ECE PARIS • LYON
ÉCOLE D'INGÉNIEURS

Algorithmique et programmation structurée en C

Cours n°6

Langage C
Fichiers textes/binaires

Antoine Hintzy

 Plan du cours

- Mémoire centrale / disque
- Fichiers textes vs binaires
- Ouverture/fermeture d'un fichier
- Lecture/écriture dans un fichier
- Positionnement dans un fichier binaire

Mémoire centrale / disque

- Dans un programme, les variables sont stockées en mémoire centrale. Elles ont une durée de vie limitée.

- Pour que les données soient stockées sur le long terme, il faut les stocker dans un fichier, sur le disque dur / SSD.

Un fichier est une série de données stockée sur un disque dur / SSD ou sur un périphérique de stockage (clé USB...).

On distingue **deux opérations** possibles :

- **Lecture d'un fichier**

  Transfert de données du fichier vers la mémoire centrale.

  Semblable au **`scanf`** qui transfère des données du clavier vers la mémoire centrale.

- **Écriture dans un fichier**

  Transfert de données de la mémoire centrale vers le fichier.

  Semblable au **`printf`** qui transfère des données du clavier vers la mémoire centrale.

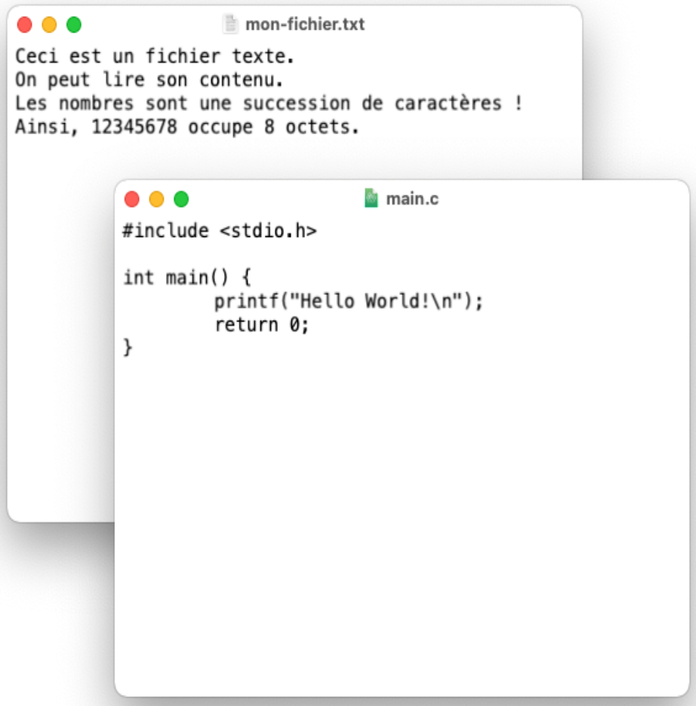
Et **deux types de fichiers** : les fichiers **textes** et les fichiers **binaires**.

Fichiers textes

Les fichiers textes contiennent une succession de caractères et peuvent être lus par un humain, via un éditeur de texte.

⚠️ Les chiffres et les nombres sont également représentés par des caractères.

(`24` occupe 2 octets, `12345678` occupe 8 octets).



Exemples :

monfichier.txt, index.html, style.css, README.md, main.c...

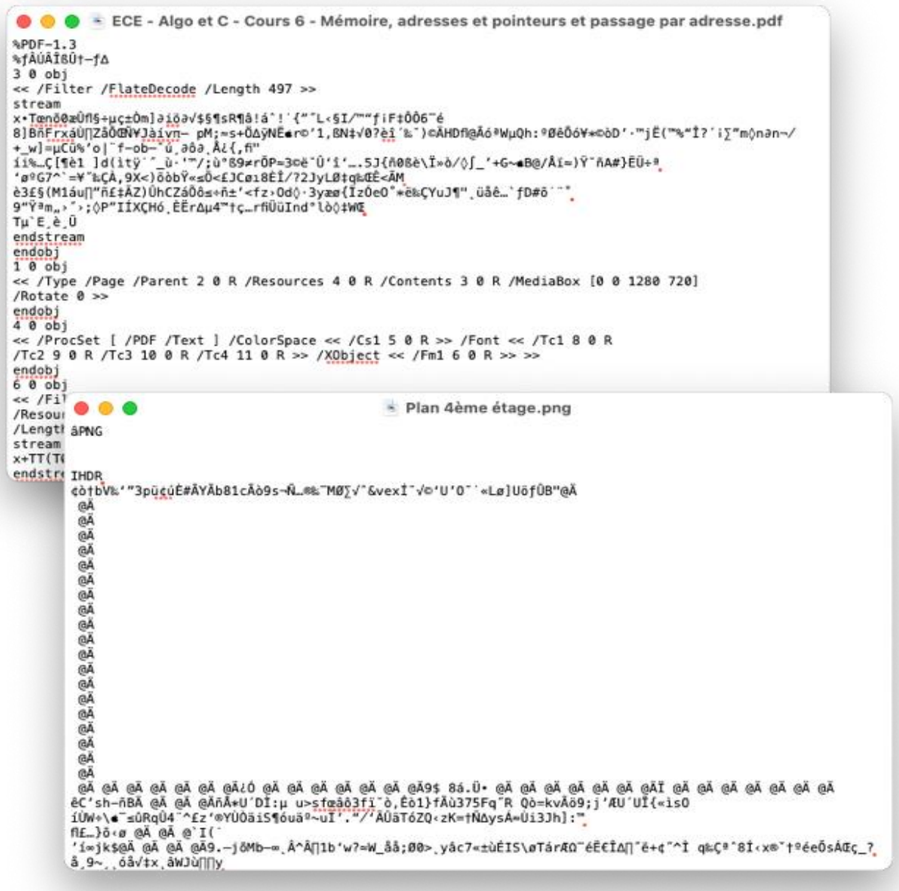
Fichiers binaires

Les fichiers binaires contiennent une succession d'octets pouvant représenter tout ce que l'on peut stocker en mémoire centrale :
des entiers, des réels, des caractères, des structures, des tableaux...

Ils s'ouvrent dans une application spécifique, généralement celle qui les a créés. Sauf s'il s'agit de fichiers standardisés (PNG, JPG, PDF...).

Exemples : rapport.pdf, cours.docx, photo.jpg, sauvegarde.ece...

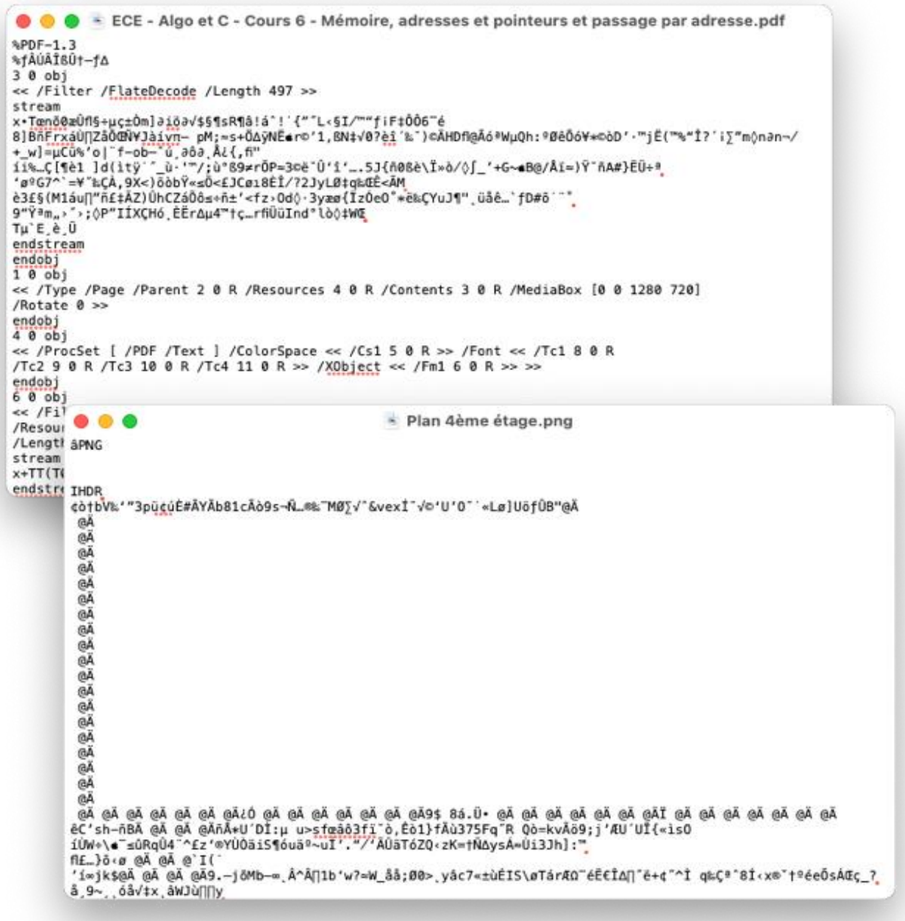
NB : Nous pouvons inventer notre propre extension de fichier binaire. Mais fichiers binaires standardisés portent une extension elle-même standardisée : .pdf, .png...



Fichiers binaires

Si on ouvre un fichier binaire avec un éditeur de texte, ce dernier interprétera chaque octet comme un caractère. Ce qui donne généralement un résultat illisible (sauf pour les chaînes de caractères).

Un entier de 4 octets sera affiché comme étant 4 caractères.



Les variables sont stockées dans les fichiers binaires de la même façon qu'elles le sont en mémoire centrale. Les nombres sont donc des nombres, et non pas une suite de caractères chiffres.

Les grands nombres occupent donc moins de place dans un fichier binaire (ex : `1000000` = 7 octets en fichier texte, contre 4 octets dans un fichier binaire (`int`)).

Ouverture / fermeture d'un fichier

Pour pouvoir manipuler un fichier, nous avons besoin de la bibliothèque standard d'entrées/sorties (**`stdio.h`**), qui contient, entre autres, le type pointeur sur fichier **`FILE*`** ainsi que les fonctions de manipulation de fichiers.

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

Nous pouvons ensuite déclarer des pointeurs sur fichier, permettant de désigner un fichier à manipuler :

```
FILE* pf = NULL; /* initialisation (ne pointe encore sur rien) */
```

Ouverture d'un fichier

Pour pouvoir manipuler un fichier, il faut lier un pointeur sur fichier (**`FILE*`**) à celui-ci. On dit alors qu'on **ouvre le fichier**. Nous utilisons pour cela la fonction **`fopen`** qui prend en paramètres le **chemin** vers le fichier à ouvrir, suivi d'une chaîne de caractères indiquant le mode d'ouverture (les droits accordés sur le fichier).

```
FILE* pf = NULL ;
pf = fopen(NULL"./monfichier.txt";, "r"); // ouvre le fichier monfichier.txt se trouvant à côté de l'exécutable
```

Le pointeur sur fichier **`pf`** est maintenant lié au fichier "**`monfichier.txt`**".

NB : si l'ouverture du fichier échoue (fichier introuvable...), alors la fonction renvoie **`NULL`**. Il est donc important de vérifier que le pointeur n'est pas nul avant de le manipuler.

Chemins sur le disque/SSD

Chemins relatifs

- Dans le dossier parent : `../monfichier.txt`
- Dans le même dossier : `./monfichier.txt`
- Dans le dossier enfant "`fichiers`" : `./fichiers/monfichier.txt`

Chemins absolus

- Sur le bureau (Windows) : `C:\\Users\\ahintzy\\Desktop\\monfichier.txt`
- Sur le bureau (MacOS) : `/Users/ahintzy/Desktop/monfichier.txt`

Modes d'ouverture de fichier

Mode	Description
`"r"`	**Lecture seule** Le fichier est ouvert à son début, prêt à lire les données. Toute tentative d'écriture provoque une erreur de segmentation.
`"w"`	**Écriture seule** Le fichier est ouvert à son début, prêt pour l'écriture de données. Toute tentative de lecture provoque une erreur de segmentation. Le fichier est écrasé s'il existait.
`"a"`	**Ajout** Le fichier est ouvert à la suite de son contenu existant, prêt pour l'ajout de données. Toute tentative de lecture provoque une erreur de segmentation. Le fichier n'est pas écrasé.

Modes d'ouverture de fichier

Mode	Description
`"r+"`	**Lecture/Écriture** Le fichier est ouvert à son début, prêt pour lire et écrire. Le fichier n'est pas écrasé.
`"w+"`	**Lecture/Écriture** Le fichier est ouvert à son début, prêt pour lire et écrire. Le fichier est écrasé s'il existait.
`"a+"`	**Lecture/Écriture** Le fichier est ouvert à la suite de son contenu, prêt pour lire ou écrire. Le fichier n'est pas écrasé.

Fermeture d'un fichier

Lorsque nous avons fini d'utiliser un fichier, nous devons le fermer en utilisant la fonction **`fclose`**. Celle-ci prend en paramètre le pointeur sur fichier correspondant.

```
fclose(pf); // le pointeur contient encore l'adresse après
pf = NULL;  // il faut donc le remettre à NULL
```

Écriture dans un fichier texte

Pour écrire dans un fichier texte, nous pouvons utiliser la fonction **`fprintf`**, semblable à la fonction **`printf`**.

```c
fprintf(pf, "%d", a);
```

`fprintf` prend comme paramètres : un pointeur sur le fichier dans lequel on veut écrire, une chaîne de format (texte accompagné de **`%d`**, **`%c`**...), puis la ou les variables à injecter dans les **`%`**... Elle retourne le nombre de caractères écrits dans le fichier, ou un nombre négatif en cas d'erreur.

NB : l'écriture reprend là où elle s'était arrêtée. En effet, le pointeur avance dans le fichier au fur et à mesure où l'on écrit dedans.

Écriture dans un fichier texte - exemple

```c
#include <stdio.h>
int main() {
    int i = 0;
    FILE* pf = fopen("../monFichier.txt", "w");
    if (pf == NULL) {
        printf("Erreur d'ouverture de fichier.");
        return 1;
    }
    for(i = 0; i < 5; i++) {
        fprintf(pf, "%d ", i*i);
    }
    fclose(pf);
    pf = NULL;
    return 0;

}
```

Écriture dans un fichier texte - exemple, avec affichage

```c
#include <stdio.h>
int main() {
    int i = 0;
    FILE* pf = fopen("../monFichier.txt", "w");
    if (pf == NULL) {
        printf("Erreur d'ouverture de fichier.");
        return 1;
    }
    for (i = 0; i < 5; i++) {
        printf("%d caractères écrits.\n", fprintf(pf, "%d ", i*i));
    }
    fclose(pf);
    pf = NULL;
    return 0;
}
```

Lecture dans un fichier texte

Pour lire dans un fichier texte, nous pouvons utiliser la fonction **`fscanf`**, semblable à la fonction **`scanf`**.

```
fscanf(pf, "%d", &a);
```

`fscanf` prend comme paramètres : un pointeur sur le fichier depuis lequel on veut lire, une chaîne de format (**`%d`**, **`%c`**...), puis la ou les adresses des variables qui recevront la/ les valeur(s) lue(s).
Elle retourne le nombre de valeurs qui ont pu être lues dans le fichier.

NB : la lecture reprend là où elle s'était arrêtée. En effet, le pointeur avance dans le fichier au fur et à mesure où l'on lit dedans.

Lecture dans un fichier texte - exemple

```c
#include <stdio.h>
#define MAX 3
int main() {
    int valeursLues[MAX], i = 0, j = 0;
    FILE* pf = fopen("../monFichier.txt", "r");
    if (pf == NULL) {
        printf("Erreur d'ouverture de fichier.");
        return 1;
    }
    // Dans la boucle suivante, l'ordre des opérandes du && est important car si le premier (i < MAX) est
    // faux, le deuxième (fscanf) n'est pas exécuté/évalué.
    // S'il l'était alors que i >= MAX, cela causerait un dépassement mémoire du tableau (tab[3] n'existe pas).
    while (i < MAX && fscanf(pf, "%d", &valeursLues[i]) == 1) {
        i += 1; // on lit une valeur supplémentaire donc +1
    }
    fclose(pf);
    pf = NULL;
    for(j = 0; j < i; j++) {
        printf("%d", valeursLues[j]);
        printf(j < i-1 ? ", " : ".");
    }
    return 0;
}
```

Lecture/écriture dans un fichier binaire

Avec les fichiers binaires, nous copions/collons des séquences d'octets (**blocs mémoire**) depuis la mémoire centrale vers le fichier.

Avant de sauvegarder dans un fichier binaire, il faut savoir quoi stocker et dans quel ordre.

En effet, une fois les octets copiés dans le fichier, il est très difficile de savoir ce qu'ils représentent. Par exemple, si on prend les 4 premiers octets d'un fichier, il est difficile, voire impossible, de savoir s'il s'agit d'un `int`, de deux `short`, ou de 4 `char`.

Ainsi, lorsque nous sauvegardons un tableau par exemple, nous devons d'abord sauvegarder sa taille logique. Au chargement, nous pourrons alors calculer le nombre d'octets à lire pour charger le tableau. **L'ordre des éléments dans le fichier doit être clair et réfléchi.**

Écriture dans un fichier binaire

Pour écrire dans un fichier binaire, nous pouvons utiliser la fonction **`fwrite`** pour effectuer une copie de **blocs mémoire** (octets) depuis la mémoire centrale vers un fichier.

```
int a = 5;
int tab[3] = {4, 6, 12};
fwrite(&a, sizeof(int), 1, pf); // on copie 1*sizeof(int) donc 4 octets
                                // à partir de l'adresse &a dans le fichier pointé par pf
fwrite(tab, sizeof(int), 3, pf); // on copie 3*sizeof(int) donc 12 octets à partir
// de l'adresse contenue dans le pointeur tab
```

`fwrite` prend comme paramètres : l'adresse dans la mémoire centrale du premier bloc à copier dans le fichier, la taille de chaque bloc mémoire à copier (ils ont la même taille), le nombre de blocs à copier et un pointeur sur le fichier dans lequel copier.
Elle retourne le nombre de blocs qui ont pu être écrits dans le fichier.

Écriture dans un fichier binaire - exemple

```c
#include <stdio.h>
int main(void) {
    const int nbNumbers = 5;
    float numbers[nbNumbers] = {3.43, 6.354, 5.6, 123.23, 64.2};
    FILE* pf = NULL;
    if ((pf = fopen("../monFichier.dat", "w")) == NULL) {
        printf("Erreur d'ouverture du fichier.\n");
        return 0;
    }
    fwrite(&nbNumbers, sizeof(int), 1, pf); // on sauvegarde la taille du tableau
    // puis le tableau
    if (fwrite(numbers, sizeof(float), nbNumbers, pf) != nbNumbers) {
        printf("Problème d'écriture dans le fichier.\n");
    }
    fclose(pf);
    pf = NULL;
    return 0;
}
```

Lecture dans un fichier binaire

Pour lire dans un fichier binaire, nous pouvons utiliser la fonction **`fread`** pour effectuer une copie de **blocs mémoire** (octets) depuis le fichier vers la mémoire centrale.

```
int a; int
tab[3];
fread(&a, sizeof(int), 1, pf); // on copie les 1*sizeof(int) = 4 prochains octets du fichier
                               // pointé par pf à partir de l'adresse &a de la mémoire centrale
fread(tab, sizeof(int), 3, pf); // on copie les 3*sizeof(int) = 12 prochains octets
                              // du fichier pointé par pf à l'adresse tab de la mémoire centrale
```

`fread` prend comme paramètres : l'adresse dans la mémoire centrale où commencer à charger les données provenant du fichier, la taille de chaque bloc mémoire à copier (ils ont la même taille), le nombre de blocs à copier et un pointeur sur le fichier depuis lequel copier/charger les données.
Elle retourne le nombre d'éléments qui ont pu être lus dans le fichier.

Lecture dans un fichier binaire - exemple

```c
#include <stdio.h>
#define MAX 10 int
main(void) {     int
nbNumbers, i;     float
numbers[MAX];
    FILE* pf = fopen("../monFichier.dat", "r");
    if (pf == NULL) {
        printf("Erreur d'ouverture du fichier.\n");
        return 0;
    }
    fread(&nbNumbers, sizeof(int), 1, pf); // grâce à cet entier, on saura ensuite combien de floats charger dans le tableau
    if (fread(numbers, sizeof(float), nbNumbers, pf) != nbNumbers) {
        printf("Problème de fichier.\n");
    }
    fclose(pf);
    pf = NULL;
    printf("Il y a %d valeurs.\n", nbNumbers);
    for(i = 0; i < nbNumbers; i++) {
        printf("%f", numbers[i]);
        printf(i < nbNumbers-1 ? ", " : ".");
    }
     return 0;
}
```

Nous sommes capables de faire le chargement car nous savons que le fichier contient un entier, suivi de N `float`. N étant l'entier lu en premier.

Se positionner dans un fichier binaire

Quand on lit ou écrit dans un fichier binaire, on se déplace automatiquement du nombre d'octets correspondants à la valeur lue/écrite. La fonction **`fseek`** permet de se positionner manuellement dans un fichier, pour pouvoir lire ou écrire à l'endroit souhaité. L'écriture dans un bloc écrase son ancienne valeur.

```
fseek(pf, 3*sizeof(int), SEEK_SET);
```

Le premier paramètre correspond au pointeur sur fichier, le second à l'offset, c'est à dire au nombre d'octets à parcourir à partir du troisième paramètre qui est soit :

- **`SEEK_SET`** pour indiquer que l'on souhaite se positionner à partir du début du fichier
- **`SEEK_END`** pour indiquer que l'on souhaite de positionner à partir de la fin du fichier
- **`SEEK_CUR`** pour indiquer que l'on souhaite de positionner à partir de la position actuelle

Se positionner dans un fichier binaire - exemple

```c
#include <stdio.h>
int main(void) {
    const int nbNumbers = 5;
    float b = 0.0;
    float numbers[nbNumbers] = {3.43, 6.354, 5.6, 123.23, 64.2};
    FILE* pf = NULL;
    if ((pf = fopen("../monFichier.dat", "w+")) == NULL) {
        printf("Erreur d'ouverture du fichier.\n");
        return 0;
    }
    fwrite(&nbNumbers, sizeof(int), 1, pf);
    if (fwrite(numbers, sizeof(float), nbNumbers, pf) != nbNumbers) {
        printf("Problème d'écriture dans le fichier.\n");
    }
    fseek(pf, 1*sizeof(int)+2*sizeof(float), SEEK_SET);
    fread(&b, sizeof(float), 1, pf);
    printf("--- %.2f ---", b); // affiche : --- 5.60 --
    fclose(pf);
    pf = NULL;
    return 0;
}
```