CM2 : Données et fichiers

Les données numériques sont une des composantes essentielles des programmes informatiques.

- Il s'agit par définition des *informations qui doivent être conservées entre deux exécutions*.
- Avec l'augmentation des capacités de stockage et leur mise en réseau, les quantités de données conservées ont considérablement augmenté au cours des dernières décennies.

Dans le cadre de ce cours, nous aborderons :

- la question du stockage de ces données sur un support informatique (Fichiers, bases de données),
- ainsi que les méthodes permettant de consulter et mettre à jour régulièrement ces données.

1. Généralités

1.1 Production des données

Tout commence par une fiche à remplir...

- Un formulaire se présentant sous la forme d'un ensemble de rubriques à remplir.
- Le **modèle de fiche** définit le format des données à enregistrer:
  - liste des rubriques du formulaire,
  - domaine de valeurs attendues dans chaque rubrique.
- A toute fiche remplie correspond un **jeu de valeurs** (ou mesure) :
  - liste de valeurs correspondant au contenu d'une fiche particulière.

Un jeu de valeurs sert à décrire :

- une personne réelle : assuré, client, étudiant, auteur, plaignant, contribuable,...
- une personne morale : fournisseur, prestataire, direction, section, promotion,...
- un objet physique : article, véhicule, composant, ingrédient, pièce,...
- un objet contractuel ou administratif : fiche de paye, contrat, procès verbal,

billet, dossier...
- un lieu : salle, local, manufacture, entrepôt, ligne de production,...
- un événement : transaction, jugement, achat, vente, acte administratif, appel téléphonique,...
- etc...

1.2 Stockage des données

- D'un point de vue informatique, un jeu de valeurs recueilli est appelé un *enregistrement*
  - correspondant à l'encodage des données recueillies sur un support numérique
- Une **structure de données** définit les données de manière logique,
  - c'est à dire l'ordre dans lequel elles doivent être lues
  - et la manière dont elles doivent être interprétées par les programmes qui les utilisent.

Exemples de structures de données (cours d'algorithmie):

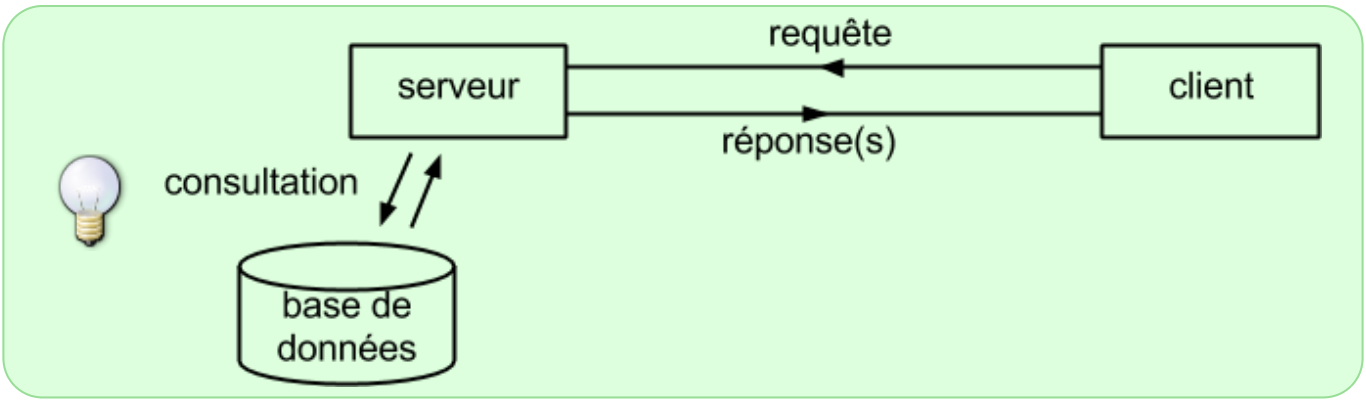
- listes,
- listes de listes,
- dictionnaires,
- arbres,...

- Mais l'encodage comprend également et principalement la **structure de stockage**,
  - définissant la manière dont les données sont physiquement stockées sur le support,
  - autrement dit la méthode de **rangement** de la série de mesures :
    - fichiers,
    - répertoires,
    - index,
    - etc...
  - reposant sur des supports de stockage (ou mémoires) :
    - mémoire centrale,
    - mémoires secondaires (disque dur, CD-ROM, memoire flash (SSD), etc...).

1.3 Requêtes

En informatique, une requête (en anglais query) est une demande de consultation, effectuée par un programme *client* à l'attention d'un programme *serveur*.

- Le programme **client** représente l'utilisateur, il s'agit du programme qui enregistre la demande de l'utilisateur, la transmet au serveur, puis met en forme visuellement la réponse du serveur.
- Les données sont centralisées au niveau du **serveur**, chargé de la gestion, de la manipulation et du stockage des données.Il traite la requête, consulte les données et transmet le résultat au client.



La requête peut être une simple référence vers un fichier, ou être l'expression d'une recherche plus spécifique (consultation de certaines fiches d'un fichier, croisement d'information (entre plusieurs fichiers), etc...). Dans ce cas, il est nécessaire d'utiliser un langage de requête (le plus souvent SQL).

On distingue quatre grands types de requêtes (approche "CRUD"):

- **Création** (***C****reate*) : ajout de nouvelles données dans la base
- **Lecture/recherche** (***R****ead*) : consulation du contenu de la base
- **Mise à jour** (***U****pdate*) : changement du contenu existant
- **Suppression** (***D****elete*) : suppression des données obsolètes

Lors d'une consultation de type lecture/recherche, il y a souvent plusieurs réponses qui correspondent à la demande. Le résultat d'une requête prend donc la forme d'un ensemble de réponses. Ces réponses sont éventuellement classées, selon la valeur d'un certain identifiant, ou selon le degré de pertinence.

Exemples :

- requêtes http : demande de consultation d'une page web (= référence vers un fichier)
- moteur de recherche : recherche de pages contenant les mots-clés spécifiés
- bases de données : utilisation d'un langage de requête :

```
SELECT *
FROM  Eleves
WHERE NOM = 'Dugenou'
```

1.4 Transport et flux de données

- La transmission des données entre programmes nécessite l'ouverture d'un canal de communication
  - entre client et serveur
  - par lequel transitent les données (les requêtes et les réponses).
- Le transport est géré
  - par le système d'exploitation (lorsque les données transitent au sein d'un même ordinateur)
  - ainsi que par des routeurs (lorsque les données transitent d'un ordinateur à l'autre sur le

réseau).

- Au niveau du client,
  - les réponses en provenance du serveur sont organisées sous la forme d'une liste,
  - qu'on appelle un **flux de données**.

La notion de flux de données signifie que les réponses sont lues dans un ordre fixe, telles qu'elles ont été écrites au niveau du serveur. On parle de lecture à accès **séquentiel** (par opposition à la lecture à accès aléatoire).

1.5 Analyse des données

L'analyse des données a pour but de recomposer l'information contenue dans les données recueillies afin d'en fournir une vue plus synthétique, ou d'extraire des informations qui n'apparaissent pas de façon explicite dans la série de mesures initiale :

REPRESENTATION DES DONNEES : Représenter des jeux de valeurs de grande taille de façon plus synthétique (algorithmes de réduction de dimension)

REGROUPEMENT ("CLUSTERING") : Définir des regroupements (ou des classements simples ou hiérarchiques) entre jeux de valeurs

COMPLETION : Méthodes de classification automatique (ou d'interpolation) visant à deviner soit la classe, soit certaines valeurs non mesurées, à partir d'un jeu de valeurs et d'une base d'exemples complets. Il existe des méthodes paramétriques ou non paramétriques.

ESTIMATION ET DECISION : Méthodes visant à estimer la "valeur" associée à un jeu de données (pour l'aide à la décision)

2. Données et fichiers

2.1 Types de données

On distingue classiquement deux grandes catégories de données :

- données **quantitatives**:
  - numérique entier ou réel, discrètes ou continues, bornées ou non. ex: poids, taille, âge, taux d'alcoolémie,…
  - temporel : date, heure
  - numéraire
  - etc…
- données **qualitatives**:
  - de type vrai/faux (données booléennes). ex: marié/non marié, majeur/non majeur
  - de type appartenance à une classe. ex: célibataire/marié/divorcé/veuf, salarié/chômeur/retraité etc…
  - de type texte (autrement dit "chaine de caractères"). ex: nom, prénom, ville,…

Parfois, la distinction entre quantitatif et qualitatif n'est pas nette:

- Possibilité d'ordonner les chaînes de caractères (de type nom/prénom) par ordre alphabétique
- On peut "traduire" une chaîne de caractères en entier (son code ASCII)
- Les valeurs logiques vrai/faux sont souvent traduites en valeurs "entières" 1/0
- Plus subtil : il est possible de définir des ordres partiels, des hiérarchies sur des données qualitatives :
  - regroupement par secteur géographique
  - notion de distance (géographique et temporelle) entre catégories
- Inversement, il est possible de rendre qualitatif des données quantitatives en réalisant des classes. Exemple : tranches d'âge 0-25/25-35/35-55/+55, mineur/majeur, recalé/passable/AB/B/TB etc...

2.2 Représentation informatique des données

D'un point de vue informatique, il n'existe pas de distinction entre le quantitatif et le qualitatif. **Tout est valeur numérique**.

Une valeur numérique (digitale) consiste en:

- un champ de bits (dont la longueur est exprimée en octets)
- un processus de décodage : le **format** (ou type)

- Les données manipulées par un programme:
  - sont codées sous un format binaire,
  - correspondant à un des types informatiques que vous connaissez déjà :
    - entier,
    - chaîne de caractères,
    - réel simple ou double précision etc...
  - ce qui implique en particulier :
    - une précision finie,
    - éventuellement des contraintes de place (le nom ou le prénom ne pouvant pas dépasser n caractères, les entiers pouvant être choisis sur un intervalle fini etc...).

Types de données standards

Les principaux types de données SQL sont les suivants :

- `CHAR (longueur)` : Ce type de données permet de stocker des chaînes de caractères de longueur fixe. `longueur` doit être inférieur à 255, sa valeur par défaut est 1.
- `VARCHAR(longueur)` : Ce type de données permet de stocker des chaînes de caractères de longueur variable. `longueur` doit être inférieur à 2000, il n'y a pas de valeur par défaut.
- TEXT : un texte de longueur quelconque

- `INTEGER` : entier (codage classique su 4 octets)
- `NUMBER(longueur)` : entier naturel, `longueur` précise le nombre maximum de chiffres significatifs stockés
- `DECIMAL (longueur,[précision])` : Le type de données decimal peut stocker jusqu'à 38 chiffres pouvant tous se trouver à droite de la virgule décimale. Il stocke une représentation exacte du nombre décimal, sans aucune approximation de la valeur stockée. `longueur` précise le nombre maximum de chiffres significatifs stockés (par défaut 38), `précision` donne le nombre maximum de chiffres après la virgule (par défaut 38), sa valeur peut être comprise entre -84 et 127. Une valeur négative signifie que le nombre est arrondi à gauche de la virgule.
- `FLOAT` : nombre à virgule flottante simple précision
- `DOUBLE` : nombre à virgule flottante double précision
- `DATE` : ce type de données permet de stocker des valeurs de type date. Format : `'2005-12-12'` (12 décembre 2005)
- `TIME` : ce type de données permet de stocker des valeurs de type heure. Format : `'10:45:02'` (10 h 45 min 2 s)
- `DATETIME` : donnée de type date-heure. Format : `'2005-12-12 10:45:02'` (12 décembre 2005, 10 h 45 m 02 s)
- `TIMESTAMP` : donnée de type date-heure (correspond à un stockage plus 'compact' que le type `DATETIME`)

2.3 Structures de données

- Les types des valeurs étant déterminés (selon les cas de taille fixe ou variable),
  - la **structure de données** correspond au "véhicule" qui servira à transporter et échanger les données (entre programmes, entre ordinateurs).
  - Différentes structures de données sont possibles pour l'encodage et le stockage d'un jeu de valeurs, voir :
    - Données non structurées
    - Données vectorielles
    - Tuples
    - Données structurées
    - Données Hiérarchisées

Le Tuple est la structure de données de base servant pour le recueil, le transport et le stockage des données.

- Un **Tuple** est une liste, **finie**, **ordonnée** et **de taille fixe** contenant une suite de valeurs.
- Chaque valeur peut obéir à un format différent
- On note *m* la taille du tuple (nombre de valeurs)

$$ t = (a_1, ..., a_m) $$ **Exemple :**

```
("Dubois", "Martine", 22, "29/10/1994", "Orléans")
```

2.3.1 Données non structurées

- Textes,
- comptes rendus,
- série de notes, de valeurs sans format précis :

–> format texte en général.

Le format `txt` désigne des données non structurées de type “texte” regroupant différents modes d’encodage :

- ASCII (caractères sans accent),
- utf8 (caractères accentués et spéciaux),
- …

2.3.2 Données vectorielles

- Chaque jeu de valeurs est codé sous la forme d’un **vecteur**
  - constitué de grandeurs entières ou réelles :
  - toutes les valeurs sont quantitatives (numériques).
- C’est un encodage adapté aux grandeurs physiques :
  - chaque champ du formulaire est codé dans un format numérique
  - il est possible de représenter les données dans un espace vectoriel

Un **vecteur** est une séquence (ordonnée et finie) de valeurs quantitatives, chaque valeur obéissant au même format numérique.

Exemple : Considérons une station météorologique enregistrant à intervalle réguliers les données de ses capteurs :

- thermomètre,
- baromètre,
- hygromètre
- et anémomètre.

Un jeu de valeurs sera constitué de *5 réels double précision* décrivant

- la température,
- la pression,
- l’humidité,
- la vitesse
- et la direction du vent.

2.3.3 Tuples

Données organisées sous la forme d'une liste de valeurs qualitatives ou quantitatives. Le tuple est la structure de base servant pour la transmission des données (simple, facile à coder et à échanger).

Un **tuple** est une séquence (ordonnée et finie) de valeurs, chaque valeur pouvant être de type qualitatif ou quantitatif, et pouvant obéir à un format numérique différent.

Exemple : considérons une fiche servant à décrire un étudiant. L'étudiant doit remplir les rubriques nom, prénom et âge, numero de voie, nom de la voie, code postal, ville. Chaque rubrique correspond à un composant d'un 7-tuplet tel que:

- les composants 1, 2, 5 et 7 sont des chaînes de caractères
- les composants 3, 4 et 6 sont des entiers

Le format csv - "Comma Separated Values"

Chaque enregistrement est codé sur une ligne, chaque valeur étant séparé par un caractère séparateur (virgule, point-virgule, tabulation,...).

Exemple :

```
Dubois,Martine,"28, rue des Lilas, 45000 Orléans",45
```

Remarques :

- les données sont des chaînes de caractères
- les guillemets sont nécessaires lorsque la valeur contient le caractère séparateur (ici la virgule)
- les noms des attributs sont éventuellement indiqué sur une ligne séparée

2.3.4 Données structurées

- Données organisées sous la forme d'une liste d'attributs.
  - Chaque attribut est défini par un nom et un format (**type**).
  - Chaque valeur est stockée sous la forme d'un couple (attribut : **valeur**).

Exemple :

Considérons une fiche servant à décrire un étudiant. L'étudiant doit remplir les rubriques nom, prénom et âge, numero de voie, nom de la voie, code postal, ville.

Chaque rubrique correspond à un attribut, où:

- nom, prenom, voie, et ville sont des attributs de type chaîne de caractères
- age et numero et code_postal sont des attributs de type entier

La structure de données sous-jacente est le **dictionnaire**, où l'attribut est la clé permettant d'accéder à la valeur.

Un **dictionnaire** est une liste non ordonnée de valeurs, chaque valeur étant associée à une clé unique (ici la clé est le nom de l'attribut).

Le format json - JavaScript Object Notation

Exemple :

```
{"nom" : "Dubois", "prénom" : "Martine", "adresse" : "28, rue des Lilas, 45000, Orléans", "âge" : 45}
```

Remarques :

- reprend la syntaxe vue en Python
- données numériques ou chaînes de caractères

2.3.5 Données Hiérarchisées

- Organisation des données correspondant à une structure d'**arbre**.
- Dans le cas d'un recueil de données, correspond à la définition de rubriques et sous-rubriques.

Exemples :

- La rubrique **adresse** peut contenir les sous-rubriques **numero**, **voie**, **code_postal** et **ville**.
- Un document contient des **chapitres**, des **sections**, des **sous-sections** etc...

Formats xml, xhtml, json, ...

Pour les données organisées de manière hiérarchique. Des balises servent à séparer les différents attributs.

Ex :

```
<nom> Dubois </nom>
<prénom> Martine </prénom>
<adresse>
  <num> 28 </num>
```

```
  <voie> rue des Lilas </voie>
  <code postal> 45000 </code postal>
  <ville> Orléans </ville>
</adresse>
<âge> 45 </âge>
```

remarque : le format `json` permet également de définir des hiérarchies

```
{
  "nom" : "Dubois",
  "prénom" : "Martine",
  "adresse" :
  {
    "numero" : 28,
    "voie" : "rue des Lilas",
    "code_postal" : 45000,
    "ville" : "Orléans"
  },
  "âge" : 45
}
```

2.4 Structures de stockage

2.4.1 Trames de données

Un jeu de valeurs encodé et stocké sur un support informatique est appelé un “**enregistrement**”. Un enregistrement, qui contient en principe plusieurs valeurs, obéit à une structure de données de type tuple.

- Les **trames de données** servent au **transport** et à la **conservation** des tuples.
- Une trame peut obéir à un format *textuel* ou *binaire*

Dubois	Martine	29, rue du Verger, Orléans	22
15 octets	15 octets	40 octets	4 octets

2.4.2 Blocs de données

Une série d’enregistrements obéit formellement à une structure de type **liste de tuples** (ou ensemble de tuples), chaque élément de la liste étant un tuple particulier.

- Les blocs de données servent à la conservation des données sur les supports de stockage
- Un bloc de données est un tableau (de taille fixe *n*) contenant des tuples.

Dubois	Martine	29, rue du Verger, Orléans	22
Gilbert	Jonas	8, rue des Fleurs, Blois	23
Dalban	Pierre	13, av. du Général, Privas	22
...	...	...	...
Manoukian	Marianne	55, place des Bleuets, Aubagne	24

n

Nous verrons dans un autre chapitre une description plus formelle d'une série d'enregistrements comme "ensemble d'entités" ou encore "relation".

La difficulté consiste à définir une structure de données permettant de gérer efficacement un tel ensemble (qui peut être de grande taille) stocké sur un disque dur. On parle de structure de stockage. Une telle structure doit permettre : d'ajouter des tuples de supprimer des tuples d'accéder rapidement à un tuple particulier (pour lire son contenu)

2.4.3 Stockage d'un jeu de valeurs

Encodage binaire :

- Définition d'une trame, en général de taille fixe.
- Chaque rubrique occupe un nombre d'octets déterminé, afin que chaque jeu de données occupe la même place en mémoire.
- L'utilisation de trames de taille fixe facilite le stockage et la conservation des données sur les supports magnétiques (disque dur, etc…)
- Les données sont transmises dans un format numérique (type) identique à celui utilisé en mémoire centrale.

Encodage textuel :

- Le jeu de données est codé dans un format descriptif (contenant à la fois les valeurs et une description des données : types, attributs, …).
- Ce format facilite la transmission d'un programme à un autre (format "plat") mais est moins propice au stockage.
- La "sérialisation" est l'opération qui consiste à encoder des données sous la forme d'un texte brut (codage ASCII ou utf8), en "perdant" le moins possible d'information.
- Des exemples de formats textes standards sont donnés en 2.1.3 Structures de données.

2.4.5 Stockage d'une séries d'enregistrements

La structure de base servant à stocker à une série d'enregistrements est le tableau de données à 2 dimensions :

- Tableau de données (data frame) = intitulé de colonnes + liste de lignes
  - Intitulé de colonne = nom de l'attribut
  - une ligne = un tuple

Propriété :

- les tuples sont implicitement ordonnées selon leur numéro de ligne
- il peut y avoir des doublons

Structure sous-jacente : tableau à 2 dimensions (ou matrice de données)

Inconvénient: il s'agit d'une structure statique. Une fois les données sauvegardées sur le disque, il est difficile d'insérer ou de supprimer des éléments.

Les structures de stockage utilisées en pratique sont beaucoup plus sophistiquées et permettent de gérer l'insertion et la suppression de tuples sans déplacer les autres tuples. Ce type de stockage repose sur un découpage d'un tableau de données en "pages" et sur l'indexation des tuples (recherche par "clé"). La complexité de cette gestion est en général laissée à un programme spécialisé (tableur, système de gestion de bases de données).

2.4.6 Stockage sur fichier

Les lignes d'une table correspondent physiquement à des enregistrements (ou tuples). Les tuples sont rangés dans des fichiers, stockés sur le disque dur.

Pourquoi stockées sur le disque dur?

- volume très important des données
- les données doivent survivre à un arrêt du serveur

Volume

Le volume est le support sur lequel sont enregistrées les données. On parle de mémoire secondaire (Disque dur, disquette,CD-ROM, etc...). Un volume est divisé en pistes concentriques numérotées de 0 à n (par ex n = 1024). Chaque piste supporte plusieurs enregistrements physiques appelés secteurs, de taille constante (1 secteur = 1 page).

Page (ou secteur)

Les pages sont les unités de base pour la lecture et l'écriture. une page est une zone contiguë de la mémoire secondaire qui peut être chargée en mémoire centrale en une opération de lecture. Taille standard : une page = 1-2 ko.

La mémoire secondaire est donc organisée comme un tableau de pages : (T[0],...,T[L-1]), où L est le nombre de pages. Chaque page fait m octets. Chaque page peut être libre ou occupée.

2.5 Fichiers et répertoires

La mémoire secondaire n'est pas directement accessible (les programmes n'ont pas la possibilité d'aller écrire directement sur le disque). Le système d'exploitation assure ainsi l'indépendance du programme par rapport à l'emplacement des données, au travers d'instructions d'entrée/sortie spécialisées. Pour que les programmes puissent écrire sur le disque, on introduit des objets intermédiaires : les fichiers. Un fichier est caractérisé par (nom, emplacement (volume, arborescence), droit d'accès, taille,...). Il s'agit d'une entité logique. Tout programme utilisant un fichier passe par le système d'exploitation qui, à partir des informations, détermine la localisation des données sur le support.

A retenir :

- Un fichier est une référence vers un ou plusieurs blocs de données, enregistrés sur un support physique.
- Un fichier est caractérisé par son descripteur, constitué de son nom, son chemin d'accès, ses droits d'accès (lecture/écriture/exécution) selon les utilisateurs, sa position sur le support, sa taille, etc...

- La gestion des fichiers est une des fonctions essentielles des systèmes d'exploitation :
  - possibilité de traiter et conserver des quantités importantes de données
  - possibilité de partager des données entre plusieurs programmes.

Opérations de base :

- *Ouverture* : initialisation d'un **flux** en lecture ou en écriture
- *Lecture* : consultation des lignes l'une après l'autre (à partir de la première ligne), dans l'ordre où elles ont été écrites sur le support
- *Ecriture* : ajout de nouvelles données à la suite ou en remplacement des données existantes

Répertoires Chaque volume possède un numéro appelé le label du volume. Tous les descripteurs de fichiers sont regroupés dans une table des matières appelée Répertoire (Directory).

Remarque : cette table des matières est en fait un fichier dont le descripteur est contenu dans le label du volume.

Organisation hiérarchique :

- Lorsque le nombre de fichiers est élevé, les descripteurs de fichiers sont classés dans plusieurs répertoires, organisés sous une forme arborescente.
- Le répertoire de plus bas niveau hiérarchique est appelé racine → chemin d'accès (path)

2.5.1 Consultation des données : lecture d'un fichier (Read)

Méthode traditionnelle pour le traitement de fichiers de petite taille. La consultation des données

nécessite d'ouvrir une "communication" entre le programme et les fichiers. Ce canal de communication permet de recevoir un flux de données.

Pour établir la communication, il faut connaître : le "chemin d'accès" aux données (disque dur local) l'"adresse" des données (lorsqu'il s'agit de données stockées sur un serveur distant)

L'opération d'ouverture de fichier initialise un descripteur de fichier, qui sert à désigner (dans le programme) le fichier sur lequel on travaille, et d'accéder au contenu du flux.

Ouverture simple:

Python

```
f = open('monfichier.dat','r')
```

Le deuxième argument représente le mode d'ouverture, ici 'r' représente une ouverture en mode lecture.

Java

```
FileReader d = new FileReader ("monfichier.dat");
BufferedReader f = new BufferedReader(d);
```

Nous utilisons ici comme descripteur un objet de type BufferedReader signifiant explicitement que les données sont chargées en mémoire tampon (le "buffer"). Les objets FileReader et BufferedReader sont définis dans la librairie `java.io` :

```
import java.io.*;
```

Ouverture avec test :

Il est important de vérifier que cette opération d'ouverture s'effectue correctement avant de poursuivre le programme (nombreuses possibilités d'erreur : fichier effacé, erreur de nom, pas de droits de lecture,...). On utilise une instruction de test spécifique pour vérifier que l'ouverture du fichier s'est correctement effectuée, de type `try...catch...` (essaye ... sinon ...) permettant de prévoir une action de secours lorsqu'une une opération "risquée" échoue.

Python

```
try :
  f = open('monfichier.dat','r')
except IOError:
  print "Erreur d'ouverture!"
```

Java

```
try{
  FileReader d = new FileReader ("monfichier.dat");
  BufferedReader f = new BufferedReader(d);
}
catch (Exception e){
  System.out.println("Problème d'ouverture!");
}
```

Lorsque l'opération d'ouverture est réalisée avec succès, le flux de données devient accessible en lecture (les premières lignes du fichier sont chargées en mémoire et une tête de lecture se positionne sur le premier caractère de la première ligne). Il ne reste plus qu'à lire les données.

La consultation des données s'effectue séquentiellement à l'aide de l'opérateur de lecture `readline`. Chaque appel à cet opérateur charge les nouvelles données et déplace la tête de lecture vers les données suivantes. Cette opération peut être effectuée plusieurs fois jusqu'à atteindre la fin de fichier.

Si on suppose que les données sont rangées sous la forme d'une série de tuples, chaque opération de lecture consiste à consulter un tuple, et à positionner la tête de lecture sur le tuple suivant.

Exemples : lecture de données texte : chaque opération de lecture lit tous les caractères jusqu'au caractère "fin de ligne".

1. Lecture d'une ligne unique :

Python

```
s = f.readline()
```

Java

```
s = f.readLine()
```

2. Lecture de toutes les lignes (la lecture s'effectue dans une boucle) + affichage de la ligne:

Python

```
for s in f :
  print s
```

Java

```
String s;
while ((s = f.readLine()) != null){
    System.out.println(s);
```

```
    }
```

2.5.2 Enregistrement des données : sauvegarde dans un fichier (Write)

L'opération de sauvegarde des données est l'opération complémentaire de la lecture. De nouvelles données doivent être enregistrées sur le disque dur en vue d'une consultation future. Le format de sauvegarde peut être de type texte ou de type binaire. Nous présentons ici la sauvegarde des données dans des formats texte.

Comme dans le cas de l'opération de lecture, il faut au préalable définir dans le programme un descripteur de fichier servant de point d'entrée pour les opération d'écriture. On effectue ce qu'on appelle une ouverture en mode "écriture".

Python

```
try :
  f = open('monfichier.dat','w')
except IOError:
  print "Erreur d'ouverture!!"
```

Java

```
try{
  FileWriter d = new FileWriter ('monfichier.dat');
  BufferedWriter f = new BufferedWriter(d);
  }
catch (Exception e){
  System.out.println("Problème d'ouverture!");
  }
```

On notera qu'il existe en python (ainsi qu'en C, C++, ...) plusieurs modes d'ouverture exprimés par le deuxième argument de la fonction open. On retiendra le mode 'w' (création d'un nouveau fichier vide) et le mode 'a' (ajout de nouvelles données à la suite d'un fichier déjà existant).

La sauvegarde dans le fichier s'effectue à l'aide d'un opérateur d'écriture. Dans le cas des chaînes de caractères, l'opérateur d'écriture sauvegarde ces données à la suite des données déjà écrites.

Python

```
f.write("Bonjour!\n")
```

Java

```
f.write("Bonjour!\n");
```

La sauvegarde de données nécessite d'effectuer un choix sur le mode d'encodage, obéissant en général à une norme bien précise (csv, json, xml, etc...). Voir section 1.3.

Une fois les opérations de lecture ou d'écriture terminées, il est nécessaire de fermer le fichier. L'opération de fermeture assure que les données sont effectivement enregistrées sur le disque (et non simplement stockées dans la mémoire tampon - voir section XXX).

Algorithmes de bas niveau (niveau système d'exploitation)

Lors de la création (et mise à jour) d'un fichier, on lui alloue un espace sur le volume. Plusieurs régions peuvent être allouées pour un seul fichier) La taille minimale d'une région d'allocation étant une page, la région allouée à un fichier est composée d'un nombre entier de pages consécutives.

Remarque : le volume possède une table des pages libres (table de bits : chaque bit correspond à une page. Pour allouer, on fait passer le bit de 0 à 1, et on ajoute l'adresse de la page au descripteur de fichier). Lorsque tout ou partie d'un fichier est effacé, certaines pages sont libérées et introduites dans la liste des pages libres.

Stratégies d'allocation

On souhaite :

- minimiser le nombre de régions allouées à un fichier
- minimiser la distance entre deux régions successives

Stratégie par page :

- premier trouvé (la première page libre dans la table)
- meilleur choix (le plus proche de la dernière page allouée)

Stratégie par bloc: on alloue des blocs composés de plusieurs pages (S'il existe des blocs libres consécutifs, ils sont fusionnés en un seul)

- Plus proche choix : la liste des blocs libres est parcourue jusqu'à trouver un bloc de la taille demandée (ou sinon, le premier bloc de taille supérieure, qui est alors découpé en deux blocs).
  - first fit : le premier bloc suffisamment grand pour les besoins
  - best fit : le plus petit bloc qui ait une taille au moins égale à la taille demandée
  - worst fit : le plus grand bloc disponible (qui est donc découpé)

On alloue des blocs de 1,2,4,8,...2K pages. Pour une taille donnée $2i-1 < n < 2i$, on commence par chercher les blocs de taille 2i, puis 2i+1, ... jusqu'à 2K , en divisant ces blocs le cas échéant.

Problème des stratégies par bloc: s'il y a trop de fichiers, on obtient des blocs de taille 1 –> nécessité de réorganiser (défragmenter)

Lecture/Ecriture

Les opérateurs lire_ligne (readline) et écrire_ligne (write) ne travaillent pas directement sur les données du fichier. Les données du fichier sont chargées en mémoire centrale dans une mémoire "tampon". L'objet f servant à décrire le fichier a comme attributs :

- t : table des pages,
- i : numero de la page courante,
- p : tableau d'octets de la page courante (mémoire tampon),
- j : position dans la page courante (tête de lecture).

Lors des opération de lecture, la mémoire tampon est mise à jour au fur et à mesure qu'on avance dans la lecture du fichier par des opérations de lecture sur le disque. En général, plusieurs pages sont chargées en avance. Lors d'une opération d'écriture, la mémoire tampon reçoit les nouvelles données à écrire. Ces données sont effectivement écrites sur le disque lorsque le tampon est suffisamment rempli ou lors de l'opération de fermeture. Au moment de l'écriture effective, le système d'exploitation fait appel à un opérateur d'allocation pour choisir le "meilleur" bloc où stocker les données.

Voir aussi :

- Gestion des fichiers sous Unix