République Algérienne Démocratique et Populaire

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique

Département Informatique

Les Fichiers

Cours Algorithmique de 1ere Année MI

Présenté par : Dr. B. BESSAA

Soient A, B, C, D et E cinq abonnés d'un opérateur téléphonique. Les numéros de ces abonnés sont:

A: 05 56 88 65 17
B: 04 39 56 88 71
C: 05 18 81 78 20
D: 05 36 55 78 27
E: 05 29 92 87 72

Alors maintenant, préparez vous, je vais vous poser quelques questions. Vous êtes prêt ?

Oui, nous sommes prêt.

Mais ne trichez pas !

D'accord.

Alors, première question:
Quel est le nombre de répétitions du chiffre 5 dans chaque numéro ?

Ehhhh, est-ce qu'on peut revenir en arrière ?

Non, j'ai dis ne trichez pas !

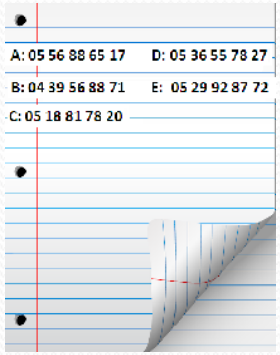
Mais monsieur, vous aurez dû nous avertir ! C'est une question qui nécessite la mémorisation des numéros.

Et vous pensez pouvoir retenir les cinq numéros en quelques secondes ?

Bon, non, mais on aurait pu les reporter sur une feuille par exemple.

Donc vous confirmer que par fois, votre cerveau (mémoire interne), ne suffit pas, et vous avez besoin d'un support physique (mémoire externe).

BESOIN



Revenons à nos algorithmes.

Depuis le début du cours d'algorithmique, on a vu plusieurs types de variables, types de base (entier, réel, caractère, booléen) et types structurés (tableau, chaine, enregistrement).

Le problème de tous ces types est que, après avoir fait des traitements, à un certain moment on va mettre fin au programme (algorithme), alors vous avez sans doute remarqué qu'une fois qu'on termine on n'a plus de traces sur nos résultats, et tout ce qu'on à fait est foutu en l'air. Et donc si vous voulez utiliser les résultats, vous devez les reporter sur cette fameuse feuille.

Quelle feuille ?

Celle que vous avez utilisé pour reporter les numéros de téléphone !

Mais les résultats peuvent avoir des tailles importantes, et on ne va pas les écrire tous à la main ?!

Ehhh, feuille oui, mais une feuille électronique et non pas papier !

Vous savez, la mémoire externe, c'est pas c'qui manque !!!, on peut avoir plusieurs types de ces mémoires.





Le problème c'est comment notre programme (algorithme) va pouvoir utiliser cette mémoire.

Pour tout traitement, un algorithme a besoin de variables de types donnés, et ce qu'on a vu jusqu'à présent n'a rien avoir avec ce qu'on est entrain de dire.

Et donc ?

Donc on a BESOIN d'un nouveau type de variable ! C'est le type

FICHIER

Définition

Un fichier est un ensemble d'informations stockées sur un support physique (disque dur, flash disque, …). C'est une structure de donnée permettant de regrouper et de stocker de manière permanente, un ensemble (infini) de variables de même type (simples ou structurés).

Mais monsieur, vous avez dit stockées sur un support physique, donc on va sortir vers une unité externe?!

Exactement, lorsqu'on parle de fichiers, on a cette notion de l'extérieur

Et comment on va gérer ça ?

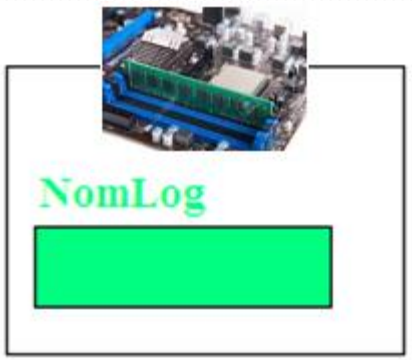
Heureusement, ce n'est pas a vous de gérer. C'est le système d'exploitation (SE) qui va prendre en charge la gestion de (l'extérieur). Tout ce que vous avez à faire est d'informer le SE que vous allez utiliser un espace mémoire externe (fichier).

Mais vous avez dit que FICHIER est un Type, donc il est utilisé par notre algorithme, comment est-ce qu'il est géré par le SE, on a deux fichiers ou quoi?

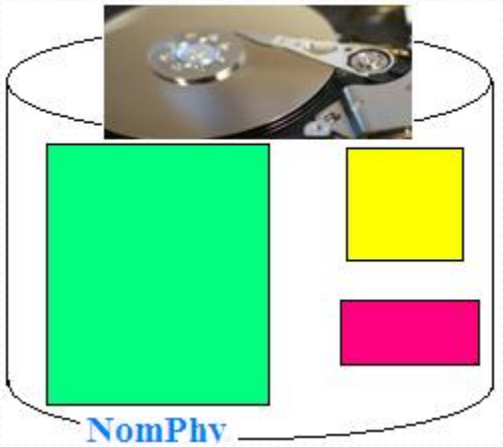
Très bonne remarque

Et ben oui, la particularité de ce nouveau type (Fichier) est qu'il existe dans deux emplacements.

En mémoire centrale



En mémoire externe



Il est utilisé dans un programme (algorithme) comme toute autre variable, il a un NOM qui l'identifie appelé Nom Logique

Il occupe un espace mémoire qui porte son NOM, mais cette fois appelé Nom Physique

Et justement c'est le SE, suite à notre demande, qui va créer un lien entre le Nom Logique et le Nom Physique

C'est ce qu'on appelle l' ASSIGNATION

Et quelle est la différence entre les deux, Logique et Physique ?

Bien, il existe plusieurs différences :

1- Type de mémoire:

La mémoire utilisée pour le fichier logique est volatile (perd les données si elle est hors tension), par contre la mémoire utilisée pour le fichier physique n'est pas volatile (les données sont sauvegardées en permanence même hors tension).

2- La vitesse:

La première (mémoire centrale) est beaucoup plus rapide que la deuxième.

3- La capacité:

En général, la mémoire utilisée pour le fichier logique (appelée mémoire tampon ou Buffer) est beaucoup plus petite que celle utilisée pour le fichier physique, surtout s'il est volumineux.

4- L'organisation:

La première est organisée suivant la logique utilisée dans l'algorithme, par contre la deuxième c'est un magasin de stockage (suite binaire).

Doucement, doucement monsieur, le type, la vitesse c'est clair. Mais Capacité, Organisation ???

Bien, je vais expliquer:

Capacité:

En effet, les données d'un fichier physique ne sont pas présentes entièrement en mémoire centrale, au fait le traitement se fait par paquets (taille du Buffer). Chaque fois qu'on termine avec un paquet, on l'envoie vers la mémoire physique et on traite un nouveau paquet.

Par exemple, si on veut créer un fichier qui contient 1000 multiples de 3. Alors, si on suppose que la taille du buffer permet de stocker 250 entiers, l'opération de création passe par le traitement de 4 paquets.

Logique

0	3	6	9	…	747

Générer 250 multiples (0 … 747)
Transférer le paquet vers le physique
Vider le Buffer

Physique

0	3	6	9	…	747

Doucement, doucemnent monsieur, le type, la vitesse c'est clair. Mais Capacité, Organisation ???

Bien, je vais expliquer:

Capacité:

En effet, les données d'un fichier physique ne sont pas présentes entièrement en mémoire centrale, au fait le traitement se fait par paquets (taille du Buffer). Chaque fois qu'on termine avec un paquet, on l'envoie vers la mémoire physique et on traite un nouveau paquets.

Par exemple, si on veut créer un fichier qui contient 1000 multiples de 3. Alors, si on suppose que la taille du buffer permet de stocker 250 entiers, l'opération de création passe par le traitement de 4 paquets.

Logique

750	753	…	1497

Générer 250 suivants (750 … 1497)
Transférer le paquet vers le physique
Vider le Buffer

Physique

0	3	6	9	…	747
750	753			…	1497

Doucement, doucemnent monsieur, le type, la vitesse c'est clair. Mais Capacité, Organisation ???

Bien, je vais expliquer:

Capacité:

En effet, les données d'un fichier physique ne sont pas présentes entièrement en mémoire centrale, au fait le traitement se fait par paquets (taille du Buffer). Chaque fois qu'on termine avec un paquet, on l'envoie vers la mémoire physique et on traite un nouveau paquets.

Par exemple, si on veut créer un fichier qui contient 1000 multiples de 3. Alors, si on suppose que la taille du buffer permet de stocker 250 entiers, l'opération de création passe par le traitement de 4 paquets.

Logique

1500	1503	…	2247

Générer 250 suivants (1500 … 2247)
Transférer le paquet vers le physique
Vider le Buffer

Physique

0	3	6	9	…	747
750	753	…			1497
1500	1503	…			2247

Doucement, doucemnent monsieur, le type, la vitesse c'est clair. Mais Capacité, Organisation ???

Bien, je vais expliquer:

Capacité:

En effet, les données d'un fichier physique ne sont pas présentes entièrement en mémoire centrale, au fait le traitement se fait par paquets (taille du Buffer). Chaque fois qu'on termine avec un paquet, on l'envoie vers la mémoire physique et on traite un nouveau paquets.

Par exemple, si on veut créer un fichier qui contient 1000 multiples de 3. Alors, si on suppose que la taille du buffer permet de stocker 250 entiers, l'opération de création passe par le traitement de 4 paquets.

Logique

2250	2253	…	2297

Générer 250 suivants (2250 … 2297)
Transférer le paquet vers le physique
Vider le Buffer

Physique

0	3	6 9 …	747
750	753	…	1497
1500	1503	…	2247
2250	2253	…	2297

Et comme on l'a fait pour 1000 multiples, on peut le faire pour 1 million, pour 1 milliard… La seule limite c'est l'espace physique disponible en mémoire secondaire. Et c'est pour ça qu'on a dit que la taille du type Fichier est infinie (théoriquement).

Et ce qu'on a fait pour transférer les données de la mémoire centrale vers la mémoire externe, on peut le faire dans le sens inverse. Si on veut faire des traitements sur des données existantes, on va les transférer paquet par paquet, et chaque fois qu'on termine le traitement avec un paquet on le remet dans le fichier physique, et on charge le suivant vers la mémoire centrale et ainsi de suite jusqu'à ce qu'on termine le traitement.

Juste une remarque

L'opération de transfert de données entre la mémoire centrale (Buffer) et la mémoire externe est assurée par le SE. Les actions du programme (algorithme) qui traitent les données manipulent le fichier logique (Buffer) et n'ont rien avoir avec la le fichier physique .

Ca c'est pour la Capacité, et pour l'Organisation ???

Pour l'organisation

Alors le fichier physique ce n'est qu'un conteneur dans lequel on stocke les données. Il n y a aucune organisation des données, c'est une suite binaire (0/1).

Exemple: Un fichier physique de taille 4 Octets créé par un programme donné

01000010010101110100000000000000

Que représente ce contenu ? Aucune idée. Même le SE (celui qui à fait le transfert de la mémoire centrale à la mémoire externe) ne le sait pas.

Et qui peut le savoir alors ?

Seul celui qui a créé le fichier peut savoir de quoi il s'agit. Car c'est lui seul qui connait la Logique suivie lors de la création et donc il peut faire une interprétation de ces données.

Alors on peut dire qu'avec des logiques différentes on peut aboutir au même contenu physique ?!

Exactement !!! Prenons comme exemple le fichier physique précédent.

01000010010101110100000000000000

Alors le fichier logique ?

Un premier créateur du fichier dit :

Moi j'ai crée un fichier d'entier de 16 bits chacun.

Donc pour cette logique, le fichier contient deux entiers

0100001001010111 **0100000000000000**

En décimal, il s'agit de : 16983 et 8192

Un deuxième créateur du fichier dit :

Moi j'ai crée un fichier de réel sous la norme IEEE754 simple précision.

Donc pour cette logique, le fichier contient un seul réel représenté sur 32 bits (1 bit de signe, 8 bits pour l'exposant et 23 bits pour la mantisse).

Si vous maitrisez toujours la Représentation de l'Information (CRI) vous allez voir qu'en décimal, il s'agit du nombre : 53.8125

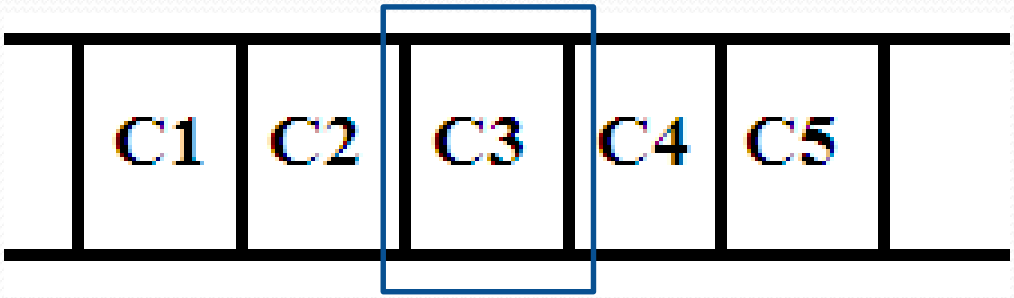
Donc voilà, lorsqu'on parle de l'organisation du fichier on prétend le fichier logique et non pas le physique.

Et justement, suivant cette organisation, on défini deux types de fichiers:

1- Les Fichiers non typés

Dans ce type de fichier, l'organisation est basée sur le caractère. Pou cela, on les appelle aussi des Fichiers Texte.
Indépendamment de la sémantique des données, le contenu de ce type de fichiers est une suite de caractères (imprimables). Donc chaque élément du fichier est un caractère (physiquement: en bits c'est équivalent à 1 octet = 8 bits).

Un élément du fichier



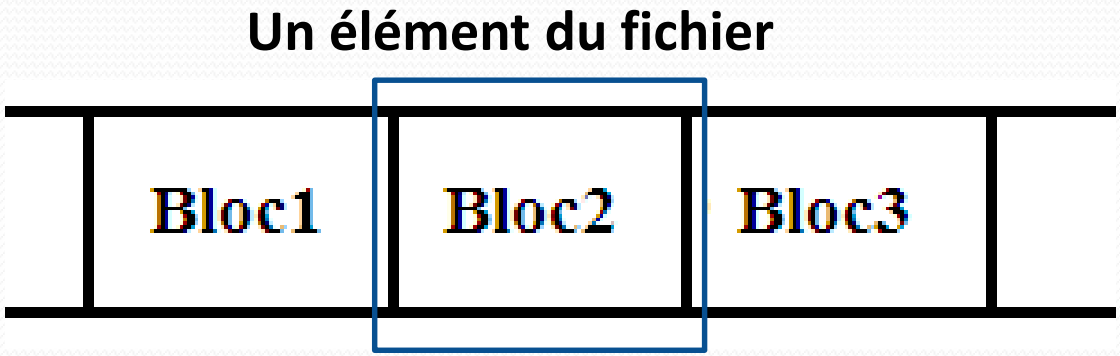
Exemple : Si on prend un réel X= 2.213264125833351E-58

Alors le fichier qui va contenir ce nombre sera composé de 20 caractères
Sa taille sera donc 20 Octets, soit 160 bits.

2	.	2	1	3	2	6	4	1	2	5	8	3	3	5	1	E	-	5	8

2- Les Fichiers typés

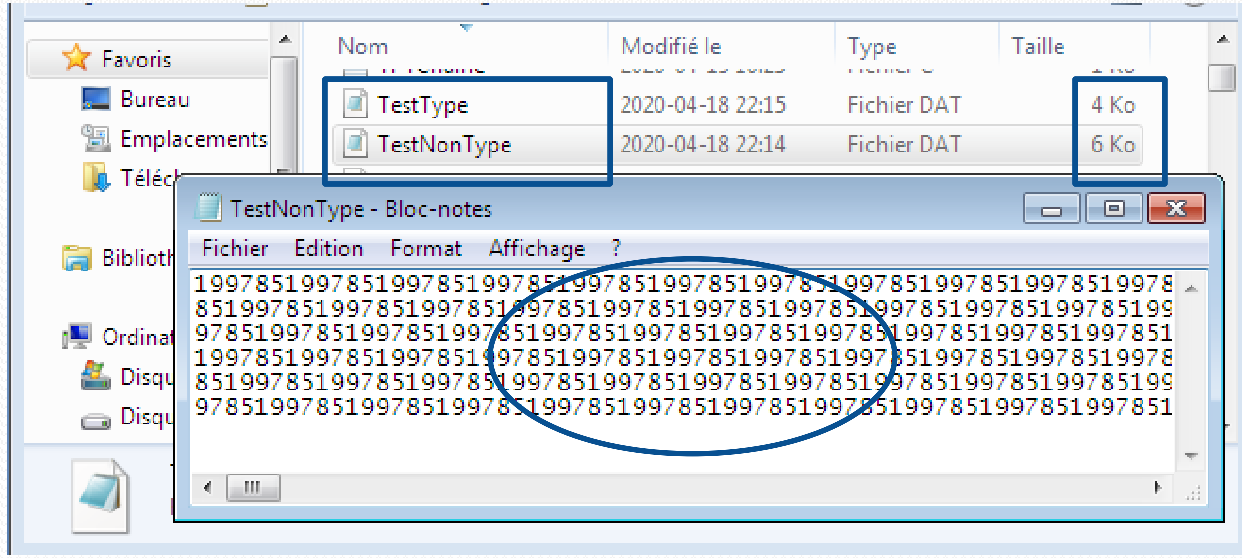
Dans ce type de fichier, l'organisation est basée sur le bloc de données. Chaque bloc a un type qui peut être simple (entier, reel,…) ou structuré (enregistrement,…). On les appelle aussi des Fichiers Binaires, car dans ce cas les données sont stockées en utilisant leurs représentations binaires et la taille d'un élément dépend de la taille de sa représentation.



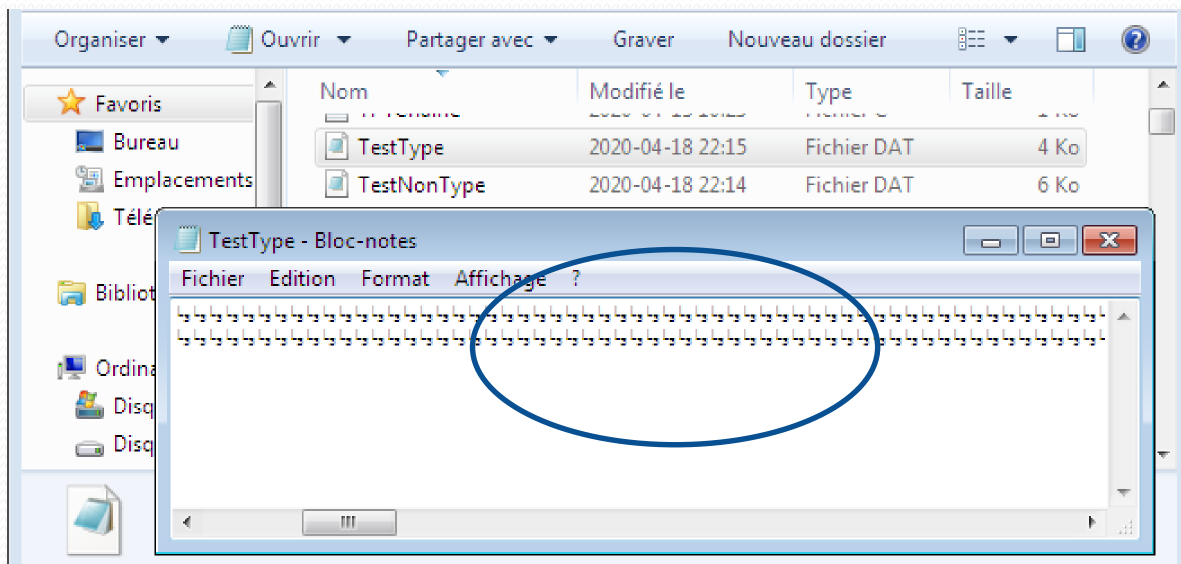
Exemple fichier de réel:
Si on prend le même réel
X= 2.213264125833351E-58
La taille du fichier sera
4 Octets soit 32 bits

Voilà un exemple : Résultat d'un programme qui crée un fichier contenant 1000 fois le nombre X = 199785 avec les 2 types de fichiers

```c
#include<stdio.h>
#include<stdlib.h>
main()
{ FILE *F=NULL;
  F=fopen("TestNonType.dat","w");
  int i,x=199785;;
  for (i=1;i<=1000;i++)
     fprintf(F,"%d",x);
  return 0;
}
```




```c
#include<stdio.h>
#include<stdlib.h>
main()
{ FILE *F=NULL;
  F=fopen("TestType.dat","wb");
  int i,x=199785;;
  for (i=1;i<=1000;i++)
     fwrite(&x,sizeof(int),1,F);
  return 0;
}
```



Donc si on prend un Fichier Typé où le type du bloc est un caractère il sera identique au Fichier Texte , Non ?

Exactement. Oui, un fichier typé de caractères est identique à un fichier texte, donc l'utilisation des fichiers typés est plus intéressante puisque elle regroupe les deux.

Passons maintenant à un autre problème.

Supposons qu'on a créé un fichier contenant les informations des étudiants de la filière MI. Une fois qu'on termine, on veut avoir les information d'un étudiant X, comment nous allons accéder à cette information ?

C'est simple, nous allons faire une recherche dans ce fichier.

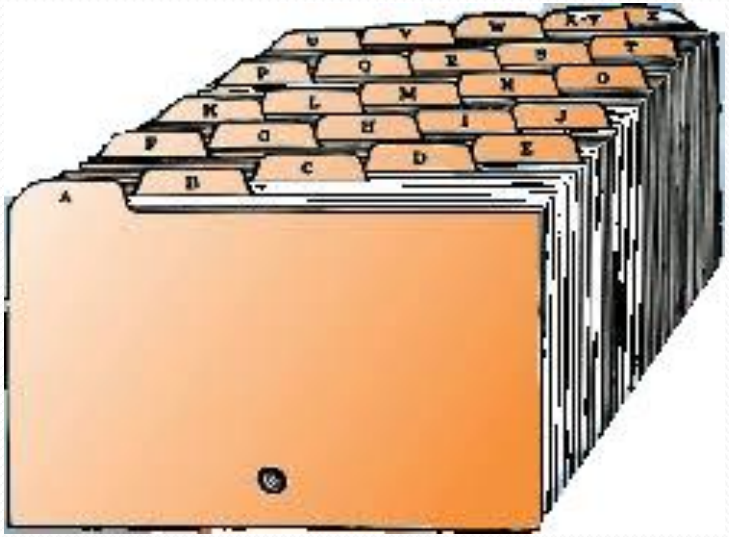
Ok, je vais donner un exemple

Alors, supposons que votre fichier se présente de cette manière

comment vous allez procéder ?

Et ben on va parcourir fiche par fiche jusqu'à ce qu'on trouve l'étudiant X.

Bien, maintenant je présente le fichier d'une autre façon



Là, c'est plus simple, on va directement à la fiche X

Je donne une troisième alternative, le fichier se présente de cette façon

comment vous allez procéder ?

Dans ce cas on accède au paquet X, ensuite on parcours fiche par fiche jusqu'à ce qu'on trouve notre étudiant.

Très Bien, vous êtes expert.

A partir de cet exemple, on voit que la manière de retrouver une information dans un fichier dépend de la présentation.

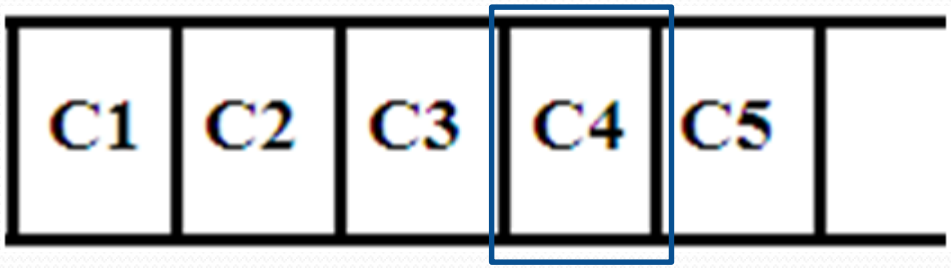
Ceci nous mène à donner une autre classification des fichiers qui se base sur le Mode d'Accès.

Alors suivant le mode d'accès, on distingue trois types de fichiers:

1- Les Fichiers à Accès Séquentiel

L'accès séquentiel consiste à traiter les informations séquentiellement, c'est à dire dans l'ordre où elles apparaissent dans le fichier. Dans ce type de fichier, pour accéder à une information, il faut parcourir toutes celles qui la précède.

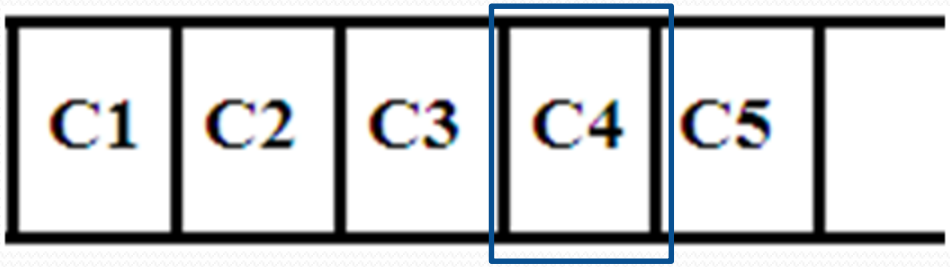
Par exemple pour accéder à C4, on doit passer par C1, C2 et C3.



2- Les Fichiers à Accès Direct

L’accès direct consiste à se placer directement sur l'information souhaitée sans parcourir celles qui la précèdent, en précisant la position de l'élément recherché. L'indication d'un numéro permet donc un accès direct et rapide à l'information ainsi référencée.

Par exemple pour accéder à C4, il suffit de donner son numéro.



3- Les Fichiers à Accès Séquentiel Indexé

Ce type d'accès 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. Le principe est de créer des fichiers supplémentaires d'index.

Dans la suite de ce cours, nous allons nous intéresser aux Fichiers à Accès Séquentiel

Un fichier à accès séquentiel peut être vu comme un ruban de longueur infinie.

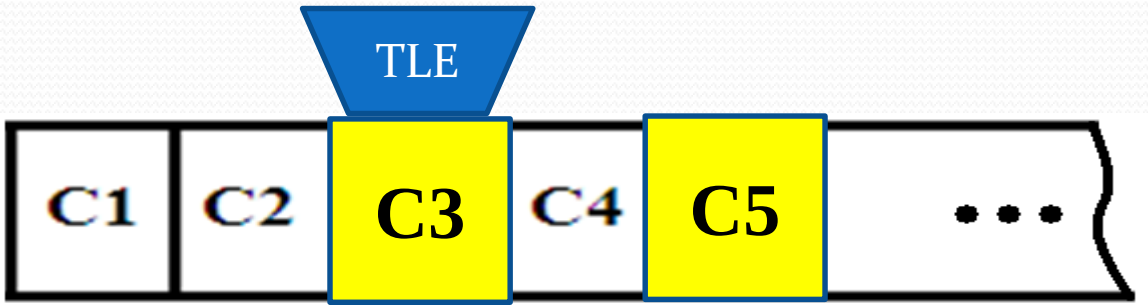
La lecture et l'écriture sur ce fichier est réalisée par une tête de lecture-écriture (TLE) permettant de lire ou écrire un seul élément à la fois.

L'élément pointé par la tête de lecture-écriture (TLE) est appelé Elément Courant

Et comme tout fichier, il possède une marque de Fin De Fichier (FDF)

L'ajout d'un élément ne peut se faire qu'en fin de fichier

L'insertion et la modification ne sont pas permises dans ce type de fichier



Maintenant il ne reste plus qu'à voir comment utiliser le type fichier dans un Algorithme

EeeeeeeN FIN?

Eeeeeeeeeet Oui. Je sais c'était un peu long, mais il faut bien comprendre le principe des fichiers. Maintenant, si vous avez bien compris, tout ce qui reste est Trèèèèès Facile.

Déclaration d'un fichier

C'est simple, un fichier est déclaré en donnant son Nom et le Type de ses éléments, en utilisant un mot clé spécifique: Fichier

<NomLog> : Fichier de <TypeElt>;

<NomLog> : c'est un identificateur du Nom Logique du fichier.
<TypeElt> : est le type des éléments du fichier, il peut être simple ou structuré

Exemple:

Fent : Fichier de entier; F1 : Fichier de reel; Fcar: Fichier de caractere;
Fetud : Fichier de TEtudiant; où TEtudiant est un type Enregistrement

Primitives de manipulation des fichiers

Chaque fois qu'on défini un nouveau type, il faut préciser les opérations qu'on peut faire avec ce type. Et bien avec le type fichier, on peut :

1- **Assigner un fichier.**
2- **Ouvrir un fichier.**
3- **Lire ou Ecrire dans un fichier.**
4- **Fermer un fichier.**

1- Assignation

Vous vous souvenez de ce mot, ce fameux lien entre le Logique et le Physique.

Alors cette opération permet de donner le nom physique du fichier où on va stocker nos données. Son exécution ne fait rien, c'est juste une information que le SE va utiliser lorsqu'il commence ses traitements.

Sa syntaxe est :

Assigner(<NomLog>,<NomPhy>);

<NomLog> : c'est l'identificateur du fichier déclaré dans la partie déclaration.

<NomPhy> : est une chaine da caractère représentant le nom physique du fichier. Elle peut contenir, éventuellement, le chemin complet sur l'unité de stockage.
Cette chaine peut être une constante, comme elle peut être une variable de type chaine.

Exemple

Assigner(Fent,'FichierEntier'); Assigner(Fcar,'Caractere.Dat');
Assigner(FEtud,'C:\Cursus\fiches\Etudiant.MI');
Assigner(F1,'Nombre.Dat');
Chemin ⟵ 'E:\resultats\Notes.dat'; Assigner(Fres,Chemin);

2- Ouverture

Pour pouvoir exploiter une fichier, il faut qu'il soit ouvert. Un fichier est ouvert ou bien pour lire ou écrire des données. Donc on a deux modes d'ouverture.

2.1- Ouverture en Mode Lecture

Sa syntaxe est:

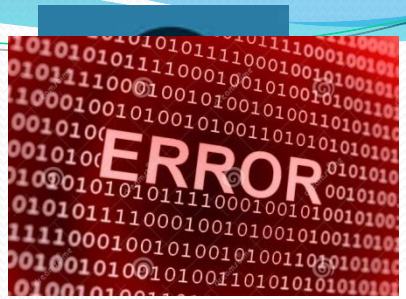
Relire(<NomLog>); Exemple : Relire(Fcar);

L'exécution de cette action fait intervenir le SE et déclenche une suite d'opérations:

- A l'aide de l'action Assigner(<NomLog>,<NomPhy>), le SE lance un avis de recherche sur la mémoire externe d'un emplacement qui s'appelle <NomPhy>

Deux résultats sont possibles :

1- Le fichier n’existe pas !



Le système déclenche une exception,
et revoie un message d’erreur

2- Le fichier existe

Le système ouvre le fichier.

et positionne la TLE sur le premier élément.

2.1- Ouverture en Mode Ecriture

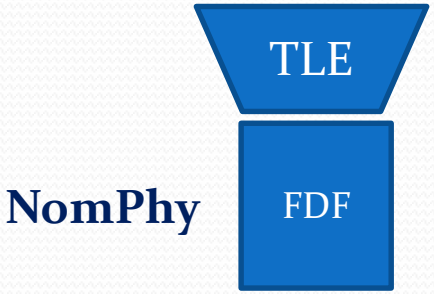
Sa syntaxe est:

Reecrire(<NomLog>); Exemple : Reecrire(Fcar);

De même pour cette action, son exécution fait intervenir le SE et à travers l'assignation, déclenche une recherche du fichier <NomPhy> :

Là aussi, deux résultats sont possibles :

1- Le fichier n'existe pas



Le système crée un fichier vide (ne contient que la marque FDF) avec le nom NomPhy
La tête TLE sera sur la marque FDF

2- Le fichier existe

Le système ouvre le fichier.

EFFACE toutes les données (le fichier devient vide) et positionne la TLE au début.

Remarque:

Il existe une autre syntaxe qu'on peut utiliser, elle regroupe les deux modes (lecture et écriture). Sa syntaxe est:

Ouvrir(<NomLog>,<Mode>);

<Mode> est un caractère qui défini le mode d'ouverture.

Mode Lecture : 'L', Exemple : Ouvrir(Fcar,'L'); équivalent à Relire(Fcar);

Mode Ecriture : 'E', Exemple : Ouvrir(Fcar,'E'); équivalent à Reecrire(Fcar);

3- Lecture ou Ecriture

Se sont deux opérations d'entrée/sortie.

3.1- Lecture

Sa syntaxe est:

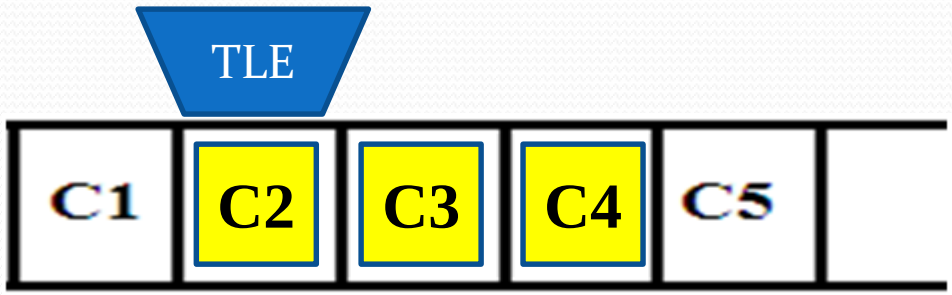
Lire(<NomLog>,<idVar>); Exemple: Lire(Fcar,X);

Cette opération s'exécute sur un fichier ouvert en mode Lecture. Elle permet de lire l'élément courant (pointé par la Tête TLE) et le met dans la variable <idVar> qui doit être du même type que celui des éléments du fichier, puis déplace la TLE vers l'élément suivant.

Exemple : Soit un fichier ouvert, où la TLE est sur C2,
et soient deux variables X et Y

Lire(Fcar,X); X

Lire(Fcar,Y); Y



3.2- Ecriture

Sa syntaxe est:

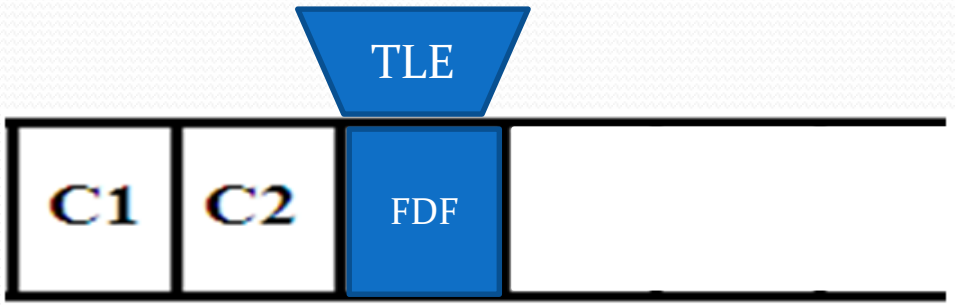
Ecrire(<NomLog>,<idVar>); Exemple: Ecrire(Fcar,X);

Cette opération s'exécute sur un fichier ouvert en mode Ecriture. Elle permet de d'écrire le contenu d'une variable <idVar>, qui doit être du même type que les éléments du fichier, dans le fichier NomLog.

L'écriture se fait toujours à la fin du fichier. Donc on déplace la marque FDF d'un pas, ce qui crée un espace vide qui va recevoir l'élément ajouté.

Exemple : Soit un fichier ouvert en écriture contenant déjà deux éléments C1 et C2. Et soit X une variable contenant C3. L'ajout de X se fait de cette manière.

Ecrire(Fcar,X); X | C3 |



3- Fermeture

C'est simple, une fois qu'on termine les traitements avec un fichier, il faut le fermer. Sa syntaxe est:

Fermer(<NomLog>); Exemple: Fermer(Fcar);

Cette opération s'exécute sur un fichier ouvert en mode Lecture ou Ecriture. Elle permet de fermer le fichier NomLog qui correspond physiquement à NomPhy. Une fois fermé, aucun traitement sur le fichier ne sera possible.

Fermer(Fcar);

Remarque

La fermeture d'un fichier ne touche pas à l'assignation, le lien entre NomLog et NomPhy existe toujours, et on peut ré-ouvrir le fichier sans renouveler l'assignation.

La Marque de Fin De Fichier (FDF)

Lorsqu'on parcourt un fichier ouvert en Lecture, on a besoin de savoir si on a atteint la fin ou non. Donc, dans un algorithme, on a besoin d'une action qui nous donne cette information. Justement, c'est la fonction FDF().

Sa syntaxe est :
FDF(<NomLog>); Exemple : FDF(Fcar);

FDF() est égale à VRAI si la TLE est sur une marque FDF et FAUX sinon.

Remarque

- Si FDF() renvoie Faux juste après l'ouverture Alors le fichier est VIDE Fsi;
- La fonction FDF() n'est utilisée qu'avec des fichiers ouverts en LECTURE

Exercice

Soient File1 et File2 deux fichiers de chaînes de caractères. Chaque chaîne représente un mot. Ecrire un algorithme qui construit un fichier File3, tel que File3 contient les mots de File1 qui n'existent pas dans File2.

Solution

Donc on a deux fichiers dont les éléments sont des chaines (type éléments).

On veut construire (créer) un troisième fichier qui va contenir les mots (éléments) de File1 qui n'existent pas dans File2.

Ce problème est bien connu, on l'a déjà vu avec les tableaux.

Il consiste à parcourir tout le fichier File1 (jusqu'à son FDF).

Pour chaque élément lu, (Lecture) on fait une recherche de cet élément dans File2 (Lire un élément puis comparer), si on le trouve, on arrête la recherche et on passe à l'élément suivant de File1.

Si on atteint la fin FDF de File2 sans le trouver on le met (Ecrire) dans le fichier File3.

```
Algorithme Mot123;
Var  F1,F2,F3:Fichier de chaine[30];
  X,Y:Chaine[30];
  Trouve:booleen;
Debut
  Assigner(F1,'File1');
  Assigner(F2,'File2');
  Assigner(F3,'File3');
  Relire(F1); Reecrire(F3); //Ouvrir F2 en Lecture et F3 en Ecriture
  Tantque Non FDF(F1)
  Faire
      Lire(F1,X); //Lire un mot de F1
      Trouve ← Faux; //on suppose que le mot n'existe pas dans F2
      Relire(F2); //Ouvrir F2 en Lecture et revenir à chaque itération au début du fichier F2
      Tantque Non FDF(F2) Et Non Trouve
      Faire
          Lire(F2,Y); //Lire un mot de F2
          Si Y=X Alors Trouve ← Vrai Fsi;  //on arrête la recherche
      Fait;
      Si Non Trouve Alors Ecrire(F3,X) Fsi;   //si on ne trouve pas on le met dans F3
      Fermer(F2);
  Fait;
  Fermer(F1); Fermer(F3);
Fin.
```

brbessaa@gmail.com