

université de science et technologie houari boumediene



TP2: FILES in C



beldjouziasmaa@gmail.com

Types de fichiers en C

En langage C, il existe deux types de fichiers :

1. Fichiers texte
2. Fichiers binaires

1. Fichiers texte

Les fichiers texte sont des fichiers contenant des caractères lisibles par l'homme, souvent encodés en ASCII (American Standard Code for Information Interchange).

Avantages :

- Peuvent être lus et modifiés facilement avec un éditeur de texte.
- Portables, c'est-à-dire qu'ils peuvent être utilisés sur différentes machines sans problème de compatibilité.

Inconvénients :

- Moins efficaces car une conversion est nécessaire avant d'être interprétés par un programme.
- Occupent plus d'espace qu'un fichier binaire équivalent.

2. Fichiers binaires

Les fichiers binaires contiennent les données dans le format interne utilisé par la machine.

Avantages :

- Occupent moins d'espace que les fichiers texte.
- Plus rapides à lire et écrire car aucune conversion n'est requise.

Inconvénients :

- Ne peuvent pas être lus directement par un humain.
- Moins portables, car leur format dépend de l'architecture de la machine qui les a générés.

Déclaration et utilisation des fichiers en C

Avant d'utiliser un fichier en C, il faut déclarer une variable de type FILE *.

A dark-themed code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. The code inside is 'FILE *nomFichier;'.

```
FILE *nomFichier;
```

Exemple :

A dark-themed code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. The code inside is 'FILE *pFile; // Déclaration d'un pointeur vers un fichier'.

```
FILE *pFile; // Déclaration d'un pointeur vers un fichier
```

Opérations sur les fichiers (Texte et Binaire)

Ouverture d'un fichier

En C, un fichier est ouvert à l'aide de la fonction fopen :

```
FILE *fopen(const char *filename, const char *mode);
```

Utilisation :

```
<variableFichier> = fopen(<nomFichier>, <mode>);
```

- variableFichier : Pointeur de type FILE * représentant le fichier.
- nomFichier : Chaîne de caractères contenant le nom du fichier.
- mode : Mode d'ouverture du fichier (lecture, écriture, ajout, etc.).

⚠ Si fopen échoue, elle retourne NULL. Il est donc important de tester la valeur de retour pour éviter des erreurs.

Remarque sur le chemin du fichier :

- Si un chemin absolu est donné (ex. "c:/putty/etud.txt"), le fichier sera créé ou lu dans ce dossier.
- Si seul le nom du fichier est donné (ex. "etud.txt"), il sera créé ou lu dans le répertoire courant du programme.



```
FILE *pFile;  
pFile = fopen("Data.txt", "rt"); // Ouvre "Data.txt" en mode lecture (texte)
```

Opérations sur les fichiers (Texte et Binaire)

Modes d'ouverture des fichiers

Tableau des modes d'ouverture :

Mode	Signification	Comportement
r	Lecture seule	Le fichier doit exister, sinon fopen() retourne NULL.
w	Écriture seule	Écrase le fichier s'il existe, sinon le crée.
a	Ajout	Ajoute les données à la fin du fichier, le crée s'il n'existe pas.
r+	Lecture et écriture	Le fichier doit exister, sinon fopen() retourne NULL.
w+	Lecture et écriture	Écrase le fichier s'il existe, sinon le crée.
a+	Lecture et ajout	Ajoute des données à la fin, le crée s'il n'existe pas.

Opérations sur les fichiers (Texte et Binaire)

Modes d'ouverture des fichiers

Fichier texte ou binaire ?

On peut ajouter "t" pour un fichier texte (optionnel, par défaut) ou "b" pour un fichier binaire.

Exemples :

- "rb" : Ouverture d'un fichier binaire en lecture seule.
- "wb" : Ouverture d'un fichier binaire en écriture seule.

Opérations sur les fichiers (Texte et Binaire)

Fermeture d'un fichierFichier texte ou binaire ?

- Une fois un fichier utilisé, il faut le fermer pour libérer les ressources associées.

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

- Renvoie 0 si la fermeture est réussie, une valeur différente de 0 en cas d'erreur.
- **Exemple :**

```
FILE *pFile = fopen("Data.txt", "rt");  
if (pFile != NULL) {  
    fclose(pFile); // Fermeture du fichier  
}
```

Opérations sur les fichiers (Texte et Binaire)

- **Tester la fin de fichier (EOF – End Of File)**

Pour vérifier si l'on a atteint la fin d'un fichier, on utilise la fonction `feof()`.

```
int feof(FILE *pFile);
```

- **Retourne 0 tant que la fin du fichier n'est pas atteinte.**
- **Retourne une valeur différente de 0 lorsque la fin du fichier est atteinte après une tentative de lecture.**
- **⚠ Attention ! En C, le test EOF ne fonctionne correctement qu'après une tentative de lecture.**
- **Exemple :**

```
FILE *pFile = fopen("Data.txt", "r");
if (pFile != NULL) {
    char c;
    while ((c = fgetc(pFile)) != EOF) { // Lire caractère par caractère
        putchar(c); // Affiche le caractère
    }
    fclose(pFile); // Fermeture du fichier
}
```

Opérations sur les fichiers texte

Opérations d'écriture

En C, plusieurs fonctions permettent d'écrire dans un fichier texte :

Fonction	Action
fprintf()	Écriture formatée dans un fichier
fputc()	Écriture d'un caractère dans un fichier
fputs()	Écriture d'une chaîne de caractères dans un fichier

Opérations sur les fichiers texte

1 fprintf() – Écriture formatée

Cette fonction fonctionne comme printf(), mais écrit dans un fichier au lieu de la console.

```
int fprintf(FILE *stream, const char *format, ...);
```

Exemple :

```
FILE *pFile = fopen("Data.txt", "w");  
if (pFile != NULL) {  
    fprintf(pFile, "Nom: %s, Age: %d\n", "Alice", 25);  
    fclose(pFile);  
}
```

Opérations sur les fichiers texte

2 fputc() - Écriture d'un caractère

Utilisée pour écrire un seul caractère dans un fichier.

```
int fputc(int c, FILE *stream);
```

Exemple :

```
FILE *pFile = fopen("Data.txt", "w");  
if (pFile != NULL) {  
    fputc('A', pFile);  
    fputc('\n', pFile);  
    fclose(pFile);  
}
```

Opérations sur les fichiers texte

3 fputs() - Écriture d'une chaîne

Permet d'écrire une chaîne de caractères dans un fichier.

```
int fputs(const char *s, FILE *stream);
```

Exemple :

```
FILE *pFile = fopen("Data.txt", "w");  
if (pFile != NULL) {  
    fputs("Bonjour tout le monde !\n", pFile);  
    fclose(pFile);  
}
```

Opérations sur les fichiers texte

Opérations de lecture

En C, plusieurs fonctions permettent de lire des données d'un fichier texte :

Fonction	Action
fscanf()	Lecture formatée
fgetc()	Lecture d'un caractère
fgets()	Lecture d'une chaîne de caractères

Opérations sur les fichiers texte

1 fscanf() – Lecture formatée

Fonction similaire à scanf(), mais utilisée pour lire des données depuis un fichier.

```
int fscanf(FILE *stream, const char *format, ...);
```

Exemple :

```
FILE *pFile = fopen("Data.txt", "r");
char nom[50];
int age;
if (pFile != NULL) {
    fscanf(pFile, "Nom: %s, Age: %d", nom, &age);
    printf("Nom: %s, Age: %d\n", nom, age);
    fclose(pFile);
}
```

Opérations sur les fichiers texte

2 fgetc() – Lecture d'un caractère

Utilisée pour lire un seul caractère d'un fichier.

```
int fgetc(FILE *stream);
```

Exemple :

```
FILE *pFile = fopen("Data.txt", "r");
char c;
if (pFile != NULL) {
    while ((c = fgetc(pFile)) != EOF) {
        putchar(c);
    }
    fclose(pFile);
}
```

Opérations sur les fichiers texte

3 **fgets()** – Lecture d'une chaîne

Lit une ligne entière depuis un fichier.

```
char *fgets(char *s, int n, FILE *stream);
```

- Lit jusqu'à (n-1) caractères ou jusqu'à une nouvelle ligne (\n).
- Ajoute automatiquement \0 à la fin de la chaîne lue.
- Retourne NULL en cas d'erreur ou si la fin du fichier est atteinte sans lecture.
- **Exemple :**

```
FILE *pFile = fopen("Data.txt", "r");
char buffer[100];
if (pFile != NULL) {
    while (fgets(buffer, sizeof(buffer), pFile) != NULL) {
        printf("%s", buffer);
    }
    fclose(pFile);
}
```

Opérations sur les fichiers texte

Écrire deux programmes en langage C :

1. Un programme permettant de créer un fichier texte contenant les informations des étudiants.
2. Un programme permettant de lire et d'afficher les informations enregistrées dans le fichier texte.

Opérations sur les fichiers texte

- **Création du fichier etud.txt**

- Demandez à l'utilisateur combien d'étudiants il souhaite enregistrer.
- Pour chaque étudiant, saisissez les informations suivantes :
 - Matricule (nombre entier long)
 - Nom (chaîne de 10 caractères max)
 - Prénom (chaîne de 10 caractères max)
 - Moyenne générale (nombre flottant)
- Enregistrez chaque étudiant sur une ligne du fichier sous la forme :

1001 beldjouzi asma 12.75

1002 beldjouzi bouchra 15.50


- Assurez-vous que le fichier est bien créé et affichez un message de confirmation.

- **Lecture du fichier etud.txt**

- Ouvrez le fichier en mode lecture.
- Lisez ligne par ligne les informations des étudiants.
- Affichez les informations sous la forme :
 - 1001 beldjouzi asma 12.75
 - 1002 beldjouzi bouchra 15.50
- **Gérez les erreurs si le fichier n'existe pas.**


Opérations sur les Fichiers Binaires

- **L'opérateur sizeof()**
- L'opérateur sizeof() permet de déterminer la taille en octets d'une variable ou d'un type de donnée.



```
sizeof(<var>) // Retourne la taille de la variable <var>  
sizeof(<type>) // Retourne la taille du type <type>
```

- **Exemple :**



```
int x, sz;  
sz = sizeof(x); // sz = 4 octets sous Dev-C++  
sz = sizeof(int); // sz = 4 octets sous Dev-C++
```

Opérations sur les Fichiers Binaires

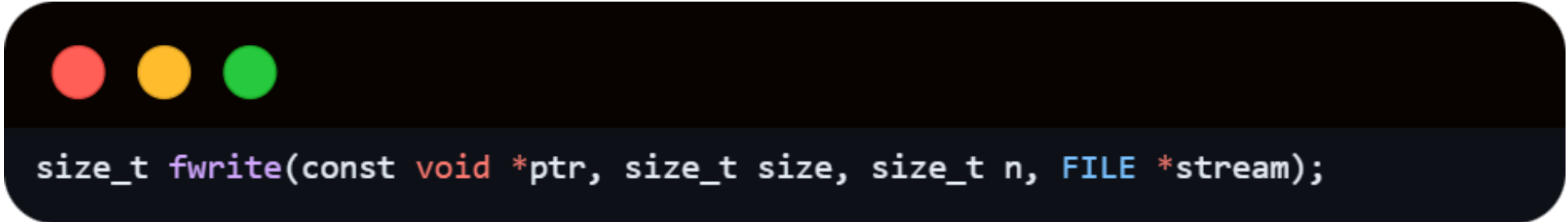
- **Opérations de Lecture**
- Fonction `fread()` – Lecture non formatée d'un fichier

```
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
```

- **Explication des paramètres :**
 - `ptr` : Adresse où stocker les données lues.
 - `size` : Taille d'un élément à lire (en octets).
 - `n` : Nombre d'éléments à lire.
 - `stream` : Pointeur vers le fichier ouvert en lecture binaire.
- **Comportement :**
 - `fread()` lit `n` éléments de `size` octets depuis `stream` et stocke les données à l'adresse `ptr`.
 - Le nombre total d'octets lus est `n * size`.
- **Valeur de retour :**
 - Nombre d'éléments lus avec succès.
 - Si ce nombre est inférieur à `n`, cela signifie qu'il y a eu une erreur ou que la fin du fichier a été atteinte.

Opérations sur les Fichiers Binaires

- **Opérations d'Écriture**
- Fonction `fwrite()` – Écriture non formatée dans un fichier



```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);
```

- **Explication des paramètres :**
- `ptr` : Adresse des données à écrire.
- `size` : Taille d'un élément à écrire (en octets).
- `n` : Nombre d'éléments à écrire.
- `stream` : Pointeur vers le fichier ouvert en écriture binaire.
- **Comportement :**
- `fwrite()` écrit `n` éléments de `size` octets dans `stream` à partir de l'adresse `ptr`.
- Le nombre total d'octets écrits est `n * size`.
- **Valeur de retour :**
- Nombre d'éléments écrits avec succès.
- Si ce nombre est inférieur à `n`, une erreur s'est produite lors de l'écriture.

Opérations sur les Fichiers Binaires

- **Création d'un fichier binaire**
- Écrivez un programme permettant de créer un fichier binaire etud.dat contenant les informations de plusieurs étudiants (numéro, nom, prénom, moyenne).
- L'utilisateur doit entrer le nombre d'étudiants, puis saisir leurs informations.
- Les informations doivent être enregistrées sous forme binaire.
- **Lecture d'un fichier binaire**
- Écrivez un programme qui lit le fichier etud.dat et affiche son contenu à l'écran.
- Conversion d'un fichier texte en fichier binaire
- Un fichier texte etd.txt contient des informations sur les étudiants (une ligne par étudiant).
- Écrivez un programme qui lit ce fichier texte et crée un fichier binaire etd.bin avec les mêmes informations.
- **Lecture du fichier binaire converti**
- Écrivez un programme qui lit le fichier etd.bin et affiche son contenu à l'écran.

Comment Tester l'Existence d'un Fichier en C

- La fonction `access()` qui fait partie de `<unistd.h>` permet de tester si un fichier existe ou non.
- Cette fonction doit être appelée ainsi :
- `access(NomPhysiqueFichier, F_OK)`

Exemple : Création d'un fichier si celui-ci n'existe pas

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

int main() {
    if (access("etd.dat", F_OK) != 0) {
        // Ouvrir le fichier "etd.dat" en écriture binaire et le créer s'il n'existe pas
        FILE *pFile = fopen("etd.dat", "wb");
        if (pFile != NULL) {
            printf("Fichier etd.dat créé avec succès.\n");
            fclose(pFile);
        } else {
            printf("Erreur lors de la création du fichier.\n");
        }
    } else {
        printf("Le fichier etd.dat existe déjà.\n");
    }
    return 0;
}
```

Remarque importante sur les fichiers en mode mise à jour

- Lorsqu'un fichier est ouvert en mode mise à jour ("r+b", "w+b", "a+b" pour les fichiers binaires), les règles suivantes doivent être respectées :
- Si une opération de lecture (fread) suit une opération d'écriture (fwrite), il faut soit :
- Vider le tampon avec fflush().
- Repositionner le fichier avec fseek() ou fsetpos().
- Si une opération d'écriture (fwrite) suit une opération de lecture (fread), il faut repositionner le fichier avec fseek() ou fsetpos(), sauf si l'opération de lecture a atteint la fin du fichier.

Si ces conditions ne sont pas respectées, le résultat peut être imprévisible.

Exemple : Utilisation correcte de `fflush()` et `fseek()`

```
#include <stdio.h>

int main() {
    FILE *file = fopen("test.dat", "w+b");
    if (file == NULL) {
        printf("Erreur d'ouverture du fichier.\n");
        return 1;
    }

    int number = 12345;
    fwrite(&number, sizeof(int), 1, file);
    fflush(file); // Vide le tampon avant de lire

    fseek(file, 0, SEEK_SET); // Repositionne le fichier au début

    int read_number;
    fread(&read_number, sizeof(int), 1, file);
    printf("Nombre lu : %d\n", read_number);

    fclose(file);
    return 0;
}
```

Différences entre printf, fprintf, sprintf et scanf, fscanf, sscanf

- Fonctions d'affichage et d'écriture :

Fonction	Description
printf	Affiche une sortie formatée sur l'écran.
sprintf	Écrit une chaîne formatée dans un buffer (tableau de char).
fprintf	Écrit une chaîne formatée dans un fichier texte.

- Fonctions d'affichage et d'écriture :

Fonction	Description
scanf	Lit une entrée formatée depuis l'entrée standard (clavier).
fscanf	Lit une entrée formatée depuis un fichier texte.
sscanf	Lit une entrée formatée depuis une chaîne de caractères.