Chapitre 10: Structures et Fichiers

1. Structures
2. Fichiers

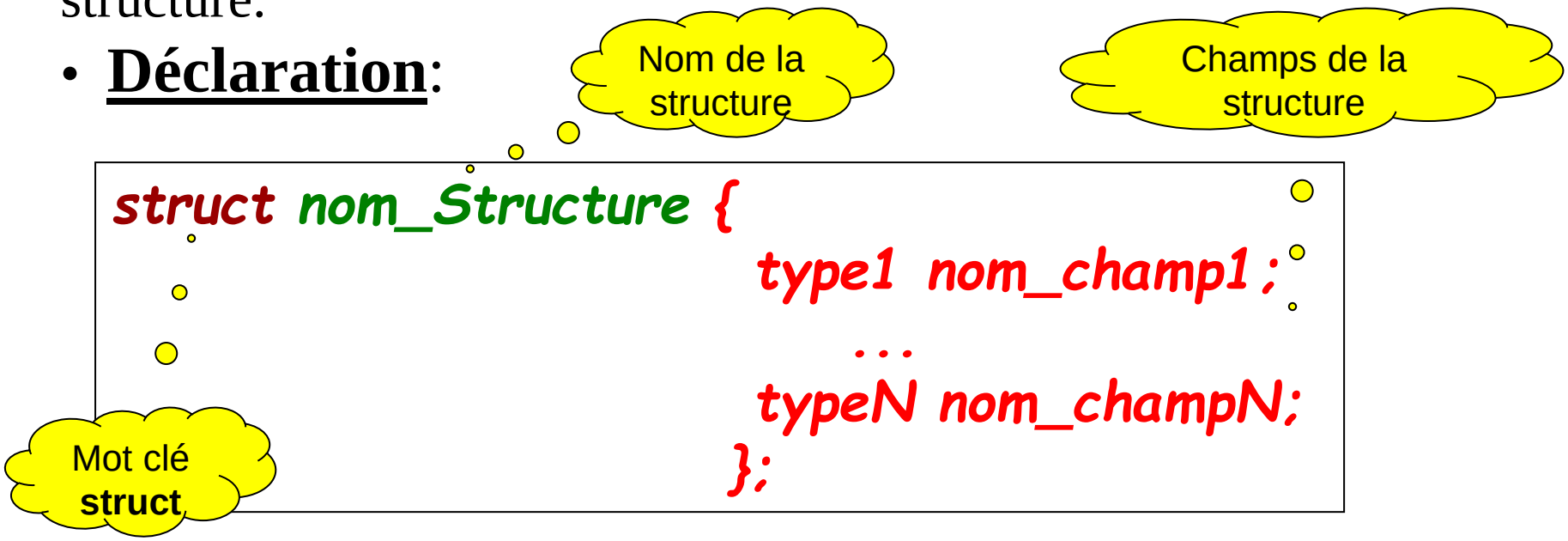
Partie N°:1

Structures

- **Définition**:

Une **structure** est un ensemble d'éléments, pouvant être de **types** différents, désigné par un seul **nom**. L'accès à chaque élément (nommé **champ**) s'effectue par son nom au sein de la structure.

- **Déclaration**:



Structures (suite)

- **Exemple**: (Déclaration d'une structure)

```
struct Etudiant {
                    char Nom[20];
                    char Prenom[20];
                    int Age;
                    float Moyenne;
                };
```

- **Remarques**:
  - Deux champs ne peuvent pas avoir le même nom.
  - Un champ peut être de n'importe quel type sauf le type de la structure dans laquelle il se trouve.

Structures (suite)

- **Définition d'une variable structurée**:
  - → La déclaration d'une structure ne réserve pas d'espace mémoire, mais elle défini le modèle (l'allure) de la structure.
  - → Après avoir déclarée une structure, il faut définir les **variables structurées**.
  - → La déclaration d'une variable structurée se fait comme suit:

```
struct Nom_Structure Nom_variable_structurée;
```

  - → On peut définir plusieurs variables structurées en les séparant avec des virgules.

```
struct Nom_Structure Nom1,Nom2,Nom3,…;
```

Structures (suite)

- **Exemple**: (Variable structurée)

```
struct Etudiant {
                    char Nom[20];
                    char Prénom[20];
                    int Age;
                    float Moyenne;
                };

struct Etudiant Etud1,Etud2,Etud3;
```

• On peut regrouper la déclaration et la définition dans une seule instruction de la façon suivante:

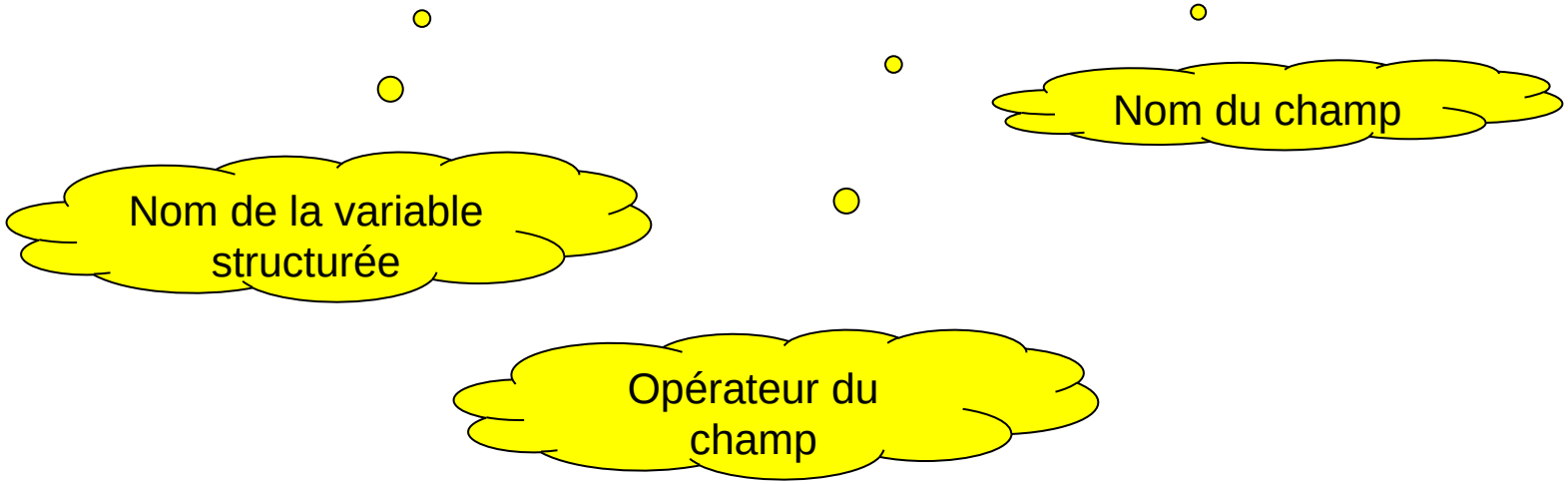
```
struct nom_Structure {
                    type1  nom_champ1;
                    ...
                    typeN nom_champN;
                }Nom1,Nom2,Nom3,…;
```

Structures (suite)

- **Accès aux champs d'une variable structurée**:

→ L'accès aux champs s'effectue à l'aide de l'opérateur du champ (**un point .**) placé entre le nom de la variable structurée et le nom du champ souhaitée.

```
Nom_variable_structurée.Nom_champ;
```



Structures (suite)

- **Exemple**: (Accès aux champs)

```
struct Etudiant {
                        char Prenom[20];
                        int Age;
                        float Moyenne;
                    };

struct Etudiant Etud1;

strcpy(Etud1.Prenom,"Ali");
Etud1.Age = 23;
Etud1.Moyenne = 15.27;
```

Structures (suite)

- **Structures imbriquées**:
  - → On peut imbriquer plusieurs structures.
  - → Exemple:

```
struct date{
                int jour;
                int mois;
                int annee;
            };
struct Etudiant {
                 char Nom[20];
                 char Prénom[20];
                 float Moyenne;
                struct date naissance;
                 }Etud1;
```

  - → Accès aux champs de la sous-structure utilise deux fois l'opérateur point (.) : Etud1.naissance.jour

Structures (suite)

- **Tableaux de structures**:
  - → Il est possible de créer un tableau comportant des éléments de type d'une structure donnée.

```
struct Nom_structure Nom_tableau[taille];
```

  - → Exemple:

```
struct date{
                int jour;
                int mois;
                int annee;
            };
struct Etudiant {
                 char Nom[20];
                 char Prénom[20];
                 float Moyenne;
                struct date naissance;
                 };
struct Etudiant Liste[30];
```

Structures (suite)

- **Utilisation de « *typedef* »:**
  - → Le mot-clé *typedef* permet d'associer un **nom** à un **type** donné.
  - → On l'utilise suivi de la déclaration d'un type puis du nom qui remplacera ce type

  - → Exemple:

```
typedef struct {
                    char Nom[20];
                    char Prénom[20];
                    float Moyenne;
                  }Etudiant;
Etudiant Etud1,Etud2,Etud3;
```

Structures (suite)

- **Structures et fonctions**:
  → Passage par valeur:

```c
#include<stdio.h>
struct enreg{
    int a;
    float b;
};
void fct(struct enreg s)
{
    s.a=0;
    s.b=1;
    printf("Dans fct: %d  %f\n",s.a,s.b);
}
main()
{
    struct enreg x;
    x.a=1; x.b=12.5;
    printf("Avant appel de fct: %d  %f\n",x.a,x.b);
    fct(x);
    printf("Apres appel de fct: %d  %f\n",x.a,x.b);
}
```

```
Avant appel de fct: 1  12.500000
Dans fct: 0  1.000000
Apres appel de fct: 1  12.500000

--------------------------------
```

Structures (suite)

- **Structures et fonctions**:

  → Passage par adresse: (Opérateur →)

```c
#include<stdio.h>
struct enreg{
    int a;
    float b;
};
void fct(struct enreg *s)
{
    s->a=0;
    s->b=1;
    printf("Dans fct: %d  %f\n",s->a,s->b);
}
main()
{
    struct enreg x;
    x.a=1; x.b=12.5;
    printf("Avant appel de fct: %d  %f\n",x.a,x.b);
    fct(&x);
    printf("Apres appel de fct: %d  %f\n",x.a,x.b);
}
```

```
Avant appel de fct: 1  12.500000
Dans fct: 0  1.000000
Apres appel de fct: 0  1.000000

--------------------------------
```

Exercice1

On suppose qu'on a les déclarations suivantes:

```
typdef struct {
                int jour;
                int mois;
                int annee;
              } date;
```

```
typedef struct {
                char Nom[20];
                char Prénom[20];
                float Moyenne;
                date naissance;
                } Etudiant ;
 Etudiant Liste[30];
```

Ecrire un programme c qui permet de saisir et afficher les informations des étudiants.

Chapitre 10: Structures et Fichiers

1. Structures
2. Fichiers

Partie N°:2

Fichiers

- **Définition**:

Un **fichier** est un ensemble d'informations numériques stockées sur un support mémoire, désigné par un **nom** et manipulé comme une **unité**.

Les caractéristiques d'un fichier sont:

- **Nom** (***suffixe.extension***)
- **Taille**
- **Chemin d'accès**
- **Format**
- …etc

Types de fichiers	Extensions
Textes et documents	.txt, .doc, .xls, …
Images	.gif, .jpg, .bmp, .png, …
Audio	.mp3, .wav, …
Vidéo	.avi, .mpg, …
Exécutables	.exe, .com, .bat, …
Compressés	.rar, .zip, …

Fichiers (suite)

-Il est possible 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...).

-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 de données** ou bien **flux** (**stream** en anglais).

Fichiers (suite)

- **Flux**:

Les entrées/sorties en langage C se font par des **flux**. Il existe deux grandes catégories de flux :

- **Flux de textes**: interprétation des caractères de contrôle.
- **Flux binaires**: pas d'interprétation des caractères de contrôle.

Un flux est représenté par un **pointeur** sur une structure de type **FILE**.

Les trois flux prédéfinis par <stdio.h> sont:

- **stderr** : Flux d'erreurs standard, par défaut, l'écran.
- **stdout** : Flux de sortie standard, par défaut, l'écran.
- **stdin** : Flux d'entrée standard, par défaut, le clavier.

Fichiers (suite)

- **Cycle de vie**:

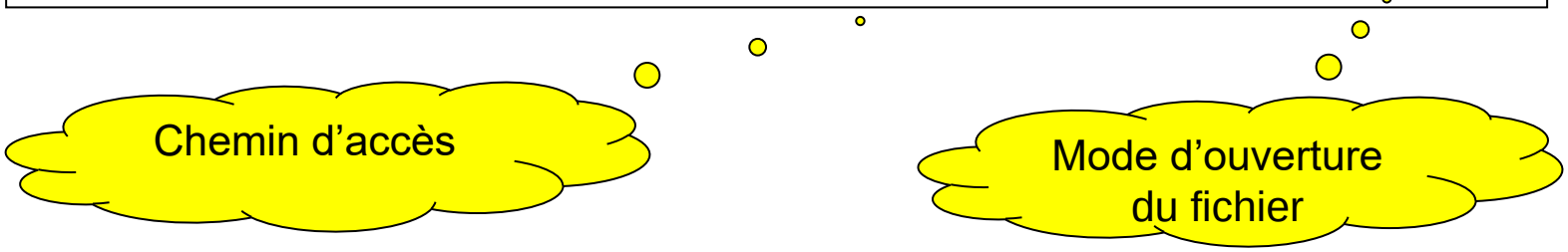
Le cycle d'utilisation d'un flux (lecture ou écriture) est le suivant:

1.Ouverture du flux
2.Traitement (lecture des données ou écriture des données)
3.Fermeture du flux

Fichiers (suite)

- **Ouverture**: (fopen)

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



- Si un fichier existe à l'intérieur d'un sous dossier, il faut doubler le caractère antislash (\\) au lieu de un (\).
- La fonction retourne un **pointeur** sur le flux demandé si l'ouverture a réussi ou bien **NULL** si elle a échoué.

Fichiers (suite)

• Pour le **flux texte**, les différents modes d'ouverture disponibles sont:

→ **"r"** : lecture à partir du début du fichier. Le fichier indiqué par le premier argument doit obligatoirement exister, sinon la fonction échoue.

→ **"w"** : écriture à partir du début du fichier. Si le fichier n'existe pas, il sera créé. Si il existe, son contenu est effacé.

→ **"a"** : écriture à partir de la fin du fichier. Si le fichier n'existe pas, il sera créé.

→ **"r+"** : lecture et écriture à partir du début du fichier. Le fichier indiqué par le premier argument doit obligatoirement exister, sinon la fonction échoue.

→ **"w+"** : lecture et écriture à partir du début du fichier. Si le fichier n'existe pas, il sera créé. Si il existe, son contenu est effacé.

→ **"a+"** : lecture et écriture à partir de la fin du fichier. Si le fichier n'existe pas, il sera créé.

Fichiers (suite)

• Pour le **flux binaire**, les différents modes d'ouverture disponibles sont:**"rb","wb","ab","r+b","w+b" et"a+b".**

• Ces modes (**flux binaire**) comportant les mêmes explications que les modes du **flux texte**.

Fichiers (suite)

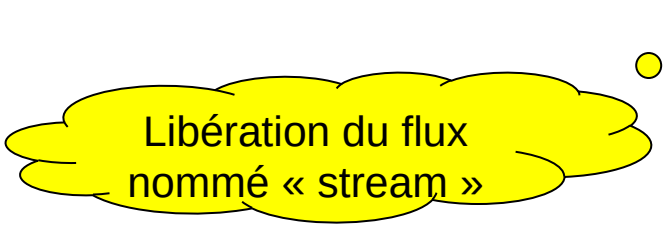
- **Exemple 1**: (Ouverture d'un flux)

```
#include <stdio.h>
Main()
{
  FILE *fich=fopen("fichier.txt","r");

  if(fich==NULL)
        printf("Erreur");

   /* Opérations sur le fichier*/

}
```

Fichiers (suite)

- **Fermeture**: (fclose)

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



- La fonction renvoie **0** en cas de réussite, et **EOF** (**E**nd **O**f **F**ile) s'elle ne réussit pas à fermer correctement le fichier.

Fichiers (suite)

- **Exemple 1**: (Ouverture et fermeture d'un flux)

```
#include <stdio.h>
Main()
{
  FILE *fich=fopen("fichier.txt","r");

  if(fich==NULL)
        printf("Erreur");

   /* Opérations sur le fichier*/

  fclose(fich);

}
```

Fichiers (suite)

- **Exemple 2**: (Ouverture et fermeture de plusieurs flux)

```
#include <stdio.h>
#define TAILLE_MAX 20
#define NB_FICHIER 5
main(void)
{   FILE * fichier[NB_FICHIER];
    char s[TAILLE_MAX];
    size_t i;
    for(i = 0; i < NB_FICHIER; ++i)
    {   sprintf(s, "fichier%d.txt", i + 1);
        fichier[i] = fopen(s, "r");
        if(fichier[i] == NULL)
        { printf("Erreur lors de l'ouverture du fichier %d\n", i + 1); }
    }
    /* ... */
    for(i = 0; i < NB_FICHIER; ++i)
    {  fclose(fichier[i]); }
}
```

Fichiers (suite)

- **Lecture d'un caractère**: (fgetc)

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

- Il faut savoir que lorsque l'on parcourt un flux, un **indicateur de position** est utilisé pour indiquer où on en est dans le flux.
-La fonction renvoie donc le **caractère lu** si tout s'est bien passé ou **EOF** si l'indicateur de position est tout à la fin du fichier (ou en cas d'erreur).

Fichiers (suite)

- **Exemple 1**: (Lecture d'un caractère)

```
#include <stdio.h>
Main()
{
   FILE *fich;
   int c;
   fich=fopen("fichier.txt","r");
   if(fich==NULL)
          printf("Erreur");

    c=fgetc(fich);
    putchar(c);

   fclose(fich);
}
```

Fichiers (suite)

- **Exemple 2**: (Lecture du contenu du fichier)

```
#include <stdio.h>
Main()
{
  FILE *fich;
  int c;
  fich=fopen("fichier.txt","r");
  if(fich==NULL)
        printf("Erreur");

   while((c=fgetc(fich))!=EOF)
     { putchar(c);  }

  fclose(fich);
}
```

Fichiers (suite)

- **Lecture d'une chaîne de caractères**: (fgets)

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

- La fonction lit caractère par caractère le nombre de caractères voulus ***(size)*** depuis le flux ***stream*.**
- La fonction possède encore deux conditions d'arrêt: **retour à la ligne et EOF**.
- Le pointeur ***s*** contient la chaîne lue.
- La fonction retourne le pointeur *s* dans le cas où la lecture est réussie et **NULL** dans les deux cas suivants: **problème de lecture** et **EOF**.

Fichiers (suite)

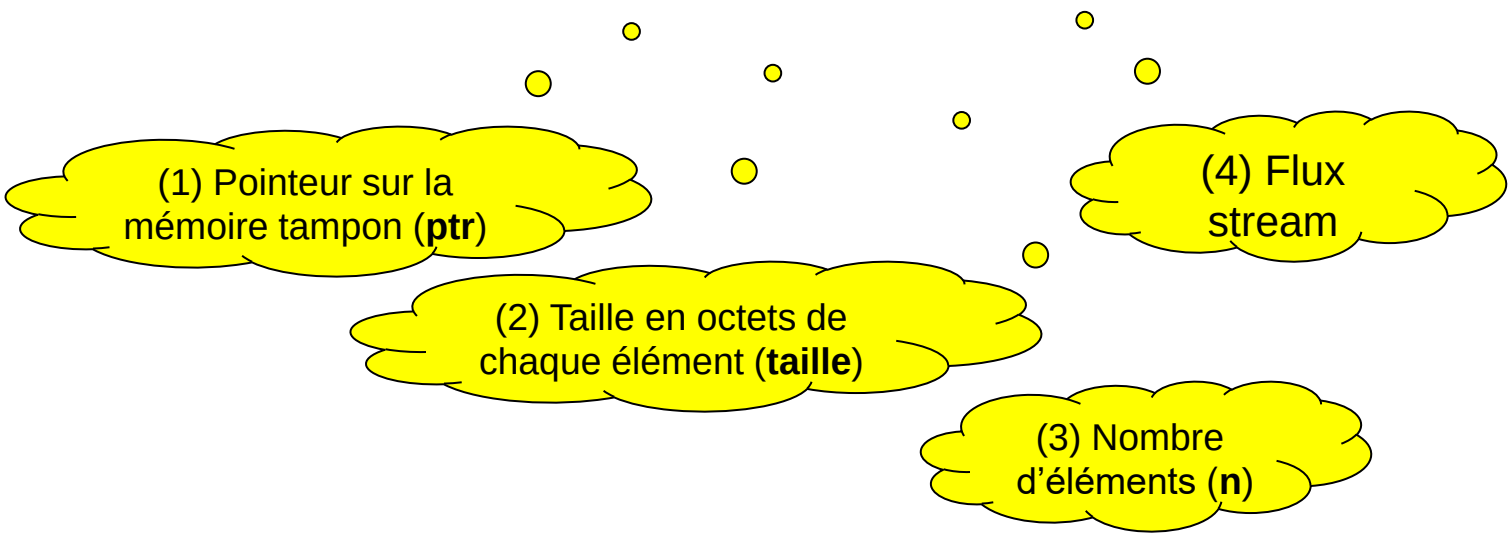
- **Exemple 1**: (Lecture d'une chaîne de caractères)

```
#include <stdio.h>
#define max 50
Main()
{
   FILE *fich;
   char chaine[max];
   fich=fopen("fichier.txt","r");
   if(fich==NULL)
            printf("Erreur");

    fgets(chaine,max,fich);
    puts(chaine);

   fclose(fich);
}
```

Fichiers (suite)

- **Lecture d'une chaîne de caractères**: (fread)

```
int fread(……,……,……,……);
```



- La fonction lit un tableau de **n** éléments , ayant chacun une **taille**, depuis le flux **stream** et stocke ces éléments à l'adresse spécifiée par **ptr**.
- La fonction retourne le nombre d'éléments lus, si cette valeur est différente de **n**, soit une erreur de lecture est survenue soit la fin du fichier est atteinte

Fichiers (suite)

- **Exemple 1**: (Lecture d'une chaine de caractères)

```
#include <stdio.h>
Main()
{
   FILE *fich;
   char ch[20];
   fich=fopen("fichier.txt","r");
   if(fich==NULL)
          printf("Erreur");

    fread(ch,sizeof(char),20,fich);
     puts(ch);
   fclose(fich);
}
```

Fichiers (suite)

- **Ecriture d'un caractère**: (fputc)

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

- La fonction écrit le caractère **c** de type int dans le flux stream. Elle renvoie la plupart du temps le caractère qui vient d'être écrit ; en cas d'erreur, **EOF** est retourné

Fichiers (suite)

- **Exemple 1**: (Ecriture de « MIPC » dans un fichier « test.txt »)

```
#include <stdio.h>
Main()
{
  FILE *fich;
  fich=fopen("test.txt","w");
  if(fich==NULL)
        printf("Erreur");

   fputc('M', fich);
   fputc('I', fich);
   fputc('P', fich);
   fputc('C', fich);

  fclose(fich);
}
```

Fichiers (suite)

• **Exemple 2**: (Ecriture de « MIPC » dans un fichier « test.txt » à l'aide d'une boucle)

```
#include <stdio.h>
Main()
{  char *ch="MIPC";
   int taille;
   FILE *fich;
   taille=strlen(ch);
   fich=fopen("test.txt","w");
   if(fich==NULL)
          printf("Erreur");

    for(i=0,i<taille;i++)
    fputc(ch[i], fich);

   fclose(fich);
}
```

Fichiers (suite)

- **Ecriture d'une chaîne de caractères**: (fputs)

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

- La fonction écrit la chaîne **s** dans le flux stream et retourne un **nombre non négatif** si elle réussit. Dans le cas contraire, elle retourne **EOF**.

Fichiers (suite)

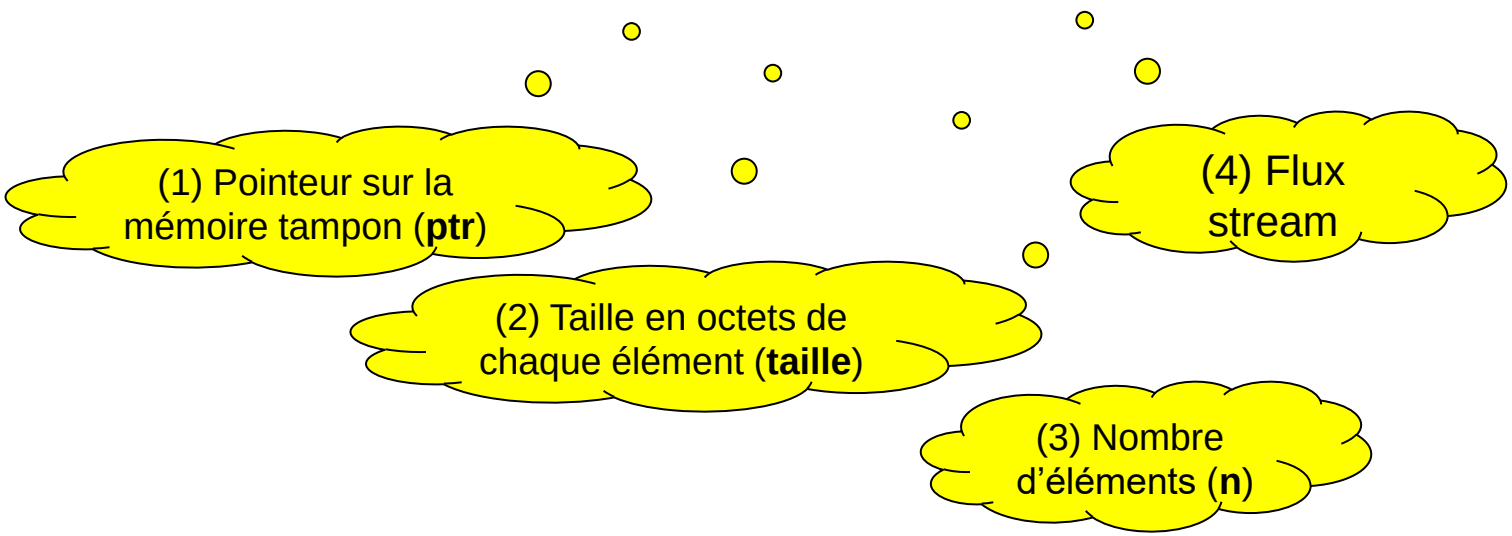
- **Exemple 1**: (Ecriture de « MIPC » dans un fichier « test.txt »)

```
#include <stdio.h>
Main()
{
  FILE *fich;
  fich=fopen("test.txt","w");
  if(fich==NULL)
        printf("Erreur");

   fputs("MIPC", fich);

  fclose(fich);
}
```

Fichiers (suite)

- **Ecriture d'une chaîne de caractères**: (fwrite)

```
int fwrite(……,……,……,……);
```



- La fonction écrit un tableau de **n** éléments , ayant chacun une **taille**, depuis l'adresse spécifiée par **ptr** vers la position courante du flux **stream**.
- La fonction retourne le nombre d'éléments effectivement écrits, s'il diffère de **n**, cela indique une erreur d'écriture.

Fichiers (suite)

- **Exemple 1**: (Ecriture et lecture d'une chaine de caractères)

```
#include <stdio.h>
#include<string.h>
Main()
{
   FILE *fich;
   char ch1[20],ch2[20];
   int lg;
   fich=fopen("fichier.txt","w+");
   if(fich==NULL)
          printf("Erreur");
    printf("Entrer une chaîne de caractères:\n");
    gets(ch1);
    lg= strlen(ch1);
    fwrite(ch1,sizeof(char),lg,fich);
     rewind(fich);
    fread(ch2,sizeof(char),20,fich);
     puts(ch2);
   fclose(fich);
}
```

Fichiers (suite)

- **Déterminer la position du curseur**: (ftell)

```
long ftell(FILE *stream);
```

- La fonction renvoie la position actuelle du curseur dans le flux passé en paramètre.

Fichiers (suite)

- **Repositionner du curseur**: (fseek)

```
int fseek(FILE *stream,long offset,int whense);
```

- ***offset***: la valeur du déplacement du curseur. Une valeur positive indique le déplacement en avant, par contre une valeur négative indique le déplacement en arrière.
- ***whense***: le point de départ utilisé par offset pour déplacer le curseur. Trois constantes sont fournies :
  - → SEEK_SET : début du fichier ;
  - → SEEK_CUR : position actuelle du curseur ;
  - → SEEK_END : fin du fichier ;
- La fonction retourne **0** si elle réussit, une valeur différente de **0** si elle échoue.

Fichiers (suite)

- **Repositionner lu curseur au début**: (rewind)

```
void rewind(FILE *stream);
```

- La fonction sert simplement à remettre le curseur au début du fichier.

Fichiers (suite)

- **Renommer un fichier**: (rename)

```
int rename(char *oldname,char *newname);
```

- Si **newname** désigne un fichier déjà existant, la fonction peut soit échouer, soit écraser l'autre fichier déjà présent, tout dépend du système d'exploitation. Si la fonction échoue, elle retourne encore une fois une valeur différente de 0.

Fichiers (suite)

- **Supprimer un fichier**: (remove)

```
int remove(char *filename);
```

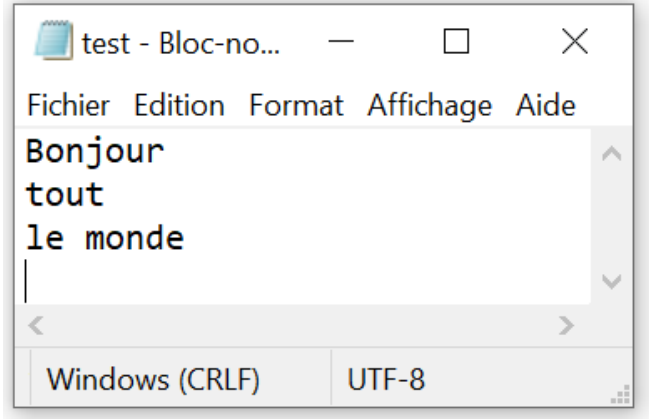
- Si elle réussit, elle retourne la valeur **0** sinon une autre valeur.

Exercice2

Ecrire un programme qui affiche le contenu d'un fichier et compte le nombre de lignes de ce fichier

Exercice2 (Solution)

```c
#include<stdio.h>
#include<stdlib.h>
#define max 200
main()
{
    FILE *fich=fopen("test.txt","r");
    char caract,chaine[max];
    int k=0,lignes=0,i;
    if(fich==NULL) printf("Erreur d ouverture \n");
    while((caract=fgetc(fich))!=EOF)
    {   putchar(caract);
        chaine[k]=caract;k++;
    }
    for(i=0;i<k;i++)
    { if(chaine[i]=='\n') lignes++; }
    if(chaine[k-1]!='\n') lignes++;
    printf("\n Nombre de lignes: %d \n",lignes);
    fclose(fich);
}
```




```
Bonjour
tout
le monde

 Nombre de lignes: 3

--------------------
```