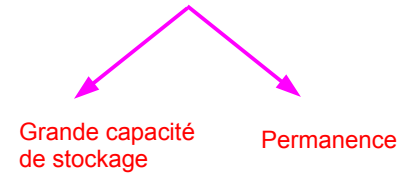


Les fichiers

Et si on apprenait à utiliser
un nouveau type d'entrée sortie

LES FICHIERS

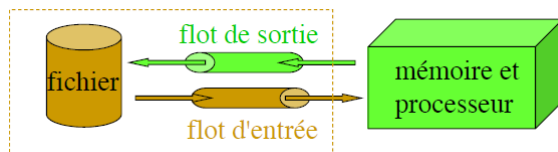
- Il est difficile de trouver UNE notation algorithmique pour les fichiers car ils représentent un concept de programmation hétérogène et multiforme.
- **Définition** : Un fichier est **une séquence d'enregistrements (fiches)** du même type (entier, réel, caractère, ...) rangée sur une **mémoire secondaire** (disque, disquette, ...).



Fichier physique

00100110 | 10100101 | 111101.....

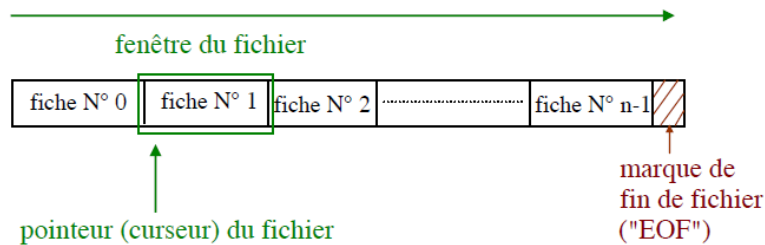
- Un accès au fichier correspond à **un nombre entier d'octets**.
- Les fichiers peuvent être considérés comme des canaux d'entrées et de sorties utilisant des **flots** de données :
 - réception de données dans le cas d'un flot de sortie ;
 - émission de données dans le cas d'un flot d'entrée.



Accès à un fichier

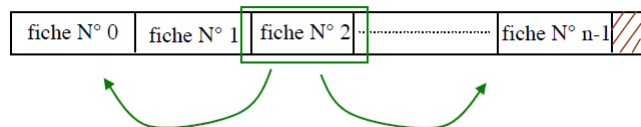
- Il existe 2 types accès à un fichier :
 - accès séquentiel ;
 - accès direct (aléatoire).
- Cette typologie ne se reflète pas dans la structure elle-même du fichier.
 - En fait, tout fichier peut être utilisé avec l'un ou l'autre des deux types d'accès.
 - Le choix du type d'accès n'est pas un choix qui concerne le fichier lui-même, mais uniquement la manière dont il va être traité par la machine. C'est donc dans l'algorithme/programme, et seulement dans ce dernier, que l'on choisit le type d'accès souhaité.

Accès séquentiel



- On ne peut accéder qu'à la donnée suivant celle qu'on vient de lire.
- On ne peut donc accéder à une information qu'en ayant au préalable examinée celle qui la précède.
- Dans le cas d'un fichier texte, cela signifie qu'on lit le fichier ligne par ligne (enregistrement par enregistrement).

Accès direct



- On peut accéder directement à l'enregistrement de son choix, en précisant le numéro de cet enregistrement.
 - Mais cela veut souvent dire une gestion fastidieuse des déplacements dans le fichier.
- Opérations sur un fichier à accès direct :
 - Les mêmes opérations sur un fichier à accès séquentiel ;
 - Utilisé principalement pour la mise à jour des fichiers.

Accès séquentiel : opérations

- Six opérations possibles :
 - création ou désignation ;
 - ouverture ;
 - lecture ;
 - écriture ;
 - fermeture ;
 - test de fin de fichier.

Manipulation de fichiers

- Définition du type fichier : `Fichier`
- Création d'un fichier :
`Créer(nom_fichier, fichier)`
- Désignation d'un fichier :
`Désigner(nom_fichier, fichier)`
- Ouverture d'un fichier :
`Ouvrir(fichier, <mode>)`
`<mode> ::= lecture | écriture | lecture/écriture`
- Fermeture d'un fichier :
`Fermer(fichier)`

Manipulation de fichiers

- Écriture dans un fichier :

`Ecrire (fichier, données)`

- Lecture d'un fichier :

`Lire (fichier, variable)`

- **Remarque** : Après chaque écriture/lecture, le pointeur (curseur) du fichier est automatiquement avancé.

- Accès direct :

`Décaler (fichier, <position>, décalage)`

`<position> ::= Début_fichier | Fin_fichier | Position_courante`

- Fin d'un fichier :

`Fin (fichier) //fonction du type booléen`

Exemples

Algorithme Lecture d'un fichier d'entiers

```
{
  variable nom_fichier : chaîne de caractères
         f_entier : Fichier
         i : entier

  Afficher("Saisir le nom du fichier d'entiers")
  Saisir(nom_fichier)

  Désigner(nom_fichier, f_entier) // désignation du fichier
  Ouvrir(f_entier, lecture) // ouverture du fichier en lecture

  Tant que (non Fin(f_entier)) // test la fin du fichier
  {
    Lire(f_entier, i) // lecture d'un entier dans le fichier
    Afficher(i, "\n")
  }

  Fermer(f_entier) // fermeture du fichier
}
```

Exemples

Algorithme Ecriture d'un fichier d'entiers

```
{
  variable nom_fichier : chaîne de caractères
         f_entier : Fichier
         i : entier

  Afficher("Saisir le nom du fichier d'entiers")
  Saisir(nom_fichier)

  Créer(nom_fichier, f_entier) // création du fichier
  Ouvrir(f_entier, écriture) // ouverture du fichier en écriture

  Pour i de 1 à 100
    Ecrire(f_entier, i) // écriture dans le fichier

  Fermer(f_entier) // fermeture du fichier
}
```

Attention, cet algorithme est pour un fichier non existant.
Pour un fichier existant, on utilise Désigner au lieu de Créer

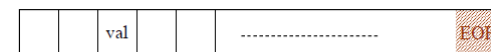
Exemple d'utilisation de fichiers à accès direct : mise à jour d'un fichier d'entiers

```
procédure maj_fich_ent(E f : Fichier)
{
  variable val, n_val, n : entier
         trouvé : booléen

  Afficher("Saisir la valeur à mettre à jour ?")
  Saisir(val)
  Afficher("Saisir la nouvelle valeur ?")
  Saisir(n_val)

  trouvé ← faux;
  Tant que ((non Fin(f)) et (non trouvé)) // recherche de val
  {
    Lire(f, n);
    Si (n=val) alors
      trouvé ← vrai
  }

  Si (trouvé) alors
  {
    Décaler(f, Position_Courante, -1); // positionner le pointeur du fichier
    Ecrire(f, n_val); // mise à jour d'une valeur
  }
  sinon
    Afficher("La valeur ", val, " n'a pas été trouvée.\n");
  }
}
```



Avant Décaler

Après Décaler



Exemple d'utilisation de fichiers à accès direct : mise à jour d'un fichier d'entiers

```
Algorithme Mise à jour
{
    variable nom : chaîne de caractères
           f : Fichier

    Afficher("Saisir le nom du fichier d'entiers")
    Saisir(nom);
    Afficher("mise à jour du fichier d'entiers : ",nom, "\n")

    Désigner(nom,f) // désignation du fichier

    Ouvrir(f,lecture/écriture) // ouverture du fichier en lecture/écriture

    maj_fich_ent(f); // appel à la procédure maj_fich_ent
    Fermer(f);
}
```

Exemple pratique : un fichier d'enregistrements

```
Algorithme Ecriture d'un fichier de Points
{
    constante entier MAX ← 100
    constante chaîne de caractères nom_fichier ← "lespoints.dat"
    variable f : Fichier
           v : vecteur de MAX Point

    Créer(nom_fichier,f) // création du fichier
    Ouvrir(f,écriture) // ouverture du fichier en écriture

    Pour i de 0 à MAX-1
        v[i] ← SaisirPoint()

    Pour i de 0 à MAX-1
        Ecrire(f,v[i]) // écriture dans le fichier

    Fermer(f_entier) // fermeture du fichier
}
```

Exemple pratique : un fichier d'enregistrements

- Écrire un algorithme réalisant la saisie de 100 points et leur stockage dans un fichier créé pour l'occasion et nommé « lespoints.dat ».

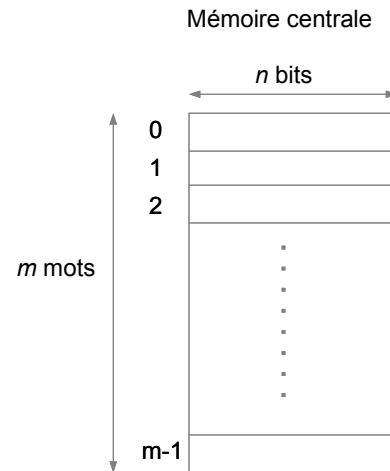
```
structure Point
{
    X,Y : réel
}

fonction SaisirPoint() : Point
{
    variable P : Point
    Afficher("Saisissez les coordonnées du point ")
    Saisir(P.X,P.Y)
    retourne P
}
```

Avant d'aborder les fichiers en C++, nous devons aborder le concept d'adressage et de pointeur.

Souvenons nous de l'organisation de la mémoire centrale

- Un mot mémoire est une unité d'information (un mot binaire – une séquence de bits) adressable en mémoire.
- Suivant l'architecture de la mémoire, le mot pourra être de 8, 16, **32** ou **64** bits.



Mot mémoire et adresses (Théorie)

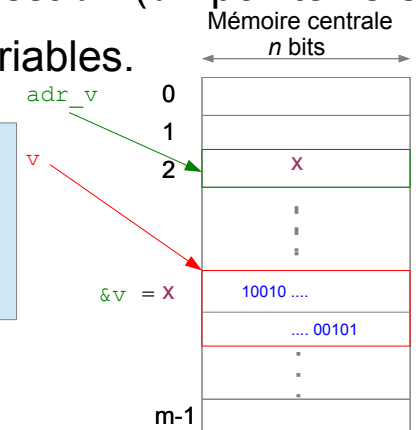
- Ainsi pour une mémoire de 4Go et des mots de 32 bits, il y a 10^9 adresses mémoires.
- Chaque mot (emplacement mémoire) est désigné par une **adresse**.
- Heureusement avec l'exemple ci-dessus, si la technologie utilisée est 32 bits, il est possible d'adresser 2^{32} valeurs, donc de désigner 2^{32} emplacements mémoire. Or avec 4 Go nous avons $10^9 < 2^{32}$
 - Il est en revanche impossible d'adresser plus de 2^{34} octets ≈ 17 Go (2^{32} emplacements * 32 bits pour la taille des emplacements / 8 bits pour avoir la taille en octets).

Opérateur d'adressage : &

- un opérateur unaire qui s'applique à la variable qui le suit ;
- l'adresse de la variable v est $\&v$ ($\&v$ pointe vers v) ;
- $\&$ ne s'applique qu'aux variables.

Ne pas confondre :

- **variable** : un espace mémoire
- **valeur** : contenu d'une zone mémoire
- **adresse** : index d'un mot mémoire
- **pointeur** : variable contenant l'adresse de début d'une zone mémoire



Exemples :

- $\text{adr}_v \leftarrow \&v \Rightarrow \text{adr}_v$ est le pointeur (adresse) vers v
- $\&120 \Rightarrow$ incorrecte
- $\&(a+b) \Rightarrow$ incorrecte.

Mot mémoire / adressage (Pratique)

- Attention ! Certains s'interrogeront peut être sur les données précédentes. En effet, en général, on ne considère qu'on ne peut adresser que 4 Go sur une architecture 32 bits !
- Mais pourquoi est ce que nous trouvons un peu plus de 4 fois plus ?
 - Nous avons considéré une architecture mémoire matérielle appelée « **adressable par mot** » (*word-addressable*) et non une architecture mémoire « **adressable par octet** » (*byte-addressable*). C'est pourtant ce dernier cas qui est le plus fréquent dans la vie courante
 - Ainsi $2^{32} \approx 4\text{Go}$ (les arrondis expliquent le facteur « un peu plus de 4 fois »)



Un entrepôt d'Amazon

La variable : la boîte/le colis/le paquet
 La valeur : le contenu
 L'adresse : la localisation
 Le pointeur : une variable contenant une adresse

L'opérateur de déréférencement : *

- l'opérateur inverse de & pour obtenir le contenu de l'adresse ;
- il s'applique à l'expression qui le suit.

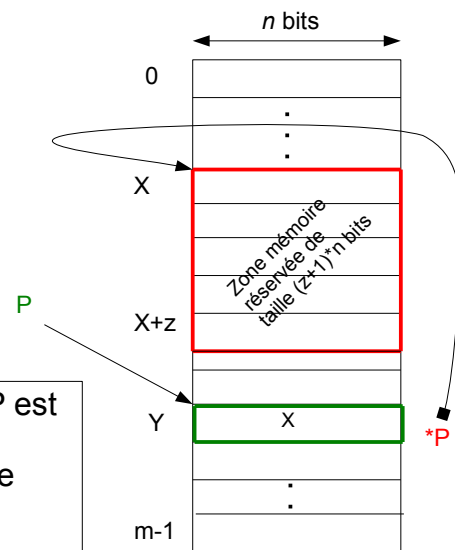
Exemples :

- $v \leftarrow *adr_v$ équivaut à $v \leftarrow *\&v$

c'est à dire $v \leftarrow$ contenu de l'adresse de v .

La notion de pointeur

- Un pointeur est donc une variable qui :
 - a un nom
 - pointe une zone mémoire (désignée par une adresse) qui doit avoir été réservée préalablement à son utilisation



Ici la variable P est un pointeur, pointant la zone mémoire à l'adresse X.

Syntaxe pour les pointeurs en algorithmique et en C++

- Déclaration d'un pointeur :
`nom_pointeur : pointeur sur type_pointé`
- Une valeur particulière NULL (ne pointe sur rien)
`nom_pointeur ← NULL`
- Déclaration d'un pointeur en C++ :
`type_pointé* nom_pointeur;`
- Une valeur particulière NULL (ne pointe sur rien)
`nom_pointeur = NULL;`

Exemple de pointeur

Algorithme Exemple_de_pointeur

```
{
    variable ptr : pointeur sur entier
           A : entier

    ptr ← &A
    ...
}
```

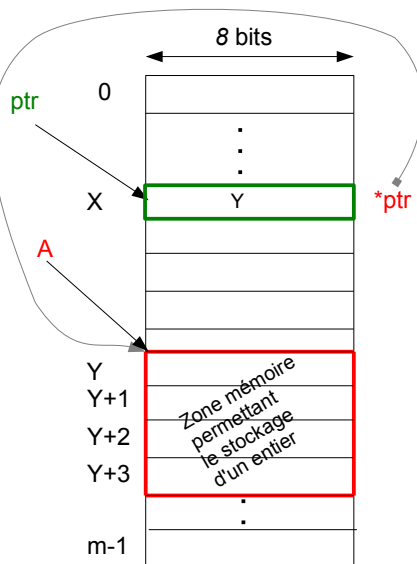
ptr est une variable de type pointeur sur entier. Il occupe donc un espace lui permettant de contenir une adresse en fonction de l'architecture utilisée. Ainsi dans le monde réel sur une architecture 32 bits le pointeur devrait occuper 32 bits (soit la même taille qu'un entier sur 32 bits – ce qui dans le cadre de notre exemple est un peu idiot ! Bref ...).

La déclaration de ptr permet de réserver la zone verte. A ce moment là, la variable ptr, de type pointeur sur entier, contient une valeur particulière par défaut qui s'appelle NULL. Cette valeur indique que le pointeur pointe sur un espace vide.

La déclaration de A permet de réserver la zone rouge.

Une fois que l'instruction d'affectation de l'adresse de A (c'est à dire Y) a eu lieu, il est possible d'accéder à l'espace mémoire permettant le stockage d'un entier (designé par son adresse de début X et représenté en rouge) via l'opérateur de déréférencement *ptr

Pour simplifier, ici, on suppose une mémoire 8 bits et contenant m emplacements, où m = 256. Ainsi, une adresse se représente sur 1 seul emplacement mémoire (sur 8 bits).



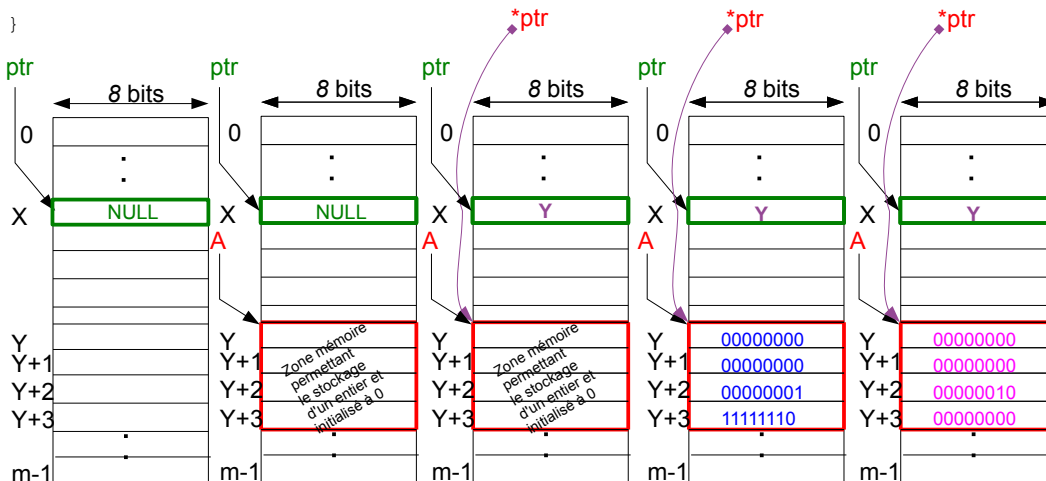
Exemple simpliste d'utilisation de pointeur

Algorithme Exemple_de_pointeur_2

```
{
    variable ptr : pointeur sur entier
           A : entier

    ptr ← &A
    A ← 510
    *ptr ← *ptr + 514 // identique à A ← *ptr + 514 ou *ptr ← A + 514 ou A ← A + 514
}
```

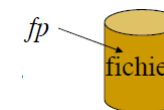
Pour simplifier, ici, on suppose une mémoire 8 bits et contenant m emplacements, où m = 256. Ainsi, une adresse se représente sur 1 seul emplacement mémoire (sur 8 bits).



Bon et tout ça pour quoi déjà ?
Ah oui ! Les fichiers en C++ !

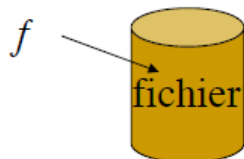
Les fichiers en C++ : fopen

- Pour ouvrir un fichier existant ou créer un nouveau fichier.
- Syntaxe : fp = fopen (nom, mode) ;
 - fp : pointeur sur le fichier en utilisant FILE *fp;
 - nom : nom du fichier et le chemin d'accès ;
 - mode : mode d'accès au fichier
 - "r" => lecture ;
 - "w" => écriture ;
 - "a" => ajout ;
 - "r+" => lecture et écriture (mise à jour).
- fopen retourne le pointeur NULL si l'ouverture (création) du fichier est impossible.



Exemple d'utilisation de `fopen`

```
⋮  
⋮  
FILE *f;  
char nom[35];  
⋮  
⋮  
f=fopen(nom,"w"); // ouverture du fichier en écriture  
⋮  
⋮
```



Les fichiers en C++ : `fwrite`

- Pour l'écriture d'un fichier.
- Syntaxe :

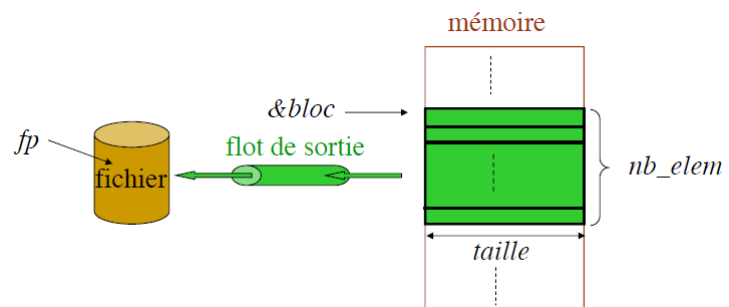
```
nb = fwrite (&bloc,taille,nb_elem,fp);
```

 - `&bloc` : adresse d'un bloc d'information, c'est à dire un tableau ("buffer");
 - `taille` : taille (en octets) du type du bloc (utilisation de `sizeof`);
 - `nb_elem` : nombre d'éléments du bloc à écrire ;
 - `nb` : nombre d'éléments du bloc effectivement écrits ;
 - `fp` : pointeur sur le fichier en utilisant `FILE *fp`
- Remarque : $nb < nb_elem$ en cas d'erreur.

Les fichiers en C++ : `fwrite`

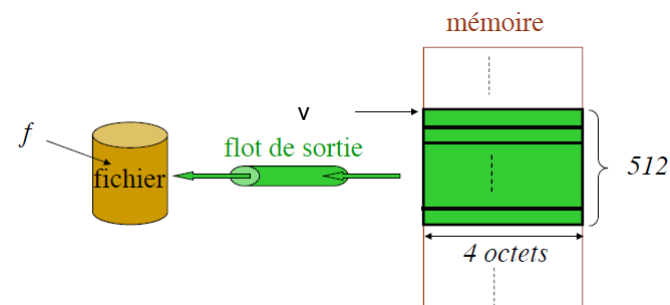
```
nb = fwrite (&bloc,taille,nb_elem,fp);
```

- `&bloc` : adresse d'un bloc d'information, c'est à dire un tableau ("buffer");
- `taille` : taille (en octets) du type du bloc (utilisation de `sizeof`);
- `nb_elem` : nombre d'éléments du bloc à écrire ;
- `nb` : nombre d'éléments du bloc effectivement écrits ;
- `fp` : pointeur sur le fichier en utilisant `FILE *fp`



Exemple d'utilisation de `fwrite`

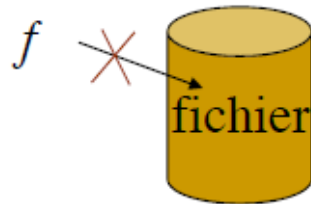
```
⋮  
⋮  
const int n = 512;  
int v[n];  
⋮  
⋮  
nb = fwrite(v,sizeof(int),n,f);  
⋮  
⋮
```



Les fichiers en C++ : `fclose`

- Pour **fermer** un fichier.
- Exemple d'utilisation de `fclose` :

```
⋮  
⋮  
fclose(f);  
⋮  
⋮
```



Oublier de fermer un fichier est une erreur fréquente !

Exemple : écriture d'un fichier d'entiers

```
#include <iostream>  
#include <stdio.h>  
using namespace std;  
  
int main()  
{  
    const int n = 512;  
    int i,v[n];  
    char nom[30];  
    FILE *f;  
  
    cout << "Saisir le nom du fichier d'entiers et son chemin d'accès" << endl;  
    cin >> nom;  
    cout << "écriture du fichier d'entiers : " << nom << endl;  
  
    if ((f = fopen(nom,"w")) == NULL) // ouverture du fichier en écriture  
        cout << "erreur d'ouverture (création) du fichier " << nom << endl;  
    else  
    {  
        for (i=0; i<n; i++) // on remplit le vecteur  
            v[i] = i;  
  
        fwrite(v,sizeof(int),n,f); // écriture de v dans le fichier  
        fclose(f); // fermeture du fichier  
    }  
}
```

Un exemple plus complet

```
⋮  
⋮  
f = fopen("/users/etudiants/f_ent.dat","w"); => EOF  
↑  
v[0] = 5;  
v[1] = 6;  
  
fwrite(v,sizeof(int),2,f); => [ 5 | 6 | EOF ]  
↑  
  
fclose(f);  
⋮  
⋮
```

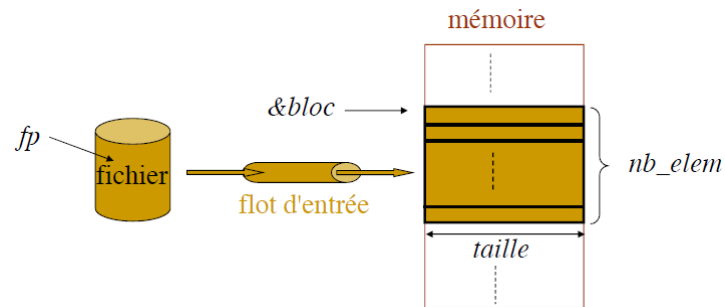
Les fichiers en C++ : `fread`

- Pour la **lecture** d'un fichier.
- Syntaxe :
`nb_lu = fread (&bloc,taille,nb_elem,fp);`
 - `&bloc`, `taille` et `fp` comme précédemment
 - `nb_elem` : nombre d'enregistrements (fiches) du fichier à lire et à placer dans le bloc ("buffer") ;
 - `nb_lu` : nombre d'enregistrements (fiches) du fichier effectivement lus et placés dans le bloc.
- Remarques :
 - `nb_lu < nb_elem` en fin du fichier (dernière lecture) ;
 - `nb_lu = 0 =>` fin du fichier
 - après la lecture de chaque enregistrement (fiche), le pointeur de fichier (curseur) est automatiquement avancé d'un enregistrement.

Les fichiers en C++ : `fread`

```
nb_lu = fread (&bloc,taille,nb_elem,fp);
```

- `&bloc`, `taille` et `fp` comme précédemment
- `nb_elem` : nombre d'enregistrements (fiches) du fichier à lire et à placer dans le bloc ("buffer") ;
- `nb_lu` : nombre d'enregistrements (fiches) du fichier effectivement lus et placés dans le bloc.



Un exemple plus complet

```

:
:
f = fopen("/users/etudiants/f_ent.dat","r"); => [5 | 6 | EOF]

nb_lu = fread(v,sizeof(int),2,f); => [5 | 6 | EOF]

cout << v[0] << " " << v[1] << endl.

fclose(f);
:
:

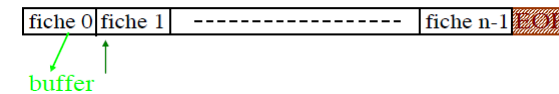
```

Exemple d'utilisation de `fread`

```

:
:
const int n = 512;
int v[n];
:
:
nb_lu = fread(v,sizeof(int),n,f);
:
:

```



Exemple : lecture d'un fichier d'entiers

```

#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    const int n = 512;
    int i,nb_lu,v[n];
    char nom[30];
    FILE *f;

    cout << "Saisir le nom du fichier d'entiers et son chemin d'accès" << endl;
    cin >> nom;
    cout << "lecture du fichier d'entiers : " << nom << endl;

    if ((f = fopen(nom,"r")) == NULL) // ouverture du fichier en lecture
        cout << "erreur d'ouverture du fichier " << nom << endl;
    else
    {
        while ((nb_lu = fread(v,sizeof(int),n,f)) > 0) // lecture du fichier
        {
            for (i=0; i<(nb_lu); i++)
                cout << v[i] << endl; // affichage des éléments lus
        }
        fclose(f); // fermeture du fichier
    }
}

```

Les fichiers en C++ : `feof`

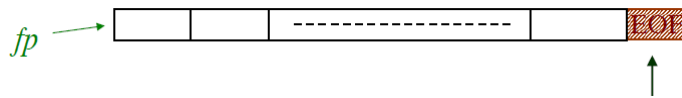
- Pour la détection de la **fin** d'un fichier.

- Syntaxe :

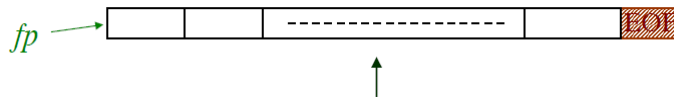
```
feof (fp) ;
```

- `fp` : pointeur sur le fichier en utilisant `FILE *fp`

- `feof = true` si le **pointeur** (curseur) du fichier associé à `fp` est à la fin ;



- `feof = false` sinon.



Les fichiers en C++ : `fseek`

- Pour **positionner le pointeur du fichier** dans le cas d'un **accès direct**

- Syntaxe :

```
fseek (fp, f_pos, origine) ;
```

- `fp` : pointeur sur le fichier en utilisant `FILE *fp`

- `f_pos` : position relative à l'origine en **octets**

- origine :

- `SEEK_SET` => début du fichier ;
- `SEEK_CUR` => position courante ;
- `SEEK_END` => fin du fichier ;

- Remarque : `fseek` retourne une valeur $\neq 0$ si erreur.

Exemple : lecture d'un fichier d'entiers en utilisant `feof`

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    const int n = 512;
    int i,nb_lu,v[n];
    char nom[30];
    FILE *f;

    cout << "Saisir le nom du fichier d'entiers et son chemin d'accès" << endl;
    cin >> nom;
    cout << "lecture du fichier d'entiers : " << nom << endl;

    if ((f = fopen(nom,"r")) == NULL) // ouverture du fichier en lecture
        cout << "erreur d'ouverture du fichier " << nom << endl;
    else
    {
        while ( !feof(f) ) // tant que la fin du fichier n'est pas atteinte
        {
            nb_lu = fread(v,sizeof(int),n,f); // lecture du fichier
            for (i=0; i<(nb_lu); i++)
                cout << v[i] << endl; // affichage des éléments lus
        }
        fclose(f); // fermeture du fichier
    }
}
```

Exemple : Mise à jour d'un fichier d'entiers

```
#include <iostream>
#include <stdio.h>
using namespace std;

void maj_fich_ent(FILE *f)
{
    int val,n_val,n;
    bool trouve;

    cout << "Saisir la valeur à mettre à jour ?" << endl;
    cin >> val;
    cout << "Saisir la nouvelle valeur ?" << endl;
    cin >> n_val;

    trouve = false;
    while ( (! feof(f)) && (! trouve) ) // recherche de val
    {
        fread(&n,sizeof(int),1,f);
        if (n == val)
            trouve = true;
    }

    if (trouve)
    {
        fseek(f,-sizeof(int),SEEK_CUR); // positionner le pointeur du fichier
        fwrite(&n_val,sizeof(int),1,f); // mise à jour d'une valeur
    }
    else cout << val << " n'est pas trouvée" << endl;
}
```

Exemple : Mise à jour d'un fichier d'entiers

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    char nom[30];
    FILE *f;

    cout << "Saisir le nom du fichier d'entiers et son chemin d'accès" << endl;
    cin >> nom;
    cout << "mise à jour du fichier d'entiers : " << nom << endl;

    if ((f = fopen(nom,"r+")) == NULL) // ouverture du fichier en "mise à jour"
        cout << "erreur d'ouverture du fichier " << nom << endl;
    else
    {
        maj_fich_ent(f);
        fclose(f); // fermeture du fichier
    }
}
```

Exemple pratique : un fichier d'enregistrements

```
#include <iostream>
#include <stdio.h>
using namespace std;

... // Insérer ici la définition du type Point et la fonction SaisirPoint

int main()
{
    const int MAX = 100;
    const char[] nom = "lespoints.dat";
    Point v[MAX];
    FILE *f;

    if ((f = fopen(nom,"w")) == NULL) // ouverture du fichier en écriture
        cout << "erreur d'ouverture (création) du fichier " << nom << endl;
    else
    {
        for (i=0; i<MAX; i++) // on remplit le vecteur
            v[i] = SaisirPoint();

        for (i=0; i<MAX; i++)
            fwrite(v,sizeof(Point),MAX,f); // écriture de v dans le fichier

        fclose(f); // fermeture du fichier
    }
}
```

Exemple pratique : un fichier d'enregistrements

- Écrire **un programme** réalisant la saisie de 100 points et leurs stockage dans un fichier créé pour l'occasion et nommé « lespoints.dat ».

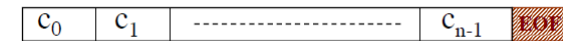
```
struct Point
{
    float X,Y;
};

Point SaisirPoint()
{
    Point P;
    cout << "Saisissez les coordonnées du point ";
    cin >> P.X >> P.Y ;
    return P;
}
```

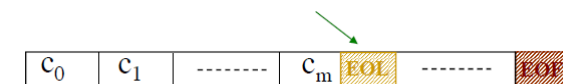
Les fichiers de texte

- Caractéristiques :

- élément de base est **caractère** ;



- fichier peut être structuré en **lignes** : chaque ligne terminée par une marque de fin de ligne (EOL pour End Of Line);



- longueur des lignes est **variable** ;
- les nombres sont représentés sous forme de suite de chiffres (des chaînes de caractères).

Les fichiers en C++ : `fgetc` et `fputc`

- Pour lire un caractère et pour écrire un caractère dans un fichier

- Syntaxe :

```
c = fgetc(fp);
```

- `fp` : pointeur sur le fichier en utilisant `FILE *fp`
- `c` : le caractère lu

```
fputc(c, fp);
```

- `c` : le caractère à écrire
- `fp` : pointeur sur le fichier en utilisant `FILE *fp`

Les fichiers en C++ : `fprintf`

- Format de données

- `%` : spécificateur de format ;
- `%d` : écriture sous forme décimale (entier) ;
- `%o` : écriture sous forme octale (entier) ;
- `%x` : écriture sous forme hexadécimale (entier) ;
- `%u` : écriture sous forme décimale non-signée (entier) ;
- `%f` : écriture sous forme flottante (réels) ;
- `%c` : écriture sous forme caractère ASCII (entier) ;
- `%s` : écriture sous forme chaîne de caractères ;

– Exemple :

```
a = 75;
```

```
fprintf(stdout, "a(déc)=%d , a(oct)=%o , a(hex)=%x , a(car)=%c", a, a, a, a);
```

Affiche à l'écran (`stdout`) – Oui ! L'écran est un fichier ! :-)

=> a(déc)=75 , a(oct)=113 , a(hex)=4b , a(car)=K

Les fichiers en C++ : `fprintf`

- Pour des écritures (sorties) formatées
- Syntaxe :

```
fprintf(fp, "format", arg1, arg2, ...);
```

- `fp` : pointeur sur le fichier en utilisant `FILE *fp`
- `format` est une chaîne de caractères avec 2 types de données :
 - texte à écrire ;
 - format de données (`arg1`, `arg2`, ...) à écrire.

- Exemples :

- `fprintf(fp, "bonjour");` => `bonjour` est écrit dans le fichier.
- `fprintf(fp, "bonjour\n");` => `bonjour` est écrit dans le fichier + 1 nouvelle ligne.

Exemple : écriture d'un fichier de type texte

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    const int max=30, n = 512;
    int i ;
    char nom[max];
    FILE *f;

    cout << "Saisir le nom du fichier texte et son chemin d'accès" << endl;
    cin >> nom;

    if ((f = fopen(nom, "w")) == NULL) //ouverture du fichier en écriture (création)
        cout << "erreur d'ouverture (création) du fichier " << nom << endl;
    else
    {
        for (i=0; i<n; i++)
            fprintf(f, "%d\n", i); //écriture par ligne dans le fichier

        fclose(f); // fermeture du fichier
    }
}
```

Chaque chiffre est codé dans le fichier en ASCII =>

- 0 à 9 : 1 octet / entier ;
- 10 à 99 : 2 octets / entier ;
- 100 à 999 : 3 octets / entier ;

Les fichiers en C++ : `fscanf`

- Pour des lectures (entrées) **formatées**

- Syntaxe :

```
fscanf(fp, "format", &arg1, &arg2, ...);
```

- `fp` : pointeur sur le fichier en utilisant `FILE *fp`
- `format` est une chaîne de caractères précisant le format de données (`arg1, arg2, ...`) à saisir.

- Exemples :

- `fscanf(fp, "%d", &a); =>` lecture d'un entier dans le fichier associé à `fp` et son affectation à `a`.
- `fscanf(fp, "%s", ch); =>` lecture d'une chaîne de caractères dans le fichier associé à `fp` et son affectation à `ch`
 - & `ch` n'a pas raison d'être car en fait, `ch` représente 1 pointeur sur 1 chaîne de caractères.
- `fscanf(stdin, "%d %s %d", &jour, mois, &an); =>` affectation de 2 entiers et d'une chaîne de caractères saisis sur le clavier (`stdin`). – Oui ! Le clavier est aussi un fichier ! :-)

Exemple : Lecture et affichage de tout fichier de type texte. v2

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    const int max = 30;
    char nom[max], c;
    FILE *f;

    cout << "Saisir le nom du fichier texte et son chemin d'accès" << endl;
    cin >> nom;

    if ((f = fopen(nom, "r")) == NULL) // ouverture du fichier en lecture
        cout << "erreur d'ouverture du fichier " << nom << endl;
    else
    {
        while ((c=fgetc(f))!=EOF) // lecture du fichier caractère par caractère
            putchar(c); // affichage caractère par caractère
            // putchar(c) == fputc(c, stdout) c-à-d écriture sur l'écran !

        fclose(f); // fermeture du fichier
    }
}
```

Exemple : Lecture et affichage de tout fichier de type texte. v1

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    const int max = 30;
    char nom[max], c;
    FILE *f;

    cout << "Saisir le nom du fichier texte et son chemin d'accès" << endl;
    cin >> nom;

    if ((f = fopen(nom, "r")) == NULL) // ouverture du fichier en lecture
        cout << "erreur d'ouverture du fichier " << nom << endl;
    else
    {
        while (fscanf(f, "%c", &c)>0) // lecture du fichier caractère par caractère
            putchar(c); // affichage caractère par caractère
            // putchar(c) == fputc(c, stdout) c-à-d écriture sur l'écran !

        fclose(f); // fermeture du fichier
    }
}
```

Et on en a fini avec les fichiers !
Ouf !