



Université Saad Dahlab – Blida 1

Faculté des Sciences

Département d'Informatique

Ingénieur en Informatique

Semestre 2 (1^{ère} année)

Algorithmique et Structure de Données 2



CHAPITRE III: FICHIERS

M^{me} AROUSSI (s_roussi@esi.dz)

2024-2025

Disponible sur <https://sites.google.com/a/esi.dz/informatiqueblida/algorithmique-et-structure-de-donn%C3%A9es>

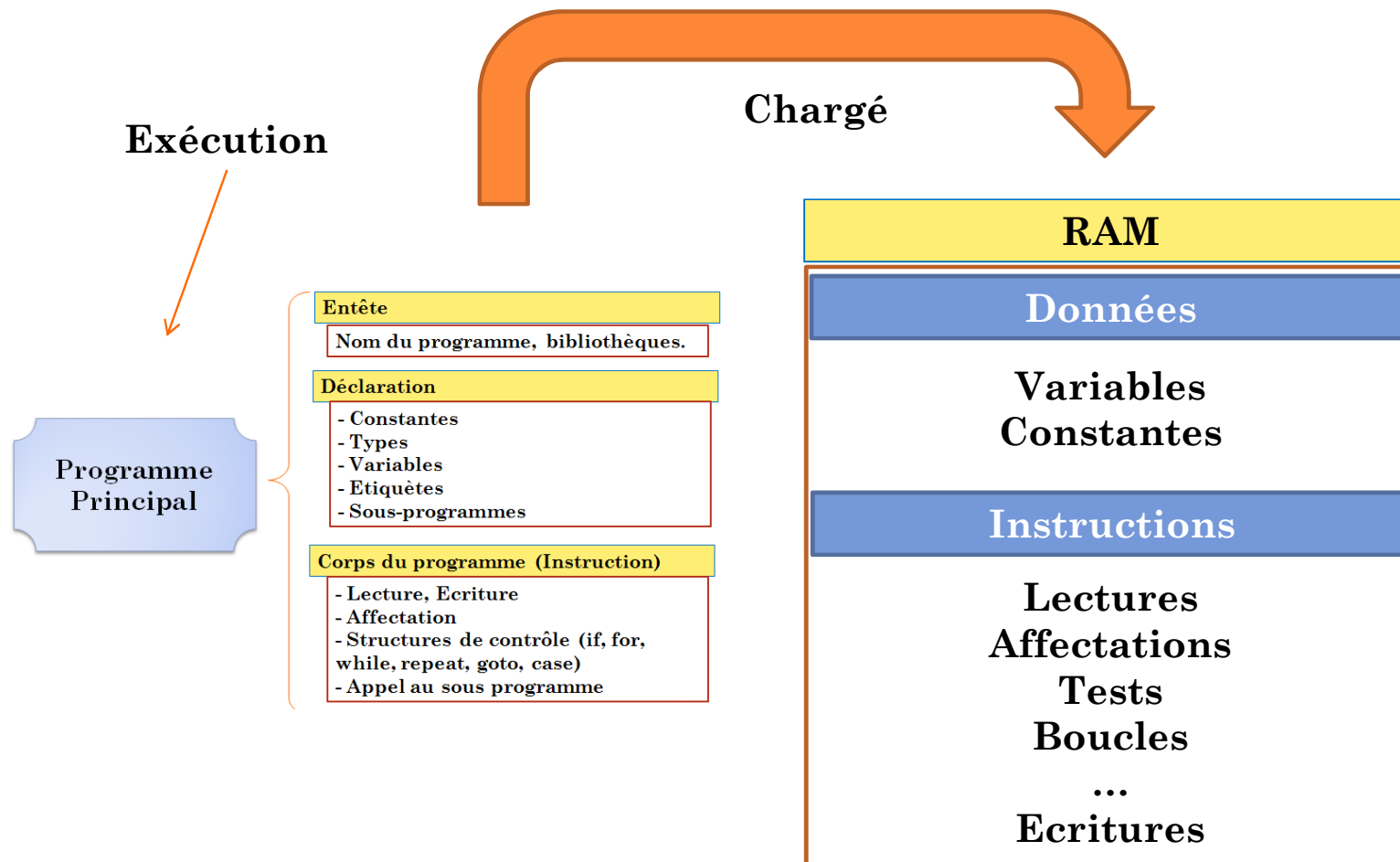
CONTENU

- ❑ Introduction sur les fichiers: Nécessité (Besoins), Tampon (Buffer), Définition, Types des fichiers, Types d'accès.
- ❑ Les fichiers et le langage C: Structure File, Ouverture et Fermeture, Lecture/Ecriture, Positionnement (accès direct), Renommer et supprimer un fichier
- ❑ Les fichiers et le langage Algorithmique
- ❑ Conclusion

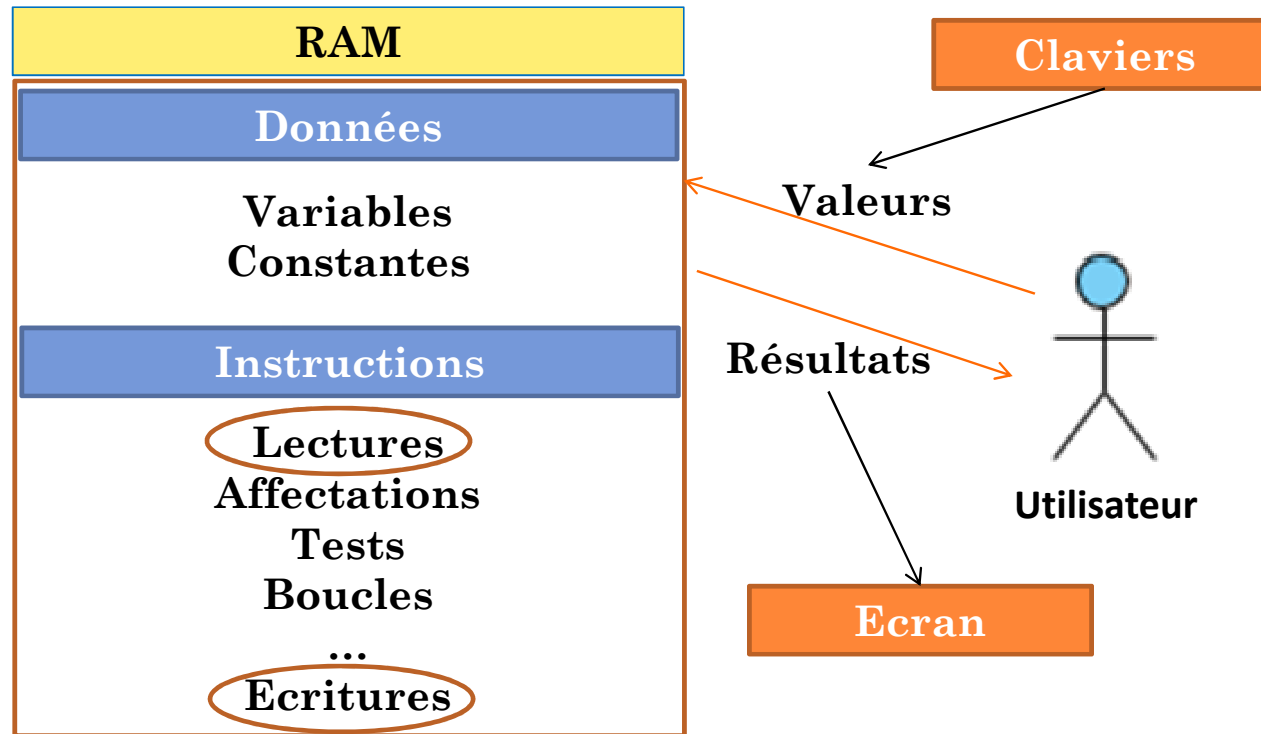
INTRODUCTION

- ❖ Jusqu'à présent, les données utilisées dans nos programmes sont :
 1. incluses dans le programme lui-même, par le programmeur,
 2. entrées à l'exécution par l'utilisateur.
- ❖ Mais évidemment, cela ne suffit pas à combler les besoins réels.
- Comment peut-on sauvegarder, dans ce cas-là, les noms et les notes des étudiants, les meilleurs scores des joueurs, les documents textes qu'on rédige...
- ✓ nécessité d'un moyen de stockage permanent
- ❖ Les fichiers sont là pour combler ce manque. Ils servent à stocker des données de manière permanente, entre deux exécutions d'un programme.

INTRODUCTION

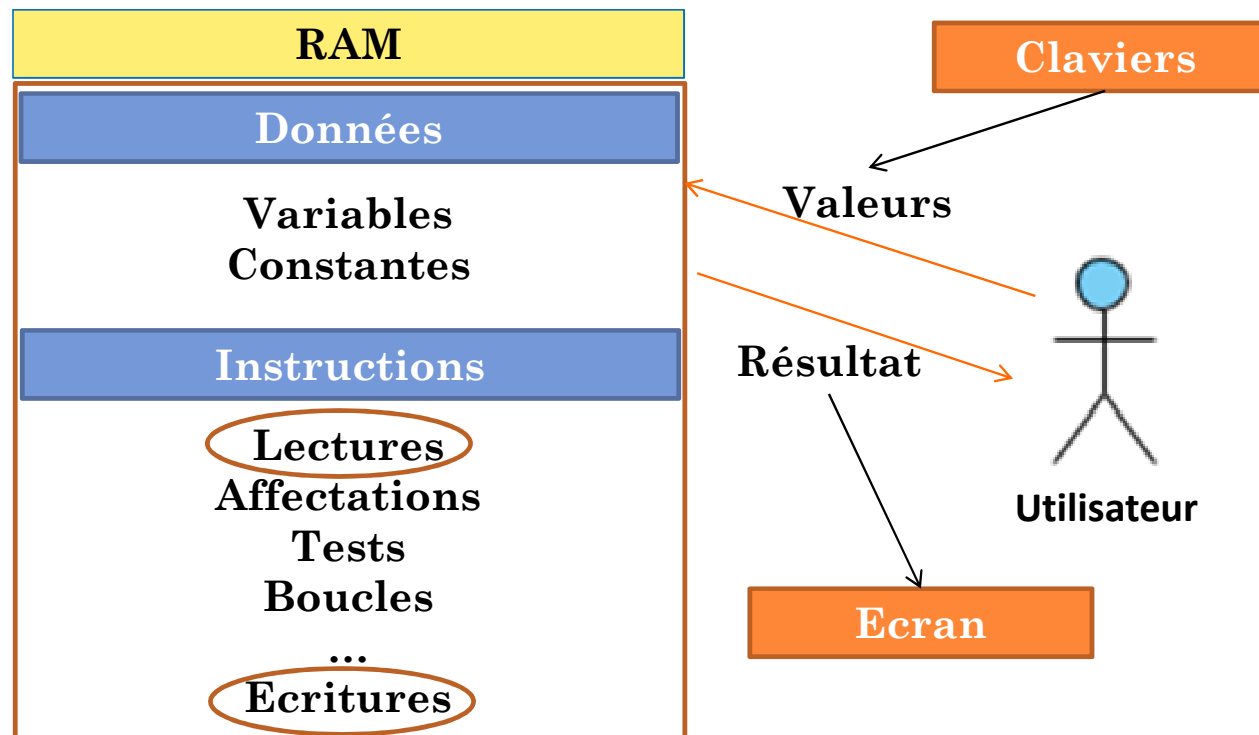


INTRODUCTION



Les données sont enregistrées dans la R.A.M.
La R.A.M est volatile

INTRODUCTION



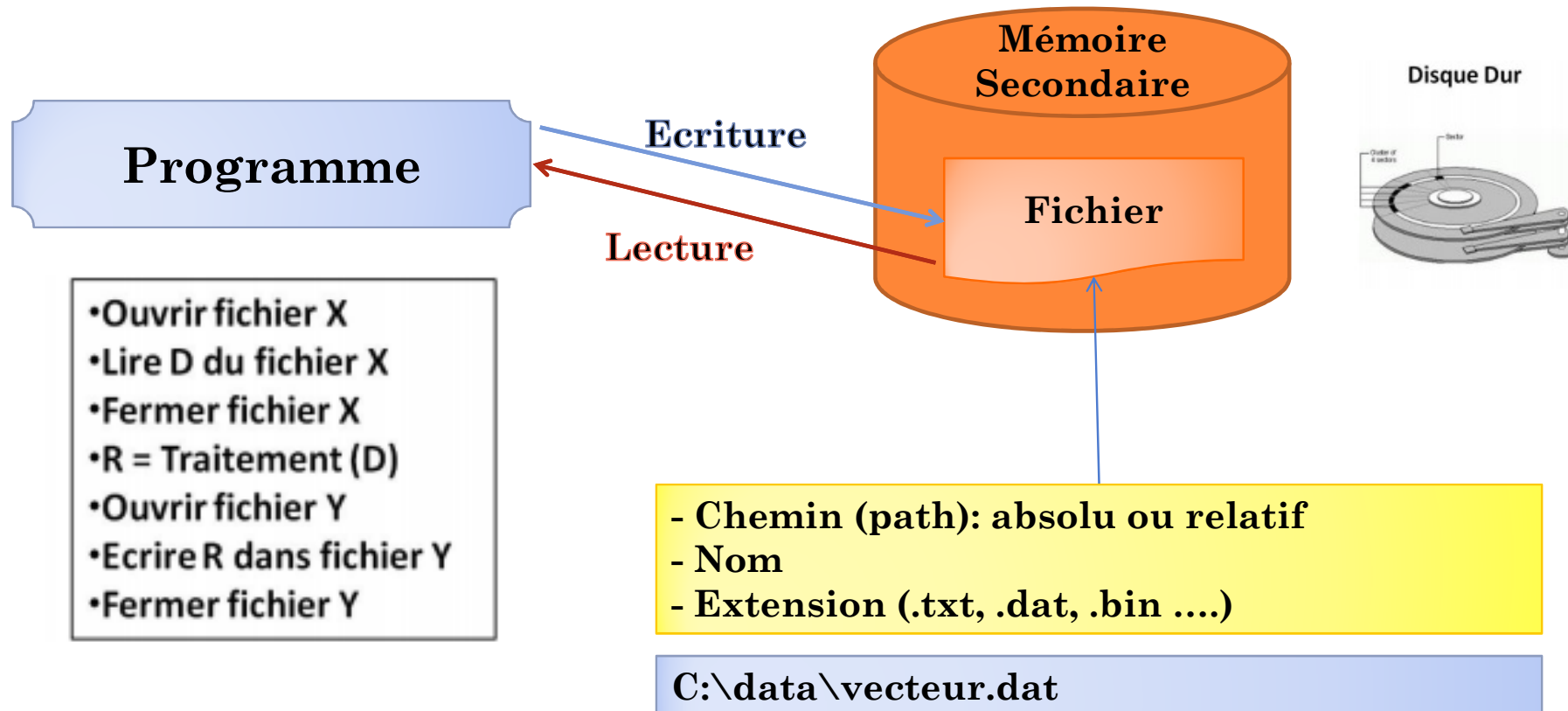
Comment enregistrer les données d'une manière permanente ?
Utilisation des FICHIERS

INTRODUCTION

- ❖ Toute donnée en mémoire externe est organisée sous forme de fichier(s).
- ❖ Un fichier (file en anglais) est une suite d'informations, enregistrée sur un support physique (disque dur, flash disque, CD-ROM) et repérée par un nom.
- ❖ En C, un fichier est une suite d'octets. Les informations contenues dans le fichier ne sont pas forcément de même type (char, int, struct ...)
- ❖ Un fichier n'est pas détruit à l'arrêt de l'ordinateur.
- ❖ La taille d'un fichier n'est pas précisée à sa création

INTRODUCTION

Disque Dur, Flash disque ...



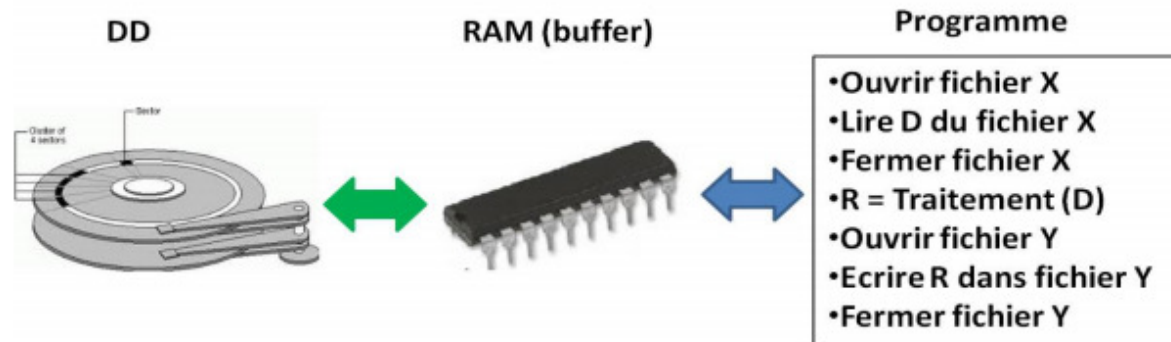
INTRODUCTION

- ❖ In order to understand why file handling is important, let us look at a few features of using files:
- ❖ **Reusability:** The data stored in the file can be accessed, updated, and deleted anywhere and anytime providing high reusability.
- ❖ **Portability:** Without losing any data, files can be transferred to another in the computer system. The risk of flawed coding is minimized with this feature.
- ❖ **Efficient :** A large amount of input may be required for some programs. File handling allows you to easily access a part of a file using few instructions which saves a lot of time and reduces the chance of errors.
- ❖ **Storage Capacity:** Files allow you to store a large amount of data without having to worry about storing everything simultaneously in a program.

INTRODUCTION

- ❖ L'accès à un fichier d'un support de stockage de masse (disque dur, disque optique,...) est coûteux : temps des transferts, mécanique détériorable,...
- ❖ Donc, il faut réduire le nombre d'accès → Utilisation d'une zone mémoire appelée : zone **tampon (buffer)**.
 - Mémoire tampon: zone de mémoire utilisée pour y manipuler des données de façon temporaire, avant de les placer ailleurs.
- ❖ Ainsi, une instruction d'écriture (resp. lecture) dans le programme ne se traduira pas immédiatement par une écriture (resp. lecture) sur le disque mais par une écriture (resp. lecture) dans le buffer, avec écriture (resp. lecture) sur disque quand le buffer est plein (resp. vide).

INTRODUCTION



❖ Dans un programme, le **système d'exploitation** fait :

1. Ouvrir un fichier.
 - créer un buffer (b) dans la RAM.
2. Lire/écrire dans le fichier ouvert.
 - lire/écrire dans le buffer (b).
3. Fermer le fichier.
 - flusher le contenu de b, libérer b,...

INTRODUCTION

❖ A file can be classified into two types based on the way the file stores the data. They are as follows:

1. **A text file** contains data in the form of ASCII characters and is generally used to store a stream of characters.
 - Each line in a text file ends with a new line character ('\n').
 - It can be read or written by any text editor.
 - They are generally stored with .txt file extension.
 - Text files can also be used to store the source code.



INTRODUCTION

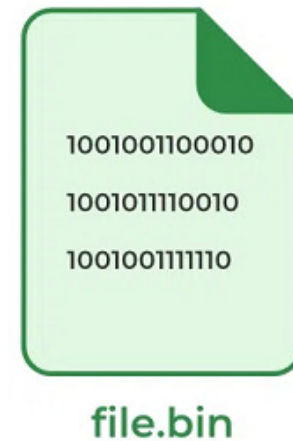
❖ A file can be classified into two types based on the way the file stores the data. They are as follows:

2. **A binary file** contains data in binary form (i.e. 0's and 1's) instead of ASCII characters. They contain data that is stored in a similar manner to how it is stored in the main memory.

➤ The binary files can be created only from within a program and their contents

- can only be read by a program.
- have specific extensions (.dat, .bin, .exe, .mp3, .png, .doc,...).

➤ More secure as they are not easily readable.



INTRODUCTION

- ❖ Selon la manière dont on recherche les informations contenues dans un fichier, on distingue trois types d'accès:
- ❖ **L'accès séquentiel** : On ne peut accéder à une information qu'en ayant au préalable examiné celle qui la précède. Dans le cas d'un fichier texte, cela signifie qu'on lit le fichier ligne par ligne.
- ❖ **L'accès direct** : On peut accéder directement à une information qu'on choisit en précisant le numéro de sa ligne.
- ❖ **L'accès indexé** : combine la rapidité de l'accès direct et la simplicité de l'accès séquentiel. Il est particulièrement adapté au traitement des gros fichiers, comme les bases de données importantes.

FICHIERS EN C

- ❖ Les opérations possibles avec les fichiers sont : Créer - Ouvrir - Fermer - Lire - Ecrire - Détruire - Renommer.
- ❖ La plupart des fonctions permettant la manipulation des fichiers sont rangées dans la bibliothèque standard **<stdio.h>**
- ❖ En langage C, les informations nécessaires à maintenir l'association programme ↔ buffer ↔ disque dur sont décrites dans une structure **FILE**.
- ❖ Parmi les informations stockées dans la structure **FILE** , on trouve l'identifiant du fichier à ouvrir, le type d'ouverture (lecture/écriture), l'adresse du buffer associé, la position du curseur de lecture, la position du curseur d'écriture,

FICHIERS EN C

- ❖ Pour utiliser un fichier, il faut donc commencer par déclarer une variable de type `FILE *`, ou plus exactement un pointeur sur `FILE` qu'on appelle aussi flux de données :
`FILE * nomPointeurFichier ;`

- ❖ L'ouverture d'un fichier se fait à l'aide de la fonction **`fopen`** de prototype

`FILE * fopen(char* nomExterneFichier, char* mode);`

- `nomExterneFichier` est une chaîne de caractères contenant le nom complet (avec le chemin) du fichier à ouvrir.
 - `mode` est une chaîne de caractères définissant le type et le mode d'accès du fichier.
 - La fonction retourne **`NULL`** si l'ouverture n'est pas possible
- ❖ La fermeture d'un fichier se fait à l'aide de la fonction **`fclose`** de prototype

`void fclose(FILE * ptrF)`

FICHIERS EN C

```
int main()
{
    // file pointer
    FILE* fptr;

    // creating file using fopen() access mode "w"
    fptr = fopen("file.txt", "w");

    // checking if the file is created
    if (fptr == NULL) {
        printf("The file is not opened. The program will exit now");
        exit(0);
    }
    else {
        printf("The file is created Successfully.");
        // // Reading/writing in file
        // ...
        // Closing the file using fclose()
        fclose(fptr);
    }

    return 0;
}
```

FICHIERS EN C

`FILE * fopen(char* nomExterneFichier, char* mode);`

- ❖ Le `nomExterneFichier` est le nom du fichier concerné figurant sur le disque

```
FILE* f1, f2, f3, f4;  
f1 = fopen("montexte1.txt", "r");           // meme dossier  
f2 = fopen("TP2/monTP2.c", "w");           // chemin relatif  
f3 = fopen("C:\\Mes_TP\\TP3\\source.h", "a"); // chemin absolu  
f4 = fopen("/home/Toto/Loulou/TD.doc", "a"); // sous linux
```

- f1 doit être situé dans le même dossier que l'exécutable (.exe)
- f2 est situé dans un sous-dossier appelé "TP2". c'est ce qu'on appelle **chemin relatif**. Peu importe l'endroit où est installé votre programme cela fonctionnera toujours.
- Il est aussi possible d'ouvrir un autre fichier n'importe où ailleurs sur le disque dur en utilisant ce qu'on appelle **chemin absolu** (par exemple le fichier f3).
- Le défaut des chemins absolus, c'est qu'ils ne fonctionnent que sur un OS précis. Sous **Linux** par exemple, on aurait dû écrire un chemin à-la-linux, comme pour le fichier f4.

FICHIERS EN C

FILE * fopen(char* nomExterneFichier, **char* mode**);

- ❖ Le second paramètre (mode) spécifie le mode d'accès au fichier, les spécificateurs de mode d'accès diffèrent suivant le type du fichier (texte ou binaire)

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
"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
"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

FICHIERS EN C

FILE * fopen(char* nomExterneFichier, **char* mode**);

Opening Modes	Description
r , r+, rb, r+b	Searches file. If the file is opened successfully fopen() loads it into memory and sets up a pointer that points to the first character in it . If the file cannot be opened fopen() returns NULL.
w, w+, wb, w+b	Searches file. If the file exists, its contents are overwritten . <i>If the file doesn't exist a new file is created</i> . Returns NULL, if unable to open the file.
a, a+, ab, a+b	Searches file. If the file is opened successfully fopen() loads it into memory and sets up a pointer that points to the last character in it . It opens only in the append mode. <i>If the file doesn't exist, a new file is created</i> . Returns NULL, if unable to open the file.

FICHIERS EN C

- ❖ Pour Lire ou Ecrire (L/E) des données dans un fichier, il existe différentes façons:
 - ❖ L/E non-formaté : on lit/écrit un caractère (fgetc, fputc), ou plusieurs caractères (fgets, fputs);
 - ❖ L/E formaté : on lit/écrit des données typées avec fscanf et printf?
 - ❖ L/E par bloc : on manipule des séquences d'octets, indépendamment de leur type (fread; fwrite).
- ❖ Un fichier texte est un cas particulier de fichier binaire : on peut donc le manipuler en accès binaire, avec les fonctions de lecture/écriture binaires (fread, fwrite). Mais on utilise surtout des fonctions d'entrée/sortie sur les chaînes de caractères (fprintf, fgets, fscanf,...), qui ressemblent à celles dont on dispose pour écrire à l'écran ou lire au clavier.

FICHIERS EN C

- ❖ Il existe plusieurs fonctions permettant de lire des données à partir d'un fichier. Il faut choisir celle qui est la plus adaptée à votre cas :

Function	Description
fgetc()	Reads a single character from the file.
fgets()	Input the whole line from the file.
fscanf()	Use formatted string and variable arguments list to take input from a file.
fread()	Reads the specified bytes of data from a binary file.

- So, it depends on you if you want to read the file line by line or character by character.

FICHIERS EN C

- ❖ Cette fonction lit un caractère à la fois depuis le fichier :

int fgetc(FILE* ptrF) ;

```
FILE* f = NULL;
int c = 0;
f = fopen("montexte.txt", "r");
if (f != NULL){
    do{
        c = fgetc(f); // On lit le caractere
        printf("%c", c); // On l'affiche
    } while (c != EOF); // On continue tant que fgetc n'
        a pas retourne EOF (fin de fichier)
}
```

- Cette fonction retourne un int : c'est le caractère qui a été lu.
- Le curseur est placé sur le caractère suivant;
- Si la fonction n'a pas pu lire de caractère, elle retourne EOF (End Of File).

FICHIERS EN C

- ❖ After reading a particular part of the file, the file pointer will be automatically moved to the end of the last read character.
- ❖ The reading functions return EOF (End Of File) when they reach the end of the file while reading. EOF indicates the end of the file and its value is implementation-defined.
- ❖ Cette constante est fréquemment utilisée dans les fonctions manipulant des fichiers, pour indiquer qu'un comportement anormal s'est produit.
- ❖ La fonction `feof()` détecte une fin de fichier dans un flux `ptrf`.

int feof (FILE * ptrF) : Returns a true value when the position indicator has reached the end of the file

FICHIERS EN C

- ❖ Cette fonction lit une chaîne (une ligne) depuis le fichier.

char* fgets(char* chaine, int nbr_Caracteres, FILE* ptrF) ;

```
#define TAILLE_MAX 1000
int main()
{ FILE* f = NULL;
  char chaine[TAILLE_MAX] = "";
  f = fopen("montexte.txt", "r");
  if (f != NULL)
      {fgets(chaine, TAILLE_MAX, f); // On lit maximum
        TAILLE_MAX caracteres du fichier, on stocke le
        tout dans "chaine"
      printf("%s", chaine); // On affiche la chaine
      fclose(f);
  }
}
```

- nbr_Caracteres est le nombre de caractères à lire. En effet, la fonction fgets s'arrête de lire la ligne si elle contient plus de nbr_Caracteres caractères.
- Elle lit au maximum une ligne (elle s'arrête au premier \n qu'elle rencontre).
- Elle retourne NULL en cas d'erreur ou si la fin du fichier (EOF) est atteinte avant toute lecture

FICHIERS EN C


- ❖ Cette fonction lit des données formatées à partir d'un flux de fichier, en respectant un format spécifié, et stocke les valeurs extraites dans les variables fournies en argument.

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

```
int main()
{
    FILE* f = NULL;
    Etudiant e = {"Toto", 19};

    int age = 0;
    f = fopen("montexte.txt", "r");
    if (f != NULL)
    {
        fscanf(fichier, "%s %d", &e.nom, &e.age);
        printf("Nom : %s, Age : %d ans", e.nom, e.age);
        fclose(f);
    }
}
```

typedef struct
{ char nom[20];
int age;
} Etudiant;



- La fonction fscanf s'utilise de la même manière que scanf.

FICHIERS EN C

- ❖ Cette fonction permet un certain nombre de données depuis un fichier :

int fread(void *p, int taille, int nombre, FILE* ptrF) ;

```
FILE* f = NULL;
Etudiant e, etu_tab[3];
f = fopen("montexte.txt", "r");
if (f != NULL)
{
    fread(&e, sizeof(Etudiant), 1, f);
    fread(etu_tab, sizeof(Etudiant), 3, f);
    fclose(f);
}
```

- fread lit dans le fichier "**nombre**" éléments, chacun de "**taille**" octets, et range les éléments lus en mémoire à l'adresse "**p**".
- fread retourne le nombre d'éléments effectivement lus.
- la lecture se fait à partir de la position courante du curseur, et déplace le curseur du nombre d'éléments lus

FICHIERS EN C

- ❖ Il existe plusieurs fonctions capables d'écrire dans un fichier. Il faut choisir celle qui est la plus adaptée à votre cas :

Function	Description
fputc()	Prints a single character into the file.
fputs()	Prints the whole line in the file and a newline at the end
fprintf()	Similar to printf(), this function use formatted string and variable arguments list to print output to the file.
fwrite()	This functions write the specified amount of bytes to the binary file.

FICHIERS EN C

- ❖ Cette fonction écrit un caractère à la fois dans le fichier :

int fputc(int caractere, FILE* ptrF) ;

```
FILE* f = NULL;
f = fopen("montexte.txt", "r+");
if (f != NULL)
{
    fputc('A', f); // Ecriture du caractere 'A'
    fclose(f);
}
else
    printf("Impossible d'ouvrir le fichier montexte.txt");
```

- La fonction fputc retourne un int, c'est un code d'erreur. Il vaut EOF si l'écriture a échoué, sinon il a une autre valeur

FICHIERS EN C

- ❖ Cette fonction écrit une chaîne de caractères dans le fichier :

int fputs(const char* chaine, FILE* ptrF) ;

```
FILE* f = NULL;
f = fopen("montexte.txt", "r+");
if (f != NULL)
{
    fputs("Ecriture de Toto\n et Loulou dans le fichier", f);
    fclose(f);
}
else
    printf("Impossible d'ouvrir le fichier montexte.txt");
```

- La fonction fputs retourne un int, c'est un code d'erreur. Il vaut EOF si l'écriture a échoué, sinon il a une autre valeur

FICHIERS EN C

- ❖ Cette fonction écrit une chaîne de caractères dans le fichier :

int fputs(const char* chaine, FILE* ptrF) ;

```
FILE* f = NULL;
f = fopen("montexte.txt", "r+");
if (f != NULL)
{
    fputs("Ecriture de Toto\n et Loulou dans le fichier", f);
    fclose(f);
}
else
    printf("Impossible d'ouvrir le fichier montexte.txt");
```

- La fonction fputs retourne un int, c'est un code d'erreur. Il vaut EOF si l'écriture a échoué, sinon il a une autre valeur

FICHIERS EN C

- ❖ Cette fonction écrit des données formatées dans un fichier spécifié par ptrF, en suivant le format donné par format et les arguments fournis.

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

```
FILE* f = NULL;
int age = 0;
f = fopen("montexte.txt", "r+");
if (f != NULL)
{
    // On demande l'age
    printf("Quel age avez-vous ? ");
    scanf("%d", &age);

    // On l'ecrit dans le fichier
    fprintf(f, "Vous avez %d ans", age);
    fclose(f);
}
else
    printf("Impossible d'ouvrir le fichier montexte.txt");
```

- La fonction fprintf s'utilise de la même manière que printf.

FICHIERS EN C

- ❖ Cette fonction écrit une chaîne de caractères dans le fichier :

int fwrite(void *p, int taille, int nombre, FILE* ptrF) ;

```
typedef struct
{ char nom[20];
  int age;
} Etudiant;
FILE* f = NULL;
Etudiant e1, etu_tab[3];
f = fopen("montexte.txt", "r+");
if (f != NULL)
{ fwrite(&e1, sizeof(Etudiant), 1, f);
  fwrite(etu_tab, sizeof(Etudiant), 3, f);
  fclose(f);
}
```

- fwrite écrit dans le fichier "nombre" éléments pointés par "p", chacun de "taille" octets ;
- Elle retourne le nombre d'éléments effectivement écrits ;
- L'écriture se fait à partir de la position courante du curseur, et déplace celui-ci du nombre d'éléments écrits.

FICHIERS EN C

- ❖ Le langage C permet un accès direct aux données d'un fichier (c-à-d. se positionner dans un fichier), en utilisant ces trois fonctions :
 - **long ftell (FILE * ptrF);** renvoie la position actuelle du curseur sous la forme d'un long.
 - **int fseek (FILE * ptrF , long déplacement , int origine);** positionne le curseur à un endroit précis.
 - **void rewind (FILE * ptrF);** remet le curseur au début du fichier (c'est équivalent à demander à la fonction fseek de positionner le curseur au début).

FICHIERS EN C

- ❖ La fonction `fseek` permet de déplacer le curseur d'un certain nombre de caractères (indiqué par `deplacement`) à partir de la position indiquée par `origine`. **`int fseek (FILE * ptrFichier , long deplacement , int origine);`**
 - Le nombre `deplacement` peut être un nombre positif (pour se déplacer en avant), nul (= 0) ou négatif (pour se déplacer en arrière).
 - Quant au nombre `origine`, il prend l'une des trois constantes (généralement des `#define`) listées ci-dessous :
 1. `SEEK_SET` : indique le début du fichier ;
 2. `SEEK_CUR` : indique la position actuelle du curseur ;
 3. `SEEK_END` : indique la fin du fichier.

FICHIERS EN C

- ❖ La fonction `fseek` permet de déplacer le curseur d'un certain nombre de caractères (indiqué par `deplacement`) à partir de la position indiquée par `origine`. **`int fseek (FILE * ptrFichier , long deplacement , int origine);`**

```
FILE* f = NULL;
Etudiant e, etu_tab[3];
f = fopen("toto.txt", "r");
if (f != NULL)
{
    fread(&e, sizeof(Etudiant), 1, f);
    fread(etu_tab, sizeof(Etudiant), 3, f);
    fseek(f, 0, SEEK_SET);
    fseek(f, sizeof(Etudiant), SEEK_CUR);
    fseek(f, -2*sizeof(Etudiant), SEEK_END);
    printf("%d    %d\n", ftell(f), sizeof(Etudiant));
    rewind(f);
    printf("%d\n", ftell(f));
    fclose(f);
}
```

FICHIERS EN C

- ❖ Le langage C permet de renommer et de supprimer un fichier existant en utilisant ces deux fonctions :

int rename (const char * ancienNom , const char * nouveauNom);

permet de renommer un fichier ;

int remove (const char * fichierASupprimer); permet de supprimer un fichier sans demander de confirmation .

- ❖ Ces fonctions ne nécessitent pas de pointeur de fichier (FILE *), il suffit d'indiquer directement le nom du fichier concerné. Elles retournent 0 en cas de succès et une valeur non nulle en cas d'échec (ex. si le fichier n'existe pas ou est utilisé par un autre programme).

FICHIERS EN C

- ❖ Le langage C permet de renommer et de supprimer un fichier existant en utilisant ces deux fonctions :

int rename (const char * ancienNom , const char * nouveauNom);

permet de renommer un fichier ;

int remove (const char * fichierASupprimer); permet de supprimer un fichier sans demander de confirmation .

```
void main()  
{  
    rename("test.txt", "test1.txt");  
    remove("test1.txt");  
}
```

FICHIERS EN LANGAGE ALGORITHMIQUE

- ❖ Dans ce cours, nous utiliserons les mêmes noms pour les modules de manipulation des fichiers, tout en respectant la syntaxe du langage algorithmique étudié précédemment (déclaration, affectation, types de modules, modes de passage, etc.). ,,,,,,Modèle de fichier

Catégorie	Langage C	Langage algorithmique
Déclaration	FILE* ptrF;	prtF : *FILE
Ouverture et fermeture	FILE *fopen(const char *filename, const char *mode);	Function fopen(filename, mode: string): *FILE
	int fclose(FILE *ptrF);	Procedure fclose(Var ptrF: *FILE);

FICHIERS EN LANGAGE ALGORITHMIQUE

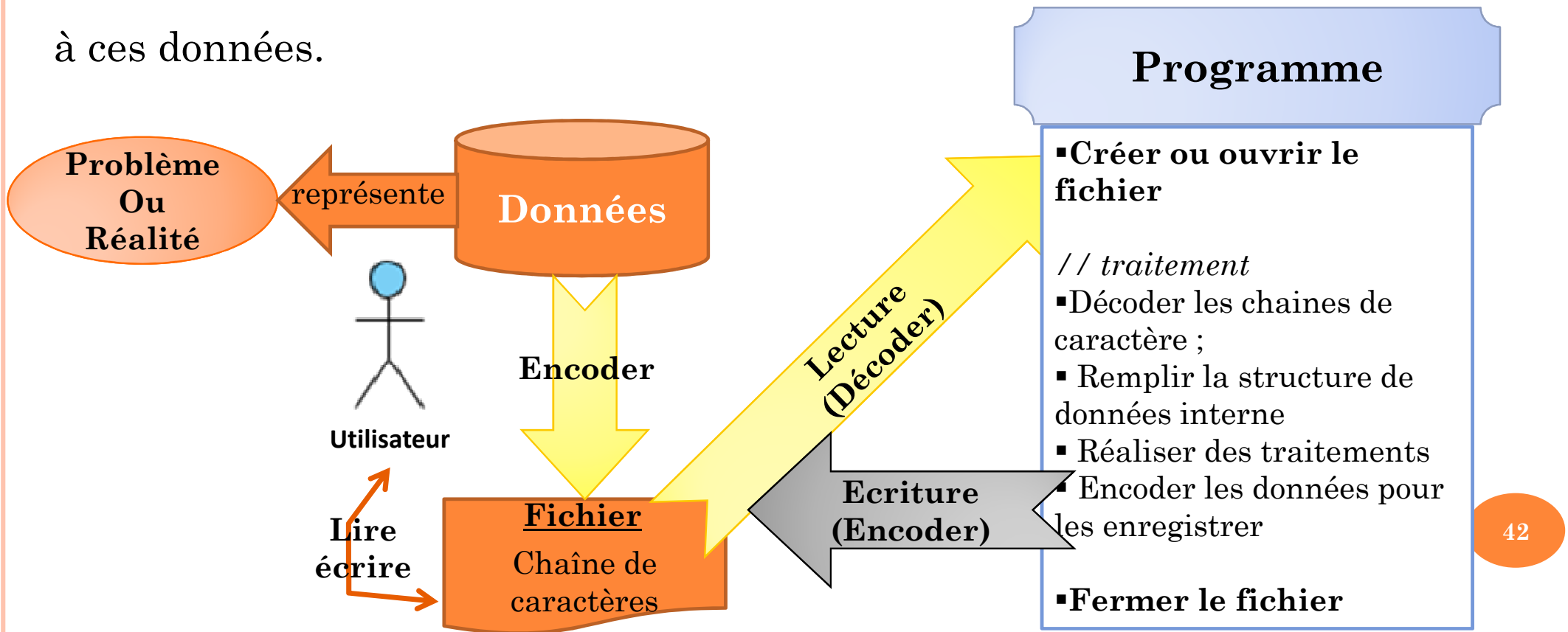
Catégorie	Langage C	Langage algorithmique
Lecture	int fgetc(FILE *ptrF);	function fgetc(ptrF: *FILE): char;
	char *fgets(char *str, int n, FILE *ptrF);	procedure fgets(Var str: string, n: integer, ptrF: *FILE)
	int fscanf(FILE *ptrF, const char *format, ...);	procedure fscanf(ptrF: *FILE,)
	size_t fread(void *ptr, size_t size, size_t count, FILE *ptrF);	procedure fread(ptr: *void, size, count: integer, ptrF: *FILE);
Écriture	int fputc(int c, FILE *ptrF);	procedure fputc(c: char, ptrF: *FILE);
	int fputs(const char *str, FILE *ptrF);	procedure fputs(str: string, ptrF: *FILE);
	int fprintf(FILE *ptrF, const char *format, ...);	Procedure fprintf(ptrF: *FILE,...);
	size_t fwrite(const void *ptr, size_t size, ...)	Procedure fwrite(ptr: * void, size, count:

FICHIERS EN LANGAGE ALGORITHMIQUE

Catégorie	Langage C	Langage algorithmique
Gestion de la position	<code>int feof(FILE *ptrF);</code>	Function <code>feof(ptrF: *FILE): boolean</code>
	<code>long ftell(FILE *ptrF);</code>	Function <code>ftell(ptrF: *FILE): integer</code>
	<code>void rewind(FILE *ptrF);</code>	Procedure <code>rewind(ptrF: *FILE);</code>
	<code>int fseek(FILE *ptrF, long dep, int origine);</code>	procedure <code>fseek(rewind(ptrF: *FILE, dep, origine: integer);</code>
Renommage et suppression	<code>int rename(const char *oldname, const char *newname);</code>	procedure <code>rename(oldname, newname: string);</code>
	<code>int remove(const char *filename);</code>	procedure <code>remove(filename: string);</code>

CONCLUSION

- ❖ En conclusion, un fichier n'est pas structuré par nature, il est vu simplement comme une séquence d'octets. C'est au programmeur de donner une structure à ces données.



SOURCES DE CE COURS

- H. YKHLEF (2022), Cours d'initiation à l'algorithmique (Les fichiers), 1ère année MI
Département Informatique, Université de Tlemcen,
<https://sites.google.com/site/informatiquemessabihi/>
- D. BERTHET, V. LABATUT. Algorithmique & programmation en langage C - vol.1 : Supports
de cours Volume 1 Période 2005-2014. Licence. Algorithmique et Programmation, Istanbul,
Turquie.
- S. BAARIR (2011), Cours d'Algorithmes et programmation II : La gestion des fichiers, Licence
Mia , Université Paris Ouest Nanterre La Défense. [https://pages.lip6.fr/Souheib.Baarir/Cours-
C/cours/Gestion%20des%20ES/Gestion%20des%20fichiers.pdf](https://pages.lip6.fr/Souheib.Baarir/Cours-C/cours/Gestion%20des%20ES/Gestion%20des%20fichiers.pdf)
- Chapitre Fichiers, https://sc-mi.univ-batna2.dz/sites/default/files/mi/files/chapitre_fichier.pdf