture de fichier");
        }
    }
}
```

fichier2.c

Si le fichier fic2.txt n'existe pas, le programme le créera

Exemple

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

int main(){
    FILE *flux=fopen("../fic3.txt","w+");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
    }else{
        /*Lecture du fichier*/
        int r=fclose(flux);
        if(r!=0){
            perror("Probleme fermeture de fichier");
        }
    }
}
```

Exemple

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

int main(){
    FILE *flux=fopen("/Users/sangnier/cours/C/Tests/fic4.txt", "w+");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
    }else{
        /*Lecture du fichier*/
        int r=fclose(flux);
        if(r!=0){
            perror("Probleme fermeture de fichier");
        }
    }
}
```

Ce programme peut faire une erreur si il y a un problème dans le chemin (par exemple, les dossiers n'existent pas

Lecture dans un fichier texte

- Pour lire caractère par caractère dans un fichier texte :
 - **int fgetc(FILE *stream);**
 - Cette fonction retourne le caractère lu et elle retourne EOF en cas d'erreur ou si elle atteint la fin du fichier.
- Pour lire une ligne :
 - **char *fgets(char * restrict str, int size, FILE * restrict stream);**
 - Cette fonction lit au plus **size-1** caractères et les met dans la chaîne str
 - La lecture s'arrête si elle rencontre le caractère '\n' avant ou si elle atteint la fin du fichier
 - Le caractère '\n' est mis dans la chaîne str
 - Elle rajoute le caractère '\0' à la fin
 - Elle retourne NULL en cas d'erreur ou si elle atteint la fin du fichier
 - **str doit pointée vers une zone allouée de taille au moins size**

Exemple

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

int main(){
    FILE *flux=fopen("fichier-test1.txt","r");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    int c=0;
    while((c=fgetc(flux))!=EOF){
        printf("Caractère lu %c\n",c);
    }
    int r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
}
```

lecture-fichier.c

Exemple

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

int main(){
    char ch[50];
    FILE *flux=fopen("fichier-test1.txt","r");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    while(fgets(ch,50,flux)!=NULL){
        printf("Lue : %s",ch);
    }
    int r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
}
```

lecture-fichier2.c

Écriture dans un fichier texte

- Pour écrire caractère par caractère dans un fichier texte :
 - **int fputc(int c, FILE *stream);**
 - Cette fonction retourne le caractère écrit et elle retourne EOF en cas d'erreur
- Pour écrire une ligne :
 - **int fputs(const char *restrict s, FILE *restrict stream);**
 - Renvoie une valeur positive en cas de succès et EOF en cas d'échec
 - Ne rajoute pas '\n' (contrairement à puts)
 - Elle n'écrit pas non plus le caractère '\0'
 - **Attention ne pas utiliser cette fonction si l'on veut écrire des '\0'**, cette fonction ne marche donc que pour du texte
- Écriture formatée :
 - **int fprintf(FILE * restrict stream, const char * restrict format, ...);**
 - Même fonctionnement que printf
 - Renvoie le nombre de caractères écrit et une valeur négative en cas d'erreur

Exemple

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

int main(){
    FILE *flux=fopen("fichier-test2.txt","w");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    int r=fputc('a',flux);
    assert(r=='a');
    r=fputc('b',flux);
    assert(r=='b');
    r=fputc('c',flux);
    assert(r=='c');
    r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
}
```

ecriture-fichier.c

Si le fichier n'existe pas, il sera créé. Si on appelle deux fois ce programme, la deuxième fois, le contenu du fichier sera réécrit

Exemple

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

int main(){
    FILE *flux=fopen("fichier-test3.txt", "a");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    int r=fputc('a',flux);
    assert(r=='a');
    r=fputc('b',flux);
    assert(r=='b');
    r=fputc('c',flux);
    assert(r=='c');
    r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
}
```

ecriture-fichier2.c

Si le fichier n'existe pas, il sera créé. Si on appelle deux fois ce programme, la deuxième fois, les écriture seront mis à la suite

Exemple

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

int main(){
    FILE *flux=fopen("fichier-test4.txt","w");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    int r=fputs("Un texte quelconque.",flux);
    assert(r>=0);
    r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
}
```

ecriture-fichier3.c

Si le fichier n'existe pas, il sera créé. Si on appelle deux fois ce programme, la deuxième fois, le contenu du fichier sera réécrit

Exemple

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

int main(){
    FILE *flux=fopen("fichier-test5.txt","w");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    int r=0;
    for(unsigned i=0;i<10;++i){
        r=fprintf(flux,"Ligne %u\n",i);
        assert(r!=EOF);
    }
    r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
    flux=fopen("fichier-test5.txt","w");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    for(unsigned i=0;i<3;++i){
        r=fprintf(flux,"Ligne bis %u\n",i);
        assert(r!=EOF);
    }
    r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
}
```

À la fin le fichier
contient :

Ligne bis 0
Ligne bis 1
Ligne bis 2

ecriture-fichier4.c

Exemple

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

int main(){
    FILE *flux=fopen("fichier-test6.txt", "a");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    int r=0;
    for(unsigned i=0;i<10;++i){
        r=fprintf(flux, "Ligne %u\n", i);
        assert(r!=EOF);
    }
    r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
    flux=fopen("fichier-test6.txt", "a");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    for(unsigned i=0;i<3;++i){
        r=fprintf(flux, "Ligne bis %u\n", i);
        assert(r!=EOF);
    }
    r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
}
```

À la fin le fichier
contient :

Ligne 0
Ligne 1
Ligne 2
Ligne 3
Ligne 4
Ligne 5
Ligne 6
Ligne 7
Ligne 8
Ligne 9
Ligne bis 0
Ligne bis 1
Ligne bis 2

ecriture-fichier4.c

Écriture/lecture dans un fichier

- Si l'on ouvre un fichier en lecture/écriture avec
 - `w+` : le contenu du fichier est effacé (et si il n'existe pas le fichier est créé)
 - `r+` : le contenu du fichier n'est pas effacé et on écrit au début (et le fichier est créé si il n'existe pas)
 - `a+` : le contenu du fichier n'est pas effacé et on écrit à la fin (et le fichier est créé si il n'existe pas)
- À tout moment, on a un curseur dans le flux qui nous indique où l'on va lire et écrire. C'est le même curseur pour les lectures et écriture.
- Le curseur pour `r+` est placé **au début** et pour `w+`, il est placé à la **fin**
- Que se passe-t-il si on mélange les lectures et écritures ? Où va le curseur ?
 - **=> Dangereux sans contrôle**

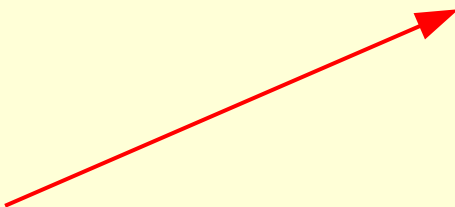
Exemple

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

int main(){
    FILE *flux=fopen("fichier-test8.txt","r+");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    int r=0;
    int c=fgetc(flux);
    if(c!=EOF){
        printf("Lu %c",c);
    }
    r=fputs("Essai\n",flux);
    assert(r!=EOF);
    c=fgetc(flux);
    if(c!=EOF){
        printf("Lu %c",c);
    }

    r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
}
```

**Il n'y a pas de
garantie
où va être fait cette
écriture
-> cela dépend du
buffer**



eclec-fichier.c

Contrôle du curseur

- On peut déplacer le curseur 'manuellement' avec la fonction :
 - **int fseek(FILE *stream, long offset, int whence);**
- La nouvelle position est obtenue en ajoutant **offset** octet à la position **whence**
- **whence peut prendre les valeurs suivantes :**
 - **SEEK_SET** : début du fichier
 - **SEEK_CUR** : position courante du buffer
 - **SEEK_END** : fin du fichier
- Retourne 0 si tout est ok, sinon -1 et met à jour **errno**
- Ainsi **fseek(flux,0,SEEK_SET)** ramène le curseur au début du fichier.
- On ne peut pas aller à une position avant le début du fichier
- Mais on peut avancer dans le fichier !

Exemple

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

int main(){
    FILE *flux=fopen("fichier-test9.txt","w+");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    int r=0;
    r=fputs("Une phrase plus longue 1234455454.\n",flux);
    assert(r!=EOF);
    r=fseek(flux,0,SEEK_SET);
    assert(r==0);
    r=fputs("ABC",flux);
    assert(r!=EOF);
    r=fseek(flux,0,SEEK_END);
    assert(r==0);
    r=fputs("Fin\n",flux);
    r=fseek(flux,0,SEEK_SET);
    assert(r==0);
    int c=fgetc(flux);
    if(c!=EOF){
        printf("Lu %c\n",c);
    }
    r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
}
```

À la fin le fichier contient :

ABC phrase plus longue 1234455454.
Fin

Exemple

```
int main(){
    FILE *flux=fopen("fichier-test10.txt","r");
    if(flux==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    int r=0;
    int cour=0;
    int dec=0;
    int i=0;
    char *st="";
    while((i=fgetc(flux))!=EOF){
        if(i!='\n'){
            ++dec;
        }else{
            r=fseek(flux,cour,SEEK_SET);
            assert(r==0);
            st=malloc(sizeof(char)*(dec+2));
            char *st2=fgets(st,dec+2,flux);
            assert(st2!=NULL);
            printf("%s",st);
            free(st);
            cour=cour+dec+1;
            dec=0;
        }
    }
    r=fclose(flux);
    if(r!=0){
        perror("Probleme fermeture de fichier");
    }
}
```

**Programme qui affiche
les lignes d'un texte
cour indique où
commence une ligne**

Flux ouverts

- Trois flux standards sont disponibles et déjà ouverts :
 - stdin (le clavier)
 - stdout (le terminail)
 - stderr (unité d'affichage des messages d'erreur/par défaut le terminal)
- Il est recommandé d'afficher les messages d'erreur sur stderr
- On peut ainsi faire du code affichant soit dans un fichier soit sur le terminal suivant des options

```
FILE *f=NULL;
if(...){
    f=stdout;
}else{
    f=fopen(...)
}
```

Lecture et écriture de fichiers binaire

- Pour ouvrir le fichier, on rajoute b au mode pour dire qu'il s'agit d'un fichier binaire (exemple de modes : rb, wb, a+b,...)
- On peut aussi lire et écrire des fichiers binaires
 - **size_t fread(void *restrict ptr, size_t size, size_t nitems, FILE *restrict stream);**
- permet de lire nitems objets de taille size et de les mettre dans ptr
 - **size_t fwrite(const void *restrict ptr, size_t size, size_t nitems, FILE *restrict stream);**
- permet d'écrire nitems objets de taille size depuis ptr
- Ces deux fonctions renvoient le nombre d'objets lus/écrits et si il y a un problème elles renvoient 0 (ou un nombre plus petit que le nombre d'objets à lire/écrire donné en argument)

Exemple

```
int main(){
    unsigned tab[10]={1,2,3,4,5,6,7,8,9,10};
    unsigned tab2[10];
    FILE *flout=fopen("fichier-tab.bin","wb");
    if(flout==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    size_t r=fwrite(tab,sizeof(unsigned),10,flout);
    if(r==0){
        perror("Probleme d'ecriture");
        exit(EXIT_FAILURE);
    }
    int r2=fclose(flout);
    if(r2!=0){
        perror("Probleme fermeture de fichier flout");
        exit(EXIT_FAILURE);
    }
    FILE *flin=fopen("fichier-tab.bin","rb");
    if(flin==NULL){
        perror("Probleme ouverture de fichier");
        exit(EXIT_FAILURE);
    }
    r=fread(tab2,sizeof(unsigned),10,flin);
    if(r==0){
        perror("Probleme de lectures");
        exit(EXIT_FAILURE);
    }
    r2=fclose(flin);
    if(r2!=0){
        perror("Probleme fermeture de fichier flin");
        exit(EXIT_FAILURE);
    }
    printf("tab2 : [");
    for(unsigned *t=tab2;t<tab2+10;++t){
        printf("%u ",*t);
    }
    printf("]\n");
}
```

Avoir plusieurs fichiers

- Pour organiser son code, il peut être utile de séparer son code dans plusieurs fichiers
- Par exemple un fichier avec la description des fonctions manipulant les listes et un fichier avec le programme principal
- Comment faire cela ?
- Comment indiquer où sont les fonctions à utiliser ?
- Comment compiler des fichiers qui ne sont pas des exécutables
- On va parler de modules en C

Fichier .c et .h et .o

- Par exemple, supposons que dans un fichier, on veut mettre toutes les fonctions liées aux listes simplement chaînées
- Pour cela on va créer deux fichiers :
 - liste.h : il contient la description du type de liste ainsi que les prototypes de chacune des fonctions, chaque prototype va être précédé du mot clef extern (pour dire que la définition de la fonction est dans un autre fichier)
 - liste.c : il va contenir les différentes descriptions des fonctions mais pas de main
- Dans liste.c on va aussi dire où trouver le type liste et les prototypes des fonctions en faisant au début **#include "liste.h"**
- **À la compilation cet include est remplacé par le contenu du fichier .h**
- Règle : si on utilise le même nom dans le .h que dans le .c correspondant
- Pour compiler liste.c, on fait
 - **gcc -Wall -c liste.c**
 - Produit un fichier objet liste.o non exécutable
- On ne fait pas de .h pour le fichier contenant le main, et il doit inclure tous les .h des fichiers qu'il utilise

Exemple

```
struct noeud {  
    int val;  
    struct noeud *succ;  
};  
  
typedef struct noeud noeud;  
  
extern noeud *creation_noeud(int);  
extern noeud *insertion_tete(noeud *,int);  
extern noeud *insertion_queue(noeud *,int);  
extern void print_list(noeud *);  
extern noeud *efface_tete(noeud *);  
extern noeud *efface_queue(noeud *);  
extern noeud *efface_val(noeud *,int);  
extern noeud *reverse(noeud *);
```

liste.h

Exemple

```
#include "liste.h"
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

noeud *creation_noeud(int v){
    noeud *n=malloc(sizeof(noeud));
    n->val=v;
    n->succ=NULL;
    return n;
}

noeud *insertion_tete(noeud *li,int v){
    noeud *n=malloc(sizeof(noeud));
    n->val=v;
    n->succ=li;
    return n;
}

....
```

liste.c

Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "liste.h"

int main(){
    noeud *li=insertion_tete(NULL,1);
    li=insertion_tete(li,2);
    li=insertion_tete(li,3);
    li=insertion_tete(li,4);
    print_list(li);
    li=insertion_queue(li,10);
    li=insertion_queue(li,11);
    print_list(li);
    li=efface_queue(li);
    print_list(li);
    li=efface_val(li,2);
    li=efface_val(li,15);
    print_list(li);
    li=efface_val(li,4);
    print_list(li);
    li=efface_queue(li);
    print_list(li);
    li=reverse(li);
    print_list(li);
}
```

prog.c

Exemple

- Pour compiler le code précédent on va faire :
 - **gcc -Wall -c liste.c**
 - => crée le fichier liste.o
 - **gcc -Wall -c prog.c**
 - => crée le fichier prog.o
 - **gcc -Wall -o prog liste.o prog.o**
 - => fait le lien entre les différents fichiers objets et génère l'exécutable
- **Attention** : quand on crée l'exécutable, **il ne faut qu'il n'y ait une seule fonction main dans le différents fichiers .o**
 - si on met un main dans liste.c, on a une erreur

Éviter les inclusions multiples

- Supposons que l'on ait liste.h

```
struct noeud {
    int val;
    struct noeud *succ;
};

typedef struct noeud noeud;

extern noeud *creation_noeud(int);
extern noeud *insertion_tete(noeud *,int);
extern noeud *insertion_queue(noeud *,int);
extern void print_list(noeud *);
extern noeud *efface_tete(noeud *);
extern noeud *efface_queue(noeud *);
extern noeud *efface_val(noeud *,int);
extern noeud *reverse(noeud *);
```

- Et l'on veuille mettre dans un fichier listtab.h la fonction qui crée une liste a partir d'un tableau

```
extern noeud *create_liste(int *);
```

- Le problème est que si l'on inclut que listtab.h ensuite, le programme ne connaît pas liste, il faut donc ajouter include liste.h à listtab
- Mais si un programme utilise liste.h et listtab.h, liste.h sera inclus deux fois

Éviter les inclusions multiples

```
#ifndef LISTE_H
#define LISTE_H
struct noeud {
    int val;
    struct noeud *succ;
};

typedef struct noeud noeud;

extern noeud *creation_noeud(int);
extern noeud *insertion_tete(noeud *,int);
extern noeud *insertion_queue(noeud *,int);
extern void print_list(noeud *);
extern noeud *efface_tete(noeud *);
extern noeud *efface_queue(noeud *);
extern noeud *efface_val(noeud *,int);
extern noeud *reverse(noeud *);
#endif
```

liste.h

LISTE_H est une macro qui sert à la précompilation, en gros on dit si la macro n'est pas définie alors on la définit et on copie le texte jusqu'à endif sinon on ne fait rien

Les prototypes ne seront copiés qu'une fois en cas de plusieurs include de liste.h qui s'enchaînent

Bien structurer son code

- Une règle importante :
 - **Dans un programme il ne peut pas y avoir deux fonctions avec le même nom même si les prototypes sont différents**
 - Ainsi on ne peut pas avoir dans un fichier fic1.c une fonction `int f()` et dans un fichier fic2.c une fonction `double f(int x)`
 - Si l'on fait `gcc -Wall -o prog fic1.o fic2.o`, le compilateur ne saura pas faire le lien correctement
- **Il est important de bien structurer son code et aussi de se rappeler de recompiler les fichiers .o dont le .c a été changé !!!!**